Andrea Turrini

Hierarchical and Compositional Verification of Cryptographic Protocols

Ph.D. Thesis

May 15, 2009

Università degli Studi di Verona
Dipartimento di Informatica

Advisor:
prof. Roberto Segala

Series N°: **TD-06-09**

Università di Verona
Dipartimento di Informatica
Strada le Grazie 15, 37134 Verona
Italy

Summary

Two important approaches to the verification of security protocols are known under the general names of *symbolic* and *computational*, respectively. In the symbolic approach messages are terms of an algebra and the cryptographic primitives are ideally secure; in the computational approach messages are bitstrings and the cryptographic primitives are secure with overwhelming probability. This means, for example, that in the symbolic approach only who knows the decryption key can decrypt a ciphertext, while in the computational approach the probability to decrypt a ciphertext without knowing the decryption key is negligible.

Usually, the cryptographic protocols are the outcome of the interaction of several components: some of them are based on cryptographic primitives, other components on other principles. In general, the result is a complex system that we would like to analyse in a modular way instead of studying it as a single system.

A similar situation can be found in the context of distributed systems, where there are several probabilistic components that interact with each other implementing a distributed algorithm. In this context, the analysis of the correctness of a complex system is very rigorous and it is based on tools from information theory such as the *simulation method* that allows us to decompose large problems into smaller problems and to verify systems hierarchically and compositionally. The simulation method consists of establishing relations between the states of two automata, called *simulation relations*, and to verify that such relations satisfy appropriate *step conditions*: each transition of the simulated system can be matched by the simulating system up to the given relation. Using a compositional approach we can study the properties of each small problem independently from the

each other, deriving the properties of the overall system. Furthermore, the hierarchical verification allows us to build several intermediate refinements between specification and implementation. Often hierarchical and compositional verification is simpler and cleaner than direct one-step verification, since each refinement may focus on specific homogeneous aspects of the implementation.

In this thesis we introduce a new simulation relation, that we call *polynomially accurate simulation*, or *approximated simulation*, that is compositional and that allows us to adopt the hierarchical approach in our analyses. The polynomially accurate simulations extend the simulation relations of the distributed systems context in both strong and weak cases taking into account the lengths of the computations and of the computational properties of the cryptographic primitives.

Besides the polynomially accurate simulations, we provide other tools that can simplify the analysis of cryptographic protocols: the first one is the concept of *conditional automaton*, that permits to safely remove events that occur with negligible probability. Starting from a machine that is attackable with negligible probability, if we build an automaton that is conditional to the absence of these attacks, then there exists a simulation. And this allows us to work with the simulation relations all the time and in particular we can also prove in a compositional way that the elimination of negligible events from an automaton is safe. This property is justified by the *conditional automaton theorem* that states that events are negligible if and only if the identity relation is an approximated simulation from the automaton and its conditional counterpart. Another tool is the *execution correspondence theorem*, that extends the one of the distributed systems context, that allows us to use the hierarchical approach. In fact, the theorem states that if we have several automata and a chain of simulations between them, then with overwhelming probability each execution of the first automaton is related to an execution of the last automaton. In other words, we have that the probability that the last automaton is not able to simulate an execution of the first one is negligible.

Finally, we use the polynomially accurate simulation framework to provide families of automata that implement commonly used cryptographic primitives and to prove that the symbolic approach is sound with respect to the computational approach.

Acknowledgements

The first person that I would like to thank is my advisor Roberto Segala for his precious guide and encouragement over these years.

A great thanks goes to Catuscia Palamidessi for her very kind hospitality and constant support while I was visiting the LIX at École Polytechnique, Paris, and all members of the Cométe and Parsifal teams.

I would also like to thank my PhD thesis referees Riccardo Focardi and Bogdan Warinschi, as well as Isabella Mastroeni and Massimo Merro for their hints and comments on my studies.

Finally, I would like to thank my family for its support over these years.

Contents

1

Introduction

1.1 Cryptography and Security

Cryptography and Security are two closely related fields of Computer Science that have a big impact on the real world. Security usually is based on cryptography, because secrecy, authenticity, anonymity and other similar properties are insured by cryptographic protocols, if they respect some well-defined requirements. For example, we can say that a protocol ensures anonymity of messages if nobody, except who has generated it, is able to say who is the sender of such message. Theoretic cryptographic protocols are the base of real protocols like AES, SSL, WEP, etc. that are used for passwords storage, secure web communications, secure wireless connections and other cases where it is required some kind of secrecy or authenticity. If a theoretical protocol contains errors that make it breakable, then all real implementations can be broken exploiting such errors.

To reduce the possibility of an attack, for example that an intruder can obtain secret informations, we must be sure that at least the theoretical protocol is correct. If we are sure that this happens, then we can concentrate all our attention on protocol implementation.

We can say that a cryptographical protocol is safe only when we are able to check that any adversary is not able to attack it successfully. This requirement is really strong, since it imposes that no adversary can attack the protocol, and not only adversaries we can conceive during the analysis of the protocol.

We can classify adversaries into two big classes, based on their capabilities: the first one is the class of *passive* adversaries that can only eavesdrop messages that are transmitted between participants of the protocol. The main aim of these adversaries is to worm out some information that can be

used for other purposes. For example, in [48] Fluhrer, Mantin and Shamir analyze the RC4 algorithm and as a side effect they show that in a wireless network protected by the WEP protocol an eavesdropper who can sniff a sufficient number of packets can then derive the WEP key. Once the adversary knows the key, it can collect sensible data as passwords or credit card numbers. The eavesdropper needs from one to four millions packets to obtain a 40-bit key while for a 128-bit key it spends some hours to break the WEP protocol (time depends on computational power and network load; see [108] for details).

The second class is composed by *active* adversaries that can interact with the protocol participants. This implies that an active attacker can intercept a message, modify it and then deliver the alterated message. The *man in the middle attack* is based on this capability, and protocols that are prone to this problem are, for example, SSL (during the exchange of public keys) and the Needham-Schroeder public key protocol [70, 71, 87].

When we start to analyze the security of a cryptographic protocol, the first thing we must consider is which kind of adversary can try to attack the protocol. We may consider only passive adversaries, for example; in this case the analysis is simpler but the guarantees are weaker. On the contrary, if we consider all adversaries, passive and active ones, then the analysis is more difficult but the guarantees are stronger. Since adversaries in the real life are usually active, then it is reasonable to prove the correctness of a protocol considering active adversaries.

The second thing we must consider when we design a new protocol is if we want to use randomization. In fact, we can define protocols that are deterministic or that involve random choices. If we decide to design our protocol using only deterministic primitives, then the proposed protocol could be prone to the *replay attack*, where a legal message m from the participant A can be sent to B more than once and each time B receives m, B supposes that m comes from A even when A is offline. In addition, an eavesdropper is able to understand when a message is repeated, even when it is sent encrypted. In fact, if we do not use randomization, then each time a message is encrypted, the resulting ciphertext is the same. Another problem of non-randomized protocols is the following: consider a public-key encryption scheme [49] and suppose that the set of plaintexts is prefixed and small. An adversary A can attack the protocol in the following way: it encrypts all plaintexts using the public key of a participant P producing a table with the associations cyphertext - plaintext. When another partici-

pant sends a message m encrypted with the public key of P, A obtains m looking for c in the table, where c is the ciphertexts corresponding to m.

To avoid such problems, one usually uses probabilistic elements like nonces (values that are used at most once) or randomized encryptions. Probabilistic elements enforce the quality of protocols but they do not ensures that resulting protocols are actually secure. In fact, we can define protocols that are malleable [50]: given one or more valid messages, the adversary can generate a new message that is valid and that it is not one of the messages produced by the participants. RSA [100], for example, is malleable.

As we have seen, we can use random values and randomized primitives to avoid some attacks. But this implies that when analyze the correctness of a protocol we must consider also all probabilistic aspects that occur in the protocol. If we omit some subtlety about probability, then later we can discover that the protocol is flawed. Consider, for example, the problem of dining cryptographers [39]. The situation is the following: there are three cryptographers that work for NSA and there is the bill to pay. NSA can decide to pay the bill or can tell to one of cryptographers to pay. In the former, everyone knows that the bill is payed by the NSA but in the latter no one (except the cryprographer that settles the bill) knows who is the payer. To achieve this result cryptographers perform this protocol: they are around the table and each one shares a coin with each neighbor. They flip coins, so each cryptographer can see two coins but not the value of the third one and he says agree or disagree if coins he see show the same side or not. The payer lies, hence he says disagree if coins are same and agree if they are different. If the number of disagrees is odd, then a cryptographer pays the dinner, else the bill is settled by NSA. If we analyze the protocol without probability (or under the hypothesis that coins are fair), then we can affirm that it is correct because each adversary can not guess the payer with a probability that is bigger that one third. If coins are not fair, then the adversary can iterate protocol a sufficient number of times with the same payer and if it knows the probability of head of each coin (but without knowing the actual values of coin tosses), then the adversary is able to identify with high probability the cryptographer that has settled the bill.

The dining cryptographers protocol is very easy to analyze manually, without using a formal model. But there exist other protocols that are more complex and a correct manual verification is infeasible, since there are too many details to consider. This situation can lead to proofs that are not very

rigorous and that contain informal reasoning that can be correct but that may hide problematic cases. So, if we want to be sure about cryptographic protocols correctness, it is better if we perform our analysis using a formal, rigorous model that ensures the correctness of our reasoning.

1.2 The Main Approaches of Protocol Verification

Several authors have studied the problem of cryptographic protocol verification, producing a large literature. We can identify basically two approaches: the first one is based on a *symbolic model* [3, 28, 45–47, 57, 64, 65, 70, 71, 77, 83, 85, 90, 91, 109] where messages are symbols and all underlying cryptographic components satisfy the requirements, axiomatically. Furthermore, the adversary can try to attack the protocol using a restricted set of actions that usually does not include the possibility to guess secrets (such as private keys). The second approach is based on a *computational model* [20–22, 26, 49–55, 112] where the exchanged messages are bitstrings and the adversary is limited only to work with bounded resources. This means that it can do what it wants to messages: it can try to guess secrets and to generate new messages, as well as modify or drop received ones.

1.2.1 The Symbolic Model Approach

The start point of symbolic model approach is the paper of Dolev and Yao [46]. In this model, cryptographic primitives used by protocol are modeled as symbolic operations or ideal boxes, which represent the security properties of the primitives in an idealized way. This means that if we have a protocol that is based on symmetric encryption, no one is able to obtain m from the encrypted message $\texttt{Enc}(m, k)$ without the knowledge of the key k. Basically, in the Dolev-Yao model, the cryptographic operations are replaced by term algebras with cancellation rules. For example, the encryption operation $\texttt{Enc}(m, k)$ becomes the term $E_k(m)$ and a typical cancellation rule is $D_k(E_k(\,\cdot\,)) = id\ (\,\cdot\,)$ where D_k is the decryption term associated with the key k and id is the identity function on messages.

Dolev-Yao model was extended in many papers, e.g. [43,45,47,79], and it is used in the automatic formal protocol verification, using tools like model checkers, theorem proovers, etc. [23, 27, 56, 64, 78, 82, 83, 85, 90]. Moreover, other formalisms are developed on Dolev-Yao model, like logics [29, 57,

80, 81], process algebras [68], π-calculus extensions [1–3], term rewriting systems [76], strand spaces [110, 111] and others [65, 77].

When we prove that there exists an attack against a protocol in the Dolev-Yao model, the same attack can be used also against all real implementations of the protocol, implementations based on real cryptographic primitives. This is possible since all attacks on this model use only polynomial time operations that can be replayed by a feasible real attacker. Using a Dolev-Yao representation of the public key authentication protocol of Needham and Schroeder [87], Lowe [70] has discovered that the protocol is prone to the man in the middle attack and later he proposed a repaired version [71] that is secure in the Dolev-Yao model. However, a correctness proof in the symbolic model is not enough to be sure that the protocol is correct when it is implemented using actual cryptographic primitives. In fact, consider an actual encryption algorithm: it is not ideal when implemented, since the adversary might guess the plaintext, so it can "decrypt" a ciphertext without knowing the corresponding decryption key, and this event is not captured by the symbolic model approach. This means that an analysis in a pure symbolic model is not enough to capture all possible attacks and thus we should consider also actual cryptographic aspects.

1.2.2 The Computational Model Approach

In the computational model, also known as the *cryptographic model*, a protocol is analyzed as it is. The correctness proofs for protocols are developed mainly following one of these two directions: in the first, it is proved that the probability of an attack to the protocol is negligible (e.g., exponentially low with respect to a security parameter); in the second, it is proved that if the attacker can break the protocol with non-negligible probability, then such attack can be used against an underlying cryptographic primitive that is supposed to satisfy required properties (for example, indistinguishability of encryptions and unforgeability of signatures).

This approach is widely used in cryptographic literature [20–22, 26, 49–55, 112] but we must be careful during the security analysis of a protocol. In fact, we must consider all possible scenarios and all adversaries to be sure that the correctness proof is complete. Moreover, we must consider all probabilistic details in the analysis, because if we omit some of them, then it is possible that we do not discover an attack when the adversary uses exactly such details to exploit an attack. The main problem in the

correctness proofs of computational model is that this kind of proofs is hand-made, it is long and very tedious since the prover must consider a lot of different cases and particulars. So he tends to take shortcuts counting on their validity but these can lead to incorrect proofs as showed, for example, in [44, 93].

Another problem is that it is quite difficult to reuse the correctness proof of one protocol inside the verification of another one. Hence, when a new protocol is developed, we must provide a complete verification, starting again from the beginning. Moreover, generally it is not possible to split proof into several steps using a hierarchical approach to analyze and verify a single cryptographic aspect at each level of the hierarchy.

1.3 A Similar Situation: The Context of Distributed System

As we have seen previously, the correctness proofs in the computational model are quite difficult and error prone. This is due to several causes, like the lack of rigour that comes from the need to take shortcuts and the impossibility of being thorough. Another cause is that it is not so easy to take into account the interaction of several entities that are based on (probabilistic) cryptographic primitives or on other principles. In fact, probability is involved into the generation of nonces and cryptographic keys, the encryption algorithm, and possibly the signing algorithm. Other sources of probability are, for example, the adversary and the network but in this case is not so easy to know which are the actual measures that describe such sources of probability.

The analysis of systems that can be modelled as several (probabilistic) machine that interact with each other is a problem already studied in the context of randomized distributed algorithms [72, 98, 99]. In particular, there are several common aspect between cryptography and distributed algorithms; for example, in both cases we have to consider probabilistic and nondeterministic behaviors and the analysis is usually performed comparing the executions of two systems that behave in a similar way or transforming the execution of one system to the execution of another one that represents an attacker. One example of the last case is the analysis of indistinguishability property: given an attacker, we build another machine whose behavior is very close to the one of the original attacker and that it is able to break the indistinguishability property. One formal model that is used into the concurrency theory to model and to study randomized distributed algorithms

is the Probabilistic Automata model [73,102–106]. This model provides the mathematical rigor that is necessary to study rigorously randomized systems and also provides the tools to relate executions of different systems: simulations and bisimulations allow us to compare the computations of two systems and to say if they behave in the same way or if their behaviors are not similar.

Moreover, concurrency theory allows us to prove properties of randomized distributed algorithm in an hierarchical and compositional way: the possibility to work hierarchically permits to model the problem at several level of abstraction, and each level represents the algorithm in a more or less detailed way. This means, for example, that we can define an abstract level where for example almost all probabilistic aspects are missing and where we can focus our attention on the specification of the problem, studying its properties to see if the specification satisfies our requirements. If it is the case, then we can define other levels where we detail the single components that form the overall algorithm. The compositionality allows us to study each single component independently from the others and to extends its properties to the overall system. This means that if we prove that a component at level i is an implementation of the same component in the level $i+1$, then we can say that the overall system at level i is an implementation of the system at level $i+1$. So this implies that whenever we generate a chain of implementations from the fully detailed system to the abstract system, by transitivity we can conclude that the fully detailed system is an implementation of the abstract system and thus it satisfies the same properties of the specification.

Compositionality is useful since it permits to consider small independent components instead of a big system with several interacting modules. Moreover we can reuse already known results in several proofs. So, if we already know that a functionality is implemented by a specific component, then we simply replace the functionality with it and by compositionality we derive that the new system is an implementation of the old one without proving it a second time.

In the context of distributed systems, the implementation is typically a form of behavioral inclusion: it can be based on traces, computations or traces that keep other informations like failures or tests. Failures usually are traces that are followed by actions that the system refuses to perform like the generation of forged signatures; tests are traces where a specific event occurs in an appropriate context like the generation of a repeated

nonce. It is proved that the implementation relation has nice properties: it is transitive and compositional and moreover it has logical characterization and this permits to know which kind of properties the implementation relation preserves.

One problem of the implementation relation is that it is difficult to check. In fact, consider the case of trace inclusion: a trace of the real system can be a huge object, possibly infinite, and it could be very hard to verify if it is also a trace of the abstract system; for example, language inclusion is a PSPACE-complete problem. Fortunately, the simulation and bisimulation relations allow us to derive global properties of traces checking the properties preserved by each computational step of the system independently from other steps like in the Hoare-style logical reasoning [60]. In fact, usually we reason about the properties that are satisfied in a state and the ones that are satisfied after performing an execution step. Then, just composing all reasoning we derive the properties that are valid in the global trace.

1.4 Our Proposal

As we have seen, concurrency theory and cryptography study systems that involve probabilistic behaviors and interaction between several entities and that present similar problems. Since probabilistic automata are an useful framework that help the analysis of randomized distributed system, we want to check if the probabilistic automata can help the verification of cryptographic protocols. To do this, we propose to use the probabilistic automata directly in the computational model and to utilize the mathematical rigor and the simulation relations of the framework to study the correctness of cryptographic protocols. We decide to model the protocol in a Dolev-Yao style [46]: participants (or agents) of the protocol are fair and they communicate using an adversarially controlled network. The adversary tries to break the protocol generating, intercepting or modifying the messages that are sent from a participant to another one.

To verify security properties of a protocol, we would like to work hierarchically, defining several adversaries that model different levels of security abstraction and then establishing some relation between adversaries. In particular, we would like to define an adversary that can be seen as an attacker inside the symbolic model and another adversary that stands for an attacker inside the cryptographic model. So, for example, we can have the adversary that ensures the correctness by construction (in particular,

it represents a network that simply forwards messages between agents or it never sends messages that can break the protocol) and the adversary that represents a possible real attacker that generates messages using some (probabilistic) function on previous messages.

In the computational model, a real adversary can always generate a message that breaks the protocol: it simply chooses a random sequence of bits of the right length and if it is lucky, then such bitstring is a message that violates the protocol. Security requirements usually impose that a message generated by the adversary breaks the protocol with a negligible probability with respect to some security parameter. In the symbolic model the adversary can not generate a message randomly, but it can only derive new messages from old ones, so we can not model the generation of a new random message in the symbolic model. This implies that standard simulation relations of probabilistic automata model can not be used to relate real and ideal adversaries, because a real attacker can generate messages and hence it can perform actions that an ideal adversary can not simulate. This means that we need to define a new extension of simulation relations, extension that permits to match the step condition up to some error. Moreover, we must take into account the fact that in the computational model both the security of the protocol and the computational power of the adversary depend on a security parameter: given a cryptographic protocol, usually its security is defined as "for each probabilistic polynomial time adversary, the probability of an attack is negligible" that means that for each constant c and polynomial p, there exists a security parameter $\bar{k}$ such that for each $k > \bar{k}$ if the adversary A runs for at most $p(k)$ steps, then the probability that A performs a successful attack is less than k^{-c}. The security parameter usually is used to establish the length of nonces and of cryptographic keys and thus the protocol depends on such parameter. For example, consider the Needham-Schroeder-Lowe protocol [70, 71, 87]:

$$
\begin{aligned}
&A \rightarrow B : \{N_a.A\}_{E_b} \\
&B \rightarrow A : \{N_a.N_b.B\}_{E_a} \\
&A \rightarrow B : \{N_b\}_{E_b}
\end{aligned}
$$

Given the security parameter k, a concrete implementation of such protocol can require that nonces N_a, N_b belong to $\{0,1\}^k$ as well as the agent identities A and B and the encryption keys E_a and E_b.

As we have seen, in the computational model both agents and adversaries are parameterized by a security parameter. This means that for each

value of security parameter, we have different adversaries and agents that work always in the same way but using different values (for example, the length of nonces) and usually the security parameter is used to fix the computational power of adversaries. Hence we should extend the simulation relation to consider also families of automata (and not only single automata), families that are parameterized by the security parameter. To consider the computational power of adversary, we can extend ordinary simulations adding some information about how many steps we have spent to reach a particular state of the automaton. To do this, we can base our simulation on automaton executions instead of automaton states, because an execution describes the sequence of states and actions that has led to its final state. In this way we are able to provide an upperbound to the computational power of an adversary bounding the execution lengths, that can be related to the security parameter via a polynomial, for example.

As we said previously, a real adversary can generate messages that an ideal adversary can not simulate but such messages must have negligible probability (otherwise the protocol is not secure). To consider these messages, we can extend simulations permitting that the matching transition matches up to an error. If we relate such error to the security parameter, then we can provide an upperbound to the global error made by a real adversary with respect to an ideal adversary after a given number of steps. In this way, if we force the number of steps to be polynomial with respect to the security parameter and the step error to be negligible, then the global error is negligible and hence the protocol is secure with respect to the computational model meaning.

The main contribution of this thesis is the definition of the *polynomially accurate simulation* relation: an extension of the simulation relation of probabilistic automata that takes into account the computational aspects of cryptographic primitives, the length of executions, and the errors we accept. Main advantages of this simulation are that we can fix an upperbound to the adversary's computational power bounding the length of executions; we can decide if the probability of unmatched executions is negligible and hence to decide if there exist attacks such that their probabilities of success are not negligible. Moreover, the verification of the protocol correctness is local, step-based and not global. In this way we can focalize our attention to a restricted set of adversarial actions and this permits to simplify the verification. This holds because the check of the step condition reduces directly to the statement of correctness of the underlying cryptographic protocols

and if we have considered enough level of abstractions, for each simulation we can analyze a single cryptographic aspect: a simulation considers only that nonces are not repeated, another considers only signs, and so on. On the contrary, when the analysis is global, we must consider all computational aspects at the same time and this makes the correctness proof more difficult.

We also prove several properties of the polynomially accurate simulations: the first one is that it is compositional, that is given the automata $\mathcal{A}$ and $\mathcal{B}$, if $\mathcal{A}$ is polynomially accurate simulated by $\mathcal{B}$, then for each context $\mathcal{C}$ compatible with both $\mathcal{A}$ and $\mathcal{B}$, it holds that $\mathcal{A}$ composed $\mathcal{C}$ is polynomially accurate simulated by $\mathcal{B}$ composed $\mathcal{C}$. Since the polynomially accurate simulation is compositional, we can study complex cryptographic protocols as the composition of several automata that model the cryptographic primitives, the agents, the adversary, and so on. Moreover, the compositionality allows us to model real and ideal cryptographic primitives using automata relating them using our new simulation, defining a library of basic results that can be used in all other security analysis of cryptographic protocols without having to prove again the existence of the simulation.

Another result that is related to the hierarchical verification is the *execution correspondence theorem*, that extends the one of the distributed systems context. In fact, the theorem states that if we have several automata and a chain of simulations between them, then with overwhelming probability each execution of the first automaton is related to an execution of the last automaton. In other words, we have that the probability that the last automaton is not able to simulate an execution of the first one is negligible. The execution correspondence theorem provides a formal justification of the use of the hierarchical verification of the security of cryptographic protocols: given the model of the concrete protocol, using compositionality we replace a different primitive at each level of refinement and we are sure that with overwhelming probability each execution of the concrete protocol is related to an execution of the idealized protocol.

The second main contribution of this thesis is the concept of *conditional automaton* that permits to safely remove events that occur with negligible probability. Starting from a machine that is attackable with negligible probability, if we build an automaton that is conditional to the absence of these attacks, then there exists a simulation. And this allows us to work with the simulation relations all the time and in particular we can also prove in a compositional way that the elimination of negligible events from an

automaton is safe. This property is justified by the *conditional automaton theorem* that states that events are negligible if and only if the identity relation is an approximated simulation from the automaton and its conditional counterpart.

1.5 Related Work

As we have seen, when we adopt the symbolic model, we have rigorous proofs but we do not take into account the computational aspects; on the other hand, the cryptographic model permits to consider the computational aspects of the protocols, but proofs are quite complex and they may be not so rigorous. Hence it seems to be interesting to have a way to use rigour and simplicity of symbolic model and obtain results in the cryptographic model.

In literature there are find several papers that are focused on the proof of the soundness of symbolic analysis. This correctness is achieved using different approaches: some author prefers to use a logic-like description of the protocol and the analysis is based on the adversary's ability to manipulate the terms that model the exchanged messages to attack the protocol and to induce the participants to complete the protocol with the adversary without realizing that they are interacting with it and not with another participant. Once the protocol is proven to be correct within the provided model, the soundness results permit to extend the correctness to the computational model, provided that cryptographic primitives respect the hypotheses of the soundness theorems.

Other groups of authors base their papers directly in the computational model and each group provides a framework that can be used to describe the protocol. In these frameworks, the protocol, the cryptographic components and the adversary are modelled using interactive Turing machines. To prove the correctness of the protocol, two or more systems are considered: one of these systems is the composition of the protocol, the adversary, and the actual cryptographic primitives; another of these systems is the composition of the protocol, another adversary that can be different from the first one, and the cryptographic primitives that ensure their correctness by construction. A protocol is said to be correct if it is not possible to distinguish between an execution of the first system and an execution of the second system except for a negligible set of cases. All these frameworks provide a compositional theorem that permits to compose several proto-

cols together or to split the analysis into several sub-problems in order to simplify the actual correctness proof reusing previous results or focusing on smaller components.

1.5.1 The Abadi and Rogaway Work

A first work on the correctness proofs is the one of Abadi and Rogaway [6,7] and subsequent works [4,62,80,81]. In these papers it is provided a formal model based on data expressions and an equivalence relation over them and the model is used to analyze the correctness of protocols that are based on symmetric encryption and the attacker is a passive adversary. Recall that a passive adversary can eavesdrop messages but it can not generate, modify or drop messages sent by participants of the protocol. Each expression represents a message that is exchanged between participants of the security protocol and such message is built up from bits and keys by pairing and encryption. The equivalence relation captures when two pieces of data "look the same" to an adversary that has no prior knowledge of the keys that are used to generate the data. The expressions are defined using a context free grammar and two expression are equivalent if their patterns are the same up to key renaming. A pattern of an expression is another expression that is the same of the original except for those encrypted components for which the key is unknown to the adversary. An example is the following: denote by K_1, K_2 two different keys, by $(\,\cdot\,,\,\cdot\,)$ the pairing and by $\{v\}_k$ the encryption of the value v under the key k. An adversary can recover the key K_1 from the expression $M = (\{\{(0,1)\}_{K_2}\}_{K_1}, K_1)$ and then the value $\{(0,1)\}_{K_2}$ but it can not obtain K_2 nor $(0,1)$. The pattern of M is the expression $(\{\Box\}_{K_1}, K_1)$ where $\Box$ denotes an arbitrary undecryptable value.

The computational model they provide is based on *indistinguishability* [53,112] over sets of strings. In particular, two distributions S_1 and S_2 are (computationally) indistinguishable if for each probabilistic polynomial time adversary A, $|p_1 - p_2|$ is negligible, where for $i = 1, 2$, p_i is the probability that A returns 1 when it receives strings chosen accordingly to S_i. In other words, the probability that A declares that the strings are chosen accordingly to S_1 when they are actually chosen from the support of S_1 is essentially the same of the probability that A declares that strings are chosen accordingly to S_1 instead of S_2.

Given an encryption scheme and a security parameter η, Abadi and Rogaway show how to map an expression to a string (the map is probabilistic,

because the used encryption scheme is probabilistic) and then they prove a soundness theorem for their formal model. Soundness theorem states that, given two acyclic expressions and a *type-0* secure encryption scheme, if expressions are equivalent, then their maps into computational model are indistinguishable. An encryption scheme is *type-0* secure if it does not reveal message repetitions, which key is used in encryption and the message length; an expression E is acyclic if it is not cyclic: an expression E is cyclic if there exists a pair of keys K_1, K_2 that occur in E, such that there exist two sub-expression E_1, E_2 of E such that E_1 is encrypted with K_1 and it contains K_2 as plain text and, similarly, E_2 is encrypted with K_2 and it contains K_1 as plain text. Cycles, such as encrypting a key under itself, are a source of errors in practice; they also lead to weakness in common computational models.

In [6, 7], authors provide a counter example to show that formal model is not complete. Such example is based on the fact that it is possible to generate two different messages that are mapped outside the set of input messages of encryption scheme and hence they are both consider as invalid messages and they are associated to the same string set.

Other papers [59, 62, 80, 81] extend the result of Abadi and Rogaway in several ways: they consider active adversaries instead of passive ones; they provide a more meaningful counter example and a limited completeness theorem for the logic of Abadi and Rogaway. [80] presents and proves the completeness theorem under the hypothesis of acyclic expressions and type-0 *confusion free* encryption scheme: an encryption scheme is confusion free if for each pair of keys k_1, k_2, if c is the ciphertext obtained encrypting the message m under k_1 and m' is the result of decrypting c using k_2, then the probability that m' is not a failure message is negligible. Other papers [62, 81] consider active adversaries and prove that the symbolic model is a safe abstraction of the computational one using hypothesis on the cryptographic primitives that are less restrictive than in [80]. [4] extends [6] defining a more complex language that considers a system of programs that communicate synchronously and hence proving a similar soundness theorem for the new language.

1.5.2 The Backes and Pfitzmann Work

A second work that considers the security analysis of cryptographic protocols and the soundness of the symbolic approach with respect to the

computational approach is the one of Backes and Pfitzmann. The authors have defined a general system model for cryptographic protocols, including a model of what adversaries and honest users can do [8, 9, 91, 92]. The model (founded on [94]) is based on *reactive systems*. A reactive system is a probabilistic extended finite machine which input can change during execution. Such machines are similar to the probabilistic I/O automata of Segala [102] but input actions can be ignored. To consider computational aspects, reactive systems are implemented with interactive probabilistic Turing Machines [49].

The standard way to prove the correctness of a protocol is the following: consider an adversary, a set of honest users and at least two reactive systems. The adversary and the honest users can interact arbitrarily and both communicate with one of the reactive systems. In particular, one of the reactive systems is a real system that uses actual cryptographic primitives and another is an ideal system that ensures the correctness of cryptographic primitives by definition. The verification is performed checking if the composition of adversary, honest users and the real system is as secure as the composition of another adversary, honest users and the ideal system.

The "as secure as" relation is also said *reactive simulatability* [95, 97] and it is defined as follows: take two systems S_1, S_2 with the same set H of honest users, an adversary A_1 for S_1 and consider which attacks A_1 can perform against H in S_1. S_1 is as secure as S_2 if there exists A_2 for S_2 such that A_2 can perform the same attacks of A_1 against H essentially with the same probability. This means that honest users are not able to distinguish when they are interacting with the first system or with the second system, except for a negligible set of cases.

The model is defined for both synchronous [95] and asynchronous [18, 96, 97] cases and main results are the reactive simulatability and a composition theorem [18], that can be used to replace a sub-system A with another system B and, if A is as secure as B, the overall system does not modify its security properties.

Authors have used the model to verify security properties of protocols like integrity [11, 95], liveness [15, 16] and secrecy [13, 14] and they have also developed a cryptographic library [10, 12, 19] in a Dolev-Yao style that can be used to prove the correctness of protocols like the one of Needham, Schroeder and Lowe [71] and the Otway-Rees protocol [89] for efficient mutual authentication (via a mutually trusted third party).

1.5.3 The Canetti Work

A third work that provides a tool to verify a cryptographic protocol is the one of Canetti. The main paper is [31] (together with the preliminary technical report [30]) that is extended by [36–38, 61, 88]. The *Universally Composable* (UC) security framework of Canetti is based on the same idea of the model of Backes and Pfitzmann: define two systems, one ideal and one real, and show that they are not distinguishable.

In particular, first it is formulated a model representing the process of protocol execution in the real-life. This is called the real-life model. Next, in order to capture the security requirements of a given task, it is formulated an ideal process for carrying out the task. It is then possible to say that a protocol securely realizes the task at hand if running the protocol in the real-life model amounts to "emulating" the ideal process for that task.

To capture the computational aspects of a protocol, the real-life model consists of a set of Interacting Turing Machines (ITMs), each one representing a party involved in the protocol, plus an ITM that plays the role of the adversary. These machines are composed with another ITM that represents the environment that is whatever is external to the current protocol execution, like other protocols runs, other adversaries, etc. Security is proved by showing that the environment is not able not distinguish when it is composed with the adversary and the real system or when it is composed with another adversary and the ideal system. Multiple instances of the modelled protocol can run in parallel and the environment can interact with other parties not only at the beginning of the execution, but also during the protocol run.

The main result of this framework, that justifies the Universally Composable name, is the universal composition theorem. Standard composition theorems permit to decompose a complex task into two or more sub-tasks that are usually simpler to design and analyze. This means that once it is designed the protocols that securely realizes the sub-tasks, it is possible to provide a protocol that implements the given task assuming that the evaluation of the sub-task is possible. Finally, the composition theorem permits to argue that the protocol composed by the already-designed sub-protocols securely realizes the given task. The composition theorem provided in the UC security framework is used as a tool for gaining confidence about the level of security of a protocol with respect to arbitrary environments. Indeed, protocols that satisfy a UC definition are ensured to maintain their

security within each protocol environment we can consider, known or unknown.

1.5.4 The Warinschi Work

Besides the previous works, in literature we find authors that provide soundness results that permit to relate the symbolic and the computational model. These soundness results are very useful in the analysis of the correctness of a protocol since they permit to study the protocol in the symbolic model and then they permit to extend the results to the computational case, provided that cryptographic primitives such as encryptions and signatures satisfy cryptographic assumptions like unforgeability of signatures or indistinguishability of encryption. We can find such results in the papers of Warinschi *et al.* [40–42]. In these works, authors define a language that is used to describe the protocol; then two models are provided: a formal execution one where messages are terms of an algebra, cryptographic primitives are correct by definition, and the attacker is a Dolev-Yao style adversary [46]; the second model is a concrete execution model where messages are bitstrings, cryptographic primitives satisfy standard requirements, and the attacker is probabilistic polynomial time adversary that can generate, intercept, drop and modify messages. To prove the soundness theorem, they consider an execution of the protocol in the concrete model and then they show that with overwhelming probability it is possible to map such execution into a symbolic one and that such symbolic execution is valid, that is, a symbolic adversary is able to generate it performing only allowed operations on the terms that represent the exchanged messages. The soundness results permit to relate several properties of the protocols, such as trace properties like entity and message authentication, and secrecy properties of nonces. Related work that obtain similar result are, for example, [6, 7, 17, 67, 81]. The main differences between these works are on the type of adversary (passive in [6, 7] and active in the other papers), the number of protocol session (priorly fixed in [67]), and the kind of cryptographic primitives (symmetric encryption [6, 7, 17, 67], public key encryption [40–42, 81], signatures [41, 42], and hash functions [40]).

1.5.5 Other Work

Other papers that provide tools to establish the correctness of cryptographic protocols are based on strand spaces, on a process calculus that is a variant of CCS or on probabilistic automata.

Strand spaces [58,110,111] are collections of strands, which are sequences of events that represent either an execution of a legitimate party of a protocol or a sequence of actions of the adversary. Strands are used to model messages that each participant expects or wants to send during a protocol run; an adversary performs a successful attack when it is able to generate a new, fresh message that leads a protocol participant to complete its strand. [58] presents a way to fix a security parameter to respect a given bound for the probability of successful attacks. Computational aspects are considered as they are, in an ordinary cryptographic point of view, not inside the strand spaces model.

In the process calculus defined by Mitchell *et al.* [68, 69, 75, 84, 86] they consider only expressions that are probabilistic polynomial time in the sense that evaluation of each process expression halts in probabilistic polynomial time. They define a probabilistic bisimulation to relate expressions, bisimulation based on an observational equivalence that is a standard relation from programming language theory that involves quantifying over all possible environments that might interact with the protocol. The probabilistic bisimulation is used to establish a soundness result for an equational proof system based on such observational equivalences. Protocol computational aspects are captured directly in the definition of the calculus.

The last approach we consider is the one based on Probabilistic Automata. The main group that works with probabilistic automata is composed by Canetti *et al.* [32–35]. They model the protocol participants as probabilistic I/O automata [72,73,102,106] and the relations between them are obtained using *simulation* relations [74]. Within the framework they propose, they formalize the notion of "implementing a specification" along the lines of the notion of "realizing an ideal functionality" within the universally composable security framework of Canetti [30]. In particular, their idea is to assert the security of a protocol directly in a concrete model without abstracting the cryptographic primitives (unlike the Dolev-Yao based models). The security of a protocol usually holds only when adversaries are computationally bounded machines and only under computational assumptions (as happens in cryptographic models). Then, correctness of a

protocol is proved establishing a simulation between a real adversary and an automaton that ensures the ideal functionality by construction.

1.6 Overview of the Thesis

This thesis is structured as follows. In Chapter 2 we introduce some preliminary notions about mathematical background, probabilistic automata and cryptography that we will use throughout the thesis. In particular, we recall the concepts that are used in probability theory like measurable spaces and functions and probability measures; for probabilistic automata, we recall the definition of the automata, the parallel composition and the simulation relation, that uses the concept of lifting of relations to probability measure; finally, we recall some cryptographic primitives like oracle machines, nonces, (pseudo-)random functions, message authentication codes, public key encryption schemes and public key signature schemes.

In Chapter 3 we introduce a new simulation relation, the polynomially accurate simulation. We define such simulation starting from the ordinary simulation between probabilistic automata and modifying it to take into account the lengths of executions, the fact that in the computational model the adversary, the participants and the cryptographic primitives are parameterized on a security parameter, and the fact that it is possible to perform some successful attack but the probability of such attack should be negligible.

In Chapter 4 we consider the relation that exists between an automaton $\mathcal{A}$ and its variants. In particular, we focus our attention on the automata obtained from $\mathcal{A}$ adding some history variable and extending its set of external actions under the hypothesis that the effect of the new actions does not interfere with the original actions. Then we define the notion of G-conditional automaton $\mathcal{A}|G$ that is the automaton obtained from $\mathcal{A}$ in the following way: each transition of $\mathcal{A}|G$ is a transition of $\mathcal{A}$ which target measure is conditioned to reach states in G. This means that the probability to reach states of $\mathcal{A}|G$ that belong to $B = S\backslash G$ is zero. Finally, we prove our Conditional Automaton Theorem: $\mathcal{A}$ is polynomially accurate simulated by $\mathcal{A}|G$ if and only if the probability to reach states in B is negligible.

In Chapter 5 we analyze the polynomially accurate simulation pointing out some limitations it presents. The first limitation is that we are not sure that it is transitive: the naive way to define transitivity does not lead to a polynomially accurate simulation but anyway we are able to prove

the Execution Correspondence Theorem that allows us to relate executions of a family of automata to executions of another one whenever we are able to find a sequence of intermediate families of automata starting with the first family and ending with the last one that are related by a chain of simulations. The second limitation is that the polynomially accurate simulation is not compositional. We overcome this problem defining a new polynomially accurate simulation, based on states instead of on execution, that implies the approximated simulation defined on executions and that is compositional. We are able to prove the two main theorems of polynomially accurate simulation also for this new notion of simulation, so we can use the first definition or the second one according to the result we want to achieve. Finally, we extend the simulation based on states to the weak case, simply replacing the matching combined transition with a weak bounded combined transition.

In Chapter 6 we apply the two notions of polynomially accurate simulations to the analysis of the cryptographic primitives. In particular, we show how the real implementation of the primitive is polynomially accurate simulated by its ideal counterpart. We obtain such simulations starting from the real implementation, modifying it by adding history variables and actions and taking the G-conditional automaton. If we look at this chain of automata and simulations, then we can observe that the we use the argumentation about the properties of the cryptographic primitive such as negligibility of repeated nonces and unforgeability of signatures only when we consider the G-conditional automaton, where we define the set $B = S \backslash G$ as the set of states where an attack occur, that is, a nonce is repeated or a signature is forged.

In Chapter 7 we use the results on cryptographic primitives to study the security of the MAP1 protocol of Bellare and Rogaway [22]. In this case study, we provide two proofs that involve the polynomially accurate simulation: the first one allows us to replace the real nonce generator with its ideal counterpart and that we perform using previous results; the second one involves the message authentication codes and we directly apply the definition of polynomially accurate simulation to point out that the negation of the step condition (that is the non-existence of the simulation) leads to the negation of the security property of the cryptographic primitive.

In Chapter 8 we consider a more complex case study: we recast the soundness result of Cortier and Warinschi [41] and we show how polyno-

mially accurate simulations can be used as a sanity check tool for existing proofs.

Finally, in Chapter 9 we sum up the major contributions of the thesis and we briefly describe future works that we would like to explore.

2

Preliminaries

In this chapter we recall some basic concepts that are used inside this thesis. In particular, we recall concepts from Probability Theory, Probabilistic Automata and Cryptography.

2.1 Probability Theory

2.1.1 Measurable Spaces

Consider a set Ω. A *σ-field* on Ω is a set $\mathcal{F} \subseteq 2^{\Omega}$ that includes Ω and is closed under complement and countable union. A *measurable space* is a pair $(\Omega, \mathcal{F})$ where Ω is a set, also called *sample space*, and $\mathcal{F}$ is a σ-field over Ω. A measurable space $(\Omega, \mathcal{F})$ is called *discrete* if $\mathcal{F} = 2^{\Omega}$.

2.1.2 Probability Measures and Spaces

A *measure* over a measurable space $(\Omega, \mathcal{F})$ is a function $\rho \colon \mathcal{F} \to \mathbb{R}^{\geqslant 0}$ such that, for each countable collection $\{\Omega_i\}_{i \in I}$ of pairwise disjoint elements of $\mathcal{F}$, $\rho(\cup_I \Omega_i) = \sum_I \rho(\Omega_i)$. A *probability measure* over a measurable space $(\Omega, \mathcal{F})$ is a measure ρ over $(\Omega, \mathcal{F})$ such that $\rho(\Omega) = 1$. A *sub-probability measure* over $(\Omega, \mathcal{F})$ is a measure over $(\Omega, \mathcal{F})$ such that $\rho(\Omega) \leqslant 1$. A measure over a discrete measurable space $(\Omega, 2^{\Omega})$ is called a *discrete measure* over Ω. The *support* of a measure ρ over $(\Omega, \mathcal{F})$, denoted by $Supp(\rho)$, is the set $\{\omega \in \Omega \mid \rho(\omega) > 0\}$.

A *probability space* is a triple $(\Omega, \mathcal{F}, \rho)$, where $(\Omega, \mathcal{F})$ is a measurable space and ρ is a probability measure on $(\Omega, \mathcal{F})$.

Given a set X, denote by $Disc(X)$ the set of discrete probability measures over X, and by $SubDisc(X)$ the set of discrete sub-probability measures over X. We call a discrete probability measure a *Dirac* measure if it

assigns measure 1 to exactly one object x (denote this measure by δ_x). We also call Dirac a sub-probability measure that assigns measure 0 to all objects. In the sequel discrete sub-probability measures are used to describe progress. If the measure of a sample space is not 1, then it means that with some non-zero probability the system does not progress.

Given a set X, a set $G \subseteq X$, and a measure $\rho \in \mathit{Disc}(X)$ such that $\rho(G) > 0$, we call the measure $\rho' \in \mathit{Disc}(X)$ the *G-conditional measure of* ρ, denoted by $\rho|G$, if for each $x \in X$,

$$\rho'(x) = \begin{cases} \rho(x)/\rho(G) & \text{if } x \in \mathit{Supp}(\rho) \cap G \\ 0 & \text{otherwise} \end{cases}$$

2.1.3 Measurable Functions and Image Measures

Let $(\Omega_1, \mathcal{F}_1)$ and $(\Omega_2, \mathcal{F}_2)$ be two measurable spaces. A function $f\colon \Omega_1 \to \Omega_2$ is said to be a *measurable function* from $(\Omega_1, \mathcal{F}_1)$ to $(\Omega_2, \mathcal{F}_2)$ if the inverse image under f of any element of $\mathcal{F}_2$ is an element of $\mathcal{F}_1$. In this case, given a measure ρ on $(\Omega_1, \mathcal{F}_1)$ it is possible to define a measure on $(\Omega_2, \mathcal{F}_2)$ via f, called the *image measure* of ρ under f and denoted by $f(\rho)$, as follows: for each $X \in \mathcal{F}_2$, $f(\rho)(X) = \rho(f^{-1}(X))$. In other words, the measure of X in $\mathcal{F}_2$ is the measure in $\mathcal{F}_1$ of those elements whose f-image is in X. The measurability of f ensures that $f(\rho)$ is indeed a well defined measure.

2.2 Relations and Lifting of a Relation to Measures

Let $\mathcal{R}_1$ be a relation from a set X to a set Y and $\mathcal{R}_2$ be a relation from a set Y to a set Z. The *composition* of $\mathcal{R}_1$ and $\mathcal{R}_2$, denoted by $\mathcal{R}_1 \circ \mathcal{R}_2$, is a relation from X to Z defined as $\mathcal{R}_1 \circ \mathcal{R}_2 = \{(x, z) \mid \exists y \in Y.(x, y) \in \mathcal{R}_1 \wedge (y, z) \in \mathcal{R}_2\}$. The extension to n relation is straightforward: given n relations $\mathcal{R}_1, \ldots, \mathcal{R}_n$ such that for each $1 \leqslant i \leqslant n$, $\mathcal{R}_i$ is a relation from a set X_i to X_{i+1}, the relation $\mathcal{R}_1 \circ \ldots \circ \mathcal{R}_n$ is the relation from X_1 to X_{n+1} defined as $(\ldots((\mathcal{R}_1 \circ \mathcal{R}_2) \circ \mathcal{R}_3) \circ \ldots) \circ \mathcal{R}_n$.

Let $\mathcal{R}_1$ be a relation from W to X and $\mathcal{R}_2$ be a relation from Y to Z. The *cross-product* of $\mathcal{R}_1$ and $\mathcal{R}_2$, denoted by $\mathcal{R}_1 \times \mathcal{R}_2$, is the relation $\mathcal{R} \subseteq (W \times Y) \times (X \times Z)$ such that $(w, y)\ \mathcal{R}\ (x, z)$ if and only if $w\ \mathcal{R}_1\ x$ and $y\ \mathcal{R}_2\ z$.

Let $\mathcal{R}$ be a relation from a set X to a set Y. The *lifting* of $\mathcal{R}$, denoted by $\mathcal{L}(\mathcal{R})$ [105], is a relation from $Disc(X)$ to $Disc(Y)$ such that $\rho_x \; \mathcal{L}(\mathcal{R}) \; \rho_y$ if and only if there exists a *weighting function* $w\colon X \times Y \to [0,1]$ such that

1. $w(u,v) > 0$ implies $u \; \mathcal{R} \; v$,
2. $\sum_{u \in X} w(u,v) = \rho_y(v)$, and
3. $\sum_{v \in Y} w(u,v) = \rho_x(u)$.

An alternative definition of lifting given in a more probabilistic style states that $\rho_1 \; \mathcal{L}(\mathcal{R}) \; \rho_2$ if and only if there exists a joint measure w with marginal measures ρ_1 and ρ_2 such that the support of w is included in $\mathcal{R}$.

Note that if $\mathcal{R}$ is an equivalence relation, then $\rho_1 \; \mathcal{L}(\mathcal{R}) \; \rho_2$ if and only if, for each equivalence class C of $\mathcal{R}$, $\rho_1(C) = \rho_2(C)$.

The lifting of a relations has some interesting properties:

Property 2.1. *Let $\mathcal{R}$ and $\mathcal{S}$ be two relations.*

1. *$x \; \mathcal{R} \; y$ if and only if $\delta_x \; \mathcal{L}(\mathcal{R}) \; \delta_y$.*
2. *$\mathcal{R} = \emptyset$ if and only if $\mathcal{L}(\mathcal{R}) = \emptyset$.*
3. *If $\mathcal{R} \subseteq \mathcal{S}$, then $\mathcal{L}(\mathcal{R}) \subseteq \mathcal{L}(\mathcal{S})$.*
4. *If $\mathcal{R}$ is reflexive, then $\mathcal{L}(\mathcal{R})$ is reflexive.*
5. *If $\mathcal{R}$ is symmetric, then $\mathcal{L}(\mathcal{R})$ is symmetric.*
6. *Let ρ_1, ρ_2, ρ_3 be three probability measures. If $\rho_1 \; \mathcal{L}(\mathcal{R}) \; \rho_2$ and $\rho_2 \; \mathcal{L}(\mathcal{S}) \; \rho_3$, then $\rho_1 \; \mathcal{L}(\mathcal{R} \circ \mathcal{S}) \; \rho_3$.*
7. *If $\mathcal{R}$ is transitive, then $\mathcal{L}(\mathcal{R})$ is transitive.*
8. *Let ρ_x, ρ_y, ρ_z be three probability measures. If $\rho_x \; \mathcal{L}(\mathcal{R}) \; \rho_y$, then $\rho_x \times \rho_z \; \mathcal{L}(\mathcal{R} \times id) \; \rho_y \times \rho_z$.*

Proof. 1. Let $\mathcal{R}$ be a relation from X to Y.

($\Rightarrow$) Define $w\colon X \times Y \to [0,1]$ as: for each $u \in X$, $v \in Y$,

$$w(u,v) = \begin{cases} 1 & \text{if } u = x \text{ and } v = y, \\ 0 & \text{otherwise.} \end{cases}$$

w is a weighing function for δ_x and δ_y. In fact:

- $w(u,v) > 0$ implies that $u = x$ and $v = y$ and thus $(u,v) = (x,y) \in \mathcal{R}$;
- for each $u \in X$,

$$\sum_{v \in Y} w(u,v) = \begin{cases} 1 & \text{if } u = x \\ 0 & \text{otherwise} \end{cases}$$

By definition of Dirac measure, we have that

$$\delta_x(u) = \begin{cases} 1 & \text{if } u = x \\ 0 & \text{otherwise} \end{cases}$$

and thus $\sum_{v \in Y} w(u,v) = \delta_x(u)$;

– for each $v \in Y$,

$$\sum_{u \in X} w(u,v) = \begin{cases} 1 & \text{if } v = y \\ 0 & \text{otherwise} \end{cases}$$

By definition of Dirac measure, we have that

$$\delta_y(v) = \begin{cases} 1 & \text{if } v = y \\ 0 & \text{otherwise} \end{cases}$$

and thus $\sum_{u \in X} w(u,v) = \delta_y(v)$.

($\Leftarrow$) $\delta_x \; \mathcal{L}(\mathcal{R}) \; \delta_y$ implies that there exists $w\colon X \times Y \to [0,1]$ such that

– for each $u \in X$, $v \in Y$, $w(u,v) > 0 \implies u \; \mathcal{R} \; v$;
– for each $u \in X$, $\delta_x(u) = \sum_{v \in Y} w(u,v)$; and
– for each $v \in Y$, $\delta_y(v) = \sum_{u \in X} w(u,v)$.

For each $u \neq x$, we have that $\delta_x(u) = 0$ and thus for each $v \in Y$, $w(u,v) = 0$. So, only for $u = x$ we can have $w(x,v) > 0$. For each $v \neq y$, we have that $\delta_y(v) = 0$ and thus for each $u \in X$, $w(u,v) = 0$. This implies that only for $u = x$ and $v = y$ we have $w(x,y) > 0$ and thus $x \; \mathcal{R} \; y$.

2. Let $\mathcal{R}$ be a relation from X to Y.

($\Rightarrow$) Suppose, for the sake of contradiction, that $\mathcal{R} = \emptyset$ while $\mathcal{L}(\mathcal{R}) \neq \emptyset$. $\mathcal{L}(\mathcal{R}) \neq \emptyset$ implies that there exist measures $\rho_x \in \mathit{Disc}(X)$ and $\rho_y \in \mathit{Disc}(Y)$ such that $\rho_x \; \mathcal{L}(\mathcal{R}) \; \rho_y$ and thus there exists a weighting function $w\colon X \times Y \to [0,1]$. Since $\rho_x(X) = 1$, then there exists $u \in X$ such that $\rho_x(u) > 0$. By condition 3 of lifting, we have that $\rho_x(u) = \sum_{v \in Y} w(u,v) > 0$ and thus there exists $v \in Y$ such that $w(u,v) > 0$. By condition 1 of lifting, we have that $w(u,v) > 0$ implies $(u,v) \in \mathcal{R}$. This contradicts the hypothesis $\mathcal{R} = \emptyset$ and thus $\mathcal{R} = \emptyset \implies \mathcal{L}(\mathcal{R}) = \emptyset$.

($\Leftarrow$) We prove $\mathcal{R} \neq \emptyset \implies \mathcal{L}(\mathcal{R}) \neq \emptyset$ that is equivalent to $\mathcal{L}(\mathcal{R}) = \emptyset \implies \mathcal{R} = \emptyset$. So, suppose that $\mathcal{R} \neq \emptyset$. This implies that there exist $x \in X$, $y \in Y$ such that $(x,y) \in \mathcal{R}$. By Property 1, it follows that $\delta_x \; \mathcal{L}(\mathcal{R}) \; \delta_y$ and thus $\mathcal{L}(\mathcal{R}) \neq \emptyset$.

3. Let $\mathcal{R}, \mathcal{S}$ be two relations from X to Y such that $\mathcal{R} \subseteq \mathcal{S}$. Let $\rho_x \in \mathit{Disc}(X)$ and $\rho_y \in \mathit{Disc}(Y)$ be two measures such that $\rho_x \ \mathcal{L}(\mathcal{R}) \ \rho_y$. This implies that there exists $w \colon X \times Y \to [0,1]$ such that
   - for each $u \in X$, $v \in Y$, $w(u,v) > 0 \implies (u,v) \in \mathcal{R}$;
   - for each $u \in X$, $\delta_x(u) = \sum_{v \in Y} w(u,v)$; and
   - for each $v \in Y$, $\delta_y(v) = \sum_{u \in X} w(u,v)$.

   For each $u \in X$ and $v \in Y$, if $w(u,v) > 0$ then $(u,v) \in \mathcal{R} \subseteq \mathcal{S}$ and thus $(u,v) \in \mathcal{S}$. This implies that w is also a weighting function with respect to $\mathcal{S}$ and thus $\rho_x \ \mathcal{L}(\mathcal{S}) \ \rho_y$. This means that for each $\rho_x \in \mathit{Disc}(X)$ and $\rho_y \in \mathit{Disc}(Y)$, if $(\rho_x, \rho_y) \in \mathcal{L}(\mathcal{R})$ then $(\rho_x, \rho_y) \in \mathcal{L}(\mathcal{S})$ and thus $\mathcal{L}(\mathcal{R}) \subseteq \mathcal{L}(\mathcal{S})$.
4. Let $\mathcal{R}$ be a reflexive relation on X and $\rho \in \mathit{Disc}(X)$ be a measure. Define $w \colon X \times X \to [0,1]$ as

$$w(u,v) = \begin{cases} \rho(u) & \text{if } v = u, \\ 0 & \text{otherwise.} \end{cases}$$

   w is a weighting function:
   - for each $u, v \in X$, $w(u,v) > 0$ implies that $v = u$ and since $\mathcal{R}$ is reflexive, we have that $(u,u) \in \mathcal{R}$;
   - for each $u \in X$, $\sum_{v \in X} w(u,v) = w(u,u) = \rho(u)$;
   - for each $v \in X$, $\sum_{u \in X} w(u,v) = w(v,v) = \rho(v)$.
5. Let $\mathcal{R}$ be a symmetric relation on X and $\rho_1, \rho_2 \in \mathit{Disc}(X)$ be two measures such that $\rho_1 \ \mathcal{L}(\mathcal{R}) \ \rho_2$. This means that there exists $w \colon X \times X \to [0,1]$ such that
   - for each $u \in X$, $v \in Y$, $w(u,v) > 0 \implies (u,v) \in \mathcal{R}$;
   - for each $u \in X$, $\sum_{v \in X} w(u,v) = \rho_1(u)$; and
   - for each $v \in X$, $\sum_{u \in X} w(u,v) = \rho_2(v)$.

   Define $w' \colon X \times X \to [0,1]$ as $w'(u,v) = w(v,u)$ w' is a weighting function:
   - for each $u, v \in X$, $w'(u,v) > 0$ implies that $w(v,u) > 0$, thus $v \ \mathcal{R} \ u$ and since $\mathcal{R}$ is symmetric, we have that $u \ \mathcal{R} \ v$;
   - for each $u \in X$, $\sum_{v \in X} w'(u,v) = \sum_{v \in X} w(v,u) = \rho_2(u)$;
   - for each $v \in X$, $\sum_{u \in X} w'(u,v) = \sum_{u \in X} w(v,u) = \rho_1(v)$.

   This implies that $\rho_2 \ \mathcal{L}(\mathcal{R}) \ \rho_1$.
6. Let $\mathcal{R}$, $\mathcal{S}$ be two relations from X to Y and from Y to Z, respectively. Let $\rho_x \in \mathit{Disc}(X)$, $\rho_y \in \mathit{Disc}(Y)$, $\rho_z \in \mathit{Disc}(Z)$ be three probability measures such that $\rho_x \ \mathcal{L}(\mathcal{R}) \ \rho_y$ and $\rho_y \ \mathcal{L}(\mathcal{S}) \ \rho_z$. This implies that there exist $w_r \colon X \times Y \to [0,1]$ and $w_s \colon Y \times Z \to [0,1]$ such that

- for each $u \in X$, $v \in Y$, $w_r(u,v) > 0 \implies (u,v) \in \mathcal{R}$;
- for each $u \in X$, $\sum_{v \in Y} w_r(u,v) = \rho_x(u)$; and
- for each $v \in Y$, $\sum_{u \in X} w_r(u,v) = \rho_y(v)$.

and

- for each $u \in Y$, $v \in Z$, $w_s(u,v) > 0 \implies (u,v) \in \mathcal{S}$;
- for each $u \in Y$, $\sum_{v \in Z} w_s(u,v) = \rho_y(u)$; and
- for each $v \in Z$, $\sum_{u \in Y} w_s(u,v) = \rho_z(v)$.

Define $w_{rs} \colon X \times Z \to [0,1]$ as $w_{rs}(u,v) = \sum_{t \in Y, \rho_y(t) \neq 0} \dfrac{w_r(u,t) w_s(t,v)}{\rho_y(t)}$.

w_{rs} is a weighting function:

- for each $u \in X$ and $v \in Z$, since $w_{rs}(u,v) > 0$ we have that $\sum_{t \in Y, \rho_y(t) \neq 0} \dfrac{w_r(u,t) w_s(t,v)}{\rho_y(t)} > 0$ and thus there exists $q \in Y$ such that $\rho_y(q) \neq 0$ and $\dfrac{w_r(u,q) w_s(q,v)}{\rho_y(q)} > 0$. Since ρ_y is a probability measure, it follows that $\rho_y(q) \gneq 0$ and thus $w_r(u,q) w_s(q,v) > 0$ that implies $w_r(u,q) > 0$ and $w_s(q,v) > 0$. Hence we have that $(u,q) \in \mathcal{R}$ and $(q,v) \in \mathcal{S}$ and thus $(u,v) \in \mathcal{R} \circ \mathcal{S}$;
- for each $u \in X$,

$$\begin{aligned}
\sum_{v \in Z} w_{rs}(u,v) &= \sum_{v \in Z} \sum_{t \in Y, \rho_y(t) \neq 0} \frac{w_r(u,t) w_s(t,v)}{\rho_y(t)} \\
&= \sum_{t \in Y, \rho_y(t) \neq 0} \sum_{v \in Z} \frac{w_r(u,t) w_s(t,v)}{\rho_y(t)} \\
&= \sum_{t \in Y, \rho_y(t) \neq 0} \frac{w_r(u,t)}{\rho_y(t)} \sum_{v \in Z} w_s(t,v) \\
&= \sum_{t \in Y, \rho_y(t) \neq 0} \frac{w_r(u,t)}{\rho_y(t)} \rho_y(t) \\
&= \sum_{t \in Y, \rho_y(t) \neq 0} w_r(u,t) \\
&= \sum_{t \in Y} w_r(u,t) \\
&= \rho_x(u)
\end{aligned}$$

We can remove the condition on $\rho_y(t) \neq 0$ from the summation since by definition of weighting function, if $\rho_y(t) = 0$, then $\sum_{u \in X} w_r(u,t) = 0$ and thus for each $u \in X$, $w_r(u,t) = 0$;

– for each $v \in Z$,

$$\begin{aligned}
\sum_{u \in X} w_{rs}(u,v) &= \sum_{u \in X} \sum_{t \in Y, \rho_y(t) \neq 0} \frac{w_r(u,t) w_s(t,v)}{\rho_y(t)} \\
&= \sum_{t \in Y, \rho_y(t) \neq 0} \sum_{u \in X} \frac{w_r(u,t) w_s(t,v)}{\rho_y(t)} \\
&= \sum_{t \in Y, \rho_y(t) \neq 0} \frac{w_s(t,v)}{\rho_y(t)} \sum_{u \in X} w_r(u,t) \\
&= \sum_{t \in Y, \rho_y(t) \neq 0} \frac{w_s(t,v)}{\rho_y(t)} \rho_y(t) \\
&= \sum_{t \in Y, \rho_y(t) \neq 0} w_s(t,v) \\
&= \sum_{t \in Y} w_s(t,v) \\
&= \rho_z(v)
\end{aligned}$$

We can remove the condition on $\rho_y(t) \neq 0$ from the sum since by definition of weighting function, if $\rho_y(t) = 0$, then $\sum_{v \in Z} w_s(t,v) = 0$ and thus for each $v \in Z$, $w_s(t,v) = 0$.

This implies that w_{rs} is a weighting function from ρ_x to ρ_z and thus $\rho_x \; \mathcal{L}(\mathcal{R} \circ \mathcal{S}) \; \rho_z$.

7. Let $\mathcal{R}$ be a transitive relation on X and let $\rho_1, \rho_2, \rho_3 \in \mathit{Disc}(X)$ be three probability measures such that $\rho_1 \; \mathcal{L}(\mathcal{R}) \; \rho_2$ and $\rho_2 \; \mathcal{L}(\mathcal{R}) \; \rho_3$. By the Property 6, we have that $\rho_1 \; \mathcal{L}(\mathcal{R} \circ \mathcal{R}) \; \rho_1$. If $\mathcal{R} \circ \mathcal{R} \subseteq \mathcal{R}$, then by Property 3 we have that $\rho_1 \; \mathcal{L}(\mathcal{R}) \; \rho_3$, as required. So, let $x, z \in X$ be two states such that $(x,z) \in \mathcal{R} \circ \mathcal{R}$. By definition of composition, it follows that there exists $y \in X$ such that $(x,y) \in \mathcal{R}$ and $(y,z) \in \mathcal{R}$. Since $\mathcal{R}$ is transitive, we have that $(x,z) \in \mathcal{R}$ and thus $\mathcal{R} \circ \mathcal{R} \subseteq \mathcal{R}$.
8. Let $\mathcal{R}$ be a relation from X to Y and id be the identity relation on Z. Let ρ_x, ρ_y, ρ_z be three probability measures in $\mathit{Disc}(X)$, $\mathit{Disc}(Y)$, and $\mathit{Disc}(Z)$, respectively. Suppose that $\rho_x \; \mathcal{L}(\mathcal{R}) \; \rho_y$. This implies that there exists a weighting function $w \colon X \times Y \to [0,1]$ such that
   – for each $u \in X$, $v \in Y$, $w(u,v) > 0$ implies $u \; \mathcal{R} \; v$,
   – for each $u \in X$, $\sum_{v \in Y} w(u,v) = \rho_x(u)$, and
   – for each $v \in Y$, $\sum_{u \in X} w(u,v) = \rho_y(v)$.

   Let $w' \colon (X \times Z) \times (Y \times Z) \to [0,1]$ be a function defined as:

$$w'((u,s),(v,t)) = \begin{cases} w(u,v)\rho_z(s) & \text{if } u \mathcal{R} v \text{ and } s \; id \; t \\ 0 & \text{otherwise} \end{cases}$$

w' is a weighting function from $\rho_x \times \rho_z$ to $\rho_y \times \rho_z$. In fact,

- for each $(u,s) \in X \times Z$, $(v,t) \in Y \times Z$, $w'((u,s),(v,t)) > 0$ implies $u \mathcal{R} v$ and $s \; id \; t$ and thus $(u,s) \mathcal{R} \times id \; (v,t)$,
- for each $(u,s) \in X \times Z$,

$$\begin{aligned} \sum_{(v,t) \in Y \times Z} w'((u,s),(v,t)) &= \sum_{(v,t) \in Y \times Z, u \mathcal{R} v, s id t} w(u,v)\rho_z(s) \\ &= \rho_z(s) \sum_{v \in Y, u \mathcal{R} v} w(u,v) \\ &= \rho_z(s) \sum_{v \in Y} w(u,v) \\ &= \rho_z(s)\rho_x(u) \\ &= \rho_x \times \rho_z(u,s) \end{aligned}$$

- for each $(v,t) \in Y \times Z$,

$$\begin{aligned} \sum_{(u,s) \in X \times Z} w'((u,s),(v,t)) &= \sum_{(u,s) \in X \times Z, u \mathcal{R} v, s id t} w(u,v)\rho_z(s) \\ &= \rho_z(t) \sum_{u \in X, u \mathcal{R} v} w(u,v) \\ &= \rho_z(t) \sum_{u \in X} w(u,v) \\ &= \rho_z(t)\rho_y(v) \\ &= \rho_y \times \rho_z(v,t) \end{aligned}$$

□

2.3 Compatible Mappings

Let $\sigma\colon X \to Y$ be a partial mapping function. We denote by $\mathrm{Dom}(\sigma)$ the set of all $x \in X$ such that $\sigma(x)$ is defined.

Two partial mapping functions $\sigma\colon X \to Y$ and $\sigma'\colon X' \to Y'$ are *compatible* if and only if for each $x \in \mathrm{Dom}(\sigma) \cap \mathrm{Dom}(\sigma')$, $\sigma(x) = \sigma'(x)$.

In other words, two mappings are compatible if they assign the same value to the common points.

2.4 Probabilistic I/O Automata

A *Probabilistic Automaton* is a tuple $(S, \bar{s}, A, D)$ where S is a set of *states*, $\bar{s} \in S$ is the *start state*, A is a set of *actions*, and $D \subseteq S \times A \times Disc(S)$ is a *transition relation*. The set of actions A is further partitioned into three sets I, O, H of input, output and internal (hidden) actions, respectively. We call the set $E = I \cup O$ the set of external actions.

Throughout the thesis we let $\mathcal{A}$ range over probabilistic automata, q, r, s range over states, a, b, c range over actions, and μ range over discrete measures over states. We also denote the generic elements of a probabilistic automaton $\mathcal{A}$ by S, $\bar{s}$, A, D, and we propagate primes and indices when necessary. Thus, for example, the probabilistic automaton $\mathcal{A}'_i$ has states S'_i, start state $\bar{s}'_i$, actions A'_i and transition relation D'_i. We also denote the start state of a probabilistic automaton $\mathcal{A}$, $\mathcal{B}$, ... by $\bar{a}$, $\bar{b}$, ..., respectively.

An element of a transition relation D is called a *transition* or a *step*. A transition $tr = (s, a, \mu)$, denoted alternatively by $s \xrightarrow{a} \mu$, is said to *leave* from state s, denoted also by $src(tr)$, to be *labeled* by a, denoted by $action(tr)$, and to *lead* to μ, denoted by $trg(tr)$ or μ_{tr}. We also say that state s *enables* action a, that action a is *enabled* from s, and that (s, a, μ) is enabled from s.

Given an action a, we denote by $D(a)$ the set of all transitions labeled by a, that is $D(a) = \{tr \in| \ action(tr) = a\}$.

Given a state s and an action a, we say that there exists a *combined transition* $s \xrightarrow{a}_C \mu$ if there exists a countable family of transitions $\{(s, a, \mu_i)\}_{i \in I}$ of $\mathcal{A}$ and a family of probabilities $\{p_i\}_{i \in I}$ such that $\sum_{i \in I} p_i = 1$ and $\mu = \sum_{i \in I} p_i \mu_i$

Let $\mathcal{A}$ be a probabilistic automaton and let $\mu \in Disc(S)$. For each $s \in Supp(\mu)$, let $s \xrightarrow{a} \mu_s$ be a combined transition of $\mathcal{A}$. Let μ' be $\sum_{s \in Supp(\mu)} \mu(s)\mu_s$. Then $\mu \xrightarrow{a} \mu'$ is called a *hyper-transition* of $\mathcal{A}$.

We adopt the notation used by Lynch in [72] to describe automata (see, for example, the coin flipper automaton of Figure 2.1). Each automaton is described by three parts: *signature*, *states*, and *transitions*. The signature lists the actions of the automaton, partitioned into *input*, *output*, and *internal*. Each action has a name, a sequence of parameters, and a set of values each parameter can assume. The states are described by a set of variables. Each variable assumes values in a given set, and the start state is identified by the initial value of each variable. Transitions describe the transition relation of the automaton. We provide a transition for each action of the

Coin Flipper

Signature:
Output:
$coin\_value(v)$, $v \in \{H, T\}$
Input:
flip
Internal:
coin_flip

State:
$value \in \{H, T\} \cup \{F, \bot\}$, initially $\bot$

Transitions:

Input *flip*
Effect:
$value := F$

Internal *coin_flip*
Precondition:
$value = F$
Effect:
$value := c$ where $c \in_R \{H, T\}$

Output $coin\_value(v)$
Precondition:
$v = value$
Effect:
$value := \bot$

Fig. 2.1. A coin flipper

signature: it contains the kind of the action (input, output, or internal), the name and the effect. Output and internal actions have also a precondition that specifies when they are enabled. Input actions are assumed to be always enabled, and thus no precondition is specified for them. The effect characterizes the states that are reached performing the transition: if we consider the action *flip* of the coin flipper of Figure 2.1, the next state is unique and it is the one with $value = F$. On the contrary, performing a *coin_flip* action, we can reach both states identified by $value = T$ and $value = H$ with probability one half. In particular, we use the symbol $\in$ to denote the fact that a value is chosen arbitrarily from some set, and we use the symbol $\in_R$ to denote the fact that a value is chosen randomly and uniformly from a finite set.

2.4.1 Executions, Traces and Schedulers

An *execution fragment* of a probabilistic automaton $\mathcal{A}$ is a sequence of alternating states and actions, $\alpha = s_0 a_1 s_1 \ldots$, starting with a state and, if the sequence is finite, ending with a state, such that, for each non final index i, there exists a transition $(s_i, a_{i+1}, \mu_{i+1})$ in D with $\mu_{i+1}(s_{i+1}) > 0$. We denote the first state s_0 of α by $\mathit{fstate}(\alpha)$. We say that an execution fragment is *finite* if it is a finite sequence, and we denote the last state of a finite execution fragment α by $\mathit{lstate}(\alpha)$. We define the *length* of an execution fragment α, denoted by $|\alpha|$, to be the number of occurrences of actions in α. The *trace* of an execution fragment α, denoted by $\mathit{trace}(\alpha)$, is the subsequence of external actions that occur in α. An *execution* of a probabilistic automaton $\mathcal{A}$ is an execution fragment of $\mathcal{A}$ whose first state is $\bar{s}$. We denote by $\mathit{Frags}^*(\mathcal{A})$ the set of finite execution fragments of $\mathcal{A}$, by $\mathit{Frags}(\mathcal{A})$ the set of finite or infinite execution fragments, and by $\mathit{Execs}^*(\mathcal{A})$, $\mathit{Execs}(\mathcal{A})$ the corresponding sets of executions. We let ν, υ range over discrete probability measures over finite executions of $\mathcal{A}$, that is, $\nu, \upsilon \in \mathit{Disc}(\mathit{Execs}^*(\mathcal{A}))$. A *scheduler* for a probabilistic automaton $\mathcal{A}$ is a function $\sigma \colon \mathit{Frags}^*(\mathcal{A}) \to \mathit{SubDisc}(D)$ such that, for each finite execution fragment α and each transition tr with $\sigma(\alpha)(tr) > 0$, $\mathit{src}(tr) = \mathit{lstate}(\alpha)$. A scheduler σ is deterministic if, for each finite execution fragment α, $\sigma(\alpha)$ is a Dirac sub-measure.

A scheduler σ can be used to describe the result of resolving nondeterminism starting from some state s. Specifically, a scheduler σ and a state s induce a probability measure $\varepsilon_{\sigma,s}$ over execution fragments as follows. The basic measurable events are the cones of finite execution fragments, where the cone of a finite execution fragment α, denoted by C_α, is the set $\{\alpha' \in \mathit{Frags}(\mathcal{A}) \mid \alpha \leqslant \alpha'\}$, where $\leqslant$ is the standard prefix preorder on sequences. The probability $\varepsilon_{\sigma,s}$ of a cone C_α is defined recursively as follows:

$$\varepsilon_{\sigma,s}(C_\alpha) = \begin{cases} 0 & \text{if } \alpha = q \text{ with } q \neq s \\ 1 & \text{if } \alpha = s \\ \varepsilon_{\sigma,s}(C_{\alpha'}) \sum_{tr \in D(a)} \sigma(\alpha)(tr)\mu_{tr}(q) & \text{if } \alpha = \alpha' a q \end{cases}$$

Standard measure theoretical arguments ensure that $\varepsilon_{\sigma,s}$ extends uniquely to the σ-field generated by cones. We call the measure $\varepsilon_{\sigma,s}$ a *probabilistic execution fragment* of $\mathcal{A}$ and we say that it is generated by σ from s. If s is the start state of $\mathcal{A}$, then we say that $\varepsilon_{\sigma,s}$ is a *probabilistic execution*.

2.4.2 Parallel Composition

Given two probabilistic automata, it is possible to define their composition, that is the automaton that behaves as the two automata when they run in parallel. We adopt the definition of [102], where automata are synchronized on their common actions.

Two probabilistic automata $\mathcal{A}_1$, $\mathcal{A}_2$ are *compatible* if $H_1 \cap A_2 = \emptyset$ and $A_1 \cap H_2 = \emptyset$, that is, the internal actions of each automaton are not actions of the other automaton.

The composition of two compatible probabilistic automata $\mathcal{A}_1$, $\mathcal{A}_2$, denoted by $\mathcal{A}_1||\mathcal{A}_2$, is a probabilistic automaton $\mathcal{A}$ where

- $S = S_1 \times S_2$,
- $\bar{s} = (\bar{s}_1, \bar{s}_2)$,
- $E = E_1 \cup E_2$, $H = H_1 \cup H_2$, and
- D is defined as follows: $((s_1, s_2), a, \mu_1 \times \mu_2) \in D$ if and only if, for each $i \in \{1, 2\}$,
  - either $a \in A_i$ and $(s_i, a, \mu_i) \in D_i$,
  - or $a \notin A_i$ and $\mu_i = \delta_{s_i}$,

  where $\mu_1 \times \mu_2((s'_1, s'_2))$ is defined as $\mu_1(s'_1)\mu_2(s'_2)$.

Given a probabilistic automaton $\mathcal{A}$, we say that the probabilistic automaton $\mathcal{A}'$ is a context for $\mathcal{A}$ if $\mathcal{A}$ and $\mathcal{A}'$ are compatible.

Given the composed automaton $\mathcal{A}_1||\mathcal{A}_2$ and the measure $\mu \in Disc(S_1 \times S_2)$, we denote by $\mu\lceil\mathcal{A}_1$ and $\mu\lceil\mathcal{A}_2$ the two measures on S_1 and S_2, respectively, defined as:

- for each $s_1 \in S_1$, $\mu\lceil\mathcal{A}_1(s_1) = \sum_{s_2 \in S_2} \mu(s_1, s_2)$;
- for each $s_2 \in S_2$, $\mu\lceil\mathcal{A}_2(s_1) = \sum_{s_1 \in S_1} \mu(s_1, s_2)$.

2.4.3 Action Hiding

Given an automaton, it is possible to hide some of its external actions from the environment. This permits to make hidden actions private and available only for specific automata. For example, consider a dice roller that models the following algorithm:

1. Flip a coin; let s the result.
2. If $s = H$, then roll a fair dice,
3. Otherwise, roll an unfair dice.

with the constraint that the value s is not revealed to the environment.

The dice roller can either flip the coin internally or require it to the coin flipper. In the latter case, we need to keep secret the communication of s between the two automata. If we want to comply to the given constraint, then we must hide the $coin\_value(v)$ action, otherwise the value of the coin is revealed to each automaton that provides the action $coin\_value(v)$.

We adopt the definition of [102]: let $\mathcal{A}$ be a probabilistic automaton with external actions E_A and internal actions H_A. Let I be a set of actions. Then $\text{Hide}_I(\mathcal{A})$ is defined to be the probabilistic automaton $\mathcal{B}$ that is the same as $\mathcal{A}$ except that $E_B = E_A \setminus I$ and $H_B = H_A \cup I$. That is, the actions in the set I are hidden from the external environment.

2.4.4 Simulation and Weak Simulation

A *simulation* from a probabilistic automaton $\mathcal{A}_1$ to probabilistic automaton $\mathcal{A}_2$ is a relation $\mathcal{R}$ from S_1 to S_2 such that

- $\bar{s}_1 \; \mathcal{R} \; \bar{s}_2$ and
- for each pair $(s_1, s_2) \in \mathcal{R}$, if $(s_1, a, \mu_1) \in D_1$, then there exists $(s_2, a, \mu_2) \in D_2$ such that $\mu_1 \; \mathcal{L}(\mathcal{R}) \; \mu_2$.

We say that $\mathcal{A}_1$ is simulated by $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \preceq \mathcal{A}_2$, if there exists a simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$.

Before defining the notion of weak simulation, we need to introduce the weak transition: given a probabilistic automaton $\mathcal{A}$, we say that there is a weak combined transition from a state s to a measure over states μ labeled by an action a, denoted by $s \stackrel{a}{\Longrightarrow}_C \mu$, if there is a scheduler σ such that the following holds for the induced measure $\varepsilon_{\sigma,s}$:

1. $\varepsilon_{\sigma,s}(\mathit{Frags}^*(\mathcal{A})) = 1$;
2. for each $\alpha \in \mathit{Frags}^*(\mathcal{A})$, if $\varepsilon_{\sigma,s}(\alpha) > 0$, then $\mathit{trace}(\alpha) = \mathit{trace}(a)$;
3. for each $q \in S$, $\varepsilon_{\sigma,s}(\{\alpha \in \mathit{Frags}^*(\alpha) \mid \mathit{lstate}(\alpha) = q\}) = \mu(q)$.

A *weak simulation* from a probabilistic automaton $\mathcal{A}_1$ to probabilistic automaton $\mathcal{A}_2$ is a relation $\mathcal{R}$ from S_1 to S_2 such that

- $\bar{s}_1 \; \mathcal{R} \; \bar{s}_2$ and
- for each pair $(s_1, s_2) \in \mathcal{R}$, if $s_1 \xrightarrow{a} \mu_1$, then there exists $s_2 \stackrel{a}{\Longrightarrow}_C \mu_2$ such that $\mu_1 \; \mathcal{L}(\mathcal{R}) \; \mu_2$.

We say that $\mathcal{A}_1$ is weakly simulated by $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \preccurlyeq \mathcal{A}_2$, if there exists a weak simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$.

It is straightforward to define the bounded weak simulation: given a probabilistic automaton $\mathcal{A}$ and a bound $l \in \mathbb{N}$, we say that there is a weak l-bounded combined transition from a state s to a measure over states μ labeled by an action a, denoted by $s \stackrel{a}{\Longrightarrow}{}^{l}_{C} \mu$, if there is a scheduler σ such that the following holds for the induced measure $\varepsilon_{\sigma,s}$:

1. $\varepsilon_{\sigma,s}(\mathit{Frags}^*(\mathcal{A})) = 1$;
2. for each $\alpha \in \mathit{Frags}^*(\mathcal{A})$, if $\varepsilon_{\sigma,s}(\alpha) > 0$, then $\mathit{trace}(\alpha) = \mathit{trace}(a)$ and $|\alpha| \leqslant l$;
3. for each $q \in S$, $\varepsilon_{\sigma,s}(\{\alpha \in \mathit{Frags}^*(\alpha) \mid \mathit{lstate}(\alpha) = q\}) = \mu(q)$.

A *weak l-bounded simulation* from a probabilistic automaton $\mathcal{A}_1$ to probabilistic automaton $\mathcal{A}_2$ is a relation $\mathcal{R}$ from S_1 to S_2 such that

- $\bar{s}_1 \mathcal{R} \bar{s}_2$ and
- for each pair $(s_1, s_2) \in \mathcal{R}$, if $s_1 \xrightarrow{a} \mu_1$, then there exists $s_2 \stackrel{a}{\Longrightarrow}{}^{l}_{C} \mu_2$ such that $\mu_1 \, \mathcal{L}(\mathcal{R}) \, \mu_2$.

We say that $\mathcal{A}_1$ is weakly l-bounded simulated by $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \preccurlyeq^{l} \mathcal{A}_2$, if there exists a weak l-bounded simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$. Note that the weak l-bounded simulation is a special case of the weak simulation.

It is known that relations $\preceq$, $\preccurlyeq$, and $\preccurlyeq^{l}$ are transitive and preserved by action hiding and parallel composition of probabilistic automata [102]. This is the key feature that enables hierarchical and modular verification.

2.5 Cryptography

In the following we assume that k is a security parameter and that *Poly* is the set of positive polynomials over $\mathbb{N}$.

Given $f \colon \mathbb{N} \to \mathbb{R}^{\geqslant 0}$, we say that f is *negligible* if for each $c \in \mathbb{N}$ there exists $\bar{n} \in \mathbb{N}$ such that for each $n > \bar{n}$, $f(n) < n^{-c}$.

2.5.1 Oracle Machines

A *probabilistic oracle machine* is a probabilistic Turing machine with an additional tape, called the *oracle tape*, and two special states, called *oracle invocation* and *oracle appeared*. The computation of the probabilistic oracle machine M on input x and with access to the oracle $f \colon \{0,1\}^* \to \{0,1\}^*$ is defined by the *successive-configuration relation*. For configurations with states different from *oracle invocation*, the next configuration is defined as

usual. Let γ be a configuration in which the state is *oracle invocation* and the content of the oracle tape is q. Then the configuration following γ is identical to γ, except that the state is *oracle appeared*, and the content of the oracle tape is $f(q)$. The string q is called M's *query*, and $f(q)$ is called the *oracle reply*. The output measure of the oracle machine M, on input x and with access to the oracle f, is denoted $M^f(x)$.

We remark that the running time of an oracle machine is the number of steps made during its computation and that the oracle's reply to each query is obtained in a single step.

2.5.2 Nonce

A *nonce* of length k is an element of $\{0,1\}^k$ that is meant to be used at most once. An ideal way to satisfy unicity of nonces is to use a repository that keeps track of the nonces distributed in the past and that responds to all requests by returning a new value each time. The practical way to satisfy the unicity of nonces is to choose them randomly from $\{0,1\}^k$. In this way, if we choose randomly one nonce of length k, the probability that it is the same of some previously chosen nonce is at most 2^{-k}. This means that:

Proposition 2.2. *For each $c \in \mathbb{N}$ and $p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$, if values $n_1, \ldots, n_{p(k)} \in \{0,1\}^k$ are given and a value n is chosen randomly from $\{0,1\}^k$, then $\boldsymbol{Pr}[n = n_i \mid 1 \leqslant i \leqslant p(k)] < k^{-c}$.*

Proof. Since for each $i = 1, \ldots, p(k)$ the probability that n is equal to n_i is 2^{-k}, then

$$\begin{aligned}\mathbf{Pr}[n = n_i \mid 1 \leqslant i \leqslant p(k)] &\leqslant \mathbf{Pr}[n = n_1] + \cdots + \mathbf{Pr}[n = n_{p(k)}] \\ &\leqslant \underbrace{2^{-k} + \cdots + 2^{-k}}_{p(k) \text{ times}} \\ &= p(k)2^{-k} \\ &< k^{-c}\end{aligned}$$

The last step is justified by the following argument: let c' be a constants such that $p(k)k^{-c'} < k^{-c}$ (that is, $k^{-c'} < k^{-c}/p(k)$). Now, each time $2^{-k} < k^{-c'}$ is satisfied, then $p(k)2^{-k} < k^{-c}$ holds too. $2^{-k} < k^{-c'}$ is true for each k such that $k/\log(k) > c'/\log(2)$. □

2.5.3 Pseudorandom Functions

A *pseudorandom function* P is a function that can not be distinguished from a truly random function R by any efficient procedure (e.g., probabilistic polynomial time algorithm). More precisely, given a pseudorandom function P and a truly random function R, if we evaluate them on a polynomial number of values, then the probability to distinguish between the interaction with P and the interaction with R is negligible.

Formally, we say that $\{f_s\colon \{0,1\}^* \to \{0,1\}^k\}_{s\in\{0,1\}^*}$ is a *pseudorandom function* if the following two conditions hold:

1. There exists a polynomial time algorithm that on input s and $x \in \{0,1\}^*$ returns $f_s(x)$.
2. For every probabilistic polynomial time machine M that samples values from a function f and returns a value in $\{0,1\}$, every $p \in \mathit{Poly}$, there exists $\bar{n} \in \mathbb{N}$ such that for each $n > \bar{n}$,

$$|\mathbf{Pr}[M^{F_n}(1^n) = 1] - \mathbf{Pr}[M^{H_n}(1^n) = 1]| < \frac{1}{p(n)}$$

where F_n is a random variable uniformly distributed over the multiset $\{f_s\}_{s\in\{0,1\}^n}$, H_n is a random variable uniformly distributed among all functions mapping arbitrarily long strings to strings of length k, $\mathbf{Pr}[M^{F_n}(1^n) = 1]$ is the probability that the machine M, on input 1^n, answers 1 provided that f is chosen according to F_n, and $\mathbf{Pr}[M^{H_n}(1^n) = 1]$ is the probability that the machine M, on input 1^n, answers 1 provided that f is chosen according to H_n.

This definition of pseudorandom function conceals technical aspects that are out the scope of this thesis. Interested readers can find a justification of such technicalities and a generalized definition of pseudorandom functions in Section 3.6 of [49].

2.5.4 Message Authentication Code

A *message authentication scheme* is a triple (G, A, V) of probabilistic polynomial time algorithms satisfying the following two conditions:

1. On input 1^k, algorithm G (called the *key-generator*) outputs a bit string.
2. For every s in the range of $G(1^k)$ and for every $\alpha \in \{0,1\}^*$, algorithms A (authentication) and V (verification) satisfy $\mathbf{Pr}[V(s, \alpha, A(s, \alpha)) = 1] = 1$ where the probability is taken over the internal coin tosses of algorithms A and V.

We call $A(s, \alpha)$ a *message authentication code* (MAC) to the document α produced using the key s.

A *forger* is a process that, on input 1^k, can obtain message authentication codes to strings of its choice, relative to a key s that is generated by $G(1^k)$ and that the forger does not know. It can also verify pairs of the form (α, β) querying the verification algorithm. Such a forger is said to *succeed* (in existential forgery) if it outputs a valid MAC to a string for which it has not requested an authentication during the attack. That is, the forger is successful if it outputs a pair (α, β) such that $V(s, \alpha, \beta) = 1$ and α is different from all strings for which an authentication has been required during the attack. A message authentication scheme is *secure* (or unforgeable) if every feasible forger succeeds with at most negligible probability.

A way to construct message authentication schemes is to use pseudorandom functions, using the following construction (cf. Construction 6.3.1 of [50]): let $\{f_s\}_{s\in\{0,1\}^*}$ be a pseudorandom function. We define a message authentication scheme (G, A, V) as follows:

- Key generation with G: on input 1^k, we uniformly select $s \in \{0,1\}^k$ and output the key s.
- Authentication with A: on input a key $s \in \{0,1\}^k$ and a string $\alpha \in \{0,1\}^*$, we compute and output $f_s(\alpha)$ as an authentication of α.
- Verification with V: on input a key $s \in \{0,1\}^k$, a string $\alpha \in \{0,1\}^*$, and an alleged authentication β, we accept if and only if $\beta = f_s(\alpha)$.

Given a key s, we say that $f_s(m)$ is the message authentication code of m with respect to the key s and that f_s is a MAC value generator.

Proposition 2.3 (cf. Proposition 6.3.2 [50]). *Suppose that $\{f_s\}_{s\in\{0,1\}^*}$ is a pseudorandom function. Then the given construction constitutes a secure message authentication scheme.*

A message authentication code can be used when an entity A wants to prove its identity to another entity B. If A and B share a secret key s and a pseudorandom function, then A can provide evidence of its identity by sending a message of the form $(a.m, f_s(a.m))$ to B, where m is some random value, a is a coding of the identity of A, and $a.m$ is the concatenation of a and m. B can rely on A's identity by verifying the correctness of the received message.

2.5.5 Signature Scheme

A *signature scheme* is a triple (G, S, V) of probabilistic polynomial time algorithms satisfying the following two conditions:

1. On input 1^k, algorithm G (called the *key-generator*) outputs a pair of bit strings.
2. For every pair (s, v) in the range of $G(1^k)$ and for every $\alpha \in \{0,1\}^*$, algorithms S (signing) and V (verification) satisfy $\mathbf{Pr}[V(v, \alpha, S(s, \alpha)) = 1] = 1$ where the probability is taken over the internal coin tosses of algorithms S and V.

We call $S(s, \alpha)$ a *signature* to the document α produced using the signing key s. Likewise, when $V(v, \alpha, \beta) = 1$, we say that β is a *valid signature* to α with respect to the verification key v.

Given a pair of keys (s, v) generated by $G(1^k)$, a *forger* for signatures is a process that, on input v, can obtain signatures to strings of its choice, produced using the key s that the forger does not know. Such a forger is said to *succeed* (in existential forgery) if it outputs a valid signature to a string for which it has not requested a signature during the attack. That is, the forger is successful if it outputs a pair (α, β) such that $V(v, \alpha, \beta) = 1$ and α is different from all strings for which a signature has been required during the attack. A signature scheme is *secure* (or unforgeable) if every feasible forger succeeds with at most negligible probability.

Message authentication schemes and signature schemes are very similar. The main difference between them is about keys generated by key generator G: a single key for message authentication schemes; a pair of keys for signature schemes. In particular, message authentication scheme can be seen as an instantiation of a signature scheme where $v = s$ and the value 1^k is provided to the forger instead of v.

2.5.6 Encryption Scheme

An *encryption scheme* is a triple (G, E, D) of probabilistic polynomial time algorithms satisfying the following two conditions:

1. On input 1^k, algorithm G (called the *key-generator*) outputs a pair of bit strings.
2. For every pair (e, d) in the range of $G(1^k)$, and for every $\alpha \in \{0,1\}^*$, algorithms E (*encryption*) and D (*decryption*) satisfy $\mathbf{Pr}[D(d, E(e, \alpha)) =$

$\alpha] = 1$ where the probability is taken over the internal coin tosses of algorithms E and D.

The integer k serves as the *security parameter* of the scheme. Each (e, d) in the range of $G(1^k)$ constitutes a pair of corresponding *encryption/decryption keys*. The string $E(e, \alpha)$ (denoted also $E_e(\alpha)$) is the *encryption of the plaintext* $\alpha \in \{0,1\}^*$ using the encryption key e, whereas $D(d, \beta)$ (denoted also $D_d(\beta)$) is the *decryption of the ciphertext* β using the decryption key d.

A *public key encryption scheme* is an encryption scheme where the decryption key d differs from encryption key e, d is kept secret while e is published, and it is infeasible to find d given e.

A public key encryption scheme (G, E, D) is said to have the *indistinguishable encryptions under (a posteriori) chosen ciphertext attacks* (or public key encryption scheme (G, D, E) is *IND-CCA*) if for every pair of probabilistic polynomial time oracle machine, A_1 and A_2, for every $p \in Poly$, there exist $q \in Poly$, $\bar{n} \in \mathbb{N}$ such that for each $n > \bar{n}$ and $z \in \{0,1\}^{q(n)}$ it holds that

$$|p_{1,n,z} - p_{2,n,z}| < \frac{1}{p(n)}$$

where

$$p_{i,n,z} \stackrel{\text{def}}{=} \mathbf{Pr}\left[\begin{array}{l} v = 1 \text{ where} \\ \qquad (e,d) := G(1^n) \\ \qquad ((x_1,x_2),\sigma) := A_1^{E_e,D_d}(e,z) \\ \qquad c := E_e(x_i) \\ \qquad v := A_2^{E_e,D_d}(\sigma,c) \end{array}\right]$$

where $|x_1| = |x_2|$ and A_2 is not allowed to make the query c to the oracle D_d.

The definition of IND-CCA is based on two oracle machines A_1 and A_2: A_1 interacts with encryption and decryption collecting information inside σ and then generates two strings x_1 and x_2 of the same length. Only one of these two strings is encrypted by the encryption algorithm. Finally, A_2 tries to discover which string corresponds to the ciphertext c. To do this, it interacts with encryption and decryption using the initial knowledge σ and c. We impose that A_2 can not require the decryption of c (since if it can invoke D_d on c, then it is immediate to know when c is the encryption of x_1 or the encryption of x_2). Note that the IND-CCA security easily implies that it is infeasible to derive d from e and that d is kept secret. In fact, if

it is feasible to derive d from e, then it is feasible for A_2 to decrypt c and correctly distinguish between an encryption of x_1 and x_2.

Proposition 2.4. *Let* $\mathsf{E} = (G, E, D)$ *be an encryption scheme and let* Ciphertext *be the set of messages that can be generated by* E*.*

If E *is IND-CCA, then for each* $g \in \mathbb{N}$ *and* $p \in Poly$*, there exists* $\bar{k} \in \mathbb{N}$ *such that for each* $k > \bar{k}$*, if values* $c_1, \ldots, c_{p(k)} \in$ Ciphertext *are given and a value* c *is obtained invoking* $E_e(m)$ *for a message* m *and a key* e *generated by* $G(1^k)$*, then* $\boldsymbol{Pr}[c = c_i \mid 1 \leqslant i \leqslant p(k)] < k^{-g}$*.*

Proof. Suppose, for the sake of contradiction, that $\mathbf{Pr}[c = c_i \mid 1 \leqslant i \leqslant p(k)] \geqslant k^{-g}$. We now define a distinguisher for the encryption scheme E.

Let B a machine that is initialized with a value b chosen randomly in $\{0, 1\}$ and that, on input $encrypt(e, m_0, m_1)$, returns $E_e(m_b)$ and, on input get_enc_key, returns e where $(e, d) = G(1^k)$.

Let A be an attacker that interacts with B as follows: it sends to B an get_enc_key request, obtaining an encryption key e. Then, it chooses two different messages m_0 and m_1 of the same length and then it sends to B $p(k)$ times the request $encrypt(e, m_0, m_1)$. Let $C = \{c_1, \ldots, c_{p(k)}\}$ be the set of returned values. Finally, A sends to B the request $encrypt(e, m_0, m_0)$; let c be the returned value. If $c \in C$, then A output 0, 1 otherwise.

A is a distinguisher for B and hence for the encryption scheme. In fact, if $b = 1$, then A outputs 1 with probability 1, since C contains only encryptions of message m_1. This implies that we can not find $c_i \in C$ such that c_i is the encryption of m_0 since by definition of encryption scheme, there does not exist two messages α and β such that $\alpha \neq \beta$ and given the pair of encryption keys $(e, d) = G(1^k)$, $D_d(E_e(\alpha)) = \beta$. If $b = 0$, then C contains only encryptions of m_0 and, by hypothesis, $c \in C$ with probability $p \geqslant k^{-c}$. So, with probability $p \geqslant k^{-c}$, A outputs 0 and with probability $1 - p \leqslant 1 - k^{-c}$ A outputs 1.

This implies that the $|\mathbf{Pr}[A \text{ outputs } 1 \mid b = 1] - \mathbf{Pr}[A \text{ outputs } 1 \mid b = 0]| = |1 - (1 - p)| = p \geqslant k^{-c}$ and this contradicts the IND-CCA property of E. □

3

Polynomially Accurate Simulations

In this chapter we define our notion of polynomially accurate simulation relation. Our aim is to define a relation where transitions are matched up to some error that is smaller than any polynomial in some security parameter k provided that computations are of polynomial length. This means that we need a relation that can "see" lengths of computations, a notion of lifting that accounts for errors, and a notion of security parameter. Furthermore, since we will also need ways to match sequences of steps, we need a way to bound the amount of extra error introduced by each step. We try to address one issue at a time, getting closer and closer to our desired notion of simulation.

The first step is to define a relation that can "see" lengths of computation. For this purpose, we define a relation on sets of executions rather than sets of states. This definition is based on a derived notion of transition that shows how finite executions evolve in a single step.

Definition 3.1. *We say that there is a step from a finite execution α to a measure $\nu \in Disc(Execs^*(\mathcal{A}))$, denoted by $\alpha \longrightarrow \nu$, if there exists a transition $(lstate(\alpha), a, \mu)$ such that, for each finite execution αas, $\nu(\alpha as) = \mu(s)$.*

Now we are able to define a simulation that relates executions instead of single states. This allows us to know how many steps we have performed in a computation, since we can obtain them from the length of the execution.

Definition 3.2. *An* execution simulation *from a probabilistic automaton $\mathcal{A}_1$ to a probabilistic automaton $\mathcal{A}_2$ is a relation $\mathcal{R}$ from $Execs^*(\mathcal{A}_1)$ to $Execs^*(\mathcal{A}_2)$ such that:*

- $\bar{s}_1 \mathcal{R} \bar{s}_2$ *and*

- *for each pair* $(\alpha_1, \alpha_2) \in \mathcal{R}$*, if* $\alpha_1 \longrightarrow \nu_1$*, then there exists* ν_2 *such that* $\alpha_2 \longrightarrow \nu_2$ *and* $\nu_1 \; \mathcal{L}(\mathcal{R}) \; \nu_2$*.*

We say that $\mathcal{A}_1$ *is execution simulated by* $\mathcal{A}_2$*, denoted by* $\mathcal{A}_1 \preceq_e \mathcal{A}_2$*, if there exists an execution simulation from* $\mathcal{A}_1$ *to* $\mathcal{A}_2$*.*

It is interesting to observe that so far we have not introduced anything new since $\preceq$ and $\preceq_e$ coincide.

Proposition 3.3. *Let* $\mathcal{A}_1$*,* $\mathcal{A}_2$ *be two probabilistic automata. Then* $\mathcal{A}_1 \preceq_e \mathcal{A}_2$ *if and only if* $\mathcal{A}_1 \preceq \mathcal{A}_2$*.*

Proof. ($\Leftarrow$) Given a simulation relation $\mathcal{R}$ from S_1 to S_2, define a relation $\mathcal{R}'$ from $Execs^*(\mathcal{A}_1)$ to $Execs^*(\mathcal{A}_2)$ such that $\alpha_1 \; \mathcal{R}' \; \alpha_2$ if and only if $lstate(\alpha_1) \; \mathcal{R} \; lstate(\alpha_2)$.

Start state condition is trivially true, since by hypothesis $\bar{s}_1 \; \mathcal{R} \; \bar{s}_2$ and thus $\bar{s}_1 \; \mathcal{R}' \; \bar{s}_2$ (with $\alpha_1 = \bar{s}_1$ and $\alpha_2 = \bar{s}_2$).

For the step condition, take α_1, α_2, ν_1 such that $\alpha_1 \; \mathcal{R}' \; \alpha_2$ and $\alpha_1 \longrightarrow \nu_1$. We must find ν_2 such that $\alpha_2 \longrightarrow \nu_2$ and $\nu_1 \; \mathcal{L}(\mathcal{R}') \; \nu_2$. By definition of $\alpha \longrightarrow \nu$, it follows that there exists a transition $(lstate(\alpha_1), a, \mu_1)$ such that, for each finite execution $\alpha_1 a s$, $\nu_1(\alpha_1 a s) = \mu_1(s)$. Since $\alpha_1 \; \mathcal{R}' \; \alpha_2$, then $lstate(\alpha_1) \; \mathcal{R} \; lstate(\alpha_2)$ and thus there must exist a transition $(lstate(\alpha_2), a, \mu_2)$ such that $\mu_1 \; \mathcal{L}(\mathcal{R}) \; \mu_2$. This means that there exists ν_2 such that $\alpha_2 \longrightarrow \nu_2$ and for each finite execution $\alpha_2 a s$, $\nu_2(\alpha_2 a s) = \mu_2(s)$. To complete the proof, we must verify that $\nu_1 \; \mathcal{L}(\mathcal{R}') \; \nu_2$. To do this, let w be the weighting function that justifies $\mu_1 \; \mathcal{L}(\mathcal{R}) \; \mu_2$ and define

$$w'(\beta_1, \beta_2) = \begin{cases} w(s_1, s_2) & \text{if } \beta_1 = \alpha_1 a s_1 \text{ and } \beta_2 = \alpha_2 a s_2 \\ 0 & \text{otherwise} \end{cases}$$

for each pair of executions (β_1, β_2).

Condition 1 of lifting is satisfied by w'. In fact if $w'(\beta_1, \beta_2) > 0$, then $w(lstate(\beta_1), lstate(\beta_2)) > 0$, thus $lstate(\beta_1) \; \mathcal{R} \; lstate(\beta_2)$ and hence $\beta_1 \; \mathcal{R}' \; \beta_2$.

Condition 2 of lifting is also satisfied by w'. In fact, given $\beta_2 = \alpha_2 a s_2$

$$\begin{aligned} \nu_2(\beta_2) &= \text{by definition of } \nu_2 \\ & \quad \mu_2(s_2) \\ &= \text{by } \mu_1 \; \mathcal{L}(\mathcal{R}) \; \mu_2 \\ & \quad \sum_{s_1 \in S_1} w(s_1, s_2) \end{aligned}$$

$$= \text{by definition of } w'$$
$$\sum_{\beta_1 \in Execs^*(\mathcal{A}_1)} w(\beta_1, \beta_2)$$

Condition 3 of lifting is also satisfied by w'. In fact, given $\beta_1 = \alpha_1 a s_1$

$$\nu_1(\beta_1) = \text{by definition of } \nu_1$$
$$\mu_1(s_1)$$
$$= \text{by } \mu_1 \; \mathcal{L}(\mathcal{R}) \; \mu_2$$
$$\sum_{s_2 \in S_2} w(s_1, s_2)$$
$$= \text{by definition of } w'$$
$$\sum_{\beta_2 \in Execs^*(\mathcal{A}_2)} w(\beta_1, \beta_2)$$

($\Rightarrow$) Conversely, given an execution simulation $\mathcal{R}$ from $Execs^*(\mathcal{A}_1)$ to $Execs^*(\mathcal{A}_2)$, define a relation $\mathcal{R}'$ from S_1 to S_2 such that $s_1 \; \mathcal{R}' \; s_2$ if and only if there exist α_1 and α_2 such that $\alpha_1 \; \mathcal{R} \; \alpha_2$, $lstate(\alpha_1) = s_1$, and $lstate(\alpha_2) = s_2$.

Start state condition is trivially true, since by hypothesis $\bar{s}_1 \; \mathcal{R} \; \bar{s}_2$ and thus $\bar{s}_1 \; \mathcal{R}' \; \bar{s}_2$.

For the step condition, take s_1, s_2, μ_1 such that $s_1 \; \mathcal{R}' \; s_2$ and $s_1 \longrightarrow \mu_1$. We must find μ_2 such that $s_2 \longrightarrow \mu_2$ and $\mu_1 \; \mathcal{L}(\mathcal{R}') \; \mu_2$. Since $s_1 \; \mathcal{R}' \; s_2$, by definition of $\mathcal{R}'$ there exist α_1 and α_2 such that $\alpha_1 \; \mathcal{R} \; \alpha_2$, $lstate(\alpha_1) = s_1$ and $lstate(\alpha_2) = s_2$. Since $s_1 \longrightarrow \mu_1$, then there exists ν_1 such that $\alpha_1 \longrightarrow \nu_1$ and for each finite execution $\alpha_1 a s$, $\nu_1(\alpha_1 a s) = \mu_1(s)$. Since $\alpha_1 \; \mathcal{R} \; \alpha_2$, then there exists ν_2 such that $\alpha_2 \longrightarrow \nu_2$ that implies that there exists μ_2 such that $s_2 \longrightarrow \mu_2$ and for each finite execution $\alpha_2 a s$, $\nu_2(\alpha_2 a s) = \mu_2(s)$.

To complete the proof, we must verify that $\mu_1 \; \mathcal{L}(\mathcal{R}') \; \mu_2$. To do this, let w be the weighting function that justifies $\nu_1 \; \mathcal{L}(\mathcal{R}) \; \nu_2$ and define

$$w'(s_1, s_2) = \begin{cases} w(\beta_1, \beta_2) & \text{if } \beta_1 = \alpha_1 a s_1 \text{ and } \beta_2 = \alpha_2 a s_2 \\ 0 & \text{otherwise} \end{cases}$$

for each pair of executions (s_1, s_2).

Condition 1 of lifting is satisfied by w'. In fact if $w'(s_1, s_2) > 0$, then $w(\alpha_1 a s_1, \alpha_2 a s_2) > 0$, thus $\alpha_1 a s_1 \; \mathcal{R} \; \alpha_2 a s_2$ and hence $s_1 \; \mathcal{R}' \; s_2$.

Condition 2 of lifting is also satisfied by w'. In fact, given a state s_2'

$$\begin{aligned}
\mu_2(s_2') &= \text{by definition of } \nu_2 \\
&\quad \nu_2(\alpha_2 a s_2') \\
&= \text{by } \nu_1 \; \mathcal{L}(\mathcal{R}) \; \nu_2 \\
&\quad \sum_{\alpha_1 a s_1' \in Execs^*(\mathcal{A}_1)} w(\alpha_1 a s_1', \alpha_2 a s_2') \\
&= \text{by definition of } w' \\
&\quad \sum_{\beta_1 \in Execs^*(\mathcal{A}_1)} w(\beta_1, \beta_2)
\end{aligned}$$

Condition 3 of lifting is also satisfied by w'. In fact, given s_1

$$\begin{aligned}
\mu_1(s_1') &= \text{by definition of } \nu_1 \\
&\quad \nu_1(\alpha_1 a s_1') \\
&= \text{by } \nu_1 \; \mathcal{L}(\mathcal{R}) \; \nu_2 \\
&\quad \sum_{\alpha_2 a s_2' \in Execs^*(\mathcal{A}_2)} w(\alpha_1 a s_1', \alpha_2 a s_2') \\
&= \text{by definition of } w' \\
&\quad \sum_{\beta_2 \in Execs^*(\mathcal{A}_2)} w(\beta_1, \beta_2)
\end{aligned}$$

□

We now generalize the notion of lifting so that two measures are not related exactly, but up to some error ε. Our definition states that two measures are related up to ε if some $(1 - \varepsilon)$ fractions of the two measures are related exactly. So, for example, suppose we have two coins: a fair coin and an unfair coin that chooses head with probability 1/3 and tail with probability 2/3. The two probability measures described by the two coins are not related by the identity relation, since the probability of head (as well as of tail) do not match. But if we admit an error of 1/3, for example, then they are related by the identity relation.

Definition 3.4. *Let $\mathcal{R}$ be a relation from X to Y and let $\varepsilon \geqslant 0$. The* ε-lifting *of $\mathcal{R}$, denoted by $\mathcal{L}(\mathcal{R}, \varepsilon)$ is a relation from $Disc(X)$ to $Disc(Y)$ defined as follows: for each pair ρ_x and ρ_y of probability measures on X and Y, respectively,*

- *if $\varepsilon \geqslant 1$, then $\rho_x \; \mathcal{L}(\mathcal{R}, \varepsilon) \; \rho_y$;*

- *if* $\varepsilon \in [0,1)$, *then* $\rho_x \ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_y$ *if there exist* $\rho'_x, \rho''_x \in Disc(X)$ *and* $\rho'_y, \rho''_y \in Disc(Y)$ *such that*
  - $\rho_x = (1-\varepsilon)\rho'_x + \varepsilon\rho''_x$,
  - $\rho_y = (1-\varepsilon)\rho'_y + \varepsilon\rho''_y$,
  - $\rho'_x \ \mathcal{L}(\mathcal{R})\ \rho'_y$.

It is interesting to observe that ε-lifting has many properties:

Property 3.5. *The ε-lifting satisfies the following properties:*

1. *For each relation $\mathcal{R}$ from X to Y, $\mathcal{L}(\mathcal{R},0)$=$\mathcal{L}(\mathcal{R})$.*
2. *For each relation $\mathcal{R}$ from X to Y, and each $\varepsilon,\varepsilon' \geqslant 0$, if $\varepsilon \leqslant \varepsilon'$, then $\mathcal{L}(\mathcal{R},\varepsilon) \subseteq \mathcal{L}(\mathcal{R},\varepsilon')$.*
3. *For each $\rho,\rho_1,\rho_2 \in Disc(X)$ and each $\varepsilon_1,\varepsilon_2 \geqslant 0$, if $\rho = (1-\varepsilon_1)\rho_1 + \varepsilon_1\rho_2$ and $\varepsilon_2 \geqslant \varepsilon_1$, then there exists $\rho_3 \in Disc(X)$ such that $\rho = (1-\varepsilon_2)\rho_1 + \varepsilon_2\rho_3$ and $\rho_3 = \left(1 - \frac{\varepsilon_1}{\varepsilon_2}\right)\rho_1 + \frac{\varepsilon_1}{\varepsilon_2}\rho_2$.*
4. *For each relation $\mathcal{R}$ on X, and each $\varepsilon \geqslant 0$, if $\mathcal{R}$ is reflexive then $\mathcal{L}(\mathcal{R},\varepsilon)$ is reflexive.*
5. *For each relation $\mathcal{R}$ from X to Y, and each $\varepsilon \geqslant 0$, if $\mathcal{R}$ is symmetric then $\mathcal{L}(\mathcal{R},\varepsilon)$ is symmetric.*
6. *For each relation $\mathcal{R}$ from X to Y, each $\varepsilon \in [0,1]$, and each measure $\rho_x,\rho'_x,\rho''_x,\rho_y$, if $\rho_x = (1-\varepsilon)\rho'_x + \varepsilon\rho''_x$ and $\rho_x \ \mathcal{L}(\mathcal{R})\ \rho_y$, then there exist measures ρ'_y,ρ''_y such that $\rho_y = (1-\varepsilon)\rho'_y + \varepsilon\rho''_y$, $\rho'_x \ \mathcal{L}(\mathcal{R})\ \rho'_y$ and $\rho''_x \ \mathcal{L}(\mathcal{R})\ \rho''_y$.*
7. *For each relation $\mathcal{R}$ from X to Y, each $\varepsilon \in [0,1]$, and each measure $\rho_x,\rho_y,\rho'_y,\rho''_y$, if $\rho_y = (1-\varepsilon)\rho'_y + \varepsilon\rho''_y$ and $\rho_x \ \mathcal{L}(\mathcal{R})\ \rho_y$, then there exist measures ρ'_x,ρ''_x such that $\rho_x = (1-\varepsilon)\rho'_x + \varepsilon\rho''_x$, $\rho'_x \ \mathcal{L}(\mathcal{R})\ \rho'_y$ and $\rho''_x \ \mathcal{L}(\mathcal{R})\ \rho''_y$.*
8. *For each relation $\mathcal{R}$ from X to Y, each relation $\mathcal{S}$ from Y to Z, each $\varepsilon_{xy},\varepsilon_{yz} \geqslant 0$, and for each measure ρ_x,ρ_y,ρ_z such that if $\rho_x \ \mathcal{L}(\mathcal{R},\varepsilon_{xy})\ \rho_y$ and $\rho_y \ \mathcal{L}(\mathcal{S},\varepsilon_{yz})\ \rho_z$, then $\rho_x \ \mathcal{L}(\mathcal{R}\circ\mathcal{S},\varepsilon_{xy}+\varepsilon_{yz})\ \rho_z$.*
9. *For each relation $\mathcal{R}$ from X to Y, each relation $\mathcal{S}$ from Y to Z, each $\varepsilon_{12},\varepsilon_{23} \geqslant 0$, and for each measure ρ_1,ρ_2,ρ_3 such that there exist ρ'_1, ρ''_1, ρ^1_2, ρ^2_2, ρ^3_2, ρ^4_2, ρ'_3, ρ''_3 such that*
   - $\rho_1 = (1-\varepsilon_{12})\rho'_1 + \varepsilon_{12}\rho''_1$,
   - $\rho_2 = (1-\varepsilon_{12})\rho^1_2 + \varepsilon_{12}\rho^2_2$,
   - $\rho'_1 \ \mathcal{L}(\mathcal{R})\ \rho^1_2$,

   and
   - $\rho_2 = (1-\varepsilon_{23})\rho^3_2 + \varepsilon_{23}\rho^4_2$,

- $\rho_3 = (1-\varepsilon_{23})\rho'_3 + \varepsilon_{23}\rho''_3$,
- $\rho_2^3\ \mathcal{L}(\mathcal{S})\ \rho'_3$,

if $\rho_2^1 = \rho_2^3$, *then* $\rho_1\ \mathcal{L}(\mathcal{R}\circ\mathcal{S}, \max\{\varepsilon_{12}, \varepsilon_{23}\})\ \rho_3$.

10. *For each relation* $\mathcal{R}$ *from* X *to* Y, *each* $\varepsilon \geqslant 0$, *and each measure* $\rho_x \in Disc(X)$ *and* $\rho_y \in Disc(Y)$, *if* $\rho_x\ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_y$, *then for each measure* $\rho_z \in Disc(Z)$, $\rho_x \times \rho_z\ \mathcal{L}(\mathcal{R}\times id, \varepsilon)\ \rho_y \times \rho_z$ *where* id *is the identity relation on* Z.

Proof. Let $\mathcal{R}$ be a relation from X to Y.

1. ($\Rightarrow$) Let $\rho_x \in Disc(X)$ and $\rho_y \in Disc(Y)$ be two distributions such that $\rho_x\ \mathcal{L}(\mathcal{R},0)\ \rho_y$. By definition of 0-lifting, there exist $\rho'_x, \rho''_x \in Disc(X)$ and $\rho'_y, \rho''_y \in Disc(Y)$ such that
   - $\rho_x = (1-0)\rho'_x + 0\rho''_x$,
   - $\rho_y = (1-0)\rho'_y + 0\rho''_y$, and
   - $\rho'_x\ \mathcal{L}(\mathcal{R})\ \rho'_y$.

   Since $\rho_x = \rho'_x$ and $\rho_y = \rho'_y$, then $\rho_x\ \mathcal{L}(\mathcal{R})\ \rho_y$.

   ($\Leftarrow$) Let $\rho_x \in Disc(X)$ and $\rho_y \in Disc(Y)$ be two distributions such that $\rho_x\ \mathcal{L}(\mathcal{R})\ \rho_y$. Define $\rho'_x = \rho''_x = \rho_x$ and $\rho'_y = \rho''_y = \rho_y$. This implies that $\rho_x = (1-0)\rho'_x + 0\rho''_x$, and $\rho_y = (1-0)\rho'_y + 0\rho''_y$. Since $\rho'_x = \rho_x$ and $\rho'_y = \rho_y$, then $\rho'_x\ \mathcal{L}(\mathcal{R})\ \rho'_y$ and thus $\rho_x\ \mathcal{L}(\mathcal{R},0)\ \rho_y$.
2. Let $\varepsilon > 0$ and let $\rho_x \in Disc(X)$ and $\rho_y \in Disc(Y)$ be two distributions such that $\rho_x\ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_y$. Let $\varepsilon' \geqslant \varepsilon$. If $\varepsilon' \geqslant 1$, then by definition of ε'-lifting, $\rho_x\ \mathcal{L}(\mathcal{R},\varepsilon')\ \rho_y$. Suppose that $\varepsilon' < 1$ and denote by γ the value $\varepsilon' - \varepsilon \geqslant 0$. By definition of ε-lifting, $\rho_x\ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_y$ implies that there exist ρ'_x, ρ''_x, ρ'_y and ρ''_y such that
   - $\rho_x = (1-\varepsilon)\rho'_x + \varepsilon\rho''_x$,
   - $\rho_y = (1-\varepsilon)\rho'_y + \varepsilon\rho''_y$, and
   - $\rho'_x\ \mathcal{L}(\mathcal{R})\ \rho'_y$.

   Define $\theta'_x = \rho'_x$ and $\theta''_x = (\gamma\rho'_x + \varepsilon\rho''_x)/(\gamma+\varepsilon)$. This implies that

$$\begin{aligned}
\rho_x &= (1-\varepsilon)\rho'_x + \varepsilon\rho''_x \\
&= (1-\varepsilon-\gamma)\rho'_x + \gamma\rho'_x + \varepsilon\rho''_x \\
&= (1-\varepsilon-\gamma)\rho'_x + (\gamma+\varepsilon)(\gamma\rho'_x + \varepsilon\rho''_x)/(\gamma+\varepsilon) \\
&= (1-(\varepsilon+\gamma))\theta'_x + (\gamma+\varepsilon)\theta''_x \\
&= (1-\varepsilon')\theta'_x + \varepsilon'\theta''_x
\end{aligned}$$

   In a similar way, define $\theta'_y = \rho'_y$ and $\theta''_y = (\gamma\rho'_y + \varepsilon\rho''_y)/(\gamma+\varepsilon)$. This implies that

$$\begin{aligned}\rho_y &= (1-\varepsilon)\rho'_y + \varepsilon\rho''_y \\ &= (1-\varepsilon-\gamma)\rho'_y + \gamma\rho'_y + \varepsilon\rho''_y \\ &= (1-\varepsilon-\gamma)\rho'_y + (\gamma+\varepsilon)(\gamma\rho'_y + \varepsilon\rho''_y)/(\gamma+\varepsilon) \\ &= (1-(\varepsilon+\gamma))\theta'_y + (\gamma+\varepsilon)\theta''_y \\ &= (1-\varepsilon')\theta'_y + \varepsilon'\theta''_y\end{aligned}$$

This implies that there exist $\theta'_x, \theta''_x \in \mathit{Disc}(X)$, $\theta'_y, \theta''_y \in \mathit{Disc}(Y)$ such that

- $\rho_x = (1-\varepsilon')\theta'_x + \varepsilon'\theta''_x$,
- $\rho_y = (1-\varepsilon')\theta'_y + \varepsilon'\theta''_y$,
- $\theta'_x \; \mathcal{L}(\mathcal{R}) \; \theta'_y$ since $\theta'_x = \rho'_x$, $\theta'_y = \rho'_y$ and $\rho'_x \; \mathcal{L}(\mathcal{R}) \; \rho'_y$

and thus $\rho_x \; \mathcal{L}(\mathcal{R}, \varepsilon') \; \rho_y$.

3. If $\varepsilon_2 = \varepsilon_1$, then $\rho_3 = \rho_2$ satisfies the required properties.
   Suppose that $\varepsilon_2 > \varepsilon_1$, then let $\varepsilon_3 = \varepsilon_2 - \varepsilon_1$ and thus $\varepsilon_1 = \varepsilon_2 - \varepsilon_3$. Replacing ε_1 in the decomposition of ρ, we have that

$$\begin{aligned}\rho &= (1-(\varepsilon_2-\varepsilon_3))\rho_1 + (\varepsilon_2-\varepsilon_3)\rho_2 \\ &= (1-\varepsilon_2)\rho_1 + \varepsilon_3))\rho_1 + \varepsilon_2\rho_2 - \varepsilon_3\rho_2 \\ &= (1-\varepsilon_2)\rho_1 + \varepsilon_2\left(\rho_2 - \frac{\varepsilon_3}{\varepsilon_2}\rho_2 + \frac{\varepsilon_3}{\varepsilon_2}\rho_1\right) \\ &= (1-\varepsilon_2)\rho_1 + \varepsilon_2\rho_3\end{aligned}$$

   where $\varepsilon_2 > 0$ since $\varepsilon_2 > \varepsilon_1 \geqslant 0$ and ρ_3 is a probability measure defined as

$$\begin{aligned}\rho_3 &= \left(1 - \frac{\varepsilon_3}{\varepsilon_2}\right)\rho_2 + \frac{\varepsilon_3}{\varepsilon_2}\rho_1 \\ &= \left(1 - \frac{\varepsilon_2-\varepsilon_1}{\varepsilon_2}\right)\rho_2 + \frac{\varepsilon_2-\varepsilon_1}{\varepsilon_2}\rho_1 \\ &= \left(1 - 1 + \frac{\varepsilon_1}{\varepsilon_2}\right)\rho_2 + \left(1 - \frac{\varepsilon_1}{\varepsilon_2}\right)\rho_1 \\ &= \left(1 - \frac{\varepsilon_1}{\varepsilon_2}\right)\rho_1 + \frac{\varepsilon_1}{\varepsilon_2}\rho_2\end{aligned}$$

   ρ_3 is a probability measure since ρ_1 and ρ_2 are probability measures and $0 \leqslant \dfrac{\varepsilon_1}{\varepsilon_2} < 1$.

4. Let $\varepsilon \geqslant 0$ and consider a measure $\rho \in \mathit{Disc}(X)$. If $\varepsilon \geqslant 1$, then by definition of ε-lifting, it follows that $\rho \; \mathcal{L}(\mathcal{R}, \varepsilon) \; \rho$. If $0 \leqslant \varepsilon < 1$, then we have that $\rho = (1-\varepsilon)\rho + \varepsilon\rho$. Since by Property 2.1(4), the reflexivity of

$\mathcal{R}$ implies the reflexivity of $\mathcal{L}(\mathcal{R})$, then we have that $\rho\ \mathcal{L}(\mathcal{R})\ \rho$ and thus $\rho\ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho$.

5. Let $\varepsilon \geqslant 0$ and consider two measures $\rho_x \in \mathit{Disc}(X)$ and $\rho_y \in \mathit{Disc}(Y)$ such that $\rho_x\ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_y$. If $\varepsilon \geqslant 1$, then by definition of ε-lifting, it follows that $\rho_y\ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_x$. If $0 \leqslant \varepsilon < 1$, then we have that there exist $\rho'_x, \rho''_x, \rho'_y, \rho''_y$ such that
   - $\rho_x = (1-\varepsilon)\rho'_x + \varepsilon\rho''_x$,
   - $\rho_y = (1-\varepsilon)\rho'_y + \varepsilon\rho''_y$, and
   - $\rho'_x\ \mathcal{L}(\mathcal{R})\ \rho'_y$.

   Since $\mathcal{R}$ is symmetric, by Property 2.1(5) we have that $\rho'_y\ \mathcal{L}(\mathcal{R})\ \rho'_x$ and thus $\rho_y\ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_x$.
6. Fix $\varepsilon \in [0,1]$ and let $\rho_x, \rho'_x, \rho''_x \in \mathit{Disc}(X)$, $\rho_y \in \mathit{Disc}(Y)$ be four measures such that $\rho_x = (1-\varepsilon)\rho'_x + \varepsilon\rho''_x$ and $\rho_x\ \mathcal{L}(\mathcal{R})\ \rho_y$. This implies that there exists $w\colon X \times Y \to [0,1]$ such that
   - for each $u \in X$, $v \in Y$, $w(u,v) > 0 \implies (u,v) \in \mathcal{R}$;
   - for each $u \in X$, $\sum_{v\in Y} w(u,v) = \rho_x(u)$; and
   - for each $v \in Y$, $\sum_{u\in X} w(u,v) = \rho_y(v)$.

   Let $w'\colon X \times Y \to [0,1]$ be the function defined as: for each $u \in X$, $v \in Y$,

$$w'(u,v) = \begin{cases} \dfrac{\rho'_x(u)w(u,v)}{\rho_x(u)} & \text{if } \rho_x(u) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

   Let ρ'_y be the function defined as: for each $v \in Y$,

$$\rho'_y(v) = \sum_{u\in X} w'(u,v) = \sum_{u\in X, \rho_x(u)\neq 0} \frac{\rho'_x(u)w(u,v)}{\rho_x(u)}$$

   It is easy to see that $\rho'_y \in \mathit{Disc}(Y)$. In fact, for each $v \in Y$, we have $\rho'_y(v) \geqslant 0$ since $\rho'_y(v)$ is the sum of non-negative values; furthermore,

$$\begin{aligned} \rho'_y(Y) &= \sum_{v\in Y} \rho'_y(v) \\ &= \sum_{v\in Y} \sum_{u\in X, \rho_x(u)\neq 0} \frac{\rho'_x(u)w(u,v)}{\rho_x(u)} \\ &= \sum_{u\in X, \rho_x(u)\neq 0} \frac{\rho'_x(u)}{\rho_x(u)} \sum_{v\in Y} w(u,v) \\ &= \sum_{u\in X, \rho_x(u)\neq 0} \frac{\rho'_x(u)}{\rho_x(u)} \rho_x(u) \end{aligned}$$

$$= \sum_{u \in X} \rho'_x(u)$$
$$= 1$$

w' is a weighting function from ρ'_x to ρ'_y. In fact,

- for each $u \in X$, $v \in Y$, $w'(u,v) > 0 \implies \dfrac{\rho'_x(u)w(u,v)}{\rho_x(u)} > 0 \implies w(u,v) > 0 \implies (u,v) \in \mathcal{R}$;
- for each $u \in X$, if $\rho_x(u) \neq 0$, $\sum_{v \in Y} w'(u,v) = \sum_{v \in Y} \dfrac{\rho'_x(u)w(u,v)}{\rho_x(u)} = \dfrac{\rho'_x(u)}{\rho_x(u)} \sum_{v \in Y} w(u,v) = \dfrac{\rho'_x(u)}{\rho_x(u)}\rho_x(u) = \rho'_x(u)$. If $\rho_x(u) = 0$, then also $\rho'_x(u) = 0$ and by definition of w' we have that $\sum_{v \in Y} w'(u,v) = \sum_{v \in Y} 0 = 0$. Thus, for each $u \in X$, $\sum_{v \in Y} w'(u,v) = \rho'_x(u)$;
- for each $v \in Y$, $\sum_{u \in X} w'(u,v) = \rho'_y(v)$ by definition of $\rho'_y(v)$.

Let $w''\colon X \times Y \to [0,1]$ be the function defined as: for each $u \in X$, $v \in Y$,

$$w''(u,v) = \begin{cases} \dfrac{\rho''_x(u)w(u,v)}{\rho_x(u)} & \text{if } \rho_x(u) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Let ρ''_y be the function defined as: for each $v \in Y$,

$$\rho''_y(v) = \sum_{u \in X} w''(u,v) = \sum_{u \in X, \rho_x(u) \neq 0} \frac{\rho''_x(u)w(u,v)}{\rho_x(u)}$$

It is easy to see that $\rho''_y \in \mathit{Disc}(Y)$. In fact, for each $v \in Y$, we have $\rho''_y(v) \geqslant 0$ since $\rho''_y(v)$ is the sum of non-negative values; furthermore,

$$\begin{aligned} \rho''_y(Y) &= \sum_{v \in Y} \rho''_y(v) \\ &= \sum_{v \in Y} \sum_{u \in X, \rho_x(u) \neq 0} \frac{\rho''_x(u)w(u,v)}{\rho_x(u)} \\ &= \sum_{u \in X, \rho_x(u) \neq 0} \frac{\rho''_x(u)}{\rho_x(u)} \sum_{v \in Y} w(u,v) \\ &= \sum_{u \in X, \rho_x(u) \neq 0} \frac{\rho''_x(u)}{\rho_x(u)} \rho_x(u) \\ &= \sum_{u \in X} \rho''_x(u) \end{aligned}$$

$$= 1$$

w'' is a weighting function from ρ''_x to ρ''_y. In fact,

- for each $u \in X$, $v \in Y$, $w''(u,v) > 0 \implies \dfrac{\rho''_x(u)w(u,v)}{\rho_x(u)} > 0 \implies w(u,v) > 0 \implies (u,v) \in \mathcal{R}$;
- for each $u \in X$, if $\rho_x(u) \neq 0$, $\sum_{v\in Y} w''(u,v) = \sum_{v\in Y} \dfrac{\rho''_x(u)w(u,v)}{\rho_x(u)} = \dfrac{\rho''_x(u)}{\rho_x(u)} \sum_{v\in Y} w(u,v) = \dfrac{\rho''_x(u)}{\rho_x(u)}\rho_x(u) = \rho''_x(u)$. If $\rho_x(u) = 0$, then also $\rho''_x(u) = 0$ and by definition of w'' we have that $\sum_{v\in Y} w''(u,v) = \sum_{v\in Y} 0 = 0$. Thus, for each $u \in X$, $\sum_{v\in Y} w''(u,v) = \rho''_x(u)$;
- for each $v \in Y$, $\sum_{u\in X} w''(u,v) = \rho''_y(v)$ by definition of $\rho''_y(v)$.

Finally, we must verify that $(1-\varepsilon)\rho'_y + \varepsilon\rho''_y = \rho_y$. In fact, for each $v \in Y$,

$$\begin{aligned}
(1-\varepsilon)\rho'_y(v) + \varepsilon\rho''_y(v) &= (1-\varepsilon)\sum_{u\in X} w'(u,v) + \varepsilon \sum_{u\in X} w''(u,v) \\
&= (1-\varepsilon) \sum_{u\in X, \rho_x(u)\neq 0} \frac{\rho'_x(u)w(u,v)}{\rho_x(u)} \\
&\quad + \varepsilon \sum_{u\in X, \rho_x(u)\neq 0} \frac{\rho''_x(u)w(u,v)}{\rho_x(u)} \\
&= \sum_{u\in X, \rho_x(u)\neq 0} w(u,v) \frac{(1-\varepsilon)\rho'_x(u) + \varepsilon\rho''_x(u)}{\rho_x(u)} \\
&= \sum_{u\in X, \rho_x(u)\neq 0} w(u,v) \frac{\rho_x(u)}{\rho_x(u)} \\
&= \sum_{u\in X} w(u,v) \\
&= \rho_y(v)
\end{aligned}$$

7. Fix $\varepsilon \in [0,1]$ and let $\rho_x \in \mathit{Disc}(X)$, $\rho_y, \rho'_y, \rho''_y \in \mathit{Disc}(Y)$ be four measures such that $\rho_y = (1-\varepsilon)\rho'_y + \varepsilon\rho''_y$ and $\rho_x \; \mathcal{L}(\mathcal{R}) \; \rho_y$. This implies that there exists $w\colon X \times Y \to [0,1]$ such that
   - for each $u \in X$, $v \in Y$, $w(u,v) > 0 \implies (u,v) \in \mathcal{R}$;
   - for each $u \in X$, $\sum_{v\in Y} w(u,v) = \rho_x(u)$; and
   - for each $v \in Y$, $\sum_{u\in X} w(u,v) = \rho_y(v)$.

   Let $w'\colon X \times Y \to [0,1]$ be the function defined as: for each $u \in X$, $v \in Y$,

$$w'(u,v) = \begin{cases} \dfrac{\rho'_y(v)w(u,v)}{\rho_y(v)} & \text{if } \rho_y(v) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Let ρ'_x be the function defined as: for each $u \in X$,

$$\rho'_x(u) = \sum_{v \in Y} w'(u,v) = \sum_{v \in Y, \rho_y(v) \neq 0} \frac{\rho'_y(v)w(u,v)}{\rho_y(v)}$$

It is easy to see that $\rho'_x \in \mathit{Disc}(X)$. In fact, for each $u \in X$, we have $\rho'_x(u) \geqslant 0$ since $\rho'_x(u)$ is the sum of non-negative values; furthermore,

$$\begin{aligned} \rho'_x(X) &= \sum_{u \in X} \rho'_x(u) \\ &= \sum_{u \in X} \sum_{v \in Y, \rho_y(v) \neq 0} \frac{\rho'_y(v)w(u,v)}{\rho_y(v)} \\ &= \sum_{v \in Y, \rho_y(v) \neq 0} \frac{\rho'_y(v)}{\rho_y(v)} \sum_{u \in X} w(u,v) \\ &= \sum_{v \in Y, \rho_y(v) \neq 0} \frac{\rho'_y(v)}{\rho_y(v)} \rho_y(v) \\ &= \sum_{v \in Y} \rho'_y(v) \\ &= 1 \end{aligned}$$

w' is a weighting function from ρ'_x to ρ'_y. In fact,

- for each $u \in X$, $v \in Y$, $w'(u,v) > 0 \implies \dfrac{\rho'_y(v)w(u,v)}{\rho_y(v)} > 0 \implies w(u,v) > 0 \implies (u,v) \in \mathcal{R}$;
- for each $u \in X$, $\sum_{v \in Y} w'(u,v) = \rho'_x(u)$ by definition of $\rho'_x(u)$;
- for each $v \in Y$, if $\rho_y(v) \neq 0$, $\sum_{u \in X} w'(u,v) = \sum_{u \in X} \dfrac{\rho'_y(v)w(u,v)}{\rho_y(v)} = \dfrac{\rho'_y(v)}{\rho_y(v)} \sum_{u \in X} w(u,v) = \dfrac{\rho'_y(v)}{\rho_y(v)} \rho_y(v) = \rho'_y(v)$. If $\rho_y(v) = 0$, then also $\rho'_y(v) = 0$ and by definition of w' we have that $\sum_{u \in X} w'(u,v) = \sum_{u \in X} 0 = 0$. Thus, for each $v \in Y$, $\sum_{u \in X} w'(u,v) = \rho'_y(v)$.

Let $w'' \colon X \times Y \to [0,1]$ be the function defined as: for each $u \in X$, $v \in Y$,

$$w''(u,v) = \begin{cases} \dfrac{\rho''_y(v)w(u,v)}{\rho_y(v)} & \text{if } \rho_y(v) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Let ρ''_x be the function defined as: for each $u \in X$,

$$\rho''_x(u) = \sum_{v \in Y} w''(u,v) = \sum_{v \in Y, \rho_y(v) \neq 0} \frac{\rho''_y(v) w(u,v)}{\rho_y(v)}$$

It is easy to see that $\rho''_x \in \mathit{Disc}(X)$. In fact, for each $u \in X$, we have $\rho''_x(u) \geqslant 0$ since $\rho''_x(u)$ is the sum of non-negative values; furthermore,

$$\begin{aligned}
\rho''_x(X) &= \sum_{u \in X} \rho''_x(u) \\
&= \sum_{u \in X} \sum_{v \in Y, \rho_y(v) \neq 0} \frac{\rho''_y(v) w(u,v)}{\rho_y(v)} \\
&= \sum_{v \in Y, \rho_y(v) \neq 0} \frac{\rho''_y(v)}{\rho_y(v)} \sum_{u \in X} w(u,v) \\
&= \sum_{v \in Y, \rho_y(v) \neq 0} \frac{\rho''_y(v)}{\rho_y(v)} \rho_y(v) \\
&= \sum_{v \in Y} \rho''_y(v) \\
&= 1
\end{aligned}$$

w'' is a weighting function from ρ''_x to ρ''_y. In fact,

- for each $u \in X$, $v \in Y$, $w''(u,v) > 0 \implies \dfrac{\rho''_y(v) w(u,v)}{\rho_y(v)} > 0 \implies w(u,v) > 0 \implies (u,v) \in \mathcal{R}$;
- for each $u \in X$, $\sum_{v \in Y} w''(u,v) = \rho''_x(u)$ by definition of $\rho''_x(u)$;
- for each $v \in Y$, if $\rho_y(v) \neq 0$, $\sum_{u \in X} w''(u,v) = \sum_{u \in X} \dfrac{\rho''_y(v) w(u,v)}{\rho_y(v)} = \dfrac{\rho''_y(v)}{\rho_y(v)} \sum_{u \in X} w(u,v) = \dfrac{\rho''_y(v)}{\rho_y(v)} \rho_y(v) = \rho''_y(v)$. If $\rho_y(v) = 0$, then also $\rho''_y(v) = 0$ and by definition of w'' we have that $\sum_{u \in X} w''(u,v) = \sum_{u \in X} 0 = 0$. Thus, for each $v \in Y$, $\sum_{u \in X} w''(u,v) = \rho''_y(v)$.

Finally, we must verify that $(1-\varepsilon)\rho'_x + \varepsilon \rho''_x = \rho_x$. In fact, for each $u \in X$,

$$\begin{aligned}
(1-\varepsilon)\rho'_x(u) + \varepsilon \rho''_x(u) &= (1-\varepsilon) \sum_{v \in Y} w'(u,v) + \varepsilon \sum_{v \in Y} w''(u,v) \\
&= (1-\varepsilon) \sum_{v \in Y, \rho_y(v) \neq 0} \frac{\rho'_y(v) w(u,v)}{\rho_y(v)} \\
&\quad + \varepsilon \sum_{v \in Y, \rho_y(v) \neq 0} \frac{\rho''_y(v) w(u,v)}{\rho_y(v)}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{v \in Y, \rho_y(v) \neq 0} w(u,v) \frac{(1-\varepsilon)\rho'_y(v) + \varepsilon \rho''_y(v)}{\rho_y(v)} \\
&= \sum_{v \in Y, \rho_y(v) \neq 0} w(u,v) \frac{\rho_y(v)}{\rho_y(v)} \\
&= \sum_{v \in Y} w(u,v) \\
&= \rho_x(u)
\end{aligned}
$$

8. Let $\mathcal{R}$ be a relation from X to Y and $\mathcal{S}$ be a relation from Y to Z. If $\varepsilon_{xy} + \varepsilon_{yz} \geqslant 1$, then $\rho_x \; \mathcal{L}(\mathcal{R} \circ \mathcal{S}, \varepsilon_{xy} + \varepsilon_{yz}) \; \rho_z$ derives directly from the definition of ε-lifting.
   If $\varepsilon_{xy} = \varepsilon_{yz} = 0$, then we have that $\rho_x \; \mathcal{L}(\mathcal{R}, 0) \; \rho_y$ and $\rho_y \; \mathcal{L}(\mathcal{S}, 0) \; \rho_z$. This implies, by Property 1, that $\rho_x \; \mathcal{L}(\mathcal{R}) \; \rho_y$ and $\rho_y \; \mathcal{L}(\mathcal{S}) \; \rho_z$ and thus, by Property 2.1(6), $\rho_x \; \mathcal{L}(\mathcal{R} \circ \mathcal{S}) \; \rho_z$. By Property 1, we have that $\rho_x \; \mathcal{L}(\mathcal{R} \circ \mathcal{S}, 0) \; \rho_z$ and thus $\rho_x \; \mathcal{L}(\mathcal{R} \circ \mathcal{S}, \varepsilon_{xy} + \varepsilon_{yz}) \; \rho_z$.
   If $\varepsilon_{xy} = 0$ and $\varepsilon_{yz} \in (0,1)$, then we have that $\rho_x \; \mathcal{L}(\mathcal{R}, 0) \; \rho_y$ and $\rho_y \; \mathcal{L}(\mathcal{S}, \varepsilon_{yz}) \; \rho_z$ imply that $\rho_x \; \mathcal{L}(\mathcal{R}) \; \rho_y$ and that there exist $\rho'_y, \rho''_y \in Disc(Y)$, $\rho'_z, \rho''_z \in Disc(Z)$ such that
   - $\rho_y = (1 - \varepsilon_{yz})\rho'_y + \varepsilon_{yz}\rho''_y$,
   - $\rho_z = (1 - \varepsilon_{yz})\rho'_z + \varepsilon_{yz}\rho''_z$, and
   - $\rho'_y \; \mathcal{L}(\mathcal{S}) \; \rho'_z$.

   Since $\rho_y = (1 - \varepsilon_{yz})\rho'_y + \varepsilon_{yz}\rho''_y$ and $\rho_x \; \mathcal{L}(\mathcal{R}) \; \rho_y$, by Property 7 we have that there exist $\rho'_x, \rho''_x \in Disc(X)$ such that
   - $\rho_x = (1 - \varepsilon_{yz})\rho'_x + \varepsilon_{yz}\rho''_x$,
   - $\rho'_x \; \mathcal{L}(\mathcal{R}) \; \rho'_y$, and
   - $\rho''_x \; \mathcal{L}(\mathcal{R}) \; \rho''_y$.

   By Property 2.1(6), $\rho'_x \; \mathcal{L}(\mathcal{R}) \; \rho'_y$ and $\rho'_y \; \mathcal{L}(\mathcal{S}) \; \rho'_z$ implies that $\rho'_x \; \mathcal{L}(\mathcal{R} \circ \mathcal{S}) \; \rho'_z$. Summing up, we have that
   - $\rho_x = (1 - \varepsilon_{yz})\rho'_x + \varepsilon_{yz}\rho''_x$,
   - $\rho_z = (1 - \varepsilon_{yz})\rho'_z + \varepsilon_{yz}\rho''_z$, and
   - $\rho'_x \; \mathcal{L}(\mathcal{R} \circ \mathcal{S}) \; \rho'_z$

   and thus $\rho_x \; \mathcal{L}(\mathcal{R} \circ \mathcal{S}, \varepsilon_{yz}) \; \rho_z$ that is $\rho_x \; \mathcal{L}(\mathcal{R} \circ \mathcal{S}, \varepsilon_{xy} + \varepsilon_{yz}) \; \rho_z$ since $\varepsilon_{xy} = 0$.
   If $\varepsilon_{xy} \in (0,1)$ and $\varepsilon_{yz} = 0$, then we have that $\rho_x \; \mathcal{L}(\mathcal{R}, \varepsilon_{xy}) \; \rho_y$ and $\rho_y \; \mathcal{L}(\mathcal{S}, 0) \; \rho_z$ imply that $\rho_y \; \mathcal{L}(\mathcal{S}) \; \rho_z$ and that there exist $\rho'_x, \rho''_x \in Disc(X)$, $\rho'_y, \rho''_y \in Disc(Y)$ such that
   - $\rho_x = (1 - \varepsilon_{xy})\rho'_x + \varepsilon_{xy}\rho''_x$,

– $\rho_y = (1-\varepsilon_{yz})\rho'_y + \varepsilon_{yz}\rho''_y$, and
– $\rho'_x\ \mathcal{L}(\mathcal{R})\ \rho'_y$.

Since $\rho_y = (1-\varepsilon_{xy})\rho'_y + \varepsilon_{xy}\rho''_y$ and $\rho_y\ \mathcal{L}(\mathcal{S})\ \rho_z$, by Property 6 we have that there exist $\rho'_z, \rho''_z \in Disc(Z)$ such that

– $\rho_z = (1-\varepsilon_{xy})\rho'_z + \varepsilon_{xy}\rho''_z$,
– $\rho'_y\ \mathcal{L}(\mathcal{S})\ \rho'_z$, and
– $\rho''_z\ \mathcal{L}(\mathcal{S})\ \rho''_z$.

By Property 2.1(6), $\rho'_x\ \mathcal{L}(\mathcal{R})\ \rho'_y$ and $\rho'_y\ \mathcal{L}(\mathcal{S})\ \rho'_z$ implies that $\rho'_x\ \mathcal{L}(\mathcal{R}\circ\mathcal{S})\ \rho'_z$. Summing up, we have that

– $\rho_x = (1-\varepsilon_{xy})\rho'_x + \varepsilon_{xy}\rho''_x$,
– $\rho_z = (1-\varepsilon_{xy})\rho'_z + \varepsilon_{xy}\rho''_z$, and
– $\rho'_x\ \mathcal{L}(\mathcal{R}\circ\mathcal{S})\ \rho'_z$

and thus $\rho_x\ \mathcal{L}(\mathcal{R}\circ\mathcal{S}, \varepsilon_{xy})\ \rho_z$ that is $\rho_x\ \mathcal{L}(\mathcal{R}\circ\mathcal{S}, \varepsilon_{xy}+\varepsilon_{yz})\ \rho_z$ since $\varepsilon_{yz} = 0$.

Now, suppose that $\varepsilon_{xy}+\varepsilon_{yz} < 1$ (and thus $0 < \varepsilon_{xy} < 1$ and $0 < \varepsilon_{yz} < 1$). By hypothesis, we have that

– $\rho_x = (1-\varepsilon_{xy})\rho^1_x + \varepsilon_{xy}\rho^2_x$,
– $\rho_y = (1-\varepsilon_{xy})\rho^1_y + \varepsilon_{xy}\rho^2_y$, and
– $\rho^1_x\ \mathcal{L}(\mathcal{R})\ \rho^1_y$,

and

– $\rho_y = (1-\varepsilon_{yz})\rho^3_y + \varepsilon_{yz}\rho^4_y$,
– $\rho_z = (1-\varepsilon_{yz})\rho^1_z + \varepsilon_{yz}\rho^2_z$, and
– $\rho^3_y\ \mathcal{L}(\mathcal{S})\ \rho^1_z$.

Now from the decompositions of ρ_y, we want to obtain a new decomposition of ρ_y with some specific property. So, let $\varepsilon = \max\varepsilon_{xy}, \varepsilon_{yz}$ and let ρ^5_y, ρ^6_y be two functions defined as

$$\rho^5_y = \frac{\min\{(1-\varepsilon_{xy})\rho^1_y, (1-\varepsilon_{yz})\rho^3_y\}}{1-\varepsilon}$$

and

$$\rho^6_y = \frac{\max\{\varepsilon_{xy}\rho^2_y, \varepsilon_{yz}\rho^4_y\}}{\varepsilon}$$

ρ^5_y and ρ^6_y are two probability measures. In fact, for each $v \in Y$, $\rho^5_y(v) \geqslant 0$ since we have that $(1-\varepsilon_{xy})\rho^1_y(u) \geqslant 0$, $(1-\varepsilon_{yz})\rho^3_y(u) \geqslant 0$, and $1-\varepsilon > 0$ (since $\varepsilon_{xy} < 1$, $\varepsilon_{yz} < 1$ and thus $\varepsilon = \max\{\varepsilon_{xy}, \varepsilon_{yz}\} < 1$).

$$\rho^5_y(\emptyset) = \frac{\min\{(1-\varepsilon_{xy})\rho^1_y(\emptyset), (1-\varepsilon_{yz})\rho^3_y(\emptyset)\}}{1-\varepsilon}$$

$$= \frac{\min\{(1-\varepsilon_{xy})0, (1-\varepsilon_{yz})0\}}{1-\varepsilon}$$
$$= 0$$

and

$$\rho_y^5(Y) = \frac{\min\{(1-\varepsilon_{xy})\rho_y^1(Y), (1-\varepsilon_{yz})\rho_y^3(Y)\}}{1-\varepsilon}$$
$$= \frac{\min\{(1-\varepsilon_{xy})1, (1-\varepsilon_{yz})1\}}{1-\varepsilon}$$
$$= \frac{1-\max\{\varepsilon_{xy}, \varepsilon_{yz}\}}{1-\varepsilon}$$
$$= \frac{1-\varepsilon}{1-\varepsilon} = 1$$

Analogously, for each $v \in Y$, $\rho_y^6(v) \geqslant 0$ since we have that $\varepsilon_{xy}\rho_y^2(u) \geqslant 0$, $\varepsilon_{yz}\rho_y^4(u) \geqslant 0$, and $\varepsilon > 0$ (since $\varepsilon_{xy} + \varepsilon_{yz} > 0$, and hence at least one between ε_{zy} and ε_{yz} is greater than 0 and thus $\varepsilon = \max\{\varepsilon_{xy}, \varepsilon_{yz}\} > 0$).

$$\rho_y^6(\emptyset) = \frac{\max\{\varepsilon_{xy}\rho_y^2(\emptyset), \varepsilon_{yz}\rho_y^4(\emptyset)\}}{\varepsilon}$$
$$= \frac{\max\{\varepsilon_{xy}0, \varepsilon_{yz}0\}}{\varepsilon}$$
$$= 0$$

and

$$\rho_y^6(Y) = \frac{\max\{\varepsilon_{xy}\rho_y^2(Y), \varepsilon_{yz}\rho_y^4(Y)\}}{\varepsilon}$$
$$= \frac{\max\{\varepsilon_{xy}1, \varepsilon_{yz}1\}}{\varepsilon}$$
$$= \frac{\varepsilon}{\varepsilon} = 1$$

By definition of ρ_y^5 and ρ_y^6 it is easy to verify that $\rho_y = (1-\varepsilon)\rho_y^5 + \varepsilon\rho_y^6$ since for each $v \in Y$ we have that $\min\{(1-\varepsilon_{xy})\rho_y^1(v), (1-\varepsilon_{yz})\rho_y^3(v)\} = (1-\varepsilon_{xy})\rho_y^1(v)$ if and only if $\max\{\varepsilon_{xy}\rho_y^2(v), \varepsilon_{yz}\rho_y^4(v)\} = \varepsilon_{xy}\rho_y^2(v)$. In fact,

$$(1-\varepsilon_{xy})\rho_y^1(v) + \varepsilon_{xy}\rho_y^2(v) = (1-\varepsilon_{yz})\rho_y^3(v) + \varepsilon_{yz}\rho_y^4(v)$$
$$(1-\varepsilon_{xy})\rho_y^1(v) - (1-\varepsilon_{yz})\rho_y^3(v) = \varepsilon_{yz}\rho_y^4(v) - \varepsilon_{xy}\rho_y^2(v)$$
$$(1-\varepsilon_{xy})\rho_y^1(v) - (1-\varepsilon_{yz})\rho_y^3(v) \leqslant 0 \iff \varepsilon_{yz}\rho_y^4(v) - \varepsilon_{xy}\rho_y^2(v) \leqslant 0$$
$$(1-\varepsilon_{xy})\rho_y^1(v) \leqslant (1-\varepsilon_{yz})\rho_y^3(v) \iff \varepsilon_{yz}\rho_y^4(v) \leqslant \varepsilon_{xy}\rho_y^2(v)$$

$$(1-\varepsilon_{xy})\rho_y^1(v) \leqslant (1-\varepsilon_{yz})\rho_y^3(v) \iff \varepsilon_{xy}\rho_y^2(v) \geqslant \varepsilon_{yz}\rho_y^4(v)$$

Let $\varepsilon_1 = \varepsilon_{xy}+\varepsilon_{yz}$. This implies that $\varepsilon_1 > \varepsilon$ and thus, by Property 3, $\rho_y = (1-\varepsilon_1)\rho_y^5+\varepsilon_1\rho_y^7$ where ρ_y^7 is the probability measure $\rho_y^7 = \left(1-\frac{\varepsilon}{\varepsilon_1}\right)\rho_y^5+\frac{\varepsilon}{\varepsilon_1}\rho_y^6$. Combining the decompositions of ρ_y, we have that

$$(1-\varepsilon_{xy})\rho_y^1+\varepsilon_{xy}\rho_y^2 = (1-\varepsilon_1)\rho_y^5+\varepsilon_1\rho_y^7$$

and thus

$$\begin{aligned}
\rho_y^1 &= \frac{(1-\varepsilon_1)\rho_y^5+\varepsilon_1\rho_y^7-\varepsilon_{xy}\rho_y^2}{1-\varepsilon_{xy}} \\
&= \frac{1-\varepsilon_1}{1-\varepsilon_{xy}}\rho_y^5+\frac{\varepsilon_{yz}}{1-\varepsilon_{xy}}\left(\frac{\varepsilon_1}{\varepsilon_{yz}}\rho_y^7-\frac{\varepsilon_{xy}}{\varepsilon_{yz}}\rho_y^2\right) \\
&= \frac{1-\varepsilon_{xy}-\varepsilon_{yz}}{1-\varepsilon_{xy}}\rho_y^5+\frac{\varepsilon_{yz}}{1-\varepsilon_{xy}}\left(\frac{\varepsilon_1}{\varepsilon_{yz}}\rho_y^7+\frac{\varepsilon_{yz}-\varepsilon_1}{\varepsilon_{yz}}\rho_y^2\right) \\
&= \left(1-\frac{\varepsilon_{yz}}{1-\varepsilon_{xy}}\right)\rho_y^5+\frac{\varepsilon_{yz}}{1-\varepsilon_{xy}}\left(\left(1-\frac{\varepsilon_1}{\varepsilon_{yz}}\right)\rho_y^2+\frac{\varepsilon_1}{\varepsilon_{yz}}\rho_y^7\right) \\
&= (1-\varepsilon_2)\rho_y^5+\varepsilon_2\rho_y^8
\end{aligned}$$

where ε_2 and ρ_y^8 are defined as

$$\varepsilon_2 = \frac{\varepsilon_{yz}}{1-\varepsilon_{xy}} \qquad \rho_y^8 = \left(1-\frac{\varepsilon_1}{\varepsilon_{yz}}\right)\rho_y^2+\frac{\varepsilon_1}{\varepsilon_{yz}}\rho_y^7$$

Since $\varepsilon_{xy} < 1$, we have that $1-\varepsilon_{xy} > 0$ and thus $\varepsilon_2 > 0$ (since $\varepsilon_{yz} > 0$). $\varepsilon_{xy}+\varepsilon_{yz} < 1$ implies that $\varepsilon_{yz} < 1-\varepsilon_{xy}$ and hence $\varepsilon_2 = \frac{\varepsilon_{yz}}{1-\varepsilon_{xy}} < 1$. ρ_y^8 is a probability measure since $\rho_y^8(\emptyset) = \left(1-\frac{\varepsilon_1}{\varepsilon_{yz}}\right)\rho_y^2(\emptyset)+\frac{\varepsilon_1}{\varepsilon_{yz}}\rho_y^7(\emptyset) = \left(1-\frac{\varepsilon_1}{\varepsilon_{yz}}\right)0+\frac{\varepsilon_1}{\varepsilon_{yz}}0 = 0$; $\rho_y^8(Y) = \left(1-\frac{\varepsilon_1}{\varepsilon_{yz}}\right)\rho_y^2(Y)+\frac{\varepsilon_1}{\varepsilon_{yz}}\rho_y^7(Y) = \left(1-\frac{\varepsilon_1}{\varepsilon_{yz}}\right)1+\frac{\varepsilon_1}{\varepsilon_{yz}}1 = 1$. The last condition ρ_y^8 must satisfy to be a probability measure is that $\rho_y^8 \geqslant 0$:

$$\begin{aligned}
\rho_y^8 \geqslant 0 &\iff \left(1-\frac{\varepsilon_1}{\varepsilon_{yz}}\right)\rho_y^2+\frac{\varepsilon_1}{\varepsilon_{yz}}\rho_y^7 \geqslant 0 \\
&\iff \rho_y^7 \geqslant \frac{\varepsilon_{yz}}{\varepsilon_1}\left(\frac{\varepsilon_1}{\varepsilon_{yz}}-1\right)\rho_y^2
\end{aligned}$$

$$\begin{aligned}
&\iff \rho_y^7 \geqslant \left(1 - \frac{\varepsilon_{yz}}{\varepsilon_1}\right) \rho_y^2 \\
&\iff \rho_y^7 \geqslant \frac{\varepsilon_{xy}}{\varepsilon_1} \rho_y^2
\end{aligned}$$

where the last equivalence is justified by

$$1 - \frac{\varepsilon_{yz}}{\varepsilon_1} = \frac{\varepsilon_1 - \varepsilon_{yz}}{\varepsilon_1} = \frac{\varepsilon_{xy} + \varepsilon_{yz} - \varepsilon_{yz}}{\varepsilon_1} = \frac{\varepsilon_{xy}}{\varepsilon_1}$$

We prove the condition $\rho_y^7 \geqslant \frac{\varepsilon_{xy}}{\varepsilon_1} \rho_y^2$ as follows:

$$\begin{aligned}
\varepsilon_{xy} + \varepsilon_{yx} > \max\{\varepsilon_{xy}, \varepsilon_{yx}\} &\implies \varepsilon_1 > \varepsilon \\
&\implies \varepsilon_1 - \varepsilon > 0 \\
&\implies (\varepsilon_1 - \varepsilon)\rho_y^5 \geqslant 0 \\
\max\{\varepsilon_{xy}\rho_y^2, \varepsilon_{yz}\rho_y^4\} \geqslant \varepsilon_{xy}\rho_y^2 &\implies (\varepsilon_1 - \varepsilon)\rho_y^5 \\
&\quad + \varepsilon \frac{\max\{\varepsilon_{xy}\rho_y^2, \varepsilon_{yz}\rho_y^4\}}{\varepsilon} \geqslant \varepsilon_{xy}\rho_y^2 \\
&\implies (\varepsilon_1 - \varepsilon)\rho_y^5 + \varepsilon\rho_y^6 \geqslant \varepsilon_{xy}\rho_y^2 \\
&\implies \varepsilon_1 \left(\left(1 - \frac{\varepsilon}{\varepsilon_1}\right)\rho_y^5 + \frac{\varepsilon}{\varepsilon_1}\rho_y^6\right) \geqslant \varepsilon_{xy}\rho_y^2 \\
&\implies \varepsilon_1\rho_y^7 \geqslant \varepsilon_{xy}\rho_y^2 \\
&\implies \rho_y^7 \geqslant \frac{\varepsilon_{xy}}{\varepsilon_1}\rho_y^2
\end{aligned}$$

Thus $\rho_y^1 = (1-\varepsilon_2)\rho_y^5 + \varepsilon_2\rho_y^8$ is a probability measure. Analogously, from

$$(1 - \varepsilon_{yz})\rho_y^3 + \varepsilon_{yz}\rho_y^4 = (1 - \varepsilon_1)\rho_y^5 + \varepsilon_1\rho_y^7$$

we obtain

$$\begin{aligned}
\rho_y^3 &= \frac{(1 - \varepsilon_1)\rho_y^5 + \varepsilon_1\rho_y^7 - \varepsilon_{yz}\rho_y^4}{1 - \varepsilon_{yz}} \\
&= \frac{1 - \varepsilon_1}{1 - \varepsilon_{yz}}\rho_y^5 + \frac{\varepsilon_{xy}}{1 - \varepsilon_{yz}}\left(\frac{\varepsilon_1}{\varepsilon_{xy}}\rho_y^7 - \frac{\varepsilon_{yz}}{\varepsilon_{xy}}\rho_y^4\right) \\
&= \frac{1 - \varepsilon_{yz} - \varepsilon_{xy}}{1 - \varepsilon_{yz}}\rho_y^5 + \frac{\varepsilon_{xy}}{1 - \varepsilon_{yz}}\left(\frac{\varepsilon_1}{\varepsilon_{xy}}\rho_y^7 + \frac{\varepsilon_{xy} - \varepsilon_1}{\varepsilon_{xy}}\rho_y^4\right) \\
&= \left(1 - \frac{\varepsilon_{xy}}{1 - \varepsilon_{yz}}\right)\rho_y^5 + \frac{\varepsilon_{xy}}{1 - \varepsilon_{yz}}\left(\left(1 - \frac{\varepsilon_1}{\varepsilon_{xy}}\right)\rho_y^4 + \frac{\varepsilon_1}{\varepsilon_{xy}}\rho_y^7\right)
\end{aligned}$$

$$= (1-\varepsilon_3)\rho_y^5 + \varepsilon_3\rho_y^9$$

where ε_3 and ρ_y^9 are defined as

$$\varepsilon_3 = \frac{\varepsilon_{xy}}{1-\varepsilon_{yz}} \qquad \rho_y^9 = \left(1 - \frac{\varepsilon_1}{\varepsilon_{xy}}\right)\rho_y^4 + \frac{\varepsilon_1}{\varepsilon_{xy}}\rho_y^7$$

Since $\varepsilon_{yz} < 1$, we have that $1-\varepsilon_{yz} > 0$ and thus $\varepsilon_3 > 0$ (since $\varepsilon_{xy} > 0$). $\varepsilon_{xy} + \varepsilon_{yz} < 1$ implies that $\varepsilon_{xy} < 1-\varepsilon_{yz}$ and hence $\varepsilon_3 = \dfrac{\varepsilon_{xy}}{1-\varepsilon_{yz}} < 1$. ρ_y^9 is a probability measure since $\rho_y^9(\emptyset) = \left(1 - \dfrac{\varepsilon_1}{\varepsilon_{xy}}\right)\rho_y^4(\emptyset) + \dfrac{\varepsilon_1}{\varepsilon_{xy}}\rho_y^7(\emptyset) = \left(1 - \dfrac{\varepsilon_1}{\varepsilon_{xy}}\right)0 + \dfrac{\varepsilon_1}{\varepsilon_{xy}}0 = 0$; $\rho_y^9(Y) = \left(1 - \dfrac{\varepsilon_1}{\varepsilon_{xy}}\right)\rho_y^4(Y) + \dfrac{\varepsilon_1}{\varepsilon_{xy}}\rho_y^7(Y) = \left(1 - \dfrac{\varepsilon_1}{\varepsilon_{xy}}\right)1 + \dfrac{\varepsilon_1}{\varepsilon_{yz}}1 = 1$. The last condition ρ_y^9 must satisfy to be a probability measure is that $\rho_y^9 \geqslant 0$:

$$\begin{aligned}
\rho_y^9 \geqslant 0 &\iff \left(1 - \frac{\varepsilon_1}{\varepsilon_{xy}}\right)\rho_y^4 + \frac{\varepsilon_1}{\varepsilon_{xy}}\rho_y^7 \geqslant 0 \\
&\iff \rho_y^7 \geqslant \frac{\varepsilon_{xy}}{\varepsilon_1}\left(\frac{\varepsilon_1}{\varepsilon_{xy}} - 1\right)\rho_y^4 \\
&\iff \rho_y^7 \geqslant \left(1 - \frac{\varepsilon_{xy}}{\varepsilon_1}\right)\rho_y^4 \\
&\iff \rho_y^7 \geqslant \frac{\varepsilon_{yz}}{\varepsilon_1}\rho_y^4
\end{aligned}$$

where the last equivalence is justified by

$$1 - \frac{\varepsilon_{xy}}{\varepsilon_1} = \frac{\varepsilon_1 - \varepsilon_{xy}}{\varepsilon_1} = \frac{\varepsilon_{xy} + \varepsilon_{yz} - \varepsilon_{xy}}{\varepsilon_1} = \frac{\varepsilon_{yz}}{\varepsilon_1}$$

We prove the condition $\rho_y^7 \geqslant \dfrac{\varepsilon_{yz}}{\varepsilon_1}\rho_y^4$ as follows:

$$\begin{aligned}
\varepsilon_{xy} + \varepsilon_{yx} > \max\{\varepsilon_{xy}, \varepsilon_{yx}\} &\implies \varepsilon_1 > \varepsilon \\
&\implies \varepsilon_1 - \varepsilon > 0 \\
&\implies (\varepsilon_1 - \varepsilon)\rho_y^5 \geqslant 0 \\
\max\{\varepsilon_{xy}\rho_y^2, \varepsilon_{yz}\rho_y^4\} \geqslant \varepsilon_{yz}\rho_y^4 &\implies (\varepsilon_1 - \varepsilon)\rho_y^5 \\
&\quad + \varepsilon\frac{\max\{\varepsilon_{xy}\rho_y^2, \varepsilon_{yz}\rho_y^4\}}{\varepsilon} \geqslant \varepsilon_{yz}\rho_y^4
\end{aligned}$$

$$\begin{aligned}
&\implies (\varepsilon_1 - \varepsilon)\rho_y^5 + \varepsilon\rho_y^6 \geqslant \varepsilon_{yz}\rho_y^4 \\
&\implies \varepsilon_1 \left(\left(1 - \frac{\varepsilon}{\varepsilon_1}\right) \rho_y^5 + \frac{\varepsilon}{\varepsilon_1}\rho_y^6 \right) \geqslant \varepsilon_{yz}\rho_y^4 \\
&\implies \varepsilon_1 \rho_y^7 \geqslant \varepsilon_{yz}\rho_y^4 \\
&\implies \rho_y^7 \geqslant \frac{\varepsilon_{yz}}{\varepsilon_1}\rho_y^4
\end{aligned}$$

Thus $\rho_y^3 = (1-\varepsilon_3)\rho_y^5 + \varepsilon_3\rho_y^9$ is a probability measure. Since $\rho_y^1 = (1-\varepsilon_2)\rho_y^5 + \varepsilon_2\rho_y^8$ and $\rho_x^1 \ \mathcal{L}(\mathcal{R})\ \rho_y^1$, by Property 7 we have that there exist $\rho_x^3, \rho_x^4 \in Disc(X)$ such that $\rho_x^1 = (1-\varepsilon_2)\rho_x^3 + \varepsilon_2\rho_x^4$ and $\rho_x^3 \ \mathcal{L}(\mathcal{R})\ \rho_y^5$. Replacing ρ_x^1 into the definition of ρ_x, we have that

$$\begin{aligned}
\rho_x &= (1-\varepsilon_{xy})\rho_x^1 + \varepsilon_{xy}\rho_x^2 \\
&= (1-\varepsilon_{xy})(1-\varepsilon_2)\rho_x^3 + (1-\varepsilon_{xy})\varepsilon_2\rho_x^4 + \varepsilon_{xy}\rho_x^2 \\
&= (1-\varepsilon_{xy})\frac{1-\varepsilon_{xy}-\varepsilon_{yz}}{1-\varepsilon_{xy}}\rho_x^3 + (1-\varepsilon_{xy})\frac{\varepsilon_{yz}}{1-\varepsilon_{xy}}\rho_x^4 + \varepsilon_{xy}\rho_x^2 \\
&= (1-\varepsilon_{xy}-\varepsilon_{yz})\rho_x^3 + \varepsilon_{yz}\rho_x^4 + \varepsilon_{xy}\rho_x^2 \\
&= (1-\varepsilon_1)\rho_x^3 + \varepsilon_1\left(\frac{\varepsilon_{yz}}{\varepsilon_1}\rho_x^4 + \frac{\varepsilon_{xy}}{\varepsilon_1}\rho_x^2\right) \\
&= (1-\varepsilon_1)\rho_x^3 + \varepsilon_1\rho_x^5
\end{aligned}$$

where ρ_x^5 is the probability measure

$$\rho_x^5 = \frac{\varepsilon_{yz}}{\varepsilon_1}\rho_x^4 + \frac{\varepsilon_{xy}}{\varepsilon_1}\rho_x^2 = \frac{\varepsilon_1 - \varepsilon_{xy}}{\varepsilon_1}\rho_x^4 + \frac{\varepsilon_{xy}}{\varepsilon_1}\rho_x^2 = \left(1 - \frac{\varepsilon_{xy}}{\varepsilon_1}\right)\rho_x^4 + \frac{\varepsilon_{xy}}{\varepsilon_1}\rho_x^2$$

Since $\rho_y^3 = (1-\varepsilon_3)\rho_y^5 + \varepsilon_3\rho_y^9$ and $\rho_y^3 \ \mathcal{L}(\mathcal{S})\ \rho_z^1$, by Property 6 we have that there exist $\rho_z^3, \rho_z^4 \in Disc(Z)$ such that $\rho_z^1 = (1-\varepsilon_3)\rho_z^3 + \varepsilon_3\rho_z^4$ and $\rho_y^5 \ \mathcal{L}(\mathcal{S})\ \rho_z^3$. Replacing ρ_z^1 into the definition of ρ_z, we have that

$$\begin{aligned}
\rho_z &= (1-\varepsilon_{yz})\rho_z^1 + \varepsilon_{yz}\rho_z^2 \\
&= (1-\varepsilon_{yz})(1-\varepsilon_3)\rho_z^3 + (1-\varepsilon_{yz})\varepsilon_3\rho_z^4 + \varepsilon_{yz}\rho_z^2 \\
&= (1-\varepsilon_{yz})\frac{1-\varepsilon_{xy}-\varepsilon_{yz}}{1-\varepsilon_{yz}}\rho_z^3 + (1-\varepsilon_{yz})\frac{\varepsilon_{xy}}{1-\varepsilon_{yz}}\rho_z^4 + \varepsilon_{yz}\rho_z^2 \\
&= (1-\varepsilon_{xy}-\varepsilon_{yz})\rho_z^3 + \varepsilon_{xy}\rho_z^4 + \varepsilon_{yz}\rho_z^2 \\
&= (1-\varepsilon_1)\rho_z^3 + \varepsilon_1\left(\frac{\varepsilon_{xy}}{\varepsilon_1}\rho_z^4 + \frac{\varepsilon_{yz}}{\varepsilon_1}\rho_z^2\right) \\
&= (1-\varepsilon_1)\rho_z^3 + \varepsilon_1\rho_z^5
\end{aligned}$$

where ρ_z^5 is the probability measure

$$\rho_z^5 = \frac{\varepsilon_{xy}}{\varepsilon_1}\rho_z^4 + \frac{\varepsilon_{yx}}{\varepsilon_1}\rho_z^2 = \frac{\varepsilon_1 - \varepsilon_{yz}}{\varepsilon_1}\rho_z^4 + \frac{\varepsilon_{yz}}{\varepsilon_1}\rho_z^2 = \left(1 - \frac{\varepsilon_{yz}}{\varepsilon_1}\right)\rho_z^4 + \frac{\varepsilon_{yz}}{\varepsilon_1}\rho_z^2$$

Since $\rho_x^3\ \mathcal{L}(\mathcal{R})\ \rho_y^5$ and $\rho_y^5\ \mathcal{L}(\mathcal{S})\ \rho_z^3$, then by Property 2.1(6) we have that $\rho_x^3\rho_z^3) \in \mathcal{L}(\mathcal{R} \circ \mathcal{S})$ and thus

$$\left.\begin{array}{l} \rho_x = (1-\varepsilon_1)\rho_x^3 + \varepsilon_1\rho_x^5 \\ \rho_z = (1-\varepsilon_1)\rho_z^3 + \varepsilon_1\rho_z^5 \\ (\rho_x^3, \rho_z^3) \in \mathcal{L}(\mathcal{R} \circ \mathcal{S}) \\ \varepsilon_1 = \varepsilon_{xy} + \varepsilon_{yz} \end{array}\right\} \implies (\rho_x, \rho_z) \in \mathcal{L}(\mathcal{R} \circ \mathcal{S}, \varepsilon_{xy} + \varepsilon_{yz})\ .$$

9. Let $\varepsilon = \max\{\varepsilon_{12}, \varepsilon_{23}\}$. This implies that
   - $\rho_1 = (1-\varepsilon)\rho_1' + (\varepsilon - \varepsilon_{12})\rho_1' + \varepsilon_{12}\rho_1''$,
   - $\rho_2 = (1-\varepsilon)\rho_2^1 + (\varepsilon - \varepsilon_{12})\rho_2^1 + \varepsilon_{12}\rho_2^2$,
   - $\rho_1'\ \mathcal{L}(\mathcal{R})\ \rho_2^1$

   and
   - $\rho_2 = (1-\varepsilon)\rho_2^3 + (\varepsilon - \varepsilon_{23})\rho_2^3 + \varepsilon_{23}\rho_2^4$,
   - $\rho_3 = (1-\varepsilon)\rho_3' + (\varepsilon - \varepsilon_{23})\rho_3' + \varepsilon_{23}\rho_3''$,
   - $\rho_2^3\ \mathcal{L}(\mathcal{S})\ \rho_3'$.

   Since $\rho_2^1 = \rho_2^3$, by Property 2.1(6) it follows that $\rho_1'\ \mathcal{L}(\mathcal{R} \circ \mathcal{S})\ \rho_3'$ and thus $\rho_1\ \mathcal{L}(\mathcal{R} \circ \mathcal{S}, \varepsilon)\ \rho_3$, that is $\rho_1\ \mathcal{L}(\mathcal{R} \circ \mathcal{S}, \max\{\varepsilon_{12}, \varepsilon_{23}\})\ \rho_3$.
10. Let $\mathcal{R}$ be a relation from X to Y, $\varepsilon \geqslant 0$, $\rho_x \in Disc(X)$ and $\rho_y \in Disc(Y)$ such that $\rho_x\ \mathcal{L}(\mathcal{R}, \varepsilon)\ \rho_y$. This implies that there exist ρ_x', ρ_x'', ρ_y', ρ_y'' such that $\rho_x = (1-\varepsilon)\rho_x' + \varepsilon\rho_x''$, $\rho_y = (1-\varepsilon)\rho_y' + \varepsilon\rho_y''$, and $\rho_x'\ \mathcal{L}(\mathcal{R})\ \rho_y'$. By Property 2.1(8), it follows that $\rho_x' \times \rho_z\ \mathcal{L}(\mathcal{R} \times id)\ \rho_y' \times \rho_z$. Since for each $(u, s) \in X \times Z$, we have that

$$\begin{aligned}(1-\varepsilon)\rho_x' \times \rho_z(u,s) + \varepsilon\rho_x'' \times \rho_z(u,s) &= (1-\varepsilon)\rho_x'(u)\rho_z(s) + \varepsilon\rho_x''(u)\rho_z(s) \\ &= ((1-\varepsilon)\rho_x'(u) + \varepsilon\rho_x''(u))\rho_z(s) \\ &= \rho_x(u)\rho_z(s) \\ &= \rho_x \times \rho_z(u,s)\end{aligned}$$

and for each $(v, t) \in Y \times Z$, we have that

$$\begin{aligned}(1-\varepsilon)\rho_y' \times \rho_z(v,t) + \varepsilon\rho_y'' \times \rho_z(v,t) &= (1-\varepsilon)\rho_y'(v)\rho_z(t) + \varepsilon\rho_y''(v)\rho_z(t) \\ &= ((1-\varepsilon)\rho_y'(v) + \varepsilon\rho_y''(v))\rho_z(t) \\ &= \rho_y(v)\rho_z(t) \\ &= \rho_y \times \rho_z(v,t)\end{aligned}$$

we have that $\rho_x \times \rho_z = (1-\varepsilon)\rho'_x \times \rho_z + \varepsilon\rho''_x \times \rho_z$ and $\rho_y \times \rho_z = (1-\varepsilon)\rho'_y \times \rho_z + \varepsilon\rho''_y \times \rho_z$. This implies, together with $\rho'_x \times \rho_z \; \mathcal{L}(\mathcal{R} \times id) \; \rho'_y \times \rho_z$, that $\rho_x \times \rho_z \; \mathcal{L}(\mathcal{R} \times id, \varepsilon) \; \rho_y \times \rho_z$. □

The Property 8 of ε-lifting can be easily extended to any number of measures and relations:

Proposition 3.6. *Let $n \in \mathbb{N}$, $n \geqslant 2$ and for each $0 \leqslant i < n$, let $\mathcal{R}_i$ be a relation from a set X_i to X_{i+1} and $\varepsilon_i \in \mathbb{R}$, $\varepsilon_i \geqslant 0$. For each $0 \leqslant i \leqslant n$, let ρ_i be a measure in $Disc(X_i)$. Let $\mathcal{R}=\mathcal{R}_1 \circ \ldots \circ \mathcal{R}_{n-1}$.*

If for each $0 \leqslant i < n$, $\rho_i \; \mathcal{L}(\mathcal{R}_i, \varepsilon_i) \; \rho_{i+1}$ then $\rho_0 \; \mathcal{L}(\mathcal{R}, \sum_{i=0}^{n-1} \varepsilon_i) \; \rho_n$.

Proof. The proof is a classical inductive argument on the number of relations.

Case $n = 2$: the result is immediate. In fact, by hypothesis we have two relations: $\mathcal{R}_0$ from X_0 to X_1, and $\mathcal{R}_1$ from X_1 to X_2, two values $\varepsilon_0 \geqslant 0$ and $\varepsilon_1 \geqslant 0$, and three measures $\rho_0 \in Disc(X_0)$, $\rho_1 \in Disc(X_1)$, and $\rho_2 \in Disc(X_2)$. If $\rho_0 \; \mathcal{L}(\mathcal{R}_0, \varepsilon_0) \; \rho_1$ and $\rho_1 \; \mathcal{L}(\mathcal{R}_1, \varepsilon_1) \; \rho_2$, then, by Property 3.5(8), we have that $\rho_0 \; \mathcal{L}(\mathcal{R}_0 \circ \mathcal{R}_1, \varepsilon_0 + \varepsilon_1) \; \rho_2$ and thus $\rho_0 \; \mathcal{L}(\mathcal{R}, \sum_{i=0}^{1} \varepsilon_i) \; \rho_2$.

Case $n > 2$: for each $0 \leqslant i < n$, let $\mathcal{R}_i$ be a relation from a set X_i to X_{i+1} and take $\varepsilon_i \in \mathbb{R}$, $\varepsilon_i \geqslant 0$. For each $0 \leqslant i \leqslant n$, let ρ_i be a measure in $Disc(X_i)$. Suppose that for each $0 \leqslant i < n$, $\rho_i \; \mathcal{L}(\mathcal{R}_i, \varepsilon_i) \; \rho_{i+1}$. Let $\mathcal{R}'=\mathcal{R}_1 \circ \ldots \circ \mathcal{R}_{n-2}$. By inductive hypothesis, we have that $\rho_0 \; \mathcal{L}(\mathcal{R}', \sum_{i=0}^{n-2} \varepsilon_i) \; \rho_{n-1}$. Since $\rho_{n-1} \; \mathcal{L}(\mathcal{R}_{n-1}, \varepsilon_{n-1}) \; \rho_n$, Property 3.5(8) implies that $\rho_0 \; \mathcal{L}(\mathcal{R}' \circ \mathcal{R}_{n-1}, (\sum_{i=0}^{n-2} \varepsilon_i) + \varepsilon_{n-1}) \; \rho_n$. $\mathcal{R}'=\mathcal{R}_1 \circ \ldots \circ \mathcal{R}_{n-2}$ implies that $\mathcal{R}' \circ \mathcal{R}_{n-1}=\mathcal{R}_1 \circ \ldots \circ \mathcal{R}_{n-2} \circ \mathcal{R}_{n-1}=\mathcal{R}$ and hence $\rho_0 \; \mathcal{L}(\mathcal{R}, \sum_{i=0}^{n-1} \varepsilon_i) \; \rho_n$, as required. □

The introduction of errors in execution simulations is then straightforward.

Definition 3.7. *An* ε-simulation *from a probabilistic automaton $\mathcal{A}_1$ to a probabilistic automaton $\mathcal{A}_2$ is a relation $\mathcal{R}$ from $Execs^*(\mathcal{A}_1)$ to $Execs^*(\mathcal{A}_2)$ such that:*

- *$\bar{s}_1 \; \mathcal{R} \; \bar{s}_2$ and*
- *for each pair $(\alpha_1, \alpha_2) \in \mathcal{R}$, if $\alpha_1 \longrightarrow \nu_1$, then there exists ν_2 such that $\alpha_2 \longrightarrow \nu_2$ and $\nu_1 \; \mathcal{L}(\mathcal{R}, \varepsilon) \; \nu_2$.*

The above definition is still not adequate for handling cryptographic protocols. The point is that we desire to reach a point where the parts of transitions that cannot be matched correspond to bad behavior like guessing a key or forging a signature. Given a finite execution α there is always a way to resolve nondeterminism so that a key is guessed; what is difficult to do is to guess a key once we have a probability measure over executions obtained by generating a key. This suggests that our step conditions should be based on measures over executions rather than single executions. Furthermore, it is convenient to consider also pairs of measures that are related up to some error γ and limit the increment of the error. This leads to a new proposal of simulation relation that we define below. However, we first need to extend the notation for transitions to measures over executions.

Definition 3.8. *Given a probabilistic automaton $\mathcal{A}$ and two probability measures $\nu, \upsilon \in Disc(Execs^*(\mathcal{A}))$, we say that there exists a transition from ν to υ, denoted by $\nu \longrightarrow \upsilon$, if there exists a scheduler σ such that for each finite execution αas, $\upsilon(C_{\alpha as}) = \nu(C_{\alpha as}) + \nu(\alpha) \sum_{tr \in D(a)} \sigma(\alpha)(tr) \cdot \mu_{tr}(s)$.*

We recall that $D(a)$ denotes the set of transitions labelled by action a and μ_{tr} denotes the target measure of transition tr.

Definition 3.9. *Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two probabilistic automata and let $\mathcal{R}$ be a relation from $Execs^*(\mathcal{A}_1)$ to $Execs^*(\mathcal{A}_2)$. We say that $\mathcal{R}$ is an* ε-execution simulation *from $\mathcal{A}_1$ to $\mathcal{A}_2$ if*

1. *$\bar{s}_1 \mathcal{R} \bar{s}_2$; and*
2. *for each $\gamma \geqslant 0$, $\nu_1 \in Disc(Execs^*(\mathcal{A}_1))$ and $\nu_2 \in Disc(Execs^*(\mathcal{A}_2))$, if*
   - *$\nu_1 \mathcal{L}(\mathcal{R}, \gamma) \nu_2$,*
   - *$\nu_1 \longrightarrow \upsilon_1$*

   then there exists υ_2 such that
   - *$\nu_2 \longrightarrow \upsilon_2$,*
   - *$\upsilon_1 \mathcal{L}(\mathcal{R}, \gamma + \varepsilon) \upsilon_2$.*

We say that $\mathcal{A}_1$ is ε-execution simulated by $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \preceq_e^{\varepsilon} \mathcal{A}_2$, if there exists an ε-execution simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$.

One meaningful property of the ε-execution simulation is that sequences of n steps can be matched up to an error $n\varepsilon$. This means that we can extend the bound on the error for a single step to the bound on the error for any number of steps. Before stating this property, we have to define what it means to reach a measure within n steps.

Definition 3.10. *Let $\mathcal{A}$ be a probabilistic automaton and σ be a scheduler for $\mathcal{A}$.*

We say that ν is a probability measure reached in at most n steps via σ if there is a sequence of probability measures $\nu_0, \ldots, \nu_n$ such that $\nu_0(\bar{s}) = 1$, $\nu_n = \nu$ and for each $0 \leqslant i < n$, σ schedules the transition $\nu_i \longrightarrow \nu_{i+1}$.

Proposition 3.11. *Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two probabilistic automata such that $\mathcal{A}_1 \preceq_e^{\varepsilon} \mathcal{A}_2$ for some $\varepsilon \geqslant 0$. Let $\mathcal{R}$ be an ε-execution simulation from $\mathcal{A}_1$ to $\mathcal{A}_2$.*

For each scheduler σ_1 for $\mathcal{A}_1$, if ν_1 is reached via σ_1 within n steps, then there exists a scheduler σ_2 for $\mathcal{A}_2$ that reaches, within n steps, a probability measure ν_2 such that $\nu_1 \; \mathcal{L}(\mathcal{R}, n\varepsilon) \; \nu_2$.

Proof. The proof is a classical inductive argument on the number of steps.

Case $n = 0$: within 0 steps, it is not possible to reach states different from start state of $\mathcal{A}_1$. This means that within 0 steps, reached measure is $\delta_{\bar{s}_1}$. Analogously, for $\mathcal{A}_2$ $\delta_{\bar{s}_2}$ is reached within 0 steps. Since $\bar{s}_1 \; \mathcal{R} \; \bar{s}_2$, then $\delta_{\bar{s}_1} \; \mathcal{L}(\mathcal{R}, 0) \; \delta_{\bar{s}_2}$.

Case $n > 0$: let ν_1 and ν_2 be two measures such that there exist schedulers σ_1 and σ_2 such that ν_1 is reached via σ_1 within n steps, ν_2 is reached via σ_2 within n steps, and $\nu_1 \; \mathcal{L}(\mathcal{R}, n\varepsilon) \; \nu_2$.

Suppose that there exists υ_1 such that $\nu_1 \longrightarrow \upsilon_1$. Let σ_1' be the scheduler that acts as σ_1 until ν_1 is reached within n steps and then it induces the step $\nu_1 \longrightarrow \upsilon_1$. By hypothesis, there exists a scheduler θ that induces the transition $\nu_2 \longrightarrow \upsilon_2$ with $\upsilon_1 \; \mathcal{L}(\mathcal{R}, n\varepsilon + \varepsilon) \; \upsilon_2$. This implies that there exists a scheduler σ_2' for $\mathcal{A}_2$ that reaches within $n + 1$ steps a measure υ_2 such that $\upsilon_1 \; \mathcal{L}(\mathcal{R}, n\varepsilon + \varepsilon) \; \upsilon_2$, that is $\upsilon_1 \; \mathcal{L}(\mathcal{R}, (n+1)\varepsilon) \; \upsilon_2$. □

We are now left with the computational aspects of our definition. For the purpose we talk about families of probabilistic automata and families of relations parameterized over a security parameter k. Furthermore, we impose the step condition only for measures that are reachable within a number of steps that is polynomial in k.

Definition 3.12. *Let $\{\mathcal{A}_k^1\}_{k \in K}$ and $\{\mathcal{A}_k^2\}_{k \in K}$ be two families of probabilistic automata; let $\mathcal{R} = \{\mathcal{R}_k\}_{k \in K}$ be a family of relations such that, for each $k \in K$, $\mathcal{R}_k$ is a relation from $Execs^*(\mathcal{A}_k^1)$ to $Execs^*(\mathcal{A}_k^2)$; let Poly be the set of positive polynomials over $\mathbb{N}$.*

We say that $\mathcal{R}$ is a polynomially accurate simulation *from $\{\mathcal{A}_k^1\}_{k \in K}$ to $\{\mathcal{A}_k^2\}_{k \in K}$ if*

1. *for each k, it holds that $\bar{s}_k^1 \; \mathcal{R}_k \; \bar{s}_k^2$;*
2. *for each $c \in \mathbb{N}$ and for each $p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$, for all probability measures ν_1 and ν_2 and for each $\gamma \geqslant 0$, if*
   - *ν_1 is reached in at most $p(k)$ steps in $\mathcal{A}_k^1$,*
   - *$\nu_1 \; \mathcal{L}(\mathcal{R}_k, \gamma) \; \nu_2$,*
   - *$\nu_1 \longrightarrow \upsilon_1$*

   then there exists υ_2 such that
   - *$\nu_2 \longrightarrow \upsilon_2$,*
   - *$\upsilon_1 \; \mathcal{L}(\mathcal{R}_k, \gamma + k^{-c}) \; \upsilon_2$.*

We write $\{\mathcal{A}_k^1\}_{k \in K} \lesssim \{\mathcal{A}_k^2\}_{k \in K}$ if there exists a polynomially accurate simulation $\mathcal{R}$ from $\{\mathcal{A}_k^1\}_{k \in K}$ to $\{\mathcal{A}_k^2\}_{k \in K}$.

The definition of polynomially accurate simulation satisfies a few important properties. The first property is that the existence of a polynomially accurate simulation allows us to match any polynomial number of steps with an error that is bounded by any polynomial; the second property is that execution simulation implies polynomially accurate simulation; and the third property is the execution correspondence.

Theorem 3.13. *Let $\{\mathcal{A}_k^1\}_{k \in K}$ and $\{\mathcal{A}_k^1\}_{k \in K}$ be two families of probabilistic automata such that $\{\mathcal{A}_k^1\}_{k \in K} \lesssim \{\mathcal{A}_k^2\}_{k \in K}$. Let $\mathcal{R} = \{\mathcal{R}_k\}_{k \in K}$ be a polynomially accurate simulation from $\{\mathcal{A}_k^1\}_{k \in K}$ to $\{\mathcal{A}_k^2\}_{k \in K}$.*

For each $c \in \mathbb{N}$, $p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ and each scheduler σ_1 for $\mathcal{A}_k^1$, if ν_1 is the probability measure induced by σ_1 after n steps, $n \leqslant p(k)$, then there exists a scheduler σ_2 for $\mathcal{A}_k^2$ that reaches, after n steps, a probability measure ν_2 such that $\nu_1 \; \mathcal{L}(\mathcal{R}_k, nk^{-c}) \; \nu_2$.

Proof. The proof is a classical inductive argument on the number of steps.

Case $n = 0$: for each $c \in \mathbb{N}$, $p \in Poly$, and $k \in \mathbb{N}$ after 0 steps, it is not possible to reach states different from start state of $\mathcal{A}_k^1$. This means that after 0 steps, reached measure is $\delta_{\bar{s}_1}$. Analogously, for $\mathcal{A}_k^2$ $\delta_{\bar{s}_2}$ is reached after 0 steps. Since $\bar{s}_1 \; \mathcal{R}_k \; \bar{s}_2$, we have that $\delta_{\bar{s}_1} \; \mathcal{L}(\mathcal{R}_k, 0) \; \delta_{\bar{s}_2}$ and thus $\delta_{\bar{s}_1} \; \mathcal{L}(\mathcal{R}_k, 0k^{-c}) \; \delta_{\bar{s}_2}$.

Case $n > 0$: fix $c \in \mathbb{N}$, $p \in Poly$. Let $\bar{k} \in \mathbb{N}$ be a value such that for each $k > \bar{k}$, $n < p(k)$. Let ν_1 and ν_2 be two measures such that there exist schedulers σ_1 and σ_2 such that ν_1 is reached via σ_1 after n steps, ν_2 is reached via σ_2 after n steps, and $\nu_1 \; \mathcal{L}(\mathcal{R}_k, nk^{-c}) \; \nu_2$.

Suppose that there exists υ_1 such that $\nu_1 \longrightarrow \upsilon_1$. Since $n < p(k)$ and ν_1 is reached after n steps, it follows that υ_1 is reached after $n+1 \leqslant p(k)$ steps. By hypothesis, $\{\mathcal{A}_k^1\}_{k\in K} \lesssim \{\mathcal{A}_k^2\}_{k\in K}$ and thus there exists $\bar{k}' \in \mathbb{N}$ such that for each $k > \bar{k}'$ there exists a scheduler θ that induces the transition $\nu_2 \longrightarrow \upsilon_2$ with $\upsilon_1 \ \mathcal{L}(\mathcal{R}, nk^{-c} + k^{-c}) \ \upsilon_2$. This implies that there exists a scheduler σ_2' for $\mathcal{A}_2$ that reaches after $n+1$ steps a measure υ_2 such that $\upsilon_1 \ \mathcal{L}(\mathcal{R}_k, nk^{-c} + k^{-c}) \ \upsilon_2$, that is $\upsilon_1 \ \mathcal{L}(\mathcal{R}_k, (n+1)k^{-c}) \ \upsilon_2$. □

The second property of polynomially accurate simulation is that it is implied by the ordinary simulation of probabilistic automata:

Proposition 3.14. *Let $\mathcal{A}_n^1$ and $\mathcal{A}_n^2$ be two automata parameterized on $n \in \mathbb{N}$.*

If for each $n \in \mathbb{N}$ $\mathcal{A}_n^1 \preceq \mathcal{A}_n^2$, then $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.

Proof. For each $n \in \mathbb{N}$, let $\mathcal{R}_n$ be the simulation that justifies $\mathcal{A}_n^1 \preceq \mathcal{A}_n^2$ and let $\mathcal{R}_n'$ be the relation from $Execs^*(\mathcal{A}_n^1)$ to $Execs^*(\mathcal{A}_n^2)$ defined as $\alpha_1 \ \mathcal{R}_n' \ \alpha_2$ if $|\alpha_1| = |\alpha_2|$ and for each $0 \leqslant i \leqslant |\alpha_1|$, $s_i^1 \ \mathcal{R} \ s_i^2$ where s_i^j denotes the i-th state of α_j. $\mathcal{R} = \{\mathcal{R}_k'\}_{k\in\mathbb{N}}$ is a polynomially accurate simulation from $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.

The condition on start states is trivially true, since by definition of ordinary simulation, it follows that for each $k \in \mathbb{N}$, $\bar{s}_k^1 \ \mathcal{R}_k \ \bar{s}_k^2$.

For the step condition, let $c \in \mathbb{N}$ and $p \in Poly$ fixed and suppose that there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ and $\gamma \geqslant 0$, we have that the measures ν_1, υ_1 and ν_2 satisfy $\nu_1 \ \mathcal{L}(\mathcal{R}_k, \gamma) \ \nu_2$ and $\nu_1 \longrightarrow \upsilon_1$. We must find υ_2 such that $\nu_2 \longrightarrow \upsilon_2$ and $\upsilon_1 \ \mathcal{L}(\mathcal{R}_k, \gamma + k^{-c}) \ \upsilon_2$.

If $\gamma \geqslant 1 - k^{-c}$, then for each measure υ_2 such that $\nu_2 \longrightarrow \upsilon_2$, υ_2 satisfies $\upsilon_1 \ \mathcal{L}(\mathcal{R}_k, \gamma + k^{-c}) \ \upsilon_2$. In fact, $\gamma \geqslant 1 - k^{-c}$ implies that $\gamma + k^{-c} \geqslant 1$ and by definition of $\mathcal{L}(\mathcal{R}_k, \gamma + k^{-c})$, it follows that $\upsilon_1 \ \mathcal{L}(\mathcal{R}_k, \gamma + k^{-c}) \ \upsilon_2$.

So, suppose that $0 \leqslant \gamma < 1 - k^{-c}$ and let σ_1 be the scheduler that induces the transition $\nu_1 \longrightarrow \upsilon_1$.

Let σ_2 be the scheduler for $\mathcal{A}_2$ defined as: for each $\alpha_2 \in Execs^*(\mathcal{A}_k^2)$ and each transition $tr_2 \in D_2$, $\sigma_2(\alpha_2)(tr_2) = \sigma_1(\alpha_1)(tr_1)$ if $\alpha_1 \ \mathcal{R}_k' \ \alpha_2$ and $\mu_{tr_1} \ \mathcal{L}(\mathcal{R}_k) \ \mu_{tr_2}$, 0 otherwise. Let w_{tr_1} be the weighting function that justifies $\mu_{tr_1} \ \mathcal{L}(\mathcal{R}_k) \ \mu_{tr_2}$.

Let υ_2 be the measure induced from ν_2 by σ_2. Then $\upsilon_1 \ \mathcal{L}(\mathcal{R}, \gamma) \ \upsilon_2$ and thus, by Property 3.5(2), $\upsilon_1 \ \mathcal{L}(\mathcal{R}, \gamma + k^{-c}) \ \upsilon_2$.

In fact, let υ_1^1 and υ_2^1 be the two measures induced by σ_1 and σ_2 from ν_1' and ν_2', respectively, and let υ_1^2 and υ_2^2 be the two measures induced by σ_1 and σ_2 from ν_1'' and ν_2'', respectively. By definition of transition, it follows that $\upsilon_1 = (1-\gamma)\upsilon_1^1 + \gamma\upsilon_1^2$ and $\upsilon_2 = (1-\gamma)\upsilon_2^1 + \gamma\upsilon_2^2$. Once we prove that $\upsilon_1^1 \; \mathcal{L}(\mathcal{R}_k') \; \upsilon_2^1$, we complete the proof.

Let w be the weighting function that justifies $\nu_1' \; \mathcal{L}(\mathcal{R}_k) \; \nu_2'$ and let w' be the function defined as: for each $\alpha_1 a s \in \mathit{Execs}^*(\mathcal{A}_k^1)$ and $\alpha_2 b t \in \mathit{Execs}^*(\mathcal{A}_k^2)$,

$$w'(\alpha_1 a s, \alpha_2 b t) = \begin{cases} w(\alpha_1 a s, \alpha_2 b t) + w(\alpha_1, \alpha_2) \sum_{tr \in D_1(a)} \sigma_1(\alpha_1)(tr) w_{tr}(s,t) \\ \qquad \text{if } \alpha_1 a s \; \mathcal{R}_k' \; \alpha_2 b t \\ 0 \\ \qquad \text{otherwise} \end{cases}$$

Condition 1 is trivially verified: let α_1 and α_2 be two executions such that $w'(\alpha_1, \alpha_2) > 0$. By definition of w', it follows that $\alpha_1 \; \mathcal{R}_k' \; \alpha_2$.

Condition 2 is also satisfied: for each $\alpha_1 a s$,

$$\begin{aligned} \upsilon_1^1(\alpha_1 a s) &= \nu_1'(\alpha_1 a s) + \nu_1'(\alpha_1) \sum_{tr \in D_1(a)} \sigma_1(\alpha_1)(tr) \cdot \mu_{tr}(s) \\ &= \sum_{\alpha_2} (w(\alpha_1 a s, \alpha_2) + w(\alpha_1, \alpha_2) \sum_{tr \in D_1(a)} \sigma_1(\alpha_1)(tr) \cdot \mu_{tr}(s)) \\ &= \sum_{\alpha_2 b t} (w(\alpha_1 a s, \alpha_2 b t) + w(\alpha_1, \alpha_2) \sum_{tr \in D_1(a)} \sigma_1(\alpha_1)(tr) \cdot w_{tr}(s,t)) \\ &= \sum_{\alpha_2 b t} w'(\alpha_1 a s, \alpha_2 b t) \end{aligned}$$

Condition 3 is also satisfied: for each $\alpha_2 a s$,

$$\begin{aligned} \upsilon_2^1(\alpha_2 b t) &= \nu_2'(\alpha_2 b t) + \nu_2'(\alpha_2) \sum_{tr \in D_2(b)} \sigma_2(\alpha_2)(tr) \cdot \mu_{tr}(t) \\ &= \sum_{\alpha_1} (w(\alpha_1, \alpha_2 b t) + w(\alpha_1, \alpha_2) \sum_{tr \in D_2(b)} \sigma_2(\alpha_2)(tr) \cdot \mu_{tr}(t)) \\ &= \sum_{\alpha_1 a s} (w(\alpha_1 a s, \alpha_2 b t) + w(\alpha_1, \alpha_2) \sum_{tr \in D_2(b)} \sigma_2(\alpha_2)(tr) \cdot \mu_{tr}(t)) \\ &= \sum_{\alpha_1 a s} (w(\alpha_1 a s, \alpha_2 b t) + w(\alpha_1, \alpha_2) \sum_{tr \in D_1(b)} \sigma_1(\alpha_1)(tr) \cdot w_{tr}(s,t)) \\ &= \sum_{\alpha_1 a s} w'(\alpha_1 a s, \alpha_2 b t) \end{aligned}$$

□

Corollary 3.15. *Let $\mathcal{A}_n^1$ and $\mathcal{A}_n^2$ be two automata parameterized on $n \in \mathbb{N}$. If for each $n \in \mathbb{N}$ $\mathcal{A}_n^1 \preceq_e \mathcal{A}_n^2$, then $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.*

Proof. By Proposition 3.3, for each $k \in \mathbb{N}$, $\mathcal{A}_k^1 \preceq_e \mathcal{A}_k^2$ implies that $\mathcal{A}_k^1 \preceq \mathcal{A}_k^2$. By Proposition 3.14, we have that $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$. □

The third property of polynomially accurate simulation is the execution correspondence:

Theorem 3.16 (Execution Correspondence Theorem). *Let $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$, $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$, …, $\{\mathcal{A}_k^n\}_{k\in\mathbb{N}}$ be n families of probabilistic automata such that there exist $n-1$ families of relations $\{\mathcal{R}_k^1\}_{k\in\mathbb{N}}$, $\{\mathcal{R}_k^2\}_{k\in\mathbb{N}}$, …, $\{\mathcal{R}_k^{n-1}\}_{k\in\mathbb{N}}$ such for each $0 < i < n$, $\{\mathcal{R}_k^i\}_{k\in\mathbb{N}}$ is a polynomially accurate simulation from $\{\mathcal{A}_k^i\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k^{i+1}\}_{k\in\mathbb{N}}$.*

For each $c \in \mathbb{N}$ and $p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ and each probability measure $\nu^1 \in Disc(Execs^*(\mathcal{A}_k^1))$, if ν^1 is reachable within $p(k)$ steps in $\mathcal{A}_k^1$, then there exists $\nu^n \in Disc(Execs^*(\mathcal{A}_k^n))$ such that ν^n is reachable within $p(k)$ steps in $\mathcal{A}_k^n$ and ν^1 $\mathcal{L}(\mathcal{R}_k^1 \circ \ldots \circ \mathcal{R}_k^{n-1}, p(k)k^{-c})$ ν^n.

Proof. Fix $c \in \mathbb{N}$, $p \in Poly$ and suppose that there exists $\bar{k}' \in \mathbb{N}$ such that for each $k > \bar{k}'$, $\nu^1 \in Disc(Execs^*(\mathcal{A}_k^1))$ is reachable within $p(k)$ steps in $\mathcal{A}_k^1$. Let s the actual number of steps performed to reach ν^1.

By Theorem 3.13, it follows that for each $c_1' \in \mathbb{N}$, there exists $\bar{k}_1 \in \mathbb{N}$ such that for each $k > \bar{k}_1$, there exists a measure $\nu^2 \in Disc(Execs^*(\mathcal{A}_k^2))$ such that ν^2 is reachable in s steps and ν^1 $\mathcal{L}(\mathcal{R}_k^1, sk^{-c_1'})$ ν^2.

Since ν^2 is reachable in $\mathcal{A}_k^2$ in s steps, $s \leqslant p(k)$, it follows that for each $c_2' \in \mathbb{N}$, there exists $\bar{k}_2 \in \mathbb{N}$ such that for each $k > \bar{k}_2$, there exists a measure $\nu^3 \in Disc(Execs^*(\mathcal{A}_k^3))$ such that ν^3 is reachable in s steps and ν^2 $\mathcal{L}(\mathcal{R}_k^2, sk^{-c_2'})$ ν^3.

Iterating this reasoning, we obtain that for each $1 < i < n$ and each $c_i' \in \mathbb{N}$, there exists $\bar{k}_i \in \mathbb{N}$ such that for each $k > \bar{k}_i$, there exists a measure $\nu^{i+1} \in Disc(Execs^*(\mathcal{A}_k^{i+1}))$ such that ν^i is reachable in s steps and ν^i $\mathcal{L}(\mathcal{R}_k^i, sk^{-c_i'})$ ν^{i+1}.

Let $\bar{k} = \max\{\bar{k}', \bar{k}_1, \ldots, \bar{k}_{n-1}\}$. Since $\bar{k} \geqslant \bar{k}'$, for each $k > \bar{k}$ the measure ν^1 is still reachable within $p(k)$ steps and $s \leqslant p(k)$. Since for each $0 < i < n$ $\bar{k} \geqslant \bar{k}_i$, we still have that for each $c_i' \in \mathbb{N}$, and each $k > \bar{k}$, there exists a measure $\nu^{i+1} \in Disc(Execs^*(\mathcal{A}_k^{i+1}))$ such that ν^{i+1} is reachable in s steps and ν^i $\mathcal{L}(\mathcal{R}_k^i, sk^{-c_i'})$ ν^{i+1}. Since for each $0 < i < n$ we can choose the value of c_i', let $c' \in \mathbb{N}$ be a value such that for each $l \in \mathbb{N}$, $l > \bar{k}$, we have that

$(n-1)l^{-c'} \leqslant l^{-c}$; for each $0 < i < n$, take $c'_i = c'$. This implies that for each $k > \bar{k}$, there exists a measure $\nu^{i+1} \in Disc(Execs^*(\mathcal{A}_k^{i+1}))$ such that ν^{i+1} is reachable in s steps and $\nu^i \; \mathcal{L}(\mathcal{R}_k^i, sk^{-c'}) \; \nu^{i+1}$.

By Proposition 3.6, it follows that $\nu^1 \; \mathcal{L}(\mathcal{R}_k^1 \circ \ldots \circ \mathcal{R}_k^{n-1}, \sum_{i=1}^{n-1} sk^{-c'}) \; \nu^n$. Consider $\sum_{i=1}^{n-1} sk^{-c'}$. We have that $\sum_{i=1}^{n-1} sk^{-c'} = s\sum_{i=1}^{n-1} k^{-c'} = s(n-1)k^{-c'} \leqslant sk^{-c} \leqslant p(k)k^{-c}$. By Property 3.5(2), it follows that ν^n is reachable in $s \leqslant p(k)$ steps and $\nu^1 \; \mathcal{L}(\mathcal{R}_k^1 \circ \ldots \circ \mathcal{R}_k^{n-1}, p(k)k^{-c}) \; \nu^n$, as required. □

4

Derived Automata and Approximated Simulations

In this chapter we study two extensions of probabilistic automata that are induced by the notion of simulation. In particular, the first extension considers the adding history variables [5] and further actions that modify only the new variables while the second extension is the generalization of a result induced by our notion of polynomially accurate simulations.

4.1 Extending Automata with Actions and Variables

Given an automaton $\mathcal{A}$, we can define several automata that are very similar to $\mathcal{A}$ but that provide more actions or whose states are described by the same variables of states of $\mathcal{A}$ plus some other variables. For example, given the coin flipper of Figure 2.1 on page 32, we can consider the automaton that remembers the sequence of heads and tails it has returned. The sequence of heads and tails can be seen as a history of the execution of the coin flipper that is updated internally.

We can also add some actions that are used by other automata to communicate information to $\mathcal{A}$, information that are stored inside the state. For example, we can put in parallel a coin flipper and a dice roller and we want that the coin flipper keeps as history the sequence of heads, tails, and values of the dice. In this case, we modify the coin flipper adding an input action that is used by the dice roller to communicate the value of the dice and the effect of such action is to add the dice value to the history.

In addition, we want to communicate information kept inside an automaton to the outside. To do this, we can add some output action that sends information to the environment. For example, the coin flipper can output the sequence of heads and tails each time it flips the coin; such in-

formation can be received by an automaton that uses the sequence of heads and tails to control its own behavior.

As we will see in the next chapters (in particular, in Chapter 6 where we talk about properties of cryptographic primitives with respect to simulations), during a hierarchical verification we define an automaton and then we modify it adding information about the history of internal choices and about external actions. Resulting automata allow us to state properties about the exchanged values that are useful to prove our results. For this reason, we want to formalize the notion of extension of an automaton and then we prove some properties of extended automata.

The first problem we find in the formalization of extension is that given two states, it is easy to say when they share some information when they are described by variables, since we can simply compare the values of variables of interest. The problem is that the definition of probabilistic automaton does not impose to use variables to characterize a state, so given two states that at least one is not described by variables, it is not easy to find a general way to say that they share some information.

Since our aim is to consider automata that are obtained adding actions or information to the states of another automaton, we suppose that states of all automata are described using variables.

Definition 4.1. *Let s be a state that is described by a set of variables V. For each $v \in V$, we define the* projection of s on v*, denoted as $s.v$, as the value of variable v in s.*

Given an automaton $\mathcal{A}$, if states are not described by variables, then we assume that states are characterized by the variable *state* that assumes values that belong to *States* and we define $\cdot.state$ as $s.state = s$ for each state s.

Now we are able to say when two states share the same information.

Definition 4.2. *Let s_1 and s_2 be two states described by sets of variables V_1 and V_2, respectively and let V be $V_1 \cap V_2$. We say that s_1* and s_2 are equal on common variables *if for each variable $v \in V$, $s_1.v = s_2.v$.*

If $V_1 \subseteq V_2$, then we say that s_1 is equal to s_2 on common variables *(denoted by $s_1 = s_2{\restriction}_{s_1}$) if s_1 and s_2 are equal on common variables.*

The notion of equality of states can be easily extended to measures over states:

Definition 4.3. *Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two automata whose states are described by variables V_1 and V_2, respectively. Suppose that $V_1 \subseteq V_2$.*

Given $\mu_1 \in Disc(States_1)$ and $\mu_2 \in Disc(States_2)$, we say that μ_1 is equal to μ_2 on common variables *(denoted by $\mu_1 = \mu_2{\restriction_{\mu_1}}$) if for each $s_1 \in Supp(\mu_1)$, $\mu_1(s_1) = \mu_2(E(s_1))$ where $E(s_1) = \{s_2 \in States_2 \mid s_1 = s_2{\restriction_{s_1}}\}$.*

Now it is straightforward to extend the notion of equality to transitions:

Definition 4.4. *Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two automata whose states are described by variables V_1 and V_2, respectively. Suppose that $V_1 \subseteq V_2$.*

Given two transitions $tr_1 = (s_1, a_1, \mu_1) \in D_1$ and $tr_2 = (s_2, a_2, \mu_2) \in D_2$, we say that tr_1 is equal to tr_2 on common variables *(denoted by $tr_1 = tr_2{\restriction_{tr_1}}$) if*

- $s_1 = s_2{\restriction_{s_1}}$
- $a_1 = a_2$, *and*
- $\mu_1 = \mu_2{\restriction_{\mu_1}}$.

Now we can define our notion of extention of automata:

Definition 4.5. *Let $\mathcal{A}$ be an automaton and let V be the set of variables that describes states of $\mathcal{A}$.*

Let W be a set of variables such that $W \cap V = \emptyset$ and let B be a set of actions such that $B \cap A = \emptyset$.

Given the automaton $\mathcal{A}'$ we say that $\mathcal{A}'$ extends $\mathcal{A}$ with state variables W and actions B *(also denoted by $\mathcal{A}' \in \mathrm{Ext}_B^W(\mathcal{A})$) if $\mathcal{A}'$ is an automaton $(S', \bar{s}', A', D')$ that satisfies the following conditions:*

compatible states: states of $\mathcal{A}'$ are described by variables $V \cup W$;
compatible start state: $\bar{s}'$ satisfies $\bar{s} = \bar{s}'{\restriction_{\bar{s}}}$;
compatible actions: $A' = A \cup B$ and $H' = H$; and
compatible transitions: for each transition $tr \in D$, there exists $tr' \in D'$ such that $tr = tr'{\restriction_{tr}}$, and for each transition $tr' = (s', a', \mu') \in D'$,
- *either $a' \in B$ and there exists $s \in States$ such that $s = s'{\restriction_s}$ and $\delta_s = \mu'{\restriction_{\delta_s}}$,*
- *or $a' \in A$ and there exists $tr \in D$ such that $tr = tr'{\restriction_{tr}}$.*

Informally, an automaton $\mathcal{A}'$ is an extension of an automaton $\mathcal{A}$ each time their behavior is the same on common parts. This means, for example, that the extra information kept by $\mathcal{A}'$ does not affect the actions that are also provided by $\mathcal{A}$, that is, we do not base our choices on the information we have in $\mathcal{A}'$ but not in $\mathcal{A}$. In a similar way, the new actions we provide in $\mathcal{A}'$ do not modify the variables that are also variables of $\mathcal{A}$. In general,

given an automaton $\mathcal{A}$ if we add history variable and actions that modify only them, then the resulting automaton is an extension of $\mathcal{A}$.

The above consideration allows us to suppose that an automaton $\mathcal{A}$ is simulated by its extension $\mathcal{A}'$, since each transition of $\mathcal{A}$ can be matched by the corresponding transition of $\mathcal{A}'$. In fact,

Lemma 4.6. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two automata such that there exists a set of variables W and a set of actions B such that $\mathcal{A}_2 \in \mathrm{Ext}_B^W(\mathcal{A}_1)$. Let V the set of variables that describe states of $\mathcal{A}_1$.*

If C is a context compatible with $\mathcal{A}_2$ such that B is contained into actions of C, then $\mathcal{A}_1||C \preceq \mathcal{A}_2||C$.

Proof. Before proving the existence of the simulation, we must be sure that $\mathcal{A}_1$ is compatible with C. This is true since, by hypothesis,

- $A_2 \cap H_C = \emptyset$ and thus $A_1 \cap H_C = \emptyset$ since $A_2 = A_1 \cup B$; and
- $H_2 \cap A_C = \emptyset$ and thus $H_1 \cap A_C = \emptyset$ since $H_2 = H_1$.

Let S_C the set of states of context C and $\mathcal{R}$ be a relation from $S \times S_C$ to $S' \times S_C$ such that $(s_1, c_1)\ \mathcal{R}\ (s_2, c_2)$ if and only if $c_2 = c_1$ and $s_1 = s_2{\restriction_{s_1}}$.

$\mathcal{R}$ is a simulation from $\mathcal{A}_1||C$ to $\mathcal{A}_2||C$.

Condition on start states is trivially true: let $\bar{c}$ be the start state of C; since by definition of $\mathrm{Ext}_B^W(\mathcal{A}_1)$ $\bar{s}_1 = \bar{s}_2{\restriction_{\bar{s}_1}}$, then $(\bar{s}_1, \bar{c})\ \mathcal{R}\ (\bar{s}_2, \bar{c})$.

For the step condition, let (s_1, c_1), (s_2, c_2) be two states of $\mathcal{A}_1||C$ and $\mathcal{A}_2||C$ respectively such that $(s_1, c_1)\ \mathcal{R}\ (s_2, c_2)$ (and thus, $c_2 = c_1$ and $s_1 = s_2{\restriction_{s_1}}$). Let a be an action and μ_1 a probability measure such that $((s_1, c_1), a, \mu_1)$ is a transition of $\mathcal{A}_1||C$; let $\mu_{\mathcal{A}_1}$ and μ_C be the two measures such that $\mu_1 = \mu_{\mathcal{A}_1} \times \mu_C$. There are three cases:

- a is an action of C but not of $\mathcal{A}_2$: definition of composition implies that $\mu_1 = \delta_{s_1} \times \mu_C$. Let $\mu_2 = \delta_{s_2} \times \mu_C$. By definition of composition, $((s_2, c_1), a, \mu_2)$ is a transition of $\mathcal{A}_2||C$ and $\mu_1\ \mathcal{L}(\mathcal{R})\ \mu_2$. In fact, for each pair of composed states (s_1', c_1'), (s_2', c_2'), define

$$w((s_1', c_1'), (s_2', c_2')) = \begin{cases} \delta_{s_1}(s_1')\delta_{s_2}(s_2')\mu_C(c_1') & \text{if } (s_1', c_1')\ \mathcal{R}\ (s_2', c_2') \\ 0 & \text{otherwise.} \end{cases}$$

  w is a weighting function:
  1. if $w((s_1', c_1'), (s_2', c_2')) > 0$, then $(s_1', c_1')\ \mathcal{R}\ (s_2', c_2')$ by definition of w;
  2. for each (s_2', c_2'),

$$\begin{aligned}
\sum_{(s_1',c_1')} w((s_1',c_1'),(s_2',c_2')) &= \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} w((s_1',c_1'),(s_2',c_2')) \\
&\quad + \sum_{(s_1',c_1')\neg\mathcal{R}(s_2',c_2')} w((s_1',c_1'),(s_2',c_2')) \\
&= \text{by definition of } w \\
&\quad \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} w((s_1',c_1'),(s_2',c_2')) \\
&= \text{by definition of } w \\
&\quad \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} \delta_{s_1}(s_1')\delta_{s_2}(s_2')\mu_C(c_1') \\
&= \text{by definition of } \mathcal{R} \\
&\quad \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} \delta_{s_1}(s_1')\delta_{s_2}(s_2')\mu_C(c_2') \\
&= \delta_{s_1}(s_1)\delta_{s_2}(s_2')\mu_C(c_2') \\
&\quad + \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2'),s_1'\neq s_1} \delta_{s_1}(s_1')\delta_{s_2}(s_2')\mu_C(c_2') \\
&= \text{by definition of } \delta_{s_1} \\
&\quad \delta_{s_2}(s_2')\mu_C(c_2') \\
&= \text{by definition of } \mu_2 \\
&\quad \mu_2(s_2',c_2').
\end{aligned}$$

3. for each (s_1', c_1'),

$$\begin{aligned}
\sum_{(s_2',c_2')} w((s_1',c_1'),(s_2',c_2')) &= \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} w((s_1',c_1'),(s_2',c_2')) \\
&\quad + \sum_{(s_1',c_1')\neg\mathcal{R}(s_2',c_2')} w((s_1',c_1'),(s_2',c_2')) \\
&= \text{by definition of } w \\
&\quad \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} w((s_1',c_1'),(s_2',c_2')) \\
&= \text{by definition of } w \\
&\quad \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} \delta_{s_1}(s_1')\delta_{s_2}(s_2')\mu_C(c_1') \\
&= \delta_{s_1}(s_1')\delta_{s_2}(s_2)\mu_C(c_2')
\end{aligned}$$

$$
\begin{aligned}
&+ \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2'),s_2'\neq s_2} \delta_{s_1}(s_1')\delta_{s_2}(s_2')\mu_C(c_2') \\
&= \text{by definition of } \delta_{s_2} \\
&\delta_{s_1}(s_1')\mu_C(c_1') \\
&= \text{by definition of } \mu_1 \\
&\mu_1(s_1', c_1').
\end{aligned}
$$

– a is an action of $\mathcal{A}_1$: by definition of composition, we have that $tr_1 = (s_1, a, \mu_{\mathcal{A}_1})$ is a transition of $\mathcal{A}_1$. By definition of extension, it follows that there exists a transition $tr_2 = (s_2, a, \mu_{\mathcal{A}_2})$ such that $tr_1 = tr_2{\uparrow}_{tr_1}$. Let μ_2 be the distribution $\mu_{\mathcal{A}_2} \times \mu_C$. By definition of composition, it follows that $((s_2, c_2), a, \mu_2)$ is a transition of $\mathcal{A}_2||C$ and $\mu_1 \; \mathcal{L}(\mathcal{R}) \; \mu_2$. In fact, for each pair of composed states (s_1', c_1'), (s_2', c_2'), define

$$
w((s_1', c_1'), (s_2', c_2')) = \begin{cases} \mu_{\mathcal{A}_1}(s_1') \dfrac{\mu_{\mathcal{A}_2}(s_2')}{\mu_{\mathcal{A}_2}(E(s_1'))} \mu_C(c_2') & \text{if } (s_1', c_1') \; \mathcal{R} \; (s_2', c_2') \\ 0 & \text{otherwise.} \end{cases}
$$

w is a weighting function:

1. if $w((s_1', c_1'), (s_2', c_2')) > 0$, then $(s_1', c_1') \; \mathcal{R} \; (s_2', c_2')$ by definition of w;
2. for each (s_2', c_2'),

$$
\begin{aligned}
\sum_{(s_1',c_1')} w((s_1', c_1'), (s_2', c_2')) &= \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} w((s_1', c_1'), (s_2', c_2')) \\
&\quad + \sum_{(s_1',c_1')\neg\mathcal{R}(s_2',c_2')} w((s_1', c_1'), (s_2', c_2')) \\
&= \text{by definition of } w \\
&\quad \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} w((s_1', c_1'), (s_2', c_2')) \\
&= \text{by definition of } w \\
&\quad \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} \mu_{\mathcal{A}_1}(s_1') \frac{\mu_{\mathcal{A}_2}(s_2')}{\mu_{\mathcal{A}_2}(E(s_1'))} \mu_C(c_2') \\
&= \text{by definition of } \mathcal{R} \\
&\quad \sum_{s_1' \text{ s.t. } s_1' = s_2'{\uparrow}_{s_1'}} \mu_{\mathcal{A}_1}(s_1') \frac{\mu_{\mathcal{A}_2}(s_2')}{\mu_{\mathcal{A}_2}(E(s_1'))} \mu_C(c_2') \\
&= \mu_C(c_2')\mu_{\mathcal{A}_2}(s_2') \sum_{s_1' \text{ s.t. } s_1' = s_2'{\uparrow}_{s_1'}} \frac{\mu_{\mathcal{A}_1}(s_1')}{\mu_{\mathcal{A}_2}(E(s_1'))}
\end{aligned}
$$

$$
\begin{aligned}
&= \text{by definition of extending automaton} \\
&\mu_C(c'_2)\mu_{\mathcal{A}_2}(s'_2)\frac{\mu_{\mathcal{A}_1}(s'_2\lfloor_V)}{\mu_{\mathcal{A}_2}(E(s'_2\lfloor_V))} \\
&= \text{by definition of extending automaton} \\
&\mu_C(c'_2)\mu_{\mathcal{A}_2}(s'_2)\frac{\mu_{\mathcal{A}_2}(E(s'_2\lfloor_V))}{\mu_{\mathcal{A}_2}(E(s'_2\lfloor_V))} \\
&= \mu_{\mathcal{A}_2}(s'_2)\mu_C(c'_2) \\
&= \text{by definition of } \mu_2 \\
&\mu_2(s'_2, c'_2).
\end{aligned}
$$

where $t_2\lfloor_V)$ denotes the single state t_1 of $\mathcal{A}_1$ such that $t_1 = t_2\lceil_{t_1}$.

3. for each (s'_1, c'_1),

$$
\begin{aligned}
\sum_{(s'_2,c'_2)} w((s'_1,c'_1),(s'_2,c'_2)) &= \sum_{(s'_1,c'_1)\mathcal{R}(s'_2,c'_2)} w((s'_1,c'_1),(s'_2,c'_2)) \\
&\quad + \sum_{(s'_1,c'_1)\neg\mathcal{R}(s'_2,c'_2)} w((s'_1,c'_1),(s'_2,c'_2)) \\
&= \text{by definition of } w \\
&\quad \sum_{(s'_1,c'_1)\mathcal{R}(s'_2,c'_2)} w((s'_1,c'_1),(s'_2,c'_2)) \\
&= \text{by definition of } w \\
&\quad \sum_{(s'_1,c'_1)\mathcal{R}(s'_2,c'_2)} \mu_{\mathcal{A}_1}(s'_1)\frac{\mu_{\mathcal{A}_2}(s'_2)}{\mu_{\mathcal{A}_2}(E(s'_1))}\mu_C(c'_2) \\
&= \text{by definition of } \mathcal{R} \\
&\quad \sum_{(s'_1,c'_1)\mathcal{R}(s'_2,c'_2)} \mu_{\mathcal{A}_1}(s'_1)\frac{\mu_{\mathcal{A}_2}(s'_2)}{\mu_{\mathcal{A}_2}(E(s'_1))}\mu_C(c'_1) \\
&= \text{by definition of } \mathcal{R} \\
&\quad \sum_{s'_2 \text{ s.t. } s'_1=s'_2\lceil_{s'_1}} \mu_{\mathcal{A}_1}(s'_1)\frac{\mu_{\mathcal{A}_2}(s'_2)}{\mu_{\mathcal{A}_2}(E(s'_1))}\mu_C(c'_1) \\
&= \mu_{\mathcal{A}_1}(s'_1)\mu_C(c'_1) \sum_{s'_2 \text{ s.t. } s'_1=s'_2\lceil_{s'_1}} \frac{\mu_{\mathcal{A}_2}(s'_2)}{\mu_{\mathcal{A}_2}(E(s'_1))} \\
&= \text{by definition of } E(s'_1) \\
&\quad \mu_{\mathcal{A}_1}(s'_1)\mu_C(c'_1)\frac{\mu_{\mathcal{A}_2}(E(s'_1))}{\mu_{\mathcal{A}_2}(E(s'_1))}
\end{aligned}
$$

$$
\begin{aligned}
&= \mu_{\mathcal{A}_1}(s'_1)\mu_C(c'_1) \\
&= \text{by definition of } \mu_1 \\
&\ \mu_1(s'_1, c'_1).
\end{aligned}
$$

– a is an action of $\mathcal{A}_2$ but not of $\mathcal{A}_1$: by definition of composition, we have that $\mu_1 = \delta_{s_1} \times \mu_C$. Let $\mu_2 = \mu_{\mathcal{A}_2} \times \mu_C$ where $\mu_{\mathcal{A}_2}$ is a measure such that $(s_2, a, \mu_{\mathcal{A}_2})$ is a transition of $\mathcal{A}_2$. By definition of composition, $((s_2, c_1), a, \mu_2)$ is a transition of $\mathcal{A}_2||C$ and $\mu_1\ \mathcal{L}(\mathcal{R})\ \mu_2$. In fact, for each pair of composed states (s'_1, c'_1), (s'_2, c'_2), define

$$
w((s'_1, c'_1), (s'_2, c'_2)) = \begin{cases} \delta_{s_1}(s'_1)\mu_{\mathcal{A}_2}(s'_2)\mu_C(c'_2) & \text{if } (s'_1, c'_1)\ \mathcal{R}\ (s'_2, c'_2) \\ 0 & \text{otherwise.} \end{cases}
$$

w is a weighting function:

1. if $w((s'_1, c'_1), (s'_2, c'_2)) > 0$, then $(s'_1, c'_1)\ \mathcal{R}\ (s'_2, c'_2)$ by definition of w;
2. for each (s'_2, c'_2),

$$
\begin{aligned}
\sum_{(s'_1, c'_1)} w((s'_1, c'_1), (s'_2, c'_2)) &= \sum_{(s'_1, c'_1)\mathcal{R}(s'_2, c'_2)} w((s'_1, c'_1), (s'_2, c'_2)) \\
&\quad + \sum_{(s'_1, c'_1)\neg\mathcal{R}(s'_2, c'_2)} w((s'_1, c'_1), (s'_2, c'_2)) \\
&= \text{by definition of } w \\
&\quad \sum_{(s'_1, c'_1)\mathcal{R}(s'_2, c'_2)} w((s'_1, c'_1), (s'_2, c'_2)) \\
&= \text{by definition of } w \\
&\quad \sum_{(s'_1, c'_1)\mathcal{R}(s'_2, c'_2)} \delta_{s_1}(s'_1)\mu_{\mathcal{A}_2}(s'_2)\mu_C(c'_2) \\
&= \delta_{s_1}(s_1)\mu_{\mathcal{A}_2}(s'_2)\mu_C(c'_2) \\
&\quad + \sum_{(s'_1, c'_1)\mathcal{R}(s'_2, c'_2), s'_1 \neq s_1} \delta_{s_1}(s'_1)\mu_{\mathcal{A}_2}(s'_2)\mu_C(c'_2) \\
&= \text{by definition of } \delta_{s_1} \\
&\ \mu_{\mathcal{A}_2}(s'_2)\mu_C(c'_2) \\
&= \text{by definition of } \mu_2 \\
&\ \mu_2(s'_2, c'_2).
\end{aligned}
$$

3. for each (s'_1, c'_1),

$$
\begin{aligned}
\sum_{(s_2',c_2')} w((s_1',c_1'),(s_2',c_2')) &= \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} w((s_1',c_1'),(s_2',c_2')) \\
&\quad + \sum_{(s_1',c_1')\neg\mathcal{R}(s_2',c_2')} w((s_1',c_1'),(s_2',c_2')) \\
&= \text{by definition of } w \\
&\quad \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} w((s_1',c_1'),(s_2',c_2')) \\
&= \text{by definition of } w \\
&\quad \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} \delta_{s_1}(s_1')\mu_{\mathcal{A}_2}(s_2')\mu_C(c_2') \\
&= \text{by definition of } \mathcal{R} \\
&\quad \sum_{(s_1',c_1')\mathcal{R}(s_2',c_2')} \delta_{s_1}(s_1')\mu_{\mathcal{A}_2}(s_2')\mu_C(c_1') \\
&= \text{by definition of } \mathcal{R} \\
&\quad \sum_{s_2' \text{ s.t. } s_1'=s_2'\restriction_{s_1'}} \delta_{s_1}(s_1')\mu_{\mathcal{A}_2}(s_2')\mu_C(c_1') \\
&= \delta_{s_1}(s_1')\mu_C(c_1') \sum_{s_2' \text{ s.t. } s_1'=s_2'\restriction_{s_1'}} \mu_{\mathcal{A}_2}(s_2') \\
&= \text{by definition of } \mu_{\mathcal{A}_2} \\
&\quad \delta_{s_1}(s_1')\mu_C(c_1') \\
&= \text{by definition of } \mu_1 \\
&\quad \mu_1(s_1',c_1').
\end{aligned}
$$

□

4.2 Simulations between Conditional Automata

With the definition of polynomially accurate simulation, we have a tool that allows us to say when two (families of) automata present a behavior that differs only on a negligible set of cases. This is very important during the verification of the security of a protocol, since given two automata that are related by a polynomially accurate simulation, then we can replace the first one with the second one and the overall behavior of the system changes only on a set of executions that occur with negligible probability.

In particular, the second one can ensure by construction that bad states can not be reached, that is, the probability to reach such states is zero.

In the previous sentence, we have called some states *bad*: given an automaton, we say that a state is bad when it represents an event we do not want to reach. For example, we say that a state s of a nonce generator is bad when in s we have a repeated nonce. In the same way, given a coin flipper that takes the history of the returned values, we may say that a state is bad when the history contains a sequence of five heads followed by two tails. We remark the fact that the classification of a state as bad or good depends on the context we are considering: if we are analyzing a cryptographic protocol, it is obvious to say that states where repeated nonces occur are bad while in the same context, it is difficult to find a general motivation to say when the states of the coin flipper are good and when they are bad.

Given a bad state or a set of bad states, we can consider the probability to reach them: if it is high, then the corresponding situations we want to avoid can occur with a probability that is too high for our purpose and that it is not acceptable. For example, consider a cryptographic protocol and label a state as bad when in such state the adversary has performed a successful attack. If we can reach such states with high probability, then we can consider the protocol not secure, since the adversary can break it with high probability.

The probability to reach bad states is usually related to the length of the execution we are analyzing. For example, consider again the case of the generation of nonces: by the main property of the nonces, we know that the probability to generate a nonce of length k that belongs a set N is negligible each time N has a polynomial size. If we add other values to N, then the probability to generate a nonce already in N grows. If N contains all nonces we have chosen during an execution, then an execution of exponential length could generate almost all available nonces and thus the probability to generate a repeated nonce grows and is 1 when $N = \{0,1\}^k$. In this case, all values we will choose are already in N and thus we will generate only repeated nonces. At the same manner, if we allow the adversary to run for an exponential time, then it is obvious that it attacks the protocol, since it can try all possible secrets until it finds the right ones. Once the adversary knows the secrets, the protocol is easily broken.

The above considerations lead us to define the probability of "bad" states with respect to the length of the execution. In particular, since our

aim is to analyse cryptographic protocols and since usually a protocol is said to be secure if the probability to attack it is negligible provided that the adversary is polynomially bounded, then we base our definition on negligible probabilities and executions of polynomial length.

Before providing such definition, we need to introduce some preliminary notions.

Definition 4.7. *We say that a state s of probabilistic automaton $\mathcal{A}$ is reachable if there exists $\alpha \in Execs^*(\mathcal{A})$ such that $lstate(\alpha) = s$.*

Definition 4.8. *Let $\mathcal{A}$ be an automaton and s be a state of $\mathcal{A}$.*

We say that s is reachable with probability p in $\mathcal{A}$ if p is equal to $\sup_\sigma\{\sum_{\{\alpha \in Execs^*(\mathcal{A}) | lstate(\alpha)=s\}} \varepsilon_{\sigma,\bar{s}}(\alpha)\}$.

We say that s is reachable with probability q within l steps in $\mathcal{A}$ if $q =$ $\sup_\sigma\{\sum_{\{\alpha \in Execs^*(\mathcal{A}) | |\alpha| \leqslant l \wedge lstate(\alpha)=s\}} \varepsilon_{\sigma,\bar{s}}(\alpha)\}$.

Definition 4.9. *For a family $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$ of probabilistic automata and a family $\{B_k\}_{k\in\mathbb{N}}$ of states, we say that $\{B_k\}_{k\in\mathbb{N}}$ is* polynomially reachable with negligible probability *in $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$ if and only if for each $c \in \mathbb{N}$, each $p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ the probability to reach states of B_k within $p(k)$ steps in $\mathcal{A}_k$ is at most k^{-c}.*

We say alternatively that $\{B_k\}_{k\in\mathbb{N}}$ is negligible *in $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$.*

As we have said at the beginning of this section, polynomially accurate simulations allow us to say when two (families of) automata present quite the same behavior. Such definition considers two families of automata that can be very different but when we prove the existence of a polynomially accurate simulation from the first family to the second one, then we can say that they behave almost in the same way.

Now, consider the two families of automata: it is true that we can consider two families whose automata are completely different but usually the second ones are obtained from the first ones removing the bad cases. For example, the first family models a nonce generator that can return repeated nonces while the second family ensures that each nonce is actually fresh.

One way to avoid to reach bad cases is the following: we identify the states that correspond to the bad cases and then we impose that the probability to reach such states is zero. We can obtain such situation taking as the target measure of each transition tr the measure μ_{tr} conditioned to the set of states that are not bad. Formally,

Definition 4.10. *For a probabilistic automaton $\mathcal{A} = (S, \bar{s}, A, D)$ and a set of states G, define the* G-conditional *of $\mathcal{A}$, denoted by $\mathcal{A}|G$, to be the probabilistic automaton $(S, \bar{s}, A, D')$ where $D' = \{(s, a, \mu)|G \mid (s, a, \mu) \in D, \mu_{tr}(G) > 0\}$ where $(s, a, \mu)|G$ is the transition $(s, a, \mu|G)$.*

One important property of the G-conditional automaton is that each reachable state is in G whenever $\bar{s} \in G$ or, in other words, the probability to reach states in $S \setminus G$ is zero.

Proposition 4.11. *Let $\mathcal{A}$ be a probabilistic automaton and G be a set of states such that $\bar{s} \in G$. Let $\mathcal{A}'$ be the automaton $\mathcal{A}|G$.*

For each state $s \in S$, if s is reachable in $\mathcal{A}'$, then $s \in G$.

Proof. Let s be a state of $\mathcal{A}$ that is reachable. If $s = \bar{s}$, then $s \in G$ by hypothesis. Consider the case $s \neq \bar{s}$: since s is reachable, there exists a finite execution $\alpha \in Execs^*(\mathcal{A}')$ such that $lstate(\alpha) = s$. Since $s \neq \bar{s}$, let β a finite execution and a an action such that $\alpha = \beta as$. By definition of execution, it follows that there exists $(lstate(\beta), a, \mu') \in D'$ such that $\mu'(s) > 0$. By definition of D', it follows that $\mu' = \mu|G$ for a measure μ such that $(lstate(\beta), a, \mu) \in D$. Since $\mu'(s) > 0$, definition of conditional measure implies that $s \in G$. □

The extension of G-conditional automaton to families of automata is now straightforward.

Definition 4.12. *For a family $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$ of probabilistic automata and a family $\{G_k\}_{k\in\mathbb{N}}$ of states, define the* G-conditional *of $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$, denoted by $\{\mathcal{A}_k\}_{k\in\mathbb{N}}|\{G_k\}_{k\in\mathbb{N}}$, to be the family $\{\mathcal{A}_k|G_k\}_{k\in\mathbb{N}}$.*

Given an automaton $\mathcal{A}$, we can easily define a G-conditional $\mathcal{A}'$ of it. The set of states G is arbitrary: if G is S, then it is immediate to see that $\mathcal{A}' = \mathcal{A}$; if G is the empty set, then $\mathcal{A}'$ does not perform transitions, since for each transition tr of $\mathcal{A}$ we have that $\mu_{tr}(\emptyset) = 0$. For intermediate cases, $\mathcal{A}'$ is more or less similar to $\mathcal{A}$: it depends on how different is G with respect to S.

We can choose G as we want: for example, G can be the set of states that enables a particular action, or that are reachable within an odd number of steps, and so on. One case we consider more interesting than the others is when G contains all states that are not considered bad, that is, $G = S \setminus B$ where B is the set of bad states that we want to avoid. Given $\mathcal{A}$ and its bad states B, we can analyze the relation between it and its G-condition

version, that is, the relation between $\mathcal{A}$ and one possible another automaton $\mathcal{G}$ that ensures that states of B are never reached.

Since there are several automata $\mathcal{A}'$ that can ensure that states of B are never reached and that such automata can be very different, it is difficult to establish a general result about the relation between $\mathcal{A}$ and $\mathcal{A}'$. In fact, $\mathcal{A}'$ can be designed in a way that it makes very difficult to compare it with $\mathcal{A}$. On the contrary, comparing $\mathcal{A}$ and $\mathcal{G}$ is quite simple, since they provide the same set of states, the same set of actions and transitions are almost the same. In fact, from the definition of G-conditional of an automaton, we know that target measures are the original ones conditioned to G and that we keep only transitions that leave from states of G.

As we have seen previously, $\mathcal{A}$ and $\mathcal{G}$ are more or less similar and this depends on how G is similar to S. In particular, the main condition is on the probability to reach states that are not in G: we have a polynomially accurate simulation from $\mathcal{A}$ to $\mathcal{G}$ each time the probability to reach states not in G is negligible and vice versa. Formally,

Theorem 4.13 (Conditional Automaton Theorem). *Let $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$ be a family of probabilistic automata and $\{G_k\}_{k\in\mathbb{N}}$ be a family of states such that, for each $k \in \mathbb{N}$, $\bar{s}_k \in G_k$. For each $k \in \mathbb{N}$ let B_k be the set $S_k \setminus G_k$. Then the family of identity relations is a polynomially accurate simulation from $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k\}_{k\in\mathbb{N}}|\{G_k\}_{k\in\mathbb{N}}$ if and only if $\{B_k\}_{k\in\mathbb{N}}$ is negligible in $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$.*

Proof. To simplify the notation, denote by $\mathcal{A}^1_k$ the automaton $\mathcal{A}_k$ and by $\mathcal{A}^2_k$ the automaton $\mathcal{A}_k|G_k$, and hence by $\{\mathcal{A}^2_k\}_{k\in\mathbb{N}}$ the family $\{\mathcal{A}_k\}_{k\in\mathbb{N}}|\{G_k\}_{k\in\mathbb{N}}$.

($\Rightarrow$) Suppose, for the sake of contradiction, that $\{B_k\}_{k\in\mathbb{N}}$ is not negligible in $\{\mathcal{A}^1_k\}_{k\in\mathbb{N}}$. This means that there exist $c \in \mathbb{N}$, $p \in \mathit{Poly}$ such that for each $\bar{k} \in \mathbb{N}$ there exists $k > \bar{k}$ and a measure ν_1 reached after $j \leqslant p(k)$ steps such that $\nu_1(B_k) \geqslant k^{-c}$. By hypothesis, $\bar{s}^1_k \in G_k$ and thus $\bar{s}^1_k \notin B_k$, hence ν_1 is reached performing at least one step.
Since $\{\mathcal{A}^1_k\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}^2_k\}_{k\in\mathbb{N}}$, by Theorem 3.13, it follows that there exists a scheduler σ for $\{\mathcal{A}^2_k\}_{k\in\mathbb{N}}$ that reaches, after j steps, a probability measure ν_2 such that for each $c' > 0$ there exists $\bar{k}' \in \mathbb{N}$ such that for each $k > \bar{k}'$ ν_1 $\mathcal{L}(id_k, jk^{-c'})$ ν_2. In particular, we can choose c' such that $jk^{-c'} < k^{-c}$, that is $c' > -\log_k(k^{-c}/j)$. By definition of $\mathcal{L}(id_k, jk^{-c'})$, it follows that there exist ν'_1, ν''_1, ν'_2, and ν''_2 such that $\nu_1 = (1-jk^{-c'})\nu'_1 + jk^{-c'}\nu''_1$, $\nu_2 = (1-jk^{-c'})\nu'_2 + jk^{-c'}\nu''_2$, and ν'_1 $\mathcal{L}(id_k)$

ν_2'. By Proposition 4.11, we have that $\nu_2(B_k) = 0$ and thus $\nu_1'(B_k) = 0$. This implies that $B_k \subseteq \mathit{Supp}(\nu_1'')$ and that $\nu_1(B_k) = jk^{-c'}\nu_1''(B_k) < k^{-c}\nu_1''(B_k) \leqslant k^{-c}$. This contradicts the hypothesis that $\nu_1(B_k) \geqslant k^{-c}$. Absurd.

($\Leftarrow$) Let $\mathcal{R} = \{id_k\}_{k\in\mathbb{N}}$ be the family of identity relations.

The condition on the start states is trivially true: by definition of conditional automaton, the start state is the same, thus for each $k \in \mathbb{N}$, $\bar{s}_k^1 \; id_k \; \bar{s}_k^2$.

For the step condition, fix $c \in \mathbb{N}$, $p \in \mathit{Poly}$, $\bar{k} \in \mathbb{N}$, $k > \bar{k}$, γ, ν_1, ν_2, υ_1 such that ν_1 is reached within $p(k)$ steps in $\mathcal{A}_k^1$, $\nu_1 \; \mathcal{L}(id_k, \gamma) \; \nu_2$, $\nu_1 \longrightarrow \upsilon_1$. We must find υ_2 such that $\nu_2 \longrightarrow \upsilon_2$ and $\upsilon_1 \; \mathcal{L}(id_k, \gamma + k^{-c}) \; \upsilon_2$.

If $\gamma \geqslant 1 - k^{-c}$, then for each measure υ_2 such that $\nu_2 \longrightarrow \upsilon_2$, υ_2 satisfies $\upsilon_1 \; \mathcal{L}(id_k, \gamma + k^{-c}) \; \upsilon_2$. In fact, $\gamma \geqslant 1 - k^{-c}$ implies that $\gamma + k^{-c} \geqslant 1$ and by definition of $\mathcal{L}(id_k, \gamma + k^{-c})$, it follows that $\upsilon_1 \; \mathcal{L}(id_k, \gamma + k^{-c}) \; \upsilon_2$.

So, suppose that $0 \leqslant \gamma < 1 - k^{-c}$. Let σ_1 be the scheduler for $\mathcal{A}_k^1$ that induces the transition $\nu_1 \longrightarrow \upsilon_1$ and σ_2 be the scheduler for $\mathcal{A}_k^2$ defined as $\sigma_2(s, a, \mu|G_k) = \sigma_1(s, a, \mu)$ when $\mu(G_k) > 0$. Let υ_2 be the measure induced by σ_2 from ν_2. We have that $\upsilon_1 \; \mathcal{L}(id_k, \gamma + k^{-c}) \; \upsilon_2$. In fact, by definition of $\mathcal{L}(id_k, \gamma)$, it follows that there exist ν_1', ν_1'', ν_2', and ν_2'' such that $\nu_1 = (1-\gamma)\nu_1' + \gamma\nu_1''$, $\nu_2 = (1-\gamma)\nu_2' + \gamma\nu_2''$, and $\nu_1' \; \mathcal{L}(id_k) \; \nu_2'$. By definition of $\mathcal{L}(id_k)$, it follows that for each $\alpha \in \mathit{Execs}^*(\mathcal{A}_k^1)$, $\nu_1'(\alpha) = \nu_2'(\alpha)$.

If $\mathit{lstate}(\alpha) \notin G_k$, then $\alpha \notin \mathit{Supp}(\nu_1')$ and thus $\alpha \in \mathit{Supp}(\nu_1'')$. In fact, suppose for the sake of contradiction that $\alpha \in \mathit{Supp}(\nu_1')$. This implies that $\nu_1'(\alpha) > 0$, thus $\nu_2'(\alpha) > 0$ and hence $\mathit{lstate}(\alpha)$ is reachable in $\mathcal{A}_k^2$. This implies, by Proposition 4.11, that $\mathit{lstate}(\alpha) \in G_k$. Absurd. Moreover, for each $\alpha \in \mathit{Supp}(\nu_1')$, each state of α is in G_k.

Let υ_1^1 and υ_2^1 be the two parts of υ_1 and υ_2 that are induced by σ_1 and σ_2 from ν_1' and ν_2', respectively; analogously, let υ_1^2 and υ_2^2 be the two measures induced from ν_1'' and ν_2'', respectively. This implies that $\upsilon_1 = (1-\gamma)\upsilon_1^1 + \gamma\upsilon_1^2$ and $\upsilon_2 = (1-\gamma)\upsilon_2^1 + \gamma\upsilon_2^2$. Now, consider αas such that $\alpha as \in \mathit{Supp}(\upsilon_1^1)$. By definition of $\nu \longrightarrow \upsilon$, it follows that

$$\begin{aligned}\upsilon_1^1(C_{\alpha as}) &= \nu_1'(C_{\alpha as}) \\ &\quad + \nu_1'(\alpha) \sum_{tr \in D_1(a)} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\ &= \nu_1'(C_{\alpha as})\end{aligned}$$

$$
\begin{aligned}
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \in G_k \\ \mu_{tr}(G_k) > 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \notin G_k \\ \mu_{tr}(G_k) > 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ \mu_{tr}(G_k) = 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&= \nu_2'(C_{\alpha a s}) \\
&+ \nu_2'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \in G_k \\ \mu_{tr}(G_k) > 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \notin G_k \\ \mu_{tr}(G_k) > 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ \mu_{tr}(G_k) = 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&= \nu_2'(C_{\alpha a s}) \\
&+ \nu_2'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \in G_k \\ \mu_{tr}(G_k) > 0}} \sigma_2(\alpha)(tr|G_k) \cdot \frac{\mu_{tr}(s)}{\mu_{tr}(G_k)} \cdot \mu_{tr}(G_k) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \notin G_k \\ \mu_{tr}(G_k) > 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ \mu_{tr}(G_k) = 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&= \nu_2'(C_{\alpha a s}) \\
&+ \nu_2'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \in G_k \\ \mu_{tr}(G_k) > 0}} \sigma_2(\alpha)(tr|G_k) \cdot \frac{\mu_{tr}(s)}{\mu_{tr}(G_k)}
\end{aligned}
$$

$$
\begin{aligned}
&- \nu_2'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \in G_k \\ \mu_{tr}(G_k) > 0}} \sigma_2(\alpha)(tr|G_k) \cdot \frac{\mu_{tr}(s)}{\mu_{tr}(G_k)} \cdot (1 - \mu_{tr}(G_k)) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \notin G_k \\ \mu_{tr}(G_k) > 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ \mu_{tr}(G_k) = 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&= \nu_2'(C_{\alpha a s}) \\
&+ \nu_2'(\alpha) \sum_{\substack{tr \in D_2(a) \\ s \in G_k}} \sigma_2(\alpha)(tr) \cdot \mu_{tr}(s) \\
&- \nu_2'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \in G_k \\ \mu_{tr}(G_k) > 0}} \sigma_2(\alpha)(tr|G_k) \cdot \frac{\mu_{tr}(s)}{\mu_{tr}(G_k)} \cdot (1 - \mu_{tr}(G_k)) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \notin G_k \\ \mu_{tr}(G_k) > 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ \mu_{tr}(G_k) = 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&= \upsilon_2^1(C_{\alpha a s}) \\
&- \nu_2'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \in G_k \\ \mu_{tr}(G_k) > 0}} \sigma_2(\alpha)(tr|G_k) \cdot \frac{\mu_{tr}(s)}{\mu_{tr}(G_k)} \cdot (1 - \mu_{tr}(G_k)) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ s \notin G_k \\ \mu_{tr}(G_k) > 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s) \\
&+ \nu_1'(\alpha) \sum_{\substack{tr \in D_1(a) \\ \mu_{tr}(G_k) = 0}} \sigma_1(\alpha)(tr) \cdot \mu_{tr}(s)
\end{aligned}
$$

Now, there are two cases: either $s \in G_k$, or $s \notin G_k$.
If $s \in G_k$, then

– $\nu_1'(\alpha)\sum_{tr\in D_1(a),\mu_{tr}(G_k)=0}\sigma_1(\alpha)(tr)\cdot\mu_{tr}(s)=0$ since for each $tr\in D_1(a)$ such that $\mu_{tr}(G_k)=0$, $\mu_{tr}(s)=0$ (otherwise we have $\mu_{tr}(G_k)\geqslant\mu_{tr}(s)>0$ that contradicts $\mu_{tr}(G_k)=0$), and
– $\nu_1'(\alpha)\sum_{tr\in D_1(a),s\notin G_k,\mu_{tr}(G_k)>0}\sigma_1(\alpha)(tr)\cdot\mu_{tr}(s)=0$ since the summation is empty.

This implies that if $s\in G_k$, then $\upsilon_1^1(C_{\alpha as})=\upsilon_2^1(C_{\alpha as})-\nu_2'(\alpha)\cdot\sum_{tr\in D_1(a),\mu_{tr}(G_k)>0}\sigma_2(\alpha)(tr|G_k)\cdot\dfrac{\mu_{tr}(s)}{\mu_{tr}(G_k)}\cdot(1-\mu_{tr}(G_k))$ and hence $\upsilon_1^1(C_{\alpha as})\leqslant\upsilon_2^1(C_{\alpha as})$.

If $s\notin G_k$, then

– $\upsilon_2^1(C_{\alpha as})=0$ otherwise s can be reached in $\mathcal{A}_k^2$ and thus, by Property 4.11, $s\in G_k$, and
– $\nu_2'(\alpha)\sum_{tr\in D_1(a),s\in G_k,\mu_{tr}(G_k)>0}\sigma_2(\alpha)(tr|G_k)\cdot\dfrac{\mu_{tr}(s)}{\mu_{tr}(G_k)}\cdot(1-\mu_{tr}(G_k))=0$ since the summation is empty.

That is, if $s\notin G_k$, then $\upsilon_1^1(C_{\alpha as})=\nu_1'(\alpha)\sum_{tr\in D_1(a),\mu_{tr}(G_k)>0}\sigma_1(\alpha)(tr)\cdot\mu_{tr}(s)+\nu_1'(\alpha)\sum_{tr\in D_1(a),\mu_{tr}(G_k)=0}\sigma_1(\alpha)(tr)\cdot\mu_{tr}(s)$.

Let $B_k^*\subseteq Supp(\upsilon_1^1)$ be the set of finite executions β such that $lstate(\beta)\in B_k$. Let G_k^* be $Supp(\upsilon_1^1)\setminus B_k^*$ (that is, the set of finite executions β such that $lstate(\beta)\in G_k$). By hypothesis, $\{B_k\}_{k\in\mathbb{N}}=\{S_k^1\setminus G_k\}_{k\in\mathbb{N}}$ is negligible in $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ and this implies that $\upsilon_1^1(B_k^*)<k^{-c}$ and thus $\upsilon_1^1(G_k^*)\geqslant 1-k^{-c}$. Suppose, for the sake of contradiction, that $\upsilon_1^1(B_k^*)\geqslant k^{-c}$. This implies that $\upsilon_1(B_k^*)=(1-\gamma)\upsilon_1^1(B_k^*)>k^{-c}\upsilon_1^1(B_k^*)\geqslant k^{-c}k^{-c}=k^{-2c}$. Since υ_1 is reached within $p(k)+1$ steps and $\upsilon_1(B_k^*)>k^{-2c}$, then $\{B_k\}_{k\in\mathbb{N}}$ is not negligible in $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$. Absurd.

Since $\upsilon_1^1(G_k^*)\geqslant 1-k^{-c}$, we have that $\upsilon_1(G_k^*)=(1-\gamma)\upsilon_1^1(G_k^*)\geqslant(1-\gamma)(1-k^{-c})=(1-(\gamma+k^{-c}))+\gamma k^{-c}$. Let υ_1' be the distribution $\upsilon_1^1|G_k^*$. Define υ_1'' as the measure $(\gamma+k^{-c})(\upsilon_1-(1-(\gamma+k^{-c}))\upsilon_1')$. So, $\upsilon_1=(1-(\gamma+k^{-c}))\upsilon_1'+(\gamma+k^{-c})\upsilon_1''$.

Analogously, define $\upsilon_2'=\upsilon_1'$ and $\upsilon_2''=(\gamma+k^{-c})(\upsilon_2-(1-(\gamma+k^{-c}))\upsilon_2')$ and thus $\upsilon_2=(1-(\gamma+k^{-c}))\upsilon_2'+(\gamma+k^{-c})\upsilon_2''$. Since $\upsilon_2'=\upsilon_1'$, then $\upsilon_1'\ \mathcal{L}(id)\ \upsilon_2'$ and thus conditions of the step of the polynomially accurate simulations are satisfied. □

The above result allows us to conditionate an automaton with respect to several set of states.

Corollary 4.14. *Let $\mathcal{A}_k$ be a family of probabilistic automata and G_k^1, G_k^2 be two families of states such that for each $k\in\mathbb{N}$, $\bar{s}_k\in G_k^1\cap G_k^2$.*

If $\{\mathcal{A}_k\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}_k\}_{k\in\mathbb{N}}|\{G_k^1\}_{k\in\mathbb{N}}$ *and* $\{\mathcal{A}_k\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}_k\}_{k\in\mathbb{N}}|\{G_k^2\}_{k\in\mathbb{N}}$, *then* $\{\mathcal{A}_k\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}_k\}_{k\in\mathbb{N}}|\{G_k^1 \cap G_k^2\}_{k\in\mathbb{N}}$.

Proof. To simplify the notation, denote by $\mathcal{A}_k^1$ and $\mathcal{A}_k^2$ the automata $\mathcal{A}_k|G_k^1$ and $\mathcal{A}_k|G_k^2$, respectively.

By Theorem 4.13, $\{\mathcal{A}_k\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ implies that the set of states $B_k^1 = S_k \setminus G_k^1$ of $\mathcal{A}_k$ is negligible in $\mathcal{A}_k$. Analogously, $B_k^2 = S_k \setminus G_k^2$ is also negligible in $\mathcal{A}_k$. By definition of negligibility of B_k^i in $\mathcal{A}_k$ for $i = 1, 2$, it follows that for each $c' \in \mathbb{N}$ and $p \in \mathit{Poly}$, there exist $\bar{k}_1 \in \mathbb{N}$ and $\bar{k}_2 \in \mathbb{N}$ such that for each $k > \max(\bar{k}_1, \bar{k}_2)$, the probability to reach states of B_k^1 within $p(k)$ steps is less than $k^{-c'}$ as well as the probability to reach states of B_k^2 within $p(k)$ steps is less than $k^{-c'}$. This implies that the probability to reach states of $B_k^1 \cup B_k^2$ is less than $2k^{-c'}$. Since c' is arbitrary, given $c \in \mathbb{N}$, we can choose c' such that there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ $2k^{-c'} \leqslant k^{-c}$ and thus the probability to reach states of $B_k^1 \cup B_k^2$ within $p(k)$ is less than k^{-c}. This implies that $B_k^1 \cup B_k^2$ is negligible in $\mathcal{A}_k$ and thus, by Theorem 4.13, $\{\mathcal{A}_k\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}_k\}_{k\in\mathbb{N}}|\{G_k^1 \cap G_k^2\}_{k\in\mathbb{N}}$. □

5

Polynomially Accurate Simulations: Limitations and Solutions

The definition of polynomially accurate simulations we have provided in the previous chapters allows us to study the security of cryptographic protocols, as we will see in Chapters 7 and 8, but it does not provide one important property: it is not compositional. Moreover, it does not seem to be transitive but actually the Execution Correspondence Theorem allows us to relate executions of two automata whenever we are able to prove a chain of polynomially accurate simulations starting from the first automaton and ending into the second one.

Compositionality is a fundamental property if we want to split an automaton into several subpieces and then to analyze each subcomponent independently from the others. Unfortunately, the polynomial accurate simulation is not compositional: consider the two automata of Figure 5.1.

Let S be the set $\{s_1, \dots s_{2^k}\}$ and U be the uniform measure over S. Suppose that for each $\mu_1 \in Disc(S_k^1)$ that is reached performing an hypertransition from U, if $\mu_1(S) > 2 \cdot 2^{-k}$, then there exists a transition (t, τ, μ_2) such that for each $i = 1, \dots, 2^k$, $\mu_2(t_i) = \mu_1(s_i)/\mu_1(S)$. It is easy to observe that $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ is polynomially accurate simulated by $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$. In fact, given that $a_i \neq a_j$ whenever $i \neq j$, we can consider the relation $\mathcal{R}_k$ such that $(\bar{s}, \bar{t}) \in \mathcal{R}_k$, and for each $i = 1, \dots, 2^k$, $(\bar{s}\tau s_{1,i}, \bar{t}\tau t) \in \mathcal{R}_k$, $(\bar{s}\tau s_{1,i}\tau s_i, \bar{t}\tau t\tau t_i) \in \mathcal{R}_k$, and $(\bar{s}\tau s_{1,i}\tau s_i a_i s_{2,i}, \bar{t}\tau t\tau t_i a_i t_{2,i}) \in \mathcal{R}_k$. $\{\mathcal{R}_k\}_{k\in\mathbb{N}}$ is a polynomially accurate simulation: for each $k \in \mathbb{N}$, $\bar{s}\ \mathcal{R}_k\ \bar{t}$. For the step condition, we distinguish between the case we start from $\delta_{\bar{s}}$ or from another measure.

If $\delta_{\bar{s}} \longrightarrow \upsilon_1$, then by definition of transition, it follows that $\upsilon_1 = \sigma_1(\bar{s})(\perp) + \sigma(\bar{s})(tr_1)U$ where $tr_1 = (\bar{s}, \tau, U)$ and σ_1 is the scheduler that induces $\delta_{\bar{s}} \longrightarrow \upsilon_1$. To match this transition, it is sufficient to choose the

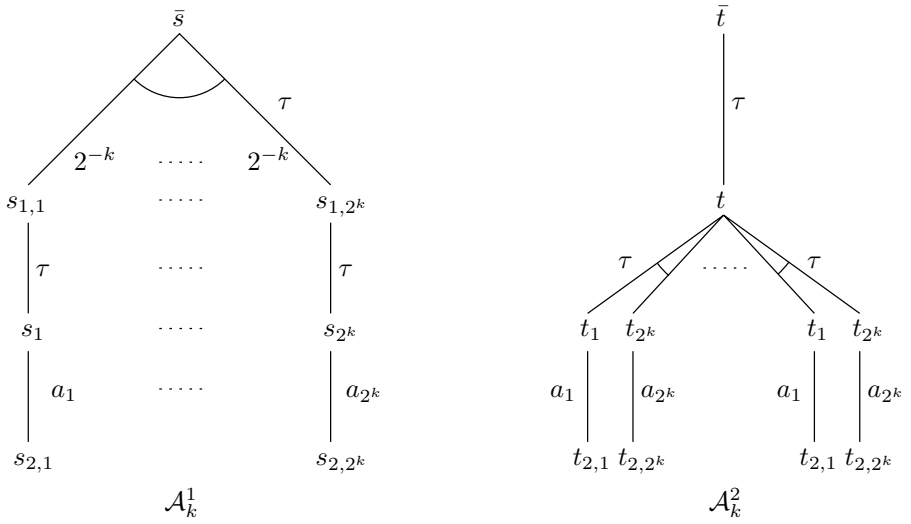

Fig. 5.1. Counter-example for compositionality of polynomially accurate simulation.

scheduler σ_2 defined as: $\sigma_2(\bar{t})(\bot) = \sigma_1(\bar{s})(\bot)$ and $\sigma(\bar{t})(tr_2) = \sigma(\bar{s})(tr_1)$ where $tr_2 = (\bar{t}, \tau, \delta_t)$. Let υ_2 be the measure induced by σ_2 from $\delta_{\bar{t}}$. It is immediate to see that $\upsilon_1 \; \mathcal{L}(\mathcal{R}_k, 0) \; \upsilon_2$ and thus $\upsilon_1 \; \mathcal{L}(\mathcal{R}_k, k^{-c}) \; \upsilon_2$.

If $\nu_1 \longrightarrow \upsilon_1$ with $\nu_1 \neq \delta_{\bar{s}}$, then we have that for each $\alpha \in \mathit{Supp}(\nu_1)$, $|\alpha| \geqslant 1$. This implies that for each $\alpha_1 as$, $\upsilon_1(\alpha_1 as) = \nu_1(\alpha_1 as)\sigma_1(\alpha_1 as)(\bot) + \sum_{tr \in D_k^1(a)} \nu_1\sigma(\alpha_1)(tr)\mu_{tr}(t)$. We can partition the support of υ_1 in several pieces: we have the set of executions α such that σ_1 does not schedule a transition. These executions can be matched exactly by $\mathcal{A}_k^2$ without performing a transition from the executions β such that $\alpha \; \mathcal{R}_k \; \beta$; then we have the set of executions $\bar{s}\tau s_{1,i}\tau s_i a_i s_{2,i}$ that are reached performing the transition $s_i \xrightarrow{a_i} \delta_{s_{2,i}}$. These executions can be matched exactly performing the transition $t_i \xrightarrow{a_i} \delta_{t_{2,i}}$ from the execution $\bar{t}\tau t\tau t_i$ and thus $\bar{s}\tau s_{1,i}\tau s_i a_i s_{2,i} \; \mathcal{R}_k \; \bar{t}\tau t\tau t_i a_i t_{2,i}$; finally, we have the executions $\bar{s}\tau s_{1,i}\tau s_i$ that are obtained performing a transition $s_{1,i} \xrightarrow{\tau} \delta_{s_i}$. Let tr be the normalized hypertransition corresponding to such transitions and let ζ be the normalization factor. Then there are two cases: either $\mu_{tr}(S) > 2 \cdot 2^{-k}$ or $\mu_{tr}(S) \leqslant 2 \cdot 2^{-k}$. In the former case, we have that there exists a transition $tr_2 = t \xrightarrow{\tau} \mu_2$ such that for each $i = 1, \ldots, 2^k$ $\mu_2(t_i) = \mu_{tr}(s_i)/\mu_{tr}(S)$. We can match exactly such hypertransition choosing the transition tr_2 with probability $\zeta\mu_{tr}(S)$. In fact, for each $\bar{s}\tau s_{1,i}\tau s_i$ we have that $\upsilon_1(\bar{s}\tau s_{1,i}\tau s_i) = \nu_1(\bar{s}\tau s_{1,i})\mu_{tr}(s_i)\zeta = \nu_2(\bar{t}\tau t)\mu_2(t_i)\mu_{tr}(S)\zeta = \upsilon_2(\bar{t}\tau t\tau t_i)\zeta\mu_{tr}(S)$. In the latter case, $\mu_{tr}(S) \leqslant 2 \cdot 2^{-k}$ implies that $\upsilon_1(\bar{s}\tau s_{1,i}\tau S) \leqslant 2 \cdot 2^{-k}$ and thus there is

nothing to match since $2 \cdot 2^{-k} < k^{-c}$ for all sufficiently large k and thus such executions can be placed into the piece of the probability measure that represents the accepted error. Summing up, we are able to perform an hypertransition $\nu_2 \longrightarrow \upsilon_2$ such that $\upsilon_1 \; \mathcal{L}(\mathcal{R}_k, k^{-c}) \; \upsilon_2$.

Now, consider the context $\mathcal{C}_k$ such that for each $i = 1, \ldots, 2^k$, it provides a transition $\bar{c} \xrightarrow{b_i} \delta_{c_i}$. Suppose that whenever $i \neq j$, we have that $b_i \neq b_j$ and $c_i \neq c_j$ and that given $c_i \neq c_j$, they enable incompatible transitions (for example, from each c_i it is enabled the transition $c_i \xrightarrow{d_i} \delta_{e_i}$ where for each $i \neq j$, $d_i \neq d_j$).

$\{\mathcal{A}_k^1||\mathcal{C}_k\}_{k\in\mathbb{N}}$ is not polynomially accurate simulated by $\{\mathcal{A}_k^2||\mathcal{C}_k\}_{k\in\mathbb{N}}$. Suppose, for the sake of contradiction, that $\{\mathcal{A}_k^1||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}_k^2||\mathcal{C}_k\}_{k\in\mathbb{N}}$. Then the relation $\mathcal{R}' = \{\mathcal{R}'_k\}_{k\in\mathbb{N}}$ where for each $k \in \mathbb{N}$, $\mathcal{R}'_k = \mathcal{R}_k \times id_k$ is a polynomially accurate simulation from $\{\mathcal{A}_k^1||\mathcal{C}_k\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k^2||\mathcal{C}_k\}_{k\in\mathbb{N}}$.

The condition on start states is trivially satisfied, since $\bar{s} \; \mathcal{R}_k \; \bar{t}$ and thus $(\bar{s}, \bar{c}) \; \mathcal{R}'_k \; (\bar{t}, \bar{c})$.

For the step condition, fix $c \in \mathbb{N}$ and $p \in Poly$. Then there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ and $\gamma = 0$, the step condition holds.

Now, consider the following case: from $(\bar{s}, \bar{c})$, the scheduler σ_1 schedules with probability 1 the transition $\delta_{\bar{s}} \xrightarrow{\tau} U$ while no transition is scheduled for $\mathcal{C}_k$. The reached measure is the uniform measure U_1 on $\{(\bar{s}, \bar{c})\tau(s_{1,i}, \bar{c}) \mid 1 \leqslant i \leqslant 2^k\}$. Such transition can be matched by $\mathcal{A}_k^2||\mathcal{C}_k$ performing only the transition $\delta_{\bar{t}} \xrightarrow{\tau} \delta_t$ for $\mathcal{A}_k^2$, obtaining the measure $\delta_{(\bar{t},\bar{c})\tau(t,\bar{c})}$. So, we have that $U_1 \; \mathcal{L}(\mathcal{R}'_k, 0) \; \delta_{(\bar{t},\bar{c})\tau(t,\bar{c})}$ and thus $U_1 \; \mathcal{L}(\mathcal{R}'_k, k^{-c}) \; \delta_{(\bar{t},\bar{c})\tau(t,\bar{c})}$.

Now, given the execution $(\bar{s}, \bar{c})\tau(s_i, \bar{c})$, σ_1 schedules with probability 1 the transition $\bar{c} \xrightarrow{b_i} \delta_{c_i}$. This means that from the uniform measure on $\{(\bar{s}, \bar{c})\tau(s_{1,i}, \bar{c}) \mid 1 \leqslant i \leqslant 2^k\}$, σ_1 induces an hypertransition that leads to the uniform measure U_1 on $\{(\bar{s}, \bar{c})\tau(s_{1,i}, \bar{c})b_i(s_{1,i}, c_i) \mid 1 \leqslant i \leqslant 2^k\}$. To match such transition in $\mathcal{A}_k^2||\mathcal{C}_k$, starting from the measure $\delta_{(\bar{t},\bar{c})\tau(t,\bar{c})}$ we schedule uniformly the transitions $\bar{c} \xrightarrow{b_i} \delta_{c_i}$ reaching the uniform measure U_2 on the set $\{(\bar{t}, \bar{c})\tau(t, \bar{c})b_i(t, c_i) \mid 1 \leqslant i \leqslant 2^k\}$. This implies that $U_1 \; \mathcal{L}(\mathcal{R}'_k, 0) \; U_2$ and thus $U_1 \; \mathcal{L}(\mathcal{R}'_k, k^{-c}) \; U_2$.

Now, from the uniform measure on $\{(\bar{s}, \bar{c})\tau(s_{1,i}, \bar{c})b_i(s_{1,i}, c_i) \mid 1 \leqslant i \leqslant 2^k\}$, σ_1 schedules for each execution $(\bar{s}, \bar{c})\tau(s_{1,i}, \bar{c})b_i(s_{1,i}, c_i)$ the transition $(s_{1,i}, \tau, \delta_{s_i})$ with probability 1 and no transition for $\mathcal{C}_k$. This implies that from the uniform measure ν_1 on $\{(\bar{s}, \bar{c})\tau(s_{1,i}, \bar{c})b_i(s_{1,i}, c_i) \mid 1 \leqslant i \leqslant 2^k\}$, σ_1 induces an hypertransition that leads to the uniform measure υ_1 on $\{(\bar{s}, \bar{c})\tau(s_{1,i}, \bar{c})b_i(s_{1,i}, c_i)\tau(s_i, c_i) \mid 1 \leqslant i \leqslant 2^k\}$.

To match such transition, we must find an hypertransition from the uniform measure ν_2 on $\{(\bar{t},\bar{c})\tau(t,\bar{c})b_i(t,c_i) \mid 1 \leqslant i \leqslant 2^k\}$ that leads to some measure υ_2 such that $\upsilon_1 \ \mathcal{L}(\mathcal{R}'_k, k^{-c}) \ \upsilon_2$. Such measure does not exist. In fact, by the definition of $\mathcal{A}^2_1$, from the state t we have a transition $t \xrightarrow{\tau} \mu$ where the measure μ is defined as $\mu(t_i) = \mu_1(s_i)/\mu_1(S)$ where μ_1 is the target of an hypertransition from U such that $\mu_1(S) > 2 \cdot 2^{-k}$. By definition of $\mathcal{A}^1_k$, it follows that S contains at least two states: if $S = \{s_i\}$ for some $1 \leqslant i \leqslant 2^k$, then $\mu_1(s) \leqslant 2^{-k}$ since s_i can be reached only from the state $s_{1,i}$ through the transition $tr = s_{1,i} \xrightarrow{\tau} \delta_{s_i}$ and $U(s_{1,i}) = 2^{-k}$, so $\mu_1(\{s_i\}) \leqslant U(s_{1,i})\sigma_1(\bar{s}\tau s_i)(tr)\delta_{s_i}(s_i) \leqslant 2^{-k}$. Moreover, this implies that for each t_i, $\mu(t_i) \leqslant 1/2$. So, we have that for each transition $t \xrightarrow{\tau} \mu$ at least two states are reached with probability greater than 0 and each one with probability at most 1/2. This implies that for each transition $t \xrightarrow{\tau} \mu$ we choose, starting from $(\bar{t},\bar{c})\tau(t,\bar{c})b_i(t,c_i)$ we obtain the executions $(\bar{t},\bar{c})\tau(t,\bar{c})b_i(t,c_i)\tau(t_j,c_i)$ with $t_j \in \mathit{Supp}(\mu)$. By definition of $\mathcal{R}'_k$, only $(\bar{t},\bar{c})\tau(t,\bar{c})b_i(t,c_i)\tau(t_i,c_i)$ can be related to $(\bar{s},\bar{c})\tau(s_{1,i},\bar{c})b_i(s_{1,i},c_i)\tau(s_i,c_i)$ while $(\bar{s},\bar{c})\tau(s_{1,i},\bar{c})b_i(s_{1,i},c_i)\tau(s_i,c_i)\neg\ \mathcal{R}'_k\ (\bar{t},\bar{c})\tau(t,\bar{c})b_i(t,c_i)\tau(t_j,c_i)$ for all $j \neq i$. Since this happens for all executions, if we fix the execution α_1 to be $(\bar{s},\bar{c})\tau(s_{1,i},\bar{c})b_i(s_{1,i},c_i)$, then we have that for each scheduler σ_2 for $\mathcal{A}^2_k||\mathcal{C}_k$,

$$
\begin{aligned}
&\sum\{\upsilon_2(\alpha_2\tau(t_i,c_i)) \mid \alpha_1\tau(s_i,c_i)\ \mathcal{R}'_k\ \alpha_2\tau(t_i,c_i)\}\\
&= \sum\{\nu_2(\alpha_2)\sigma_2(\alpha_2)(tr)\mu_{tr}(t_i) \mid \alpha_1\tau(s_i,c_i)\ \mathcal{R}'_k\ \alpha_2\tau(t_i,c_i)\}\\
&\leqslant \sum\{\nu_2(\alpha_2)\sigma_2(\alpha_2)(tr)\frac{1}{2} \mid \alpha_1\tau(s_i,c_i)\ \mathcal{R}'_k\ \alpha_2\tau(t_i,c_i)\}\\
&= \frac{1}{2}\sum\{\nu_2(\alpha_2)\sigma_2(\alpha_2)(tr) \mid \alpha_1\tau(s_i,c_i)\ \mathcal{R}'_k\ \alpha_2\tau(t_i,c_i)\}\\
&= \frac{1}{2}\sum\{\nu_2(\alpha_2) \mid \alpha_1\tau(s_i,c_i)\ \mathcal{R}'_k\ \alpha_2\tau(t_i,c_i)\}\\
&= \frac{1}{2}\cdot 2^{-k}
\end{aligned}
$$

Thus, we are not able to match the transition $\nu_1 \longrightarrow \upsilon_1$ with an error at most k^{-c} since $\sum_{\alpha_1\tau(s_i,c_i)}\{\upsilon_2(\alpha_2\tau(t_i,c_i)) \mid \alpha_1\tau(s_i,c_i)\ \mathcal{R}'_k\ \alpha_2\tau(t_i,c_i)\} = \sum_{\alpha_1\tau(s_i,c_i)}\{\frac{1}{2}\cdot 2^{-k} \mid \alpha_1\tau(s_i,c_i)\ \mathcal{R}'_k\ \alpha_2\tau(t_i,c_i)\} = \frac{1}{2}$ while we have that $\sum_{\alpha_1\tau(s_i,c_i)}\{\upsilon_1(\alpha_1\tau(s_i,c_i))\} = 1$. □

5.1 The State Polynomially Accurate Simulation

Since $\mathcal{A}_k^2||\mathcal{C}_k$ is not able to match the transition performed by $\mathcal{A}_k^1||\mathcal{C}_k$, we have that $\{\mathcal{A}_k^1||\mathcal{C}_k\}_{k\in\mathbb{N}}$ can not be polynomially accurate simulated by $\{\mathcal{A}_k^2||\mathcal{C}_k\}_{k\in\mathbb{N}}$.

Since the definition of polynomially accurate simulation does not ensure the compositionality, we modify it to achieve such property preserving the results of the polynomially accurate simulation: the Execution Correspondence Theorem and the Conditional Automaton Theorem.

The first thing we consider is the notion of approximated lifting: the definition 3.4 states that $\rho_1\ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_2$ if there exist ρ_1', ρ_1'', ρ_2', and ρ_2'' such that $\rho_1 = (1-\varepsilon)\rho_1' + \varepsilon\rho_1''$, $\rho_2 = (1-\varepsilon)\rho_2' + \varepsilon\rho_2''$ and $\rho_1'\ \mathcal{L}(\mathcal{R})\ \rho_2'$. Now we want to define an equivalent notion of approximated lifting that is not based on the decomposition of the measures, but only on a different notion of weighting function:

Definition 5.1. *Let $\mathcal{R}$ be a relation from X to Y and let ρ_x, ρ_y be two measures in $Disc(X)$ and $Disc(Y)$, respectively. Given $\varepsilon \in \mathbb{R}^{\geqslant 0}$, we say that $\rho_x\ \mathcal{L}_w(\mathcal{R},\varepsilon)\ \rho_y$ if and only if $\varepsilon \geqslant 1$ or, when $\varepsilon \in [0,1)$, there exists an* ε-weighting function $w_\varepsilon\colon X\times Y \to [0,1]$ *such that*

1. *$Supp(w_\varepsilon) \subseteq \mathcal{R}$, that is for each $u\in X$, $v\in Y$, $w_\varepsilon(u,v) > 0 \implies u\ \mathcal{R}\ v$,*
2. *$w_\varepsilon(u,Y) \leqslant \rho_x(u)$, that is for each $u\in X$, $\sum_{v\in Y} w_\varepsilon(u,v) \leqslant \rho_x(u)$,*
3. *$w_\varepsilon(X,v) \leqslant \rho_y(v)$, that is for each $v\in Y$, $\sum_{u\in X} w_\varepsilon(u,v) \leqslant \rho_y(v)$, and*
4. *$w_\varepsilon(X,Y) \geqslant 1-\varepsilon$, that is $\sum_{u\in X, v\in Y} w_\varepsilon(u,v) \geqslant 1-\varepsilon$.*

Note that conditions 2 and 3 impose that $\sum_{u\in X,v\in Y} w_\varepsilon(u,v) \leqslant 1$.

It is interesting to observe that so far we have not introduced anything new since the two notions of approximated lifting coincide.

Before proving the equivalence between the two notions of approximated lifting, we want to prove that for each ε-weighting function w_ε, there exists another ε-weighting function w_ε' such that $w_\varepsilon'(X,Y) = 1-\varepsilon$.

Lemma 5.2. *Let $\mathcal{R}$ be a relation from X to Y, $\varepsilon \in \mathbb{R}^{\geqslant 0}$ and $\rho_x \in Disc(X)$, $\rho_y \in Disc(Y)$ be two probability measures such that $\rho_x\ \mathcal{L}_w(\mathcal{R},\varepsilon)\ \rho_y$. Let $w_\varepsilon\colon X\times Y \to [0,1]$ be an ε-weighting function that witnesses $\rho_x\ \mathcal{L}_w(\mathcal{R},\varepsilon)\ \rho_y$.*

Then there exists another ε-weighting function $w_\varepsilon'\colon X\times Y\to[0,1]$ that witnesses $\rho_x\ \mathcal{L}_w(\mathcal{R},\varepsilon)\ \rho_y$ and $w_\varepsilon'(X,Y) = 1-\varepsilon$.

Proof. Given w_ε, let ζ be $w_\varepsilon(X,Y)$ and γ be the value $\dfrac{1-\varepsilon}{\zeta}$. If $\zeta = 0$, then by condition 4 follows that $\varepsilon = 1$ and there is nothing to prove,

since we already have that $w_\varepsilon(X,Y) = 1-\varepsilon$. So suppose that $\zeta > 0$. This implies that $\gamma \leqslant 1$ since by hypothesis $\zeta = w_\varepsilon(X,Y) \geqslant 1-\varepsilon$. If $\gamma = 1$, then it follows that $\zeta = 1-\varepsilon$ and again there is nothing to prove, since $w_\varepsilon(X,Y) = 1-\varepsilon$. Suppose that $0 < \gamma < 1$ and define w'_ε as $w'_\varepsilon(u,v) = \gamma w_\varepsilon(u,v)$ for each pair (u,v). The condition 1 is trivially verified since $w'_\varepsilon(u,v) > 0 \implies \gamma w_\varepsilon(u,v) > 0 \implies w_\varepsilon(u,v) > 0 \implies u \mathcal{R} v$; conditions 2 and 3 are also satisfied: $\sum_{v\in Y} w'_\varepsilon(u,v) = \sum_{v\in Y} \gamma w_\varepsilon(u,v) = \gamma \sum_{v\in Y} w_\varepsilon(u,v) \leqslant \gamma\rho_x(u) \leqslant \rho_x(u)$ and $\sum_{u\in X} w'_\varepsilon(u,v) = \sum_{u\in X} \gamma w_\varepsilon(u,v) = \gamma \sum_{u\in X} w_\varepsilon(u,v) \leqslant \gamma\rho_y(v) \leqslant \rho_y(v)$; for the condition 4, $\sum_{u\in X, v\in Y} w'_\varepsilon(u,v) = \sum_{u\in X,v\in Y} \gamma w_\varepsilon(u,v) = \gamma \sum_{u\in X,v\in Y} w_\varepsilon(u,v) = \gamma\zeta = \dfrac{1-\varepsilon}{\zeta}\zeta = 1-\varepsilon$.

Since $\sum_{u\in X,v\in Y} w_\varepsilon(u,v) = 1-\varepsilon$ implies $\sum_{u\in X,v\in Y} w_\varepsilon(u,v) \geqslant 1-\varepsilon$, then the two definitions of w_ε are equivalent. □

Proposition 5.3. *Let $\mathcal{R}$ be a relation from X to Y and let $\rho_x \in Disc(X)$, $\rho_y \in Disc(Y)$ be two measures.*

For each $\varepsilon \in \mathbb{R}^{\geqslant 0}$, $\rho_x\ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_y$ if and only if $\rho_x\ \mathcal{L}_w(\mathcal{R},\varepsilon)\ \rho_y$.

Proof. ($\Rightarrow$) Suppose that $\rho_x\ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_y$. If $\varepsilon \geqslant 1$, then there is nothing to prove since by definition of $\mathcal{L}_w(\mathcal{R},\varepsilon)$, we have that $\rho_x\ \mathcal{L}_w(\mathcal{R},\varepsilon)\ \rho_y$.

So, suppose that $\varepsilon \in [0,1)$. By definition of $\mathcal{L}(\mathcal{R},\varepsilon)$, it follows that there exist ρ'_x, ρ''_x, ρ'_y, and ρ''_y such that $\rho_x = (1-\varepsilon)\rho'_x + \varepsilon\rho''_x$, $\rho_y = (1-\varepsilon)\rho'_y + \varepsilon\rho''_y$, and $\rho'_x\ \mathcal{L}(\mathcal{R})\ \rho'_y$. This implies that there exists a weighting function $w\colon X\times Y \to [0,1]$ such that $w(x,y) > 0$ implies $x\ \mathcal{R}\ y$, $\sum_{u\in X} w(u,v) = \rho_y(v)$, and $\sum_{v\in Y} w(u,v) = \rho_x(u)$. Define $w'\colon X\times Y \to [0,1]$ as: for each pair (u,v), $w'(u,v) = (1-\varepsilon)w(u,v)$.

w' is an ε-weighting function from ρ_x to ρ_y. In fact,

– condition 1 is trivially satisfied:

$$\begin{aligned} w'(u,v) > 0 &\implies (1-\varepsilon)w(u,v) > 0 \\ &\implies w(u,v) > 0 \\ &\implies u\ \mathcal{R}\ v \end{aligned}$$

– condition 2 is verified:

$$\begin{aligned} \sum_{u\in X} w'(u,v) &= \sum_{u\in X} (1-\varepsilon)w(u,v) \\ &= (1-\varepsilon)\sum_{u\in X} w(u,v) \end{aligned}$$

$$\begin{aligned} &= (1-\varepsilon)\rho_y(v) \\ &\leqslant \rho_y(v) \end{aligned}$$

– also condition 3 is satisfied:

$$\begin{aligned} \sum_{v\in Y} w'(u,v) &= \sum_{v\in Y}(1-\varepsilon)w(u,v) \\ &= (1-\varepsilon)\sum_{v\in Y} w(u,v) \\ &= (1-\varepsilon)\rho_x(u) \\ &\leqslant \rho_x(u) \end{aligned}$$

– and finally, condition 4 is verified:

$$\begin{aligned} \sum_{\substack{u\in X\\ v\in Y}} w'(u,v) &= \sum_{\substack{u\in X\\ v\in Y}}(1-\varepsilon)w(u,v) \\ &= (1-\varepsilon)\sum_{\substack{u\in X\\ v\in Y}} w(u,v) \\ &= (1-\varepsilon)\sum_{u\in X} \rho_x(u) \\ &= (1-\varepsilon)1 \\ &= 1-\varepsilon \end{aligned}$$

($\Leftarrow$) Suppose that $\rho_x \ \mathcal{L}_w(\mathcal{R},\varepsilon)\ \rho_y$. If $\varepsilon \geqslant 1$, then there is nothing to prove since by definition of $\mathcal{L}(\mathcal{R},\varepsilon)$, we have that $\rho_x \ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_y$. So, suppose that $\varepsilon \in (0,1)$. By definition of $\mathcal{L}_w(\mathcal{R},\varepsilon)$, it follows that there exists an ε-weighting function w_ε such that

1. $w_\varepsilon(u,v) > 0 \implies u \,\mathcal{R}\, v$,
2. $\sum_{u\in X} w_\varepsilon(u,v) \leqslant \rho_y(v)$,
3. $\sum_{v\in Y} w_\varepsilon(u,v) \leqslant \rho_x(u)$, and
4. $\sum_{u\in X, v\in Y} w_\varepsilon(u,v) = 1-\varepsilon$.

If $\varepsilon = 0$, then condition 4 implies that $\sum_{u\in X} w_\varepsilon(u,v) = \rho_y(v)$ and $\sum_{v\in Y} w_\varepsilon(u,v) = \rho_x(u)$. This means that w_ε is an ordinary weighting function and thus $\rho_x \ \mathcal{L}(\mathcal{R})\ \rho_y$. Since $\rho_x = (1-\varepsilon)\rho_x + \varepsilon\rho_x$ and $\rho_y = (1-\varepsilon)\rho_y + \varepsilon\rho_y$, we have that $\rho_x \ \mathcal{L}(\mathcal{R},\varepsilon)\ \rho_y$.

Now, suppose that $\varepsilon \in (0,1)$. Let μ_x and μ_y be the two marginals of w_ε (that is, for each $u \in X$, $\mu_x(u) = \sum_{v\in Y} w_\varepsilon(u,v)$ and for each

$v \in Y$, $\mu_y(v) = \sum_{u \in X} w_\varepsilon(u,v)$) and define $\rho'_x = \dfrac{\mu_x}{1-\varepsilon}$, $\rho''_x = \dfrac{\rho_x - \mu_x}{\varepsilon}$, $\rho'_y = \dfrac{\mu_y}{1-\varepsilon}$, and $\rho''_y = \dfrac{\rho_y - \mu_y}{\varepsilon}$.

It follows that $(1-\varepsilon)\rho'_x + \varepsilon\rho''_x = (1-\varepsilon)\dfrac{\mu_x}{1-\varepsilon} + \varepsilon\dfrac{\rho_x - \mu_x}{\varepsilon} = \mu_x + \rho_x - \mu_x = \rho_x$ and $(1-\varepsilon)\rho'_y + \varepsilon\rho''_y = (1-\varepsilon)\dfrac{\mu_y}{1-\varepsilon} + \varepsilon\dfrac{\rho_y - \mu_y}{\varepsilon} = \mu_y + \rho_y - \mu_y = \rho_y$.

To prove $\rho_x \; \mathcal{L}(\mathcal{R}, \varepsilon) \; \rho_y$, we must verify that $\rho'_x \; \mathcal{L}(\mathcal{R}) \; \rho'_y$. So, let $w' \colon X \times Y \to [0,1]$ be defined as: $w(u,v) = \dfrac{w_\varepsilon(u,v)}{1-\varepsilon}$. We must verify that w' is a weighting function from ρ'_x to ρ'_y:

– condition 1 is trivially verified:

$$\begin{aligned} w'(u,v) > 0 &\implies \frac{w_\varepsilon(u,v)}{1-\varepsilon} > 0 \\ &\implies w_\varepsilon(u,v) > 0 \\ &\implies u \; \mathcal{R} \; v \end{aligned}$$

– condition 2 is verified:

$$\begin{aligned} \sum_{u \in X} w'(u,v) &= \sum_{u \in X} \frac{w_\varepsilon(u,v)}{1-\varepsilon} \\ &= \frac{\sum_{u \in X} w_\varepsilon(u,v)}{1-\varepsilon} \\ &= \frac{\mu_y(v)}{1-\varepsilon} \\ &= \rho'_y(v) \end{aligned}$$

– also condition 3 is satisfied:

$$\begin{aligned} \sum_{v \in Y} w'(u,v) &= \sum_{v \in Y} \frac{w_\varepsilon(u,v)}{1-\varepsilon} \\ &= \frac{\sum_{v \in Y} w_\varepsilon(u,v)}{1-\varepsilon} \\ &= \frac{\mu_x(u)}{1-\varepsilon} \\ &= \rho'_x(u) \end{aligned}$$

□

Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two probabilistic automata, $\zeta \in \mathbb{R}^{\geqslant 0}$ and $\mathcal{R}$ be a relation from S_1 to S_2. Given two states s_1 and s_2, we say that $s_1 \; \mathcal{R}(\zeta) \; s_2$ if $s_1 \; \mathcal{R} \; s_2$

and for each transition $(s_1, a, \mu_1) \in D_1$, there exists a combined transition (s_2, a, μ_2) such that $\mu_1 \; \mathcal{L}_w(\mathcal{R}, \zeta) \; \mu_2$.

Proposition 5.4. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two automata and let $\mathcal{R}$ be a relation from S_1 to S_2. Let $\varepsilon, \gamma \in \mathbb{R}^{\geqslant 0}$ be two values and μ_1, μ_2 be two distributions in $Disc(S_1)$ and $Disc(S_2)$, respectively, such that there exists an ε-weighting function w_ε that justifies $\mu_1 \; \mathcal{L}_w(\mathcal{R}, \varepsilon) \; \mu_2$.*

If $\sum\{w_\varepsilon(s_1, s_2) \mid s_1 \neg \mathcal{R}(\gamma) \; s_2\} < \gamma$, and $\mu_1 \longrightarrow \varphi_1$ then there exists φ_2 such that $\varphi_1 \; \mathcal{L}_w(\mathcal{R}, \varepsilon + \gamma) \; \varphi_2$ and $\mu_2 \longrightarrow \varphi_2$.

Proof. Given two states s_1 and s_2 such that $s_1 \; \mathcal{R}(\gamma) \; s_2$ and a transition $tr_1 = (s_1, a, \rho_1)$, denote by $m(s_2, tr_1)$ the combined transition (s_2, a, ρ_2) such that $\rho_1 \; \mathcal{L}_w(\mathcal{R}, \gamma) \; \rho_2$ and by $w_\gamma^{s_2, tr_1}$ the ε-weighting function that justifies $\rho_1 \; \mathcal{L}_w(\mathcal{R}, \gamma) \; \rho_2$.

By definition of $\mu_1 \longrightarrow \varphi_1$, it follows that there exists a family of probabilities $\{p_{tr_1}\}_{tr_1 \in D_1}$ such that for each state $s_1 \in Supp(\mu_1)$, $\sum_{tr_1 \in D_1, src(tr_1) = s_1} p_{tr_1} = 1$ and $\varphi_1 = \sum_{tr_1 \in D_1} p_{tr_1} \mu_1(src(tr_1)) \mu_{tr_1}$. Let $\{p_{m(s_2, tr_1)}\}_{s_2 \in S_2, tr_1 \in D_1}$ be the family of values for each for each $s_2 \in S_2$, $p_{m(s_2, tr_1)} = p_{tr_1}$. Let φ_2 be defined as: for each $s_2' \in S_2$, $\varphi_2(s_2') = \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2, tr_1)} w_\varepsilon(src(tr_1), s_2) \sum_{s_1' \in Supp(\mu_{tr_1})} w_\gamma^{s_2, tr_1}(s_1', s_2')$.

Now, we need to verify that $\varphi_1 \; \mathcal{L}_w(\mathcal{R}, \varepsilon + \gamma) \; \varphi_2$. To do this, consider the function $w_{\varepsilon+\gamma}$ defined as:

$$w_{\varepsilon+\gamma}(s_1', s_2') = \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2, tr_1)} \cdot w_\varepsilon(src(tr_1), s_2) w_\gamma^{s_2, tr_1}(s_1', s_2')$$

$w_{\varepsilon+\gamma}$ is an ε-weighting function from φ_1 to φ_2:

- $w_{\varepsilon+\gamma}(s_1', s_2') > 0$ implies that there exists $tr_1 \in D_1$, $s_2 \in Supp(\mu_2)$ such that $p_{m(s_2, tr_1)} w_\varepsilon(src(tr_1), s_2) w_\gamma^{s_2, tr_1}(s_1', s_2') > 0$ and this means that $w_\gamma^{s_2, tr_1}(s_1', s_2') > 0$. By definition of $w_\gamma^{s_2, tr_1}$, it follows that $s_1' \; \mathcal{R} \; s_2'$;
- for each $s_2' \in S_2$,

$$\begin{aligned}
&\sum_{s_1' \in S_1} w_{\varepsilon+\gamma}(s_1', s_2') \\
&= \sum_{s_1' \in S_1} \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2, tr_1)} w_\varepsilon(src(tr_1), s_2) w_\gamma^{s_2, tr_1}(s_1', s_2') \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2, tr_1)} w_\varepsilon(src(tr_1), s_2) \sum_{s_1' \in S_1} w_\gamma^{s_2, tr_1}(s_1', s_2') \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2, tr_1)} w_\varepsilon(src(tr_1), s_2) \sum_{s_1' \in Supp(\mu_{tr_1})} w_\gamma^{s_2, tr_1}(s_1', s_2')
\end{aligned}$$

$$
\begin{aligned}
&= \varphi_2(s'_2) \\
&\leqslant \varphi_2(s'_2)
\end{aligned}
$$

– for each $s'_1 \in S_1$,

$$
\begin{aligned}
&\sum_{s'_2 \in S_2} w_{\varepsilon+\gamma}(s'_1, s'_2) \\
&= \sum_{s'_2 \in S_2} \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) w_\gamma^{s_2,tr_1}(s'_1, s'_2) \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) \sum_{s'_2 \in S_2} w_\gamma^{s_2,tr_1}(s'_1, s'_2) \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) \mu_{tr}(s'_1) \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{tr_1} w_\varepsilon(src(tr_1), s_2) \mu_{tr}(s'_1) \\
&= \sum_{tr_1 \in D_1} p_{tr_1} \mu_{tr}(s'_1) \sum_{s_2 \in Supp(\mu_2)} w_\varepsilon(src(tr_1), s_2) \\
&\leqslant \sum_{tr_1 \in D_1} p_{tr_1} \mu_{tr}(s'_1) \mu_1(src(tr_1)) \\
&\leqslant \varphi_1(s'_1)
\end{aligned}
$$

– finally,

$$
\begin{aligned}
&\sum_{s'_1 \in S_2} \sum_{s'_2 \in S_2} w_{\varepsilon+\gamma}(s'_1, s'_2) \\
&= \sum_{s'_1 \in S_2} \sum_{s'_2 \in S_2} \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) w_\gamma^{s_2,tr_1}(s'_1, s'_2) \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) \sum_{s'_1 \in S_2} \sum_{s'_2 \in S_2} w_\gamma^{s_2,tr_1}(s'_1, s'_2) \\
&\geqslant \sum_{s'_1 \in S_2} \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2)(1-\gamma) \\
&= (1-\gamma) \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{tr_1} w_\varepsilon(src(tr_1), s_2) \\
&= (1-\gamma) \sum_{tr_1 \in D_1} p_{tr_1} \sum_{s_2 \in Supp(\mu_2)} w_\varepsilon(src(tr_1), s_2) \\
&\geqslant (1-\gamma) \sum_{tr_1 \in D_1} p_{tr_1}(1-\varepsilon)
\end{aligned}
$$

$$
\begin{aligned}
&= (1-\gamma)(1-\varepsilon) \sum_{tr_1 \in D_1} p_{tr_1} \\
&= (1-\gamma)(1-\varepsilon) \\
&= (1-\gamma-\varepsilon) + \gamma\varepsilon \\
&\geqslant 1-\gamma-\varepsilon
\end{aligned}
$$

This implies that $w_{\varepsilon+\gamma}$ is an ε-weighting function from φ_1 to φ_2 and thus $\varphi_1 \; \mathcal{L}_w(\mathcal{R}, \varepsilon+\gamma) \; \varphi_2$. □

Now we are able to define the new notion of polynomially accurate simulation:

Definition 5.5. *Let $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ and $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ be two families of probabilistic automata; let $\mathcal{R}= \{\mathcal{R}_k\}_{k\in\mathbb{N}}$ be a family of relations such that, for each $k \in \mathbb{N}$, $\mathcal{R}_k$ is a relation from S_k^1 to S_k^2; let Poly be the set of positive polynomials over $\mathbb{N}$.*

We say that $\mathcal{R}$ is a state polynomially accurate simulation *from $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ if*

1. *for each k, it holds that $\bar{s}_k^1 \; \mathcal{R}_k \; \bar{s}_k^2$;*
2. *for each $c \in \mathbb{N}$ and $p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$, for all probability measures μ_1 and μ_2 and for each $\gamma \geqslant 0$, if μ_1 is reached within $p(k)$ steps in $\mathcal{A}_k^1$, then there exists an ε-weighting function w_γ for $\mu_1 \; \mathcal{L}_w(\mathcal{R}_k, \gamma) \; \mu_2$ such that $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg \; \mathcal{R}_k(k^{-c}) \; s_2\} < k^{-c}$.*

We write $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ if there exists a state polynomially accurate simulation $\mathcal{R}$ from $\{\mathcal{A}_k^1\}_{k\in K}$ to $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.

It is quite easy to see that the state polynomially accurate simulation relation is implied by the ordinary simulation of probabilistic automata:

Proposition 5.6. *Let $\mathcal{A}_n^1$ and $\mathcal{A}_n^2$ be two automata parameterized on $n \in \mathbb{N}$.*

If for each $n \in \mathbb{N}$ $\mathcal{A}_n^1 \preceq \mathcal{A}_n^2$, then $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.

Proof. For each $n \in \mathbb{N}$, let $\mathcal{R}_n$ be the simulation that justifies $\mathcal{A}_n^1 \preceq \mathcal{A}_n^2$. $\mathcal{R}= \{\mathcal{R}_k\}_{k\in\mathbb{N}}$ is a state polynomially accurate simulation from $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.

The condition on start states is trivially true, since by definition of ordinary simulation, it follows that for each $k \in \mathbb{N}$, $\bar{s}_k^1 \; \mathcal{R}_k \; \bar{s}_k^2$.

Suppose, for the sake of contradiction, that the step condition does not hold. This means that there exists $c \in \mathbb{N}$ and $p \in Poly$ such that for each

$\bar{k} \in \mathbb{N}$ there exists $k > \bar{k}$ such that there exist μ_1, μ_2, and $\gamma \geqslant 0$ such that μ_1 is reached within $p(k)$ steps in $\mathcal{A}_k^1$ and for each ε-weighting function w_γ for $\mu_1\ \mathcal{L}_w(\mathcal{R}_k, \gamma)\ \mu_2$ we have that $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg\ \mathcal{R}_k(k^{-c})\ s_2\} \geqslant k^{-c}$.

$s_1 \neg\ \mathcal{R}_k(k^{-c})\ s_2$ implies that either $s_1 \neg\ \mathcal{R}_k\ s_2$ or $s_1\ \mathcal{R}_k\ s_2$ and there exists $tr_1 = (s_1, a, \rho_1) \in D_k^1$ such that for each combined transition $tr_2 = (s_2, a, \rho_2)$, $\rho_1 \neg\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \rho_2$. In the former case, $s_1 \neg\ \mathcal{R}_k\ s_2$ implies that $w_\gamma(s_1, s_2) = 0$. Otherwise, if $w_\gamma(s_1, s_2) > 0$, then by property 1 of the definition of ε-weighting function, we have that $s_1\ \mathcal{R}_k\ s_2$. So, $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg\ \mathcal{R}_k(k^{-c})\ s_2\} \geqslant k^{-c}$ is due to pair of states (s_1, s_2) such that $s_1\ \mathcal{R}_k\ s_2$ and there exists $tr_1 = (s_1, a, \rho_1) \in D_k^1$ such that for each combined transition $tr_2 = (s_2, a, \rho_2)$, $\rho_1 \neg\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \rho_2$. Take a pair of such states, say (s_1, s_2). Since $\mathcal{A}_k^1 \preceq \mathcal{A}_k^2$, $s_1\ \mathcal{R}_k\ s_2$ and $s_1 \xrightarrow{a} \rho_1$ implies that there exists a transition (s_2, a, ρ_2) such that $\rho_1\ \mathcal{L}(\mathcal{R}_k)\ \rho_2$. By definition of combined transition, it follows that (s_2, a, ρ_2) is also a combined transition and by Property 3.5(1) we have that $\rho_1\ \mathcal{L}(\mathcal{R}_k, 0)\ \rho_2$. This implies, by Property 3.5(2), that $\rho_1\ \mathcal{L}(\mathcal{R}_k, k^{-c})\ \rho_2$ and thus, by Proposition 5.3, that $\rho_1\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \rho_2$. Summing up, we have that for each pair of states (s_1, s_2) such that $w_\gamma(s_1, s_2) > 0$, it holds that $s_1\ \mathcal{R}_k(k^{-c})\ s_2$. Thus, $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg\ \mathcal{R}_k(k^{-c})\ s_2\} = 0 < k^{-c}$. Absurd. □

The two definitions of polynomially accurate simulation are quite different: we consider measures over executions for $\lesssim$ while we base the definition of $\lesssim_s$ on measures over states; anyway, we are able to prove that the state polynomially accurate simulation relation implies the polynomially accurate simulation relation:

Proposition 5.7. *Let $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ and $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ be two families of automata. If $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$, then $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.*

Proof. Let $\{\mathcal{R}_k\}_{k\in\mathbb{N}}$ be a state polynomially accurate simulation that justifies $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$. Define $\{\mathcal{R}'_k\}_{k\in\mathbb{N}}$ as the family of relations $\mathcal{R}'_k$ such that each $\mathcal{R}'_k$ is a relation from $Execs^*(\mathcal{A}_k^1)$ to $Execs^*(\mathcal{A}_k^2)$ and given two executions $\alpha_1 = s_0 a_1 s_1 \dots a_m s_m$ and $\alpha_2 = t_0 b_1 t_1 \dots b_n t_n$, $\alpha_1\ \mathcal{R}'_k\ \alpha_2$ if

- $m = n$,
- $s_0\ \mathcal{R}_k\ t_0$, and
- for each $0 < i \leqslant n$, $a_i = b_i$ and $s_i\ \mathcal{R}_k\ t_i$.

$\{\mathcal{R}'_k\}_{k\in\mathbb{N}}$ is a polynomially accurate simulation from $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.

Condition on start states is trivially true: by definition of $\lesssim_s$, it follows that for each $k \in \mathbb{N}$, $\bar{s}_k^1 \ \mathcal{R}_k \ \bar{s}_k^2$ and thus, by definition of $\mathcal{R}'_k$, we have that $\bar{s}_k^1 \ \mathcal{R}'_k \ \bar{s}_k^2$.

Suppose, for the sake of contradiction, that the step condition does not hold. This implies that there exist $c \in \mathbb{N}$, $p \in \mathit{Poly}$ such that for each $\bar{k} \in \mathbb{N}$, there exists $k > \bar{k}$, ν_1, ν_2 and $\gamma \geqslant 0$ such that ν_1 is reached within $p(k)$ steps in $\mathcal{A}_k^1$, $\nu_1 \ \mathcal{L}(\mathcal{R}'_k, \gamma) \ \nu_2$, and $\nu_1 \longrightarrow \upsilon_1$ and there does not exist υ_2 such that $\nu_2 \longrightarrow \upsilon_2$ and $\upsilon_1 \ \mathcal{L}(\mathcal{R}'_k, \gamma + k^{-c}) \ \upsilon_2$.

Now, consider ν_1 and ν_2 and the measures μ_1 and μ_2 defined as: $\mu_i(s) = \sum\{\nu_i(\alpha) \mid \mathit{lstate}(\alpha) = s\}$. Since $\nu_1 \ \mathcal{L}(\mathcal{R}'_k, \gamma) \ \nu_2$, it is easy to check that $\mu_1 \ \mathcal{L}(\mathcal{R}_k, \gamma) \ \mu_2$ and thus there exists an ε-weighting function w_γ such that $\mu_1 \ \mathcal{L}_w(\mathcal{R}_k, \gamma) \ \mu_2$. In fact, $\nu_1 \ \mathcal{L}(\mathcal{R}'_k, \gamma) \ \nu_2$ implies that there exists a weighting function w between ν'_1 and ν'_2 where $\nu'_1 \ \mathcal{L}(\mathcal{R}'_k) \ \nu'_2$, $\nu_1 = (1-\gamma)\nu'_1 + \gamma\nu''_1$ and $\nu_2 = (1-\gamma)\nu'_2 + \gamma\nu''_2$. Let w' be the function defined as: $w'(s_1, s_2) = (1-\gamma)\sum\{w(\alpha_1, \alpha_2) \mid \mathit{lstate}(\alpha_1) = s_1 \wedge \mathit{lstate}(\alpha_2) = s_2\}$. w' is an ε-weighting function from μ_1 to μ_2:

- $w'(s_1, s_2) > 0$ implies that $(1-\gamma)\sum\{w(\alpha_1, \alpha_2) \mid \mathit{lstate}(\alpha_1) = s_1 \wedge \mathit{lstate}(\alpha_2) = s_2\} > 0$ and thus there exists α_1, α_2 such that $w(\alpha_1, \alpha_2) > 0$, $\mathit{lstate}(\alpha_1) = s_1$ and $\mathit{lstate}(\alpha_2) = s_2$. This implies that $\alpha_1 \ \mathcal{R}'_k \ \alpha_2$ and hence $\mathit{lstate}(\alpha_1) \ \mathcal{R}_k \ \mathit{lstate}(\alpha_2)$ that is $s_1 \ \mathcal{R}_k \ s_2$;
- for each $s_2 \in S_k^2$,

$$
\begin{aligned}
&\sum_{s_1 \in S_k^1} w'(s_1, s_2) \\
&= \sum_{s_1 \in S_k^1} (1-\gamma) \sum\{w(\alpha_1, \alpha_2) \mid \mathit{lstate}(\alpha_1) = s_1 \wedge \mathit{lstate}(\alpha_2) = s_2\} \\
&= (1-\gamma) \sum_{\mathit{lstate}(\alpha_2) = s_2} \sum_{s_1 \in S_k^1} \{w(\alpha_1, \alpha_2) \mid \mathit{lstate}(\alpha_1) = s_1\} \\
&= (1-\gamma) \sum_{\mathit{lstate}(\alpha_2) = s_2} \sum\{w(\alpha_1, \alpha_2)\} \\
&= (1-\gamma) \sum_{\mathit{lstate}(\alpha_2) = s_2} \nu'_2(\alpha_2) \\
&= (1-\gamma) \sum\{\nu'_2(\alpha_2) \mid \mathit{lstate}(\alpha_2) = s_2\} \\
&\leqslant (1-\gamma)\mu_2(s_2) \\
&\leqslant \mu_2(s_2)
\end{aligned}
$$

- for each $s_1 \in S_k^1$,

$$
\begin{aligned}
&\sum_{s_2 \in S_k^2} w'(s_1, s_2) \\
&= \sum_{s_2 \in S_k^2} (1-\gamma) \sum \{w(\alpha_1, \alpha_2) \mid \mathit{lstate}(\alpha_1) = s_1 \wedge \mathit{lstate}(\alpha_2) = s_2\} \\
&= (1-\gamma) \sum_{\mathit{lstate}(\alpha_1) = s_1} \sum_{s_2 \in S_k^2} \{w(\alpha_1, \alpha_2) \mid \mathit{lstate}(\alpha_2) = s_2\} \\
&= (1-\gamma) \sum_{\mathit{lstate}(\alpha_1) = s_1} \sum \{w(\alpha_1, \alpha_2)\} \\
&= (1-\gamma) \sum_{\mathit{lstate}(\alpha_1) = s_1} \nu_1'(\alpha_1) \\
&= (1-\gamma) \sum \{\nu_1'(\alpha_1) \mid \mathit{lstate}(\alpha_1) = s_1\} \\
&\leqslant (1-\gamma)\mu_1(s_1) \\
&\leqslant \mu_1(s_1)
\end{aligned}
$$

– finally,

$$
\begin{aligned}
&\sum_{\substack{s_1 \in S_k^1 \\ s_2 \in S_k^2}} w'(s_1, s_2) \\
&= \sum_{\substack{s_1 \in S_k^1 \\ s_2 \in S_k^2}} (1-\gamma) \sum \{w(\alpha_1, \alpha_2) \mid \mathit{lstate}(\alpha_1) = s_1 \wedge \mathit{lstate}(\alpha_2) = s_2\} \\
&= (1-\gamma) \sum_{\substack{s_1 \in S_k^1 \\ s_2 \in S_k^2}} \{w(\alpha_1, \alpha_2) \mid \mathit{lstate}(\alpha_1) = s_1 \wedge \mathit{lstate}(\alpha_2) = s_2\} \\
&= (1-\gamma) \sum_{\substack{\alpha_1 \in \mathit{Execs}^*(\mathcal{A}_k^1) \\ \alpha_2 \in \mathit{Execs}^*(\mathcal{A}_k^2)}} \{w(\alpha_1, \alpha_2)\} \\
&= (1-\gamma)1 \\
&= 1-\gamma
\end{aligned}
$$

This means that w' is an ε-weighting function from μ_1 to μ_2 and thus $\mu_1 \; \mathcal{L}_w(\mathcal{R}_k, \gamma) \; \mu_2$.

Since $\mu_1 \; \mathcal{L}_w(\mathcal{R}_k, \gamma) \; \mu_2$ and μ_1 is reachable within $p(k)$ steps in $\mathcal{A}_k^1$, we have that there exists an ε-weighting function w_γ from μ_1 to μ_2 such that $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg \, \mathcal{R}_k(k^{-c}) \; s_2\} < k^{-c}$.

Let $\eta_{\alpha,tr}$ be the measure defined as: for each βas,

$$\eta_{\alpha,tr}(\beta as) = \begin{cases} \rho_{tr}(s) & \text{if } \beta = \alpha,\ lstate(\alpha) = src(tr) \text{ and } a = action(tr), \\ 0 & \text{otherwise.} \end{cases}$$

By definition of $\nu_1 \longrightarrow \upsilon_1$, it follows that there exists a scheduler σ_1 such that $\upsilon_1 = \sum_{\alpha_1}(\sigma_1(\alpha_1)(\bot) + \sum_{tr}\sigma_1(\alpha_1)(tr)\eta_{\alpha_1,tr})\nu_1(\alpha_1)$.

Given two executions α_1 and α_2 such that $lstate(\alpha_1)\ \mathcal{R}_k\ lstate(\alpha_2)$ and a transition $tr = (lstate(\alpha_1), a, \rho_1)$, denote by $m(\alpha_1, tr, \alpha_2)$ the combined transition $(lstate(\alpha_2), a, \rho_2)$ such that $\rho_1\ \mathcal{L}_w(\mathcal{R}_k, \zeta)\ \rho_2$ and by $w_\zeta^{m(\alpha_1,tr,\alpha_2)}$ the ε-weighting function that justifies $\rho_1\ \mathcal{L}_w(\mathcal{R}_k, \zeta)\ \rho_2$ for some $\zeta \in \mathbb{R}^{\geqslant 0}$.

Define $w_\zeta^{*m(\alpha_1,tr,\alpha_2)}$ as

$$w_\zeta^{*m(\alpha_1,tr,\alpha_2)}(\alpha_1', \alpha_2') = \begin{cases} w_\zeta^{m(\alpha_1,tr,\alpha_2)}(s_1, s_2) \\ \qquad \text{if } \alpha_1' = \alpha_1 a s_1,\ \alpha_2' = \alpha_2 a s_2,\ a = action(tr), \\ 0 \\ \qquad \text{otherwise.} \end{cases}$$

Since $\nu_1\ \mathcal{L}(\mathcal{R}_k', \gamma)\ \nu_2$, denote by w_γ^* the ε-weighting function that justifies $\nu_1\ \mathcal{L}(\mathcal{R}_k', \gamma)\ \nu_2$.

Now, define the scheduler σ_2 for $\mathcal{A}_k^2$ as $\sigma_2(\alpha_2) = \sum_{\alpha_1 \mathcal{R}_k(\varepsilon) \alpha_2} w_\gamma^*(\alpha_1, \alpha_2) \cdot \sum_{tr \in Supp(\sigma_1(\alpha_1))} \sigma(\alpha_1)(tr)\delta_{m(\alpha_1,tr,\alpha_2)}$. Denote by $\sigma_2(\alpha_2)(\bot)$ the value $1 - \sum_{tr_2}\sigma_2(\alpha_2)(tr_2)$.

Let $\eta_{\alpha_1,tr_1,\alpha_2}$ be the measure defined as: for each $\beta_2 as$,

$$\eta_{\alpha_1,tr_1,\alpha_2}(\beta_2 as) = \begin{cases} \rho_{m(\alpha_1,tr_1,\alpha_2)}(s) & \text{if } \beta_2 = \alpha_2 \text{ and } a = action(tr_1), \\ 0 & \text{otherwise} \end{cases}$$

and define the measure υ_2 as: $\upsilon_2 = \sum_{\alpha_2}\sum_{\alpha_1} w_\gamma^*(\alpha_1, \alpha_2)(\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2} + \sum_{tr_1 \in Supp(\sigma_1(\alpha_1))} \sigma_1(\alpha_1)(tr_1)\eta_{\alpha_1,tr_1,\alpha_2})$. Finally, define the function $w_{\gamma+k^{-c}}$ as $\sum_{\alpha_2}\sum_{\alpha_1} w_\gamma^*(\alpha_1, \alpha_2)(\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2} + \sum_{tr_1}\sigma_1(\alpha_1)(tr_1) w_{k^{-c}}^{*m(\alpha_1,tr_1,\alpha_2)})$.

$w_{\gamma+k^{-c}}$ is an ε-weighting function from υ_1 to υ_2. In fact,

- $w_{\gamma+k^{-c}}(\alpha_1', \alpha_2') > 0$ implies $\sum_{\alpha_2}\sum_{\alpha_1} w_\gamma^*(\alpha_1, \alpha_2)(\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2}(\alpha_2') + \sum_{tr_1}\sigma_1(\alpha_1)(tr_1) w_{k^{-c}}^{*m(\alpha_1,tr_1,\alpha_2)}(\alpha_1', \alpha_2')) > 0$. This implies that there exist α_1, α_2 such that $w_\gamma^*(\alpha_1, \alpha_2) > 0$ and thus $\alpha_1\ \mathcal{R}_k'\ \alpha_2$. Now, we have that at least one between the addends $\sigma_1(\alpha_1)(tr_1) w_{k^{-c}}^{*m(\alpha_1,tr_1,\alpha_2)}(\alpha_1', \alpha_2'))$ and $\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2}(\alpha_2')$ is greater than 0. In the former case, there exists tr_1 such that $\sigma(\alpha_1)(tr_1) > 0$. By definition of $w_{k^{-c}}^{*m(\alpha_1,tr_1,\alpha_2)}$ it follows that there exist states s_1 and s_2 such that $\alpha_1' = \alpha_1 action(tr_1)s_1$, $\alpha_2' = \alpha_2 action(tr_1)s_2$ and $w_{k^{-c}}^{*m(\alpha_1,tr_1,\alpha_2)}(\alpha_1', \alpha_2') > 0$ that implies that

$w_{k^{-c}}^{m(\alpha_1,tr_1,\alpha_2)}(s_1,s_2) > 0$ and thus, by definition of $w_{k^{-c}}^{m(\alpha_1,tr_1,\alpha_2)}$, it follows that $s\ \mathcal{R}_k\ t$ and thus, by definition of $\mathcal{R}'_k$, $\alpha'_1\ \mathcal{R}'_k\ \alpha'_2$. In the latter case (that is, $\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2}(\alpha'_2) > 0$), we have that $\alpha'_1 = \alpha_1$ and that $\alpha'_2 = \alpha_2$. Since $\alpha_1\ \mathcal{R}'_k\ \alpha_2$, it follows that $\alpha'_1\ \mathcal{R}'_k\ \alpha'_2$.

– for each α'_1,

$$
\begin{aligned}
&\sum_{\alpha'_2} w_{\gamma+k^{-c}}(\alpha'_1,\alpha'_2)\\
&= \sum_{\alpha'_2} \sum_{\alpha_2\in Supp(\nu_2)} \sum_{\alpha_1} w^*_\gamma(\alpha_1,\alpha_2)(\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2}(\alpha'_2)\\
&\qquad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1) w_{k^{-c}}^{*m(\alpha_1,tr_1,\alpha_2)}(\alpha'_1,\alpha'_2))\\
&= \sum_{\alpha_2\in Supp(\nu_2)} \sum_{\alpha_1} w^*_\gamma(\alpha_1,\alpha_2)(\sigma_1(\alpha_1)(\bot)\\
&\qquad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1) \sum_{\alpha'_2} w_{k^{-c}}^{*m(\alpha_1,tr_1,\alpha_2)}(\alpha'_1,\alpha'_2))\\
&= \sum_{\alpha_2\in Supp(\nu_2)} \sum_{\alpha_1} w^*_\gamma(\alpha_1,\alpha_2)(\sigma_1(\alpha_1)(\bot)\\
&\qquad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1) \sum_{\alpha_2 as} w_{k^{-c}}^{m(\alpha_1,tr_1,\alpha_2)}(lstate(\alpha'_1),s))\\
&= \sum_{\alpha_2\in Supp(\nu_2)} \sum_{\alpha_1} w^*_\gamma(\alpha_1,\alpha_2) + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1)\mu_{tr_1}(lstate(\alpha'_1)))\\
&= \sum_{\alpha_2\in Supp(\nu_2)} \sum_{\alpha_1} w^*_\gamma(\alpha_1,\alpha_2)(\sigma_1(\alpha_1)(\bot) + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1)\eta_{\alpha_1,tr_1}(\alpha'_1))\\
&= \sum_{\alpha_1} \sum_{\alpha_2\in Supp(\nu_2)} w^*_\gamma(\alpha_1,\alpha_2)(\sigma_1(\alpha_1)(\bot) + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1)\eta_{\alpha_1,tr_1}(\alpha'_1))\\
&= \sum_{\alpha_1} (\sigma_1(\alpha_1)(\bot) + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1)\eta_{\alpha_1,tr_1}(\alpha'_1)) \sum_{\alpha_2\in Supp(\nu_2)} w^*_\gamma(\alpha_1,\alpha_2)\\
&= \sum_{\alpha_1} (\sigma_1(\alpha_1)(\bot) + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1)\eta_{\alpha_1,tr_1}(\alpha'_1))\nu_1(\alpha_1)\\
&= \upsilon_1(\alpha'_1)
\end{aligned}
$$

– for each α'_2,

$$
\begin{aligned}
&\sum_{\alpha'_1} w_{\gamma+k^{-c}}(\alpha'_1,\alpha'_2)\\
&= \sum_{\alpha'_1} \sum_{\alpha_2\in Supp(\nu_2)} \sum_{\alpha_1} w^*_\gamma(\alpha_1,\alpha_2)(\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2}(\alpha'_2)
\end{aligned}
$$

$$
\begin{aligned}
&\quad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1) w_{k^{-c}}^{*m(\alpha_1,tr_1,\alpha_2)}(\alpha_1', \alpha_2')) \\
&= \sum_{\alpha_2 \in Supp(\nu_2)} \sum_{\alpha_1} w_\gamma^*(\alpha_1, \alpha_2)(\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2}\alpha_2' \\
&\quad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1) \sum_{\alpha_1'} w_{k^{-c}}^{*m(\alpha_1,tr_1,\alpha_2)}(\alpha_1', \alpha_2')) \\
&= \sum_{\alpha_2 \in Supp(\nu_2)} \sum_{\alpha_1} w_\gamma^*(\alpha_1, \alpha_2)(\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2}(\alpha_2') \\
&\quad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1) \sum_{\alpha_1 as} w_{k^{-c}}^{m(\alpha_1,tr_1,\alpha_2)}(s, lstate(\alpha_2'))) \\
&= \sum_{\alpha_2 \in Supp(\nu_2)} \sum_{\alpha_1} w_\gamma^*(\alpha_1, \alpha_2)(\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2}(\alpha_2') \\
&\quad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1)\rho_{m(\alpha_1,tr_1,\alpha_2)}(lstate(\alpha_2'))) \\
&= \sum_{\alpha_2 \in Supp(\nu_2)} \sum_{\alpha_1} w_\gamma^*(\alpha_1, \alpha_2)(\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2}(\alpha_2') \\
&\quad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1)\eta_{\alpha_1,tr_1,\alpha_2}(\alpha_2')) \\
&= \nu_2(\alpha_2')
\end{aligned}
$$

– finally,

$$
\begin{aligned}
&\sum_{\alpha_1'} \sum_{\alpha_2'} w_{\gamma+k^{-c}}(\alpha_1', \alpha_2') \\
&= \sum_{\alpha_1'} \sum_{\alpha_2'} \sum_{\alpha_2 \in Supp(\nu_2)} \sum_{\alpha_1} w_\gamma^*(\alpha_1, \alpha_2)(\sigma_1(\alpha_1)(\bot)\delta_{\alpha_2}(\alpha_2') \\
&\quad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1) w_{k^{-c}}^{*m(\alpha_1,tr_1,\alpha_2)}(\alpha_1', \alpha_2')) \\
&= \sum_{\alpha_2 \in Supp(\nu_2)} \sum_{\alpha_1} w_\gamma^*(\alpha_1, \alpha_2)(\sigma_1(\alpha_1)(\bot) \\
&\quad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1) \sum_{\alpha_1'} \sum_{\alpha_2'} w_{k^{-c}}^{*m(\alpha_1,tr_1,\alpha_2)}(\alpha_1', \alpha_2')) \\
&\geqslant \sum_{\alpha_2 \in Supp(\nu_2)} \sum_{\alpha_1} w_\gamma^*(\alpha_1, \alpha_2)(\sigma_1(\alpha_1)(\bot) \\
&\quad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1)(1 - k^{-c}))
\end{aligned}
$$

$$
\begin{aligned}
&\geqslant \sum_{\alpha_2 \in Supp(\nu_2)} \sum_{\alpha_1} w^*_\gamma(\alpha_1,\alpha_2)((1-k^{-c})\sigma_1(\alpha_1)(\bot) \\
&\quad + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1)(1-k^{-c})) \\
&= (1-k^{-c}) \sum_{\alpha_2 \in Supp(\nu_2)} \sum_{\alpha_1} w^*_\gamma(\alpha_1,\alpha_2)(\sigma_1(\alpha_1)(\bot) + \sum_{tr_1} \sigma_1(\alpha_1)(tr_1)) \\
&= (1-k^{-c}) \sum_{\alpha_2 \in Supp(\nu_2)} \sum_{\alpha_1} w^*_\gamma(\alpha_1,\alpha_2) \\
&\geqslant (1-k^{-c})(1-\gamma) \\
&= (1-\gamma-k^{-c}) + \gamma k^{-c} \\
&\geqslant 1-\gamma-k^{-c}
\end{aligned}
$$

□

As we have seen, the polynomially accurate simulation defined over states implies the one defined over executions. An important result of the simulations is that a set of states is negligible in an automaton $\mathcal{A}$ if and only if $\mathcal{A}$ is simulated by its G-conditional automaton. We can prove the same result using the state polynomially accurate simulations.

Theorem 5.8 (Conditional Automaton Theorem). *Let $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$ be a family of probabilistic automata and $\{G_k\}_{k\in\mathbb{N}}$ be a family of states such that, for each $k \in \mathbb{N}$, $\bar{s}_k \in G_k$. For each $k \in \mathbb{N}$ let B_k be the set $S_k \backslash G_k$. Then the family of identity relations is a state polynomially accurate simulation from $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k\}_{k\in\mathbb{N}}|\{G_k\}_{k\in\mathbb{N}}$ if and only if $\{B_k\}_{k\in\mathbb{N}}$ is negligible in $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$.*

Proof. To simplify the notation, denote by $\mathcal{A}^1_k$ the automaton $\mathcal{A}_k$ and by $\mathcal{A}^2_k$ the automaton $\mathcal{A}_k|G_k$, and hence by $\{\mathcal{A}^2_k\}_{k\in\mathbb{N}}$ the family $\{\mathcal{A}_k\}_{k\in\mathbb{N}}|\{G_k\}_{k\in\mathbb{N}}$.

(⇒) Since $\{\mathcal{A}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k\}_{k\in\mathbb{N}}|\{G_k\}_{k\in\mathbb{N}}$, by Proposition 5.7 we have that $\{\mathcal{A}_k\}_{k\in\mathbb{N}} \lesssim \{\mathcal{A}_k\}_{k\in\mathbb{N}}|\{G_k\}_{k\in\mathbb{N}}$ and thus, by Theorem 4.13, $\{B_k\}_{k\in\mathbb{N}}$ is negligible in $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$.

(⇐) Let $\mathcal{R}= \{id_k\}_{k\in\mathbb{N}}$ be the family of identity relations.

The condition on the start states is trivially true: by definition of conditional automaton, the start state is the same, thus for each $k \in \mathbb{N}$, $\bar{s}^1_k \; id_k \; \bar{s}^2_k$.

For the step condition, fix $c \in \mathbb{N}$, $p \in Poly$, $\bar{k} \in \mathbb{N}$, $k > \bar{k}$, γ, μ_1, μ_2 such that μ_1 is reached in at most $p(k)$ steps in $\mathcal{A}^1_k$, and $\mu_1 \; \mathcal{L}_w(id_k, \gamma) \; \mu_2$. Let

w_γ the γ-weighting function that justifies $\mu_1\ \mathcal{L}_w(id_k, \gamma)\ \mu_2$. We must verify that $\sum\{w_\gamma(s_1, s_2) \mid (s_1, s_2) \notin id_k(k^{-c})\} < k^{-c}$.
Suppose that $\sum\{w_\gamma(s_1, s_2) \mid (s_1, s_2) \notin id_k(k^{-c})\} \geqslant k^{-c}$. The condition $(s_1, s_2) \notin id_k(k^{-c})$ implies that either $(s_1, s_2) \notin id_k$ or $(s_1, s_2) \in id_k$ and there exists $s_1 \xrightarrow{a} \rho_1$ such that for each $s_2 \xrightarrow{a} \rho_2$, $\rho_1 \neg\ \mathcal{L}_w(id_k, k^{-c})\ \rho_2$. By definition of w_γ, it follows that $(s_1, s_2) \notin id_k$ implies $w_\gamma(s_1, s_2) = 0$ and thus $\sum\{w_\gamma(s_1, s_2) \mid (s_1, s_2) \notin id_k(k^{-c})\} = \sum\{w_\gamma(s_1, s_2) \mid (s_1, s_2) \notin id_k(k^{-c}) \wedge (s_1, s_2) \in id_k\}$.
Since $\mathcal{A}_k^2$ is the G_k-conditional of $\mathcal{A}_k^1$, it follows that for each transition (s_1, a, μ_1), in $\mathcal{A}_k^2$ there is a transition (s_2, a, ρ_2) where $\rho_2 = \rho_1|G_k$ if $\rho_1(G_k) > 0$. Since $(s_1, s_2) \notin id_k(k^{-c})$, it follows that there exists a transition (s_1, a, ρ_1), denoted by tr_{s_1}, such that $\rho_1(B_k) \geqslant k^{-c}$ (otherwise we can satisfy $\rho_1\ \mathcal{L}_w(id_k, k^{-c})\ \rho_1|G_k$). This implies that there exists a set S of states, $S \subseteq Supp(\mu_1)$, such that $\sum_{s_1 \in S, s_2 \in S_k^2}\{w(s_1, s_2)\} \geqslant k^{-c}$ and $s_1 \in S$ enables the transition tr_{s_1}. Let σ be a scheduler for $\mathcal{A}_k^1$ such that induces the measure μ_1 and for each execution α_1 such that $lstate(\alpha_1) = s_1 \in S$, $\sigma(\alpha_1) = \delta_{tr_{s_1}}$, that is, the scheduler chooses with probability 1 the transition that can not be simulated by $\mathcal{A}_k^2$. With this scheduler, the probability to reach states of B_k is $\sum_{s_1 \in S, s_2 \in S_k^2}\{w_\gamma(s_1, s_2)\mu_{tr_{s_1}}(B_k)\} \geqslant \sum_{s_1 \in S, s_2 \in S_k^2}\{k^{-c} w_\gamma(s_1, s_2)\} \geqslant k^{-c} \sum_{s_1 \in S, s_2 \in S_k^2}\{w_\gamma(s_1, s_2)\} \geqslant k^{-c} k^{-c} = k^{-2c}$. This implies that within $p(k) + 1$ steps, states of B_k are reached with probability at least k^{-2c} but this contradicts the negligibility of $\{B_k\}_{k \in \mathbb{N}}$ in $\{\mathcal{A}_k^1\}_{k \in \mathbb{N}}$. Since we have obtained an absurd, it follows that $\sum\{w_\gamma(s_1, s_2) \mid (s_1, s_2) \notin id_k(k^{-c})\} < k^{-c}$. □

The definition state polynomially accurate simulation satisfies few important properties. The first property is that sequences of n steps can be matched up to an error nk^{-c}; the second one is the compositionality; the third property is the execution correspondence.

Theorem 5.9. *Let $\{\mathcal{A}_k^1\}_{k \in \mathbb{N}}$ and $\{\mathcal{A}_k^1\}_{k \in \mathbb{N}}$ be two families of probabilistic automata such that $\{\mathcal{A}_k^1\}_{k \in \mathbb{N}} \lesssim_s \{\mathcal{A}_k^2\}_{k \in \mathbb{N}}$. Let $\mathcal{R} = \{\mathcal{R}_k\}_{k \in \mathbb{N}}$ be a state polynomially accurate simulation from $\{\mathcal{A}_k^1\}_{k \in \mathbb{N}}$ to $\{\mathcal{A}_k^2\}_{k \in \mathbb{N}}$.*

For each $c \in \mathbb{N}$, $p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ and each scheduler σ_1 for $\mathcal{A}_k^1$, if μ_1 is the probability measure induced by σ_1 after n steps, $n \leqslant p(k)$, then there exists a scheduler σ_2 for $\mathcal{A}_k^2$ that reaches, after n steps, a probability measure μ_2 such that $\mu_1\ \mathcal{L}_w(\mathcal{R}_k, nk^{-c})\ \mu_2$.

Proof. The proof is a classical inductive argument on the number of steps.

Case $n = 0$: for each $c \in \mathbb{N}$, $p \in \mathit{Poly}$, and $k \in \mathbb{N}$ after 0 steps, it is not possible to reach states different from start state of $\mathcal{A}^1_k$. This means that after 0 steps, reached measure is $\delta_{\bar{s}_1}$. Analogously, for $\mathcal{A}^2_k$ $\delta_{\bar{s}_2}$ is reached after 0 steps. Since $\bar{s}_1\ \mathcal{R}_k\ \bar{s}_2$, we have that $\delta_{\bar{s}_1}\ \mathcal{L}_w(\mathcal{R}_k, 0)\ \delta_{\bar{s}_2}$ and thus $\delta_{\bar{s}_1}\ \mathcal{L}_w(\mathcal{R}_k, 0k^{-c})\ \delta_{\bar{s}_2}$.

Case $n > 0$: fix $c \in \mathbb{N}$, $p \in \mathit{Poly}$. Let $\bar{k} \in \mathbb{N}$ be a value such that for each $k > \bar{k}$, $n < p(k)$. Let μ_1 and μ_2 be two measures such that there exist schedulers σ_1 and σ_2 such that μ_1 is reached via σ_1 after n steps, μ_2 is reached via σ_2 after n steps, and $\mu_1\ \mathcal{L}_w(\mathcal{R}_k, nk^{-c})\ \mu_2$. Let $w_{nk^{-c}}$ be the ε-weighting function that justifies $\mu_1\ \mathcal{L}_w(\mathcal{R}_k, nk^{-c})\ \mu_2$.
Suppose that there exists φ_1 such that $\mu_1 \longrightarrow \varphi_1$. Since $n < p(k)$ and μ_1 is reached after n steps, it follows that φ_1 is reached after $n + 1 \leqslant p(k)$ steps. By hypothesis, $\{\mathcal{A}^1_k\}_{k\in K} \lesssim_s \{\mathcal{A}^2_k\}_{k\in K}$ and thus there exists $\bar{k}' \in \mathbb{N}$ such that for each $k > \bar{k}'$ $\sum\{w_{nk^{-c}}(s_1, s_2) \mid s_1 \neg\ \mathcal{R}_k(k^{-c})\ s_2\} < k^{-c}$ and this implies, by Proposition 5.4, that there exists φ_2 such that $\varphi_1\ \mathcal{L}_w(\mathcal{R}_k, nk^{-c} + k^{-c})\ \varphi_2$, that is $\varphi_1\ \mathcal{L}_w(\mathcal{R}_k, (n + 1)k^{-c})\ \varphi_2$. □

We now turn to compositionality:

Theorem 5.10. *Let $\{\mathcal{A}^1_k\}_{k\in\mathbb{N}}$ and $\{\mathcal{A}^2_k\}_{k\in\mathbb{N}}$ be two families of automata such that $\{\mathcal{A}^1_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}^2_k\}_{k\in\mathbb{N}}$.*

For each context $\mathcal{C}_k$ compatible with both $\mathcal{A}^1_k$ and $\mathcal{A}^2_k$,

$$\{\mathcal{A}^1_k || \mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}^2_k || \mathcal{C}_k\}_{k\in\mathbb{N}}$$

Before proving the theorem, we need the following result:

Lemma 5.11. *Let $\{\mathcal{A}^1_k\}_{k\in\mathbb{N}}$ and $\{\mathcal{A}^2_k\}_{k\in\mathbb{N}}$ be two families of automata such that $\{\mathcal{A}^1_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}^2_k\}_{k\in\mathbb{N}}$. Let $\{\mathcal{R}_k\}_{k\in\mathbb{N}}$ be a state polynomially accurate simulation that justifies $\{\mathcal{A}^1_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}^2_k\}_{k\in\mathbb{N}}$. For each context $\mathcal{C}_k$ compatible with both $\mathcal{A}^1_k$ and $\mathcal{A}^2_k$, define $\{\mathcal{R}'_k\}_{k\in\mathbb{N}}$ as the family of relations $\mathcal{R}'_k \subseteq (S^1_k \times S^c_k) \times (S^2_k \times S^c_k)$, where S^c_k is the set of states of $\mathcal{C}_k$, such that for each measure μ_1 and μ_2, $\mu_1\ \mathcal{L}_w(\mathcal{R}'_k, \gamma)\ \mu_2$ if and only if*

- $\mu_1\ \mathcal{L}_w(\mathcal{R}_k \times id_k, \gamma)\ \mu_2$,
- $\mu_1 \lceil \mathcal{A}^1_k\ \mathcal{L}_w(\mathcal{R}_k, \gamma)\ \mu_2 \lceil \mathcal{A}^2_k$, *and*
- $\mu_1 \lceil \mathcal{C}_k = \mu_2 \lceil \mathcal{C}_k$.

For each $s_1 \in S^1_k$, $s_2 \in S^2_k$, and $s_c \in S^c_k$, if $s_1\ \mathcal{R}_k(\varepsilon)\ s_2$, then $(s_1, s_c)\ \mathcal{R}'_k(\varepsilon)\ (s_2, s_c)$.

Proof. Consider (s_1, s_c) and two transitions $s_1 \xrightarrow{a} \mu_1$ and $s_c \xrightarrow{b} \mu_c$. By hypothesis, there exists a combined transition $s_2 \xrightarrow{a}_c \mu_2$ such that $\mu_1 \, \mathcal{L}(\mathcal{R}_k, \varepsilon) \, \mu_2$. Property 3.5(10) implies that $\mu_1 \times \mu_c \, \mathcal{L}(\mathcal{R}_k \times id_k, \varepsilon) \, \mu_2 \times \mu_c$ and thus $(s_1, s_c) \, \mathcal{R}'_k(\varepsilon) \, (s_2, s_c)$. Observe that $(s_1, s_c) \neg \mathcal{R}'_k(\varepsilon) \, (s_2, s_c)$ implies that $s_1 \neg \mathcal{R}_k(\varepsilon) \, s_2$. □

of Theorem 5.10. Let $\{\mathcal{R}_k\}_{k\in\mathbb{N}}$ be a state polynomially accurate simulation that justifies $\{\mathcal{A}^1_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}^2_k\}_{k\in\mathbb{N}}$. Define $\{\mathcal{R}'_k\}_{k\in\mathbb{N}}$ as the family of relations $\mathcal{R}'_k \subseteq (S^1_k \times S^c_k) \times (S^2_k \times S^c_k)$, where S^c_k is the set of states of $\mathcal{C}_k$, such that for each measure μ_1 and μ_2, $\mu_1 \, \mathcal{L}_w(\mathcal{R}'_k, \gamma) \, \mu_2$ if and only if

- $\mu_1 \, \mathcal{L}_w(\mathcal{R}_k \times id_k, \gamma) \, \mu_2$,
- $\mu_1 \lceil \mathcal{A}^1_k \, \mathcal{L}_w(\mathcal{R}_k, \gamma) \, \mu_2 \lceil \mathcal{A}^2_k$, and
- $\mu_1 \lceil \mathcal{C}_k = \mu_2 \lceil \mathcal{C}_k$.

We now show that $\{\mathcal{R}'_k\}_{k\in\mathbb{N}}$ is a state polynomially accurate from $\{\mathcal{A}^1_k || \mathcal{C}_k\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}^2_k || \mathcal{C}_k\}_{k\in\mathbb{N}}$.

Condition on start states is trivially true: let $\bar{s}^c_k$ be the start state of $\mathcal{C}_k$. By hypothesis, we know that $\bar{s}^1_k \, \mathcal{R}_k \, \bar{s}^2_k$ and thus $\delta_{\bar{s}^1_k} \, \mathcal{L}(\mathcal{R}_k) \, \delta_{\bar{s}^2_k}$. Since $\delta_{\bar{s}^c_k} = \delta_{\bar{s}^c_k}$, it follows that $(\delta_{\bar{s}^1_k} \times \delta_{\bar{s}^c_k}) \, \mathcal{L}(\mathcal{R}'_k) \, (\delta_{\bar{s}^2_k} \times \delta_{\bar{s}^c_k})$ and thus $(\bar{s}^1_k, \bar{s}^c_k) \, \mathcal{R}'_k \, (\bar{s}^2_k, \bar{s}^c_k)$.

For the step condition, let $c \in \mathbb{N}$ and $p \in \mathit{Poly}$ and suppose that there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$, each $\gamma \geqslant 0$, and each measure $\mu_1 \in \mathit{Disc}(S^1_k \times S^c_k)$ and $\mu_2 \in \mathit{Disc}(S^2_k \times S^c_k)$, it holds that $\mu_1 \, \mathcal{L}_w(\mathcal{R}'_k, \gamma) \, \mu_2$. Let w'_γ be the ε-weighting function that justifies $\mu_1 \, \mathcal{L}_w(\mathcal{R}'_k, \gamma) \, \mu_2$. To prove the step condition, we must show that $\sum\{w'_\gamma((s_1, s_c), (s_2, s_c)) \mid (s_1, s_c) \neg \mathcal{R}'_k(k^{-c}) \, (s_2, s_c)\} < k^{-c}$. In fact, denoted by $w_\gamma(s_1, s_2)$ the value $\sum_{s_c \in S^c_k} w'_\gamma((s_1, s_c), (s_2, s_c))$,

$$
\begin{aligned}
&\sum_{\substack{(s_1,s^1_c)\in S^1_k \times S^c_k \\ (s_2,s^2_c)\in S^2_k \times S^c_k}} \{w'_\gamma((s_1, s^1_c), (s_2, s^2_c)) \mid (s_1, s_c) \neg \mathcal{R}'_k(k^{-c}) \, (s_2, s^2_c)\} \\
= &\sum_{\substack{(s_1,s_c)\in S^1_k \times S^c_k \\ (s_2,s_c)\in S^2_k \times S^c_k}} \{w'_\gamma((s_1, s_c), (s_2, s_c)) \mid (s_1, s_c) \neg \mathcal{R}'_k(k^{-c}) \, (s_2, s_c)\} \\
= &\sum_{\substack{s_1\in S^1_k \\ s_2\in S^2_k}} \{ \sum_{s_c \in S^c_k} w'_\gamma((s_1, s_c), (s_2, s_c)) \mid s_1 \neg \mathcal{R}_k(k^{-c}) \, s_2\} \\
= &\sum_{\substack{s_1\in S^1_k \\ s_2\in S^2_k}} \{w_\gamma(s_1, s_2) \mid s_1 \neg \mathcal{R}_k(k^{-c}) \, s_2\}
\end{aligned}
$$

$$< k^{-c}$$

where the first step is justified by the fact that for each $(s_1, s_c^1) \in S_k^1 \times S_k^c$ and each $(s_2, s_c^2) \in S_k^2 \times S_k^c$, if $s_c^1 \neq s_c^2$, then $w'_\gamma((s_1, s_c^1), (s_2, s_c^2)) = 0$; the second step is due to the contrapositive of Lemma 5.11; the third step by the definition of w_γ and the forth step by the step condition for $\mathcal{R}$ and the fact that w_γ is an ε-weighing function for $\mu_1 \lceil \mathcal{A}_k^1 \; \mathcal{L}_w(\mathcal{R}, \gamma) \; \mu_2 \lceil \mathcal{A}_k^2$. In fact, we have that

- the first condition is trivially satisfied: $w_\varepsilon(s_1, s_2) > 0$ implies that $\sum_{s_c \in S_k^c} w'_\gamma((s_1, s_c), (s_2, s_c)) > 0$ and thus there exists at least one $s_c \in S_k^c$ such that $w'_\gamma((s_1, s_c), (s_2, s_c)) > 0$. This implies that $(s_1, s_c) \; \mathcal{R}'_k \; (s_2, s_c)$, that is $(s_1, s_c) \; \mathcal{R}_k \times id_k \; (s_2, s_c)$ and thus $s_1 \; \mathcal{R}_k \; s_2$;
- the second condition is also verified: for each $s_1 \in S_k^1$,

$$\begin{aligned}
\sum_{s_2 \in S_k^2} w_\gamma(s_1, s_2) &= \sum_{s_2 \in S_k^2} \sum_{s_c \in S_k^c} w'_\gamma((s_1, s_c), (s_2, s_c)) \\
&= \sum_{(s_2, s_c) \in S_k^2 \times S_k^c} w'_\gamma((s_1, s_c), (s_2, s_c)) \\
&= \sum_{s'_c \in S_k^c} \sum_{(s_2, s_c) \in S_k^2 \times S_k^c} w'_\gamma((s_1, s'_c), (s_2, s_c)) \\
&\leqslant \sum_{s'_c \in S_k^c} \mu_1(s_1, s'_c) \\
&= \mu_1 \lceil \mathcal{A}_k^1(s_1)
\end{aligned}$$

- as well as the third condition: for each $s_2 \in S_k^2$,

$$\begin{aligned}
\sum_{s_1 \in S_k^1} w_\gamma(s_1, s_2) &= \sum_{s_1 \in S_k^1} \sum_{s_c \in S_k^c} w'_\gamma((s_1, s_c), (s_2, s_c)) \\
&= \sum_{(s_1, s_c) \in S_k^1 \times S_k^c} w'_\gamma((s_1, s_c), (s_2, s_c)) \\
&= \sum_{s'_c \in S_k^c} \sum_{(s_1, s_c) \in S_k^1 \times S_k^c} w'_\gamma((s_1, s_c), (s_2, s'_c)) \\
&\leqslant \sum_{s'_c \in S_k^c} \mu_2(s_2, s'_c) \\
&= \mu_2 \lceil \mathcal{A}_k^2(s_2)
\end{aligned}$$

- finally, also the fourth condition is satisfied:

$$\begin{aligned}
\sum_{\substack{s_1\in S_k^1\\ s_2\in S_k^2}} w_\gamma(s_1,s_2) &= \sum_{\substack{s_1\in S_k^1\\ s_2\in S_k^2}} \sum_{s_c\in S_k^c} w'_\gamma((s_1,s_c),(s_2,s_c)) \\
&= \sum_{\substack{s_1\in S_k^1\\ s_2\in S_k^2\\ s_c\in S_k^c}} w'_\gamma((s_1,s_c),(s_2,s_c)) \\
&= \sum_{\substack{(s_1,s_c^1)\in S_k^1\times S_k^c\\ (s_2,s_c^2)\in S_k^2\times S_k^c}} w'_\gamma((s_1,s_c^1),(s_2,s_c^2)) \\
&\geqslant 1-\gamma
\end{aligned}$$

This completes the proof, since we have that w_γ is an ε-weighing function for $\mu_1 \lceil \mathcal{A}_k^1 \; \mathcal{L}_w(\mathcal{R},\gamma) \; \mu_2 \lceil \mathcal{A}_k^2$. □

Similarly to the case of polynomially accurate simulations, we are able to prove the

Theorem 5.12 (Execution Correspondence Theorem). *Let* $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$, $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$, …, $\{\mathcal{A}_k^n\}_{k\in\mathbb{N}}$ *be* n *families of probabilistic automata such that there exist* $n-1$ *families of relations* $\{\mathcal{R}_k^1\}_{k\in\mathbb{N}}$, $\{\mathcal{R}_k^2\}_{k\in\mathbb{N}}$, …, $\{\mathcal{R}_k^{n-1}\}_{k\in\mathbb{N}}$ *such for each* $0<i<n$, $\{\mathcal{R}_k^i\}_{k\in\mathbb{N}}$ *is a state polynomially accurate simulation from* $\{\mathcal{A}_k^i\}_{k\in\mathbb{N}}$ *to* $\{\mathcal{A}_k^{i+1}\}_{k\in\mathbb{N}}$.

For each $c\in\mathbb{N}$ *and* $p\in Poly$, *there exists* $\bar{k}\in\mathbb{N}$ *such that for each* $k>\bar{k}$ *and each probability measure* $\mu^1\in Disc(S_k^1)$, *if* μ^1 *is reachable within* $p(k)$ *steps in* $\mathcal{A}_k^1$, *then there exists* $\mu^n\in Disc(S_k^n)$ *such that* μ^n *is reachable within* $p(k)$ *steps in* $\mathcal{A}_k^n$ *and* $\mu^1 \; \mathcal{L}(\mathcal{R}_k^1\circ\ldots\circ\mathcal{R}_k^{n-1}, p(k)k^{-c}) \; \mu^n$.

Proof. Fix $c\in\mathbb{N}$, $p\in Poly$ and suppose that there exists $\bar{k}'\in\mathbb{N}$ such that for each $k>\bar{k}'$, $\mu^1\in Disc(S_k^1)$ is reachable within $p(k)$ steps in $\mathcal{A}_k^1$. Let s the actual number of steps performed to reach μ^1.

By Theorem 5.9, it follows that for each $c_1'\in\mathbb{N}$, there exists $\bar{k}_1\in\mathbb{N}$ such that for each $k>\bar{k}_1$, there exists a measure $\mu^2\in Disc(S_k^2)$ such that μ^2 is reachable in s steps and $\mu^1 \; \mathcal{L}_w(\mathcal{R}_k^1, sk^{-c_1'}) \; \mu^2$.

Since μ^2 is reachable in $\mathcal{A}_k^2$ in s steps, $s\leqslant p(k)$, it follows that for each $c_2'\in\mathbb{N}$, there exists $\bar{k}_2\in\mathbb{N}$ such that for each $k>\bar{k}_2$, there exists a measure $\mu^3\in Disc(S_k^3)$ such that μ^3 is reachable in s steps and $\mu^2 \; \mathcal{L}_w(\mathcal{R}_k^2, sk^{-c_2'}) \; \mu^3$.

Iterating this reasoning, we obtain that for each $1<i<n$ and each $c_i'\in\mathbb{N}$, there exists $\bar{k}_i\in\mathbb{N}$ such that for each $k>\bar{k}_i$, there exists a measure

$\mu^{i+1} \in Disc(S_k^{i+1})$ such that μ^i is reachable in s steps and $\mu^i \; \mathcal{L}_w(\mathcal{R}_k^i, sk^{-c_i'})$ μ^{i+1}.

Let $\bar{k} = \max\{\bar{k}', \bar{k}_1, \ldots, \bar{k}_{n-1}\}$. Since $\bar{k} \geqslant \bar{k}'$, for each $k > \bar{k}$ the measure μ^1 is still reachable within $p(k)$ steps and $s \leqslant p(k)$. Since for each $0 < i < n$ $\bar{k} \geqslant \bar{k}_i$, we still have that for each $c_i' \in \mathbb{N}$, and each $k > \bar{k}$, there exists a measure $\mu^{i+1} \in Disc(S_k^{i+1})$ such that μ^{i+1} is reachable in s steps and $\mu^i \; \mathcal{L}_w(\mathcal{R}_k^i, sk^{-c_i'}) \; \mu^{i+1}$. Since for each $0 < i < n$ we can choose the value of c_i', let $c' \in \mathbb{N}$ be a value such that for each $l \in \mathbb{N}$, $l > \bar{k}$, we have that $(n-1)l^{-c'} \leqslant l^{-c}$; for each $0 < i < n$, take $c_i' = c'$. This implies that for each $k > \bar{k}$, there exists a measure $\mu^{i+1} \in Disc(S_k^{i+1})$ such that μ^{i+1} is reachable in s steps and $\mu^i \; \mathcal{L}_w(\mathcal{R}_k^i, sk^{-c'}) \; \mu^{i+1}$.

Propositions 3.6 and 5.3 imply that $\mu^1 \; \mathcal{L}(\mathcal{R}_k^1 \circ \ldots \circ \mathcal{R}_k^{n-1}, \sum_{i=1}^{n-1} sk^{-c'})$ μ^n. Consider $\sum_{i=1}^{n-1} sk^{-c'}$. We have that $\sum_{i=1}^{n-1} sk^{-c'} = s\sum_{i=1}^{n-1} k^{-c'} = s(n-1)k^{-c'} \leqslant sk^{-c} \leqslant p(k)k^{-c}$. By Property 3.5(2), it follows that μ^n is reachable in $s \leqslant p(k)$ steps and $\mu^1 \; \mathcal{L}(\mathcal{R}_k^1 \circ \ldots \circ \mathcal{R}_k^{n-1}, p(k)k^{-c}) \; \mu^n$, that implies, by Proposition 5.3, that $\mu^1 \; \mathcal{L}_w(\mathcal{R}_k^1 \circ \ldots \circ \mathcal{R}_k^{n-1}, p(k)k^{-c}) \; \mu^n$, as required. □

The state polynomially accurate simulation is preserved by the hiding operation:

Proposition 5.13. *Let $\mathcal{A}_k$ and $\mathcal{B}_k$ be two families of automata such that $\{\mathcal{A}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k\}_{k\in\mathbb{N}}$. Let H be a set of actions.*

Then $\{\mathrm{Hide}_H(\mathcal{A}_k)\}_{k\in\mathbb{N}} \lesssim_s \{\mathrm{Hide}_H(\mathcal{B}_k)\}_{k\in\mathbb{N}}$.

Proof. There is nothing to prove, since the definition of state polynomially accurate simulation does not care if an action is internal or external. □

5.2 The State Weak Polynomially Accurate Simulation

The two notions of polynomially accurate simulations, the one based on executions and the one based on states, are both *strong*, that is, they do not distinguish between external and internal actions. This means that given two families of automata $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ and $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$, each time $\mathcal{A}_k^1$ performs a transition also $\mathcal{A}_k^2$ must perform a matching transition, even if the matched step involves only transitions labelled by internal actions.

The main scope of the internal actions is to model the internal computation of the automaton. Consider, for example, the coin flipper *CF* in Figure 5.2: the automaton flips the coin when it performs the internal action *coin_flip*. Now, consider the slightly different automaton *CF′* where

Coin Flipper

Signature:

Output:
$coin\_value(v)$, $v \in \{H, T\}$

Input:
flip

Internal:
coin_flip

Transitions:

Input *flip*
Effect:
$value := F$

Internal *coin_flip*
Precondition:
$value = F$
Effect:
$value := c$ where $c \in_R \{H, T\}$

Output $coin\_value(v)$
Precondition:
$v = value$
Effect:
$value := \perp$

Fig. 5.2. A coin flipper

the effect of the input action *flip* is $value := c$ where $c \in_R \{H, T\}$ (instead of $value := F$) and where we remove the *coin_flip* action. Intuitively, the two automata implement the same functionality, but CF' does not simulate CF. In fact, CF' is not able to simulate the transition $s \xrightarrow{flip} \delta_{s_f}$ where $s_f.value = F$, since it can only perform the transition $s' \xrightarrow{flip} \mu$ where $\mu(s'_h) = 0.5$, $\mu(s'_t) = 0.5$, $s_h.value = H$, and $s_t.value = T$.

To abstract from the internal computation, [105] has introduced the concept of weak simulation that requires that each transition is matched up to internal actions. This means that we can perform an arbitrary number of internal steps before and after performing a transition labelled by the same action of the matched transition.

We adopt the same approach to define the weak version of the state polynomially accurate simulation: we require that it is negligible the probability that given two related states s_1 and s_2, for each transition enabled by s_1 there does not exist a matching combined weak transition from s_2. We must take care of the definition of combined weak transition we use to define the weak state polynomially accurate simulation: we can not use the definition that is used for the ordinary weak simulation since it does

not impose a bound to the number of internal steps that are performed in the matching transition. In fact, if we do not impose a bound, then we are not able to prove again the Execution Correspondence theorem; in fact, given a measure μ_1 of the first automaton, we know that there exists a matching measure μ_2 reached by the second automaton but then we can not say that there exists a measure μ_3 for the third automaton since μ_3 exists if μ_2 is reached within a polynomial number of steps and we are not able to prove that μ_2 satisfies such constraint. Since we want to prove the Execution Correspondence theorem also for the weak version of the state polynomially accurate simulations, we must be able to state that the measure μ_2 is reached within a polynomial number of steps. We can achieve this property in several ways: the first one, that is also the simpler one, is that there exists a bound $l \in \mathbb{N}$ such that for each value of the security parameter k, the length of each weak transition is bounded by l; the second one is that the bound l polynomially depends on the value of k, that is, there exists $l \in \mathit{Poly}$ such that for each value of the security parameter k, the length of each weak transition is bounded by $l(k)$; the third one is that the expected length of the weak transitions is polynomially bounded.

In this thesis we adopt the second approach: the length of each weak transition is polynomially bounded. We leave the third approach as future work since it requires further study.

Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two probabilistic automata, $\zeta \in \mathbb{R}^{\geqslant 0}$, $l \in \mathbb{N}$ and $\mathcal{R}$ be a relation from S_1 to S_2. Given two states s_1 and s_2, we say that $s_1 \; \mathcal{R}^l(\zeta) \; s_2$ if $s_1 \; \mathcal{R} \; s_2$ and for each transition $(s_1, a, \mu_1) \in D_1$, there exists a weak combined l-bounded transition (s_2, a, μ_2) such that $\mu_1 \; \mathcal{L}_w(\mathcal{R}, \zeta) \; \mu_2$.

Proposition 5.14. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two automata and let $\mathcal{R}$ be a relation from S_1 to S_2. Let $\varepsilon, \gamma \in \mathbb{R}^{\geqslant 0}$, $l \in \mathbb{N}$ be three values and μ_1, μ_2 be two distributions in $\mathit{Disc}(S_1)$ and $\mathit{Disc}(S_2)$, respectively, such that there exists an ε-weighting function w_ε that justifies $\mu_1 \; \mathcal{L}_w(\mathcal{R}, \varepsilon) \; \mu_2$.*

If $\sum\{w_\varepsilon(s_1, s_2) \mid s_1 \neg \; \mathcal{R}^l(\gamma) \; s_2\} < \gamma$, and $\mu_1 \longrightarrow \varphi_1$ then there exists φ_2 such that $\varphi_1 \; \mathcal{L}_w(\mathcal{R}, \varepsilon + \gamma) \; \varphi_2$ and $\mu_2 \Longrightarrow^l \varphi_2$.

Proof. Given two states s_1 and s_2 such that $s_1 \; \mathcal{R}^l(\gamma) \; s_2$ and a transition $tr_1 = (s_1, a, \rho_1)$, denote by $m(s_2, tr_1)$ the weak l-bounded combined transition (s_2, a, ρ_2) such that $\rho_1 \; \mathcal{L}_w(\mathcal{R}, \gamma) \; \rho_2$ and by $w_\gamma^{s_2, tr_1}$ the ε-weighting function that justifies $\rho_1 \; \mathcal{L}_w(\mathcal{R}, \gamma) \; \rho_2$.

By definition of $\mu_1 \longrightarrow \varphi_1$, it follows that there exists a family of probabilities $\{p_{tr_1}\}_{tr_1 \in D_1}$ such that for each state $s_1 \in \mathit{Supp}(\mu_1)$,

$\sum_{tr_1 \in D_1, src(tr_1)=s_1} p_{tr_1} = 1$ and $\varphi_1 = \sum_{tr_1 \in D_1} p_{tr_1} \mu_1(src(tr_1))\mu_{tr_1}$. Let $\{p_{m(s_2,tr_1)}\}_{s_2 \in S_2, tr_1 \in D_1}$ be the family of values for each for each $s_2 \in S_2$, $p_{m(s_2,tr_1)} = p_{tr_1}$. Let φ_2 be defined as: for each $s'_2 \in S_2$, $\varphi_2(s'_2) = \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) \sum_{s'_1 \in Supp(\mu_{tr_1})} w_\gamma^{s_2,tr_1}(s'_1, s'_2)$.

Now, we need to verify that $\varphi_1 \; \mathcal{L}_w(\mathcal{R}, \varepsilon + \gamma) \; \varphi_2$. To do this, consider the function $w_{\varepsilon+\gamma}$ defined as:

$$w_{\varepsilon+\gamma}(s'_1, s'_2) = \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} \cdot w_\varepsilon(src(tr_1), s_2) w_\gamma^{s_2,tr_1}(s'_1, s'_2)$$

$w_{\varepsilon+\gamma}$ is an ε-weighting function from φ_1 to φ_2:

- $w_{\varepsilon+\gamma}(s'_1, s'_2) > 0$ implies that there exists $tr_1 \in D_1$, $s_2 \in Supp(\mu_2)$ such that $p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) w_\gamma^{s_2,tr_1}(s'_1, s'_2) > 0$ and this means that $w_\gamma^{s_2,tr_1}(s'_1, s'_2) > 0$. By definition of $w_\gamma^{s_2,tr_1}$, it follows that $s'_1 \; \mathcal{R} \; s'_2$;
- for each $s'_2 \in S_2$,

$$\begin{aligned}
&\sum_{s'_1 \in S_1} w_{\varepsilon+\gamma}(s'_1, s'_2) \\
&= \sum_{s'_1 \in S_1} \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) w_\gamma^{s_2,tr_1}(s'_1, s'_2) \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) \sum_{s'_1 \in S_1} w_\gamma^{s_2,tr_1}(s'_1, s'_2) \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) \sum_{s'_1 \in Supp(\mu_{tr_1})} w_\gamma^{s_2,tr_1}(s'_1, s'_2) \\
&= \varphi_2(s'_2) \\
&\leqslant \varphi_2(s'_2)
\end{aligned}$$

- for each $s'_1 \in S_1$,

$$\begin{aligned}
&\sum_{s'_2 \in S_2} w_{\varepsilon+\gamma}(s'_1, s'_2) \\
&= \sum_{s'_2 \in S_2} \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) w_\gamma^{s_2,tr_1}(s'_1, s'_2) \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) \sum_{s'_2 \in S_2} w_\gamma^{s_2,tr_1}(s'_1, s'_2) \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) \mu_{tr}(s'_1) \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{tr_1} w_\varepsilon(src(tr_1), s_2) \mu_{tr}(s'_1)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{tr_1 \in D_1} p_{tr_1} \mu_{tr}(s'_1) \sum_{s_2 \in Supp(\mu_2)} w_\varepsilon(src(tr_1), s_2) \\
&\leqslant \sum_{tr_1 \in D_1} p_{tr_1} \mu_{tr}(s'_1) \mu_1(src(tr_1)) \\
&\leqslant \varphi_1(s'_1)
\end{aligned}
$$

– finally,

$$
\begin{aligned}
&\sum_{s'_1 \in S_2} \sum_{s'_2 \in S_2} w_{\varepsilon+\gamma}(s'_1, s'_2) \\
&= \sum_{s'_1 \in S_2} \sum_{s'_2 \in S_2} \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) w_\gamma^{s_2,tr_1}(s'_1, s'_2) \\
&= \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2) \sum_{s'_1 \in S_2} \sum_{s'_2 \in S_2} w_\gamma^{s_2,tr_1}(s'_1, s'_2) \\
&\geqslant \sum_{s'_1 \in S_2} \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{m(s_2,tr_1)} w_\varepsilon(src(tr_1), s_2)(1-\gamma) \\
&= (1-\gamma) \sum_{tr_1 \in D_1} \sum_{s_2 \in Supp(\mu_2)} p_{tr_1} w_\varepsilon(src(tr_1), s_2) \\
&= (1-\gamma) \sum_{tr_1 \in D_1} p_{tr_1} \sum_{s_2 \in Supp(\mu_2)} w_\varepsilon(src(tr_1), s_2) \\
&\geqslant (1-\gamma) \sum_{tr_1 \in D_1} p_{tr_1}(1-\varepsilon) \\
&= (1-\gamma)(1-\varepsilon) \sum_{tr_1 \in D_1} p_{tr_1} \\
&= (1-\gamma)(1-\varepsilon) \\
&= (1-\gamma-\varepsilon) + \gamma\varepsilon \\
&\geqslant 1-\gamma-\varepsilon
\end{aligned}
$$

This implies that $w_{\varepsilon+\gamma}$ is an ε-weighting function from φ_1 to φ_2 and thus $\varphi_1 \; \mathcal{L}_w(\mathcal{R}, \varepsilon+\gamma) \; \varphi_2$. □

Now we are able to define the new notion of state weak polynomially accurate simulation:

Definition 5.15. *Let $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ and $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ be two families of probabilistic automata; let $\mathcal{R}= \{\mathcal{R}_k\}_{k\in\mathbb{N}}$ be a family of relations such that, for each $k \in \mathbb{N}$, $\mathcal{R}_k$ is a relation from S_k^1 to S_k^2; let Poly be the set of positive polynomials over $\mathbb{N}$.*

We say that $\mathcal{R}$ *is a* state weak polynomially accurate simulation *from* $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ *to* $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ *if*

1. *for each* k*, it holds that* $\bar{s}_k^1 \; \mathcal{R}_k \; \bar{s}_k^2$*;*
2. *for each* $c \in \mathbb{N}$ *and* $p \in Poly$*, there exist* $l \in Poly$ *and* $\bar{k} \in \mathbb{N}$ *such that for each* $k > \bar{k}$*, for all probability measures* μ_1 *and* μ_2 *and for each* $\gamma \geqslant 0$*, if* μ_1 *is reached within* $p(k)$ *steps in* $\mathcal{A}_k^1$*, then there exists an* ε*-weighting function* w_γ *for* $\mu_1 \; \mathcal{L}_w(\mathcal{R}_k, \gamma) \; \mu_2$ *such that* $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg \; \mathcal{R}_k^{l(k)}(k^{-c}) \; s_2\} < k^{-c}$*.*

We write $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lessapprox_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ *if there exists a state weak polynomially accurate simulation* $\mathcal{R}$ *from* $\{\mathcal{A}_k^1\}_{k\in K}$ *to* $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$*.*

It is quite easy to see that the state weak polynomially accurate simulation relation is implied by the ordinary weak l-bounded simulation of probabilistic automata:

Proposition 5.16. *Let* $\mathcal{A}_n^1$ *and* $\mathcal{A}_n^2$ *be two automata parameterized on* $n \in \mathbb{N}$*.*

If there exists $l \in \mathbb{N}$ *such that for each* $n \in \mathbb{N}$ $\mathcal{A}_n^1 \preccurlyeq^l \mathcal{A}_n^2$*, then* $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lessapprox_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$*.*

Proof. Let $\bar{l} \in \mathbb{N}$ such that for each $n \in \mathbb{N}$, $\mathcal{A}_n^1 \preccurlyeq^{\bar{l}} \mathcal{A}_n^2$. Let $\mathcal{R}_n$ be the weak simulation that justifies $\mathcal{A}_n^1 \preccurlyeq^{\bar{l}} \mathcal{A}_n^2$. $\mathcal{R}= \{\mathcal{R}_k\}_{k\in\mathbb{N}}$ is a state weak polynomially accurate simulation from $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.

The condition on start states is trivially true, since by definition of ordinary simulation, it follows that for each $k \in \mathbb{N}$, $\bar{s}_k^1 \; \mathcal{R}_k \; \bar{s}_k^2$.

Suppose, for the sake of contradiction, that the step condition does not hold. This means that there exists $c \in \mathbb{N}$ and $p \in Poly$ such that for each $l \in Poly$ and $\bar{k} \in \mathbb{N}$ there exists $k > \bar{k}$ such that there exist μ_1, μ_2, and $\gamma \geqslant 0$ such that μ_1 is reached within $p(k)$ steps in $\mathcal{A}_k^1$ and for each ε-weighting function w_γ for $\mu_1 \; \mathcal{L}_w(\mathcal{R}_k, \gamma) \; \mu_2$ we have that $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg \; \mathcal{R}_k^{l(k)}(k^{-c}) \; s_2\} \geqslant k^{-c}$.

$s_1 \neg \; \mathcal{R}_k^{l(k)}(k^{-c}) \; s_2$ implies that either $s_1 \neg \; \mathcal{R}_k \; s_2$ or $s_1 \; \mathcal{R}_k \; s_2$ and there exists $tr_1 = (s_1, a, \rho_1) \in D_k^1$ such that for each weak $l(k)$-bounded combined transition $tr_2 = (s_2, a, \rho_2)$, $\rho_1 \neg \; \mathcal{L}_w(\mathcal{R}_k, k^{-c}) \; \rho_2$. In the former case, $s_1 \neg \; \mathcal{R}_k \; s_2$ implies that $w_\gamma(s_1, s_2) = 0$. Otherwise, if $w_\gamma(s_1, s_2) > 0$, then by property 1 of the definition of ε-weighting function, we have that $s_1 \; \mathcal{R}_k \; s_2$. So, $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg \; \mathcal{R}_k^{l(k)}(k^{-c}) \; s_2\} \geqslant k^{-c}$ is due to pair of states (s_1, s_2) such that $s_1 \; \mathcal{R}_k \; s_2$ and there exists $tr_1 = (s_1, a, \rho_1) \in D_k^1$

such that for each weak $l(k)$-bounded combined transition $tr_2 = (s_2, a, \rho_2)$, $\rho_1 \neg\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \rho_2$. Take a pair of such states, say (s_1, s_2). Since $\mathcal{A}_k^1 \preccurlyeq^{\bar{l}} \mathcal{A}_k^2$, $s_1\ \mathcal{R}_k\ s_2$ and $s_1 \xrightarrow{a} \rho_1$ implies that there exists a weak $\bar{l}$-bounded combined transition (s_2, a, ρ_2) such that $\rho_1\ \mathcal{L}(\mathcal{R}_k)\ \rho_2$. By Property 3.5(1) we have that $\rho_1\ \mathcal{L}(\mathcal{R}_k, 0)\ \rho_2$. This implies, by Property 3.5(2), that $\rho_1\ \mathcal{L}(\mathcal{R}_k, k^{-c})\ \rho_2$ and thus, by Proposition 5.3, that $\rho_1\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \rho_2$. Summing up, we have that for each pair of states (s_1, s_2) such that $w_\gamma(s_1, s_2) > 0$, it holds that $s_1\ \mathcal{R}_k(k^{-c})\ s_2$. Thus, $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg\ \mathcal{R}_k^{\bar{l}}(k^{-c})\ s_2\} = 0 < k^{-c}$. Take $l \in \mathit{Poly}$, such that for each $n \in \mathbb{N}$, $l(n) = \bar{l}$. This implies that $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg\ \mathcal{R}_k^{l}(k^{-c})\ s_2\} = 0 < k^{-c}$. Absurd. □

It is straightforward to prove that the state polynomially accurate simulation relation implies the state weak polynomially accurate simulation relation:

Proposition 5.17. *Let $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ and $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ be two families of automata. If $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$, then for each $l > 0$, $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lessapprox_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.*

Proof. Let $\{\mathcal{R}_k\}_{k\in\mathbb{N}}$ be a state polynomially accurate simulation that justifies $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$. Then $\{\mathcal{R}_k\}_{k\in\mathbb{N}}$ is also a state weak polynomially accurate simulation.

The condition on start states is trivially true, since by definition of state polynomially accurate simulation, it follows that for each $k \in \mathbb{N}$, $\bar{s}_k^1\ \mathcal{R}_k\ \bar{s}_k^2$.

For the step condition, fix $c \in \mathbb{N}$ and $p \in \mathit{Poly}$. Then by definition of state polynomially accurate simulation there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$, μ_1, μ_2 and $\gamma \geqslant 0$, if μ_1 is reached within $p(k)$ steps in $\mathcal{A}_k^1$ then there exists an ε-weighting function w_γ such that $\mu_1\ \mathcal{L}_w(\mathcal{R}_k, \gamma)\ \mu_2$ and $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg\ \mathcal{R}_k(k^{-c})\ s_2\} < k^{-c}$. Since each combined transition is also a weak 1-boundend combined transition, it follows that $\sum\{w_\gamma(s_1, s_2) \mid s_1 \neg\ \mathcal{R}_k^1(k^{-c})\ s_2\} < k^{-c}$ and thus the step condition is satisfied, where we choose $l \in \mathit{Poly}$ as the constant polynomial 1. □

The definition state polynomially accurate simulation satisfies few important properties. The first property is that sequences of n steps can be matched up to an error nk^{-c}; the second one is the compositionality; the third property is the execution correspondence.

Theorem 5.18. *Let $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ and $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ be two families of probabilistic automata such that $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lessapprox_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$. Let $\mathcal{R} = \{\mathcal{R}_k\}_{k\in\mathbb{N}}$ be a state weak polynomially accurate simulation from $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.*

For each $c \in \mathbb{N}$, $p \in Poly$, there exist $l \in Poly$ and $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$ and each scheduler σ_1 for $\mathcal{A}_k^1$, if μ_1 is the probability measure induced by σ_1 after n steps, $n \leqslant p(k)$, then there exists a scheduler σ_2 for $\mathcal{A}_k^2$ that reaches, after $nl(k)$ steps, a probability measure μ_2 such that $\mu_1 \; \mathcal{L}_w(\mathcal{R}_k, nk^{-c}) \; \mu_2$.

Proof. The proof is a classical inductive argument on the number of steps.

Case $n = 0$: for each $c \in \mathbb{N}$, $p \in Poly$, and $k \in \mathbb{N}$ after 0 steps, it is not possible to reach states different from start state of $\mathcal{A}_k^1$. This means that after 0 steps, reached measure is $\delta_{\bar{s}_1}$. Analogously, for $\mathcal{A}_k^2$ $\delta_{\bar{s}_2}$ is reached after 0 steps. Since $\bar{s}_1 \; \mathcal{R}_k \; \bar{s}_2$, we have that $\delta_{\bar{s}_1} \; \mathcal{L}_w(\mathcal{R}_k, 0) \; \delta_{\bar{s}_2}$ and thus $\delta_{\bar{s}_1} \; \mathcal{L}_w(\mathcal{R}_k, 0k^{-c}) \; \delta_{\bar{s}_2}$.

Case $n > 0$: fix $c \in \mathbb{N}$, $p \in Poly$. Let $l \in Poly$ and $\bar{k} \in \mathbb{N}$ be a value such that for each $k > \bar{k}$, $n < p(k)$. Let μ_1 and μ_2 be two measures such that there exist schedulers σ_1 and σ_2 such that μ_1 is reached via σ_1 after n steps, μ_2 is reached via σ_2 after $nl(k)$ steps, and $\mu_1 \; \mathcal{L}_w(\mathcal{R}_k, nk^{-c}) \; \mu_2$. Let $w_{nk^{-c}}$ be the ε-weighting function that justifies $\mu_1 \; \mathcal{L}_w(\mathcal{R}_k, nk^{-c}) \; \mu_2$. Suppose that there exists φ_1 such that $\mu_1 \longrightarrow \varphi_1$. Since $n < p(k)$ and μ_1 is reached after n steps, it follows that φ_1 is reached after $n + 1 \leqslant p(k)$ steps. By hypothesis, $\{\mathcal{A}_k^1\}_{k \in K} \lessapprox_s \{\mathcal{A}_k^2\}_{k \in K}$ and thus there exists $\bar{k}' \in \mathbb{N}$ such that for each $k > \bar{k}'$ $\sum\{w_{nk^{-c}}(s_1, s_2) \mid s_1 \neg \; \mathcal{R}_k^{l(k)}(k^{-c}) \; s_2\} < k^{-c}$ and this implies, by Proposition 5.14, that there exists φ_2 such that $\mu_2 \Longrightarrow^{l(k)} \varphi_2$ and $\varphi_1 \; \mathcal{L}_w(\mathcal{R}_k, nk^{-c} + k^{-c}) \; \varphi_2$, that is $\varphi_1 \; \mathcal{L}_w(\mathcal{R}_k, (n+1)k^{-c}) \; \varphi_2$ and φ_2 is reached after $nl(k) + l(k) = (n+1)l(k)$ steps in $\mathcal{A}_k^2$. □

We now turn to compositionality:

Theorem 5.19. *Let $\{\mathcal{A}_k^1\}_{k \in \mathbb{N}}$ and $\{\mathcal{A}_k^2\}_{k \in \mathbb{N}}$ be two families of automata such that $\{\mathcal{A}_k^1\}_{k \in \mathbb{N}} \lessapprox_s \{\mathcal{A}_k^2\}_{k \in \mathbb{N}}$.*

For each context $\mathcal{C}_k$ compatible with both $\mathcal{A}_k^1$ and $\mathcal{A}_k^2$,

$$\{\mathcal{A}_k^1 || \mathcal{C}_k\}_{k \in \mathbb{N}} \lessapprox_s \{\mathcal{A}_k^2 || \mathcal{C}_k\}_{k \in \mathbb{N}}$$

Proof. Let $\{\mathcal{R}_k\}_{k \in \mathbb{N}}$ be a state weak polynomially accurate simulation that justifies $\{\mathcal{A}_k^1\}_{k \in \mathbb{N}} \lessapprox_s \{\mathcal{A}_k^2\}_{k \in \mathbb{N}}$. Define $\{\mathcal{R}'_k\}_{k \in \mathbb{N}}$ as the family of relations $\mathcal{R}'_k \subseteq (S_k^1 \times S_k^c) \times (S_k^2 \times S_k^c)$ where S_k^c is the set of states of $\mathcal{C}_k$ such that for each measure μ_1 and μ_2, $\mu_1 \; \mathcal{L}_w(\mathcal{R}'_k, \gamma) \; \mu_2$ if and only if

– $\mu_1 \; \mathcal{L}_w(\mathcal{R}_k \times id_k, \gamma) \; \mu_2$,

– $\mu_1 \lceil \mathcal{A}_k^1 \; \mathcal{L}_w(\mathcal{R}_k, \gamma) \; \mu_2 \lceil \mathcal{A}_k^2$, and
– $\mu_1 \lceil \mathcal{C}_k = \mu_2 \lceil \mathcal{C}_k$.

We now show that $\{\mathcal{R}'_k\}_{k\in\mathbb{N}}$ is a state polynomially accurate from $\{\mathcal{A}_k^1 || \mathcal{C}_k\}_{k\in\mathbb{N}}$ to $\{\mathcal{A}_k^2 || \mathcal{C}_k\}_{k\in\mathbb{N}}$.

Condition on start states is trivially true: let $\bar{s}_k^c$ be the start state of $\mathcal{C}_k$. By hypothesis, we know that $\bar{s}_k^1 \; \mathcal{R}_k \; \bar{s}_k^2$ and thus $\delta_{\bar{s}_k^1} \; \mathcal{L}(\mathcal{R}_k) \; \delta_{\bar{s}_k^2}$. Since $\delta_{\bar{s}_k^c} = \delta_{\bar{s}_k^c}$, it follows that $(\delta_{\bar{s}_k^1} \times \delta_{\bar{s}_k^c}) \; \mathcal{L}(\mathcal{R}'_k) \; (\delta_{\bar{s}_k^2} \times \delta_{\bar{s}_k^c})$ and thus $(\bar{s}_k^1, \bar{s}_k^c) \; \mathcal{R}'_k \; (\bar{s}_k^2, \bar{s}_k^c)$.

For the step condition, let $c \in \mathbb{N}$ and $p \in \mathit{Poly}$ and suppose that there exists $l \in \mathit{Poly}$ and $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$, each $\gamma \geqslant 0$, and each measure $\mu_1 \in \mathit{Disc}(S_k^1 \times S_k^c)$ and $\mu_2 \in \mathit{Disc}(S_k^2 \times S_k^c)$, it holds that $\mu_1 \; \mathcal{L}_w(\mathcal{R}'_k, \gamma) \; \mu_2$. Let w'_γ be the ε-weighting function that justifies $\mu_1 \; \mathcal{L}_w(\mathcal{R}'_k, \gamma) \; \mu_2$. To prove the step condition, we must show that $\sum\{w'_\gamma((s_1, s_c), (s_2, s_c)) \mid (s_1, s_c) \neg \; \mathcal{R}_k'^{l(k)}(k^{-c}) \; (s_2, s_c)\} < k^{-c}$. In fact, denoted by $w_\gamma(s_1, s_2)$ the value $\sum_{s_c \in S_k^c} w'_\gamma((s_1, s_c), (s_2, s_c))$,

$$
\begin{aligned}
&\sum_{\substack{(s_1, s_c^1) \in S_k^1 \times S_k^c \\ (s_2, s_c^2) \in S_k^2 \times S_k^c}} \{w'_\gamma((s_1, s_c^1), (s_2, s_c^2)) \mid (s_1, s_c) \neg \; \mathcal{R}_k'^{l(k)}(k^{-c}) \; (s_2, s_c^2)\} \\
= &\sum_{\substack{(s_1, s_c) \in S_k^1 \times S_k^c \\ (s_2, s_c) \in S_k^2 \times S_k^c}} \{w'_\gamma((s_1, s_c), (s_2, s_c)) \mid (s_1, s_c) \neg \; \mathcal{R}_k'^{l(k)}(k^{-c}) \; (s_2, s_c)\} \\
= &\sum_{\substack{s_1 \in S_k^1 \\ s_2 \in S_k^2}} \{ \sum_{s_c \in S_k^c} w'_\gamma((s_1, s_c), (s_2, s_c)) \mid s_1 \neg \; \mathcal{R}_k^{l(k)}(k^{-c}) \; s_2\} \\
= &\sum_{\substack{s_1 \in S_k^1 \\ s_2 \in S_k^2}} \{w_\gamma(s_1, s_2) \mid s_1 \neg \; \mathcal{R}_k^{l(k)}(k^{-c}) \; s_2\} \\
< \; &k^{-c}
\end{aligned}
$$

where the first step is justified by the fact that for each $(s_1, s_c^1) \in S_k^1 \times S_k^c$ and each $(s_2, s_c^2) \in S_k^2 \times S_k^c$, if $s_c^1 \neq s_c^2$, then $w'_\gamma((s_1, s_c^1), (s_2, s_c^2)) = 0$; the second step is due to the contrapositive of Lemma 5.11; the third step by the definition of w_γ and the forth step by the step condition for $\mathcal{R}$ and the fact that w_γ is an ε-weighing function for $\mu_1 \lceil \mathcal{A}_k^1 \; \mathcal{L}_w(\mathcal{R}, \gamma) \; \mu_2 \lceil \mathcal{A}_k^2$. In fact, we have that

– the first condition is trivially satisfied: $w_\varepsilon(s_1, s_2) > 0$ implies that $\sum_{s_c \in S_k^c} w'_\gamma((s_1, s_c), (s_2, s_c)) > 0$ and thus there exists at least one $s_c \in S_k^c$

such that $w'_\gamma((s_1, s_c), (s_2, s_c)) > 0$. This implies that $(s_1, s_c)\ \mathcal{R}'_k\ (s_2, s_c)$, that is $(s_1, s_c)\ \mathcal{R}_k \times id_k\ (s_2, s_c)$ and thus $s_1\ \mathcal{R}_k\ s_2$;

– the second condition is also verified: for each $s_1 \in S^1_k$,

$$\begin{aligned}
\sum_{s_2 \in S^2_k} w_\gamma(s_1, s_2) &= \sum_{s_2 \in S^2_k} \sum_{s_c \in S^c_k} w'_\gamma((s_1, s_c), (s_2, s_c)) \\
&= \sum_{(s_2, s_c) \in S^2_k \times S^c_k} w'_\gamma((s_1, s_c), (s_2, s_c)) \\
&= \sum_{s'_c \in S^c_k} \sum_{(s_2, s_c) \in S^2_k \times S^c_k} w'_\gamma((s_1, s'_c), (s_2, s_c)) \\
&\leqslant \sum_{s'_c \in S^c_k} \mu_1(s_1, s'_c) \\
&= \mu_1 \lceil \mathcal{A}^1_k(s_1)
\end{aligned}$$

– as well as the third condition: for each $s_2 \in S^2_k$,

$$\begin{aligned}
\sum_{s_1 \in S^1_k} w_\gamma(s_1, s_2) &= \sum_{s_1 \in S^1_k} \sum_{s_c \in S^c_k} w'_\gamma((s_1, s_c), (s_2, s_c)) \\
&= \sum_{(s_1, s_c) \in S^1_k \times S^c_k} w'_\gamma((s_1, s_c), (s_2, s_c)) \\
&= \sum_{s'_c \in S^c_k} \sum_{(s_1, s_c) \in S^1_k \times S^c_k} w'_\gamma((s_1, s_c), (s_2, s'_c)) \\
&\leqslant \sum_{s'_c \in S^c_k} \mu_2(s_2, s'_c) \\
&= \mu_2 \lceil \mathcal{A}^2_k(s_2)
\end{aligned}$$

– finally, also the fourth condition is satisfied:

$$\begin{aligned}
\sum_{\substack{s_1 \in S^1_k \\ s_2 \in S^2_k}} w_\gamma(s_1, s_2) &= \sum_{\substack{s_1 \in S^1_k \\ s_2 \in S^2_k}} \sum_{s_c \in S^c_k} w'_\gamma((s_1, s_c), (s_2, s_c)) \\
&= \sum_{\substack{s_1 \in S^1_k \\ s_2 \in S^2_k \\ s_c \in S^c_k}} w'_\gamma((s_1, s_c), (s_2, s_c)) \\
&= \sum_{\substack{(s_1, s^1_c) \in S^1_k \times S^c_k \\ (s_2, s^2_c) \in S^2_k \times S^c_k}} w'_\gamma((s_1, s^1_c), (s_2, s^2_c)) \\
&\geqslant 1 - \gamma
\end{aligned}$$

This completes the proof, since we have that w_γ is an ε-weighing function for $\mu_1 \lceil \mathcal{A}_k^1 \; \mathcal{L}_w(\mathcal{R}, \gamma) \; \mu_2 \lceil \mathcal{A}_k^2$. □

Similarly to the case of polynomially accurate simulations, we are able to prove the

Theorem 5.20 (Execution Correspondence Theorem). *Let* $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$, $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$, ..., $\{\mathcal{A}_k^n\}_{k\in\mathbb{N}}$ *be* n *families of probabilistic automata such that there exist* $l \in \mathbb{N}$ *and* $n-1$ *families of relations* $\{\mathcal{R}_k^1\}_{k\in\mathbb{N}}$, $\{\mathcal{R}_k^2\}_{k\in\mathbb{N}}$, ..., $\{\mathcal{R}_k^{n-1}\}_{k\in\mathbb{N}}$ *such for each* $0 < i < n$, $\{\mathcal{R}_k^i\}_{k\in\mathbb{N}}$ *is a state weak polynomially accurate simulation from* $\{\mathcal{A}_k^i\}_{k\in\mathbb{N}}$ *to* $\{\mathcal{A}_k^{i+1}\}_{k\in\mathbb{N}}$.

For each $c \in \mathbb{N}$ *and* $p \in Poly$, *there exist* $l \in Poly$ *and* $\bar{k} \in \mathbb{N}$ *such that for each* $k > \bar{k}$ *and each probability measure* $\mu^1 \in Disc(S_k^1)$, *if* μ^1 *is reachable within* $p(k)$ *steps in* $\mathcal{A}_k^1$, *then there exists* $\mu^n \in Disc(S_k^n)$ *such that* μ^n *is reachable within* $l(k)^{n-1}p(k)$ *steps in* $\mathcal{A}_k^n$ *and* $\mu^1 \; \mathcal{L}(\mathcal{R}_k^1 \circ \ldots \circ \mathcal{R}_k^{n-1}, p(k)k^{-c}) \; \mu^n$.

Proof. Fix $c \in \mathbb{N}$, $p \in Poly$ and suppose that there exists $\bar{k}' \in \mathbb{N}$ such that for each $k > \bar{k}'$, $\mu^1 \in Disc(S_k^1)$ is reachable within $p(k)$ steps in $\mathcal{A}_k^1$. Let s the actual number of steps performed to reach μ^1.

By Theorem 5.18, it follows that for each $c_1' \in \mathbb{N}$, there exist $l_1 \in Poly$ and $\bar{k}_1 \in \mathbb{N}$ such that for each $k > \bar{k}_1$, there exists a measure $\mu^2 \in Disc(S_k^2)$ such that μ^2 is reachable in $l_1(k)s$ steps and $\mu^1 \; \mathcal{L}_w(\mathcal{R}_k^1, sk^{-c_1'}) \; \mu^2$.

Since μ^2 is reachable in $\mathcal{A}_k^2$ in $l(k)s$ steps, it follows that for each $c_2' \in \mathbb{N}$ and $p_2 \in Poly$, there exist $l_2 \in Poly$ and $\bar{k}_2 \in \mathbb{N}$ such that for each $k > \bar{k}_2$, $l_1(k)s \leqslant p_2(k)$ and there exists a measure $\mu^3 \in Disc(S_k^3)$ such that μ^3 is reachable in $l_1(k)l_2(k)s$ steps and $\mu^2 \; \mathcal{L}_w(\mathcal{R}_k^2, l_1(k)sk^{-c_2'}) \; \mu^3$.

Iterating this reasoning, we obtain that for each $1 < i < n$ and each $c_i' \in \mathbb{N}$ and $p_i \in Poly$, there exist $l_i \in Poly$ and $\bar{k}_i \in \mathbb{N}$ such that for each $k > \bar{k}_i$, $(\prod_{j=1}^i l_j(k))s \leqslant p_i(k)$ and there exists a measure $\mu^{i+1} \in Disc(S_k^{i+1})$ such that μ^{i+1} is reachable in $(\prod_{j=1}^i l_j(k))s$ steps and $\mu^i \; \mathcal{L}_w(\mathcal{R}_k^i, (\prod_{j=1}^i l_j(k))sk^{-c_i'}) \; \mu^{i+1}$.

Let $\bar{k} = \max\{\bar{k}', \bar{k}_1, \ldots, \bar{k}_{n-1}\}$ and $\bar{l} \in Poly$ be a polynomial such that for each $k \in \mathbb{N}$, $\bar{l}(k) \geqslant \prod_{i=1}^{n-1} l_i(k)$. Since $\bar{k} \geqslant \bar{k}'$, for each $k > \bar{k}$ the measure μ^1 is still reachable within $\bar{l}(k)p(k)$ steps and $s \leqslant \bar{l}(k)p(k)$. Since for each $0 < i < n$ $\bar{k} \geqslant \bar{k}_i$, we still have that for each $c_i' \in \mathbb{N}$ and each $k > \bar{k}$, there exists a measure $\mu^{i+1} \in Disc(S_k^{i+1})$ such that μ^{i+1} is reachable in $\bar{l}(k)s$ steps and $\mu^i \; \mathcal{L}_w(\mathcal{R}_k^i, \bar{l}(k)sk^{-c_i'}) \; \mu^{i+1}$. Since for each $0 < i < n$ we can choose the value of c_i', let $c' \in \mathbb{N}$ be a value such that for each $u \in \mathbb{N}$,

$u > \bar{k}$, we have that $(n-1)\bar{l}(k)u^{-c'} \leqslant u^{-c}$; for each $0 < i < n$, take $c'_i = c'$. This implies that for each $k > \bar{k}$, there exists a measure $\mu^{i+1} \in Disc(S_k^{i+1})$ such that μ^{i+1} is reachable in $\bar{l}(k)s$ steps and $\mu^i \ \mathcal{L}_w(\mathcal{R}_k^i, \bar{l}(k)sk^{-c'}) \ \mu^{i+1}$. Propositions 3.6 and 5.3 imply that $\mu^1 \ \mathcal{L}(\mathcal{R}_k^1 \circ \ldots \circ \mathcal{R}_k^{n-1}, \sum_{i=1}^{n-1} \bar{l}(k)sk^{-c'})$ μ^n. Consider the error $\sum_{i=1}^{n-1} \bar{l}(k)sk^{-c'}$. We have that $\sum_{i=1}^{n-1} \bar{l}(k)sk^{-c'} = (n-1)\bar{l}(k)sk^{-c'} \leqslant sk^{-c} \leqslant p(k)k^{-c}$. Property 3.5(2) implies that the measure μ^n is reachable in $\bar{l}(k)s \leqslant \bar{l}(k)p(k)$ steps and $\mu^1 \ \mathcal{L}(\mathcal{R}_k^1 \circ \ldots \circ \mathcal{R}_k^{n-1}, p(k)k^{-c})$ μ^n, that implies, by Proposition 5.3, that $\mu^1 \ \mathcal{L}_w(\mathcal{R}_k^1 \circ \ldots \circ \mathcal{R}_k^{n-1}, p(k)k^{-c})$ μ^n, as required. □

Also the state weak polynomially accurate simulation is preserved by the hiding operation:

Proposition 5.21. *Let $\mathcal{A}_k$ and $\mathcal{B}_k$ be two families of automata such that $\{\mathcal{A}_k\}_{k\in\mathbb{N}} \lessapprox_s \{\mathcal{B}_k\}_{k\in\mathbb{N}}$. Let H be a set of actions.*

Then $\{\mathrm{Hide}_H(\mathcal{A}_k)\}_{k\in\mathbb{N}} \lessapprox_s \{\mathrm{Hide}_H(\mathcal{B}_k)\}_{k\in\mathbb{N}}$.

Proof. There is nothing to prove, since the definition of state weak polynomially accurate simulation does not care if an action is internal or external. □

6

Cryptographic Primitives and Simulations

In this chapter we will show how polynomially accurate simulations can be used to analyze cryptographic primitives. In particular, for each cryptographic component defined in Section 2.5, we provide an automaton that models the primitive and we show how to use polynomially accurate simulations to establish a relation between the concrete implementation of the component and its ideal counterpart.

6.1 Nonces

The first primitive we consider is nonce generation. We model a nonce generator with an automaton that provides two families of actions: a family of input actions $get\_nonce(A)$ and a family of output actions $ret\_nonce(A, n)$. A is an identifier of the entity that requires a nonce and it belongs to the set $\mathbb{A}$ that contains all entity's identifiers. Each state of the nonce generator is identified by the family of variables $value_A$ that assume values that belong to the union of the set of all nonces and the singleton $\{\perp\}$; the start state is the one where for each $A \in \mathbb{A}$, $value_A$ is $\perp$. The overall nonce generator automaton $NG_k(\mathbb{A})$ is depicted in Figure 6.1.

We do not require that elements of $\mathbb{A}$ must be of a particular type. They can be the identity of an agent, a combination of agent identities and session identifiers, integer values, and so on and so forth. In this way we abstract from the actual meaning of entity's identifier and thus we can define and study an automaton that represents a generic nonce generator that can be used each time we need to include a nonce generator inside the correctness proof we are carrying out.

Nonce Generator $NG_k(\mathbb{A})$

Signature:

Input:

$get\_nonce(A)$, $A \in \mathbb{A}$

Output:

$ret\_nonce(A, n)$, $n \in \{0,1\}^k$, $A \in \mathbb{A}$

State:

$value_A \in \{0,1\}^k \cup \{\perp\}$, initially $\perp$, $A \in \mathbb{A}$

Transitions:

Input $get\_nonce(A)$

Effect:

$value_A := v$ where $v \in_R \{0,1\}^k$

Output $ret\_nonce(A, n)$

Precondition:

$n = value_A$

Effect:

$value_A := \perp$

Fig. 6.1. The Nonce Generator

Input $get\_nonce(A)$

Effect:

$value_A := v$ where $v \in_R \{0,1\}^k$

Output $ret\_nonce(A, n)$

Precondition:

$n = value_A$

Effect:

$value_A := \perp$

The input action $get\_nonce(A)$ is used by entity identified by A to require a nonce. When $get\_nonce(A)$ occurs, a value v is chosen randomly and uniformly from the set $\{0,1\}^k$ that contains all nonces of length k. v is the chosen nonce and it is saved into the state variable $value_A$ that is designed to keep the value of the next nonce that we will return to A. Since $value_A$ is now different from $\perp$, then the $ret\_nonce(A, v)$ output action is enabled, the nonce v is returned to A and $value_A$ is set to $\perp$. We reset $value_A$ to its initial value $\perp$ to ensure that the action $ret\_nonce(A, n)$ is enabled only after an input $get\_nonce(A)$ and that when the nonce n is returned to A, it can not be sent to A again until A requires another nonce using the $get\_nonce(A)$ action. Note that we do not ensure that each nonce request is followed by the return of a nonce. In fact, suppose that the entity

A requires a nonce and then another nonce before the $ret\_nonce(A, v)$ action is performed: the second $get\_nonce(A)$ chooses another value v_n and it replaces the old value v of $value_A$ with v_n. This implies that $ret\_nonce(A, v)$ is not more enabled while $ret\_nonce(A, v_n)$ is enabled now. This means that the first nonce request has been lost and A receives only one nonce.

We think that the above situation is acceptable, since it is reasonable that when someone requires a nonce, then it waits until the nonce is generated and returned.

The nonce generator NG_k we defined and that is fully depicted in Figure 6.1 is one of the several models we can provide. For example, it is possible to define a nonce generator that returns exactly one nonce for each $get\_nonce(A)$ it receives: for example, we can define $value_A$ as a queue of nonces; the effect of $get\_nonce(A)$ is to enqueue the chosen value while $ret\_nonce(A, n)$ is enabled when the queue is not empty and n is equal to the head of the queue; the effect is to dequeue n. We can also define a simpler nonce generator which provides only one input action get_nonce and output actions $ret\_nonce(n)$ for $n \in \{0,1\}^k$. This automaton is simpler but it makes more complicated to define other automata that require nonces to the nonce generator. In fact, if there are two or more agents that interact with the nonce generator, then we must consider that the first agent (as well as all other agents) receives a nonce either when it has required a nonce or when another agent has required a nonce. In the first case, the received value is the expected one and it can be used; in the second case, the received value is not expected and thus it must be ignored by the agent. This implies that the agent must be able to decide when the input action must be considered and when it should be ignored. This result can be achieved using a state variable that represents a program counter that keeps track of the current step: if the program counter means "wait for a nonce", then the nonce is used and the program counter is updated; otherwise we ignore the received value and we do not change the current state.

6.1.1 Automaton and Properties of Nonces

In Section 2.5.2 we have seen that the main property of nonces is that for each $c \in \mathbb{N}$ and $p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$, if values $n_1, \ldots, n_{p(k)} \in \{0,1\}^k$ are given and a value n is chosen randomly from $\{0,1\}^k$, then $\mathbf{Pr}[n = n_i \mid 1 \leqslant i \leqslant p(k)] < k^{-c}$. In other words, if we choose randomly a nonce, then the probability that it belongs to a given set

of values is negligible. Since there are no conditions on how $n_1, \ldots, n_{p(k)}$ are chosen, they can be values chosen randomly in $\{0,1\}^k$ by the nonce generator itself.

Since repeated nonces occur with negligible probability, we can abstract from them and consider the case where nonces are never repeated: if we are sure that nonces are actually fresh, then we can simplify the proof of correctness of a protocol since we must not consider the case of repeated nonces and the probability of an attack in such case.

To be sure that the nonce generator does not choose nonces that are already returned in the past, we must modify it in such a way that nonces can not be repeated by construction.

Removing internal generated repeated nonces

We can be sure that generated nonces are not repeated in several ways: for example, we can collect all generated nonces into a set and then choose the next value from all possible nonces except for collected ones. Alternatively, we can keep a set of fresh nonces and each time a nonce is requested, it is chosen uniformly from such set and then it is removed.

We adopt the second approach and we obtain the desired result in two steps: first, we add to NG_k a state variables containing all fresh nonces and a family of variables that keep information about the freshness of the last generated value; second, we impose that nonces can be chosen only between fresh nonces.

For the first step, let $NG^1_k(\mathbb{A})$ be the automaton obtained from $NG_k(\mathbb{A})$ adding the state variable $\mathit{fresh\_nonces} \subseteq \{0,1\}^k$ with initial value $\{0,1\}^k$ and the family of state variables $\mathit{is\_fresh}_A \subseteq \{T, F, \bot\}$ each one with initial value $\bot$. Moreover, each $\mathit{get\_nonce}(A)$ action also updates $\mathit{fresh\_nonces}$ removing the chosen value v from $\mathit{fresh\_nonces}$ and $\mathit{is\_fresh}_A$ assigning value T or F whenever v belongs to $\mathit{fresh\_nonces}$ or not, respectively; each $\mathit{ret\_nonce}(A)(n)$ action also updates $\mathit{is\_fresh}_A$ resetting it to the initial value $\bot$, as depicted in Figure 6.2.

It is immediate to see that $NG^1_k(\mathbb{A})$ is an extension of $NG_k(\mathbb{A})$. In fact, we simply add some history variables that keep information about the freshness of nonces and all nonces not yet chosen. Such variables are updated internally and do not affect the choice of the next nonce. Moreover, they are updated in a "deterministic" way, that is, when $NG_k(\mathbb{A})$ reaches s' from s with probability p, then also $NG^1_k(\mathbb{A})$ reaches s'_1 from s_1

NG_k

Input $get\_nonce(A)$
Effect:
$value_A := v$ where $v \in_R \{0,1\}^k$

Output $ret\_nonce(A,n)$
Precondition:
$n = value_A$
Effect:
$value_A := \bot$

NG_k^1

Input $get\_nonce(A)$
Effect:
$value_A := v$ where $v \in_R \{0,1\}^k$
$is\_fresh_A := \begin{cases} T & \text{if } v \in fresh\_nonces \\ F & \text{otherwise} \end{cases}$
$fresh\_nonces := fresh\_nonces \setminus \{v\}$

Output $ret\_nonce(A,n)$
Precondition:
$n = value_A$
Effect:
$is\_fresh_A := \bot$
$value_A := \bot$

Fig. 6.2. $get\_nonce(A)$ and $ret\_nonce(A,n)$ of $NG_k(\mathbb{A})$ and $NG_k^1(\mathbb{A})$

with probability p where $s = s_1{\restriction}_s$ and $s' = s_1'{\restriction}_{s'}$. In fact, when NG_k performs a transition labelled by $ret\_nonce(A,n)$, it actually performs a transition $tr = (s, ret\_nonce(A,n), \delta_{s'})$ where $s.value_A = n$ and $s'.value_A = \bot$. NG_k also performs a transition $tr_1 = (s_1, ret\_nonce(A,n), \delta_{s_1'})$ where $s_1.value_A = n$ and $s_1'.value_A = \bot$. If we consider the variables we have added, we can note that $fresh\_nonces$ is not modified by $ret\_nonce(A,n)$ and that is_fresh_A is reset to $\bot$.

That is, $tr = tr_1{\restriction}_{tr}$.

Similarly, when NG_k performs a transition tr labelled by $get\_nonce(A)$, also NG_k^1 performs a transition tr_1 labelled by $get\_nonce(A)$ such that $tr = tr_1{\restriction}_{tr}$.

The above intuition is formalized by the following result:

Lemma 6.1. *Let $\mathbb{A}$ be a set of (identities of) agents and W be the set $\{fresh\_nonces\} \cup \{is\_fresh_A \mid A \in \mathbb{A}\}$ where variables are defined as above. For each $k \in \mathbb{N}$, $NG_k^1(\mathbb{A}) \in \mathrm{Ext}_\emptyset^W(NG_k(\mathbb{A}))$.*

Proof. To prove the statement of the Lemma, we need to check if the requirements of Definition 4.5 are satisfied:

compatible states: let v be a state variable of $NG_k^1(\mathbb{A})$. Then v is either $value_A$ for $A \in \mathbb{A}$ and thus it is a state variable of $NG_k(\mathbb{A})$, or it is *fresh_nonces* or is_fresh_A for some $A \in \mathbb{A}$ and thus $v \in W$;

compatible start state: $\bar{s}_k^1$ is identified by value $\perp$ for each $value_A$ and is_fresh_A (with $A \in \mathbb{A}$), and $\{0,1\}^k$ for *fresh_nonces*. Since $\bar{s}_k$ is identified by the value $\perp$ for each $value_A$ (with $A \in \mathbb{A}$), then $\bar{s}_k = \bar{s}_k^1{\restriction}_{\bar{s}_k}$;

compatible actions: $NG_k^1(\mathbb{A})$ and $NG_k(\mathbb{A})$ provides the same set of actions, so this condition is trivially verified; and

compatible transitions: let $tr_1 = (s_1, a, \mu_1) \in D_k^1$ be a transition of NG_k^1. Since A of NG_k^1 is equal to A of $NG_k(\mathbb{A})$ by definition of $NG_k^1(\mathbb{A})$, then we must verify that there exists a transition $tr = (s, a, \mu) \in D_k$ such that $tr = tr_1{\restriction}_{tr}$. There are two cases:

- $a = get\_nonce(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_nonce(A)$, it follows that for each $v \in \{0,1\}^k$, $\mu_1(s_1') = 2^{-k}$ where s_1' is the state of NG_k^1 such that $s_1'.value_A = v$, $s_1'.fresh\_nonces = s_1.fresh\_nonces \setminus \{v\}$, $s_1'.is\_fresh_A = T$ if $v \in s_1.fresh\_nonces$, F otherwise, and for each $B \in \mathbb{A}$, $B \neq A$, $s_1'.value_B = s_1.value_B$, and $s_1'.is\_fresh_B = s_1.is\_fresh_B$. Let s be the state of $NG_k(\mathbb{A})$ such that for each $B \in \mathbb{A}$, $s.value_B = s_1.value_B$. By definition of action $get\_nonce(A)$, s enables $get\_nonce(A)$ that leads to the measure μ such that for each $v \in \{0,1\}^k$, $\mu(s') = 2^{-k}$ where s' is the state of $NG_k(\mathbb{A})$ such that $s'.value_A = v$, and for each $B \in \mathbb{A}$, $B \neq A$, $s'.value_B = s.value_B$. Since $s = s_1{\restriction}_s$ and $\mu = \mu_1{\restriction}_\mu$, then $tr = tr'{\restriction}_{tr}$.
- $a = ret\_nonce(A, n)$ for some $A \in \mathbb{A}$ and $n \in \{0,1\}^k$: by definition of action $ret\_nonce(A, n)$, it follows that $s_1.value_A = n$ and that $\mu_1 = \delta_{s_1'}$ where s_1' is the state of $NG_k^1(\mathbb{A})$ such that $s_1'.fresh\_nonces = s_1.fresh\_nonces$, and $s_1'.value_A = \perp$, $s_1'.is\_fresh_A = \perp$, and for each $B \in \mathbb{A}$, $B \neq A$, $s_1'.value_B = s_1.value_B$, and $s_1'.is\_fresh_B = s_1.is\_fresh_B$. Let s be the state of $NG_k(\mathbb{A})$ such that $s.value_A = n$, and for each $B \in \mathbb{A}$, $B \neq A$, $s.value_B = s_1.value_B$. By definition of action $ret\_nonce(A, n)$, s enables $ret\_nonce(A, n)$ that leads to the measure $\delta_{s'}$ where s' is the state of $NG_k(\mathbb{A})$ such that $s'.value_A = \perp$,

for each $B \in \mathbb{A}$, $B \neq A$, $s'.value_B = s.value_B$. Since $s = s_1\lceil_s$ and $\delta_{s'} = \delta_{s'_1}\lceil_{\delta_{s'}}$, then $tr = tr'\lceil_{tr}$.

Since the requirements of Definition 4.5 are satisfied, for each $k \in \mathbb{N}$, $NG^1_k(\mathbb{A}) \in \mathrm{Ext}^W_\emptyset(NG_k(\mathbb{A}))$. □

The existence of a polynomially accurate simulation from $NG_k(\mathbb{A})$ to $NG^1_k(\mathbb{A})$ in now straightforward:

Proposition 6.2. *Let $\mathbb{A}$ be a set of (identities of) agents. For each context $\mathcal{C}_k$ compatible with $NG_k(\mathbb{A})$,*

$$\{NG_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim \{NG^1_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.1, for each $k \in \mathbb{N}$, $NG^1_k(\mathbb{A}) \in \mathrm{Ext}^W_B(NG_k(\mathbb{A}))$ where B is the empty set and W is $\{\mathit{fresh\_nonces}\} \cup \{\mathit{is\_fresh}_A \mid A \in \mathbb{A}\}$. This implies, by Lemma 4.6, that for each context C'_k compatible with $NG_k(\mathbb{A})$ such that $B \subseteq A_{C'_k}$, $NG_k(\mathbb{A})||C'_k \preceq NG^1_k(\mathbb{A})||C'_k$ and thus, by Proposition 5.6, $\{NG_k(\mathbb{A})||C'_k\}_{k\in\mathbb{N}} \lesssim_s \{NG^1_k(\mathbb{A})||C'_k\}_{k\in\mathbb{N}}$. Since $B = \emptyset$, each context $\mathcal{C}_k$ compatible with $NG_k(\mathbb{A})$ satisfies $B \subseteq A_{\mathcal{C}_k}$ and thus for each context $\mathcal{C}_k$ compatible with $NG_k(\mathbb{A})$, $\{NG_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{NG^1_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

For the second step, let $NG^2_k(\mathbb{A})$ be the automaton obtained from $NG^1_k(\mathbb{A})$ modifying each $\mathit{get\_nonce}(A)$ action as follows: v is chosen randomly from $\mathit{fresh\_nonces}$ instead of $\{0,1\}^k$, as depicted in Figure 6.3.

It is immediate to see that $NG^2_k(\mathbb{A})$ simulates the G_k-conditional of $NG^1_k(\mathbb{A})$ where for each $k \in \mathbb{N}$, G_k is the set of states of $NG^1_k(\mathbb{A})$ such that for each $A \in \mathbb{A}$, $\neq .\mathit{is\_fresh}_A F$. In fact, we simply restrict the support of the measure described by the $\mathit{get\_nonce}(A)$ action to the set $\mathit{fresh\_nonces}$, that is, we condition the choice of the next nonce to the fact that it is fresh (and thus, $\neq .\mathit{is\_fresh}_A F$ is always satisfied).

Note that $NG^2_k(\mathbb{A})$ is not the G_k-conditional of $NG^1_k(\mathbb{A})$ since it provides more transitions than $NG^1_k(\mathbb{A})|G_k$. In fact, $NG^2_k(\mathbb{A})$ enables transitions that leave from a state not in G_k and that leads to a probability measures μ such that $\mu(G_k) = 0$. For example, let s be the state such that $s.\mathit{is\_fresh}_A = F$, $s.\mathit{is\_fresh}_B = F$, $s.\mathit{value}_A = n$. Thus, by definition of $\mathit{ret\_nonce}$ actions, s enables the action $\mathit{ret\_nonce}(A,n)$ that leads to the measure $\delta_{s'}$ where in s' only value_A and $\mathit{is\_fresh}_A$ are different with respect to the values in s. Thus $s'.\mathit{is\_fresh}_B$ is still F and hence $s' \notin G_k$.

The above intuition is formalized by the following result:

NG_k^1

Input $get\_nonce(A)$
Effect:
$value_A := v$ where $v \in_R \{0,1\}^k$
$is\_fresh_A := \begin{cases} T & \text{if } v \in fresh\_nonces \\ F & \text{otherwise} \end{cases}$
$fresh\_nonces := fresh\_nonces \setminus \{v\}$

NG_k^2

Input $get\_nonce(A)$
Effect:
$value_A := v$ where $v \in_R fresh\_nonces$
$is\_fresh_A := \begin{cases} T & \text{if } v \in fresh\_nonces \\ F & \text{otherwise} \end{cases}$
$fresh\_nonces := fresh\_nonces \setminus \{v\}$

Fig. 6.3. $get\_nonce(A)$ of $NG_k^1(\mathbb{A})$ and $NG_k^2(\mathbb{A})$

Lemma 6.3. *Given $NG_k^1(\mathbb{A})$, let B_k be the set of states of NG_k^1 such that $s \in B_k$ if there exists $A \in \mathbb{A}$ such that $s.is\_fresh_A = F$. Let G_k be the set $S_k^1 \setminus B_k$.*

For each $k \in \mathbb{N}$, $NG_k^1(\mathbb{A})|G_k \preceq NG_k^2(\mathbb{A})$.

Proof. For each $k \in \mathbb{N}$, let id_k be the identity relation on states. id_k is a simulation from $NG_k^1(\mathbb{A})|G_k$ to $NG_k^2(\mathbb{A})$.

The condition on start states is trivially true: let $\bar{s}_k^1$ be the start state of $NG_k^1(\mathbb{A})|G_k$ and $\bar{s}_k^2$ be the start state of $NG_k^2(\mathbb{A})$. By definition of conditional, it follows that $\bar{s}_k^1$ is the start state of $NG_k^1(\mathbb{A})$. Since by definition of $NG_k^2(\mathbb{A})$, the only difference between $NG_k^1(\mathbb{A})$ and $NG_k^2(\mathbb{A})$ is on the definition of the action $get\_nonce(A)$, we have that $\bar{s}_k^2 = \bar{s}_k^1$ and thus $\bar{s}_k^1\ id_k\ \bar{s}_k^2$.

For the step condition, let s_1 and s_2 be two states of $NG_k^1(\mathbb{A})|G_k$ and $NG_k^2(\mathbb{A})$, respectively, such that $s_1\ id_k\ s_2$. Let (s_1, a, μ_1) be a transition of $NG_k^1(\mathbb{A})|G_k$ that leaves from s_1. We must find μ_2 such that (s_2, a, μ_2) is a transition of $NG_k^2(\mathbb{A})$ and $\mu_1\ \mathcal{L}(id_k)\ \mu_2$. There are two cases:

- $a = get\_nonce(A)$ for some $A \in \mathbb{A}$: by definition of $get\_nonce(A)$, it follows that μ_1 is the probability measure $\rho|G_k$ where ρ is the measure that for each $v \in \{0,1\}^k$, ρ assigns probability 2^{-k} to the state s_v such that $s_v.value_A = v$, $s_v.is\_fresh_A = T$ if $v \in s_1.fresh\_nonces$, F otherwise, $s_v.fresh\_nonces = s_1.fresh\_nonces \setminus \{v\}$, and for each $B \in \mathbb{A} \setminus \{A\}$,

$s_v.value_B = s_1.value_B$, and $s_v.is\_fresh_B = s_1.is\_fresh_B$. By definition of conditional measure, we know that for each $B \in \mathbb{A} \setminus \{A\}$, $s_1.is\_fresh_B \neq F$, otherwise if there exists $B \in \mathbb{A} \setminus \{A\}$ such that $s_1.is\_fresh_B = F$, then for all states s_v $s_v.is\_fresh_B = F$ and thus $\rho(G_k) = 0$. Denoted by FN_1 the set $s_1.fresh\_nonces$, it follows that μ_1 is the measure that for each $v \in FN_1$, μ_1 assigns probability $1/|FN_1|$ to the state s_v such that $s_v.value_A = v$, $s_v.is\_fresh_A = T$ $s_v.fresh\_nonces = FN_1 \setminus \{v\}$, and for each $B \in \mathbb{A} \setminus \{A\}$, $s_v.value_B = s_1.value_B$, and $s_v.is\_fresh_B = s_1.is\_fresh_B$.

By definition of $get\_nonce(A)$, also s_2 enables a transition labelled by $get\_nonce(A)$ that reaches a measure μ_2 such that, denoted by FN_2 the set $s_2.fresh\_nonces$ (that is the same of FN_1 since s_1 id_k s_2), for each $v \in FN_2$, μ_2 assigns probability $1/|FN_2|$ to the state s_v such that $s_v.value_A = v$, $s_v.is\_fresh_A = T$ $s_v.fresh\_nonces = FN \setminus \{v\}$, and for each $B \in \mathbb{A} \setminus \{A\}$, $s_v.value_B = s_2.value_B$, and $s_v.is\_fresh_B = s_2.is\_fresh_B$. Since for each $v \in FN_1 = FN_2$, $\mu_1(s_v) = \mu_2(s_v)$, then $\mu_1 \; \mathcal{L}(id_k) \; \mu_2$, as required.

– $a = ret\_nonce(A, n)$ for some $A \in \mathbb{A}$ and $n \in \{0,1\}^k$: by definition of the action $ret\_nonce(A, n)$, it follows that s_1 satisfies $s_1.value_A = n$. Moreover, it follows that $\mu_1 = \delta_{s_1'}$ where s_1' is the state such that $s_1'.value_A = \bot$, $s_1'.is\_fresh_A = \bot$, and all other variables that describe s_1' have the same value of the variables that describe s_1. Since s_1 id_k s_2, we have that also s_2 satisfies $s_2.value_A = n$ and thus it enables the transition $(s_2, ret\_nonce(A, n), \mu_2)$ where μ_2 is the measure $\delta_{s_2'}$ where s_2' is the state such that $s_2'.value_A = \bot$, $s_2'.is\_fresh_A = \bot$, and all other variables that describe s_2' have the same value of the variables that describe s_2. This implies that s_1' id_k s_2' and thus $\delta_{s_1'}$ $\mathcal{L}(id_k)$ $\delta_{s_2'}$, that is $\mu_1 \; \mathcal{L}(id_k) \; \mu_2$, as required.

Since for each action a, if s_1 enables a transition labelled by a that leads to μ_1, then we can find μ_2 such that $(s_2, a, \mu_2) \in D_2$ and $\mu_1 \; \mathcal{L}(id_k) \; \mu_2$, then the step condition is satisfied. □

The existence of a polynomially accurate simulation from $NG_k^1(\mathbb{A})|G_k$ to $NG_k^2(\mathbb{A})$ in now straightforward:

Proposition 6.4. *Let $\mathbb{A}$ be a set of agents. Given $NG_k^1(\mathbb{A})$, let B_k be the set of states of NG_k^1 such that $s \in B_k$ if there exists $A \in \mathbb{A}$ such that $s.is\_fresh_A = F$. Let G_k be the set $S_k^1 \setminus B_k$.*

For each context $\mathcal{C}_k$ compatible with $NG_k^1(\mathbb{A})$,

$$\{(NG^1_k(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{NG^2_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.3, we have that for each $k \in \mathbb{N}$, $NG^1_k(\mathbb{A})|G_k \preceq NG^2_k(\mathbb{A})$. This implies, by compositionality of $\preceq$, that for each context $\mathcal{C}_k$ compatible with $NG^1_k(\mathbb{A})$, $NG^1_k(\mathbb{A})|G_k||\mathcal{C}_k \preceq NG^2_k(\mathbb{A})||\mathcal{C}_k$. This implies, by Proposition 5.6, that $\{NG^1_k(\mathbb{A})|G_k||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{NG^2_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

To complete the chain of simulations from $NG_k(\mathbb{A})$ to $NG^2_k(\mathbb{A})$, we need to prove that

Proposition 6.5. *Let $\mathbb{A}$ be a set of agents. Given $NG^1_k(\mathbb{A})$, let B_k be the set of states of NG^1_k such that $s \in B_k$ if there exists $A \in \mathbb{A}$ such that $s.is\_fresh_A = F$. Let G_k be the set $S^1_k \setminus B_k$.*

For each context $\mathcal{C}_k$ compatible with $NG^1_k(\mathbb{A})$,

$$\{NG^1_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(NG^1_k(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. Let G_k be the set of states of NG^1_k such that $s \in G_k$ if and only if for each $A \in \mathbb{A}$ $s.is\_fresh_A \neq F$, that is, the chosen value is fresh. Let B_k be $S^1_k \setminus G_k$ (that is, states s' of NG^1_k such that $s'.is\_fresh_A = F$ for some $A \in \mathbb{A}$).

Theorem 5.8 states that $\{NG^1_k(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{NG^1_k(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$ if and only if $\{B_k\}_{k\in\mathbb{N}}$ is negligible in $\{NG^1_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$.

Suppose, for the sake of contradiction, that $\{NG^1_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$ is not simulated by $\{(NG^1_k(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$. This implies that $\{B_k\}_{k\in\mathbb{N}}$ is not negligible in $\{NG^1_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$ and thus that there exists $c \in \mathbb{N}$, $p \in \mathit{Poly}$ such that for each $\bar{k} \in \mathbb{N}$ there exists $k > \bar{k}$ such the probability to reach states of B_k within $p(k)$ steps is at least k^{-c}, that is, the probability to reach states s such that $s.is\_fresh_A = F$ for some $A \in \mathbb{A}$ within $p(k)$ steps is at least k^{-c}. By definition of the automaton $NG^1_k(\mathbb{A})$, it follows that $s.is\_fresh_A$ can assume value F only as the effect of a transition that leaves from some state s' and that is labelled by action $get\_nonce(A)$. This happens only when the randomly chosen value v assigned to $s.value_A$ does not belong to the set $FN = s'.fresh\_nonces$. Within $p(k)$ steps, we can perform at most $p(k)$ transitions labelled by $get\_nonce(Z)$ with $Z \in \mathbb{A}$. Since by definition of $get\_nonce(Z)$ we remove at most one value from $fresh\_nonces$ each time we perform a transition labelled by $get\_nonce(Z)$, within $p(k)$ steps the set $RN = \{0,1\}^k \setminus FN$ has cardinality at most $p(k)$. This means that with probability at least k^{-c}, we have chosen randomly in $\{0,1\}^k$ a value v that

it is equal to some $n \in RN$ with $|RN| \leqslant p(k)$. This contradicts the Proposition 2.2 and thus $\{NG_k^1(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{NG_k^1(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$. By Theorem 5.10, it follows that $\{NG_k^1(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(NG_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

Avoiding collisions with externally generated nonces

As we have seen above, there exists a simulation from the nonce generator that can generate repeated nonces to the nonce generator that ensures that returned nonces are never repeated. Sometimes, the fact that NG_k^2 does not generate repeated nonces is not sufficient to satisfy the required properties: it is still possible that NG_k^2 generates a nonce that is equal to some adversary's generated nonce.

If we want to be sure that the nonce generator returns values that have never occurred previously, then we must modify the automaton adding the knowledge of environment's generated nonces. We can provide such knowledge using one of the following approaches: the nonce generator receives as input sets of environment's generated nonces, or it receives messages. In the second approach, given a message m, we can easily extract nonces from m when it is a plaintext or a signature (since we can recover the signed text from the signature itself), but when m is a ciphertext, we must decrypt it to know which nonces it contains. This means that we need also the decryption keys associated to used encryption keys or some other way that extracts nonces from encrypted messages.

We do not consider this second approach since we do not think it is reasonable: it seems to be quite strange that a nonce generator requires private decryption keys to generate nonces.

So we consider only the first approach, that is also simpler than the second one since it do not require to parse messages and to open encryptions. This means that we provide the nonce generator with the nonces generated by the environment, that is, if the environment chooses nonces $n_1, \ldots, n_l$, then we send the set $\{n_1, \ldots, n_l\}$ to the generator. We do this adding a state variable *used_nonces* that contains all nonces used by the environment and an input action $used\_nonces(N)$ that updates *used_nonces*:

Input $used\_nonces(N)$
  Effect:
    $used\_nonces := used\_nonces \cup N$

Moreover, we add a family of variables $is\_not\_used_A \in \{T, F, \perp\}$ with initial value $\perp$, one for each $A \in \mathbb{A}$. Then we modify the $get\_nonce(A)$ and $ret\_nonce(A, n)$ as follows:

NG_k^2

Input $get\_nonce(A)$
  Effect:
    $value_A := v$ where $v \in_R \{0,1\}^k$
    $is\_fresh_A := \begin{cases} T & \text{if } v \in fresh\_nonces \\ F & \text{otherwise} \end{cases}$
    $fresh\_nonces := fresh\_nonces \setminus \{v\}$

Output $ret\_nonce(A, n)$
  Precondition:
    $n = value_A$
  Effect:
    $is\_fresh_A := \perp$
    $value_A := \perp$

NG_k^3

Input $get\_nonce(A)$
  Effect:
    $value_A := v$ where $v \in_R \{0,1\}^k$
    $is\_fresh_A := \begin{cases} T & \text{if } v \in fresh\_nonces \\ F & \text{otherwise} \end{cases}$
    $is\_not\_used_A := \begin{cases} F & \text{if } v \in used\_nonces \\ T & \text{otherwise} \end{cases}$
    $fresh\_nonces := fresh\_nonces \setminus \{v\}$

Output $ret\_nonce(A, n)$
  Precondition:
    $n = value_A$
  Effect:
    $is\_fresh_A := \perp$
    $is\_not\_used_A := \perp$
    $value_A := \perp$

Let $NG_k^3(\mathbb{A})$ be the resulting automaton. It is immediate to see that $NG_k^3(\mathbb{A})$ is an extension of $NG_k^2(\mathbb{A})$. In fact, we simply add an history variable that keeps information about all nonces chosen by the environment. Such variable is updated only by $used\_nonces(N)$ and it does not affect the choice of the next nonce and thus each transition of $NG_k^3(\mathbb{A})$ either is identical to a transition of $NG_k^2(\mathbb{A})$ or it is the new action that does

not modify the state variables that describe states of $NG_k^2(\mathbb{A})$. Moreover, we add a family of variables $is\_not\_used_A$ that are updated internally by $NG_k^3(\mathbb{A})$ and that do not change the behavior of the automaton.

The above intuition is formalized by the following result:

Lemma 6.6. *Let $\mathbb{A}$ be a set of agents, B be the set $\{used\_nonces(N) \mid N \subseteq \{0,1\}^k\}$ and W be the set $\{used\_nonces\} \cup \{is\_not\_used_A \mid A \in \mathbb{A}\}$ where actions and variables are defined as above. For each $k \in \mathbb{N}$, $NG_k^3(\mathbb{A}) \in \mathrm{Ext}_B^W(NG_k^2(\mathbb{A}))$.*

Proof. To prove the statement of the Lemma, we need to check if the requirements of Definition 4.5 are satisfied:

compatible states: let v be a state variable of $NG_k^3(\mathbb{A})$. Then v is either in $\{value_A, fresh\_nonces, is\_fresh_A\}$ for $A \in \mathbb{A}$ and thus it is a state variable of $NG_k^2(\mathbb{A})$, or it is $used\_nonces$ or $is\_not\_used_A$ for some $A \in \mathbb{A}$ and thus $v \in W$;

compatible start state: $\bar{s}_k^3$ is identified by values $\perp$ for each variable $value_A$, is_fresh_A and $is\_not\_used_A$ (with $A \in \mathbb{A}$), $\{0,1\}^k$ for $fresh\_nonces$, and $\emptyset$ for $used\_nonces$. Since $\bar{s}_k^2$ is identified by the value $\perp$ for each $value_A$ and is_fresh_A (with $A \in \mathbb{A}$), and by $\{0,1\}^k$ for $fresh\_nonces$, then $\bar{s}_k^2 = \bar{s}_k^3{\restriction}_{\bar{s}_k^2}$;

compatible actions: each action a of $NG_k^3(\mathbb{A})$ is either a $get\_nonce(A)$, $ret\_nonce(A,n)$ (and thus it is an action of $NG_k^2(\mathbb{A})$) or it is the action $used\_nonces(N)$ and thus it belongs to B; and

compatible transitions: let $tr_3 = (s_3, a, \mu_3) \in D_k^3$ be a transition of NG_k^3. We must find $tr_2 = (s_2, a, \mu_2) \in D_k^2$ such that $tr_2 = tr_3{\restriction}_{tr_2}$ when a is an action of NG_k^2 or the measure δ_{s_2} satisfies $\delta_{s_2} = \mu_3{\restriction}_{\delta_{s_2}}$. There are two cases:

- $a = get\_nonce(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_nonce(A)$, it follows that for each $v \in fresh\_nonces$, $\mu_3(s_3') = 1/|fresh\_nonces|$ where s_3' is the state of NG_k^3 such that $s_3'.value_A = v$, $s_3'.fresh\_nonces = s_3.fresh\_nonces \setminus \{v\}$, $s_3'.is\_not\_used_A = F$ if $v \in s_3.used\_nonces$, T otherwise, $s_3'.is\_fresh_A = T$ if v belongs to $s_3.fresh\_nonces$, F otherwise, $s_3'.used\_nonces = s_3.used\_nonces$, and for each $B \in \mathbb{A}$, $B \neq A$, $s_3'.value_B = s_3.value_B$, $s_3'.is\_fresh_B = s_3.is\_fresh_B$, and $s_3'.is\_not\_used_B = s_3.is\_not\_used_B$. Let s_2 be a state of $NG_k^2(\mathbb{A})$ such that $s_2 = s_3{\restriction}_{s_2}$. By definition of action $get\_nonce(A)$, s_2 enables $get\_nonce(A)$ that leads to the measure μ_2 such that for each $v \in fresh\_nonces$, $\mu_2(s_2') = 1/|fresh\_nonces|$ where

s'_2 is the state of $NG^2_k(\mathbb{A})$ such that $s'.value_A = v$, $s'_3.fresh\_nonces = s_3.fresh\_nonces \setminus \{v\}$, $s'_3.is\_fresh_A = T$ if $v \in s_3.fresh\_nonces$, F otherwise, and for each $B \in \mathbb{A}$, $B \neq A$, $s'_3.value_B = s_3.value_B$, and $s'_3.is\_fresh_B = s_3.is\_fresh_B$. This means that $\mu_2 = \mu_3\restriction_{\mu_2}$, and thus $tr_2 = tr_3\restriction_{tr_2}$.

- $a = ret\_nonce(A, n)$ for some $A \in \mathbb{A}$ and $n \in \{0,1\}^k$: by definition of action $ret\_nonce(A, n)$, it follows that $s_3.value_A = n$ and that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state of $NG^3_k(\mathbb{A})$ such that $s'_3.fresh\_nonces = s_3.fresh\_nonces$, and $s'_3.used\_nonces = s_3.used\_nonces$, $s'_3.value_A = \bot$, $s'_3.is\_not\_used_A = \bot$, $s'_3.is\_fresh_A = \bot$, and for each $B \in \mathbb{A}$, $B \neq A$, $s'_3.value_B = s_3.value_B$, $s'_3.is\_fresh_B = s_3.is\_fresh_B$, and $s'_3.is\_not\_used_B = s_3.is\_not\_used_B$. Let s_2 be the state of $NG^2_k(\mathbb{A})$ such that $s_2 = s_3\restriction_{s_2}$. By definition of action $ret\_nonce(A, n)$, s_2 enables $ret\_nonce(A, n)$ that leads to the measure $\delta_{s'_2}$ where s'_2 is the state of $NG^2_k(\mathbb{A})$ such that $s'_2.value_A = \bot$, $s'_2.is\_fresh_A = \bot$, $s'_2.fresh\_nonces = s_2.fresh\_nonces$, and for each $B \in \mathbb{A}$, $B \neq A$, $s'_2.value_B = s_2.value_B$ and $s'_2.is\_fresh_B = s_2.is\_fresh_B$. Since $s_2 = s_3\restriction_{s_2}$ and $\delta_{s'_2} = \delta_{s'_3}\restriction_{\delta_{s'_2}}$, then $tr_2 = tr'_3\restriction_{tr_2}$;
- $a = used\_nonces(N)$ for some $N \subseteq \{0,1\}^k$: by definition of action $used\_nonces(N)$, it follows that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state of $NG^3_k(\mathbb{A})$ that is identical to s_3 except for the $used\_nonces$ variable where $s'_3.used\_nonces = s_3.used\_nonces \cup N$. Let s_2 be a state of NG^2_k such that $s_2 = s_3\restriction_{s_2}$. Thus, $\delta_{s_2} = \mu_3\restriction_{\delta_{s_2}}$.

Since the requirements of Definition 4.5 are satisfied, for each $k \in \mathbb{N}$, $NG^3_k(\mathbb{A}) \in \mathrm{Ext}^W_B(NG^2_k(\mathbb{A}))$. □

The existence of a polynomially accurate simulation from $NG^2_k(\mathbb{A})$ to $NG^3_k(\mathbb{A})$ in now straightforward:

Proposition 6.7. *Let $\mathbb{A}$ be a set of (identities of) agents and let $\mathcal{C}_k$ be a context compatible with $NG^2_k(\mathbb{A})$ such that $\{used\_nonces(N) \mid N \subseteq \{0,1\}^k\} \subseteq A_{\mathcal{C}_k}$. Then,*

$$\{NG^2_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{NG^3_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.6, for each $k \in \mathbb{N}$, $NG^3_k(\mathbb{A}) \in \mathrm{Ext}^W_B(NG^2_k(\mathbb{A}))$ where B be the set $\{used\_nonces(N) \mid N \subseteq \{0,1\}^k\}$ defined as above and W be the set of variables $\{used\_nonces\} \cup \{is\_not\_used_A \mid A \in \mathbb{A}\}$ that are defined as above. This implies, by Lemma 4.6, that for each context $\mathcal{C}_k$ compatible

with $NG_k^2(\mathbb{A})$ such that $B \subseteq A_{\mathcal{C}_k}$, $NG_k^2(\mathbb{A})||\mathcal{C}_k \preceq NG_k^3(\mathbb{A})||\mathcal{C}_k$ and thus, by Proposition 5.6, $\{NG_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{NG_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

To take into account external used nonces, the action $get\_nonce(A)$ is modified as follows: instead of choosing the next value v randomly in $fresh\_nonces$, it is chosen randomly in $fresh\_nonces \setminus used\_nonces$. Let $NG_k^4(\mathbb{A})$ be the resulting automata.

It is immediate to see that $NG_k^4(\mathbb{A})$ simulates the G_k-conditional of $NG_k^3(\mathbb{A})$ where for each $k \in \mathbb{N}$, G_k is the set of states of $NG_k^3(\mathbb{A})$ such that for each $A \in \mathbb{A}$, $\neq .is\_not\_used_A F$. In fact, we simply restrict the support of the measure described by the $get\_nonce(A)$ action to the set $fresh\_nonces \setminus used\_nonces$, that is, we condition the choice of the next nonce to the fact that it is not one of the values chosen by the context (and thus, $\neq .is\_not\_used_A F$ is always satisfied).

Note that $NG_k^4(\mathbb{A})$ is not the G_k-conditional of $NG_k^3(\mathbb{A})$ since it provides more transitions than $NG_k^3(\mathbb{A})|G_k$. In fact, $NG_k^4(\mathbb{A})$ enables transitions that leave from a state not in G_k and that leads to a probability measures μ such that $\mu(G_k) = 0$. For example, let s be the state such that $s.is\_not\_used_A = F$, $s.is\_not\_used_B = F$, $s.value_A = n$. Thus, by definition of ret_nonce actions, s enables the action $ret\_nonce(A, n)$ that leads to the measure $\delta_{s'}$ where in s' only $value_A$, is_fresh_A and $is\_not\_used_A$ are different with respect to the values in s. Thus $s'.is\_not\_used_B$ is still F and hence $s' \notin G_k$.

The above intuition is formalized by the following result:

Lemma 6.8. *Given $NG_k^3(\mathbb{A})$, let B_k be the set of states of NG_k^3 such that $s \in B_k$ if there exists $A \in \mathbb{A}$ such that $s.is\_not\_used_A = F$. Let G_k be the set $S_k^3 \setminus B_k$.*

For each $k \in \mathbb{N}$, $NG_k^3(\mathbb{A})|G_k \preceq NG_k^4(\mathbb{A})$.

Proof. For each $k \in \mathbb{N}$, let id_k be the identity relation on states. id_k is a simulation from $NG_k^3(\mathbb{A})|G_k$ to $NG_k^4(\mathbb{A})$.

The condition on start states is trivially true: let $\bar{s}_k^3$ be the start state of $NG_k^3(\mathbb{A})|G_k$ and $\bar{s}_k^4$ be the start state of $NG_k^4(\mathbb{A})$. By definition of conditional, it follows that $\bar{s}_k^3$ is the start state of $NG_k^3(\mathbb{A})$. Since by definition of $NG_k^4(\mathbb{A})$, the only difference between $NG_k^3(\mathbb{A})$ and $NG_k^4(\mathbb{A})$ is on the definition of the action $get\_nonce(A)$, we have that $\bar{s}_k^4 = \bar{s}_k^3$ and thus $\bar{s}_k^3\ id_k\ \bar{s}_k^4$.

For the step condition, let s_3 and s_4 be two states of $NG_k^3(\mathbb{A})|G_k$ and $NG_k^4(\mathbb{A})$, respectively, such that $s_3\ id_k\ s_4$. Let (s_3, a, μ_3) be a transition of $NG_k^3(\mathbb{A})|G_k$ that leaves from s_3. We must find μ_4 such that (s_4, a, μ_4) is a transition of $NG_k^4(\mathbb{A})$ and $\mu_3\ \mathcal{L}(id_k)\ \mu_4$. There are three cases:

– $a = \mathit{get\_nonce}(A)$ for some $A \in \mathbb{A}$: by definition of $\mathit{get\_nonce}(A)$, it follows that μ_3 is the probability measure $\rho|G_k$ where ρ is the measure that for each $v \in \mathit{fresh\_nonces}$, ρ assigns probability $1/|\mathit{fresh\_nonces}|$ to the state s_v such that $s_v.\mathit{value}_A = v$, $s_v.\mathit{is\_fresh}_A = T$ if $v \in s_3.\mathit{fresh\_nonces}$, F otherwise, $s_v.\mathit{is\_not\_used}_A = F$ if $v \in s_3.\mathit{used\_nonces}$, T otherwise, $s_v.\mathit{used\_nonces} = s_3.\mathit{used\_nonces}$, $s_v.\mathit{fresh\_nonces} = s_3.\mathit{fresh\_nonces} \setminus \{v\}$, and for each $B \in \mathbb{A}\setminus\{A\}$, $s_v.\mathit{value}_B = s_3.\mathit{value}_B$, $s_v.\mathit{is\_not\_used}_B = s_3.\mathit{is\_not\_used}_B$, and $s_v.\mathit{is\_fresh}_B = s_3.\mathit{is\_fresh}_B$. By definition of conditional measure, we know that for each $B \in \mathbb{A}\setminus\{A\}$, $s_3.\mathit{is\_not\_used}_B \neq F$, otherwise if there exists $B \in \mathbb{A} \setminus \{A\}$ such that $s_3.\mathit{is\_not\_used}_B = F$, then for all states s_v $s_v.\mathit{is\_not\_used}_B = F$ and thus $\rho(G_k) = 0$. Denoted by UN_3 the set $s_3.\mathit{used\_nonces}$ and by FN_3 the set $s_3.\mathit{fresh\_nonces}$, it follows that μ_3 is the measure that for each $v \in FN_3 \setminus UN_3$, μ_3 assigns probability $1/|FN_3 \setminus UN_3|$ to the state s_v such that $s_v.\mathit{value}_A = v$, $s_v.\mathit{is\_fresh}_A = T$ if $v \in s_3.\mathit{fresh\_nonces}$, F otherwise, $s_v.\mathit{is\_not\_used}_A = F$ if $v \in s_3.\mathit{used\_nonces}$, T otherwise, $s_v.\mathit{used\_nonces} = s_3.\mathit{used\_nonces}$, $s_v.\mathit{fresh\_nonces} = s_3.\mathit{fresh\_nonces} \setminus \{v\}$, and for each $B \in \mathbb{A} \setminus \{A\}$, we have that $s_v.\mathit{value}_B = s_3.\mathit{value}_B$, $s_v.\mathit{is\_not\_used}_B = s_3.\mathit{is\_not\_used}_B$, and $s_v.\mathit{is\_fresh}_B = s_3.\mathit{is\_fresh}_B$.
By definition of $\mathit{get\_nonce}(A)$, also s_4 enables a transition labelled by $\mathit{get\_nonce}(A)$ that reaches a measure μ_4 such that, denoted by FN_4 the set $s_4.\mathit{fresh\_nonces}$ and by UN_4 the set $s_4.\mathit{used\_nonces}$, for each $v \in FN_4 \setminus UN_4$, measure μ_4 assigns probability $1/|FN_4 \setminus UN_4|$ to the state s_v such that $s_v.\mathit{value}_A = v$, $s_v.\mathit{is\_fresh}_A = T$, $s_v.\mathit{is\_not\_used}_A = T$, $s_v.\mathit{fresh\_nonces} = FN_4 \setminus \{v\}$, $s_v.\mathit{used\_nonces} = UN_4$, and for each $B \in \mathbb{A}\setminus\{A\}$, $s_v.\mathit{value}_B = s_4.\mathit{value}_B$, $s_v.\mathit{is\_not\_used}_B = s_4.\mathit{is\_not\_used}_B$, and $s_v.\mathit{is\_fresh}_B = s_4.\mathit{is\_fresh}_B$. Since for each $v \in FN_3\setminus UN_3 = FN_4\setminus UN_4$, $\mu_3(s_v) = \mu_4(s_v)$, then $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
– $a = \mathit{ret\_nonce}(A, n)$ for some $A \in \mathbb{A}$ and $n \in \{0,1\}^k$: by definition of the action $\mathit{ret\_nonce}(A, n)$, it follows that s_3 satisfies $s_3.\mathit{value}_A = n$. Moreover, it follows that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state such that $s'_3.\mathit{value}_A = \bot$, $s'_3.\mathit{is\_fresh}_A = \bot$, $s'_3.\mathit{is\_not\_used}_A = \bot$, and all other variables that describe s'_3 have the same value of the variables that describe s_3. Since $s_3\ id_k\ s_4$, we have that also s_4 satisfies $s_4.\mathit{value}_A = n$ and thus it enables the transition $(s_4, \mathit{ret\_nonce}(A, n), \mu_4)$ where μ_4 is the measure $\delta_{s'_4}$ where s'_4 is the state such that $s'_4.\mathit{value}_A = \bot$, $s'_4.\mathit{is\_fresh}_A = \bot$, $s'_4.\mathit{is\_not\_used}_A = \bot$, and all other variables that describe s'_4 have the same value of the variables that describe s_4. This

implies that $s'_3\ id_k\ s'_4$ and thus $\delta_{s'_3}\ \mathcal{L}(id_k)\ \delta_{s'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.

– $a = used\_nonces(N)$ for some $N \subseteq \{0,1\}^k$: by definition of action $used\_nonces(N)$, it follows that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state such that $s'_3.used\_nonces = s_3.used\_nonces \cup N$ and all other variables that describe s'_3 have the same value of the variables that describe s_3. Also s_4 enables a transition $(s_4, used\_nonces(N), \mu_4)$ where $\mu_4 = \delta_{s'_4}$ where s'_4 is the state such that $s'_4.used\_nonces = s_4.used\_nonces \cup N$ and all other variables that describe s'_4 have the same value of the variables that describe s_4. Since by hypothesis $s_3\ id_k\ s_4$, it follows that $s_3.used\_nonces = s_4.used\_nonces$ and thus $s'_3.used\_nonces = s'_4.used\_nonces$. This implies that $s'_3\ id_k\ s'_4$ and thus $\delta_{s'_3}\ \mathcal{L}(id_k)\ \delta_{s'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.

Since for each action a, if s_3 enables a transition labelled by a that leads to μ_3, then we can find μ_4 such that $(s_4, a, \mu_4) \in D_4$ and $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, then the step condition is satisfied. □

The existence of a polynomially accurate simulation from $NG^3_k(\mathbb{A})|G_k$ to $NG^4_k(\mathbb{A})$ in now straightforward:

Proposition 6.9. *Given $NG^3_k(\mathbb{A})$, let B_k be the set of states of NG^3_k such that $s \in B_k$ if there exists $A \in \mathbb{A}$ such that $s.is\_not\_used_A = F$. Let G_k be the set $S^3_k \setminus B_k$.*

For each context $\mathcal{C}_k$ compatible with $NG^3_k(\mathbb{A})$,

$$\{(NG^3_k(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{NG^4_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.8, we have that for each $k \in \mathbb{N}$, $NG^3_k(\mathbb{A})|G_k \preceq NG^4_k(\mathbb{A})$. This implies, by compositionality of $\preceq$, that for each context $\mathcal{C}_k$ compatible with $NG^3_k(\mathbb{A})$, $(NG^3_k(\mathbb{A})|G_k)||\mathcal{C}_k \preceq NG^4_k(\mathbb{A})||\mathcal{C}_k$. This implies, by Proposition 5.6, that $\{(NG^3_k(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{NG^4_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

To complete the chain of simulations from $NG^2_k(\mathbb{A})$ to $NG^3_k(\mathbb{A})$, we need to prove that

Proposition 6.10. *Let $\mathbb{A}$ be a set of (identities of) agents. Let G_k be the set of states of NG^3_k such that $s \in G_k$ if and only if for each $A \in \mathbb{A}$ $s.is\_not\_used_A \neq F$. Let B_k be $S^3_k \setminus G_k$. For each context $\mathcal{C}_k$ compatible with NG^3_k, if there exists $q \in Poly$ such that for each action $used\_nonces(N)$ of $\mathcal{C}_k$, $|N| < q(k)$, then*

$$\{NG^3_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim \{(NG^3_k(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. Theorem 5.8 states that B_k is negligible in NG_k^3 if and only if $\{NG_k^3(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{NG_k^3(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$.

Suppose, for the sake of contradiction, that $\{NG_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$ is not simulated by $\{(NG_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$. This implies that $\{B_k\}_{k\in\mathbb{N}}$ is not negligible in $\{NG_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$, that is, the probability of reaching states of B_k in a polynomial number of steps is not negligible. Since for each step of $NG_k^2(\mathbb{A})||\mathcal{C}_k$ at most a polynomial number of values are added to *used_nonces*, then at most a polynomial number of values can be added to *used_nonces* in an execution of polynomial length. Since B_k is not negligible in $\{NG_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$, then with non negligible probability $NG_k^2(\mathbb{A})||\mathcal{C}_k$ has chosen a nonce that is not a fresh nonce, that is, it is equal to a value chosen by the environment. This contradicts Proposition 2.2 and thus $\{NG_k^3(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{NG_k^3(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$. This implies, by Theorem 5.10, that $\{NG_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(NG_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

6.2 Encryption

The second primitive we consider is encryption. We model an encryption scheme $\mathsf{E} = (\mathtt{KGen}, \mathtt{Enc}, \mathtt{Dec})$ with an automaton that provides several families of actions: a family of input actions *get_public_encrypt*(A) that is used by other automata to ask the public key associated to agent A and the corresponding *ret_public_encrypt*(A, E) that returns the public key E of A:

Input *get_public_encrypt*(A)
  Effect:
    if $ek(A) = \perp$ then
      $(ek_A, dk_A) := \mathtt{KGen}(1^k)$
    fi
    $pk_A := ek_A$

Output *ret_public_encrypt*(A, E)
  Precondition:
    $pk_A = E$
  Effect:
    $pk_A := \perp$

a family of input actions *get_corrupt_encrypt*(A) that is used by other automata to ask both public and private keys of a corrupted agent A and the corresponding *ret_corrupt_encrypt*(A, E, D) that returns both public and private encryption keys E and D of A, respectively:

Input $get\_corrupt\_encrypt(A)$
  Effect:
    if $ek_A = \bot$ then
      $(ek_A, dk_A) := \texttt{KGen}(1^k)$
    fi
    $ck_A := (ek_A, dk_A)$

Output $ret\_corrupt\_encrypt(A, E, D)$
  Precondition:
    $ck_A = (E, D)$
  Effect:
    $ck_A := \bot$

Finally, the automaton provides actions to encrypt and decrypt messages: a family of input actions $get\_encrypt(A, M)$ that is used to require the encryption of M under the public key of A and the corresponding output action $ret\_encrypt(A, C)$ that returns the ciphertext C obtained invoking the encryption algorithm `Enc`:

Input $get\_encrypt(A, M)$
  Effect:
    $enc\_value_A := \texttt{Enc}(ek_A, M)$

Output $ret\_encrypt(A, C)$
  Precondition:
    $enc\_value_A = C$
  Effect:
    $enc\_value_A := \bot$

and a family of input actions $get\_decrypt(A, C)$ that is used require the decryption of C under the private key of A and the corresponding output action $ret\_decrypt(A, M)$ that returns the plaintext M obtained invoking the decryption algorithm `Dec`:

Input $get\_decrypt(A, C)$
  Effect:
    $dec\_value_A := \texttt{Dec}(dk_A, C)$

Output $ret\_decrypt(A, M)$
  Precondition:
    $dec\_value_A = M$
  Effect:
    $dec\_value_A := \bot$

Figure 6.4 depicts the complete encryption automaton.

6.2.1 Automaton and Properties of Encryption

In Section 2.5.6 we have seen that the main property of IND-CCA encryption schemes is that when we encrypt a message m, then the probability that the resulting ciphertext is repeated is negligible.

Now we want to show that the encryption automaton we defined satisfies this property. To do this, we consider another encryption automaton that ensures that generated ciphertexts are never repeated and then we show that there exists a polynomially accurate simulation between them.

Encryption automaton $\mathcal{E}_k(\mathbb{A})$

Signature:

Input:

$get\_public\_encrypt(A)$, $A \in \mathbb{A}$
$get\_corrupt\_encrypt(A)$, $A \in \mathbb{A}$
$get\_encrypt(A, M)$, $A \in \mathbb{A}$, $M \in$ Message
$get\_decrypt(A, C)$, $A \in \mathbb{A}$, $C \in$ Ciphertext

Output:

$ret\_public\_encrypt(A, E)$, $A \in \mathbb{A}$, $E \in$ EKey
$ret\_corrupt\_encrypt(A, E, D)$, $A \in \mathbb{A}$, $E \in$ EKey, $D \in$ DKey
$ret\_encrypt(A, C)$, $A \in \mathbb{A}$, $C \in$ Ciphertext
$ret\_decrypt(A, M)$, $A \in \mathbb{A}$, $M \in$ Message

State:

$ek_A \in \{\bot\} \cup$ EKey, $A \in \mathbb{A}$, initially $\bot$
$dk_A \in \{\bot\} \cup$ DKey, $A \in \mathbb{A}$, initially $\bot$
$pk_A \in \{\bot\} \cup$ EKey, $A \in \mathbb{A}$, initially $\bot$
$ck_A \in \{\bot\} \cup$ EKey $\times$ DKey, $A \in \mathbb{A}$, initially $\bot$
$enc\_value_A \in \{\bot\} \cup$ Ciphertext, $A \in \mathbb{A}$, initially $\bot$
$dec\_value_A \in \{\bot\} \cup$ Message, $A \in \mathbb{A}$, initially $\bot$

Transitions:

Input $get\_public\_encrypt(A)$
Effect:
if $ek_A = \bot$ then
$(ek_A, dk_A) := \texttt{KGen}(1^k)$
fi
$pk_A := ek_A$

Input $get\_corrupt\_encrypt(A)$
Effect:
if $ek_A = \bot$ then
$(ek_A, dk_A) := \texttt{KGen}(1^k)$
fi
$ck_A := (ek_A, dk_A)$

Input $get\_encrypt(A, M)$
Effect:
$enc\_value_A := \texttt{Enc}(ek_A, M)$

Input $get\_decrypt(A, C)$
Effect:
$dec\_value_A := \texttt{Dec}(dk_A, C)$

Output $ret\_public\_encrypt(A, E)$
Precondition:
$pk_A = E$
Effect:
$pk_A := \bot$

Output $ret\_corrupt\_encrypt(A, E, D)$
Precondition:
$ck_A = (E, D)$
Effect:
$ck_A := \bot$

Output $ret\_encrypt(A, C)$
Precondition:
$enc\_value_A = C$
Effect:
$enc\_value_A := \bot$

Output $ret\_decrypt(A, M)$
Precondition:
$dec\_value_A = M$
Effect:
$dec\_value_A := \bot$

Fig. 6.4. Encryption automaton $\mathcal{E}_k(\mathbb{A})$

$\mathcal{E}_k(\mathbb{A})$

Input $get\_encrypt(A, M)$
Effect:
$enc\_value_A := \texttt{Enc}(ek_A, M)$

Output $ret\_encrypt(A, C)$
Precondition:
$enc\_value_A = C$
Effect:
$enc\_value_A := \bot$

$\mathcal{E}_k^1(\mathbb{A})$

Input $get\_encrypt(A, M)$
Effect:
$enc\_value_A := c$ where $c = \texttt{Enc}(ek_A, M)$
$is\_fresh_A := \begin{cases} F & \text{if } c \in generated\_ciphers \\ T & \text{otherwise} \end{cases}$
$generated\_ciphers := generated\_ciphers \cup \{c\}$

Output $ret\_encrypt(A, C)$
Precondition:
$enc\_value_A = C$
Effect:
$enc\_value_A := \bot$
$is\_fresh_A := \bot$

Fig. 6.5. $get\_encrypt(A, M)$ and $ret\_encrypt(A, C)$ of $\mathcal{E}_k$ and $\mathcal{E}_k^1$

Removing internally generated repeated ciphertexts

A possible way to define such automaton is to add some history variables that keep all previous generated ciphertexts and the information about the freshness of the generated encryption. Moreover, we modify the $get\_encrypt(A, M)$ action imposing that the ciphertext c returned by the `Enc` algorithm is not an old ciphertext and then c is added to the set of already generated ciphertexts. We consider also the set of variables that stores the information about the freshness of last computed ciphertexts to simplify the identification of the states where a repeated ciphertext is generated. Note that we are using a technique that is analogous to the one we used for nonces in Section 6.1.1.

For the first step, let $\mathcal{E}_k^1$ be the automaton obtained from $\mathcal{E}_k$ adding the state variable $generated\_ciphers \subseteq$ Ciphertext with initial value $\emptyset$ and the

family of state variables $is\_fresh_A \in \{T, F, \perp\}$ with initial value $\perp$ where $A \in \mathbb{A}$. Moreover, each $get\_encrypt(A, M)$ action modifies is_fresh_A assigning F if the ciphertext returned by $\mathtt{Enc}(ek_A, M)$ is in $generated\_ciphers$; T otherwise and also updates $generated\_ciphers$ adding the encrypted value enc_value_A to $generated\_ciphers$, as depicted in Figure 6.5. The action $ret\_encrypt(A, C)$ reset the is_fresh_A to its initial value $\perp$.

It is immediate to see that $\mathcal{E}_k^1(\mathbb{A})$ is an extension of $\mathcal{E}_k(\mathbb{A})$. In fact, we simply add some history variables that keep information about the generated ciphertexts and the freshness of the ciphertexts. Such variables are updated internally and do not affect the behavior of the automaton. Moreover, they are updated in a "deterministic" way, that is, when $\mathcal{E}_k(\mathbb{A})$ reaches s' from s with probability p, then also $\mathcal{E}_k^1(\mathbb{A})$ reaches s_1' from s_1 with probability p where $s = s_1{\restriction_s}$ and $s' = s_1'{\restriction_{s'}}$. In fact, when $\mathcal{E}_k(\mathbb{A})$ performs a transition labelled by $get\_encrypt(A, M)$, it actually performs a transition $tr = (s, get\_encrypt(A, M), \mu)$ where for each $s' \in Supp(\mu)$, $s'.enc\_value_A = C$ where C is the value returned by $\mathtt{Enc}(s.ek_A, M)$ and $\mu(s')$ is the probability that $\mathtt{Enc}(s.ek_A, M)$ generates C. $\mathcal{E}_k^1(\mathbb{A})$ also performs a transition $tr_1 = (s_1, get\_encrypt(A, M), \mu_1)$ for each $s_1' \in Supp(\mu_1)$, $s_1'.enc\_value_A = C$ where C is the value returned by $\mathtt{Enc}(s_1.ek_A, M)$ and $\mu(s')$ is the probability that $\mathtt{Enc}(s_1.ek_A, M)$ generates C. That is, $tr = tr_1{\restriction_{tr}}$.

The above intuition is formalized by the following result:

Lemma 6.11. *Let $\mathbb{A}$ be a set of (identities of) agents and W be the set of variables $\{generated\_ciphers\} \cup \{is\_fresh_A \mid A \in \mathbb{A}\}$ defined as above. For each $k \in \mathbb{N}$, $\mathcal{E}_k^1(\mathbb{A}) \in \mathrm{Ext}_\emptyset^W(\mathcal{E}_k(\mathbb{A}))$.*

Proof. To prove the statement of the Lemma, we need to check if the requirements of Definition 4.5 are satisfied:

compatible states: let v be a state variable of $\mathcal{E}_k^1(\mathbb{A})$. Then v is either one of ek_A, dk_A, pk_A, ck_A, enc_value_A, and dec_value_A for $A \in \mathbb{A}$ and thus it is a state variable of $\mathcal{E}_k(\mathbb{A})$, or v is either $generated\_ciphers$ or is_fresh_A for $A \in \mathbb{A}$, and thus $v \in W$;

compatible start state: $\bar{s}_k^1$ is identified by value $\perp$ for each ek_A, dk_A, pk_A, ck_A, enc_value_A, dec_value_A, and is_fresh_A (with $A \in \mathbb{A}$), and by $\emptyset$ for $generated\_ciphers$. Since $\bar{s}_k$ is identified by the value $\perp$ for each ek_A, dk_A, pk_A, ck_A, enc_value_A, and dec_value_A (with $A \in \mathbb{A}$), then $\bar{s}_k = \bar{s}_k^1{\restriction_{\bar{s}_k}}$;

compatible actions: $\mathcal{E}_k^1(\mathbb{A})$ and $\mathcal{E}_k(\mathbb{A})$ provides the same set of actions, so this condition is trivially verified; and

compatible transitions: let $tr_1 = (s_1, a, \mu_1) \in D_k^1$ be a transition of $\mathcal{E}_k^1$. Since A of $\mathcal{E}_k^1$ is equal to A of $\mathcal{E}_k(\mathbb{A})$ by definition of $\mathcal{E}_k^1(\mathbb{A})$, then we must verify that there exists a transition $tr = (s, a, \mu) \in D_k$ such that $tr = tr_1 \lceil_{tr}$. There are eight cases:

- $a = get\_public\_encrypt(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_public\_encrypt(A)$, we can identify two cases: $s_1.ek_A = \perp$ or $s_1.ek_A \neq \perp$. If $s_1.ek_A = \perp$, then for each state s_1' of $\mathcal{E}_k^1$, $\mu_1(s_1') = \rho(e, d)$ and s_1' is identified by the same values of s_1 except for the following values: $s_1'.ek_A = e$, $s_1'.dk_A = d$, and $s_1'.pk_A = e$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Let s be the state of $\mathcal{E}_k(\mathbb{A})$ such that $s = s_1 \lceil_s$. By definition of action $get\_public\_encrypt(A)$, s enables $get\_public\_encrypt(A)$ that leads to the measure μ such that for each state s' of $\mathcal{E}_k$, $\mu(s') = \rho(e, d)$ and s' is identified by the same values of s except for the following values: $s'.ek_A = e$, $s'.dk_A = d$, and $s'.pk_A = e$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Thus $\mu = \mu_1 \lceil_\mu$, and hence $tr = tr_1 \lceil_{tr}$.
  If $s_1.ek_A \neq \perp$, then $\mu_1 = \delta_{s_1'}$ where s_1' is the state of $\mathcal{E}_k^1(\mathbb{A})$ that is identified by the same values of s_1 except for the value of the variable pk_A where $s_1'.pk_A = s_1.ek_A$. Let s be the state of $\mathcal{E}_k(\mathbb{A})$ such that $s = s_1 \lceil_s$. By definition of action $get\_public\_encrypt(A)$, s enables $get\_public\_encrypt(A)$ that leads to the measure $\delta_{s'}$ where s' is the state of $\mathcal{E}_k(\mathbb{A})$ that is identified by the same values of s except for variable pk_A where $s'.pk_A = s.ek_A$. Thus $\delta_{s'} = \delta_{s_1'} \lceil_{\delta_{s'}}$, and hence $tr = tr_1 \lceil_{tr}$.
- $a = ret\_public\_encrypt(A, E)$ for some $A \in \mathbb{A}$ and $E \in$ EKey: by definition of action $ret\_public\_encrypt(A, E)$, it follows that $s_1.pk_A = E$ and that $\mu_1 = \delta_{s_1'}$ where s_1' is the state of $\mathcal{E}_k^1(\mathbb{A})$ that is identified by the same values of s_1 except for the value of the variable pk_A where $s_1'.pk_A = \perp$. Let s be the state of $\mathcal{E}_k(\mathbb{A})$ such that $s = s_1 \lceil_s$. By definition of action $ret\_public\_encrypt(A, E)$, s enables $ret\_public\_encrypt(A, E)$ that leads to the measure $\delta_{s'}$ where s' is the state of $\mathcal{E}_k(\mathbb{A})$ that is identified by the same values of s except for variable pk_A where $s'.pk_A = \perp$. Thus $\delta_{s'} = \delta_{s_1'} \lceil_{\delta_{s'}}$, and hence $tr = tr_1 \lceil_{tr}$.

- $a = get\_corrupt\_encrypt(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_corrupt\_encrypt(A)$, we can identify two cases: $ek_A = \bot$ or $ek_A \neq \bot$. If $ek_A = \bot$, then for each state s_1' of $\mathcal{E}_k^1$, $\mu_1(s_1') = \rho(e, d)$ and s_1' is identified by the same values of s_1 except for the following values: $s_1'.ek_A = e$, $s_1'.dk_A = d$, and $s_1'.ck_A = (e, d)$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Let s be the state of $\mathcal{E}_k(\mathbb{A})$ such that $s = s_1\lceil_s$. By definition of the action $get\_corrupt\_encrypt(A)$, s enables $get\_corrupt\_encrypt(A)$ that leads to the measure μ such that for each state s' of $\mathcal{E}_k$, $\mu(s') = \rho(e, d)$ and s' is identified by the same values of s except for the following values: $s'.ek_A = e$, $s'.dk_A = d$, and $s'.ck_A = (e, d)$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Thus $\mu = \mu_1\lceil_\mu$, and hence $tr = tr_1\lceil_{tr}$.
  If $ek_A \neq \bot$, then $\mu_1 = \delta_{s_1'}$ where s_1' is the state of $\mathcal{E}_k^1(\mathbb{A})$ that is identified by the same values of s_1 except for the value of the variable ck_A where $s_1'.ck_A = (s_1.ek_A, s_1.dk_A)$. Let s be the state of $\mathcal{E}_k(\mathbb{A})$ such that $s = s_1\lceil_s$. By definition of the action $get\_corrupt\_encrypt(A)$, s enables $get\_corrupt\_encrypt(A)$ that leads to the measure $\delta_{s'}$ where s' is the state of $\mathcal{E}_k(\mathbb{A})$ that is identified by the same values of s except for variable ck_A where $s'.ck_A = (s.ek_A, s.dk_A)$. Thus $\delta_{s'} = \delta_{s_1'}\lceil_{\delta_{s'}}$, and hence $tr = tr_1\lceil_{tr}$.
- $a = ret\_corrupt\_encrypt(A, E, D)$ for some $A \in \mathbb{A}$, $E \in$ EKey, and $D \in$ DKey: by definition of action $ret\_corrupt\_encrypt(A, E, D)$, it follows that $s_1.ck_A = (E, D)$ and that $\mu_1 = \delta_{s_1'}$ where s_1' is the state of $\mathcal{E}_k^1(\mathbb{A})$ that is identified by the same values of s_1 except for ck_A where $s_1'.ck_A = \bot$. Let s be the state of $\mathcal{E}_k(\mathbb{A})$ such that $s = s_1\lceil_s$. By definition of action $ret\_corrupt\_encrypt(A, E, D)$, s enables $ret\_corrupt\_encrypt(A, E, D)$ that leads to the measure $\delta_{s'}$ where s' is the state of $\mathcal{E}_k(\mathbb{A})$ that is identified by the same values of s except for variable ck_A where $s'.ck_A = \bot$. Thus $\delta_{s'} = \delta_{s_1'}\lceil_{\delta_{s'}}$, and hence $tr = tr_1\lceil_{tr}$.
- $a = get\_encrypt(A, M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of the action $get\_encrypt(A, M)$, it follows that for each state s_1' of $\mathcal{E}_k^1$, $\mu_1(s_1') = \rho(C)$ and s_1' is identified by the same values of s_1 except for the following values: $s_1'.is\_fresh_A = F$ if $C \in s_1.generated\_ciphers$, T otherwise, $s_1'.enc\_value_A = C$, and $s_1'.generated\_ciphers = s_1.generated\_ciphers \cup \{C\}$. ρ is the probabil-

ity measure induced over Ciphertext by the probabilistic algorithm $\texttt{Enc}(s_1.ek_A, M)$. Let s be the state of $\mathcal{E}_k(\mathbb{A})$ such that $s = s_1{\restriction}_s$. By definition of action $get\_encrypt(A, M)$, s enables $get\_encrypt(A, M)$ that leads to the measure ρ where for each state s' of $\mathcal{E}_k(\mathbb{A})$ that is identified by the same values of s except for variable enc_value_A, we have that $\mu(s') = \rho(C)$ and $s'.enc\_value_A = C$. Thus $\mu = \mu_1{\restriction}_\mu$, and hence $tr = tr_1{\restriction}_{tr}$.

- $a = ret\_encrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of action $ret\_encrypt(A, C)$, it follows that $s_1.enc\_value_A = C$ and that $\mu_1 = \delta_{s_1'}$ where s_1' is the state of $\mathcal{E}_k^1(\mathbb{A})$ that is identified by the same values of s_1 except for enc_value_A that assumes the value $s_1'.enc\_value_A = \perp$ and for is_fresh_A that is reset to $\perp$. Let s be the state of $\mathcal{E}_k(\mathbb{A})$ such that $s = s_1{\restriction}_s$. By definition of action $ret\_encrypt(A, C)$, s enables $ret\_encrypt(A, C)$ that leads to the measure $\delta_{s'}$ where s' is the state of $\mathcal{E}_k(\mathbb{A})$ that is identified by the same values of s except for variable enc_value_A where $s'.enc\_value_A = \perp$. Thus $\delta_{s'} = \delta_{s_1'}{\restriction}_{\delta_{s'}}$, and hence $tr = tr_1{\restriction}_{tr}$.
- $a = get\_decrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of action $get\_decrypt(A, C)$, it follows that $\mu_1 = \delta_{s_1'}$ where s_1' is the state of $\mathcal{E}_k^1(\mathbb{A})$ that is identified by the same values of s_1 except for dec_value_A where $s_1'.dec\_value_A = \texttt{Dec}(s_1.dk_A, C)$. Let s be the state of $\mathcal{E}_k(\mathbb{A})$ such that $s = s_1{\restriction}_s$. By definition of action $get\_decrypt(A, C)$, s enables $get\_decrypt(A, C)$ that leads to the measure $\delta_{s'}$ where s' is the state of $\mathcal{E}_k(\mathbb{A})$ that is identified by the same values of s except for variable dec_value_A where $s'.dec\_value_A = \texttt{Dec}(s.dk_A, C)$. Thus $\delta_{s'} = \delta_{s_1'}{\restriction}_{\delta_{s'}}$, and hence $tr = tr_1{\restriction}_{tr}$.
- $a = ret\_decrypt(A, M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of action $ret\_decrypt(A, M)$, it follows that $s_1.dec\_value_A = M$ and that $\mu_1 = \delta_{s_1'}$ where s_1' is the state of $\mathcal{E}_k^1(\mathbb{A})$ that is identified by the same values of s_1 except for dec_value_A where $s_1'.dec\_value_A = \perp$. Let s be the state of $\mathcal{E}_k(\mathbb{A})$ such that $s = s_1{\restriction}_s$. By definition of action $ret\_decrypt(A, M)$, s enables $ret\_decrypt(A, M)$ that leads to the measure $\delta_{s'}$ where s' is the state of $\mathcal{E}_k(\mathbb{A})$ that is identified by the same values of s except for variable dec_value_A where $s'.dec\_value_A = \perp$. Thus $\delta_{s'} = \delta_{s_1'}{\restriction}_{\delta_{s'}}$, and hence $tr = tr_1{\restriction}_{tr}$.

Since the requirements of Definition 4.5 are satisfied, for each $k \in \mathbb{N}$, $\mathcal{E}_k^1(\mathbb{A}) \in \mathrm{Ext}_\emptyset^W(\mathcal{E}_k(\mathbb{A}))$. □

The existence of a polynomially accurate simulation from $\mathcal{E}_k(\mathbb{A})$ to $\mathcal{E}_k^1(\mathbb{A})$ in now straightforward:

Proposition 6.12. *Let $\mathbb{A}$ be a set of (identities of) agents. For each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k(\mathbb{A})$,*

$$\{\mathcal{E}_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^1(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.11, for each $k \in \mathbb{N}$, $\mathcal{E}_k^1(\mathbb{A}) \in \mathrm{Ext}_B^W(\mathcal{E}_k(\mathbb{A}))$ where B is the empty set and W is $\{\mathit{generated\_ciphers}\} \cup \{\mathit{is\_fresh}_A \mid A \in \mathbb{A}\}$. This implies, by Lemma 4.6, that for each context C_k' compatible with $\mathcal{E}_k(\mathbb{A})$ such that $B \subseteq A_{C_k'}$, $\mathcal{E}_k(\mathbb{A})||C_k' \preceq \mathcal{E}_k^1(\mathbb{A})||C_k'$ and thus, by Proposition 5.6, $\{\mathcal{E}_k(\mathbb{A})||C_k'\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^1(\mathbb{A})||C_k'\}_{k\in\mathbb{N}}$. Since $B = \emptyset$, each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k(\mathbb{A})$ satisfies $B \subseteq A_{\mathcal{C}_k}$ and thus for each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k(\mathbb{A})$, $\{\mathcal{E}_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^1(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

For the second step, let $\mathcal{E}_k^2$ be the automaton obtained from $\mathcal{E}_k^1$ modifying each $\mathit{get\_encrypt}(A, M)$ action as follows: the encryption algorithm `Enc` is invoked until it returns a value that is a fresh ciphertext, that is not inside $\mathit{generated\_ciphers}$, as depicted in Figure 6.6.

It is immediate to see that $\mathcal{E}_k^2(\mathbb{A})$ simulates the G_k-conditional of $\mathcal{E}_k^1(\mathbb{A})$ where for each $k \in \mathbb{N}$, G_k is the set of states of $\mathcal{E}_k^1(\mathbb{A})$ such that for each $A \in \mathbb{A}$, $\neq .\mathit{is\_fresh}_A F$. In fact, we iterate the generation of the encryption of M under the public key of A until it returns a value that is not equal to an already generated ciphertext. That is, we condition the choice of the ciphertext to the fact that it is fresh (and thus, $\neq .\mathit{is\_fresh}_A F$ is always satisfied).

Note that $\mathcal{E}_k^2(\mathbb{A})$ is not the G_k-conditional of $\mathcal{E}_k^1(\mathbb{A})$ since it provides more transitions than $\mathcal{E}_k^1(\mathbb{A})|G_k$. In fact, $\mathcal{E}_k^2(\mathbb{A})$ enables transitions that leave from a state not in G_k and that leads to a probability measures μ such that $\mu(G_k) = 0$. For example, let s be the state such that $s.\mathit{is\_fresh}_A = F$, $s.\mathit{is\_fresh}_B = F$, $s.\mathit{enc\_value}_A = C$. Thus, by definition of $\mathit{ret\_encrypt}$ actions, s enables the action $\mathit{ret\_encrypt}(A, C)$ that leads to the measure $\delta_{s'}$ where in s' only $\mathit{enc\_value}_A$ and $\mathit{is\_fresh}_A$ are different with respect to the values in s. Thus $s'.\mathit{is\_fresh}_B$ is still F and hence $s' \notin G_k$.

The above intuition is formalized by the following result:

Lemma 6.13. *Given $\mathcal{E}_k^1(\mathbb{A})$, let B_k be the set of states of $\mathcal{E}_k^1$ such that $s \in B_k$ if there exists $A \in \mathbb{A}$ such that $s.\mathit{is\_fresh}_A = F$. Let G_k be the set $S_k^1 \setminus B_k$.*

For each $k \in \mathbb{N}$, $\mathcal{E}_k^1(\mathbb{A})|G_k \preceq \mathcal{E}_k^2(\mathbb{A})$.

$\mathcal{E}_k^1$

Input $get\_encrypt(A, M)$
Effect:
$enc\_value_A := c$ where $c = \texttt{Enc}(ek_A, M)$
$is\_fresh_A := \begin{cases} F & \text{if } c \in generated\_ciphers \\ T & \text{otherwise} \end{cases}$
$generated\_ciphers := generated\_ciphers \cup \{c\}$

$\mathcal{E}_k^2$

Input $get\_encrypt(A, M)$
Effect:
repeat
$c = \texttt{Enc}(ek_A, M)$
until $c \notin generated\_ciphers$
$enc\_value_A := c$
$is\_fresh_A := \begin{cases} F & \text{if } c \in generated\_ciphers \\ T & \text{otherwise} \end{cases}$
$generated\_ciphers := generated\_ciphers \cup \{c\}$

Fig. 6.6. *get_encrypt* of $\mathcal{E}_k^1$ and $\mathcal{E}_k^2$

Proof. For each $k \in \mathbb{N}$, let id_k be the identity relation on states. id_k is a simulation from $\mathcal{E}_k^1(\mathbb{A})|G_k$ to $\mathcal{E}_k^2(\mathbb{A})$.

The condition on start states is trivially true: let $\bar{s}_k^1$ be the start state of $\mathcal{E}_k^1(\mathbb{A})|G_k$ and $\bar{s}_k^2$ be the start state of $\mathcal{E}_k^2(\mathbb{A})$. By definition of conditional, it follows that $\bar{s}_k^1$ is the start state of $\mathcal{E}_k^1(\mathbb{A})$. Since by definition of $\mathcal{E}_k^2(\mathbb{A})$, the only difference between $\mathcal{E}_k^1(\mathbb{A})$ and $\mathcal{E}_k^2(\mathbb{A})$ is on the definition of the action $get\_encrypt(A, M)$, we have that $\bar{s}_k^2 = \bar{s}_k^1$ and thus $\bar{s}_k^1 \; id_k \; \bar{s}_k^2$.

For the step condition, let s_1 and s_2 be two states of $\mathcal{E}_k^1(\mathbb{A})|G_k$ and $\mathcal{E}_k^2(\mathbb{A})$, respectively, such that $s_1 \; id_k \; s_2$. Let (s_1, a, μ_1) be a transition of $\mathcal{E}_k^1(\mathbb{A})|G_k$ that leaves from s_1. We must find μ_2 such that (s_2, a, μ_2) is a transition of $\mathcal{E}_k^2(\mathbb{A})$ and $\mu_1 \; \mathcal{L}(id_k) \; \mu_2$. There are eight cases:

– $a = get\_public\_encrypt(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_public\_encrypt(A)$, we can identify two cases: $s_1.ek_A = \perp$ or $s_1.ek_A \neq \perp$. If $s_1.ek_A = \perp$, then for each state s_1' of $\mathcal{E}_k^1$, $\mu_1(s_1') = \rho_1(e, d)$ and s_1' is identified by the same values of s_1 except for the following values: $s_1'.ek_A = e$, $s_1'.dk_A = d$, and $s_1'.pk_A = e$ where ρ_1 is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Since $s_1 \; id_k \; s_2$, we have that also s_2 satisfies $s_2.ek_A = \perp$,

and hence for each state s_2' of $\mathcal{E}_k^2$, $\mu_2(s_2') = \rho_2(e,d)$ and s_2' is identified by the same values of s_2 except for the following values: $s_2'.ek_A = e$, $s_2'.dk_A = d$, and $s_2'.pk_A = e$ where ρ_2 is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. This implies that $s_1'\ id_k\ s_2'$ for each state s_1' and s_2' satisfying above conditions and since $\rho_1 = \rho_2$, we have that $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.
If $s_1.ek_A \neq \perp$, then $\mu_1 = \delta_{s_1'}$ where s_1' is the state of $\mathcal{E}_k^1(\mathbb{A})$ that is identified by the same values of s_1 except for pk_A where $s_1'.pk_A = s_1.ek_A$. Since $s_1\ id_k\ s_2$, we have that also s_2 satisfies $s_2.ek_A \neq \perp$, and hence by definition of action $get\_public\_encrypt(A)$, s_2 enables $get\_public\_encrypt(A)$ that leads to the measure $\delta_{s_2'}$ where s_2' is the state of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable pk_A where $s_2'.pk_A = s_2.ek_A$. This implies that $s_1'\ id_k\ s_2'$ and thus $\delta_{s_1'}\ \mathcal{L}(id_k)\ \delta_{s_2'}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

– $a = ret\_public\_encrypt(A,E)$ for some $A \in \mathbb{A}$ and $E \in$ EKey: by definition of the action $ret\_public\_encrypt(A,E)$, it follows that s_1 satisfies $s_1.pk_A = E$. Moreover, it follows that $\mu_1 = \delta_{s_1'}$ where s_1' is the state such that $s_1'.pk_A = \perp$, and all other variables that describe s_1' have the same value of the variables that describe s_1. Since $s_1\ id_k\ s_2$, we have that also s_2 satisfies $s_2.pk_A = E$ and thus it enables the transition $(s_2, ret\_public\_encrypt(A,E), \mu_2)$ where μ_2 is the measure $\delta_{s_2'}$ where s_2' is the state such that $s_2'.pk_A = \perp$, and all other variables that describe s_2' have the same value of the variables that describe s_2. This implies that $s_1'\ id_k\ s_2'$ and thus $\delta_{s_1'}\ \mathcal{L}(id_k)\ \delta_{s_2'}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.
– $a = get\_corrupt\_encrypt(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_corrupt\_encrypt(A)$, we can identify two cases: $s_1.ek_A = \perp$ or $s_1.ek_A \neq \perp$. If $s_1.ek_A = \perp$, then for each state s_1' of $\mathcal{E}_k^1$, $\mu_1(s_1') = \rho_1(e,d)$ and s_1' is identified by the same values of s_1 except for the following values: $s_1'.ek_A = e$, $s_1'.dk_A = d$, and $s_1'.ck_A = (e,d)$ where ρ_1 is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Since $s_1\ id_k\ s_2$, we have that also s_2 satisfies $s_2.ek_A = \perp$, and hence for each state s_2' of $\mathcal{E}_k^2$, $\mu_2(s_2') = \rho_2(e,d)$ and s_2' is identified by the same values of s_2 except for the following values: $s_2'.ek_A = e$, $s_2'.dk_A = d$, and $s_2'.ck_A = (e,d)$ where ρ_2 is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. This implies that $s_1'\ id_k\ s_2'$ for each state s_1' and s_2' satisfying above conditions and since $\rho_1 = \rho_2$, we have that $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

If $s_1.ek_A \neq \perp$, then $\mu_1 = \delta_{s'_1}$ where s'_1 is the state of $\mathcal{E}^1_k(\mathbb{A})$ that is identified by the same values of s_1 except for ck_A where $s'_1.ck_A = (s_1.ek_A, s_1.dk_A)$. Since $s_1\ id_k\ s_2$, we have that also s_2 satisfies $s_2.ek_A \neq \perp$, and hence by definition of action $get\_corrupt\_encrypt(A)$, s_2 enables $get\_corrupt\_encrypt(A)$ that leads to the measure $\delta_{s'_2}$ where s'_2 is the state of $\mathcal{E}^2_k(\mathbb{A})$ that is identified by the same values of s_2 except for variable ck_A where $s'_2.ck_A = (s_2.ek_A, s_2.dk_A)$. This implies that $s'_1\ id_k\ s'_2$ and thus $\delta_{s'_1}\ \mathcal{L}(id_k)\ \delta_{s'_2}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

– $a = ret\_corrupt\_encrypt(A, E, D)$ for some $A \in \mathbb{A}$, $E \in$ EKey and $D \in$ DKey: by definition of the action $ret\_corrupt\_encrypt(A, E, D)$, it follows that s_1 satisfies $s_1.ck_A = (E, D)$. Moreover, it follows that $\mu_1 = \delta_{s'_1}$ where s'_1 is the state such that $s'_1.ck_A = \perp$, and all other variables that describe s'_1 have the same value of the variables that describe s_1. Since $s_1\ id_k\ s_2$, we have that also s_2 satisfies $s_2.ck_A = (E, D)$ and thus it enables the transition $(s_2, ret\_corrupt\_encrypt(A, E, D), \mu_2)$ where μ_2 is the measure $\delta_{s'_2}$ where s'_2 is the state such that $s'_2.ck_A = \perp$, and all other variables that describe s'_2 have the same value of the variables that describe s_2. This implies that $s'_1\ id_k\ s'_2$ and thus $\delta_{s'_1}\ \mathcal{L}(id_k)\ \delta_{s'_2}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

– $a = get\_encrypt(A, M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of $get\_encrypt(A, M)$, it follows that for each state s'_1 of $\mathcal{E}^1_k$, $\mu_1(s'_1) = \rho_1(C)$ and s'_1 is identified by the same values of s_1 except for the following values: $s'_1.is\_fresh_A = F$ if $C \in s_1.generated\_ciphers$, T otherwise, $s'_1.enc\_value_A = C$, and $s'_1.generated\_ciphers = s_1.generated\_ciphers \cup \{C\}$. ρ_1 is the probability measure induced over Ciphertext by the probabilistic algorithm $\texttt{Enc}(s_1.ek_A, M)$ conditioned to G_k. Since $s_1\ id_k\ s_2$, we have that also s_2 enables the transition $(s_2, get\_encrypt(A, M), \mu_2)$ where for each state s'_2 of $\mathcal{E}^2_k$, $\mu_2(s'_2) = \rho_2(C)$ and s'_2 is identified by the same values of s_2 except for the following values: $s'_2.is\_fresh_A = F$ if $C \in s_2.generated\_ciphers$, T otherwise, $s'_2.enc\_value_A = C$, and $s'_2.generated\_ciphers = s_2.generated\_ciphers \cup \{C\}$. ρ_2 is the probability measure induced over Ciphertext by the probabilistic algorithm $\texttt{Enc}(s_2.ek_A, M)$ that is iterated until it returns a value that does not belong to $s_2.generated\_ciphers$. This implies that $s'_1\ id_k\ s'_2$ for each state s'_1 and s'_2 satisfying above conditions and if $\rho_1 = \rho_2$, then $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

ρ_1 is equal to ρ_2 since for each $s \in G_k$, $\rho_1(s) = \rho_2(s)$. In fact, by definition of conditional, $\rho_1(s) = \rho(s)/\rho(G_k)$ where ρ is the probability

measure induced over Ciphertext by $\texttt{Enc}(s_1.ek_A, M)$, $\rho(G_k) > 0$ and $\rho(B_k) < 1$. By definition of $get\_encrypt(A, M)$ action of $\mathcal{E}_k^2$, we have that $\rho_2(s) = \sum_{i=0}^{+\infty} \rho(B_k)^i \rho(s) = \rho(s) \sum_{i=0}^{+\infty} \rho(B_k)^i = \rho(s) \dfrac{1}{1-\rho(B_k)} = \rho(s)/\rho(G_k) = \rho_1(s)$.

- $a = ret\_encrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of the action $ret\_encrypt(A, C)$, it follows that s_1 satisfies $s_1.enc\_value_A = C$. Moreover, it follows that $\mu_1 = \delta_{s_1'}$ where s_1' is the state such that $s_1'.enc\_value_A = \bot$, $s_1'.is\_fresh_A = \bot$, and all other variables that describe s_1' have the same value of the variables that describe s_1. Since $s_1\ id_k\ s_2$, we have that also s_2 satisfies $s_2.enc\_value_A = C$ and thus it enables the transition $(s_2, ret\_encrypt(A, C), \mu_2)$ where μ_2 is the measure $\delta_{s_2'}$ where s_2' is the state such that $s_2'.enc\_value_A = \bot$, $s_2'.is\_fresh_A = \bot$, and all other variables that describe s_2' have the same value of the variables that describe s_2. This implies that $s_1'\ id_k\ s_2'$ and thus $\delta_{s_1'}\ \mathcal{L}(id_k)\ \delta_{s_2'}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.
- $a = get\_decrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of the action $get\_decrypt(A, C)$, it follows that $\mu_1 = \delta_{s_1'}$ where s_1' is the state such that $s_1'.dec\_value_A = \texttt{Dec}(s_1.dk_A, C)$, and all other variables that describe s_1' have the same value of the variables that describe s_1. Since $s_1\ id_k\ s_2$, we have that also s_2 enables the transition $(s_2, get\_decrypt(A, C), \mu_2)$ where μ_2 is the measure $\delta_{s_2'}$ where s_2' is the state such that $s_2'.dec\_value_A = \texttt{Dec}(s_2.dk_A, C)$, and all other variables that describe s_2' have the same value of the variables that describe s_2. This implies that $s_1'\ id_k\ s_2'$ and thus $\delta_{s_1'}\ \mathcal{L}(id_k)\ \delta_{s_2'}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.
- $a = ret\_decrypt(A, M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of the action $ret\_decrypt(A, M)$, it follows that s_1 satisfies $s_1.dec\_value_A = M$. Moreover, it follows that $\mu_1 = \delta_{s_1'}$ where s_1' is the state such that $s_1'.dec\_value_A = \bot$, and all other variables that describe s_1' have the same value of the variables that describe s_1. Since $s_1\ id_k\ s_2$, we have that also s_2 satisfies $s_2.dec\_value_A = M$ and thus it enables the transition $(s_2, ret\_decrypt(A, M), \mu_2)$ where μ_2 is the measure $\delta_{s_2'}$ where s_2' is the state such that $s_2'.dec\_value_A = \bot$, and all other variables that describe s_2' have the same value of the variables that describe s_2. This implies that $s_1'\ id_k\ s_2'$ and thus $\delta_{s_1'}\ \mathcal{L}(id_k)\ \delta_{s_2'}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

Since for each action a, if s_1 enables a transition labelled by a that leads to μ_1, then we can find μ_2 such that $(s_2, a, \mu_2) \in D_2$ and $\mu_1 \, \mathcal{L}(id_k) \, \mu_2$, the step condition is satisfied. □

The existence of a polynomially accurate simulation from $\mathcal{E}_k^1(\mathbb{A})|G_k$ to $\mathcal{E}_k^2(\mathbb{A})$ in now straightforward:

Proposition 6.14. *Let $\mathbb{A}$ be a set of agents. Given $\mathcal{E}_k^1(\mathbb{A})$, let B_k be the set of states of $\mathcal{E}_k^1$ such that $s \in B_k$ if there exists $A \in \mathbb{A}$ such that $s.is\_fresh_A = F$. Let G_k be the set $S_k^1 \setminus B_k$.*

For each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k^1(\mathbb{A})$,

$$\{(\mathcal{E}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.13, we have that for each $k \in \mathbb{N}$, $\mathcal{E}_k^1(\mathbb{A})|G_k \preceq \mathcal{E}_k^2(\mathbb{A})$. This implies that for each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k^1(\mathbb{A})$, $(\mathcal{E}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k \preceq \mathcal{E}_k^2(\mathbb{A})||\mathcal{C}_k$. Finally, by Proposition 5.6, we have that $\{(\mathcal{E}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

To complete the chain of simulations from $\mathcal{E}_k(\mathbb{A})$ to $\mathcal{E}_k^2(\mathbb{A})$, we need to prove that

Proposition 6.15. *Let E be an IND-CCA encryption scheme and $\mathbb{A}$ be a set of agents. Let G_k be the set of states of $\mathcal{E}_k^1$ such that $s \in G_k$ if and only if for each $A \in \mathbb{A}$ $s.is\_fresh_A \neq F$, that is, the generated ciphertext is fresh. Let B_k be $S_k^1 \setminus G_k$ (that is, states s' of $\mathcal{E}_k^1$ such that $s'.is\_fresh_A = F$ for some $A \in \mathbb{A}$).*

For each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k^1(\mathbb{A})$,

$$\{\mathcal{E}_k^1(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{E}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. Theorem 5.8 states that $\{\mathcal{E}_k^1(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^1(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$ if and only if $\{B_k\}_{k\in\mathbb{N}}$ is negligible in $\{\mathcal{E}_k^1(\mathbb{A})\}_{k\in\mathbb{N}}$.

Suppose, for the sake of contradiction, that $\{\mathcal{E}_k^1(\mathbb{A})\}_{k\in\mathbb{N}}$ is not simulated by $\{\mathcal{E}_k^1(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$. This implies that $\{B_k\}_{k\in\mathbb{N}}$ is not negligible in $\{\mathcal{E}_k^1(\mathbb{A})\}_{k\in\mathbb{N}}$ and thus that there exists $c \in \mathbb{N}$, $p \in \mathit{Poly}$ such that for each $\bar{k} \in \mathbb{N}$ there exists $k > \bar{k}$ such the probability to reach states of B_k within $p(k)$ steps is at least k^{-c}, that is, the probability to reach states s such that $s.is\_fresh_A = F$ for some $A \in \mathbb{A}$ within $p(k)$ steps is at least k^{-c}. By definition of the automaton $\mathcal{E}_k^1(\mathbb{A})$, it follows that $s.is\_fresh_A$ can assume value F only as the effect of a transition that leaves from some state

s' and that is labelled by action $get\_encrypt(A, M)$. This happens only when the ciphertext value $\mathtt{Enc}(s'.ek_A, M)$ assigned to $s.enc\_value_A$ belongs to the set $RC = s'.generated\_ciphers$. Within $p(k)$ steps, we can perform at most $p(k)$ transitions labelled by $get\_encrypt(Z, P)$ with $Z \in \mathbb{A}$ and $P \in$ Message. Since by definition of $get\_encrypt(Z, P)$ we add at most one value to *generated_ciphers* each time we perform a transition labelled by $get\_encrypt(Z, P)$, within $p(k)$ steps the set RC has cardinality at most $p(k)$. This means that with probability at least k^{-c}, we have generated a ciphertext that it is equal to some $c \in RC$ with $|RC| \leqslant p(k)$. Since the encryption scheme E is IND-CCA, this contradicts the Proposition 2.4 and thus $\{\mathcal{E}_k^1(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^1(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$. This implies, by Theorem 5.10, that $\{\mathcal{E}_k^1(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{E}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

Avoiding collisions with externally generated ciphertexts

As we have seen above, there exists a simulation from the encryption automaton that can generate repeated ciphertexts to the encryption automaton that ensures that returned ciphertexts are never repeated. Sometimes, the fact that $\mathcal{E}_k^2$ does not generate repeated ciphertexts is not sufficient to satisfy the required properties: it is still possible that $\mathcal{E}_k^2$ generates an encryption that is equal to some adversary's generated ciphertext.

If we want to be sure that the encryption automaton returns values that have never occurred previously, then we must modify the automaton adding the knowledge of environment's generated ciphertexts. We can provide such knowledge using one of the following approaches: the encryption automaton receives as input sets of environment's generated ciphertexts, or it receives messages. In the second approach, given a message m, we can easily extract ciphertext from m when it is a plaintext or a signature (since we can recover the signed text from the signature itself), but when m is a ciphertext, we must decrypt it to know which other ciphertexts it contains. This is not problematic, since we already know the decryption keys associated to the agents.

First approach: set of ciphertexts

The simplest way to provide the encryption automaton with the ciphertexts generated by the environment is to use sets of ciphertexts, that is, if the environment produces ciphertexts $c_1, \ldots, c_l$, then we send the set $\{c_1, \ldots, c_l\}$ to the automaton. We do this adding a state variable *used_ciphers* that

contains all ciphertexts used by the environment and an input action $used\_ciphers(CT)$ that updates $used\_ciphers$:

Input $used\_ciphers(CT)$
  Effect:
    $used\_ciphers := used\_ciphers \cup CT$

Moreover, each $get\_encrypt(A, M)$ action modifies $is\_not\_used_A$ assigning F if the ciphertext returned by $\texttt{Enc}(ek_A, M)$ is in $used\_ciphers$; T otherwise as depicted in Figure 6.7. The action $ret\_encrypt(A, C)$ resets the $is\_not\_used_A$ to its initial value $\bot$. Let $\mathcal{E}_k^3$ be the resulting automaton.

It is immediate to see that $\mathcal{E}_k^3(\mathbb{A})$ is an extension of $\mathcal{E}_k^2(\mathbb{A})$. In fact, we simply add some history variables that keep information about all encryption generated by the environment. Such variables do not affect the behavior of the other actions and thus each transition of $\mathcal{E}_k^3(\mathbb{A})$ either is almost identical to a transition of $\mathcal{E}_k^2(\mathbb{A})$ or it is the new action that does not modify the state variables that describe states of $\mathcal{E}_k^2(\mathbb{A})$.

The above intuition is formalized by the following result:

Lemma 6.16. *Let $\mathbb{A}$ be a set of agents, B be the set $\{used\_ciphers(CT) \mid CT \subseteq \text{Ciphertext}\}$ and W be the set $\{used\_ciphers\} \cup \{is\_not\_used_A \mid A \in \mathbb{A}\}$ where action and variables are defined as above. For each $k \in \mathbb{N}$, $\mathcal{E}_k^3(\mathbb{A}) \in \text{Ext}_B^W(\mathcal{E}_k^2(\mathbb{A}))$.*

Proof. To prove the statement of the Lemma, we need to check if the requirements of Definition 4.5 are satisfied:

compatible states: let v be a state variable of $\mathcal{E}_k^3(\mathbb{A})$. Then v is either one of $generated\_ciphers$, ek_A, dk_A, pk_A, ck_A, enc_value_A, dec_value_A, is_fresh_A for $A \in \mathbb{A}$ and thus it is a state variable of $\mathcal{E}_k^2(\mathbb{A})$, or v is $used\_ciphers$ or $is\_not\_used_A$ for some $A \in \mathbb{A}$ and thus $v \in W$;

compatible start state: $\bar{s}_k^3$ is identified by value $\bot$ for each variable ek_A, dk_A, pk_A, ck_A, enc_value_A, dec_value_A, is_fresh_A, and $is\_not\_used_A$ (with $A \in \mathbb{A}$), and by $\emptyset$ for $generated\_ciphers$ and $used\_ciphers$. Since $\bar{s}_k^2$ is identified by the value $\bot$ for each variable ek_A, dk_A, pk_A, ck_A, enc_value_A, dec_value_A, and is_fresh_A (with $A \in \mathbb{A}$), and by $\emptyset$ for $generated\_ciphers$, then $\bar{s}_k^2 = \bar{s}_k^3 \restriction_{\bar{s}_k^2}$;

compatible actions: by definition of $\mathcal{E}_k^3(\mathbb{A})$, it provides the same actions of $\mathcal{E}_k^2(\mathbb{A})$ plus the actions $used\_ciphers(CT)$ that belong to B;

compatible transitions: let $tr_3 = (s_3, a, \mu_3) \in D_k^3$ be a transition of $\mathcal{E}_k^3$. Suppose that $a = used\_ciphers(CT)$. Then we must verify that there exists

$\mathcal{E}_k^2(\mathbb{A})$

Input $get\_encrypt(A, M)$
Effect:
repeat
$c = \mathtt{Enc}(ek_A, M)$
until $c \notin generated\_ciphers$
$enc\_value_A := c$
$$is\_fresh_A := \begin{cases} F & \text{if } c \in generated\_ciphers \\ T & \text{otherwise} \end{cases}$$
$generated\_ciphers := generated\_ciphers \cup \{c\}$

Output $ret\_encrypt(A, C)$
Precondition:
$enc\_value_A = C$
Effect:
$is\_fresh_A := \bot$
$enc\_value_A := \bot$

$\mathcal{E}_k^3(\mathbb{A})$

Input $get\_encrypt(A, M)$
Effect:
repeat
$c = \mathtt{Enc}(ek_A, M)$
until $c \notin generated\_ciphers$
$enc\_value_A := c$
$$is\_fresh_A := \begin{cases} F & \text{if } c \in generated\_ciphers \\ T & \text{otherwise} \end{cases}$$
$$is\_not\_used_A := \begin{cases} F & \text{if } c \in used\_ciphers \\ T & \text{otherwise} \end{cases}$$
$generated\_ciphers := generated\_ciphers \cup \{c\}$

Output $ret\_encrypt(A, C)$
Precondition:
$enc\_value_A = C$
Effect:
$is\_fresh_A := \bot$
$is\_not\_used_A := \bot$
$enc\_value_A := \bot$

Fig. 6.7. $get\_encrypt(A, M)$ and $ret\_encrypt(A, C)$ of $\mathcal{E}_k^2$ and $\mathcal{E}_k^3$

a state s_2 of $\mathcal{E}_k^2$ such that $s_2 = s_3{\restriction}_{s_2}$ and $\delta_{s_2} = \mu_3{\restriction}_{\delta_{s_2}}$. By definition of $used\_ciphers(CT)$, it follows that $\mu_3 = \delta_{s_3'}$ where s_3' is the state of $\mathcal{E}_k^3(\mathbb{A})$ that is identified by the same values of s_3 except for $used\_ciphers$ where $s_3'.used\_ciphers = s_3.used\_ciphers \cup CT$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_3{\restriction}_{s_2}$. Since the only difference between s_3 and s_3' is on the value of $used\_ciphers$, and since $used\_ciphers$ is not a variable that characterizes s_2, we have that $s_2 = s_3'{\restriction}_{s_2}$ and thus $\delta_{s_2} = \delta_{s_3'}{\restriction}_{\delta_{s_2}}$.
Suppose that $a \in A_k^2$, that is, it is an action of $\mathcal{E}_k^2(\mathbb{A})$. Then we must verify that there exists a transition $tr_2 = (s_2, a, \mu_2) \in D_k^2$ such that $tr_2 = tr_3{\restriction}_{tr_2}$. There are eight cases:

- $a = get\_public\_encrypt(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_public\_encrypt(A)$, we can identify two cases: $s_3.ek_A = \perp$ or $s_3.ek_A \neq \perp$. If $s_3.ek_A = \perp$, then for each state s_3' of $\mathcal{E}_k^3$, $\mu_3(s_3') = \rho(e, d)$ and s_3' is identified by the same values of s_3 except for the following values: $s_3'.ek_A = e$, $s_3'.dk_A = d$, and $s_3'.pk_A = e$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_3{\restriction}_{s_2}$. By definition of action $get\_public\_encrypt(A)$, s_2 enables $get\_public\_encrypt(A)$ that leads to the measure μ_2 such that for each state s_2' of $\mathcal{E}_k^2$, $\mu_2(s_2') = \rho(e, d)$ and s_2' is identified by the same values of s_2 except for the following values: $s_2'.ek_A = e$, $s_2'.dk_A = d$, and $s_2'.pk_A = e$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Thus $\mu_2 = \mu_3{\restriction}_{\mu_2}$, and hence $tr_2 = tr_3{\restriction}_{tr_2}$.
  If $s_3.ek_A \neq \perp$, then $\mu_3 = \delta_{s_3'}$ where s_3' is the state of $\mathcal{E}_k^3(\mathbb{A})$ that is identified by the same values of s_3 except for pk_A where $s_3'.pk_A = s_3.ek_A$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_3{\restriction}_{s_2}$. By definition of $get\_public\_encrypt(A)$, s_2 enables $get\_public\_encrypt(A)$ that leads to the measure $\delta_{s_2'}$ where s' is the state of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable pk_A where $s_2'.pk_A = s_2.ek_A$. Thus $\delta_{s_2'} = \delta_{s_3'}{\restriction}_{\delta_{s_2'}}$, and hence $tr_2 = tr_3{\restriction}_{tr_2}$.
- $a = ret\_public\_encrypt(A, E)$ for some $A \in \mathbb{A}$ and $E \in$ EKey: by definition of action $ret\_public\_encrypt(A, E)$, it follows that $s_3.pk_A = E$ and that $\mu_3 = \delta_{s_3'}$ where s_3' is the state of $\mathcal{E}_k^3(\mathbb{A})$ that is identified by the same values of s_3 except for pk_A where $s_3'.pk_A = \perp$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_3{\restriction}_{s_2}$. By definition of action $ret\_public\_encrypt(A, E)$, s_2 enables $ret\_public\_encrypt(A, E)$ that leads to the measure $\delta_{s_2'}$ where s_2' is the state of $\mathcal{E}_k^2(\mathbb{A})$ that

is identified by the same values of s_2 except for variable pk_A where $s_2'.pk_A = \perp$. Thus $\delta_{s_2'} = \delta_{s_3'}\lceil_{\delta_{s_2'}}$, and hence $tr_2 = tr_3\lceil_{tr_2}$.

- $a = get\_corrupt\_encrypt(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_corrupt\_encrypt(A)$, we can identify two cases: $s_3.ek_A = \perp$ or $s_3.ek_A \neq \perp$. If $s_3.ek_A = \perp$, then for each state s_3' of $\mathcal{E}_k^3$, $\mu_3(s_3') = \rho(e,d)$ and s_3' is identified by the same values of s_3 except for the following values: $s_3'.ek_A = e$, $s_3'.dk_A = d$, and $s_3'.ck_A = (e,d)$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_3\lceil_{s_2}$. By definition of action $get\_corrupt\_encrypt(A)$, s_2 enables $get\_corrupt\_encrypt(A)$ that leads to the measure μ_2 such that for each state s_2' of $\mathcal{E}_k^2$, $\mu_2(s_2') = \rho(e,d)$ and s_2' is identified by the same values of s_2 except for the following values: $s_2'.ek_A = e$, $s_2'.dk_A = d$, and $s_2'.ck_A = (e,d)$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Thus $\mu_2 = \mu_3\lceil_{\mu_2}$, and hence $tr_2 = tr_3\lceil_{tr_2}$.
  If $s_3.ek_A \neq \perp$, then $\mu_3 = \delta_{s_3'}$ where s_3' is the state of $\mathcal{E}_k^3(\mathbb{A})$ that is identified by the same values of s_3 except for ck_A where $s_3'.ck_A = (s_3.ek_A, s_3.dk_A)$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_3\lceil_{s_2}$. By definition of action $get\_corrupt\_encrypt(A)$, s_2 enables $get\_corrupt\_encrypt(A)$ that leads to the measure δ_{s_2} where s_2' is the state of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable ck_A where $s_2'.ck_A = (s_2.ek_A, s_2.dk_A)$. Thus $\delta_{s_2'} = \delta_{s_3'}\lceil_{\delta_{s_2'}}$, and hence $tr_2 = tr_3\lceil_{tr_2}$.
- $a = ret\_corrupt\_encrypt(A,E,D)$ for some $A \in \mathbb{A}$, $E \in$ EKey, and $D \in$ DKey: by definition of action $ret\_corrupt\_encrypt(A,E,D)$, it follows that $s_3.ck_A = (E,D)$ and that $\mu_3 = \delta_{s_3'}$ where s_3' is the state of $\mathcal{E}_k^3(\mathbb{A})$ that is identified by the same values of s_3 except for ck_A where $s_3'.ck_A = \perp$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_3\lceil_{s_2}$. By definition of action $ret\_corrupt\_encrypt(A,E,D)$, s_2 enables $ret\_corrupt\_encrypt(A,E,D)$ that leads to the measure δ_{s_2} where s_2' is the state of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable ck_A where $s_2'.ck_A = \perp$. Thus $\delta_{s_2'} = \delta_{s_3'}\lceil_{\delta_{s_2'}}$, and hence $tr_2 = tr_3\lceil_{tr_2}$.
- $a = get\_encrypt(A,M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of the action $get\_encrypt(A,M)$, it follows that for each state s_3' of $\mathcal{E}_k^3$, $\mu_3(s_3') = \rho(C)$ and s_3' is identified by the same values of s_3 except for the following values: $s_3'.is\_fresh_A = F$ if

$C \in s_3.generated\_ciphers$, T otherwise, $s'_3.enc\_value_A = C$, and $s'_3.generated\_ciphers = s_3.generated\_ciphers \cup \{C\}$. ρ is the probability measure induced over Ciphertext by the probabilistic algorithm $\texttt{Enc}(s_3.ek_A, M)$. Let s_2 be the state of $\mathcal{E}^2_k(\mathbb{A})$ such that $s_2 = s_3{\restriction}_{s_2}$. By definition of action $get\_encrypt(A, M)$, s_2 enables $get\_encrypt(A, M)$ that leads to the measure ρ where for each state s'_2 of $\mathcal{E}^2_k(\mathbb{A})$ that is identified by the same values of s_2 except for variable enc_value_A, we have that $\mu(s'_2) = \rho(C)$ and $s'_2.enc\_value_A = C$. Thus $\mu_2 = \mu_3{\restriction}_{\mu_2}$, and hence $tr_2 = tr_3{\restriction}_{tr_2}$.

- $a = ret\_encrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of action $ret\_encrypt(A, C)$, it follows that $s_3.enc\_value_A = C$ and that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state of $\mathcal{E}^3_k(\mathbb{A})$ that is identified by the same values of s_3 except for enc_value_A that assumes the value $s'_3.enc\_value_A = \perp$ and for is_fresh_A and $is\_not\_used_A$ that are reset to $\perp$. Let s_2 be the state of $\mathcal{E}^2_k(\mathbb{A})$ such that $s_2 = s_3{\restriction}_{s_2}$. By definition of action $ret\_encrypt(A, C)$, s_2 enables $ret\_encrypt(A, C)$ that leads to the measure $\delta_{s'_2}$ where s'_2 is the state of $\mathcal{E}^2_k(\mathbb{A})$ that is identified by the same values of s_2 except for variable enc_value_A where $s'_2.enc\_value_A = \perp$. Thus $\delta_{s'_2} = \delta_{s'_3}{\restriction}_{\delta_{s'_2}}$, and hence $tr_2 = tr_3{\restriction}_{tr_2}$.
- $a = get\_decrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of action $get\_decrypt(A, C)$, it follows that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state of $\mathcal{E}^3_k(\mathbb{A})$ that is identified by the same values of s_3 except for dec_value_A where $s'_3.dec\_value_A = \texttt{Dec}(s_3.dk_A, C)$. Let s_2 be the state of $\mathcal{E}^2_k(\mathbb{A})$ such that $s_2 = s_3{\restriction}_{s_2}$. By definition of action $get\_decrypt(A, C)$, s_2 enables $get\_decrypt(A, C)$ that leads to the measure $\delta_{s'_2}$ where s'_2 is the state of $\mathcal{E}^2_k(\mathbb{A})$ that is identified by the same values of s_2 except for variable dec_value_A where $s'_2.dec\_value_A = \texttt{Dec}(s_2.dk_A, C)$. Thus $\delta_{s'_2} = \delta_{s'_3}{\restriction}_{\delta_{s'_2}}$, and hence $tr_2 = tr_3{\restriction}_{tr_2}$.
- $a = ret\_decrypt(A, M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of action $ret\_decrypt(A, M)$, it follows that $s_3.dec\_value_A = M$ and that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state of $\mathcal{E}^3_k(\mathbb{A})$ that is identified by the same values of s_3 except for dec_value_A where $s'_3.dec\_value_A = \perp$. Let s_2 be the state of $\mathcal{E}^2_k(\mathbb{A})$ such that $s_2 = s_3{\restriction}_{s_2}$. By definition of action $ret\_decrypt(A, M)$, s_2 enables $ret\_decrypt(A, M)$ that leads to the measure $\delta_{s'_2}$ where s'_2 is the state of $\mathcal{E}^2_k(\mathbb{A})$ that is identi-

fied by the same values of s_2 except for variable dec_value_A where $s_2'.dec\_value_A = \perp$. Thus $\delta_{s_2'} = \delta_{s_3'}\lceil_{\delta_{s_2'}}$, and hence $tr_2 = tr_3\lceil_{tr_2}$.

Since the requirements of Definition 4.5 are satisfied, for each $k \in \mathbb{N}$, $\mathcal{E}_k^3(\mathbb{A}) \in \mathrm{Ext}_B^W(\mathcal{E}_k^2(\mathbb{A}))$. □

The existence of a polynomially accurate simulation from $\mathcal{E}_k^2(\mathbb{A})$ to $\mathcal{E}_k^3(\mathbb{A})$ in now straightforward:

Proposition 6.17. *Let $\mathbb{A}$ be a set of (identities of) agents. For each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k^2(\mathbb{A})$ such that $\{used\_ciphers(CT) \mid CT \subseteq \mathrm{Ciphertext}\} \subseteq A_{\mathcal{C}_k}$,*

$$\{\mathcal{E}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.16, for each $k \in \mathbb{N}$, $\mathcal{E}_k^3(\mathbb{A}) \in \mathrm{Ext}_B^W(\mathcal{E}_k^2(\mathbb{A}))$ where B is the set of actions $\{used\_ciphers(CT) \mid CT \subseteq \mathrm{Ciphertext}\}$ and W is $\{used\_ciphers\} \cup \{is\_not\_used_A \mid A \in \mathbb{A}\}$. This implies, by Lemma 4.6, that for each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k^2(\mathbb{A})$ such that $B \subseteq A_{\mathcal{C}_k}$, $\mathcal{E}_k^2(\mathbb{A})||\mathcal{C}_k \preceq \mathcal{E}_k^3(\mathbb{A})||\mathcal{C}_k$ and thus, by Proposition 5.6, $\{\mathcal{E}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

For the second step, let $\mathcal{E}_k^4$ be the automaton obtained from $\mathcal{E}_k^3$ modifying each $get\_encrypt(A, M)$ action as follows: the encryption algorithm `Enc` is invoked until it returns a value that is a ciphertext that is not yet used, that is not inside $used\_ciphers$, as depicted in Figure 6.8.

It is immediate to see that $\mathcal{E}_k^4(\mathbb{A})$ simulates the G_k-conditional of $\mathcal{E}_k^3(\mathbb{A})$ where for each $k \in \mathbb{N}$, G_k is the set of states of $\mathcal{E}_k^3(\mathbb{A})$ such that for each $A \in \mathbb{A}$, $s.is\_not\_used_A \neq F$. In fact, we iterate the generation of the encryption of M under the public key of A until it returns a value that is not equal to an already used ciphertext. That is, we condition the choice of the ciphertext to the fact that it is fresh (and thus, $s.is\_not\_used_A \neq F$ is always satisfied).

Note that $\mathcal{E}_k^4(\mathbb{A})$ is not the G_k-conditional of $\mathcal{E}_k^3(\mathbb{A})$ since it provides more transitions than $\mathcal{E}_k^3(\mathbb{A})|G_k$. In fact, $\mathcal{E}_k^4(\mathbb{A})$ enables transitions that leave from a state not in G_k and that leads to a probability measures μ such that $\mu(G_k) = 0$. For example, let s be the state such that $s.is\_not\_used_A = F$, $s.is\_not\_used_B = F$, $s.enc\_value_A = C$. Thus, by definition of $ret\_encrypt$ actions, s enables the action $ret\_encrypt(A, C)$ that leads to the measure $\delta_{s'}$ where in s' only enc_value_A and is_fresh_A are different with respect to the values in s. Thus $s'.is\_not\_used_B$ is still F and hence $s' \notin G_k$.

The above intuition is formalized by the following result:

$\mathcal{E}_k^3$

Input $get\_encrypt(A, M)$
Effect:
repeat
$c = \mathtt{Enc}(ek_A, M)$
until $c \notin generated\_ciphers$
$enc\_value_A := c$
$$is\_fresh_A := \begin{cases} F & \text{if } c \in generated\_ciphers \\ T & \text{otherwise} \end{cases}$$
$$is\_not\_used_A := \begin{cases} F & \text{if } c \in used\_ciphers \\ T & \text{otherwise} \end{cases}$$
$generated\_ciphers := generated\_ciphers \cup \{c\}$

$\mathcal{E}_k^4$

Input $get\_encrypt(A, M)$
Effect:
repeat
repeat
$c = \mathtt{Enc}(ek_A, M)$
until $c \notin generated\_ciphers$
until $c \notin used\_ciphers$
$enc\_value_A := c$
$$is\_fresh_A := \begin{cases} F & \text{if } c \in generated\_ciphers \\ T & \text{otherwise} \end{cases}$$
$$is\_not\_used_A := \begin{cases} F & \text{if } c \in used\_ciphers \\ T & \text{otherwise} \end{cases}$$
$generated\_ciphers := generated\_ciphers \cup \{c\}$

Fig. 6.8. *get_encrypt* of $\mathcal{E}_k^3$ and $\mathcal{E}_k^4$

Lemma 6.18. *Given $\mathcal{E}_k^3(\mathbb{A})$, let B_k be the set of states of $\mathcal{E}_k^3$ such that $s \in B_k$ if there exists $A \in \mathbb{A}$ such that $s.is\_not\_used_A = F$. Let G_k be the set $S_k^3 \setminus B_k$.*

For each $k \in \mathbb{N}$, $\mathcal{E}_k^3(\mathbb{A})|G_k \preceq \mathcal{E}_k^4(\mathbb{A})$.

Proof. For each $k \in \mathbb{N}$, let id_k be the identity relation on states. id_k is a simulation from $\mathcal{E}_k^3(\mathbb{A})|G_k$ to $\mathcal{E}_k^4(\mathbb{A})$.

The condition on start states is trivially true: let $\bar{s}_k^3$ be the start state of $\mathcal{E}_k^3(\mathbb{A})|G_k$ and $\bar{s}_k^4$ be the start state of $\mathcal{E}_k^4(\mathbb{A})$. By definition of conditional, it follows that $\bar{s}_k^3$ is the start state of $\mathcal{E}_k^3(\mathbb{A})$. Since by definition of $\mathcal{E}_k^4(\mathbb{A})$, the

only difference between $\mathcal{E}_k^3(\mathbb{A})$ and $\mathcal{E}_k^4(\mathbb{A})$ is on the definition of the action $get\_encrypt(A, M)$, we have that $\bar{s}_k^4 = \bar{s}_k^3$ and thus $\bar{s}_k^3\ id_k\ \bar{s}_k^4$.

For the step condition, let s_3 and s_4 be two states of $\mathcal{E}_k^3(\mathbb{A})|G_k$ and $\mathcal{E}_k^4(\mathbb{A})$, respectively, such that $s_3\ id_k\ s_4$. Let (s_3, a, μ_3) be a transition of $\mathcal{E}_k^3(\mathbb{A})|G_k$ that leaves from s_3. We must find μ_4 such that (s_4, a, μ_4) is a transition of $\mathcal{E}_k^4(\mathbb{A})$ and $\mu_3\ \mathcal{L}(id_k)\ \mu_4$. There are nine cases:

- $a = get\_public\_encrypt(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_public\_encrypt(A)$, we can identify two cases: $s_3.ek_A = \perp$ or $s_3.ek_A \neq \perp$. If $s_3.ek_A = \perp$, then for each state s_3' of $\mathcal{E}_k^3$, $\mu_3(s_3') = \rho_3(e, d)$ and s_3' is identified by the same values of s_3 except for the following values: $s_3'.ek_A = e$, $s_3'.dk_A = d$, and $s_3'.pk_A = e$ where ρ_3 is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Since $s_3\ id_k\ s_4$, we have that also s_4 satisfies $s_4.ek_A = \perp$, and hence for each state s_4' of $\mathcal{E}_k^4$, $\mu_4(s_4') = \rho_4(e, d)$ and s_4' is identified by the same values of s_4 except for the following values: $s_4'.ek_A = e$, $s_4'.dk_A = d$, and $s_4'.pk_A = e$ where ρ_4 is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. This implies that $s_3'\ id_k\ s_4'$ for each state s_3' and s_4' satisfying above conditions and since $\rho_3 = \rho_4$, we have that $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
  If $s_3.ek_A \neq \perp$, then $\mu_3 = \delta_{s_3'}$ where s_3' is the state of $\mathcal{E}_k^3(\mathbb{A})$ that is identified by the same values of s_3 except for pk_A where $s_3'.pk_A = s_3.ek_A$. Since $s_3\ id_k\ s_4$, we have that also s_4 satisfies $s_4.ek_A \neq \perp$, and hence by definition of action $get\_public\_encrypt(A)$, s_4 enables $get\_public\_encrypt(A)$ that leads to the measure $\delta_{s_4'}$ where s_4' is the state of $\mathcal{E}_k^4(\mathbb{A})$ that is identified by the same values of s_4 except for variable pk_A where $s_4'.pk_A = s_4.ek_A$. This implies that $s_3'\ id_k\ s_4'$ and thus $\delta_{s_3'}\ \mathcal{L}(id_k)\ \delta_{s_4'}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
- $a = ret\_public\_encrypt(A, E)$ for some $A \in \mathbb{A}$ and $E \in$ EKey: by definition of the action $ret\_public\_encrypt(A, E)$, it follows that s_3 satisfies $s_3.pk_A = E$. Moreover, it follows that $\mu_3 = \delta_{s_3'}$ where s_3' is the state such that $s_3'.pk_A = \perp$, and all other variables that describe s_3' have the same value of the variables that describe s_3. Since $s_3\ id_k\ s_4$, we have that also s_4 satisfies $s_4.pk_A = E$ and thus it enables the transition $(s_4, ret\_public\_encrypt(A, E), \mu_4)$ where μ_4 is the measure $\delta_{s_4'}$ where s_4' is the state such that $s_4'.pk_A = \perp$, and all other variables that describe s_4' have the same value of the variables that describe s_4. This implies that $s_3'\ id_k\ s_4'$ and thus $\delta_{s_3'}\ \mathcal{L}(id_k)\ \delta_{s_4'}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.

- $a = \mathit{get\_corrupt\_encrypt}(A)$ for some $A \in \mathbb{A}$: by definition of the action $\mathit{get\_corrupt\_encrypt}(A)$, we can identify two cases: $s_3.ek_A = \bot$ or $s_3.ek_A \neq \bot$. If $s_3.ek_A = \bot$, then for each state s'_3 of $\mathcal{E}^3_k$, $\mu_3(s'_3) = \rho_3(e, d)$ and s'_3 is identified by the same values of s_3 except for the following values: $s'_3.ek_A = e$, $s'_3.dk_A = d$, and $s'_3.ck_A = (e, d)$ where ρ_3 is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Since $s_3\ id_k\ s_4$, we have that also s_4 satisfies $s_4.ek_A = \bot$, and hence for each state s'_4 of $\mathcal{E}^4_k$, $\mu_4(s'_4) = \rho_4(e, d)$ and s'_4 is identified by the same values of s_4 except for the following values: $s'_4.ek_A = e$, $s'_4.dk_A = d$, and $s'_4.ck_A = (e, d)$ where ρ_4 is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. This implies that $s'_3\ id_k\ s'_4$ for each state s'_3 and s'_4 satisfying above conditions and since $\rho_3 = \rho_4$, we have that $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
  If $s_3.ek_A \neq \bot$, then $\mu_3 = \delta_{s'_3}$ where s'_3 is the state of $\mathcal{E}^3_k(\mathbb{A})$ that is identified by the same values of s_3 except for ck_A where $s'_3.ck_A = (s_3.ek_A, s_3.dk_A)$. Since $s_3\ id_k\ s_4$, we have that also s_4 satisfies $s_4.ek_A \neq \bot$, and hence by definition of action $\mathit{get\_corrupt\_encrypt}(A)$, s_4 enables $\mathit{get\_corrupt\_encrypt}(A)$ that leads to the measure $\delta_{s'_4}$ where s'_4 is the state of $\mathcal{E}^4_k(\mathbb{A})$ that is identified by the same values of s_4 except for variable ck_A where $s'_4.ck_A = (s_4.ek_A, s_4.dk_A)$. This implies that $s'_3\ id_k\ s'_4$ and thus $\delta_{s'_3}\ \mathcal{L}(id_k)\ \delta_{s'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
- $a = \mathit{ret\_corrupt\_encrypt}(A, E, D)$ for some $A \in \mathbb{A}$, $E \in$ EKey and $V \in$ VKey: by definition of the action $\mathit{ret\_corrupt\_encrypt}(A, E, D)$, it follows that s_3 satisfies $s_3.ck_A = (E, D)$. Moreover, it follows that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state such that $s'_3.ck_A = \bot$, and all other variables that describe s'_3 have the same value of the variables that describe s_3. Since $s_3\ id_k\ s_4$, we have that also s_4 satisfies $s_4.ck_A = (E, D)$ and thus it enables the transition $(s_4, \mathit{ret\_corrupt\_encrypt}(A, E, D), \mu_4)$ where μ_4 is the measure $\delta_{s'_4}$ where s'_4 is the state such that $s'_4.ck_A = \bot$, and all other variables that describe s'_4 have the same value of the variables that describe s_4. This implies that $s'_3\ id_k\ s'_4$ and thus $\delta_{s'_3}\ \mathcal{L}(id_k)\ \delta_{s'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
- $a = \mathit{get\_encrypt}(A, M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of $\mathit{get\_encrypt}(A, M)$, it follows that for each state s'_3 of $\mathcal{E}^3_k$, $\mu_3(s'_3) = \rho_3(C)$ and s'_3 is identified by the same values of s_3 except for the following values: $s'_3.\mathit{is\_fresh}_A = F$ if $C \in s_3.\mathit{generated\_ciphers}$, T otherwise, $s'_3.\mathit{enc\_value}_A = C$, and $s'_3.\mathit{generated\_ciphers} = s_3.\mathit{generated\_ciphers} \cup \{C\}$. ρ_3 is the probability measure induced over Ciphertext by the proba-

bilistic algorithm $\texttt{Enc}(s_3.ek_A, M)$ conditioned to the set of states where for each $A \in \mathbb{A}$ $is\_fresh_A \neq F$ conditioned to G_k. Since s_3 id_k s_4, we have that also s_4 enables the transition $(s_4, get\_encrypt(A, M), \mu_4)$ where for each state s'_4 of $\mathcal{E}_k^4$, $\mu_4(s'_4) = \rho_4(C)$ and s'_4 is identified by the same values of s_4 except for the following values: $s'_4.is\_fresh_A = F$ if $C \in s_4.generated\_ciphers$, T otherwise, $s'_4.enc\_value_A = C$, and $s'_4.generated\_ciphers = s_4.generated\_ciphers \cup \{C\}$. ρ_4 is the probability measure induced over Ciphertext by the probabilistic algorithm $\texttt{Enc}(s_4.ek_A, M)$ conditioned to the set of states where for each $A \in \mathbb{A}$ $is\_fresh_A \neq F$ that is iterated until it returns a value that does not belong to $s_4.used\_ciphers$. This implies that s'_3 id_k s'_4 for each state s'_3 and s'_4 satisfying above conditions and if $\rho_3 = \rho_4$, then μ_3 $\mathcal{L}(id_k)$ μ_4, as required.

ρ_3 is equal to ρ_4 since for each $s \in G_k$, $\rho_3(s) = \rho_4(s)$. In fact, by definition of conditional, $\rho_3(s) = \rho(s)/\rho(G_k)$ where ρ is the probability measure induced over Ciphertext by $\texttt{Enc}(s_3.ek_A, M)$, $\rho(G_k) > 0$ and $\rho(B_k) < 1$. By definition of $get\_encrypt(A, M)$ action of $\mathcal{E}_k^4$, we have that $\rho_4(s) = \sum_{i=0}^{+\infty} \rho(B_k)^i \rho(s) = \rho(s) \sum_{i=0}^{+\infty} \rho(B_k)^i = \rho(s) \dfrac{1}{1 - \rho(B_k)} = \rho(s)/\rho(G_k) = \rho_3(s)$.

- $a = ret\_encrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of the action $ret\_encrypt(A, C)$, it follows that s_3 satisfies $s_3.enc\_value_A = C$. Moreover, it follows that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state such that $s'_3.enc\_value_A = \perp$, $s'_3.is\_fresh_A = \perp$, and all other variables that describe s'_3 have the same value of the variables that describe s_3. Since s_3 id_k s_4, we have that also s_4 satisfies $s_4.enc\_value_A = C$ and thus it enables the transition $(s_4, ret\_encrypt(A, C), \mu_4)$ where μ_4 is the measure $\delta_{s'_4}$ where s'_4 is the state such that $s'_4.enc\_value_A = \perp$, $s'_4.is\_fresh_A = \perp$, $s'_4.is\_not\_used_A = \perp$ and all other variables that describe s'_4 have the same value of the variables that describe s_4. This implies that s'_3 id_k s'_4 and thus $\delta_{s'_3}$ $\mathcal{L}(id_k)$ $\delta_{s'_4}$, that is μ_3 $\mathcal{L}(id_k)$ μ_4, as required.
- $a = get\_decrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of the action $get\_decrypt(A, C)$, it follows that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state such that $s'_3.dec\_value_A = \texttt{Dec}(s_3.dk_A, C)$, and all other variables that describe s'_3 have the same value of the variables that describe s_3. Since s_3 id_k s_4, we have that also s_4 enables the transition $(s_4, get\_decrypt(A, C), \mu_4)$ where μ_4 is the measure $\delta_{s'_4}$ where s'_4 is the

state such that $s'_4.dec\_value_A = \mathtt{Dec}(s_4.dk_A, C)$, and all other variables that describe s'_4 have the same value of the variables that describe s_4. This implies that $s'_3\ id_k\ s'_4$ and thus $\delta_{s'_3}\ \mathcal{L}(id_k)\ \delta_{s'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.

- $a = ret\_decrypt(A, M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of the action $ret\_decrypt(A, M)$, it follows that s_3 satisfies $s_3.dec\_value_A = M$. Moreover, it follows that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state such that $s'_3.dec\_value_A = \bot$, and all other variables that describe s'_3 have the same value of the variables that describe s_3. Since $s_3\ id_k\ s_4$, we have that also s_4 satisfies $s_4.dec\_value_A = M$ and thus it enables the transition $(s_4, ret\_decrypt(A, M), \mu_4)$ where μ_4 is the measure $\delta_{s'_4}$ where s'_4 is the state such that $s'_4.dec\_value_A = \bot$, and all other variables that describe s'_4 have the same value of the variables that describe s_4. This implies that $s'_3\ id_k\ s'_4$ and thus $\delta_{s'_3}\ \mathcal{L}(id_k)\ \delta_{s'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
- $a = used\_ciphers(CT)$ for some $CT \subseteq$ Ciphertext: by definition of $used\_ciphers(CT)$, it follows that $\mu_3 = \delta_{s'_3}$ where s'_3 is the state such that $s'_3.used\_ciphers = s_3.used\_ciphers \cup CT$, and all other variables that describe s'_3 have the same value of the variables that describe s_3. Since $s_3\ id_k\ s_4$, we have that also s_4 enables the transition $(s_4, used\_ciphers(CT), \mu_4)$ where μ_4 is the measure $\delta_{s'_4}$ where s'_4 is the state such that $s'_4.used\_ciphers = s_4.used\_ciphers \cup CT$, and all other variables that describe s'_4 have the same value of the variables that describe s_4. This implies that $s'_3\ id_k\ s'_4$ and thus $\delta_{s'_3}\ \mathcal{L}(id_k)\ \delta_{s'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.

Since for each action a, if s_3 enables a transition labelled by a that leads to μ_3, then we can find μ_4 such that $(s_4, a, \mu_4) \in D_4$ and $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, the step condition is satisfied. □

It is straightforward to prove that $\mathcal{E}^3_k(\mathbb{A})|G_k$ is polynomially simulated by $\mathcal{E}^4_k(\mathbb{A})$:

Proposition 6.19. *Given $\mathcal{E}^3_k(\mathbb{A})$, let B_k be the set of states of $\mathcal{E}^3_k$ such that $s \in B_k$ if there exists $A \in \mathbb{A}$ such that $s.is\_not\_used_A = F$. Let G_k be the set $S^3_k \setminus B_k$.*

For each context $\mathcal{C}_k$ compatible with $\mathcal{E}^3_k(\mathbb{A})$,

$$\{(\mathcal{E}^3_k(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}^4_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.18, we have that for each $k \in \mathbb{N}$, $\mathcal{E}_k^3(\mathbb{A})|G_k \preceq \mathcal{E}_k^4(\mathbb{A})$. This implies that for each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k^3(\mathbb{A})$, $(\mathcal{E}_k^3(\mathbb{A})|G_k)||\mathcal{C}_k \preceq \mathcal{E}_k^4(\mathbb{A})||\mathcal{C}_k$. Finally, by Proposition 5.6, we have that $\{(\mathcal{E}_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^4(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

The complete the chain of polynomially accurate simulations from $\mathcal{E}_k^3(\mathbb{A})$ to $\mathcal{E}_k^4(\mathbb{A})$ we need to prove:

Proposition 6.20. *Let* E *be an IND-CCA encryption scheme and* $\mathbb{A}$ *be a set of agents. Let* G_k *be the set of states of* $\mathcal{E}_k^3$ *such that* $s \in G_k$ *if and only if for each* $A \in \mathbb{A}$ $s.is\_not\_used_A \neq F$*, that is, the generated ciphertext is fresh. Let* B_k *be* $S_k^3 \setminus G_k$ *(that is, states* s' *of* $\mathcal{E}_k^3$ *such that* $s'.is\_not\_used_A = F$ *for some* $A \in \mathbb{A}$*).*

For each context $\mathcal{C}_k$ *compatible with* $\mathcal{E}_k^4(\mathbb{A})$*, if there exists* $q \in Poly$ *such that for each action* $used\_ciphers(CT)$ *of* $\mathcal{C}_k$*,* $|CT| < q(k)$*, then*

$$\{\mathcal{E}_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{E}_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. Theorem 5.8 states that $\{\mathcal{E}_k^3(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^3(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$ if and only if $\{B_k\}_{k\in\mathbb{N}}$ is negligible in $\{\mathcal{E}_k^3(\mathbb{A})\}_{k\in\mathbb{N}}$.

Suppose, for the sake of contradiction, that $\{\mathcal{E}_k^3(\mathbb{A})\}_{k\in\mathbb{N}}$ is not simulated by $\{\mathcal{E}_k^3(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$. This implies that $\{B_k\}_{k\in\mathbb{N}}$ is not negligible in $\{\mathcal{E}_k^3(\mathbb{A})\}_{k\in\mathbb{N}}$ and thus that there exists $c \in \mathbb{N}$, $p \in Poly$ such that for each $\bar{k} \in \mathbb{N}$ there exists $k > \bar{k}$ such the probability to reach states of B_k within $p(k)$ steps is at least k^{-c}, that is, the probability to reach states s such that $s.is\_not\_used_A = F$ for some $A \in \mathbb{A}$ within $p(k)$ steps is at least k^{-c}. By definition of the automaton $\mathcal{E}_k^3(\mathbb{A})$, it follows that $s.is\_not\_used_A$ can assume value F only as the effect of a transition that leaves from some state s' and that is labelled by action $get\_encrypt(A, M)$. This happens only when the ciphertext value $\mathtt{Enc}(s'.ek_A, M)$ assigned to $s.enc\_value_A$ belongs to the set $UC = s'.used\_ciphers$. Within $p(k)$ steps, we can perform at most $p(k)$ transitions labelled by $used\_ciphers(CT)$. Since by hypothesis we add at most $q(k)$ values to $used\_ciphers$ each time we perform a transition labelled by $used\_ciphers(CT)$, within $p(k)$ steps the set UC has cardinality at most $p(k)q(k)$. This means that with probability at least k^{-c}, we have generated a ciphertext that it is equal to some $c \in UC$ with $|UC| \leqslant p(k)q(k)$. Since the encryption scheme E is IND-CCA, this contradicts Proposition 2.4 and thus $\{\mathcal{E}_k^3(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^3(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$. This implies, by Theorem 5.10, that $\{\mathcal{E}_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{E}_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

Second approach: messages and decryption keys

We can adopt also the second approach, since we simply need to receives messages instead of ciphertexts. Let $\mathcal{E}_k^5$ be the automaton obtained from $\mathcal{E}_k^2$ adding a set of used ciphertexts $used\_ciphers \subseteq$ Ciphertext initialized to $\emptyset$, a set of variables $is\_not\_used_A \in \{T, F, \bot\}$ all initialized to $\bot$ where $A \in \mathbb{A}$ and one input action $send(m)$:

Input $send(m)$
  Effect:
    $used\_ciphers := used\_ciphers \cup extractCiphertexts(decrypt\_key, m)$

where $decrypt\_key$ is a vector such that for each $A \in \mathbb{A}$, $decrypt\_key(A) = dk_A$.

Note that the effect of the $send(m)$ input action depends on the function $extractCiphertexts(decrypt\_key, m)$ (depicted in Figure 6.9) that parses recursively m and uses keys of $decrypt\_key$ to decrypt ciphertexts. We assume that there exists a function $agent$ that, on input a ciphertext c, returns the identity of the agent A which is associated to the encryption key used to obtain c and that $decrypt\_key(A)$ is not $\bot$. We suppose that there exists a function type that, given a message, returns the type of the message that can be a nonce, a ciphertexts, an identity, a signature, or a pairing. We assume also that keys and ciphertexts are obtained using the encryption scheme $\mathsf{E} = (\mathtt{KGen}, \mathtt{Enc}, \mathtt{Dec})$.

Moreover, each $get\_encrypt(A, M)$ action modifies $is\_not\_used_A$ assigning F if the ciphertext returned by $\mathtt{Enc}(ek_A, M)$ is in $used\_ciphers$; T otherwise as depicted in Figure 6.10. The action $ret\_encrypt(A, C)$ reset the $is\_not\_used_A$ to its initial value $\bot$. Let $\mathcal{E}_k^5$ be the resulting automaton.

It is immediate to see that $\mathcal{E}_k^5(\mathbb{A})$ is an extension of $\mathcal{E}_k^2(\mathbb{A})$. In fact, we simply add some history variables that keep information about all encryption generated by the environment. Such variables do not affect the behavior of the other actions and thus each transition of $\mathcal{E}_k^5(\mathbb{A})$ is either almost identical to a transition of $\mathcal{E}_k^2(\mathbb{A})$ or it is the new action that does not modify the state variables that describe states of $\mathcal{E}_k^2(\mathbb{A})$.

The above intuition is formalized by the following result:

Lemma 6.21. *Let $\mathbb{A}$ be a set of agents, B be the set $\{send(m) \mid m \in$ Message$\}$ and W be the set $\{used\_ciphers\} \cup \{is\_not\_used_A \mid A \in \mathbb{A}\}$ where actions and variables are defined as above. For each $k \in \mathbb{N}$, $\mathcal{E}_k^5(\mathbb{A}) \in \mathrm{Ext}_B^W(\mathcal{E}_k^2(\mathbb{A}))$.*

```
function extractCiphertexts(decrypt_key, message)
  if type(message) =id then
    used_ciphers := ∅
  fi
  if type(message) = nonce then
    used_ciphers := ∅
  fi
  if type(message) = pair then
    used_ciphers := extractCiphertexts(decrypt_key, left(message)) ∪
                    extractCiphertexts(decrypt_key, right(message))
  fi
  if type(message) = signature then
    used_ciphers := extractCiphertexts(decrypt_key, msg(message))
  fi
  if type(message) = encryption then
    A := agent(message)
    plaintext := Dec(decrypt_key(A), message)
    used_ciphers := {message} ∪ extractCiphertexts(decrypt_key, plaintext)
  fi
  return used_ciphers
```

Fig. 6.9. The *extractCiphertexts* function

Proof. To prove the statement of the Lemma, we need to check if the requirements of Definition 4.5 are satisfied:

compatible states: let v be a state variable of $\mathcal{E}_k^5(\mathbb{A})$. Then v is either one of *generated_ciphers*, ek_A, dk_A, pk_A, ck_A, enc_value_A, dec_value_A, is_fresh_A for $A \in \mathbb{A}$ and thus it is a state variable of $\mathcal{E}_k^2(\mathbb{A})$, or v is *used_ciphers* or $is\_not\_used_A$ for some $A \in \mathbb{A}$ and thus $v \in W$;

compatible start state: $\bar{s}_k^5$ is identified by value $\perp$ for each ek_A, dk_A, pk_A, ck_A, enc_value_A, dec_value_A, is_fresh_A, and $is\_not\_used_A$ (with $A \in \mathbb{A}$), and by $\emptyset$ for *generated_ciphers* and *used_ciphers*. Since $\bar{s}_k^2$ is identified by the value $\perp$ for each ek_A, dk_A, pk_A, ck_A, enc_value_A, dec_value_A, and is_fresh_A (with $A \in \mathbb{A}$), and by $\emptyset$ for *generated_ciphers*, then $\bar{s}_k^2 = \bar{s}_k^5 \restriction_{\bar{s}_k^2}$;

compatible actions: by definition of $\mathcal{E}_k^5(\mathbb{A})$, it provides the same actions of $\mathcal{E}_k^2(\mathbb{A})$ plus the actions $send(m)$ that belong to B;

compatible transitions: let $tr_5 = (s_5, a, \mu_5) \in D_k^5$ be a transition of $\mathcal{E}_k^5$. Suppose that $a = send(m)$. Then we must verify that there exists a state s_2 of $\mathcal{E}_k^2$ such that $s_2 = s_5 \restriction_{s_2}$ and $\delta_{s_2} = \mu_5 \restriction_{\delta_{s_2}}$. By definition of $send(m)$, it follows that $\mu_5 = \delta_{s_5'}$ where s_5' is the state of $\mathcal{E}_k^5(\mathbb{A})$ that is identified by the same values of s_5 except for *used_ciphers* where $s_5'.used\_ciphers =$

$\mathcal{E}_k^2(\mathbb{A})$

Input $get\_encrypt(A, M)$
Effect:
repeat
$c = \mathtt{Enc}(ek_A, M)$
until $c \notin generated\_ciphers$
$enc\_value_A := c$
$is\_fresh_A := \begin{cases} F & \text{if } c \in generated\_ciphers \\ T & \text{otherwise} \end{cases}$
$generated\_ciphers := generated\_ciphers \cup \{c\}$

Output $ret\_encrypt(A, C)$
Precondition:
$enc\_value_A = C$
Effect:
$is\_fresh_A := \bot$
$enc\_value_A := \bot$

$\mathcal{E}_k^5(\mathbb{A})$

Input $get\_encrypt(A, M)$
Effect:
repeat
$c = \mathtt{Enc}(ek_A, M)$
until $c \notin generated\_ciphers$
$enc\_value_A := c$
$is\_fresh_A := \begin{cases} F & \text{if } c \in generated\_ciphers \\ T & \text{otherwise} \end{cases}$
$is\_not\_used_A := \begin{cases} F & \text{if } c \in used\_ciphers \\ T & \text{otherwise} \end{cases}$
$generated\_ciphers := generated\_ciphers \cup \{c\}$

Output $ret\_encrypt(A, C)$
Precondition:
$enc\_value_A = C$
Effect:
$is\_fresh_A := \bot$
$is\_not\_used_A := \bot$
$enc\_value_A := \bot$

Fig. 6.10. $get\_encrypt(A, M)$ and $ret\_encrypt(A, C)$ of $\mathcal{E}_k^2$ and $\mathcal{E}_k^5$

$s_5.used\_ciphers \cup extractCiphertexts(decrypt\_key, m)$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_5{\restriction}_{s_2}$. Since the only difference between s_5 and s_5' is on the value of *used_ciphers*, and since *used_ciphers* is not a variable that characterizes s_2, we have that $s_2 = s_5'{\restriction}_{s_2}$ and thus $\delta_{s_2} = \delta_{s_5'}{\restriction}_{\delta_{s_2}}$.

Suppose that $a \in A_k^2$, that is, it is an action of $\mathcal{E}_k^2(\mathbb{A})$. Then we must verify that there exists a transition $tr_2 = (s_2, a, \mu_2) \in D_k^2$ such that $tr_2 = tr_5{\restriction}_{tr_2}$. There are eight cases:

- $a = get\_public\_encrypt(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_public\_encrypt(A)$, we can identify two cases: $s_5.ek_A = \perp$ or $s_5.ek_A \neq \perp$. If $s_5.ek_A = \perp$, then for each state s_5' of $\mathcal{E}_k^5$, $\mu_5(s_5') = \rho(e, d)$ and s_5' is identified by the same values of s_5 except for the following values: $s_5'.ek_A = e$, $s_5'.dk_A = d$, and $s_5'.pk_A = e$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_5{\restriction}_{s_2}$. By definition of action $get\_public\_encrypt(A)$, s_2 enables $get\_public\_encrypt(A)$ that leads to the measure μ_2 such that for each state s_2' of $\mathcal{E}_k^2$, $\mu_2(s_2') = \rho(e, d)$ and s_2' is identified by the same values of s_2 except for the following values: $s_2'.ek_A = e$, $s_2'.dk_A = d$, and $s_2'.pk_A = e$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Thus $\mu_2 = \mu_5{\restriction}_{\mu_2}$, and hence $tr_2 = tr_5{\restriction}_{tr_2}$.

  If $s_5.ek_A \neq \perp$, then $\mu_5 = \delta_{s_5'}$ where s_5' is the state of $\mathcal{E}_k^5(\mathbb{A})$ that is identified by the same values of s_5 except for pk_A where $s_5'.pk_A = s_5.ek_A$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_5{\restriction}_{s_2}$. By definition of $get\_public\_encrypt(A)$, s_2 enables $get\_public\_encrypt(A)$ that leads to the measure $\delta_{s_2'}$ where s_2' is the state of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable pk_A where $s_2'.pk_A = s_2.ek_A$. Thus $\delta_{s_2'} = \delta_{s_5'}{\restriction}_{\delta_{s_2'}}$, and hence $tr_2 = tr_5{\restriction}_{tr_2}$.
- $a = ret\_public\_encrypt(A, E)$ for some $A \in \mathbb{A}$ and $E \in$ EKey: by definition of action $ret\_public\_encrypt(A, E)$, it follows that $s_5.pk_A = E$ and that $\mu_5 = \delta_{s_5'}$ where s_5' is the state of $\mathcal{E}_k^5(\mathbb{A})$ that is identified by the same values of s_5 except for pk_A where $s_5'.pk_A = \perp$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_5{\restriction}_{s_2}$. By definition of action $ret\_public\_encrypt(A, E)$, s_2 enables $ret\_public\_encrypt(A, E)$ that leads to the measure $\delta_{s_2'}$ where s_2' is the state of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable pk_A where $s_2'.pk_A = \perp$. Thus $\delta_{s_2'} = \delta_{s_5'}{\restriction}_{\delta_{s_2'}}$, and hence $tr_2 = tr_5{\restriction}_{tr_2}$.

- $a = \mathit{get\_corrupt\_encrypt}(A)$ for some $A \in \mathbb{A}$: by definition of the action $\mathit{get\_corrupt\_encrypt}(A)$, we can identify two cases: $s_5.ek_A = \perp$ or $s_5.ek_A \neq \perp$. If $s_5.ek_A = \perp$, then for each state s'_5 of $\mathcal{E}_k^5$, $\mu_5(s'_5) = \rho(e,d)$ and s'_5 is identified by the same values of s_5 except for the following values: $s'_5.ek_A = e$, $s'_5.dk_A = d$, and $s'_5.ck_A = (e,d)$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\mathtt{KGen}(1^k)$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_5{\restriction}_{s_2}$. By definition of action $\mathit{get\_corrupt\_encrypt}(A)$, s_2 enables $\mathit{get\_corrupt\_encrypt}(A)$ that leads to the measure μ_2 such that for each state s'_2 of $\mathcal{E}_k^2$, $\mu_2(s'_2) = \rho(e,d)$ and s'_2 is identified by the same values of s_2 except for the following values: $s'_2.ek_A = e$, $s'_2.dk_A = d$, and $s'_2.ck_A = (e,d)$ where ρ is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\mathtt{KGen}(1^k)$. Thus $\mu_2 = \mu_5{\restriction}_{\mu_2}$, and hence $tr_2 = tr_5{\restriction}_{tr_2}$.
  If $s_5.ek_A \neq \perp$, then $\mu_5 = \delta_{s'_5}$ where s'_5 is the state of $\mathcal{E}_k^5(\mathbb{A})$ that is identified by the same values of s_5 except for ck_A where $s'_5.ck_A = (s_5.ek_A, s_5.dk_A)$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_5{\restriction}_{s_2}$. By definition of action $\mathit{get\_corrupt\_encrypt}(A)$, s_2 enables $\mathit{get\_corrupt\_encrypt}(A)$ that leads to the measure $\delta_{s'_2}$ where s'_2 is the state of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable ck_A where $s'_2.ck_A = (s_2.ek_A, s_2.dk_A)$. Thus $\delta_{s'_2} = \delta_{s'_5}{\restriction}_{\delta_{s'_2}}$, and hence $tr_2 = tr_5{\restriction}_{tr_2}$.
- $a = \mathit{ret\_corrupt\_encrypt}(A,E,D)$ for some $A \in \mathbb{A}$, $E \in$ EKey, and $D \in$ DKey: by definition of action $\mathit{ret\_corrupt\_encrypt}(A,E,D)$, it follows that $s_5.ck_A = (E,D)$ and that $\mu_5 = \delta_{s'_5}$ where s'_5 is the state of $\mathcal{E}_k^5(\mathbb{A})$ that is identified by the same values of s_5 except for ck_A where $s'_5.ck_A = \perp$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_5{\restriction}_{s_2}$. By definition of action $\mathit{ret\_corrupt\_encrypt}(A,E,D)$, s_2 enables $\mathit{ret\_corrupt\_encrypt}(A,E,D)$ that leads to the measure $\delta_{s'_2}$ where s'_2 is the state of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable ck_A where $s'_2.ck_A = \perp$. Thus $\delta_{s'_2} = \delta_{s'_5}{\restriction}_{\delta_{s'_2}}$, and hence $tr_2 = tr_5{\restriction}_{tr_2}$.
- $a = \mathit{get\_encrypt}(A,M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of the action $\mathit{get\_encrypt}(A,M)$, it follows that for each state s'_5 of $\mathcal{E}_k^5$, $\mu_5(s'_5) = \rho(C)$ and s'_5 is identified by the same values of s_5 except for the following values: $s'_5.\mathit{is\_fresh}_A = F$ if $C \in s_5.\mathit{generated\_ciphers}$, T otherwise, $s'_5.\mathit{enc\_value}_A = C$, and $s'_5.\mathit{generated\_ciphers} = s_5.\mathit{generated\_ciphers} \cup \{C\}$. ρ is the probabil-

ity measure induced over Ciphertext by the probabilistic algorithm $\mathtt{Enc}(s_5.ek_A, M)$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_5{\restriction}_{s_2}$. By definition of action $get\_encrypt(A, M)$, s_2 enables $get\_encrypt(A, M)$ that leads to the measure ρ where for each state s_2' of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable enc_value_A, we have that $\mu(s_2') = \rho(C)$ and $s_2'.enc\_value_A = C$. Thus $\mu_2 = \mu_5{\restriction}_{\mu_2}$, and hence $tr_2 = tr_5{\restriction}_{tr_2}$.

- $a = ret\_encrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of action $ret\_encrypt(A, C)$, it follows that $s_5.enc\_value_A = C$ and that $\mu_5 = \delta_{s_5'}$ where s_5' is the state of $\mathcal{E}_k^5(\mathbb{A})$ that is identified by the same values of s_5 except for enc_value_A that assumes the value $s_5'.enc\_value_A = \bot$ and for is_fresh_A and $is\_not\_used_A$ that are reset to $\bot$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_5{\restriction}_{s_2}$. By definition of action $ret\_encrypt(A, C)$, s_2 enables $ret\_encrypt(A, C)$ that leads to the measure $\delta_{s_2'}$ where s_2' is the state of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable enc_value_A where $s_2'.enc\_value_A = \bot$. Thus $\delta_{s_2'} = \delta_{s_5'}{\restriction}_{\delta_{s_2'}}$, and hence $tr_2 = tr_5{\restriction}_{tr_2}$.
- $a = get\_decrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of action $get\_decrypt(A, C)$, it follows that $\mu_5 = \delta_{s_5'}$ where s_5' is the state of $\mathcal{E}_k^5(\mathbb{A})$ that is identified by the same values of s_5 except for dec_value_A where $s_5'.dec\_value_A = \mathtt{Dec}(s_5.dk_A, C)$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_5{\restriction}_{s_2}$. By definition of action $get\_decrypt(A, C)$, s_2 enables $get\_decrypt(A, C)$ that leads to the measure $\delta_{s_2'}$ where s_2' is the state of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable dec_value_A where $s_2'.dec\_value_A = \mathtt{Dec}(s_2.dk_A, C)$. Thus $\delta_{s_2'} = \delta_{s_5'}{\restriction}_{\delta_{s_2'}}$, and hence $tr_2 = tr_5{\restriction}_{tr_2}$.
- $a = ret\_decrypt(A, M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of action $ret\_decrypt(A, M)$, it follows that $s_5.dec\_value_A = M$ and that $\mu_5 = \delta_{s_5'}$ where s_5' is the state of $\mathcal{E}_k^5(\mathbb{A})$ that is identified by the same values of s_5 except for dec_value_A where $s_5'.dec\_value_A = \bot$. Let s_2 be the state of $\mathcal{E}_k^2(\mathbb{A})$ such that $s_2 = s_5{\restriction}_{s_2}$. By definition of action $ret\_decrypt(A, M)$, s_2 enables $ret\_decrypt(A, M)$ that leads to the measure $\delta_{s_2'}$ where s_2' is the state of $\mathcal{E}_k^2(\mathbb{A})$ that is identified by the same values of s_2 except for variable dec_value_A where $s_2'.dec\_value_A = \bot$. Thus $\delta_{s_2'} = \delta_{s_5'}{\restriction}_{\delta_{s_2'}}$, and hence $tr_2 = tr_5{\restriction}_{tr_2}$.

Since the requirements of Definition 4.5 are satisfied, for each $k \in \mathbb{N}$, $\mathcal{E}_k^5(\mathbb{A}) \in \mathrm{Ext}_B^W(\mathcal{E}_k^2(\mathbb{A}))$. □

The existence of a polynomially accurate simulation from $\mathcal{E}_k^2(\mathbb{A})$ to $\mathcal{E}_k^5(\mathbb{A})$ in now straightforward:

Proposition 6.22. *Let $\mathbb{A}$ be a set of (identities of) agents. For each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k^2(\mathbb{A})$ such that $\{send(m) \mid m \in \mathrm{Message}\} \subseteq A_{\mathcal{C}_k}$,*

$$\{\mathcal{E}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^5(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.21, for each $k \in \mathbb{N}$, $\mathcal{E}_k^5(\mathbb{A}) \in \mathrm{Ext}_B^W(\mathcal{E}_k^2(\mathbb{A}))$ where B is the set of actions $\{send(m) \mid m \in \mathrm{Message}\}$ and W is $\{used\_ciphers\} \cup \{is\_not\_used_A \mid A \in \mathbb{A}\}$. This implies, by Lemma 4.6, that for each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k^2(\mathbb{A})$ such that $B \subseteq A_{\mathcal{C}_k}$, $\mathcal{E}_k^2(\mathbb{A})||\mathcal{C}_k \preceq \mathcal{E}_k^5(\mathbb{A})||\mathcal{C}_k$ and thus, by Proposition 5.6, $\{\mathcal{E}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^5(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

For the second step, let $\mathcal{E}_k^6$ be the automaton obtained from $\mathcal{E}_k^5$ modifying each $get\_encrypt(A, M)$ action as follows: the encryption algorithm `Enc` is invoked until it returns a value that is a ciphertext that is not yet used, that is not inside *used_ciphers*, as depicted in Figure 6.11.

It is immediate to see that $\mathcal{E}_k^6(\mathbb{A})$ simulates the G_k-conditional of $\mathcal{E}_k^5(\mathbb{A})$ where for each $k \in \mathbb{N}$, G_k is the set of states of $\mathcal{E}_k^5(\mathbb{A})$ such that for each $A \in \mathbb{A}$, $s.is\_not\_used_A \neq F$. In fact, we iterate the generation of the encryption of M under the public key of A until it returns a value that is not equal to an already used ciphertext. That is, we condition the choice of the ciphertext to the fact that it is fresh (and thus, $s.is\_not\_used_A \neq F$ is always satisfied).

Note that $\mathcal{E}_k^6(\mathbb{A})$ is not the G_k-conditional of $\mathcal{E}_k^5(\mathbb{A})$ since it provides more transitions than $\mathcal{E}_k^5(\mathbb{A})|G_k$. In fact, $\mathcal{E}_k^6(\mathbb{A})$ enables transitions that leave from a state not in G_k and that leads to a probability measures μ such that $\mu(G_k) = 0$. For example, let s be the state such that $s.is\_not\_used_A = F$, $s.is\_not\_used_B = F$, $s.enc\_value_A = C$. Thus, by definition of *ret_encrypt* actions, s enables the action $ret\_encrypt(A, C)$ that leads to the measure $\delta_{s'}$ where in s' only enc_value_A and $is\_not\_used_A$ are different with respect to the values in s. Thus $s'.is\_not\_used_B$ is still F and hence $s' \notin G_k$.

The above intuition is formalized by the following result:

Lemma 6.23. *Given $\mathcal{E}_k^5(\mathbb{A})$, let B_k be the set of states of $\mathcal{E}_k^5$ such that $s \in B_k$ if there exists $A \in \mathbb{A}$ such that $s.is\_not\_used_A = F$. Let G_k be the set $S_k^5 \setminus B_k$.*

For each $k \in \mathbb{N}$, $\mathcal{E}_k^5(\mathbb{A})|G_k \preceq \mathcal{E}_k^6(\mathbb{A})$.

$\mathcal{E}_k^5$

Input $get\_encrypt(A, M)$
Effect:
repeat
$c = \mathtt{Enc}(ek_A, M)$
until $c \notin generated\_ciphers$
$enc\_value_A := c$
$$is\_fresh_A := \begin{cases} F & \text{if } c \in generated\_ciphers \\ T & \text{otherwise} \end{cases}$$
$$is\_not\_used_A := \begin{cases} F & \text{if } c \in used\_ciphers \\ T & \text{otherwise} \end{cases}$$
$generated\_ciphers := generated\_ciphers \cup \{c\}$

$\mathcal{E}_k^6$

Input $get\_encrypt(A, M)$
Effect:
repeat
repeat
$c = \mathtt{Enc}(ek_A, M)$
until $c \notin generated\_ciphers$
until $c \notin used\_ciphers$
$enc\_value_A := c$
$$is\_fresh_A := \begin{cases} F & \text{if } c \in generated\_ciphers \\ T & \text{otherwise} \end{cases}$$
$$is\_not\_used_A := \begin{cases} F & \text{if } c \in used\_ciphers \\ T & \text{otherwise} \end{cases}$$
$generated\_ciphers := generated\_ciphers \cup \{c\}$

Fig. 6.11. *get_encrypt* of $\mathcal{E}_k^5$ and $\mathcal{E}_k^6$

Proof. For each $k \in \mathbb{N}$, let id_k be the identity relation on states. id_k is a simulation from $\mathcal{E}_k^5(\mathbb{A})|G_k$ to $\mathcal{E}_k^6(\mathbb{A})$.

The condition on start states is trivially true: let $\bar{s}_k^5$ be the start state of $\mathcal{E}_k^5(\mathbb{A})|G_k$ and $\bar{s}_k^6$ be the start state of $\mathcal{E}_k^6(\mathbb{A})$. By definition of conditional, it follows that $\bar{s}_k^5$ is the start state of $\mathcal{E}_k^5(\mathbb{A})$. Since by definition of $\mathcal{E}_k^6(\mathbb{A})$, the only difference between $\mathcal{E}_k^5(\mathbb{A})$ and $\mathcal{E}_k^6(\mathbb{A})$ is on the definition of the action $get\_encrypt(A, M)$, we have that $\bar{s}_k^6 = \bar{s}_k^5$ and thus $\bar{s}_k^5 \; id_k \; \bar{s}_k^6$.

For the step condition, let s_5 and s_6 be two states of $\mathcal{E}_k^5(\mathbb{A})|G_k$ and $\mathcal{E}_k^6(\mathbb{A})$, respectively, such that $s_5 \; id_k \; s_6$. Let (s_5, a, μ_5) be a transition of

$\mathcal{E}_k^5(\mathbb{A})|G_k$ that leaves from s_5. We must find μ_6 such that (s_6, a, μ_6) is a transition of $\mathcal{E}_k^6(\mathbb{A})$ and $\mu_5\ \mathcal{L}(id_k)\ \mu_6$. There are nine cases:

- $a = get\_public\_encrypt(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_public\_encrypt(A)$, we can identify two cases: $s_5.ek_A = \perp$ or $s_5.ek_A \neq \perp$. If $s_5.ek_A = \perp$, then for each state s'_5 of $\mathcal{E}_k^5$, $\mu_5(s'_5) = \rho_5(e, d)$ and s'_5 is identified by the same values of s_5 except for the following values: $s'_5.ek_A = e$, $s'_5.dk_A = d$, and $s'_5.pk_A = e$ where ρ_5 is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Since $s_5\ id_k\ s_6$, we have that also s_6 satisfies $s_6.ek_A = \perp$, and hence for each state s'_6 of $\mathcal{E}_k^6$, $\mu_6(s'_6) = \rho_6(e, d)$ and s'_6 is identified by the same values of s_6 except for the following values: $s'_6.ek_A = e$, $s'_6.dk_A = d$, and $s'_6.pk_A = e$ where ρ_6 is the probability measure induced over EKey $\times$ DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. This implies that $s'_5\ id_k\ s'_6$ for each state s'_5 and s'_6 satisfying above conditions and since $\rho_5 = \rho_6$, we have that $\mu_5\ \mathcal{L}(id_k)\ \mu_6$, as required.
  If $s_5.ek_A \neq \perp$, then $\mu_5 = \delta_{s'_5}$ where s'_5 is the state of $\mathcal{E}_k^5(\mathbb{A})$ that is identified by the same values of s_5 except for pk_A where $s'_5.pk_A = s_5.ek_A$. Since $s_5\ id_k\ s_6$, we have that also s_6 satisfies $s_6.ek_A \neq \perp$, and hence by definition of action $get\_public\_encrypt(A)$, s_6 enables $get\_public\_encrypt(A)$ that leads to the measure $\delta_{s'_6}$ where s'_6 is the state of $\mathcal{E}_k^6(\mathbb{A})$ that is identified by the same values of s_6 except for variable pk_A where $s'_6.pk_A = s_6.ek_A$. This implies that $s'_5\ id_k\ s'_6$ and thus $\delta_{s'_5}\ \mathcal{L}(id_k)\ \delta_{s'_6}$, that is $\mu_5\ \mathcal{L}(id_k)\ \mu_6$, as required.
- $a = ret\_public\_encrypt(A, E)$ for some $A \in \mathbb{A}$ and $E \in$ EKey: by definition of the action $ret\_public\_encrypt(A, E)$, it follows that s_5 satisfies $s_5.pk_A = E$. Moreover, it follows that $\mu_5 = \delta_{s'_5}$ where s'_5 is the state such that $s'_5.pk_A = \perp$, and all other variables that describe s'_5 have the same value of the variables that describe s_5. Since $s_5\ id_k\ s_6$, we have that also s_6 satisfies $s_6.pk_A = E$ and thus it enables the transition $(s_6, ret\_public\_encrypt(A, E), \mu_6)$ where μ_6 is the measure $\delta_{s'_6}$ where s'_6 is the state such that $s'_6.pk_A = \perp$, and all other variables that describe s'_6 have the same value of the variables that describe s_6. This implies that $s'_5\ id_k\ s'_6$ and thus $\delta_{s'_5}\ \mathcal{L}(id_k)\ \delta_{s'_6}$, that is $\mu_5\ \mathcal{L}(id_k)\ \mu_6$, as required.
- $a = get\_corrupt\_encrypt(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_corrupt\_encrypt(A)$, we can identify two cases: $s_5.ek_A = \perp$ or $s_5.ek_A \neq \perp$. If $s_5.ek_A = \perp$, then for each state s'_5 of $\mathcal{E}_k^5$, $\mu_5(s'_5) = \rho_5(e, d)$ and s'_5 is identified by the same values of s_5 except for the following values: $s'_5.ek_A = e$, $s'_5.dk_A = d$, and $s'_5.ck_A = (e, d)$ where ρ_5 is the proba-

bility measure induced over EKey × DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Since $s_5\ id_k\ s_6$, we have that also s_6 satisfies $s_6.ek_A = \perp$, and hence for each state s'_6 of $\mathcal{E}_k^6$, $\mu_6(s'_6) = \rho_6(e,d)$ and s'_6 is identified by the same values of s_6 except for the following values: $s'_6.ek_A = e$, $s'_6.dk_A = d$, and $s'_6.ck_A = (e,d)$ where ρ_6 is the probability measure induced over EKey × DKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. This implies that $s'_5\ id_k\ s'_6$ for each state s'_5 and s'_6 satisfying above conditions and since $\rho_5 = \rho_6$, we have that $\mu_5\ \mathcal{L}(id_k)\ \mu_6$, as required.
If $s_5.ek_A \neq \perp$, then $\mu_5 = \delta_{s'_5}$ where s'_5 is the state of $\mathcal{E}_k^5(\mathbb{A})$ that is identified by the same values of s_5 except for ck_A where $s'_5.ck_A = (s_5.ek_A, s_5.dk_A)$. Since $s_5\ id_k\ s_6$, we have that also s_6 satisfies $s_6.ek_A \neq \perp$, and hence by definition of action *get_corrupt_encrypt*(A), s_6 enables *get_corrupt_encrypt*(A) that leads to the measure $\delta_{s'_6}$ where s'_6 is the state of $\mathcal{E}_k^6(\mathbb{A})$ that is identified by the same values of s_6 except for variable ck_A where $s'_6.ck_A = (s_6.ek_A, s_6.dk_A)$. This implies that $s'_5\ id_k\ s'_6$ and thus $\delta_{s'_5}\ \mathcal{L}(id_k)\ \delta_{s'_6}$, that is $\mu_5\ \mathcal{L}(id_k)\ \mu_6$, as required.

– $a =$ *ret_corrupt_encrypt*(A,E,D) for some $A \in \mathbb{A}$, $E \in$ EKey and $V \in$ VKey: by definition of the action *ret_corrupt_encrypt*(A,E,D), it follows that s_5 satisfies $s_5.ck_A = (E,D)$. Moreover, it follows that $\mu_5 = \delta_{s'_5}$ where s'_5 is the state such that $s'_5.ck_A = \perp$, and all other variables that describe s'_5 have the same value of the variables that describe s_5. Since $s_5\ id_k\ s_6$, we have that also s_6 satisfies $s_6.ck_A = (E,D)$ and thus it enables the transition $(s_6, \textit{ret\_corrupt\_encrypt}(A,E,D), \mu_6)$ where μ_6 is the measure $\delta_{s'_6}$ where s'_6 is the state such that $s'_6.ck_A = \perp$, and all other variables that describe s'_6 have the same value of the variables that describe s_6. This implies that $s'_5\ id_k\ s'_6$ and thus $\delta_{s'_5}\ \mathcal{L}(id_k)\ \delta_{s'_6}$, that is $\mu_5\ \mathcal{L}(id_k)\ \mu_6$, as required.
– $a =$ *get_encrypt*(A,M) for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of the action *get_encrypt*(A,M), it follows that for each state s'_5 of $\mathcal{E}_k^5$, $\mu_5(s'_5) = \rho_5(C)$ and s'_5 is identified by the same values of s_5 except for the following values: $s'_5.is\_not\_used_A = F$ if $C \in s_5.generated\_ciphers$, T otherwise, $s'_5.enc\_value_A = C$, and $s'_5.generated\_ciphers = s_5.generated\_ciphers \cup \{C\}$. ρ_5 is the probability measure induced over Ciphertext by the probabilistic algorithm $\texttt{Enc}(s_5.ek_A, M)$ conditioned to the set of states where for each $A \in \mathbb{A}$ $is\_not\_used_A \neq F$ conditioned to G_k. Since $s_5\ id_k\ s_6$, we have that also s_6 enables the transition $(s_6, \textit{get\_encrypt}(A,M), \mu_6)$ where for each state s'_6 of $\mathcal{E}_k^6$, $\mu_6(s'_6) = \rho_6(C)$ and s'_6 is identified by the

same values of s_6 except for the following values: $s'_6.is\_not\_used_A = F$ if $C \in s_6.generated\_ciphers$, T otherwise, $s'_6.enc\_value_A = C$, and $s'_6.generated\_ciphers = s_6.generated\_ciphers \cup \{C\}$. ρ_6 is the probability measure induced over Ciphertext by the probabilistic algorithm $\texttt{Enc}(s_6.ek_A, M)$ conditioned to the set of states where for each $A \in \mathbb{A}$ $is\_not\_used_A \neq F$ that is iterated until it returns a value that does not belong to $s_6.used\_ciphers$. This implies that $s'_5\ id_k\ s'_6$ for each state s'_5 and s'_6 satisfying above conditions and if $\rho_5 = \rho_6$, then $\mu_5\ \mathcal{L}(id_k)\ \mu_6$, as required.

ρ_5 is equal to ρ_6 since for each $s \in G_k$, $\rho_5(s) = \rho_6(s)$. In fact, by definition of conditional, $\rho_5(s) = \rho(s)/\rho(G_k)$ where ρ is the probability measure induced over Ciphertext by $\texttt{Enc}(s_5.ek_A, M)$, $\rho(G_k) > 0$ and $\rho(B_k) < 1$. By definition of $get\_encrypt(A, M)$ action of $\mathcal{E}^6_k$, we have that $\rho_6(s) = \sum_{i=0}^{+\infty} \rho(B_k)^i \rho(s) = \rho(s) \sum_{i=0}^{+\infty} \rho(B_k)^i = \rho(s) \dfrac{1}{1-\rho(B_k)} = \rho(s)/\rho(G_k) = \rho_5(s)$.

- $a = ret\_encrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of the action $ret\_encrypt(A, C)$, it follows that s_5 satisfies $s_5.enc\_value_A = C$. Moreover, it follows that $\mu_5 = \delta_{s'_5}$ where s'_5 is the state such that $s'_5.enc\_value_A = \perp$, $s'_5.is\_not\_used_A = \perp$, $s'_5.is\_fresh_A = \perp$, and all other variables that describe s'_5 have the same value of the variables that describe s_5. Since $s_5\ id_k\ s_6$, we have that also s_6 satisfies $s_6.enc\_value_A = C$ and thus it enables the transition $(s_6, ret\_encrypt(A, C), \mu_6)$ where μ_6 is the measure $\delta_{s'_6}$ where s'_6 is the state such that $s'_6.enc\_value_A = \perp$, $s'_6.is\_not\_used_A = \perp$, $s'_6.is\_fresh_A = \perp$ and all other variables that describe s'_6 have the same value of the variables that describe s_6. This implies that $s'_5\ id_k\ s'_6$ and thus $\delta_{s'_5}\ \mathcal{L}(id_k)\ \delta_{s'_6}$, that is $\mu_5\ \mathcal{L}(id_k)\ \mu_6$, as required.
- $a = get\_decrypt(A, C)$ for some $A \in \mathbb{A}$ and $C \in$ Ciphertext: by definition of the action $get\_decrypt(A, C)$, it follows that $\mu_5 = \delta_{s'_5}$ where s'_5 is the state such that $s'_5.dec\_value_A = \texttt{Dec}(s_5.dk_A, C)$, and all other variables that describe s'_5 have the same value of the variables that describe s_5. Since $s_5\ id_k\ s_6$, we have that also s_6 enables the transition $(s_6, get\_decrypt(A, C), \mu_6)$ where μ_6 is the measure $\delta_{s'_6}$ where s'_6 is the state such that $s'_6.dec\_value_A = \texttt{Dec}(s_6.dk_A, C)$, and all other variables that describe s'_6 have the same value of the variables that describe s_6. This implies that $s'_5\ id_k\ s'_6$ and thus $\delta_{s'_5}\ \mathcal{L}(id_k)\ \delta_{s'_6}$, that is $\mu_5\ \mathcal{L}(id_k)\ \mu_6$, as required.

- $a = \mathit{ret\_decrypt}(A, M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of the action $\mathit{ret\_decrypt}(A, M)$, it follows that s_5 satisfies $s_5.\mathit{dec\_value}_A = M$. Moreover, it follows that $\mu_5 = \delta_{s'_5}$ where s'_5 is the state such that $s'_5.\mathit{dec\_value}_A = \bot$, and all other variables that describe s'_5 have the same value of the variables that describe s_5. Since $s_5\ \mathit{id}_k\ s_6$, we have that also s_6 satisfies $s_6.\mathit{dec\_value}_A = M$ and thus it enables the transition $(s_6, \mathit{ret\_decrypt}(A, M), \mu_6)$ where μ_6 is the measure $\delta_{s'_6}$ where s'_6 is the state such that $s'_6.\mathit{dec\_value}_A = \bot$, and all other variables that describe s'_6 have the same value of the variables that describe s_6. This implies that $s'_5\ \mathit{id}_k\ s'_6$ and thus $\delta_{s'_5}\ \mathcal{L}(\mathit{id}_k)\ \delta_{s'_6}$, that is $\mu_5\ \mathcal{L}(\mathit{id}_k)\ \mu_6$, as required.
- $a = \mathit{send}(m)$ for some $m \in$ Message: by definition of $\mathit{send}(m)$, it follows that $\mu_5 = \delta_{s'_5}$ where s'_5 is the state such that $s'_5.\mathit{used\_ciphers}$ is equal to $s_5.\mathit{used\_ciphers} \cup \mathit{extractCiphertexts}(\mathit{decrypt\_key}, m)$, and all other variables that describe s'_5 have the same value of the variables that describe s_5. Since $s_5\ \mathit{id}_k\ s_6$, we have that also s_6 enables the transition $(s_6, \mathit{send}(m), \mu_6)$ where μ_6 is the measure $\delta_{s'_6}$ where s'_6 is the state such that $s'_6.\mathit{used\_ciphers} = s_6.\mathit{used\_ciphers} \cup \mathit{extractCiphertexts}(\mathit{decrypt\_key}, m)$, and all other variables that describe s'_6 have the same value of the variables that describe s_6. This implies that $s'_5\ \mathit{id}_k\ s'_6$ and thus $\delta_{s'_5}\ \mathcal{L}(\mathit{id}_k)\ \delta_{s'_6}$, that is $\mu_5\ \mathcal{L}(\mathit{id}_k)\ \mu_6$, as required.

Since for each action a, if s_5 enables a transition labelled by a that leads to μ_5, then we can find μ_6 such that $(s_6, a, \mu_6) \in D_6$ and $\mu_5\ \mathcal{L}(\mathit{id}_k)\ \mu_6$, the step condition is satisfied. □

The extension of the above result to the polynomially accurate simulation is straightforward:

Proposition 6.24. *Given $\mathcal{E}^5_k(\mathbb{A})$, let B_k be the set of states of $\mathcal{E}^5_k$ such that $s \in B_k$ if there exists $A \in \mathbb{A}$ such that $s.\mathit{is\_not\_used}_A = F$. Let G_k be the set $S^5_k \setminus B_k$.*

For each context $\mathcal{C}_k$ compatible with $\mathcal{E}^5_k(\mathbb{A})|G_k$,

$$\{(\mathcal{E}^5_k(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}^6_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.23, it follows that for each $k \in \mathbb{N}$, $\mathcal{E}^5_k(\mathbb{A})|G_k \preceq \mathcal{E}^6_k(\mathbb{A})$. By compositionality of $\preceq$, we have that for each $k \in \mathbb{N}$ and each context $\mathcal{C}_k$ compatible with $\mathcal{E}^5_k(\mathbb{A})|G_k$, $(\mathcal{E}^5_k(\mathbb{A})|G_k)||\mathcal{C}_k \preceq \mathcal{E}^6_k(\mathbb{A})||\mathcal{C}_k$. By Proposition 5.6, it follows that for each context $\mathcal{C}_k$ compatible with $\mathcal{E}^5_k(\mathbb{A})|G_k$, $\{(\mathcal{E}^5_k(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}^6_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

To complete the chain of simulations from $\mathcal{E}_k^5(\mathbb{A})$ to $\mathcal{E}_k^6(\mathbb{A})$, we need to prove that

Proposition 6.25. *Let* E *be an IND-CCA encryption scheme and* $\mathbb{A}$ *be a set of agents. Let* G_k *be the set of states of* $\mathcal{E}_k^5$ *such that* $s \in G_k$ *if and only if for each* $A \in \mathbb{A}$ $s.is\_not\_used_A \neq F$. *Let* B_k *be* $S_k^5 \setminus G_k$.

For each context $\mathcal{C}_k$ *compatible with* $\mathcal{E}_k^6(\mathbb{A})$, *if there exists* $q \in Poly$ *such that for each action* $send(m)$, $\text{size}(m) < q(k)$, *then*

$$\{\mathcal{E}_k^5(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{E}_k^5(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. Theorem 5.8 states that $\{\mathcal{E}_k^5(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^5(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$ if and only if $\{B_k\}_{k\in\mathbb{N}}$ is negligible in $\{\mathcal{E}_k^5(\mathbb{A})\}_{k\in\mathbb{N}}$.

Suppose, for the sake of contradiction, that $\{\mathcal{E}_k^5(\mathbb{A})\}_{k\in\mathbb{N}}$ is not simulated by $\{(\mathcal{E}_k^5(\mathbb{A})|G_k)\}_{k\in\mathbb{N}}$. This implies that $\{B_k\}_{k\in\mathbb{N}}$ is not negligible in $\{\mathcal{E}_k^5(\mathbb{A})\}_{k\in\mathbb{N}}$ and thus that there exists $c \in \mathbb{N}$, $p \in Poly$ such that for each $\bar{k} \in \mathbb{N}$ there exists $k > \bar{k}$ such the probability to reach states of B_k within $p(k)$ steps is at least k^{-c}, that is, the probability to reach states s such that $s.is\_not\_used_A = F$ for some $A \in \mathbb{A}$ within $p(k)$ steps is at least k^{-c}. By definition of the automaton $\mathcal{E}_k^5(\mathbb{A})$, it follows that $s.is\_not\_used_A$ can assume value F only as the effect of a transition that leaves from some state s' and that is labelled by action $get\_encrypt(A, M)$. This happens only when the ciphertext value $\mathtt{Enc}(s'.ek_A, M)$ assigned to $s.enc\_value_A$ belongs to the set $UC = s'.used\_ciphers$. Within $p(k)$ steps, we can perform at most $p(k)$ transitions labelled by $send(m)$. Since by hypothesis we add at most $q(k)$ values to $used\_ciphers$ each time we perform a transition labelled by $send(m)$, within $p(k)$ steps, the set UC has cardinality at most $p(k)q(k)$. This means that with probability at least k^{-c}, we have generated a ciphertext that it is equal to some $c \in UC$ with $|UC| \leqslant p(k)q(k)$. Since the encryption scheme E is IND-CCA, this contradicts the Proposition 2.4 and thus $\{\mathcal{E}_k^5(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^5(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$. This implies, by Theorem 5.10, that $\{\mathcal{E}_k^5(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{E}_k^5(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

6.3 Signature

The third primitive we consider is digital signature. We model a signature scheme $\mathsf{S} = (\mathtt{KGen}, \mathtt{Sig}, \mathtt{Ver})$ with an automaton that provides several families of actions: a family of input actions $get\_public\_sign(A)$ that is used by other automata to ask for the public key associated to agent A and the corresponding $ret\_public\_sign(A, V)$ that returns the public key V of A:

Input $get\_public\_sign(A)$
Effect:
if $sk_A = \bot$ then
$(sk_A, vk_A) := \texttt{KGen}(1^k)$
fi
$pk_A := vk_A$

Output $ret\_public\_sign(A, V)$
Precondition:
$pk_A = V$
Effect:
$pk_A := \bot$

a family of input actions $get\_corrupt\_sign(A)$ that is used by other automata to ask both public and private key of a corrupted agent A and the corresponding $ret\_corrupt\_sign(A, V, S)$ that returns both public and private signion keys V and S of A, respectively:

Input $get\_corrupt\_sign(A)$
Effect:
if $sk_A = \bot$ then
$(sk_A, vk_A) := \texttt{KGen}(1^k)$
fi
$ck_A := (vk_A, sk_A)$

Output $ret\_corrupt\_sign(A, V, S)$
Precondition:
$ck_A = (V, S)$
Effect:
$ck_A := \bot$

Finally, the automaton provides actions to sign and verify messages: a family of input actions $get\_sign(A, M)$ that is used require the signature of M under the private key of A and the corresponding output action $ret\_sign(A, S)$ that returns the signature S obtained invoking the signing algorithm `Sig`:

Input $get\_sign(A, M)$
Effect:
$sig\_value_A := \texttt{Sig}(sk_A, M)$

Output $ret\_sign(A, S)$
Precondition:
$sig\_value_A = S$
Effect:
$sig\_value_A := \bot$

and a family of input actions $get\_verify\_sign(A, S)$ that is used require the verification of S under the public key of A and the corresponding output action $ret\_verify\_sign(A, B)$ that returns the validity B of S obtained invoking the verification algorithm `Ver`:

Input $get\_verify\_sign(A, S)$
Effect:
$ver\_value_A := \texttt{Ver}(vk_A, S)$

Output $ret\_verify\_sign(A, B)$
Precondition:
$ver\_value_A = B$
Effect:
$ver\_value_A := \bot$

Figure 6.12 depicts the complete signature automaton.

6.3.1 Automaton and Properties of Signatures

Unlike in the previous section, it is not so easy to define other automata that ensures by construction the properties of the digital signature. For example, an IND-CCA encryption scheme imposes to use randomized algorithms, otherwise the attacker can distinguish ciphertexts in the following way: it generates two messages m_0 and m_1 and then it asks the encryption algorithm to encrypt them. Let c_b be the resulting ciphertext, where b is the bit that parameterizes the left-right encryption oracle. Then the attacker computes the ciphertexts c_0 and c_1 of m_0 and m_1, respectively. It can compute them by itself or simply it can ask the left-right encryption oracle the pairs (m_0, m_0) and (m_1, m_1). Finally, the attacker compares c_b with c_0 and c_1 and outputs 1 when $c_b = c_1$, 0 when $c_b = c_0$. Since the encryption scheme is not randomized, each time we require the encryption of m, the value we obtain does not change, so with probability 1 the attacker guesses the bit b.

On the contrary, it is not mandatory to use randomized algorithms to define a signature scheme. In fact, the only property the signature scheme requires is that it is not feasible to generate a valid signature for a message that has not signed yet. Note that we do not impose restrictions about the signature of already signed messages; we do not care if they are repeated or not, because the repetition of signatures does not influence the unforgeability of the signature scheme.

Anyway, suppose that the signature scheme is randomized and that ensures that the probability to generate repeated signatures is negligible. Formally,

Definition 6.26. *Let* $\mathsf{S} = (G, S, V)$ *be a signature scheme.*

We say that S *is a* non-repeating *unforgeable signature scheme if the following conditions hold:*

- S *is an unforgeable signature scheme, and*
- *for each* $c \in \mathbb{N}$ *and* $p \in Poly$*, there exists* $\bar{k} \in \mathbb{N}$ *such that for each* $k > \bar{k}$*, we have that for each* (s, v) *in the range of* $G(1^k)$ *and for every* $\alpha \in \{0,1\}^*$*,* $\boldsymbol{Pr}[\beta_i = \beta_j \mid 1 \leqslant i, j \leqslant p(k), i \neq j] < k^{-c}$ *where for each* $1 \leqslant i \leqslant p(k)$*,* $\beta_i = S(s, \alpha)$*.*

A possible way to define an automaton that models a non-repeating unforgeable signature scheme and that ensures that generated signatures

Signature automaton $\mathcal{S}_k(\mathbb{A})$

Signature:

Input:

$get\_public\_sign(A)$, $A \in \mathbb{A}$
$get\_corrupt\_sign(A)$, $A \in \mathbb{A}$
$get\_sign(A, M)$, $A \in \mathbb{A}$, $M \in$ Message
$get\_verify\_sign(A, S)$, $A \in \mathbb{A}$, $S \in$ Signature

Output:

$ret\_public\_sign(A, V)$, $A \in \mathbb{A}$, $V \in$ VKey
$ret\_corrupt\_sign(A, V, S)$, $A \in \mathbb{A}$, $V \in$ VKey, $S \in$ SKey
$ret\_sign(A, S)$, $A \in \mathbb{A}$, $S \in$ Signature
$ret\_verify\_sign(A, B)$, $A \in \mathbb{A}$, $B \in \{T, F\}$

State:

sk_A $\in \{\bot\} \cup$ SKey, $A \in \mathbb{A}$, initially $\bot$
vk_A $\in \{\bot\} \cup$ VKey, $A \in \mathbb{A}$, initially $\bot$
pk_A $\in \{\bot\} \cup$ VKey, $A \in \mathbb{A}$, initially $\bot$
ck_A $\in \{\bot\} \cup$ VKey $\times$ SKey, $A \in \mathbb{A}$, initially $\bot$
sig_value_A $\in \{\bot\} \cup$ Signature, $A \in \mathbb{A}$, initially $\bot$
ver_value_A $\in \{\bot\} \cup \{T, F\}$, $A \in \mathbb{A}$, initially $\bot$

Transitions:

Input $get\_public\_sign(A)$
Effect:
if $sk_A = \bot$ then
$(sk_A, vk_A) := \mathtt{KGen}(1^k)$
fi
$pk_A := vk_A$

Input $get\_corrupt\_sign(A)$
Effect:
if $sk_A = \bot$ then
$(sk_A, vk_A) := \mathtt{KGen}(1^k)$
fi
$ck_A := (vk_A, sk_A)$

Input $get\_sign(A, M)$
Effect:
$sig\_value_A := \mathtt{Sig}(sk_A, M)$

Input $get\_verify\_sign(A, S)$
Effect:
$ver\_value_A := \mathtt{Ver}(vk_A, S)$

Output $ret\_public\_sign(A, V)$
Precondition:
$pk_A = V$
Effect:
$pk_A := \bot$

Output $ret\_corrupt\_sign(A, V, S)$
Precondition:
$ck_A = (V, S)$
Effect:
$ck_A := \bot$

Output $ret\_sign(A, S)$
Precondition:
$sig\_value_A = S$
Effect:
$sig\_value_A := \bot$

Output $ret\_verify\_sign(A, B)$
Precondition:
$ver\_value_A = B$
Effect:
$ver\_value_A := \bot$

Fig. 6.12. Signature automaton $\mathcal{S}_k(\mathbb{A})$

$\mathcal{S}_k(\mathbb{A})$

Input $get\_sign(A, M)$
Effect:
$sig\_value_A := \mathtt{Sig}(sk_A, M)$

Output $ret\_sign(A, S)$
Precondition:
$sig\_value_A = S$
Effect:
$sig\_value_A := \perp$

$\mathcal{S}_k^1(\mathbb{A})$

Input $get\_sign(A, M)$
Effect:
$sig\_value_A := s$ where $s = \mathtt{Sig}(sk_A, M)$
$is\_fresh_A := \begin{cases} F & \text{if } s \in generated\_signs \\ T & \text{otherwise} \end{cases}$
$generated\_signs := generated\_signs \cup \{s\}$

Output $ret\_sign(A, S)$
Precondition:
$sig\_value_A = S$
Effect:
$sig\_value_A := \perp$
$is\_fresh_A := \perp$

Fig. 6.13. $get\_sign(A, M)$ and $ret\_sign(A, S)$ of $\mathcal{S}_k$ and $\mathcal{S}_k^1$

are always fresh is to add some history variable that keeps all previous generated values and the information about the freshness of the generated signature, as we have already done for nonces and ciphertexts. This result can be achieved modifying the $get\_sign(A, M)$ action imposing that the signature s returned by the `Sig` algorithm is not an old signature and then adding s to the set of already generated signatures. We consider also some variable that keeps information about the freshness of the last computed signature to simplify the identification of the states where a repeated signature is generated.

For the first step, let $\mathcal{S}_k^1$ be the automaton obtained from $\mathcal{S}_k$ adding the state variable $generated\_signs \subseteq$ Signature with initial value $\emptyset$ and the family of state variables $is\_fresh_A \in \{T, F, \perp\}$ with initial value $\perp$ where $A \in \mathbb{A}$. Moreover, each $get\_sign(A, M)$ action modifies is_fresh_A assigning

F if the signature s returned by $\texttt{Sig}(sk_A, M)$ belongs to *generated_signs*; T otherwise and also updates *generated_signs* adding s to *generated_signs*, as depicted in Figure 6.13. The action $ret\_sign(A, S)$ reset the is_fresh_A variable to its initial value $\perp$.

It is immediate to see that $\mathcal{S}_k^1(\mathbb{A})$ is an extension of $\mathcal{S}_k(\mathbb{A})$. In fact, we simply add some history variables that keep information about the generated signatures and the freshness of them. Such variables are updated internally and do not affect the behavior of the automaton. Moreover, they are updated in a "deterministic" way, that is, when $\mathcal{S}_k(\mathbb{A})$ reaches t' from t with probability p, then also $\mathcal{S}_k^1(\mathbb{A})$ reaches t_1' from t_1 with probability p where $t = t_1\lceil_t$ and $t' = t_1'\lceil_{t'}$. In fact, when $\mathcal{S}_k(\mathbb{A})$ performs a transition labelled by $get\_sign(A, M)$, it actually performs a transition $tr = (t, get\_sign(A, M), \mu)$ where for each $t' \in Supp(\mu)$, $t'.sig\_value_A = s$ where s is the value returned by $\texttt{Sig}(t.sk_A, M)$ and $\mu(t')$ is the probability that $\texttt{Sig}(t.sk_A, M)$ generates s. $\mathcal{S}_k^1(\mathbb{A})$ also performs a transition $tr_1 = (t_1, get\_sign(A, M), \mu_1)$ for each $t_1' \in Supp(\mu_1)$, $t_1'.sig\_value_A = s$ where s is the value returned by $\texttt{Sig}(t_1.sk_A, M)$ and $\mu(t')$ is the probability that $\texttt{Sig}(t_1.sk_A, M)$ generates s. That is, $tr = tr_1\lceil_{tr}$.

The above intuition is formalized by the following result:

Lemma 6.27. *Let $\mathbb{A}$ be a set of (identities of) agents and W be the set of variables $\{generated\_signs\} \cup \{is\_fresh_A \mid A \in \mathbb{A}\}$ defined as above. For each $k \in \mathbb{N}$, $\mathcal{S}_k^1(\mathbb{A}) \in \mathrm{Ext}_\emptyset^W(\mathcal{S}_k(\mathbb{A}))$.*

Proof. To prove the statement of the Lemma, we need to check if the requirements of Definition 4.5 are satisfied:

compatible states: let v be a state variable of $\mathcal{S}_k^1(\mathbb{A})$. Then v is either one of sk_A, vk_A, pk_A, ck_A, sig_value_A, and ver_value_A for $A \in \mathbb{A}$ and thus it is a state variable of $\mathcal{S}_k(\mathbb{A})$, or v is either *generated_signs* or is_fresh_A for $A \in \mathbb{A}$, and thus $v \in W$;

compatible start state: $\bar{s}_k^1$ is identified by value $\perp$ for each sk_A, vk_A, pk_A, ck_A, sig_value_A, ver_value_A, and is_fresh_A (with $A \in \mathbb{A}$), and by $\emptyset$ for *generated_signs*. Since $\bar{s}_k$ is identified by the value $\perp$ for each sk_A, vk_A, pk_A, ck_A, sig_value_A, and ver_value_A (with $A \in \mathbb{A}$), then $\bar{s}_k = \bar{s}_k^1\lceil_{\bar{s}_k}$;

compatible actions: $\mathcal{S}_k^1(\mathbb{A})$ and $\mathcal{S}_k(\mathbb{A})$ provides the same set of actions, so this condition is trivially verified; and

compatible transitions: let $tr_1 = (t_1, a, \mu_1) \in D_k^1$ be a transition of $\mathcal{S}_k^1$. Since A of $\mathcal{S}_k^1$ is equal to A of $\mathcal{S}_k(\mathbb{A})$ by definition of $\mathcal{S}_k^1(\mathbb{A})$, then we

must verify that there exists a transition $tr = (t, a, \mu) \in D_k$ such that $tr = tr_1 \restriction_{tr}$. There are eight cases:

- $a = get\_public\_sign(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_public\_sign(A)$, we can identify two cases: $t_1.sk_A = \perp$ or $t_1.sk_A \neq \perp$. If $t_1.sk_A = \perp$, then for each state t_1' of $\mathcal{S}_k^1$, $\mu_1(t_1') = \rho(s, v)$ and t_1' is identified by the same values of t_1 except for the following values: $t_1'.sk_A = s$, $t_1'.vk_A = v$, and $t_1'.pk_A = v$ where ρ is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Let t be the state of $\mathcal{S}_k(\mathbb{A})$ such that $t = t_1 \restriction_t$. By definition of action $get\_public\_sign(A)$, t enables $get\_public\_sign(A)$ that leads to the measure μ such that for each state t' of $\mathcal{S}_k$, $\mu(t') = \rho(s, v)$ and t' is identified by the same values of t except for the following values: $t'.sk_A = s$, $t'.vk_A = v$, and $t'.pk_A = v$ where ρ is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Thus $\mu = \mu_1 \restriction_\mu$, and hence $tr = tr_1 \restriction_{tr}$.
  If $t_1.sk_A \neq \perp$, then $\mu_1 = \delta_{t_1'}$ where t_1' is the state of $\mathcal{S}_k^1(\mathbb{A})$ that is identified by the same values of t_1 except for the value of the variable pk_A where $t_1'.pk_A = t_1.vk_A$. Let t be the state of $\mathcal{S}_k(\mathbb{A})$ such that $t = t_1 \restriction_t$. By definition of action $get\_public\_sign(A)$, t enables $get\_public\_sign(A)$ that leads to the measure $\delta_{t'}$ where t' is the state of $\mathcal{S}_k(\mathbb{A})$ that is identified by the same values of t except for variable pk_A where $t'.pk_A = t.vk_A$. Thus $\delta_{t'} = \delta_{t_1'} \restriction_{\delta_{t'}}$, and hence $tr = tr_1 \restriction_{tr}$.
- $a = ret\_public\_sign(A, V)$ for some $A \in \mathbb{A}$ and $V \in$ VKey: by definition of action $ret\_public\_sign(A, V)$, it follows that $t_1.sk_A = V$ and that $\mu_1 = \delta_{t_1'}$ where t_1' is the state of $\mathcal{S}_k^1(\mathbb{A})$ that is identified by the same values of t_1 except for the value of the variable pk_A where $t_1'.pk_A = \perp$. Let t be the state of $\mathcal{S}_k(\mathbb{A})$ such that $t = t_1 \restriction_t$. By definition of action $ret\_public\_sign(A, V)$, t enables $ret\_public\_sign(A, V)$ that leads to the measure $\delta_{t'}$ where t' is the state of $\mathcal{S}_k(\mathbb{A})$ that is identified by the same values of t except for variable pk_A where $t'.pk_A = \perp$. Thus $\delta_{t'} = \delta_{t_1'} \restriction_{\delta_{t'}}$, and hence $tr = tr_1 \restriction_{tr}$.
- $a = get\_corrupt\_sign(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_corrupt\_sign(A)$, we can identify two cases: $t_1.sk_A = \perp$ or $t_1.sk_A \neq \perp$. If $t_1.sk_A = \perp$, then for each state t_1' of $\mathcal{S}_k^1$, $\mu_1(t_1') = \rho(s, v)$ and t_1' is identified by the same values of t_1 except for the following values: $t_1'.sk_A = s$, $t_1'.vk_A = v$, and $t_1'.ck_A = (v, s)$ where ρ is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Let t be the state of $\mathcal{S}_k(\mathbb{A})$ such

that $t = t_1\lceil_t$. By definition of the action *get_corrupt_sign*(A), t enables *get_corrupt_sign*(A) that leads to the measure μ such that for each state t' of $\mathcal{S}_k$, $\mu(t') = \rho(s, v)$ and t' is identified by the same values of t except for the following values: $t'.sk_A = s$, $t'.vk_A = v$, and $t'.ck_A = (v, s)$ where ρ is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Thus $\mu = \mu_1\lceil_\mu$, and hence $tr = tr_1\lceil_{tr}$.
If $t_1.sk_A \neq \perp$, then $\mu_1 = \delta_{t'_1}$ where t'_1 is the state of $\mathcal{S}^1_k(\mathbb{A})$ that is identified by the same values of t_1 except for the value of the variable ck_A where $t'_1.ck_A = (t_1.vk_A, t_1.sk_A)$. Let t be the state of $\mathcal{S}_k(\mathbb{A})$ such that $t = t_1\lceil_t$. By definition of the action *get_corrupt_sign*(A), t enables *get_corrupt_sign*(A) that leads to the measure $\delta_{t'}$ where t' is the state of $\mathcal{S}_k(\mathbb{A})$ that is identified by the same values of t except for variable ck_A where $t'.ck_A = (t.vk_A, t.sk_A)$. Thus $\delta_{t'} = \delta_{t'_1}\lceil_{\delta_{t'}}$, and hence $tr = tr_1\lceil_{tr}$.

- a = *ret_corrupt_sign*(A, V, S) for some $A \in \mathbb{A}$, $S \in$ SKey, and $V \in$ VKey: by definition of action *ret_corrupt_sign*(A, V, S), it follows that $s_1.ck_A = (V, S)$ and that $\mu_1 = \delta_{t'_1}$ where t'_1 is the state of $\mathcal{S}^1_k(\mathbb{A})$ that is identified by the same values of t_1 except for ck_A where $t'_1.ck_A = \perp$. Let t be the state of $\mathcal{S}_k(\mathbb{A})$ such that $t = t_1\lceil_t$. By definition of action *ret_corrupt_sign*(A, V, S), t enables *ret_corrupt_sign*(A, V, S) that leads to the measure $\delta_{t'}$ where t' is the state of $\mathcal{S}_k(\mathbb{A})$ that is identified by the same values of t except for variable ck_A where $t'.ck_A = \perp$. Thus $\delta_{t'} = \delta_{t'_1}\lceil_{\delta_{t'}}$, and hence $tr = tr_1\lceil_{tr}$.
- a = *get_sign*(A, M) for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of the action *get_sign*(A, M), it follows that for each state t'_1 of $\mathcal{S}^1_k$, $\mu_1(t'_1) = \rho(S)$ and t'_1 is identified by the same values of t_1 except for the following values that are: $t'_1.is\_fresh_A = F$ if $S \in s_1.generated\_signs$, T otherwise, $t'_1.sig\_value_A = S$, and $t'_1.generated\_signs = t_1.generated\_signs \cup \{S\}$. ρ is the probability measure induced over Signature by the probabilistic algorithm $\texttt{Sig}(t_1.sk_A, M)$. Let t be the state of $\mathcal{S}_k(\mathbb{A})$ such that $t = t_1\lceil_t$. By definition of action *get_sign*(A, M), t enables *get_sign*(A, M) that leads to the measure ρ where for each state t' of $\mathcal{S}_k(\mathbb{A})$ that is identified by the same values of t except for variable sig_value_A, we have that $\mu(t') = \rho(S)$ and $t'.sig\_value_A = S$. Thus $\mu = \mu_1\lceil_\mu$, and hence $tr = tr_1\lceil_{tr}$.

- $a = \mathit{ret\_sign}(A, S)$ for some $A \in \mathbb{A}$ and $S \in$ Signature: by definition of action $\mathit{ret\_sign}(A, S)$, it follows that $t_1.\mathit{sig\_value}_A = S$ and that $\mu_1 = \delta_{t'_1}$ where t'_1 is the state of $\mathcal{S}^1_k(\mathbb{A})$ that is identified by the same values of t_1 except for $\mathit{sig\_value}_A$ that assumes the value $t'_1.\mathit{sig\_value}_A = \bot$ and for $\mathit{is\_fresh}_A$ that is reset to $\bot$. Let t be the state of $\mathcal{S}_k(\mathbb{A})$ such that $t = t_1{\restriction_t}$. By definition of action $\mathit{ret\_sign}(A, S)$, t enables $\mathit{ret\_sign}(A, S)$ that leads to the measure $\delta_{t'}$ where t' is the state of $\mathcal{S}_k(\mathbb{A})$ that is identified by the same values of t except for variable $\mathit{sig\_value}_A$ where $t'.\mathit{sig\_value}_A = \bot$. Thus $\delta_{t'} = \delta_{t'_1}{\restriction_{\delta_{t'}}}$, and hence $tr = tr_1{\restriction_{tr}}$.
- $a = \mathit{get\_verify\_sign}(A, S)$ for some $A \in \mathbb{A}$ and $S \in$ Signature: by definition of action $\mathit{get\_verify\_sign}(A, S)$, it follows that $\mu_1 = \delta_{t'_1}$ where t'_1 is the state of $\mathcal{S}^1_k(\mathbb{A})$ that is identified by the same values of t_1 except for $\mathit{ver\_value}_A$ where $t'_1.\mathit{ver\_value}_A = \mathtt{Ver}(t_1.vk_A, S)$. Let t be the state of $\mathcal{S}_k(\mathbb{A})$ such that $t = t_1{\restriction_t}$. By definition of action $\mathit{get\_verify\_sign}(A, S)$, t enables $\mathit{get\_verify\_sign}(A, S)$ that leads to the measure $\delta_{t'}$ where t' is the state of $\mathcal{S}_k(\mathbb{A})$ that is identified by the same values of t except for variable $\mathit{ver\_value}_A$ where $t'.\mathit{ver\_value}_A = \mathtt{Ver}(t.vk_A, S)$. Thus $\delta_{t'} = \delta_{t'_1}{\restriction_{\delta_{t'}}}$, and hence $tr = tr_1{\restriction_{tr}}$.
- $a = \mathit{ret\_verify\_sign}(A, B)$ for some $A \in \mathbb{A}$ and $B \in \{T, F\}$: by definition of action $\mathit{ret\_verify\_sign}(A, B)$, it follows that $t_1.\mathit{ver\_value}_A = B$ and that $\mu_1 = \delta_{t'_1}$ where t'_1 is the state of $\mathcal{S}^1_k(\mathbb{A})$ that is identified by the same values of t_1 except for $\mathit{ver\_value}_A$ where $t'_1.\mathit{ver\_value}_A = \bot$. Let t be the state of $\mathcal{S}_k(\mathbb{A})$ such that $t = t_1{\restriction_t}$. By definition of action $\mathit{ret\_verify\_sign}(A, B)$, t enables $\mathit{ret\_verify\_sign}(A, B)$ that leads to the measure $\delta_{t'}$ where t' is the state of $\mathcal{S}_k(\mathbb{A})$ that is identified by the same values of t except for variable $\mathit{ver\_value}_A$ where $t'.\mathit{ver\_value}_A = \bot$. Thus $\delta_{t'} = \delta_{t'_1}{\restriction_{\delta_{t'}}}$, and hence $tr = tr_1{\restriction_{tr}}$.

Since the requirements of Definition 4.5 are satisfied, for each $k \in \mathbb{N}$, $\mathcal{S}^1_k(\mathbb{A}) \in \mathrm{Ext}^W_\emptyset(\mathcal{S}_k(\mathbb{A}))$. □

The existence of a polynomially accurate simulation from $\mathcal{S}_k(\mathbb{A})$ to $\mathcal{S}^1_k(\mathbb{A})$ in now straightforward:

Proposition 6.28. *Let $\mathbb{A}$ be a set of (identities of) agents. For each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k(\mathbb{A})$,*

$$\{\mathcal{S}_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}^1_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

$\mathcal{S}_k^1$

Input $get\_sign(A, M)$
Effect:
$sig\_value_A := s$ where $s = \mathtt{Sig}(sk_A, M)$
$is\_fresh_A := \begin{cases} F & \text{if } s \in generated\_signs \\ T & \text{otherwise} \end{cases}$
$generated\_signs := generated\_signs \cup \{s\}$

$\mathcal{S}_k^2$

Input $get\_sign(A, M)$
Effect:
repeat
$s = \mathtt{Sig}(sk_A, M)$
until $s \notin generated\_signs$
$sig\_value_A := s$
$is\_fresh_A := \begin{cases} F & \text{if } s \in generated\_signs \\ T & \text{otherwise} \end{cases}$
$generated\_signs := generated\_signs \cup \{s\}$

Fig. 6.14. *get_sign* of $\mathcal{S}_k^1$ and $\mathcal{S}_k^2$

Proof. By Lemma 6.27, for each $k \in \mathbb{N}$, $\mathcal{S}_k^1(\mathbb{A}) \in \mathrm{Ext}_B^W(\mathcal{S}_k(\mathbb{A}))$ where B is the empty set and W is $\{generated\_signs\} \cup \{is\_fresh_A \mid A \in \mathbb{A}\}$. This implies, by Lemma 4.6, that for each context C'_k compatible with $\mathcal{S}_k(\mathbb{A})$ such that $B \subseteq A_{C'_k}$, $\mathcal{S}_k(\mathbb{A})||C'_k \preceq \mathcal{S}_k^1(\mathbb{A})||C'_k$ and thus, by Proposition 5.6, $\{\mathcal{S}_k(\mathbb{A})||C'_k\}_{k\in\mathbb{N}} \leq_s \{\mathcal{S}_k^1(\mathbb{A})||C'_k\}_{k\in\mathbb{N}}$. Since $B = \emptyset$, each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k(\mathbb{A})$ satisfies $B \subseteq A_{\mathcal{C}_k}$ and thus for each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k(\mathbb{A})$, $\{\mathcal{S}_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \leq_s \{\mathcal{S}_k^1(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

For the second step, let $\mathcal{S}_k^2$ be the automaton obtained from $\mathcal{S}_k^1$ modifying each $get\_sign(A, M)$ action as follows: the encryption algorithm $\mathtt{Sig}$ is invoked until it returns a value that is a fresh signature, that is not inside $generated\_signs$, as depicted in Figure 6.14.

It is immediate to see that $\mathcal{S}_k^2(\mathbb{A})$ simulates the G_k-conditional of $\mathcal{S}_k^1(\mathbb{A})$ where for each $k \in \mathbb{N}$, G_k is the set of states of $\mathcal{S}_k^1(\mathbb{A})$ such that for each $A \in \mathbb{A}$, $\neq .is\_fresh_A F$. In fact, we iterate the generation of the signature of M under the secret key of A until we obtain a value that is not equal to an already generated signature. That is, we condition the choice of the signature to the fact that it is fresh (and thus, $\neq .is\_fresh_A F$ is always satisfied).

Note that $\mathcal{S}_k^2(\mathbb{A})$ is not the G_k-conditional of $\mathcal{S}_k^1(\mathbb{A})$ since it provides more transitions than $\mathcal{S}_k^1(\mathbb{A})|G_k$. In fact, $\mathcal{S}_k^2(\mathbb{A})$ enables transitions that leave from a state not in G_k and that leads to a probability measures μ such that $\mu(G_k) = 0$. For example, let t be the state such that $t.is\_fresh_A = F$, $t.is\_fresh_B = F$, $t.sig\_value_A = S$. Thus, by definition of *ret_sign* actions, t enables the action $ret\_sign(A, S)$ that leads to the measure $\delta_{t'}$ where t' is identified by the same values of t except for sig_value_A and is_fresh_A that are $\perp$. Thus $t'.is\_fresh_B$ is still F and hence $t' \notin G_k$.

The above intuition is formalized by the following result:

Lemma 6.29. *Given $\mathcal{S}_k^1(\mathbb{A})$, let B_k be the set of states of $\mathcal{S}_k^1$ such that $t \in B_k$ if there exists $A \in \mathbb{A}$ such that $t.is\_fresh_A = F$. Let G_k be the set $S_k^1 \setminus B_k$.*

For each $k \in \mathbb{N}$, $\mathcal{S}_k^1(\mathbb{A})|G_k \preceq \mathcal{S}_k^2(\mathbb{A})$.

Proof. For each $k \in \mathbb{N}$, let id_k be the identity relation on states. id_k is a simulation from $\mathcal{S}_k^1(\mathbb{A})|G_k$ to $\mathcal{S}_k^2(\mathbb{A})$.

The condition on start states is trivially true: let $\bar{s}_k^1$ be the start state of $\mathcal{S}_k^1(\mathbb{A})|G_k$ and $\bar{s}_k^2$ be the start state of $\mathcal{S}_k^2(\mathbb{A})$. By definition of conditional, it follows that $\bar{s}_k^1$ is the start state of $\mathcal{S}_k^1(\mathbb{A})$. Since by definition of $\mathcal{S}_k^2(\mathbb{A})$, the only difference between $\mathcal{S}_k^1(\mathbb{A})$ and $\mathcal{S}_k^2(\mathbb{A})$ is on the definition of the action $get\_sign(A, M)$, we have that $\bar{s}_k^2 = \bar{s}_k^1$ and thus $\bar{s}_k^1\ id_k\ \bar{s}_k^2$.

For the step condition, let t_1 and t_2 be two states of $\mathcal{S}_k^1(\mathbb{A})|G_k$ and $\mathcal{S}_k^2(\mathbb{A})$, respectively, such that $t_1\ id_k\ t_2$. Let (t_1, a, μ_1) be a transition of $\mathcal{S}_k^1(\mathbb{A})|G_k$ that leaves from t_1. We must find μ_2 such that (t_2, a, μ_2) is a transition of $\mathcal{S}_k^2(\mathbb{A})$ and $\mu_1\ \mathcal{L}(id_k)\ \mu_2$. There are eight cases:

- $a = get\_public\_sign(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_public\_sign(A)$, we can identify two cases: $t_1.sk_A = \perp$ or $t_1.sk_A \neq \perp$. If $t_1.sk_A = \perp$, then for each state t_1' of $\mathcal{S}_k^1|G_k$, $\mu_1(t_1') = \rho_1(s, v)$ and t_1' is identified by the same values of t_1 except for the following values: $t_1'.sk_A = s$, $t_1'.vk_A = v$, and $t_1'.pk_A = v$ where ρ_1 is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Since $t_1\ id_k\ t_2$, we have that also t_2 satisfies $t_2.sk_A = \perp$, and hence for each state t_2' of $\mathcal{S}_k^2$, $\mu_2(t_2') = \rho_2(s, v)$ and t_2' is identified by the same values of t_2 except for the following values: $t_2'.sk_A = s$, $t_2'.vk_A = v$, and $t_2'.pk_A = v$ where ρ_2 is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. This implies that $t_1'\ id_k\ t_2'$ for each state t_1' and t_2' satisfying above conditions and since $\rho_1 = \rho_2$, we have that $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

If $t_1.sk_A \neq \perp$, then $\mu_1 = \delta_{t'_1}$ where t'_1 is the state of $\mathcal{S}^1_k(\mathbb{A})|G_k$ that is identified by the same values of t_1 except for pk_A where $t'_1.pk_A = t_1.vk_A$. Since $t_1\ id_k\ t_2$, we have that also t_2 satisfies $t_2.sk_A \neq \perp$, and hence by definition of action $get\_public\_sign(A)$, t_2 enables $get\_public\_sign(A)$ that leads to the measure $\delta_{t'_2}$ where t'_2 is the state of $\mathcal{S}^2_k(\mathbb{A})$ that is identified by the same values of t_2 except for variable pk_A where $t'_2.pk_A = t_2.vk_A$. This implies that $t'_1\ id_k\ t'_2$ and thus $\delta_{t'_1}\ \mathcal{L}(id_k)\ \delta_{t'_2}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

– $a = ret\_public\_sign(A, V)$ for some $A \in \mathbb{A}$ and $V \in$ VKey: by definition of the action $ret\_public\_sign(A, V)$, it follows that t_1 satisfies $t_1.pk_A = V$. Moreover, it follows that $\mu_1 = \delta_{t'_1}$ where t'_1 is the state such that $t'_1.pk_A = \perp$, and all other variables that describe t'_1 have the same value of the variables that describe t_1. Since $t_1\ id_k\ t_2$, we have that also t_2 satisfies $t_2.pk_A = V$ and thus it enables the transition $(t_2, ret\_public\_sign(A, V), \mu_2)$ where μ_2 is the measure $\delta_{t'_2}$ where t'_2 is the state such that $t'_2.pk_A = \perp$, and all other variables that describe t'_2 have the same value of the variables that describe t_2. This implies that $t'_1\ id_k\ t'_2$ and thus $\delta_{t'_1}\ \mathcal{L}(id_k)\ \delta_{t'_2}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

– $a = get\_corrupt\_sign(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_corrupt\_sign(A)$, we can identify two cases: $t_1.sk_A = \perp$ or $t_1.sk_A \neq \perp$. If $t_1.sk_A = \perp$, then for each state t'_1 of $\mathcal{S}^1_k|G_k$, $\mu_1(t'_1) = \rho_1(s, v)$ and t'_1 is identified by the same values of t_1 except for the following values: $t'_1.sk_A = s$, $t'_1.vk_A = v$, and $t'_1.ck_A = (v, s)$ where ρ_1 is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Since $t_1\ id_k\ t_2$, we have that also t_2 satisfies $t_2.sk_A = \perp$, and hence for each state t'_2 of $\mathcal{S}^2_k$, $\mu_2(t'_2) = \rho_2(s, v)$ and t'_2 is identified by the same values of t_2 except for the following values: $t'_2.sk_A = s$, $t'_2.vk_A = v$, and $t'_2.ck_A = (v, s)$ where ρ_2 is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. This implies that $t'_1\ id_k\ t'_2$ for each state t'_1 and t'_2 satisfying above conditions and since $\rho_1 = \rho_2$, we have that $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.
If $t_1.sk_A \neq \perp$, then $\mu_1 = \delta_{t'_1}$ where t'_1 is the state of $\mathcal{S}^1_k(\mathbb{A})|G_k$ that is identified by the same values of t_1 except for ck_A where $t'_1.ck_A = (t_1.vk_A, t_1.sk_A)$. Since $t_1\ id_k\ t_2$, we have that also t_2 satisfies $t_2.sk_A \neq \perp$, and hence by definition of action $get\_corrupt\_sign(A)$, t_2 enables $get\_corrupt\_sign(A)$ that leads to the measure $\delta_{t'_2}$ where t'_2 is the state of $\mathcal{S}^2_k(\mathbb{A})$ that is identified by the same values of t_2 except for variable

ck_A where $t'_2.ck_A = (t_2.vk_A, t_2.sk_A)$. This implies that $t'_1\ id_k\ t'_2$ and thus $\delta_{t'_1}\ \mathcal{L}(id_k)\ \delta_{t'_2}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

- $a = ret\_corrupt\_sign(A, V, S)$ for some $A \in \mathbb{A}$, $S \in$ SKey and $V \in$ VKey: by definition of the action $ret\_corrupt\_sign(A, V, S)$, it follows that t_1 satisfies $t_1.ck_A = (V, S)$. Moreover, it follows that $\mu_1 = \delta_{t'_1}$ where t'_1 is the state such that $t'_1.ck_A = \perp$, and all other variables that describe t'_1 have the same value of the variables that describe t_1. Since $t_1\ id_k\ t_2$, we have that also t_2 satisfies $t_2.ck_A = (V, S)$ and thus it enables the transition $(t_2, ret\_corrupt\_sign(A, V, S), \mu_2)$ where μ_2 is the measure $\delta_{t'_2}$ where t'_2 is the state such that $t'_2.ck_A = \perp$, and all other variables that describe t'_2 have the same value of the variables that describe t_2. This implies that $t'_1\ id_k\ t'_2$ and thus $\delta_{t'_1}\ \mathcal{L}(id_k)\ \delta_{t'_2}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.
- $a = get\_sign(A, M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of $get\_sign(A, M)$, it follows that for each state t'_1 of $\mathcal{S}^1_k$, $\mu_1(t'_1) = \rho_1(S)$ and t'_1 is identified by the same values of t_1 except for the following values: $t'_1.is\_fresh_A = F$ if $S \in t_1.generated\_signs$, T otherwise, $t'_1.sig\_value_A = S$, and $t'_1.generated\_signs = t_1.generated\_signs \cup \{S\}$. ρ_1 is the probability measure induced over Signature by the probabilistic algorithm $\mathtt{Sig}(t_1.sk_A, M)$ conditioned to G_k. Since $t_1\ id_k\ t_2$, we have that also t_2 enables the transition $(t_2, get\_sign(A, M), \mu_2)$ where for each state t'_2 of $\mathcal{S}^2_k$, $\mu_2(t'_2) = \rho_2(S)$ and t'_2 is identified by the same values of t_2 except for the following values: $t'_2.is\_fresh_A = F$ if $S \in t_2.generated\_signs$, T otherwise, $t'_2.sig\_value_A = S$, and $t'_2.generated\_signs = t_2.generated\_signs \cup \{C\}$. ρ_2 is the probability measure induced over Signature by the probabilistic algorithm $\mathtt{Sig}(t_2.sk_A, M)$ that is iterated until it returns a value that does not belong to $t_2.generated\_signs$. This implies that $t'_1\ id_k\ t'_2$ for each state t'_1 and t'_2 satisfying above conditions and if $\rho_1 = \rho_2$, then $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.
  ρ_1 is equal to ρ_2 since for each $t \in G_k$, $\rho_1(t) = \rho_2(t)$. In fact, by definition of conditional, $\rho_1(t) = \rho(t)/\rho(G_k)$ where ρ is the probability measure induced over Signature by $\mathtt{Sig}(t_1.sk_A, M)$, $\rho(G_k) > 0$ and $\rho(B_k) < 1$. By definition of $get\_sign(A, M)$ action of $\mathcal{S}^2_k$, we have that $\rho_2(t) = \sum_{i=0}^{+\infty} \rho(B_k)^i \rho(t) = \rho(t) \sum_{i=0}^{+\infty} \rho(B_k)^i = \rho(t) \dfrac{1}{1 - \rho(B_k)} = \rho(t)/\rho(G_k) = \rho_1(t)$.
- $a = ret\_sign(A, S)$ for some $A \in \mathbb{A}$ and $S \in$ Signature: by definition of the action $ret\_sign(A, S)$, it follows that t_1 satisfies $t_1.sig\_value_A =$

S. Moreover, it follows that $\mu_1 = \delta_{t'_1}$ where t'_1 is the state such that $t'_1.sig\_value_A = \bot$, $t'_1.is\_fresh_A = \bot$, and all other variables that describe t'_1 have the same value of the variables that describe t_1. Since $t_1\ id_k\ t_2$, we have that also t_2 satisfies $t_2.sig\_value_A = S$ and thus it enables the transition $(t_2, ret\_sign(A,S), \mu_2)$ where μ_2 is the measure $\delta_{t'_2}$ where t'_2 is the state such that $t'_2.sig\_value_A = \bot$, $t'_2.is\_fresh_A = \bot$, and all other variables that describe t'_2 have the same value of the variables that describe t_2. This implies that $t'_1\ id_k\ t'_2$ and thus $\delta_{t'_1}\ \mathcal{L}(id_k)\ \delta_{t'_2}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

- $a = get\_verify\_sign(A,S)$ for some $A \in \mathbb{A}$ and $S \in$ Signature: by definition of the action $get\_verify\_sign(A,S)$, it follows that $\mu_1 = \delta_{t'_1}$ where t'_1 is the state such that $t'_1.ver\_value_A = \mathtt{Ver}(t_1.vk_A, S)$, and all other variables that describe t'_1 have the same value of the variables that describe t_1. Since $t_1\ id_k\ t_2$, we have that also t_2 enables the transition $(t_2, get\_verify\_sign(A,S), \mu_2)$ where μ_2 is the measure $\delta_{t'_2}$ where t'_2 is the state such that $t'_2.ver\_value_A = \mathtt{Ver}(t_2.vk_A, S)$, and all other variables that describe t'_2 have the same value of the variables that describe t_2. This implies that $t'_1\ id_k\ t'_2$ and thus $\delta_{t'_1}\ \mathcal{L}(id_k)\ \delta_{t'_2}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.
- $a = ret\_verify\_sign(A,B)$ for some $A \in \mathbb{A}$ and $B \in \{T,F\}$: by definition of the action $ret\_verify\_sign(A,B)$, it follows that t_1 satisfies $t_1.ver\_value_A = B$. Moreover, it follows that $\mu_1 = \delta_{t'_1}$ where t'_1 is the state such that $t'_1.ver\_value_A = \bot$, and all other variables that describe t'_1 have the same value of the variables that describe t_1. Since $t_1\ id_k\ t_2$, we have that also t_2 satisfies $t_2.ver\_value_A = B$ and thus it enables the transition $(t_2, ret\_verify\_sign(A,B), \mu_2)$ where μ_2 is the measure $\delta_{t'_2}$ where t'_2 is the state such that $t'_2.ver\_value_A = \bot$, and all other variables that describe t'_2 have the same value of the variables that describe t_2. This implies that $t'_1\ id_k\ t'_2$ and thus $\delta_{t'_1}\ \mathcal{L}(id_k)\ \delta_{t'_2}$, that is $\mu_1\ \mathcal{L}(id_k)\ \mu_2$, as required.

The step condition is satisfied since for each action a if t_1 enables a transition labelled by a that leads to μ_1, then we can find μ_2 such that $(t_2, a, \mu_2) \in D_k^2$ and $\mu_1\ \mathcal{L}(id_k)\ \mu_2$. □

The existence of a polynomially accurate simulation from $\mathcal{S}_k^1(\mathbb{A})|G_k$ to $\mathcal{S}_k^2(\mathbb{A})$ in now straightforward:

Proposition 6.30. *Let $\mathbb{A}$ be a set of agents. Given $\mathcal{S}_k^1(\mathbb{A})$, let B_k be the set of states of $\mathcal{S}_k^1$ such that $t \in B_k$ if there exists $A \in \mathbb{A}$ such that $t.is\_fresh_A = F$. Let G_k be the set $S_k^1 \setminus B_k$.*

For each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k^1(\mathbb{A})$,

$$\{(\mathcal{S}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.29, we have that for each $k \in \mathbb{N}$, $\mathcal{S}_k^1(\mathbb{A})|G_k \preceq \mathcal{S}_k^2(\mathbb{A})$. This implies that for each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k^1(\mathbb{A})$, $(\mathcal{S}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k \preceq \mathcal{S}_k^2(\mathbb{A})||\mathcal{C}_k$. Finally, by Proposition 5.6, we have that $\{(\mathcal{S}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

To complete the chain of simulations from $\mathcal{S}_k(\mathbb{A})$ to $\mathcal{S}_k^2(\mathbb{A})$, we need to prove that

Proposition 6.31. *Let S be a non-repeating unforgeable signature scheme and $\mathbb{A}$ be a set of agents. Let G_k be the set of states of $\mathcal{S}_k^1$ such that $t \in G_k$ if and only if for each $A \in \mathbb{A}$ $t.is\_fresh_A \neq F$, that is, the generated signature is fresh. Let B_k be $S_k^1 \setminus G_k$ (that is, states t' of $\mathcal{S}_k^1$ such that $t'.is\_fresh_A = F$ for some $A \in \mathbb{A}$).*

For each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k^1(\mathbb{A})$,

$$\{\mathcal{S}_k^1(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{S}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. Theorem 5.8 states that $\{\mathcal{S}_k^1(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}_k^1(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$ if and only if $\{B_k\}_{k\in\mathbb{N}}$ is negligible in $\{\mathcal{S}_k^1(\mathbb{A})\}_{k\in\mathbb{N}}$.

Suppose, for the sake of contradiction, that $\{\mathcal{S}_k^1(\mathbb{A})\}_{k\in\mathbb{N}}$ is not simulated by $\{\mathcal{S}_k^1(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$. This implies that $\{B_k\}_{k\in\mathbb{N}}$ is not negligible in $\{\mathcal{S}_k^1(\mathbb{A})\}_{k\in\mathbb{N}}$ and thus that there exists $c \in \mathbb{N}$, $p \in \mathit{Poly}$ such that for each $\bar{k} \in \mathbb{N}$ there exists $k > \bar{k}$ such the probability to reach states of B_k within $p(k)$ steps is at least k^{-c}, that is, the probability to reach states t such that $t.is\_fresh_A = F$ for some $A \in \mathbb{A}$ within $p(k)$ steps is at least k^{-c}. By definition of the automaton $\mathcal{S}_k^1(\mathbb{A})$, it follows that $t.is\_fresh_A$ can assume value F only as the effect of a transition that leaves from some state t' and that is labelled by action $get\_sign(A, M)$. This happens only when the signature value $\mathtt{Sig}(t'.sk_A, M)$ assigned to $t.sig\_value_A$ belongs to the set $RC = t'.generated\_signs$. Within $p(k)$ steps, we can perform at most $p(k)$ transitions labelled by $get\_sign(Z, P)$ with $Z \in \mathbb{A}$ and $P \in$ Message. Since by definition of $get\_sign(Z, P)$ we add at most one value to $generated\_signs$ each time we perform a transition labelled by

get_sign(Z, P), within $p(k)$ steps the set RC has cardinality at most $p(k)$. This means that with probability at least k^{-c}, we have generated a signature that it is equal to some $c \in RC$ with $|RC| \leqslant p(k)$. Since the encryption scheme S is non-repeating unforgeable, this contradicts the Definition 6.26 and thus $\{\mathcal{S}_k^1(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}_k^1(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$. This implies, by Theorem 5.10, that $\{\mathcal{S}_k^1(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{S}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

Avoiding collisions with externally generated signatures

As we have seen above, there exists a simulation from the signature automaton that can generate repeated signatures to the signature automaton that ensures that returned signatures are never repeated. Sometimes, the fact that $\mathcal{S}_k^2$ does not generate repeated signatures is not sufficient to satisfy the required properties: it is still possible that $\mathcal{S}_k^2$ generates a sign that is equal to some adversary's generated signature.

If we want to be sure that the signature automaton returns values that have never occurred previously, then we must modify it adding the knowledge of environment's generated signatures. We can provide such knowledge using the following approach: the encryption automaton receives as input sets of environment's generated signatures and then it ensures that generated values do not belong to such sets as well as previously returned values.

This means that we provide the signature automaton with the signatures generated by the environment sending the set $\{s_1, \ldots, s_l\}$ to the automaton, where $s_1, \ldots, s_l$ are signatures generated by the environment. We do this adding a state variable *used_signatures* that contains all signatures used by the environment and an input action *used_signatures*(SN) that updates *used_signatures*:

Input *used_signatures*(SN)
  Effect:
    used_signatures := *used_signatures* $\cup$ SN

To simplify the characterization of states where the generated signature matches with some value received from the environment, we add the family of state variables $is\_not\_used_A$ for $A \in \mathbb{A}$ that assume values in $\{T, F, \perp\}$ with initial value $\perp$. Moreover, each *get_sign*(A, M) action modifies $is\_not\_used_A$ assigning F if the ciphertext returned by $\mathtt{Sig}(sk_A, M)$ is in *used_signatures*; T otherwise, as depicted in Figure 6.15. The action

$\mathcal{S}_k^2(\mathbb{A})$

Input $get\_sign(A, M)$
Effect:
repeat
$s = \mathtt{Sig}(sk_A, M)$
until $s \notin generated\_signs$
$sig\_value_A := s$
$is\_fresh_A := \begin{cases} F & \text{if } s \in generated\_signs \\ T & \text{otherwise} \end{cases}$
$generated\_signs := generated\_signs \cup \{s\}$

Output $ret\_encrypt(A, C)$
Precondition:
$sig\_value_A = C$
Effect:
$is\_fresh_A := \bot$
$sig\_value_A := \bot$

$\mathcal{S}_k^3(\mathbb{A})$

Input $get\_sign(A, M)$
Effect:
repeat
$s = \mathtt{Sig}(sk_A, M)$
until $s \notin generated\_signs$
$sig\_value_A := s$
$is\_fresh_A := \begin{cases} F & \text{if } s \in generated\_signs \\ T & \text{otherwise} \end{cases}$
$is\_not\_used_A := \begin{cases} F & \text{if } s \in used\_signatures \\ T & \text{otherwise} \end{cases}$
$generated\_signs := generated\_signs \cup \{s\}$

Output $ret\_encrypt(A, C)$
Precondition:
$sig\_value_A = C$
Effect:
$is\_fresh_A := \bot$
$is\_not\_used_A := \bot$
$sig\_value_A := \bot$

Fig. 6.15. $get\_sign(A, M)$ and $ret\_sign(A, S)$ of $\mathcal{S}_k^2$ and $\mathcal{S}_k^3$

ret_sign(A, S) also resets the $is\_not\_used_A$ to its initial value $\perp$. Let $\mathcal{S}_k^3$ be the resulting automaton.

It is immediate to see that $\mathcal{S}_k^3(\mathbb{A})$ is an extension of $\mathcal{S}_k^2(\mathbb{A})$. In fact, we simply add some history variables that keep information about all signatures generated by the environment. Such variables do not affect the behavior of the other actions and thus each transition of $\mathcal{S}_k^3(\mathbb{A})$ is either almost identical to a transition of $\mathcal{S}_k^2(\mathbb{A})$ or the new action that does not modify the state variables that describe states of $\mathcal{S}_k^2(\mathbb{A})$.

The above intuition is formalized by the following result:

Lemma 6.32. *Let $\mathbb{A}$ be a set of agents, B be the set $\{used\_signatures(SN) \mid SN \subseteq \text{Signature}\}$ and W be the set $\{used\_signatures\} \cup \{is\_not\_used_A \mid A \in \mathbb{A}\}$ where action and variables are defined as above. For each $k \in \mathbb{N}$, $\mathcal{S}_k^3(\mathbb{A}) \in \text{Ext}_B^W(\mathcal{S}_k^2(\mathbb{A}))$.*

Proof. To prove the statement of the Lemma, we need to check if the requirements of Definition 4.5 are satisfied:

compatible states: let v be a state variable of $\mathcal{S}_k^3(\mathbb{A})$. Then v is either one of *generated_signs*, sk_A, vk_A, pk_A, ck_A, sig_value_A, ver_value_A, is_fresh_A for $A \in \mathbb{A}$ and thus it is a state variable of $\mathcal{S}_k^2(\mathbb{A})$, or v is *used_signatures* or $is\_not\_used_A$ for some $A \in \mathbb{A}$ and thus $v \in W$;

compatible start state: $\bar{s}_k^3$ is identified by value $\perp$ for each sk_A, vk_A, pk_A, ck_A, sig_value_A, ver_value_A, is_fresh_A, and $is\_not\_used_A$ (with $A \in \mathbb{A}$), and by $\emptyset$ for *generated_signs* and *used_signatures*. Since $\bar{s}_k^2$ is identified by the value $\perp$ for each sk_A, vk_A, pk_A, ck_A, sig_value_A, ver_value_A, and is_fresh_A (with $A \in \mathbb{A}$), and by $\emptyset$ for *generated_signs*, then $\bar{s}_k^2 = \bar{s}_k^3 \restriction_{\bar{s}_k^2}$;

compatible actions: by definition of $\mathcal{S}_k^3(\mathbb{A})$, it provides the same actions of $\mathcal{S}_k^2(\mathbb{A})$ plus the actions $used\_signatures(SN)$ that belong to B;

compatible transitions: let $tr_3 = (t_3, a, \mu_3) \in D_k^3$ be a transition of $\mathcal{S}_k^3$. Suppose that $a = used\_signatures(SN)$. Then we must verify that there exists a state t_2 of $\mathcal{S}_k^2$ such that $t_2 = t_3 \restriction_{t_2}$ and $\delta_{t_2} = \mu_3 \restriction_{\delta_{t_2}}$. By definition of $used\_signatures(SN)$, it follows that $\mu_3 = \delta_{t_3'}$ where t_3' is the state of $\mathcal{S}_k^3(\mathbb{A})$ that is identified by the same values of t_3 except for *used_signatures* where $t_3'.used\_signatures = t_3.used\_signatures \cup SN$. Let t_2 be the state of $\mathcal{S}_k^2(\mathbb{A})$ such that $t_2 = t_3 \restriction_{t_2}$. Since the only difference between t_3 and t_3' is on the value of *used_signatures*, and since *used_signatures* is not a variable that characterizes t_2, we have that $t_2 = t_3' \restriction_{t_2}$ and thus $\delta_{t_2} = \delta_{t_3' \restriction_{\delta_{t_2}}}$.

Suppose that $a \in A_k^2$, that is, it is an action of $\mathcal{S}_k^2(\mathbb{A})$. Then we must verify that there exists a transition $tr_2 = (t_2, a, \mu_2) \in D_k^2$ such that $tr_2 = tr_3{\restriction}_{tr_2}$. There are eight cases:

- $a = \mathit{get\_public\_sign}(A)$ for some $A \in \mathbb{A}$: by definition of the action $\mathit{get\_public\_sign}(A, M)$, we can identify two cases: $t_3.sk_A = \bot$ or $t_3.sk_A \neq \bot$. If $t_3.sk_A = \bot$, then for each state t_3' of $\mathcal{S}_k^3$, $\mu_3(t_3') = \rho(s, v)$ and t_3' is identified by the same values of t_3 except for the following values: $t_3'.sk_A = s$, $t_3'.vk_A = v$, and $t_3'.pk_A = v$ where ρ is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\mathtt{KGen}(1^k)$. Let t_2 be the state of $\mathcal{S}_k^2(\mathbb{A})$ such that $t_2 = t_3{\restriction}_{t_2}$. By definition of action $\mathit{get\_public\_sign}(A)$, t_2 enables $\mathit{get\_public\_sign}(A)$ that leads to the measure μ_2 such that for each state t_2' of $\mathcal{S}_k^2$, $\mu_2(t_2') = \rho(s, v)$ and t_2' is identified by the same values of t_2 except for the following values: $t_2'.sk_A = s$, $t_2'.vk_A = v$, and $t_2'.pk_A = s$ where ρ is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\mathtt{KGen}(1^k)$. Thus $\mu_2 = \mu_3{\restriction}_{\mu_2}$, and hence $tr_2 = tr_3{\restriction}_{tr_2}$.
  If $t_3.sk_A \neq \bot$, then $\mu_3 = \delta_{t_3'}$ where t_3' is the state of $\mathcal{S}_k^3(\mathbb{A})$ that is identified by the same values of t_3 except for pk_A where $t_3'.pk_A = t_3.vk_A$. Let t_2 be the state of $\mathcal{S}_k^2(\mathbb{A})$ such that $t_2 = t_3{\restriction}_{t_2}$. By definition of $\mathit{get\_public\_sign}(A)$, t_2 enables $\mathit{get\_public\_sign}(A)$ that leads to the measure $\delta_{t_2'}$ where t_2' is the state of $\mathcal{S}_k^2(\mathbb{A})$ that is identified by the same values of t_2 except for variable pk_A where $t_2'.pk_A = t_2.vk_A$. Thus $\delta_{t_2'} = \delta_{t_3'}{\restriction}_{\delta_{t_2'}}$, and hence $tr_2 = tr_3{\restriction}_{tr_2}$.
- $a = \mathit{ret\_public\_sign}(A, V)$ for some $A \in \mathbb{A}$ and $E \in$ EKey: by definition of action $\mathit{ret\_public\_sign}(A, V)$, it follows that $t_3.pk_A = V$ and that $\mu_3 = \delta_{t_3'}$ where t_3' is the state of $\mathcal{S}_k^3(\mathbb{A})$ that is identified by the same values of t_3 except for pk_A where $t_3'.pk_A = \bot$. Let t_2 be the state of $\mathcal{S}_k^2(\mathbb{A})$ such that $t_2 = t_3{\restriction}_{t_2}$. By definition of action $\mathit{ret\_public\_sign}(A, V)$, t_2 enables $\mathit{ret\_public\_sign}(A, V)$ that leads to the measure $\delta_{t_2'}$ where t_2' is the state of $\mathcal{S}_k^2(\mathbb{A})$ that is identified by the same values of t_2 except for variable pk_A where $t_2'.pk_A = \bot$. Thus $\delta_{t_2'} = \delta_{t_3'}{\restriction}_{\delta_{t_2'}}$, and hence $tr_2 = tr_3{\restriction}_{tr_2}$.
- $a = \mathit{get\_corrupt\_sign}(A)$ for some $A \in \mathbb{A}$: by definition of the action $\mathit{get\_corrupt\_sign}(A)$, we can identify two cases: $t_3.sk_A = \bot$ or $t_3.sk_A \neq \bot$. If $t_3.sk_A = \bot$, then for each state t_3' of $\mathcal{S}_k^3$, $\mu_3(t_3') = \rho(s, v)$ and t_3' is identified by the same values of t_3 except for the following values: $t_3'.sk_A = s$, $t_3'.vk_A = v$, and $t_3'.ck_A = (v, s)$

where ρ is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Let t_2 be the state of $\mathcal{S}_k^2(\mathbb{A})$ such that $t_2 = t_3\lceil_{t_2}$. By definition of action *get_corrupt_sign*(A), t_2 enables *get_corrupt_sign*(A) that leads to the measure μ_2 such that for each state t_2' of $\mathcal{S}_k^2$, $\mu_2(t_2') = \rho(s, v)$ and t_2' is identified by the same values of t_2 except for the following values: $t_2'.sk_A = s$, $t_2'.vk_A = v$, and $t_2'.ck_A = (v, s)$ where ρ is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Thus $\mu_2 = \mu_2\lceil_{\mu_2}$, and hence $tr_2 = tr_3\lceil_{tr_2}$.
If $t_3.sk_A \neq \perp$, then $\mu_3 = \delta_{t_3'}$ where t_3' is the state of $\mathcal{S}_k^3(\mathbb{A})$ that is identified by the same values of t_3 except for ck_A where $t_3'.ck_A = (t_3.vk_A, t_3.sk_A)$. Let t_2 be the state of $\mathcal{S}_k^2(\mathbb{A})$ such that $t_2 = t_3\lceil_{t_2}$. By definition of action *get_corrupt_sign*(A), t_2 enables *get_corrupt_sign*(A) that leads to the measure $\delta_{t_2'}$ where t_2' is the state of $\mathcal{S}_k^2(\mathbb{A})$ that is identified by the same values of t_2 except for variable ck_A where $t_2'.ck_A = (t_2.vk_A, t_2.sk_A)$. Thus $\delta_{t_2'} = \delta_{t_3'}\lceil_{\delta_{t_2'}}$, and hence $tr_2 = tr_3\lceil_{tr_2}$.

- $a =$ *ret_corrupt_sign*(A, V, S) for some $A \in \mathbb{A}$, $V \in$ VKey, and $S \in$ SKey: by definition of action *ret_corrupt_sign*(A, V, S), it follows that $t_3.ck_A = (V, S)$ and that $\mu_3 = \delta_{t_3'}$ where t_3' is the state of $\mathcal{S}_k^3(\mathbb{A})$ that is identified by the same values of t_3 except for ck_A where $t_3'.ck_A = \perp$. Let t_2 be the state of $\mathcal{S}_k^2(\mathbb{A})$ such that $t_2 = t_3\lceil_{t_2}$. By definition of action *ret_corrupt_sign*(A, V, S), t_2 enables *ret_corrupt_sign*(A, V, S) that leads to the measure $\delta_{t_2'}$ where t_2' is the state of $\mathcal{S}_k^2(\mathbb{A})$ that is identified by the same values of t_2 except for variable ck_A where $t_2'.ck_A = \perp$. Thus $\delta_{t_2'} = \delta_{t_3'}\lceil_{\delta_{t_2'}}$, and hence $tr_2 = tr_3\lceil_{tr_2}$.
- $a =$ *get_sign*(A, M) for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of the action *get_sign*(A, M), it follows that for each state t_3' of $\mathcal{S}_k^3$, $\mu_3(t_3') = \rho(S)$ and t_3' is identified by the same values of t_3 except for the following values that are: $t_3'.is\_fresh_A = F$ if $S \in t_3.generated\_signs$, T otherwise, $t_3'.sig\_value_A = S$, and $t_3'.generated\_signs = t_3.generated\_signs \cup \{S\}$. ρ is the probability measure induced over Signature by the probabilistic algorithm $\texttt{Sig}(t_3.sk_A, M)$. Let t_2 be the state of $\mathcal{S}_k^2(\mathbb{A})$ such that $t_2 = t_3\lceil_{t_2}$. By definition of action *get_sign*(A, M), t_2 enables *get_sign*(A, M) that leads to the measure ρ where for each state t_2' of $\mathcal{S}_k^2(\mathbb{A})$ that is identified by the same values of t_2 except for variable sig_value_A, we have

that $\mu(t'_2) = \rho(S)$ and $t'_2.sig\_value_A = S$. Thus $\mu_2 = \mu_3 \restriction_{\mu_2}$, and hence $tr_2 = tr_3 \restriction_{tr_2}$.

- $a = ret\_sign(A, S)$ for some $A \in \mathbb{A}$ and $S \in$ Signature: by definition of action $ret\_sign(A, S)$, it follows that $t_3.sig\_value_A = S$ and that $\mu_3 = \delta_{t'_3}$ where t'_3 is the state of $\mathcal{S}^3_k(\mathbb{A})$ that is identified by the same values of t_3 except for sig_value_A that assumes the value $t'_3.sig\_value_A = \perp$ and for is_fresh_A and $is\_not\_used_A$ that are reset to $\perp$. Let t_2 be the state of $\mathcal{S}^2_k(\mathbb{A})$ such that $t_2 = t_3 \restriction_{t_2}$. By definition of action $ret\_sign(A, S)$, t_2 enables $ret\_sign(A, S)$ that leads to the measure $\delta_{t'_2}$ where t'_2 is the state of $\mathcal{S}^2_k(\mathbb{A})$ that is identified by the same values of t_2 except for variable sig_value_A where $t'_2.sig\_value_A = \perp$. Thus $\delta_{t'_2} = \delta_{t'_3} \restriction_{\delta_{t'_2}}$, and hence $tr_2 = tr_3 \restriction_{tr_2}$.
- $a = get\_verify\_sign(A, S)$ for some $A \in \mathbb{A}$ and $S \in$ Signature: by definition of action $get\_verify\_sign(A, S)$, it follows that $\mu_3 = \delta_{t'_3}$ where t'_3 is the state of $\mathcal{S}^3_k(\mathbb{A})$ that is identified by the same values of t_3 except for ver_value_A where $t'_3.ver\_value_A = \mathtt{Ver}(t_3.vk_A, S)$. Let t_2 be the state of $\mathcal{S}^2_k(\mathbb{A})$ such that $t_2 = t_3 \restriction_{t_2}$. By definition of action $get\_verify\_sign(A, S)$, t_2 enables $get\_verify\_sign(A, S)$ that leads to the measure $\delta_{t'_2}$ where t'_2 is the state of $\mathcal{S}^2_k(\mathbb{A})$ that is identified by the same values of t_2 except for variable ver_value_A where $t'_2.ver\_value_A = \mathtt{Ver}(s_2.vk_A, S)$. Thus $\delta_{t'_2} = \delta_{t'_3} \restriction_{\delta_{t'_2}}$, and hence $tr_2 = tr_3 \restriction_{tr_2}$.
- $a = ret\_verify\_sign(A, B)$ for some $A \in \mathbb{A}$ and $B \in \{T, F\}$: by definition of action $ret\_verify\_sign(A, B)$, it follows that $t_3.ver\_value_A = B$ and that $\mu_3 = \delta_{t'_3}$ where t'_3 is the state of $\mathcal{S}^3_k(\mathbb{A})$ that is identified by the same values of t_3 except for ver_value_A where $t'_3.ver\_value_A = \perp$. Let t_2 be the state of $\mathcal{S}^2_k(\mathbb{A})$ such that $t_2 = t_3 \restriction_{t_2}$. By definition of action $ret\_verify\_sign(A, B)$, t_2 enables $ret\_verify\_sign(A, B)$ that leads to the measure $\delta_{t'_2}$ where t'_2 is the state of $\mathcal{S}^2_k(\mathbb{A})$ that is identified by the same values of t_2 except for variable ver_value_A where $t'_2.ver\_value_A = \perp$. Thus $\delta_{t'_2} = \delta_{t'_3} \restriction_{\delta_{t'_2}}$, and hence $tr_2 = tr_3 \restriction_{tr_2}$.

Since the requirements of Definition 4.5 are satisfied, for each $k \in \mathbb{N}$, $\mathcal{S}^3_k(\mathbb{A}) \in \mathrm{Ext}^W_B(\mathcal{S}^2_k(\mathbb{A}))$. □

The existence of a polynomially accurate simulation from $\mathcal{S}^2_k(\mathbb{A})$ to $\mathcal{S}^3_k(\mathbb{A})$ in now straightforward:

Proposition 6.33. *Let $\mathbb{A}$ be a set of (identities of) agents. For each context $\mathcal{C}_k$ compatible with $\mathcal{S}^2_k(\mathbb{A})$ such that $\{used\_signatures(SN) \mid SN \subseteq$*

Signature$\} \subseteq A_{\mathcal{C}_k}$,

$$\{\mathcal{S}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \leq_s \{\mathcal{S}_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.32, for each $k \in \mathbb{N}$, $\mathcal{S}_k^3(\mathbb{A}) \in \mathrm{Ext}_B^W(\mathcal{S}_k^2(\mathbb{A}))$ where B is the set of actions $\{used\_signatures(SN) \mid SN \subseteq \text{Signature}\}$ and W is $\{used\_signatures\} \cup \{is\_not\_used_A \mid A \in \mathbb{A}\}$. This implies, by Lemma 4.6, that for each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k^2(\mathbb{A})$ such that $B \subseteq A_{\mathcal{C}_k}$, $\mathcal{S}_k^2(\mathbb{A})||\mathcal{C}_k \preceq \mathcal{S}_k^3(\mathbb{A})||\mathcal{C}_k$ and thus, by Proposition 5.6, $\{\mathcal{S}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \leq_s \{\mathcal{S}_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

For the second step, let $\mathcal{S}_k^4$ be the automaton obtained from $\mathcal{S}_k^3$ modifying each $get\_sign(A, M)$ action as follows: the signature algorithm `Sig` is invoked until it returns a value that is a signature that is not yet used, that is not inside *used_signatures*, as depicted in Figure 6.16.

It is immediate to see that $\mathcal{S}_k^4(\mathbb{A})$ simulates the G_k-conditional of $\mathcal{S}_k^3(\mathbb{A})$ where for each $k \in \mathbb{N}$, G_k is the set of states of $\mathcal{S}_k^3(\mathbb{A})$ such that for each $A \in \mathbb{A}$, $\neq .is\_not\_used_A F$. In fact, we iterate the generation of the signature of M under the private key of A until it returns a value that is not equal to an already used signature. That is, we condition the choice of the signature to the fact that it is fresh (and thus, $\neq .is\_not\_used_A F$ is always satisfied).

Note that $\mathcal{S}_k^4(\mathbb{A})$ is not the G_k-conditional of $\mathcal{S}_k^3(\mathbb{A})$ since it provides more transitions than $\mathcal{S}_k^3(\mathbb{A})|G_k$. In fact, $\mathcal{S}_k^4(\mathbb{A})$ enables transitions that leave from a state not in G_k and that leads to a probability measures μ such that $\mu(G_k) = 0$. For example, let t be the state such that $t.is\_not\_used_A = F$, $t.is\_not\_used_B = F$, $t.sig\_value_A = S$. Thus, by definition of *ret_sign* actions, t enables the action $ret\_sign(A, S)$ that leads to the measure $\delta_{t'}$ where t' is identified by the same values of t except for variables sig_value_A and is_fresh_A that are $\perp$. Thus $t'.is\_not\_used_B$ is still F and hence $t' \notin G_k$.

The above intuition is formalized by the following result:

Lemma 6.34. *Given $\mathcal{S}_k^3(\mathbb{A})$, let B_k be the set of states of $\mathcal{S}_k^3$ such that $t \in B_k$ if there exists $A \in \mathbb{A}$ such that $t.is\_not\_used_A = F$. Let G_k be the set $S_k^3 \setminus B_k$.*

For each $k \in \mathbb{N}$, $\mathcal{S}_k^3(\mathbb{A})|G_k \preceq \mathcal{S}_k^4(\mathbb{A})$.

Proof. For each $k \in \mathbb{N}$, let id_k be the identity relation on states. id_k is a simulation from $\mathcal{S}_k^3(\mathbb{A})|G_k$ to $\mathcal{S}_k^4(\mathbb{A})$.

The condition on start states is trivially true: let $\bar{s}_k^3$ be the start state of $\mathcal{S}_k^3(\mathbb{A})|G_k$ and $\bar{s}_k^4$ be the start state of $\mathcal{S}_k^4(\mathbb{A})$. By definition of conditional, it

$\mathcal{S}_k^3$

Input $get\_sign(A, M)$
Effect:
repeat
$s = \mathtt{Sig}(sk_A, M)$
until $s \notin generated\_signs$
$sig\_value_A := s$
$is\_fresh_A := \begin{cases} F & \text{if } s \in generated\_signs \\ T & \text{otherwise} \end{cases}$
$is\_not\_used_A := \begin{cases} F & \text{if } s \in used\_signatures \\ T & \text{otherwise} \end{cases}$
$generated\_signs := generated\_signs \cup \{s\}$

$\mathcal{S}_k^4$

Input $get\_sign(A, M)$
Effect:
repeat
repeat
$s = \mathtt{Sig}(sk_A, M)$
until $s \notin generated\_signs$
until $s \notin used\_signatures$
$sig\_value_A := s$
$is\_fresh_A := \begin{cases} F & \text{if } s \in generated\_signs \\ T & \text{otherwise} \end{cases}$
$is\_not\_used_A := \begin{cases} F & \text{if } s \in used\_signatures \\ T & \text{otherwise} \end{cases}$
$generated\_signs := generated\_signs \cup \{s\}$

Fig. 6.16. *get_sign* of $\mathcal{S}_k^3$ and $\mathcal{S}_k^4$

follows that $\bar{s}_k^3$ is the start state of $\mathcal{S}_k^3(\mathbb{A})$. Since by definition of $\mathcal{S}_k^4(\mathbb{A})$, the only difference between $\mathcal{S}_k^3(\mathbb{A})$ and $\mathcal{S}_k^4(\mathbb{A})$ is on the definition of the action $get\_sign(A, M)$, we have that $\bar{s}_k^4 = \bar{s}_k^3$ and thus $\bar{s}_k^3\ id_k\ \bar{s}_k^4$.

For the step condition, let t_3 and t_4 be two states of $\mathcal{S}_k^3(\mathbb{A})|G_k$ and $\mathcal{S}_k^4(\mathbb{A})$, respectively, such that $t_3\ id_k\ t_4$. Let (t_3, a, μ_3) be a transition of $\mathcal{S}_k^3(\mathbb{A})|G_k$ that leaves from t_3. We must find μ_4 such that (t_4, a, μ_4) is a transition of $\mathcal{S}_k^4(\mathbb{A})$ and $\mu_3\ \mathcal{L}(id_k)\ \mu_4$. There are nine cases:

– $a = get\_public\_sign(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_public\_sign(A)$, we can identify two cases: $t_3.sk_A = \perp$ or $t_3.sk_A \neq \perp$. If $t_3.sk_A = \perp$, then for each state t_3' of $\mathcal{S}_k^3$, $\mu_3(t_3') = \rho_3(s, v)$ and t_3'

is identified by the same values of t_3 except for the following values: $t'_3.sk_A = s$, $t'_3.vk_A = v$, and $t'_3.pk_A = v$ where ρ_3 is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Since $t_3\ id_k\ t_4$, we have that also t_4 satisfies $t_4.sk_A = \perp$, and hence for each state t'_4 of $\mathcal{S}^4_k$, $\mu_4(t'_4) = \rho_4(s, v)$ and t'_4 is identified by the same values of t_4 except for the following values: $t'_4.sk_A = s$, $t'_4.vk_A =$, and $t'_4.pk_A = v$ where ρ_4 is the probability measure induced over SKey $\times$ VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. This implies that $t'_3\ id_k\ t'_4$ for each state t'_3 and t'_4 satisfying above conditions and since $\rho_3 = \rho_4$, we have that $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
If $t_3.sk_A \neq \perp$, then $\mu_3 = \delta_{t'_3}$ where t'_3 is the state of $\mathcal{S}^3_k(\mathbb{A})$ that is identified by the same values of t_3 except for pk_A where $t'_3.pk_A = t_3.vk_A$. Since $t_3\ id_k\ t_4$, we have that also t_4 satisfies $t_4.sk_A \neq \perp$, and hence by definition of action $get\_public\_sign(A)$, t_4 enables $get\_public\_sign(A)$ that leads to the measure $\delta_{t'_4}$ where t'_4 is the state of $\mathcal{S}^4_k(\mathbb{A})$ that is identified by the same values of t_4 except for variable pk_A where $t'_4.pk_A = t_4.vk_A$. This implies that $t'_3\ id_k\ t'_4$ and thus $\delta_{t'_3}\ \mathcal{L}(id_k)\ \delta_{t'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.

- $a = ret\_public\_sign(A, V)$ for some $A \in \mathbb{A}$ and $V \in$ VKey: by definition of the action $ret\_public\_sign(A, V)$, it follows that t_3 satisfies $t_3.pk_A = V$. Moreover, it follows that $\mu_3 = \delta_{t'_3}$ where t'_3 is the state such that $t'_3.pk_A = \perp$, and all other variables that describe t'_3 have the same value of the variables that describe t_3. Since $t_3\ id_k\ t_4$, we have that also t_4 satisfies $t_4.pk_A = V$ and thus it enables the transition $(t_4, ret\_public\_sign(A, V), \mu_4)$ where μ_4 is the measure $\delta_{t'_4}$ where t'_4 is the state such that $t'_4.pk_A = \perp$, and all other variables that describe s'_4 have the same value of the variables that describe t_4. This implies that $t'_3\ id_k\ t'_4$ and thus $\delta_{t'_3}\ \mathcal{L}(id_k)\ \delta_{t'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
- $a = get\_corrupt\_sign(A)$ for some $A \in \mathbb{A}$: by definition of the action $get\_corrupt\_sign(A)$, we can identify two cases: $t_3.sk_A = \perp$ or $t_3.sk_A \neq \perp$. If $t_3.sk_A = \perp$, then for each state t'_3 of $\mathcal{S}^3_k$, $\mu_3(t'_3) = \rho_3(s, v)$ and t'_3 is identified by the same values of t_3 except for the following values: $t'_3.sk_A = s$, $t'_3.vk_A = v$, and $t'_3.ck_A = (v, s)$ where ρ_3 is the probability measure induced over SKey$\times$VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. Since $t_3\ id_k\ t_4$, we have that also t_4 satisfies $t_4.sk_A = \perp$, and hence for each state t'_4 of $\mathcal{S}^4_k$, $\mu_4(t'_4) = \rho_4(s, v)$ and t'_4 is identified by the same values of t_4 except for the following values: $t'_4.sk_A = s$, $t'_4.vk_A = v$, and $t'_4.ck_A = (v, s)$ where ρ_4 is the probability measure induced over

SKey × VKey by the probabilistic algorithm $\texttt{KGen}(1^k)$. This implies that $t'_3\ id_k\ t'_4$ for each state t'_3 and t'_4 satisfying above conditions and since $\rho_3 = \rho_4$, we have that $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
If $t_3.sk_A \neq \bot$, then $\mu_3 = \delta_{t'_3}$ where t'_3 is the state of $\mathcal{S}^3_k(\mathbb{A})$ that is identified by the same values of t_3 except for ck_A where $t'_3.ck_A = (t_3.vk_A, t_3.sk_A)$. Since $t_3\ id_k\ t_4$, we have that also t_4 satisfies $t_4.sk_A \neq \bot$, and hence by definition of action $get\_corrupt\_sign(A)$, t_4 enables $get\_corrupt\_sign(A)$ that leads to the measure $\delta_{t'_4}$ where t'_4 is the state of $\mathcal{S}^4_k(\mathbb{A})$ that is identified by the same values of t_4 except for variable ck_A where $t'_4.ck_A = (t_4.vk_A, t_4.sk_A)$. This implies that $t'_3\ id_k\ t'_4$ and thus $\delta_{t'_3}\ \mathcal{L}(id_k)\ \delta_{t'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.

– $a = ret\_corrupt\_sign(A, V, S)$ for some $A \in \mathbb{A}$, $V \in$ VKey and $S \in$ SKey: by definition of the action $ret\_corrupt\_sign(A, V, S)$, it follows that t_3 satisfies $t_3.ck_A = (V, S)$. Moreover, it follows that $\mu_3 = \delta_{t'_3}$ where t'_3 is the state such that $t'_3.ck_A = \bot$, and all other variables that describe t'_3 have the same value of the variables that describe t_3. Since $t_3\ id_k\ t_4$, we have that also t_4 satisfies $t_4.ck_A = (V, S)$ and thus it enables the transition $(t_4, ret\_corrupt\_sign(A, V, S), \mu_4)$ where μ_4 is the measure $\delta_{t'_4}$ where t'_4 is the state such that $t'_4.ck_A = \bot$, and all other variables that describe t'_4 have the same value of the variables that describe t_4. This implies that $t'_3\ id_k\ t'_4$ and thus $\delta_{t'_3}\ \mathcal{L}(id_k)\ \delta_{t'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
– $a = get\_sign(A, M)$ for some $A \in \mathbb{A}$ and $M \in$ Message: by definition of $get\_sign(A, M)$, it follows that for each state t'_3 of $\mathcal{S}^3_k$, $\mu_3(t'_3) = \rho_3(S)$ and t'_3 is identified by the same values of t_3 except for the following values: $t'_3.is\_fresh_A = F$ if $S \in t_3.generated\_signs$, T otherwise, $t'_3.sig\_value_A = S$, and $t'_3.generated\_signs = t_3.generated\_signs \cup \{S\}$. ρ_3 is the probability measure induced over Signature by the probabilistic algorithm $\texttt{Sig}(t_3.sk_A, M)$ conditioned to the set of states such that for each $A \in \mathbb{A}$ $is\_fresh_A \neq F$ conditioned to G_k. Since $t_3\ id_k\ t_4$, we have that also t_4 enables the transition $(t_4, get\_sign(A, M), \mu_4)$ where for each state t'_4 of $\mathcal{S}^4_k$, $\mu_4(t'_4) = \rho_4(S)$ and t'_4 is identified by the same values of t_4 except for the following values: $t'_4.is\_fresh_A = F$ if $S \in t_4.generated\_signs$, T otherwise, $t'_4.sig\_value_A = S$, and $t'_4.generated\_signs = t_4.generated\_signs \cup \{S\}$. ρ_4 is the probability measure induced over Signature by the probabilistic algorithm $\texttt{Sig}(t_4.sk_A, M)$ conditioned to the set of states such that for each $A \in \mathbb{A}$ $is\_fresh_A \neq F$ that is iterated until it returns a value that does not belong to $t_4.used\_signatures$. This implies that $t'_3\ id_k\ t'_4$ for

each state t'_3 and t'_4 satisfying above conditions and if $\rho_3 = \rho_4$, then $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.

ρ_3 is equal to ρ_4 since for each $t \in G_k$, $\rho_3(t) = \rho_4(t)$. In fact, by definition of conditional, $\rho_3(t) = \rho(t)/\rho(G_k)$ where ρ is the probability measure induced over Signature by $\mathtt{Sig}(t_3.sk_A, M)$, $\rho(G_k) > 0$ and $\rho(B_k) < 1$. By definition of $get\_sign(A, M)$ action of $\mathcal{S}^4_k$, we have that $\rho_4(t) = \sum_{i=0}^{+\infty} \rho(B_k)^i \rho(t) = \rho(t) \sum_{i=0}^{+\infty} \rho(B_k)^i = \rho(t) \dfrac{1}{1-\rho(B_k)} = \rho(t)/\rho(G_k) = \rho_3(t)$.

- $a = ret\_sign(A, S)$ for some $A \in \mathbb{A}$ and $S \in$ Signature: by definition of the action $ret\_sign(A, S)$, it follows that t_3 satisfies $t_3.sig\_value_A = S$. Moreover, it follows that $\mu_3 = \delta_{t'_3}$ where t'_3 is the state such that $t'_3.sig\_value_A = \bot$, $t'_3.is\_fresh_A = \bot$, $t'_3.is\_not\_used_A = \bot$, and all other variables that describe t'_3 have the same value of the variables that describe t_3. Since $t_3\ id_k\ t_4$, we have that also t_4 satisfies $t_4.sig\_value_A = S$ and thus it enables the transition $(t_4, ret\_sign(A, S), \mu_4)$ where μ_4 is the measure $\delta_{t'_4}$ where t'_4 is the state such that $t'_4.sig\_value_A = \bot$, $t'_4.is\_fresh_A = \bot$, $t'_4.is\_not\_used_A = \bot$ and all other variables that describe t'_4 have the same value of the variables that describe t_4. This implies that $t'_3\ id_k\ t'_4$ and thus $\delta_{t'_3}\ \mathcal{L}(id_k)\ \delta_{t'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
- $a = get\_verify\_sign(A, S)$ for some $A \in \mathbb{A}$ and $S \in$ Signature: by definition of the action $get\_verify\_sign(A, S)$, it follows that $\mu_3 = \delta_{t'_3}$ where t'_3 is the state such that $t'_3.ver\_value_A = \mathtt{Ver}(t_3.vk_A, S)$, and all other variables that describe t'_3 have the same value of the variables that describe t_3. Since $t_3\ id_k\ t_4$, we have that also t_4 enables the transition $(t_4, get\_verify\_sign(A, S), \mu_4)$ where μ_4 is the measure $\delta_{t'_4}$ where t'_4 is the state such that $t'_4.ver\_value_A = \mathtt{Ver}(t_4.vk_A, S)$, and all other variables that describe t'_4 have the same value of the variables that describe t_4. This implies that $t'_3\ id_k\ t'_4$ and thus $\delta_{t'_3}\ \mathcal{L}(id_k)\ \delta_{t'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.
- $a = ret\_verify\_sign(A, B)$ for some $A \in \mathbb{A}$ and $B \in \{T, F\}$: by definition of the action $ret\_verify\_sign(A, B)$, it follows that t_3 satisfies $t_3.ver\_value_A = B$. Moreover, it follows that $\mu_3 = \delta_{t'_3}$ where t'_3 is the state such that $t'_3.ver\_value_A = \bot$, and all other variables that describe t'_3 have the same value of the variables that describe t_3. Since $t_3\ id_k\ t_4$, we have that also t_4 satisfies $t_4.ver\_value_A = B$ and thus it enables the transition $(t_4, ret\_verify\_sign(A, B), \mu_4)$ where μ_4 is the measure $\delta_{t'_4}$

where t'_4 is the state such that $t'_4.ver\_value_A = \bot$, and all other variables that describe t'_4 have the same value of the variables that describe t_4. This implies that $t'_3\ id_k\ t'_4$ and thus $\delta_{t'_3}\ \mathcal{L}(id_k)\ \delta_{t'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.

– $a = used\_signatures(SN)$ for some $SN \subseteq$ Signature: by definition of $used\_signatures(SN)$, it follows that $\mu_3 = \delta_{t'_3}$ where t'_3 is the state such that $t'_3.used\_signatures = t_3.used\_signatures \cup SN$, and all other variables that describe t'_3 have the same value of the variables that describe t_3. Since $s_3\ id_k\ t_4$, we have that also t_4 enables the transition $(t_4, used\_signatures(SN), \mu_4)$ where μ_4 is the measure $\delta_{t'_4}$ where t'_4 is the state such that $t'_4.used\_signatures = t_4.used\_signatures \cup SN$, and all other variables that describe t'_4 have the same value of the variables that describe t_4. This implies that $t'_3\ id_k\ t'_4$ and thus $\delta_{t'_3}\ \mathcal{L}(id_k)\ \delta_{t'_4}$, that is $\mu_3\ \mathcal{L}(id_k)\ \mu_4$, as required.

The step condition is satisfied since for each action a, if t_3 enables a transition labelled by a that leads to μ_3, then we can find μ_4 such that $(t_4, a, \mu_4) \in D_k^4$ and $\mu_3\ \mathcal{L}(id_k)\ \mu_4$. □

It is straightforward to prove that $\mathcal{S}_k^3(\mathbb{A})|G_k$ is polynomially simulated by $\mathcal{S}_k^4(\mathbb{A})$:

Proposition 6.35. *Given $\mathcal{S}_k^3(\mathbb{A})$, let B_k be the set of states of $\mathcal{S}_k^3$ such that $t \in B_k$ if there exists $A \in \mathbb{A}$ such that $t.is\_not\_used_A = F$. Let G_k be the set $S_k^3 \setminus B_k$.*

For each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k^3(\mathbb{A})$,

$$\{(\mathcal{S}_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}_k^4(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. By Lemma 6.34, we have that for each $k \in \mathbb{N}$, $\mathcal{S}_k^3(\mathbb{A})|G_k \preceq \mathcal{S}_k^4(\mathbb{A})$. This implies that for each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k^3(\mathbb{A})$, $(\mathcal{S}_k^3(\mathbb{A})|G_k)||\mathcal{C}_k \preceq \mathcal{S}_k^4(\mathbb{A})||\mathcal{C}_k$. Finally, by Proposition 5.6, we have that $\{(\mathcal{S}_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}_k^4(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

The complete the chain of polynomially accurate simulations from $\mathcal{S}_k^3(\mathbb{A})$ to $\mathcal{S}_k^4(\mathbb{A})$ we need to prove:

Proposition 6.36. *Let S be a non-repeating unforgeable signature scheme and $\mathbb{A}$ be a set of agents. Let G_k be the set of states of $\mathcal{S}_k^3$ such that $t \in G_k$ if and only if for each $A \in \mathbb{A}$ $t.is\_not\_used_A \neq F$, that is, the generated ciphertext is fresh. Let B_k be $S_k^3 \setminus G_k$ (that is, states t' of $\mathcal{S}_k^3$ such that $t'.is\_not\_used_A = F$ for some $A \in \mathbb{A}$).*

For each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k^4(\mathbb{A})$, if there exists $q \in Poly$ such that for each action used_signatures(SN) of $\mathcal{C}_k$, $|SN| < q(k)$, then

$$\{\mathcal{S}_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{S}_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$$

Proof. Theorem 5.8 states that $\{\mathcal{S}_k^3(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}_k^3(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$ if and only if $\{B_k\}_{k\in\mathbb{N}}$ is negligible in $\{\mathcal{S}_k^3(\mathbb{A})\}_{k\in\mathbb{N}}$.

Suppose, for the sake of contradiction, that $\{\mathcal{S}_k^3(\mathbb{A})\}_{k\in\mathbb{N}}$ is not simulated by $\{\mathcal{S}_k^3(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$. This implies that $\{B_k\}_{k\in\mathbb{N}}$ is not negligible in $\{\mathcal{S}_k^3(\mathbb{A})\}_{k\in\mathbb{N}}$ and thus that there exists $c \in \mathbb{N}$, $p \in Poly$ such that for each $\bar{k} \in \mathbb{N}$ there exists $k > \bar{k}$ such the probability to reach states of B_k within $p(k)$ steps is at least k^{-c}, that is, the probability to reach states t such that $t.is\_not\_used_A = F$ for some $A \in \mathbb{A}$ within $p(k)$ steps is at least k^{-c}. By definition of the automaton $\mathcal{S}_k^3(\mathbb{A})$, it follows that $t.is\_not\_used_A$ can assume value F only as the effect of a transition that leaves from some state t' and that is labelled by action $get\_sign(A, M)$. This happens only when the signature value $\mathtt{Sig}(t'.sk_A, M)$ assigned to $t.sig\_value_A$ belongs to the set $US = t'.used\_signatures$. Within $p(k)$ steps, we can perform at most $p(k)$ transitions labelled by $used\_signatures(SN)$. Since by hypothesis we add at most $q(k)$ values to *used_signatures* each time we perform a transition labelled by $used\_signatures(SN)$, within $p(k)$ steps the set US has cardinality at most $p(k)q(k)$. This means that with probability at least k^{-c}, we have generated a signature that it is equal to some $s \in US$ with $|US| \leqslant p(k)q(k)$. Since the signature scheme S is non-repeating unforgeable, this contradicts the Definition 6.26 and thus $\{\mathcal{S}_k^3(\mathbb{A})\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}_k^3(\mathbb{A})|G_k\}_{k\in\mathbb{N}}$. This implies, by Theorem 5.10, that $\{\mathcal{S}_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{S}_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}}$. □

7

A Simple Case Study: the MAP1 Protocol

In this chapter we illustrate the use of polynomially accurate simulations via a simple case study that deals with the Mutual Authentication Protocol MAP1 of Bellare and Rogaway [22]. The protocol uses nonces to guarantee freshness and pseudorandom functions as message authentication tool. We describe the MAP1 protocol and the structure of our correctness proof. In particular, we prove the correctness result in two steps: in the first one we avoid the generation of repeated nonces; in the second one we impose that the adversary can not produce fresh valid message authentication codes.

7.1 The Protocol

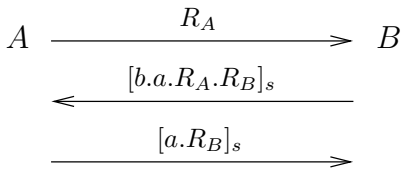

Fig. 7.1. MAP1 protocol.

Let $\{f_s\}_{s\in\{0,1\}^*}$ be a pseudorandom function, and let $[x]_s$ denote the message $(x, f_s(x))$ where $f_s(x)$ is the message authentication code of x with respect to s.

The MAP1 protocol is used to establish a mutual authentication between any two agents A and B among a set of agents $\mathbb{A}$ who share a key s. At the beginning, all agents share a pseudorandom function and a secret random element $s \in \{0,1\}^k$, where k is the security parameter. When agent A

wants to communicate with agent B, A sends to B a random challenge (a nonce) $R_A \in_R \{0,1\}^k$. B responds by making up a random challenge $R_B \in_R \{0,1\}^k$ and returning $[b.a.R_A.R_B]_s$, where a and b are descriptions of the identity of agents A and B, respectively. Then, A checks that the message received from B is of the right form and that it is correctly tagged as coming from B. If it is, A sends to B the message $[a.R_B]_s$ and accepts. B checks that the message from A is of the right form and that it is correctly tagged as coming from A. If it is, B accepts. Fig. 7.1 depicts how the MAP1 protocol works.

The definition of correctness proposed by Bellare and Rogaway in [22] is based on the concept of *matching conversation*. All agents communicate via an adversarial network E, controlled by a probabilistic polynomial time algorithm, that can block, delay and/or modify messages, and possibly create new messages. Two agents A and B have a matching conversation if the following conditions hold:

1. every message that A sends out, except possibly the last, is subsequently delivered to B, with the response to this message being returned to A as its own next message;
2. every message B receives was previously generated by A and each message that B sends out is subsequently delivered to A, with the response that this message generates being returned to B as its own next message.

The first condition states that when A (that plays as a sender or initiator agent) sends a message to B, the message is not modified or blocked by the adversary E (except for the last message) and the response of B is correctly delivered to A, without changing the messages order. The second condition is very similar to the first one, but it is based on B's point of view (B plays as a receiver or responder agent).

Given an adversary E (that does not know the secret key s shared by the agents), E breaks the MAP1 protocol if it completes a mutual authentication with some agent X persuading X that the other participant is another agent Y. This means that X completes the protocol without a matching conversation with Y. More formally, MAP1 is a secure mutual authentication protocol if

- for each pair of agents X and Y, if X and Y have a matching conversation, then both agents accept;

- for any probabilistic polynomial time adversary E, the probability that E induces an agent X to accept a communication with another agent Y without a matching conversation with Y is negligible.

E, during the attack, can play as initiator or responder, or even in both roles if it tries to break the MAP1 protocol interacting with several agents.

7.2 The Security Proof of Bellare-Rogaway

The original proof that MAP1 is a secure mutual authentication protocol can be found in Appendix A of [22]. The proof is split into two parts. First it is shown that the probability of breaking the protocol when the agents share a truly random function is negligible; then it is shown that an adversary E that successfully attacks the MAP1 protocol with a non-negligible probability can be turned into a distinguisher for a pseudorandom function.

The second step is rather standard in cryptography: the distinguisher is an algorithm that simulates the interaction between the adversary E and the agents and that queries the message authentication scheme whenever it simulates a real agent that computes a message authentication code. The distinguisher returns 1 whenever it successfully induces an agent A to accept without a matching conversation. The probability of returning 1 is then significantly different if the message authentication scheme is given by a truly random function or by a pseudorandom function. Though this construction is described in a semi-formal language, it is quite standard and widely accepted.

The first step is based on an explicit computation of the probability that the adversary induces acceptance without a matching conversation when the message authentication scheme is given by a truly random function. The short proof must be read with great attention because of the high number of potential pitfalls. It is a classical proof where we reason about global properties of computations by arguing back and forth about properties of different computational steps. These are typical arguments employed in correctness proofs for distributed and concurrent systems. In the specific case the argument is complicated further by the presence of probabilities. More or less the argument is a sequence of semi-formal statements about what messages are generated, in what order, who can have generated them (and with which probability), and whether messages can

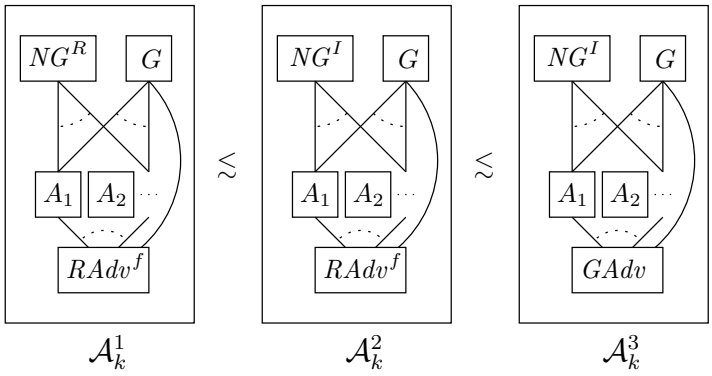

Fig. 7.2. The three levels of abstraction for MAP1.

be repeated (and with which probability). Arguments about uniqueness of nonces and unforgeability of message authentication codes are intermixed. Our suggestion is that the use of polynomially accurate simulations in this context can provide us with the same simplifications that the simulation method provided in the area of distributed systems (cf. [72]).

7.3 Our Correctness Proof

We now give an outline of the correctness proof of the MAP1 protocol based on polynomially accurate simulations. We describe the protocol at three levels of abstraction. The lowest level description consists of the actual agents that receive the secret s from a secret generator and receive nonces from a device that generates random numbers. The adversary is controlled by a generic probabilistic polynomial time algorithm. At the intermediate level nonces are generated by an ideal device that keeps track of what was distributed earlier, while at the highest level the adversary is purely nondeterministic and is not permitted to generate new message authentication codes without obtaining them from the agents. Figure 7.2 depicts the three levels of abstraction.

The highest level abstraction is similar in style to the Dolev-Yao model where we assume perfect cryptography, while the description in these three levels is similar in style to the game transformations proposed in [24,25,107]. The most abstract system can be shown easily not to exhibit any attack by employing ordinary well known techniques for purely nondeterministic systems. The novel element here is the use of simulation relations to relate the three levels.

We exhibit a polynomially accurate simulation for each pair of neighbor abstractions, use results on polynomially accurate simulation to argue that the probability of low level computations that do not have corresponding high level computations is negligible, and use the fact that at the highest level there are no attacks to deduce that at the lowest level the probability of attack is negligible. The crucial and interesting point is that at each level we can use general results or, when we prove the existence of the polynomially accurate simulation, the negation of the step condition is the negation of the key property of nonces or the definition of a successful forger for a message authentication scheme depending on the simulation relation we are analyzing.

Now we give a more detailed description of the three levels of the abstraction. The lowest level, depicted on the left of Figure 7.2, consists of several automata, each one parameterized by a security parameter k (we do not add such parameter to the automata names for clarity). The automaton G is a secret generator that generates and provides the agents with a secret s that is used as the key of the message authentication scheme of MAP1 protocol. The automaton NG^R models a real nonce generator. Whenever an agent needs a nonce, it sends a request to NG^R and obtains a random value taken from $\{0,1\}^k$ as answer. The set $\{A_1, A_2, \dots\}$ is a numerable set of automata that describe end-points of sessions of the protocol. That is, each automaton A_i corresponds to some oracle $\Pi^t_{X,Y}$ of [22], where oracle $\Pi^t_{X,Y}$ describes the participant X trying to authenticate participant Y in session t, where t is different for each authentication attempt. Communication between agents and secret and nonce generators is private, while communication between agents is performed using a network that is controlled by the adversary $RAdv_f$. The network keeps a history variable that contains all previous messages sent and received by agents, which is used to select the next action to perform (e.g., delivering messages, casting new messages, blocking messages, ...). The choices of the network should be computable in probabilistic polynomial time. For this reason, the adversary $RAdv_f$ is parameterized by a probabilistic polynomial time function f, so that the transition enabled from a state s is $f(s)$.

The intermediate level, depicted in the middle of Figure 7.2, differs from the lowest level only in the nonce generator automaton. NG^I models an ideal nonce generator that ensures that nonces are never repeated. This implies that unicity of nonces chosen by agents is guaranteed by definition.

$G_k(\mathbb{A})$

Signature:

Output:

$secret((X, Y, t), s)$, $s \in \{0,1\}^k$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

$secret\_for\_adv(s)$, $s \in \{0,1\}^k$

State:

$value \in \{0,1\}^k$, initially $v \in_R \{0,1\}^k$

Transitions:

Output $secret\_for\_adv(s)$
Precondition:
$s = value$
Effect:
none

Output $secret((X, Y, t), s)$
Precondition:
$s = value$
Effect:
none

Fig. 7.3. The Secret Generator, G_k

The highest level, depicted in the right of Figure 7.2, differs from the intermediate level only in the automaton that controls the network. The new adversary, denoted by *GAdv*, is a nondeterministic automaton that is permitted to perform any action except for casting new message authentication codes without obtaining them from the agents. More precisely, we define a function *Not_Bad* that, given a secret s and a history *history*, returns the set of messages where all subparts that are tagged correctly with a message authentication code relative to s are taken from *history*. That is, no new correct tag is cast. Then we require *GAdv* to generate only those messages that are in the outcome of function *Not_Bad*. This implies that unforgeability of message authentication scheme is warranted by definition.

Automata Specification

We now provide the automata that describe the participants and adversaries of the MAP1 protocol.

Figure 7.3 depicts the secret generator G_k. It starts with a secret s, chosen randomly in $\{0,1\}^k$, which is then sent to all agents via actions of the form $secret(X, Y, t)$. The secret is sent also to the adversary via action *secret_for_adv*, though the real adversary will discard the value received.

Real Nonce Generator $NG_k^R(\mathbb{A})$

Signature:

Input:

$get\_nonce(X, Y, t)$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

Output:

$ret\_nonce((X, Y, t), n)$, $n \in \{0,1\}^k$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

State:

$value_{X,Y}^t \in \{0,1\}^k \cup \{\perp\}$, initially $\perp$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

Transitions:

Input $get\_nonce(X, Y, t)$

Effect:

$value_{X,Y}^t := v$ where $v \in_R \{0,1\}^k$

Output $ret\_nonce((X, Y, t), n)$

Precondition:

$n = value_{X,Y}^t$

Effect:

$value_{X,Y}^t := \perp$

Fig. 7.4. The Real Nonce Generator, $NG_k^R(\mathbb{A})$

The value of the secret will be used by the good adversary to prevent the generation of forged message authentication codes.

Figure 7.4 shows the real nonce generators $NG_k^R(\mathbb{A})$. When the automaton receives an input *get_nonce*, it chooses a new nonce and assigns it to a local variable to be used by the corresponding *ret_nonce* action. It is essentially the same of the generic nonce generator NG of Figure 6.1 where each component (action or state variable) is parameterized on X, Y and t instead of A. $NG_k^R(\mathbb{A})$ is the same automaton of $NG_k(\mathbb{A}')$ of Section 6.1, where $\mathbb{A}'$ is the set of agent's identities $\{(X, Y, t) \mid X, Y \in \mathbb{A}, t \in \mathbb{N}\}$.

Figure 7.5 shows the ideal nonce generators $NG_k^I(\mathbb{A})$. When the automaton receives an input *get_nonce*, it chooses a new, fresh nonce and assigns it to a local variable to be used by the corresponding *ret_nonce* action. We are sure that the chosen nonce is fresh since it is taken from the set *fresh_nonces* that contains all nonces that are not chosen yet. It is the same as the generic nonce generator $NG_k^2(\mathbb{A}')$ of Section 6.1.1 where $\mathbb{A}'$ is the set of agent's identities $\{(X, Y, t) \mid X, Y \in \mathbb{A}, t \in \mathbb{N}\}$.

Figures from 7.6 to 7.8 depict the $MAP1(X, Y, t)$ automaton that describes an agent X trying to authenticate to another agent Y in session t.

Ideal Nonce Generator $NG_k^I(\mathbb{A})$

Signature:

Input:

$get\_nonce(X, Y, t)$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

Output:

$ret\_nonce((X, Y, t), n)$, $n \in \{0,1\}^k$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

State:

$value^t_{X,Y}$	$\in \{0,1\}^k \cup \{\perp\}$,	initially $\perp$,	$X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$is\_fresh^t_{X,Y}$	$\in \{T, F, \perp\}$,	initially $\perp$,	$X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$fresh\_nonces$	$\in \{0,1\}^k$,	initially $\{0,1\}^k$	

Transitions:

Input $get\_nonce(X, Y, t)$

Effect:

$value^t_{X,Y} := v$ where $v \in_R fresh\_nonces$

$is\_fresh^t_{X,Y} := T$

$fresh\_nonces := fresh\_nonces \setminus \{v\}$

Output $ret\_nonce((X, Y, t), n)$

Precondition:

$n = value^t_{X,Y}$

Effect:

$value^t_{X,Y} := \perp$

Fig. 7.5. The Ideal Nonce Generator $NG_k^I(\mathbb{A})$

Agent X may play either as a sender or as a receiver, and the role of X is determined by the first input received by the automaton: if the first input is a *start_init* action, then X acts as sender (or initiator) agent; if the first input is a *receive1* action, then X acts as a receiver agent. The state of the automaton has two variables R_X, R_Y that store local copies of the nonces of X and Y, respectively; a variable *secret* that stores the secret key of the message authentication scheme; a variable *accept* that assumes value true when the automaton accepts the authentication; a *nonce_requested* variables that is used to remember when a nonce request is pending; and a program counter *pc* that keeps track of the current position in the flow of the MAP1 protocol. The automaton switches to an error state ($pc =$ `error`) as soon as an unexpected input or a badly formatted message is received. From the error state the automaton does not perform any output action and ignores the effects of all input actions. The sequence of actions follows the MAP1 protocol as proposed in [22].

$MAP1^t_{X,Y_k}$

Signature:

Input:

$start\_init(X, Y, t)$
$receive1((X, Y, t), m)$, $m \in \{0,1\}^k$
$receive2((X, Y, t), m)$, $m \in \{0,1\}^{5k}$
$receive3((X, Y, t), m)$, $m \in \{0,1\}^{3k}$
$ret\_nonce((X, Y, t), n)$, $n \in \{0,1\}^k$
$secret((X, Y, t), s)$, $s \in \{0,1\}^k$

Output:

$get\_nonce(X, Y, t)$
$send1((X, Y, t), m)$, $m \in \{0,1\}^k$
$send2((X, Y, t), m)$, $m \in \{0,1\}^{5k}$
$send3((X, Y, t), m)$, $m \in \{0,1\}^{3k}$

State:

R_X, R_Y	$\in \{0,1\}^k \cup \{\perp\}$,	initially $\perp$
$secret$	$\in \{0,1\}^k \cup \{\perp\}$,	initially $\perp$
pc	$\in \{$`error`, `end`, `wait1`, `wait2`, `wait3`, `send1`, `send2`, `send3`$\}$,	initially `wait1`
$nonce\_requested$	$\in \{T, F\}$,	initially F
$accept$	$\in \{T, F\}$,	initially F

Transitions:

Input $secret((X, Y, t), s)$
Effect:
$secret := s$

Output $get\_nonce(X, Y, t)$
Precondition:
$pc \in \{$`send1`, `send2`$\} \wedge R_X = \perp \wedge \neg nonce\_requested$
Effect:
$nonce\_requested := T$

Input $ret\_nonce((X, Y, t), n)$
Effect:
if $\neg nonce\_requested$ then
$pc :=$ `error`
else
$R_X := n$
$nonce\_requested := F$
fi

Fig. 7.6. The MAP1 Agent, $MAP1^t_{X,Y_k}$, Part I

Transitions:

Input $start\_init(X, Y, t)$
  Effect:
    if $pc = \texttt{wait1}$ then
      $pc := \texttt{send1}$
    else
      $pc := \texttt{error}$
    fi

Output $send1((X, Y, t), m)$
  Precondition:
    $pc = \texttt{send1} \wedge m = R_X \neq \perp \wedge secret \neq \perp$
  Effect:
    $pc := \texttt{wait2}$

Input $receive2((X, Y, t), m)$
  Effect:
    if $pc = \texttt{wait2} \wedge$
      $\exists r \in \{0,1\}^k . m = [y.x.R_X.r]_{secret}$ then
      $R_Y := r$
      $pc := \texttt{send3}$
    else
      $pc := \texttt{error}$
    fi

Output $send3((X, Y, t), m)$
  Precondition:
    $pc = \texttt{send3} \wedge m = [x.R_Y]_{secret}$
  Effect:
    $pc := \texttt{end}$
    $accept := T$

Fig. 7.7. The MAP1 Agent, $MAP1^t_{X,Y\,k}$, Part II

Figures 7.9 and 7.10 show the good adversary. First of all the adversary waits for the secret from the secret generator G. Then it alternates internal generation of messages according to function Not_Bad, which guarantees no forging of signatures, and delivery of messages to agents. All inputs from the agents are simply added to the history.

Figures 7.11 and 7.12 depict the real adversary $RAdv^f_k(\mathbb{A})$. Also in this case the adversary waits for the secret from G, but the actual value of the secret is discarded. After that, the adversary behaves sequentially: it generates internally a new message, including the destination, according to a probabilistic polynomial time function f, it forwards the generated

Transitions:

Input $receive1((X, Y, t), m)$
  Effect:
    if $pc = \texttt{wait1}$ then
      $pc := \texttt{send2}$
      $R_Y := m$
    else
      $pc := \texttt{error}$
    fi

Output $send2((X, Y, t), m)$
  Precondition:
    $pc = \texttt{send2} \wedge R_X \neq \perp \wedge secret \neq \perp \wedge$
      $m = [x.y.R_Y.R_X]_{secret}$
  Effect:
    $pc := \texttt{wait3}$

Input $receive3((X, Y, t), m)$
  Effect:
    if $pc = \texttt{wait3} \wedge m = [y.R_X]_{secret}$ then
      $pc := \texttt{end}$
      $accept := T$
    else
      $pc := \texttt{error}$
    fi

Fig. 7.8. The MAP1 Agent, ${MAP1}^t_{X,Y\,k}$, Part III

message to the chosen destination, and, if specified in the MAP1 protocol, waits for the answer. Then the cycle is repeated. The correctness of the cycle is guaranteed by a boolean variable *enable_action_creation*, which is true only when a new message can be generated.

Some Considerations on the Automata

We have been very careful in the definition of the real adversary, and in particular we have ensured that its behavior is sequential. One reason for doing this is that in the definition of correct message authentication schemes the forger is a sequential process, and thus, if we want the negation of the step condition to become the definition of a forger, we need to make sure that we will deal with a sequential process.

It would be desirable to be able to reason with a more general, non-sequential, adversary, but unfortunately it is not possible to do it in the

$GAdv_k(\mathbb{A})$

Signature:

Input:

$secret\_for\_adv(s)$, $s \in \{0,1\}^k$
$send1((X,Y,t),m)$, $m \in \{0,1\}^k$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$send2((X,Y,t),m)$, $m \in \{0,1\}^{5k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$send3((X,Y,t),m)$, $m \in \{0,1\}^{3k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

Output:

$start\_init(X,Y,t)$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$receive1((X,Y,t),m)$, $m \in \{0,1\}^k$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$receive2((X,Y,t),m)$, $m \in \{0,1\}^{5k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$
$receive3((X,Y,t),m)$, $m \in \{0,1\}^{3k}$, $X, Y \in \mathbb{A}$, $t \in \mathbb{N}$

Internal:

$create\_message$

State:

$history \in \text{Sequences}(Actions(\mathbb{A}) \times M)$, initially $\emptyset$,
$message \in M \cup \{\perp\}$, initially $\perp$
$secret \in \{0,1\}^k \cup \{\perp\}$, initially $\perp$
where $M = \{0,1\}^k \cup \{0,1\}^{3k} \cup \{0,1\}^{5k}$

Transitions:

Input $secret\_for\_adv(s)$
Effect:
$secret := s$

Internal $create\_message$
Precondition:
$secret \neq \perp$
Effect:
$message := m \in Not\_Bad(secret, history)$

Fig. 7.9. The Good Adversary, $GAdv_k(\mathbb{A})$, Part I

current setting. Suppose we allow the real adversary to generate messages according to f in any order, without necessarily waiting for the answers from the agents. Then we can build a scheduler, and an appropriate function f, where the adversary initializes k sessions of the MAP1 protocol, say $S_1, \ldots, S_k$, and make sure that session S_i responds only if the i^{th} bit of the secret is 1. In this way the adversary knows the value of the secret and it is therefore able to sign messages. In other words we can resolve nondeterminism to create a covert channel [66] that communicates the secret to the adversary. Solutions to this problem are studied already in the

Transitions:

Output $start\_init(X, Y, t)$
  Precondition:
    $secret \neq \perp$
  Effect:
    $history := history \vdash (start\_init, (X, Y, t))$

Input $send1((X, Y, t), m)$
  Effect:
    $history := history \vdash (send1, ((X, Y, t), m))$

Output $receive1((X, Y, t), m)$
  Precondition:
    $m = message$
  Effect:
    $history := history \vdash (receive1, ((X, Y, t), m))$

Input $send2((X, Y, t), m)$
  Effect:
    $history := history \vdash (send2, ((X, Y, t), m))$

Output $receive2((X, Y, t), m)$
  Precondition:
    $m = message$
  Effect:
    $history := history \vdash (receive2, ((X, Y, t), m))$

Input $send3((X, Y, t), m)$
  Effect:
    $history := history \vdash (send3, ((X, Y, t), m))$

Output $receive3((X, Y, t), m)$
  Precondition:
    $m = message$
  Effect:
    $history := history \vdash (receive3, ((X, Y, t), m))$

Fig. 7.10. The Good Adversary, $GAdv_k(\mathbb{A})$, Part II

literature [34, 86, 97] and it is worth investigating how polynomially accurate simulations can be adapted to such frameworks. Here we have chosen to avoid restrictions to the schedulers to keep the treatment simple and to show that it is also possible to remove dangerous nondeterminism and work with unrestricted schedulers.

Another observation about our definition of the adversaries is that we have separated message generation from message delivery. We could easily

$RAdv_k^f(\mathbb{A})$

Signature:

Input:

$secret\_for\_adv(s)$, $s \in \{0,1\}^k$
$send1((X,Y,t),m)$, $m \in \{0,1\}^k$, $X,Y \in \mathbb{A}$, $t \in \mathbb{N}$
$send2((X,Y,t),m)$, $m \in \{0,1\}^{5k}$, $X,Y \in \mathbb{A}$, $t \in \mathbb{N}$
$send3((X,Y,t),m)$, $m \in \{0,1\}^{3k}$, $X,Y \in \mathbb{A}$, $t \in \mathbb{N}$

Output:

$start\_init(X,Y,t)$, $X,Y \in \mathbb{A}$, $t \in \mathbb{N}$
$receive1((X,Y,t),m)$, $m \in \{0,1\}^k$, $X,Y \in \mathbb{A}$, $t \in \mathbb{N}$
$receive2((X,Y,t),m)$, $m \in \{0,1\}^{5k}$, $X,Y \in \mathbb{A}$, $t \in \mathbb{N}$
$receive3((X,Y,t),m)$, $m \in \{0,1\}^{3k}$, $X,Y \in \mathbb{A}$, $t \in \mathbb{N}$

Internal:

$create\_message$

State:

$history$	$\in$ Sequences$(Actions(\mathbb{A}) \times M)$,	initially $\emptyset$,
$action$	$\in Actions(\mathbb{A}) \cup \{\perp\}$,	initially $\perp$
$enable\_action\_creation$	$\in \{T,F\}$,	initially T
$message$	$\in M \cup \{\perp\}$,	initially $\perp$
$run\_enabled$	$\in \{T,F\}$,	initially F

where $M = \{0,1\}^k \cup \{0,1\}^{3k} \cup \{0,1\}^{5k}$

Transitions:

Input $secret\_for\_adv(s)$
Effect:
$run\_enabled := T$

Internal $create\_action$
Precondition:
$run\_enabled \wedge enable\_action\_creation$
Effect:
$enable\_action\_creation := F$
$(action, message) := f(history)$

Fig. 7.11. The Real Adversary, $RAdv_k^f(\mathbb{A})$, Part I

avoid this separation, but in this case we would be forced to use probabilistic automata with generative transitions [104] to describe the real adversary, which have a more complex theory. Once again, our choice is to keep the presentation simple and focus on the ideas behind polynomially accurate simulations.

Transitions:

Output $start\_init(X, Y, t)$
  Precondition:
    $action = start\_init(X, Y, t)$
  Effect:
    $action := \perp$
    $history := history \vdash (start\_init, ((X, Y, t), m))$

Input $send1((X, Y, t), m)$
  Effect:
    $history := history \vdash (send1, ((X, Y, t), m))$
    $enable\_action\_creation := T$

Output $receive1((X, Y, t), m)$
  Precondition:
    $= receive1^{t}_{X,Y} \wedge m = message$
  Effect:
    $history := history \vdash (receive1, ((X, Y, t), m))$
    $action := \perp$

Input $send2((X, Y, t), m)$
  Effect:
    $history := history \vdash (send2, ((X, Y, t), m))$
    $enable\_action\_creation := T$

Output $receive2((X, Y, t), m)$
  Precondition:
    $= receive2^{t}_{X,Y} \wedge m = message$
  Effect:
    $history := history \vdash (receive2, ((X, Y, t), m))$
    $action := \perp$

Input $send3((X, Y, t), m)$
  Effect:
    $history := history \vdash (send3, ((X, Y, t), m))$
    $enable\_action\_creation := T$

Output $receive3((X, Y, t), m)$
  Precondition:
    $action = receive3^{t}_{X,Y} \wedge m = message$
  Effect:
    $history := history \vdash (receive3, ((X, Y, t), m))$
    $enable\_action\_creation := T$
    $action := \perp$

Fig. 7.12. The Real Adversary, $RAdv^{f}_{k}(\mathbb{A})$, Part II

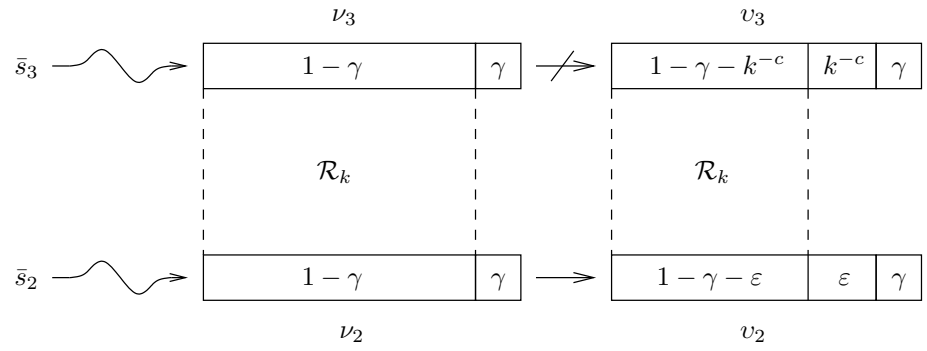

Fig. 7.13. Graphical representation of the step $\nu_2 \longrightarrow \upsilon_2$.

The Security Proof

Now we are ready to prove the correctness of the MAP1 protocol. For the first step, there is nothing to prove, since we can use the results of the Section 6.1.1. For the second step of the proof, we apply directly the definition of the polynomially accurate simulation, showing how the negation of the step condition leads to the negation of the main property of the underlying cryptographic primitive (in this case, the security of the message authentication scheme).

Proposition 7.1. $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^3\}_{k\in\mathbb{N}}$

Proof. Define the relation family $\{\mathcal{R}_k\}_{k\in\mathbb{N}}$ as the family of identity relations.

The condition on start states is trivially true. Suppose that step condition does not hold. This means that there exist $c \in \mathbb{N}$, $p \in Poly$ such that for all $\bar{k} \in \mathbb{N}$ there exist $k > \bar{k}$, ν_2, ν_3, γ and υ_2 such that ν_2 is reached by $\mathcal{A}_k^2$ within $p(k)$ steps, $\nu_2\ \mathcal{L}(\mathcal{R}_k, \gamma)\ \nu_3$, $\nu_2 \longrightarrow \upsilon_2$ and there is no υ_3 such that $\nu_3 \longrightarrow \upsilon_3$ and $\upsilon_2\ \mathcal{L}(\mathcal{R}_k, \gamma + k^{-c})\ \upsilon_3$.

Figure 7.13 gives a graphical representation of the transition $\nu_2 \longrightarrow \upsilon_2$ where ε represents the part of the transition that can not be emulated from ν_3; hence $\varepsilon > k^{-c}$.

By definition of $\mathcal{A}_k^2$ and $\mathcal{A}_k^3$, it follows that $\mathcal{A}_k^2$ is the composed automaton $G_k||NG_k^I(\mathbb{A})||MAP1^t_{X,Y_k}||RAdv_k^f(\mathbb{A})$ and $\mathcal{A}_k^3$ is the composed automaton $G_k||NG_k^I(\mathbb{A})||MAP1^t_{X,Y_k}||GAdv_k(\mathbb{A})$. This implies that the transitions of $\mathcal{A}_k^2$ that can not be simulated by $\mathcal{A}_k^3$ are only the ones of $RAdv_k^f(\mathbb{A})$ (since each transition of other automata is easily matched choosing the

same transition). By definition of $RAdv_k^f(\mathbb{A})$ and $GAdv_k(\mathbb{A})$, it follows that $RAdv_k^f(\mathbb{A})$ might generate a fresh valid message authentication code while $GAdv$ may perform any action except for casting new message authentication codes without obtaining them from the agents, and this implies that the ε fraction above corresponds to generation of new message authentication codes. That is, the right side of Figure 7.13 represents a computation of $\mathcal{A}_k^2$ of length at most $p(k)+1$ where a new message authentication code is generated with probability at least k^{-c}. Summing up, there exist $c \in \mathbb{N}$, $p \in Poly$ such that for all $\bar{k} \in \mathbb{N}$ there exist $k > \bar{k}$ and υ_2 such that υ_2 has length at most $p(k)+1$ and the probability to generate new message authentication codes in υ_2 is at least k^{-c}. This contradicts the fact that the message authentication scheme used by MAP1 is a secure message authentication scheme since the statement above is the negation of the negligible probability of successful forger after using a polynomial $p'(k)$ to denote $p(k)+1$ and observing that at most $p'(k)$ message authentication codes are requested within $p'(k)$ steps. □

As we can see, in the above proof the negation of the step condition leads to a negation of properties of underlying cryptographic primitives. In fact we have negated the security of the message authentication scheme used to define the MAP1 protocol.

Now we are ready to relate the executions of the concrete level to executions of the abstract model. In fact, denote by $MAP1_k$ the composition of automata ${MAP1}^t_{X,Y_k}$ with $X, Y \in \mathbb{A}$ and $t \in \mathbb{N}$; moreover, consider the following naming:

- $\mathcal{B}_k^1$ for $G_k(\mathbb{A})||NG_k(\mathbb{A})||MAP1_k||RAdv_k^f(\mathbb{A})$,
- $\mathcal{B}_k^2$ for $G_k(\mathbb{A})||NG_k^1(\mathbb{A})||MAP1_k||RAdv_k^f(\mathbb{A})$,
- $\mathcal{B}_k^3$ for $G_k(\mathbb{A})||(NG_k^1(\mathbb{A})|G_k)||MAP1_k||RAdv_k^f(\mathbb{A})$,
- $\mathcal{B}_k^4$ for $G_k(\mathbb{A})||NG_k^2(\mathbb{A})||MAP1_k||RAdv_k^f(\mathbb{A})$, and
- $\mathcal{B}_k^5$ for $G_k(\mathbb{A})||NG_k^2(\mathbb{A})||MAP1_k||GAdv_k(\mathbb{A})$.

We have the following chain of simulations:

- by Proposition 6.2, we have that $\{\mathcal{B}_k^1\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k^2\}_{k\in\mathbb{N}}$;
- Proposition 6.5 implies that $\{\mathcal{B}_k^2\}_{\lesssim_s}\{\mathcal{B}_k^3\}_{k\in\mathbb{N}}$;
- by Proposition 6.4 it follows that $\{\mathcal{B}_k^3\}_{k\in\mathbb{N}} \lesssim_s \{G_k(\mathcal{B}_k^4\}_{k\in\mathbb{N}}$;
- Proposition 7.1 implies that $\{\mathcal{B}_k^4\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k^5\}_{k\in\mathbb{N}}$.

The above results imply that we have a chain of simulations from $\{\mathcal{B}_k^1\}_{k\in\mathbb{N}}$ to $\{\mathcal{B}_k^5\}_{k\in\mathbb{N}}$ and thus, by Theorem 3.16, we know that for each $c \in \mathbb{N}$ e

$p \in Poly$, there exists $\bar{k} \in \mathbb{N}$ such that for each $k > \bar{k}$, if ν_1 is a probability measure reached within $p(k)$ steps in $\{\mathcal{B}_k^1\}_{k\in\mathbb{N}}$, then there exists a probability measure ν_5 such that ν_5 is reached within $p(k)$ steps in $\{\mathcal{B}_k^5\}_{k\in\mathbb{N}}$ and $\nu_1\ \mathcal{L}(\mathcal{R}_k, p(k)k^{-c})\ \nu_5$ where $\{\mathcal{R}_k\}_{k\in\mathbb{N}}$ is the family of relations $\mathcal{R}_k$ such that each relation $\mathcal{R}_k$ is the composition of the relations that justify the chain of simulations.

If we analyze the relations we use in the chain of simulations, we find that they are always the identity relation, except for the first relation that is the identity relation on common variables. This implies that also the composition of such relations is the identity relation (on common variables) and thus we know that each execution of the real representation of the protocol is also an execution of the ideal automaton, except for a negligible set of executions. This means that the probability of an attack is negligible.

8

A Case Study: the Dolev-Yao Soundness

In this chapter we apply our polynomially accurate simulations to the soundness result of Cortier and Warinschi [41] and we show how probabilistic automata and approximated simulations can be used to provide a more rigorous proof of the soundness result, highlighting the technical details that are considered in an informal way.

We start recalling the protocol syntax, the execution models and the soundness results of [41]. Finally, in Section 8.4, we present our analysis of the proof, showing how probabilistic automata and polynomially accurate simulations permit to point out the details that are usually treated only informally.

8.1 Protocol Syntax

The cryptographic protocols considered in [41] are specified in a language similar to the one of [101]. In such language, protocols are described by ordered set of roles: given the set of participants, the behavior of the participant p is specified by a role R_p. Each role is a list of steps performed by the participant and each step is represented using two message patterns (m_l, m_r). Each participant p knows which is the next step it should perform, so when it receives a message m, it tries to match m with the left pattern m_l of the next step. If m and m_l match, then p replies with a message m' that is obtained from m_r by instantiating variables of m_r with corresponding values in m, previously received messages and p's initial knowledge. Start point of the protocol is denoted by the pattern init that can be used only once and as the left pattern of a step. In particular, such step is the first step of the role that specifies the initiator of the protocol. Similarly, pattern

stop is used to denote the end of the protocol and it can be used only as the right pattern of a step.

The language is based on an algebraic signature with the following sorts: ID, SKey, VKey, EKey, DKey, Nonce, Label, Ciphertext, Signature, Pair, and Term. ID is used for agent identities; SKey, VKey, EKey, and DKey contain keys for signing, verifying, encryption and decryption, respectively; Nonce for nonces; Label for labels; Ciphertext for ciphertexts; Signature for signatures, Pair for pairs and Term is a supersort containing all other sorts.

Nine operations are defined on sorts:

- $\mathsf{ek} : \mathsf{ID} \to \mathsf{EKey}$
- $\mathsf{dk} : \mathsf{ID} \to \mathsf{DKey}$
- $\mathsf{sk} : \mathsf{ID} \to \mathsf{SKey}$
- $\mathsf{vk} : \mathsf{ID} \to \mathsf{VKey}$
- $\mathsf{ag} : \mathbb{N} \to \mathsf{Label}$
- $\mathsf{adv} : \mathbb{N} \to \mathsf{Label}$
- $\langle \_, \_ \rangle : \mathsf{Term} \times \mathsf{Term} \to \mathsf{Pair}$
- $\{\_\}^{\_}_{\_} : \mathsf{EKey} \times \mathsf{Term} \times \mathsf{Label} \to \mathsf{Ciphertext}$
- $[\_]^{\_}_{\_} : \mathsf{SKey} \times \mathsf{Term} \times \mathsf{Label} \to \mathsf{Signature}$

The first four operations (ek, dk, sk, and vk) associate the corresponding key to the given agent identity; ag and adv are used to generate labels. A label represents the randomness used during the encryption/signature of a value and we define two functions to distinguish between the randomness used by agents (represented by ag function) from the one used by the adversary (represented by adv). The last three operations are pairing, public key encryption and signing and are used to build up composed terms.

The protocols are specified using the algebra of term constructed over the above signature from a set X of sorted variables, that is, $\mathsf{X} = \mathsf{X}.n \cup \mathsf{X}.a \cup \mathsf{X}.c \cup \mathsf{X}.s \cup \mathsf{X}.l$ where $\mathsf{X}.n$, $\mathsf{X}.a$, $\mathsf{X}.c$, $\mathsf{X}.s$, and $\mathsf{X}.l$ denote the set of variables of sort nonce, agent, ciphertext, signature, and labels, respectively. If $p \in \mathbb{N}$ is the number of the protocol participants, then the set $\mathsf{X}.a$ is fixed to be $\mathsf{X}.a = \{A_1, \ldots, A_p\}$. The set of nonce variables is partitioned with respect to the agent that generates them, that is, $\mathsf{X}.n = \bigcup_{A \in \mathsf{X}.a} \mathsf{X}_n(A)$ where $\mathsf{X}_n(A) = \{X_A^j \mid j \in \mathbb{N}\}$ and X_A^j denotes the jth nonce variable of agent A.

We define the size of a term t inductively as follows:

$$\text{size}(t) = \begin{cases} 1 & \text{if } t = N \in \mathsf{X}.n \\ \text{size}(t_1) + 1 & \text{if } t = \{t_1\}_{t_3}^{t_2} \\ \text{size}(t_1) + 1 & \text{if } t = [t_1]_{t_3}^{t_2} \\ \text{size}(t_1) + \text{size}(t_2) + 1 & \text{if } t = \langle t_1, t_2 \rangle \end{cases}$$

8.1.1 Roles and Protocols

The messages sent by protocol participants are specified using terms in $\mathtt{T}_\Sigma(\mathsf{X})$, the free algebra generated by X over the signature Σ. A *role* describes the individual behavior of each protocol participant specifying the sequence of message receptions and transmissions. Usually, a participant receives a message as input and then generates a reply as output. In some cases, the participant can generate an output without having received an input previously. In this case, the special term init $\notin \mathtt{T}_\Sigma(\mathsf{X})$ is used instead of the expected message reception. Similarly, the special term stop $\notin \mathtt{T}_\Sigma(\mathsf{X})$ is used to specify that a message reception is not followed by a message transmission.

A protocol generally involves several participants, each one with a different behavior. If the protocol involves k participants, then it is called a k-party protocol and it is defined by k (possibly different) roles.

Definition 8.1. *Let* Roles *be the set* $((\{\text{init}\} \cup \mathtt{T}_\Sigma(\mathsf{X})) \times (\mathtt{T}_\Sigma(\mathsf{X}) \cup \{\text{stop}\}))^*$.
A k-party protocol *is a mapping* $\Pi \colon \{1, \ldots, k\} \to \text{Roles}$.

The above definition of k-party protocol is very general and does not distinguish between protocols that can be implemented from non-implementable protocols. For example, it is possible to define a protocol where three participants A_1, A_2, and A_3 are involved: A_1 generates a message m that is then encrypted with the encryption key of A_2 and the resulting ciphertext c is sent to A_3. A_3 receives c, extracts m and replies with m signed with the signing key of A_2. Such protocol is obviously non-implementable, since A_3 can not decrypt the c to obtain m and furthermore it can not generate the signature of m under the signing key of A_2, since it is a standard assumption that each participant is honest (until it is corrupted by the adversary) and it does not know private keys of other participants.

Before defining executable protocols, that is protocols that can be implemented and that satisfy standard assumptions, we define the initial knowledge of a participant. The initial knowledge contains the private keys of the

$$\frac{}{S \vdash_a m}\ m \in S \qquad \frac{}{S \vdash_a b, \mathsf{ek}(b), \mathsf{vk}(b)}\ b \in \mathsf{X}.a \qquad \text{Initial knowledge}$$

$$\frac{S \vdash_a m_1 \quad S \vdash_a m_2}{S \vdash_a \langle m_1, m_2 \rangle} \qquad \frac{S \vdash_a \langle m_1, m_2 \rangle}{S \vdash_a m_i}\ i \in \{1,2\} \qquad \text{Pairing and unpairing}$$

$$\frac{S \vdash_a \mathsf{ek}(b) \quad S \vdash_a m}{S \vdash_a \{m\}^{\mathsf{ag}(i)}_{\mathsf{ek}(b)}}\ i \in \mathbb{N} \qquad \frac{S \vdash_a \{m\}^{l}_{\mathsf{ek}(b)} \quad S \vdash_a \mathsf{dk}(b)}{S \vdash_a m} \qquad \text{Encryption and decryption}$$

$$\frac{S \vdash_a \mathsf{sk}(b) \quad S \vdash_a m}{S \vdash_a [m]^{\mathsf{ag}(i)}_{\mathsf{sk}(b)}}\ i \in \mathbb{N} \qquad \text{Signature}$$

Fig. 8.1. Deduction rules for agents

participant (decryption and signing key) and all nonces it will use during the protocol run.

Definition 8.2. *If A is a variable, or constants of sort agent, we define the* knowledge *of A, denoted by* $\mathbf{kn}(A)$*, as* $\mathbf{kn}(A) = \{\mathsf{dk}(A), \mathsf{sk}(A)\} \cup \mathsf{X}_n(A)$.

The fact that initial knowledge contains cryptographic private keys permits to do not consider the problem of generating and distributing cryptographic keys. This assumption is widely used when a new protocol (that does not try to resolve the problem of generating and distributing cryptographic keys) is proposed (see, for example, [63, 71, 87]).

Now we can define when a protocol is executable, that is, it is implementable. Informally, a protocol is executable if response messages can be deduced from the initial knowledge and previously received messages. Moreover, it is required that a participant is actually able to open encryptions and to verify signatures as well as the participant is able to perform equality tests implicitly defined by the repetitions of variables.

The following definition of executable protocol is different from the original definition provided in [41]: we have expanded it to make it more readable and more complete.

Definition 8.3. *Let* $\Pi\colon \{1, \ldots, n\} \to$ Roles *be an n-party protocol. Suppose that for each $j \in \{1, \ldots, n\}$, the role of the protocol associated with participant j is* $\Pi(j) = ((l^j_1, r^j_1), \ldots, (l^j_{k_j}, r^j_{k_j}))$.

Let $\vdash_a$ *be the deduction defined by rules given in Figure 8.1.*

Π is an executable protocol *if following conditions hold:*

1. *the protocol has the* executable decryption property*: for all $A_i \in \mathsf{X}.a$ the only encryption keys that are contained in terms $l^i_1, \ldots, l^i_{k_i}$ are $\mathsf{ek}(A_i)$;*

2. *the protocol has the* executable verification property*: for each* $1 \leqslant i \leqslant n$*, each* $1 \leqslant k \leqslant k_i$*, and each* $A \in \mathsf{X}.a$ *we require that whenever* l^i_k *contains a signature* $[t]^{\lambda}_{\mathsf{sk}(A)}$ *for some term* $t \in \mathtt{T}_{\Sigma}(\mathsf{X})$*,* $\lambda \in \mathsf{ag}(i)$*, the term* t *can be computed from* $l^i_1, \ldots, l^i_k, \mathbf{kn}(A_i)$ *by Dolev-Yao operations, i.e.,* $l^i_1, \ldots, l^i_k, \mathbf{kn}(A_i) \vdash_a t$*;*
3. *the protocol has the* computable message property*: for all* $1 \leqslant k \leqslant k_j$ *we require that* r^i_k *can be computed from* $l^i_1, \ldots, l^i_k, \mathbf{kn}(A_i)$ *by Dolev-Yao operations, i.e.,* $l^i_1, \ldots, l^i_k, \mathbf{kn}(A_i) \vdash_a r^i_k$*;*
4. *the protocol has the* already known variable property*: for each* $1 \leqslant i \leqslant n$*, each* $1 \leqslant k \leqslant k_i$*, the variables of* r^i_k *are contained in the union of the variables of* $l^i_1, \ldots, l^i_k$*,* $\mathsf{X}.a$ *and* $\mathsf{X}_n(A_i)$*.*
   In addition, the terms $l^i_1, \ldots, l^i_k$ *do not contain label variables and for any subterm* $\{m\}^{\lambda}_{\mathsf{ek}(B)}$ *of* $r^i_1, \ldots, r^i_k$*,* λ *is a label variable and for any* $\{m'\}^{\lambda}_{\mathsf{ek}(B')}$ *subterm of* $r^i_1, \ldots, r^i_k$*, we have* $m = m'$ *and* $B = B'$*, and for any subterm* $[m]^{\lambda}_{\mathsf{sk}(B)}$ *of* $r^i_1, \ldots, r^i_k$*,*
   - *either* λ *is a label variable and for any* $[m']^{\lambda}_{\mathsf{sk}(B')}$ *subterm of* $r^i_1, \ldots, r^i_k$*, we have* $m = m'$ *and* $B = B'$*,*
   - *or* λ *is of the form* $\mathsf{ag}(p)$ *(*$p \in \mathbb{N}$*) and* $B = A_i$*.*

Before explaining the conditions of executable protocols, we explain the deduction rules of Figure 8.1. Consider the first row, identified as initial knowledge: if our knowledge is S, then we can deduct each element of S as well as the identity and public keys of each protocol's participant. These rules model the fact that we know who are the participants of the protocol and that public keys are already distributed to all participants.

The second two rules (pairing and unpairing) are very simple: if we know two messages m_1 and m_2, then we can generate the message $\langle m_1, m_2 \rangle$ that is the pair composed by m_1 and m_2. On the contrary, if we know a message m that is the pair $\langle m_1, m_2 \rangle$, then we can retreive each component of the pair, that is m_1 and m_2.

The third row (encryption and decryption) models our capability to generate encrypted messages and to decrypt ciphertexts. The left rule states that if we can deduct an encryption key $\mathsf{ek}(b)$ and a message m, then we are able to generate $\{m\}^{\mathsf{ag}(i)}_{\mathsf{ek}(b)}$: the encryption of m under $\mathsf{ek}(b)$ where $\mathsf{ag}(i)$ is an agent's label. Note that this rule (in conjunction with initial knowledge rules) does not permit to encrypt a message with the encryption key of an agent that is not involved into the protocol.

The right rule of the third row states that we are able to decrypt a message only if we can deduct the decryption key associated with the encryption key used to generate the ciphertext.

Finally, the signature rule of the forth row models our capability to generate signatures. In particular, if we are able to deduce a message m and a signature key $\mathsf{sk}(b)$, then we are able to produce the signature $[m]^{\mathsf{ag}(i)}_{\mathsf{sk}(b)}$; $\mathsf{ag}(i)$ is an agent's label.

Now we can start to explain the properties that characterize executable protocols. Note that such properties are sufficient to achieve the soundness result, even if probably we can find less restrictive conditions that permit to characterize executable protocols. We do not care about how much restrictive properties are, since we want to use approximated simulations to make more rigorous the soundness proof following the same hypothesis, not to question the accuracy of hypothesis.

The first condition, the executable decryption property, states that for each agent A_i, encryption keys used in the left side of each step of the corresponding role belong to A_i.

The second condition, the executable verification property, states that for each signature $[t]^{\lambda}_{\mathsf{sk}(A)}$ contained in the left side of a step of a role, it is possible to derive t from previously received messages together with the initial knowledge. The meaning of this property is that when we receive a signature $[t]^{\lambda}_{\mathsf{sk}(A)}$, we can verify it if we are able to obtain t.

The third condition, the computable message property, states that each term on the right side of a step of a role is derivable from previously received messages together with the initial knowledge. The meaning of this property is that we can generate a term only if we can compute it from our knowledge.

The fourth condition, the already known variable property, states that each variable we use when we generate a term must be a variable we already know. Moreover, the label variables must satisfy further restrictions: a label variable that occurs in an encryption term can not occur in different encryption terms; for the signatures, there is the same restriction and given $[m]^{\lambda}_{\mathsf{sk}(B)}$, if $\lambda = \mathsf{ag}(p)$ for some $p \in \mathbb{N}$, then signature is produced using the signature key of the agent corresponding to the current role.

8.2 Execution Models

The soundness result of [41] relates two different execution models: in the first one, the formal execution model, agents and adversary exchange sym-

bolic elements of a term algebra and the adversary can produce its messages only following standard Dolev-Yao restrictions [46]. In the second model, the computational execution model, messages are bitstrings and no restrictions are imposed to the adversary and agents which are implemented by probabilistic polynomial time Turing machines.

8.2.1 Formal Execution Model

In the formal execution model, messages exchanged between agents and the adversary are terms of a free algebra T^f defined as:

$$
\begin{array}{lll}
T^f ::= a & & \text{identities} \\
\quad | \ \mathsf{ek}(a) \mid \mathsf{dk}(a) \mid \mathsf{sk}(a) \mid \mathsf{vk}(a) & & \text{keys} \\
\quad | \ n(a,j,s) & & \text{agent nonces} \\
\quad | \ n(j) & & \text{adversary nonces} \\
\quad | \ \langle T^f, T^f \rangle & & \text{pairing} \\
\quad | \ \{T^f\}^{\mathsf{ag}(i)}_{\mathsf{ek}(a)} \mid \{T^f\}^{\mathsf{adv}(i)}_{\mathsf{ek}(a)} & & \text{encryption} \\
\quad | \ [T^f]^{\mathsf{ag}(i)}_{\mathsf{sk}(a)} \mid [T^f]^{\mathsf{adv}(i)}_{\mathsf{sk}(a)} & & \text{sign}
\end{array}
$$

where $a \in \mathsf{ID}$, $i, j, s \in \mathbb{N}$.

Formal execution model is defined as a state transition system (S, T). Each *global state* s is a triple (SId, f, H) where

- $\mathsf{SId} \subseteq \mathbb{N} \times \mathbb{N} \times \mathsf{ID}^k$ is the set of session ids. Given a session id $(n, j, (a_1, \ldots, a_k))$, n identifies the session, j represents the index of the role executed in session n and $a_1, \ldots, a_n \in \mathsf{ID}$ are the identities of the parties involved in the protocol;
- $f \colon \mathsf{SId} \to ((\mathsf{X} \to T^f) \times \mathbb{N} \times \mathbb{N})$ keeps the local state for each session. Given a session id sid, $f(\mathsf{sid}) = (\sigma, i, p)$ where σ partially instantiates variables of the role $\Pi(i)$ and p is the control point of the program, that is which step of the protocol is the next one to execute;
- H contains the history of the execution. That is, it contains messages generated by agents (as terms of T^f) and possibly private cryptographic keys.

There are three kinds of transition:

- $(\mathsf{SId}, f, H) \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\mathsf{SId}, f, H \cup \bigcup_{1 \leqslant j \leqslant l} \mathbf{kn}(a_j))$. Agents $a_1, \ldots, a_l$ are corrupted by the adversary and the effect of such action is to extend the history with private keys of corrupted agents;

– $(\mathsf{SId}, f, H) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{SId}', f', H')$. A new session is instantiated by the adversary; i is the index of the initial role of the new session and a_1, $\ldots$, a_k are actual agents involved in the session. The effect of such action is to reach a new global state (SId', f', H') whose components are defined as follows: let s be a the new session identifier (for example, $s = |\mathsf{SId}|+1$). $\mathsf{SId}' = \mathsf{SId} \cup \{(s, i, (a_1, \ldots, a_k))\}$ and $H' = H$. Function f' is defined as $f'(\mathsf{sid}) = f(\mathsf{sid})$ for $\mathsf{sid} \in \mathsf{SId}$ and $f'(s, i, (a_1, \ldots, a_k)) = (\sigma, i, 1)$ where $\sigma \colon \mathsf{X} \to T^f$ is partially defined as

$$\begin{cases} \sigma(A_j) = a_j & 1 \leqslant j \leqslant k \\ \sigma(X^j_{A_i}) = n(a_i, j, s) & j \in \mathbb{N} \end{cases}$$

This means that a new session is instantiated, the history is extended with the session just created and the local state of the new session maps only agent identities and nonces of the agent corresponding to the initial role i;

– $(\mathsf{SId}, f, H) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}', f', H')$. A new message $m \in T^f$ is sent by the adversary to the session identified by the session id $\mathsf{sid} \in \mathsf{SId}$. The effect of such action depends on the message m. In fact, suppose that $f(\mathsf{sid}) = (\sigma, j, p)$ for some σ, j, and p and let $\Pi(j) = ((l^j_1, r^j_1), \ldots, (l^j_k, r^j_k))$. There are two cases:
  – either there exists a substitution θ such that $m = l^j_p \sigma\theta$, then

$$f'(\mathsf{sid}') = \begin{cases} (\sigma \cup \theta, j, p+1) & \text{if } \mathsf{sid}' = \mathsf{sid} \\ f(\mathsf{sid}') & \text{if } \mathsf{sid}' \in \mathsf{SId} \setminus \{\mathsf{sid}\} \end{cases}$$

    and $H' = H \cup \{r^j_p \sigma\theta\}$,
  – or simply $f' = f$ and $H' = H$.

  This means that if the message m sent by the adversary to session sid matches the expected one, then the global state is updated adding the response message generated by the protocol to the history, variables instantiation σ of the local state is updated adding new association defined in θ and program counter is increased to the next step of the protocol. On the contrary, if the message m can not be matched with the expected one (for example, because it uses different values for the same nonce or it is a pair instead of an encrypted message), then the global state remains unchanged and no information is stored inside the history about matching failure.

$\dfrac{}{S \vdash m}\ m \in S$	$\dfrac{}{S \vdash b, \mathsf{ek}(b), \mathsf{vk}(b)}\ b \in \mathsf{X}.a$	Initial knowledge
$\dfrac{S \vdash m_1 \quad S \vdash m_2}{S \vdash \langle m_1, m_2 \rangle}$	$\dfrac{S \vdash \langle m_1, m_2 \rangle}{S \vdash m_i}\ i \in \{1,2\}$	Pairing and unpairing
$\dfrac{S \vdash \mathsf{ek}(b) \quad S \vdash m}{S \vdash \{m\}^{\mathsf{adv}(i)}_{\mathsf{ek}(b)}}\ i \in \mathbb{N}$	$\dfrac{S \vdash \{m\}^{l}_{\mathsf{ek}(b)} \quad S \vdash \mathsf{dk}(b)}{S \vdash m}$	Encryption and decryption
$\dfrac{S \vdash \mathsf{sk}(b) \quad S \vdash m}{S \vdash [m]^{\mathsf{adv}(i)}_{\mathsf{sk}(b)}}\ i \in \mathbb{N}$	$\dfrac{S \vdash [m]^{l}_{\mathsf{sk}(b)}}{S \vdash m}$	Signature

Fig. 8.2. Deduction rules for adversary

Definition 8.4. *Let* $\mathsf{SID} = \mathbb{N} \times \mathbb{N} \times \mathsf{ID}^k$ *be the set of all session ids.*

A symbolic execution trace *is a sequence of global states* (SId, f, H) *of the state transition system. The set of all symbolic execution traces is* $\mathrm{SymbTr} = (\mathsf{SID} \times (\mathsf{SID} \to ((\mathsf{X} \to T^f) \times \mathbb{N} \times \mathbb{N})) \times 2^{T^f})^*$.

Definition 8.5. *A symbolic execution trace* $(\mathsf{SId}_1, f_1, H_1) \ldots (\mathsf{SId}_n, f_n, H_n)$ *is* valid *if*

- $H_1 = \mathsf{SId}_1 = \emptyset$*;*
- $(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{a}} (\mathsf{SId}_2, f_2, H_2)$ *for* $\mathbf{a} \in \{\mathbf{corrupt}, \mathbf{new}, \mathbf{send}\}$*;*
- *for each* $1 < i < n$, $(\mathsf{SId}_i, f_i, H_i) \xrightarrow{\mathbf{a}} (\mathsf{SId}_{i+1}, f_{i+1}, H_{i+1})$ *for* $\mathbf{a} \in \{\mathbf{new}, \mathbf{send}\}$*; and*
- *if* $(\mathsf{SId}_i, f_i, H_i) \xrightarrow{\mathbf{send}(\mathsf{sid}, m)} (\mathsf{SId}_{i+1}, f_{i+1}, H_{i+1})$*, then* $H_i \vdash m$

where deduction $\vdash$ *is defined by deduction rules given in Figure 8.2.*

Given a protocol Roles*, the set of valid symbolic execution traces is denoted by* $\mathrm{Exec}^s(\mathrm{Roles})$.

Lemma 8.6 (Lemma 1 [41]). *Let* M *be a set of messages and* m *be a message. If* $M \nvdash m$*, then:*

- *either there exists a subterm* $[t]_k$ *of* m *which is not a subterm of terms in* M*,*
- *or there exists a subterm* t *of* m*, i.e.* $t = m|_p$ *for some path* p*, such that for each path* $p' \leqslant p$*,* $M \nvdash m|_{p'}$*, and* t *appears under an encryption in* M*, i.e. there exist a term* $m' \in M$ *and contexts* $\mathcal{C}$ *and* $\mathcal{C}'$ *such that* $m' = \mathcal{C}[\{\mathcal{C}'[t]\}_k]$ *with* $M \nvdash k^{-1}$*.*

The above Lemma characterizes which messages can not be derived by the adversary. In particular, a message m can not be derived using the $\vdash$

deduction when one of the following two cases occurs: it contains a signed message $[t]_k$ and such signature is not available as a subterm of terms of M. This implies that $[t]_k$ is a fresh signature that can not be found in M; or it contains a message t that appears encrypted in some message $m' \in M$ but in m it does not appear as a subterm of some derivable term. This means that t is encrypted in M, it does not appear as plaintext in M, and in m it is not a subterm of some derivable message. For example, suppose that m is the message $\{N\}_{\mathsf{ek}(a)}$ and that $M = \{m_b, m_c, m_d\}$ where $m_x = \{N\}_{\mathsf{ek}(x)}$ for $x \in \{b, c, d\}$. In this case, $M \not\vdash m$ since m contains an encryption of N that it is not available in M, thus m can be obtained only opening one of m_b, m_c, and m_d. On the contrary, for $M = \{s_a, m_b, m_c\}$ with $s_a = [\{N\}_{\mathsf{ek}(a)}]_{\mathsf{sk}(a)}$, $M \vdash m$ since $\{N\}_{\mathsf{ek}(a)}$ can be derived from s_a (using the right Sign rule of Figure 8.2).

8.2.2 Concrete Execution Model

The second model used in [41] is the *concrete execution model*. In such model, exchanged messages are bitstrings parameterized by a security parameter η that is used to establish, for example, the size of nonces and cryptographic keys.

Set of valid messages is denoted by $\mathcal{C}^\eta$ and it is partitioned into sets $\mathcal{C}^\eta.a$, $\mathcal{C}^\eta.n$, $\mathcal{C}^\eta.e$, $\mathcal{C}^\eta.v$, $\mathcal{C}^\eta.c$, $\mathcal{C}^\eta.s$, and $\mathcal{C}^\eta.p$ that represent agent identities, nonces, encryption keys, verification keys, ciphertexts, signatures, and pairs, respectively.

Given a message $m \in \mathcal{C}^\eta$, any implementation must permit to efficiently recover the type of the message, that is, if it is a pair or an encryption key. To do this, the total function $\mathsf{type}\colon \mathcal{C}^\eta \to \{a, n, e, v, c, s, p\}$ is used.

To share public keys between agents and adversary, two functions are used: $\mathsf{ek}\colon \mathcal{C}^\eta.a \to \mathcal{C}^\eta.e$ and $\mathsf{vk}\colon \mathcal{C}^\eta.a \to \mathcal{C}^\eta.v$. These functions are supposed to be available to all parties, publicly computable and efficiently invertible, that is given a public key, it is easy to recover corresponding agent.

The concrete implementation of encryption and signing is based on two schemes: a *public key encryption scheme* $\mathsf{E} = (\mathtt{KGen}_e, \mathtt{Enc}, \mathtt{Dec})$ and a signature scheme $\mathsf{S} = (\mathtt{KGen}_s, \mathtt{Sig}, \mathtt{Ver})$. We suppose that, given a ciphertext or a signature, it is easy to derive the public key associated to the message and, for signatures, the signed message. Pairing is implemented by some standard and efficiently invertible encoding function $\langle \cdot, \cdot \rangle\colon \mathcal{C}^\eta \times \mathcal{C}^\eta \to \mathcal{C}^\eta.p$.

We define the size of a message $m \in$ Message inductively as follows:

$$\text{size}(m) = \begin{cases} 1 & \text{if type}(m) \in \{a, n, e, v\} \\ \text{size}(p) + 1 & \text{if type}(m) = c \text{ and } m = \texttt{Enc}(e, p) \\ \text{size}(p) + 1 & \text{if type}(m) = s \text{ and } m = \texttt{Sig}(s, p) \\ \text{size}(m_1) + \text{size}(m_2) + 1 & \text{if type}(m) = p \text{ and } m = \langle m_1, m_2 \rangle \end{cases}$$

The concrete execution model is defined as a state transition system (S, T), similarly to the formal execution model. Each *global state* s is a triple (SId, f) where

- $SId \subseteq \mathbb{N} \times \mathbb{N} \times \mathcal{C}^\eta.a^k$ is the set of session ids. Given a session id $(n, j, (a_1, \ldots, a_k))$, n identifies the session, j represents the index of the role executed in session n and $a_1, \ldots, a_n \in \mathcal{C}^\eta.a$ are the identities of the parties involved in the protocol;
- $f \colon SId \to ((X \to \mathcal{C}^\eta) \times \mathbb{N} \times \mathbb{N})$ keeps the local state for each session. Given a session id sid, $f(sid) = (\sigma, i, p)$ where $\sigma \colon X \to \mathcal{C}^\eta$ partially instantiates variables of the role $\Pi(i)$ and p is the control point of the program, that is which step of the protocol is the next one to execute.

Global states of the concrete execution model are essentially the same of the formal execution model, except that history H is missing.

There are three kinds of transition and they are very similar to ones of formal execution model:

- $(SId, f) \xrightarrow{\mathsf{corrupt}(a_1,\ldots,a_l)} (SId, f)$. Agents $a_1, \ldots, a_l \in \mathcal{C}^\eta.a$ are corrupted by the adversary and the effect of such action is to generate public and private keys for corrupted agents using key generation algorithms $\texttt{KGen}_e$ and $\texttt{KGen}_s$: public keys are published and private keys are given to the adversary;
- $(SId, f) \xrightarrow{\mathsf{new}(i,a_1,\ldots,a_k)} (SId', f')$. A new session is instantiated by the adversary; i is the index of the initial role of the new session and $a_1, \ldots, a_k \in \mathcal{C}^\eta.a$ are actual agents involved in the session. The effect of such action is to reach a new global state (SId', f) whose components are defined as follows: let s be a the new session identifier (for example, $s = |SId|+1$). $SId' = SId \cup \{(s, i, (a_1, \ldots, a_k))\}$. Function f' is defined as $f'(sid) = f(sid)$ for $sid \in SId$ and $f'(s, i, (a_1, \ldots, a_k)) = (\sigma, i, 1)$ where $\sigma \colon X \to \mathcal{C}^\eta$ is partially defined as

$$\begin{cases} \sigma(A_j) = a_j & 1 \leqslant j \leqslant k \\ \sigma(X^j_{A_i}) = n(a_i, j, s) \in_R \mathcal{C}^\eta.n & j \in \mathbb{N}. \end{cases}$$

In addition, for each term $\{t\}^l_{\mathsf{ek}(A_j)}$ and each term $[t]^l_{\mathsf{sk}(A_i)}$ that are sent (i.e. occurring within some r^i_j of $\Pi(i)$) we choose random coins $re^{sid}(t, A_j, l)$ and $rs^{sid}(t, A_i, l)$ respectively. These coins will later be used in randomizing the encryption and signing functions in the concrete implementation.

This means that when a new session is instantiated, the history is extended with the session just created and the local state of the new session maps only agent identities and nonces of the agent corresponding to the initial role i;

– $(SId, f) \xrightarrow{\mathsf{send}(sid,m)} (SId, f')$. A new message $m \in \mathcal{C}^\eta$ is sent by the adversary to the session identified by the session id $sid \in SId$. The effect of such action depends on the message m. In fact, suppose that $f(sid) = (\sigma, j, p)$ for some σ, j, and p and let $\Pi(j) = ((l^j_1, r^j_1), \ldots, (l^j_k, r^j_k))$.
First step is to parse m as an instantiaton of l^j_p. Let θ be the resulting mapping that assignees variables of l^j_p to values of $\mathcal{C}^\eta$. If the parsing procedure fails (this can happen for example when types of bitstring and term do not match), then $f' = f$.
If the parsing procedure ends correctly, the second step is to update local state (σ, j, p) accordingly to θ. There are two cases, depending on σ and θ:
  – either θ and σ are compatible, then

$$f'(sid') = \begin{cases} (\sigma \cup \theta, j, p+1) & \text{if } sid' = sid \\ f(sid') & \text{if } sid' \in SId \setminus \{sid\} \end{cases}$$

  – or simply $f' = f$.

  If θ and σ are compatible, then the message r^j_p is computed: each variable X of r^j_p is replaced by $(\sigma \cup \theta)(X)$ and actual encryptions and signatures are obtained as result of the use of algorithms `Enc` and `Sig`. Resulting bitstring is then given to the adversary as answer of $\mathsf{send}(sid, m)$ action.

Definition 8.7. *Let $\mathcal{SID} = \mathbb{N} \times \{0, \ldots, k\} \times \mathcal{C}^\eta.a^k$ be the set of all session ids.*

A concrete trace *is a sequence of global states (f, SId) of the state transition system. The set of all concrete traces is*

$$\text{ConcTr} = \bigcup\nolimits_\eta (\mathcal{SID} \times (\mathcal{SID} \to ((X \to \mathcal{C}^\eta.a) \times \mathbb{N} \times \mathbb{N})))^*$$

Given an adversary A and a protocol Π, we can obtain several concrete traces. This is due to the fact that the adversary is probabilistic

and the cryptographic primitives used by the protocol are randomized. In fact nonces, encryption and signatures are generated by random choices (for nonces) or by randomized algorithms (for encryption and signatures). Since the adversary runs in polynomial time, then at most a polynomial number of random coins are used by the protocol. Let p_A and p_Π two polynomials that bound the number of random coins used by adversary and protocol, respectively. Let $(R_A, R_\Pi) \in \{0,1\}^{p_A(\eta)} \times \{0,1\}^{p_\Pi(\eta)}$ be a pair of sequences of random coins. (R_A, R_Π) determines a unique concrete trace $(SId_1, f_1), (SId_2, f_2), \ldots$ which is denoted by $\mathrm{Exec}^c_{\Pi(R_\Pi),A(R_A)}(\eta)$.

8.3 Relating Symbolic and Concrete Traces

Concrete traces can be regarded as instantiations of formal traces via appropriate renaming of the variables.

Definition 8.8. *Let* $\mathsf{sid} = (t, i, (b_1, \ldots b_k))$ *be a symbolic session id and* $sid = (s, j, (a_1, \ldots, a_l))$ *be a concrete session id.*

We say that sid *is equal to* sid *(denoted by* $\mathsf{sid} = sid$*) if* $s = t$*,* $i = j$*,* $k = l$*, and for each* $z \in \{1, \ldots, k\}$*,* a_z *and* b_z *denote the same agent.*

Let SId *be a set of symbolic session ids and* SId *be a set of concrete session ids.*

$\mathsf{SId} = SId$ *if for each* $\mathsf{sid} \in \mathsf{SId}$ *there exists* $sid \in SId$ *such that* $\mathsf{sid} = sid$ *and for each* $sid \in SId$ *there exists* $\mathsf{sid} \in \mathsf{SId}$ *such that* $sid = \mathsf{sid}$*.*

Definition 8.9. *Take a concrete trace* $t^c = (SId_1, g_1), \ldots, (SId_m, g_m)$ *and a symbolic execution trace* $t^s = (\mathsf{SId}_1, f_1, H_1), \ldots, (\mathsf{SId}_n, f_n, H_n)$*.*

Trace t^c *is a* concrete instantiation *of* t^s *(or alternatively* t^s *is a* symbolic representation *of* t^c*), denoted by* $t^s \preceq t^c$*, if* $m = n$ *and there exists an injective function* $c\colon T^f \to \mathcal{C}^\eta$ *such that for each* $i \in \{1, \ldots, n\}$ *the following conditions hold:*

- $SId_i = \mathsf{SId}_i$*,*
- *for each* $sid \in \mathsf{SId}_i$*, if* $f_i(sid) = (\sigma, j, p)$ *and* $g_i(sid) = (\tau, l, q)$*, then*
  - $\tau = c \circ \sigma$*,*
  - $j = l$*, and*
  - $p = q$*.*

Lemma 8.10 (Lemma 2 [41]). *Let* Π *be an executable protocol. If in the concrete implementation* E *is an IND-CCA encryption scheme and* S *is a*

unforgeable encryption scheme, then for any probabilistic polynomial time algorithm A

$$\boldsymbol{Pr}[\exists t^s \in \mathrm{Exec}^s(\Pi) \mid t^s \preceq \mathrm{Exec}^c_{\Pi(R_\Pi),A(R_A)}(\eta)] \geqslant 1 - \nu_A(\eta)$$

where the probability is over the choice $(R_\Pi, R_A) \in_R \{0,1\}^{p_A(\eta)} \times \{0,1\}^{p_\Pi(\eta)}$ *and* $\nu_A(\,\cdot\,)$ *is some negligible function.*

The above Lemma states that with overwhelming probability the concrete traces of a protocol are instantiations of *valid* symbolic execution traces.

8.3.1 Proof of Lemma 8.10

The original proof of Lemma 8.10 in [41] is done in two steps: in the first one a mapping from bitstrings to symbols is defined while in the second one it is shown that with overwhelming probability the resulting symbolic trace is a valid trace.

In Step I, random coins R_Π and R_A are fixed and then the execution of adversary A is started. During the execution, a function $c\colon \mathcal{C}^\eta \to T^f$ is maintained and it is used to map each bitstring that occurs to a formal term. The function c is initialized by mapping agent identities in $\mathcal{C}^\eta.a$ to symbolic names, that is, $c(a_i) = a_i$. Since R_Π is fixed, then also cryptographic keys are fixed. So, if cryptographic keys for agent a_i are ek_i, dk_i, sk_i, and vk_i, then c is extended with $c(ek_i) = \mathsf{ek}(a_i)$, $c(dk_i) = \mathsf{dk}(a_i)$, $c(sk_i) = \mathsf{sk}(a_i)$, and $c(vk_i) = \mathsf{vk}(a_i)$.

Adversary A uses three kinds of actions to interact with the protocol. Thus function c must be updated accordingly:

$\mathsf{corrupt}(a_1, \ldots, a_l)$: mapping c is not modified, since the effect of a corrupt action is to generate keys for agents $a_1, \ldots, a_l$, to publish public keys, and to give private keys to the adversary. Function c already maps private and public keys to corresponding symbols, since keys are uniquely determined by R_Π that is fixed;

$\mathsf{new}(s, i, (a_1, \ldots, a_k))$: the effect of a new action is to generate keys for involved agents and publish public keys. Moreover, all nonces used by agents are generated. Similarly to corrupt, we do not update mapping c to consider keys, since they are already mapped, but we need to update c to relate concrete and symbolic nonces. To do this, since nonces are determined by R_Π that is fixed, each concrete nonce is mapped to

the corresponding symbolic nonce. That is, if nonce n is associated to variable $X_{a_i}^j$ (for some $j \in \mathbb{N}$), then $c(n) = n(a_i, j, s)$;

$\mathsf{send}(sid, m)$: as first thing, the parse tree of m is constructed recursively: given m, if $\text{type}(m) = c$ (that is, m is a ciphertext), then it is recovered the encryption key e used to obtain m, the agent a which e refers and then the original plaintext p using $\mathsf{dk}(a)$ (that is known). Then it p is parsed; if $\text{type}(m) = p$ (that is, m is a pair), then elements m_1, m_2 such that $m = \langle m_1, m_2 \rangle$ are parsed; if $\text{type}(m) = s$ (that is, m is a signature), then the signed message m' is recovered from m together with the verification key v and then m' is parsed. For all other cases ($\text{type}(m) \in \{a, n, e, v\}$) parsing procedure stops, since in such cases m is a basic element.

At the end of parsing procedure, we have obtained a parsing tree whose leafs are nonces, agent identities, or cryptographic keys. Mapping c is already defined for agent identities, cryptographic keys, and almost all nonces but there can be some nonce that is not mapped into symbolic nonces. In fact, c is already defined on all nonces generated by agents but it is not defined on nonces generated by adversary. In this case, c is extended mapping each adversary's nonce to a fresh adversary's symbolic nonce. The mapping of other nodes of the parsing tree is then a straightforward bottom-up procedure.

Once the mapping c is defined for all messages that are exchanged in the execution $\text{Exec}^c_{\Pi(R_\Pi),A(R_A)}(\eta)$, then the symbolic trace t^s is defined as $c(\text{Exec}^c_{\Pi(R_\Pi),A(R_A)}(\eta))$ that is the sequence of queries of A where:

- each query $\mathsf{corrupt}(a_1, \ldots, a_l)$ is replaced by $\mathbf{corrupt}(a_1, \ldots, a_l)$;
- each query $\mathsf{new}(s, i, (a_1, \ldots, a_k))$ is replaced by $\mathbf{new}(s, i, (a_1, \ldots, a_k))$;
- each query $\mathsf{send}(sid, m)$ is replaced by $\mathbf{send}(\mathsf{sid}, c(m))$ where $\mathsf{sid} = sid$.

The inverse of the function c maps the symbolic trace t^s to the execution $\text{Exec}^c_{\Pi(R_\Pi),A(R_A)}(\eta)$, denoted by $c^{-1}(t^s) = \text{Exec}^c_{\Pi(R_\Pi),A(R_A)}(\eta)$.

Once function c is fully defined on the concrete trace $\text{Exec}^c_{\Pi(R_\Pi),A(R_A)}(\eta)$, in Step II it is shown that resulting symbolic trace t^s is valid with overwhelming probability. This is done supposing that t^s is not valid with non-negligible probability and then showing that adversary A is an attacker of the joint security of encryption scheme E and signature scheme S. The proof that A is an attacker of the joint security is a standard proof of computational model: A is used as a subroutine by another adversary B that behaves as a forger for the signature scheme or as a distinguisher for the encryption

scheme that performs a successful attack with non-negligible probability. In particular, B is a forger when A generates the invalid query $\mathsf{send}(sid, m)$ where $c(m)$ contains some signature $[t]_{sk}$ that did not occur at all in the prior communication, that is A has generated a fresh and valid signature of t without obtaining it from the agent that own secret key sk. On the other hand, B is a distinguisher when A generates the invalid $\mathsf{send}(sid, m)$ where $c(m)$ contains some formal term t such that there exists a message m sent by honest parties and two contexts $\mathcal{C}$ and $\mathcal{C}'$ such that $m' = \mathcal{C}'[\{\mathcal{C}[c^{-1}(t)]\}_{\mathsf{ek}(a)}]$ for some honest agent a and the term $\{\mathcal{C}[c^{-1}(t)]\}_{\mathsf{ek}(a)}$ do not appear on the path from m to $c^{-1}(t)$ in the parse tree of m.

8.3.2 Some Notes About The Proof of Lemma 8.10

As we have seen previously, the proof is performed in two steps: the first one defines a mapping c from bitstrings to symbolic terms, while the second ones shows that the resulting symbolic trace is valid with overwhelming probability. Note that probability is considered only in the second step of the proof, assuming that c obtained from first step is a mapping that can be inverted and hence c^{-1} is injective.

These assumptions are generally true but there are some cases that invalidate them, even if they do not affect the validity of the proof. Suppose, for example, that there exists a nonce, say n, that is mapped into X^1_a. Suppose that adversary A performs a $\mathsf{new}(s, i, (a_1, \ldots, a_l))$ and that n is used another time, for example because it is assigned to $X^2_{a_3}$ where a and a_3 denotes two different agents. There are two cases: either the pair $(n, X^2_{a_3})$ is simply added to c (and hence c, considered as a relation, contains both $(n, X^2_{a_3})$ and (n, X^1_a)), or the pair (n, X^1_a) is replaced by $(n, X^2_{a_3})$. In the first case, c is not a function anymore, since it maps the same value n to two different symbols (X^1_a and $X^2_{a_3}$) and c^{-1} is not injective and thus it can not be used as the injective function that justifies $t^s \preceq \mathrm{Exec}^c_{\Pi(R_\Pi), A(R_A)}(\eta)$ (see Definition 8.9). So we must replace (n, X^1_a) with $(n, X^2_{a_3})$. In this case each symbolic message that uses X^1_a actually contains $X^2_{a_3}$ as subterm instead of X^1_a and this can condition second step of the proof, since an invalid trace could not lead to an attack to the joint security of cryptographic schemes.

We have the same situation when we consider adversary's generated nonces. In fact, it can generate randomly a nonce n that is equal to some agent's generated nonce. In this case, when c is updated after a send action, n is not mapped into the set of adversary's symbolic nonces but it is

associated to the nonce of some agent. This implies that a simple guess can be confused with the decryption of a ciphertext, when n is never sent as cleartext in all prior communication.

Claim 2.2 states that the probability of repeated nonces is negligible thus above situation occurs with negligible probability. This means that we can exclude traces with repeated nonces before performing first step of the proof, so the resulting function c is really an invertible mapping and when we consider an invalid symbolic trace at second step, then we are sure it does not depend on wrong mapping but only on adversary's choices.

Other problems are related to encryptions and signatures. In particular, the concrete model is not able to forward signatures and ciphertexts since by definition of the `send` action, each time they are needed, they are obtained invoking the `Enc` and `Sig` algorithms. This implies that with high probability the bitstring we use is not the same of the one we received. This means that we map two different bitstrings to the same symbolic term and hence when we consider the inverse c^{-1} of the resulting map c, c^{-1} is not a function.

8.4 Analysis of the Soundness Proof using Probabilistic Automata

As we have seen in the previous section, proof of Lemma 8.10 is correct but some steps require to be particularly careful, since they are not so immediate and depends on properties that hold only after the step we are performing.

In the following of the section we will show how probabilistic automata and approximated simulations can be used to manage in an easier way the technicalities we found in the proof. In particular, we are able to remove problematic cases before the generation of the mapping from bitstrings to symbols, so the resulting mapping satisfies required properties.

8.4.1 An Overview

To model a generic cryptographic protocol, we use several automata; each automaton represents a single functionality and their composition models the interaction between the cryptographic protocol and the adversary. Figure 8.3 shows an overview of the model we develop, which kind of automata we use and the way they interact. In particular, NG represents the nonce

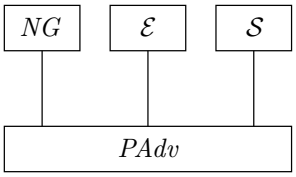

Fig. 8.3. Automata model of a generic cryptographic protocol

generator, $\mathcal{E}$ the encryption oracle, $\mathcal{S}$ the signature oracle, and *PAdv* the protocol and the adversary, which constitute a single automaton.

We describe the protocol at several levels of abstraction. The lowest level description consists on the actual protocol that receives cryptographic keys from a key generator, receives nonces from a device that generates random numbers and performs cryptographic operations querying them to the oracles that encrypt and sign messages, decrypt ciphertexts and verify signatures. The adversary is controlled by a generic probabilistic polynomial time algorithm.

The highest level abstraction is similar in style to the Dolev-Yao model where we assume perfect cryptography.

8.4.2 The Modeling

We now define the automata we use to analyze the soundness proof of [41]. Each automaton is parameterized by a security parameter $k \in \mathbb{N}$ and the set of identities of agents $\mathbb{A}$.

Nonce Generator

For the nonce generator, we consider the automaton $NG_k(\mathbb{A})$ that represents a *real nonce generator*, that is a nonce generator that chooses next nonce flipping a coin without ensuring that chosen value is not a repeated nonce.

We model the $NG_k(\mathbb{A})$ automaton exactly as the $NG_k(\mathbb{A})$ automaton defined in Section 6.1 and that we depict again in Figure 8.4.

Encryption Oracle

The probabilistic automaton $\mathcal{E}_k(\mathbb{A})$ that models the encryption scheme $\mathsf{E} = (\mathtt{KGen}, \mathtt{Enc}, \mathtt{Dec})$ is exactly the encryption automaton $\mathcal{E}_k(\mathbb{A})$ defined in Section 6.2 and that is shown in Figure 8.5.

Nonce Generator $NG_k(\mathbb{A})$

Signature:

Input:

$get\_nonce(A)$, $A \in \mathbb{A}$

Output:

$ret\_nonce(A, n)$, $n \in \{0,1\}^k$, $A \in \mathbb{A}$

State:

$value_A \in \{0,1\}^k \cup \{\perp\}$, initially $\perp$, $A \in \mathbb{A}$

Transitions:

Input $get\_nonce(A)$

Effect:

$value_A := v$ where $v \in_R \{0,1\}^k$

Output $ret\_nonce(A, n)$

Precondition:

$n = value_A$

Effect:

$value_A := \perp$

Fig. 8.4. The Nonce Generator $NG_k(\mathbb{A})$

Signature Oracle

Similarly to the case of the encryption oracle, the automaton $\mathcal{S}_k(\mathbb{A})$ that models the signature scheme $\mathsf{S} = (\mathtt{KGen}, \mathtt{Sig}, \mathtt{Ver})$ is exactly the signature automaton $\mathcal{S}_k(\mathbb{A})$ defined in Section 6.3 and that is depicted in Figure 8.6.

Adversary and Protocol: the Concrete Model

We now describe the automaton that models the protocol plus the adversary. We have chosen to use a single automaton for two main motivations: the first one is that we adopt the same approach of [41], the second one is that in this way we provide a simpler automaton. In fact, if we represent the protocol and the adversary using two different automata, then we must provide both automata with a set of actions that are used to exchange messages; moreover, the protocol must notify the adversary about the failure/rejection of the message it sent to the protocol. This can be done either by a new action, or by a new message. In both cases, the adversary receives an input that is not provided in the model considered in [41].

Encryption automaton $\mathcal{E}_k(\mathbb{A})$

Signature:

Input:

$get\_public\_encrypt(A)$, $A \in \mathbb{A}$
$get\_corrupt\_encrypt(A)$, $A \in \mathbb{A}$
$get\_encrypt(A, M)$, $A \in \mathbb{A}$, $M \in$ Message
$get\_decrypt(A, C)$, $A \in \mathbb{A}$, $C \in$ Ciphertext

Output:

$ret\_public\_encrypt(A, E)$, $A \in \mathbb{A}$, $E \in$ EKey
$ret\_corrupt\_encrypt(A, E, D)$, $A \in \mathbb{A}$, $E \in$ EKey, $D \in$ DKey
$ret\_encrypt(A, C)$, $A \in \mathbb{A}$, $C \in$ Ciphertext
$ret\_decrypt(A, M)$, $A \in \mathbb{A}$, $M \in$ Message

State:

ek_A $\in \{\bot\} \cup$ EKey, $A \in \mathbb{A}$, initially $\bot$
dk_A $\in \{\bot\} \cup$ DKey, $A \in \mathbb{A}$, initially $\bot$
pk_A $\in \{\bot\} \cup$ EKey, $A \in \mathbb{A}$, initially $\bot$
ck_A $\in \{\bot\} \cup$ EKey $\times$ DKey, $A \in \mathbb{A}$, initially $\bot$
enc_value_A $\in \{\bot\} \cup$ Ciphertext, $A \in \mathbb{A}$, initially $\bot$
dec_value_A $\in \{\bot\} \cup$ Message, $A \in \mathbb{A}$, initially $\bot$

Transitions:

Input $get\_public\_encrypt(A)$
Effect:
if $ek_A = \bot$ then
$(ek_A, dk_A) := \mathtt{KGen}(1^k)$
fi
$pk_A := ek_A$

Input $get\_corrupt\_encrypt(A)$
Effect:
if $ek_A = \bot$ then
$(ek_A, dk_A) := \mathtt{KGen}(1^k)$
fi
$ck_A := (ek_A, dk_A)$

Input $get\_encrypt(A, M)$
Effect:
$enc\_value_A := \mathtt{Enc}(ek_A, M)$

Input $get\_decrypt(A, C)$
Effect:
$dec\_value_A := \mathtt{Dec}(dk_A, C)$

Output $ret\_public\_encrypt(A, E)$
Precondition:
$pk_A = E$
Effect:
$pk_A := \bot$

Output $ret\_corrupt\_encrypt(A, E, D)$
Precondition:
$ck_A = (E, D)$
Effect:
$ck_A := \bot$

Output $ret\_encrypt(A, C)$
Precondition:
$enc\_value_A = C$
Effect:
$enc\_value_A := \bot$

Output $ret\_decrypt(A, M)$
Precondition:
$dec\_value_A = M$
Effect:
$dec\_value_A := \bot$

Fig. 8.5. Encryption automaton $\mathcal{E}_k(\mathbb{A})$

Signature automaton $\mathcal{S}_k(\mathbb{A})$

Signature:

Input:

$get\_public\_sign(A)$, $A \in \mathbb{A}$
$get\_corrupt\_sign(A)$, $A \in \mathbb{A}$
$get\_sign(A, M)$, $A \in \mathbb{A}$, $M \in$ Message
$get\_verify\_sign(A, S)$, $A \in \mathbb{A}$, $S \in$ Signature

Output:

$ret\_public\_sign(A, V)$, $A \in \mathbb{A}$, $V \in$ VKey
$ret\_corrupt\_sign(A, V, S)$, $A \in \mathbb{A}$, $V \in$ VKey, $S \in$ SKey
$ret\_sign(A, S)$, $A \in \mathbb{A}$, $S \in$ Signature
$ret\_verify\_sign(A, B)$, $A \in \mathbb{A}$, $B \in \{T, F\}$

State:

sk_A $\in \{\bot\} \cup$ SKey, $A \in \mathbb{A}$, initially $\bot$
vk_A $\in \{\bot\} \cup$ VKey, $A \in \mathbb{A}$, initially $\bot$
pk_A $\in \{\bot\} \cup$ VKey, $A \in \mathbb{A}$, initially $\bot$
ck_A $\in \{\bot\} \cup$ VKey $\times$ SKey, $A \in \mathbb{A}$, initially $\bot$
sig_value_A $\in \{\bot\} \cup$ Signature, $A \in \mathbb{A}$, initially $\bot$
ver_value_A $\in \{\bot\} \cup \{T, F\}$, $A \in \mathbb{A}$, initially $\bot$

Transitions:

Input $get\_public\_sign(A)$
Effect:
if $sk_A = \bot$ then
$(sk_A, vk_A) := \mathtt{KGen}(1^k)$
fi
$pk_A := vk_A$

Input $get\_corrupt\_sign(A)$
Effect:
if $sk_A = \bot$ then
$(sk_A, vk_A) := \mathtt{KGen}(1^k)$
fi
$ck_A := (vk_A, sk_A)$

Input $get\_sign(A, M)$
Effect:
$sig\_value_A := \mathtt{Sig}(sk_A, M)$

Input $get\_verify\_sign(A, S)$
Effect:
$ver\_value_A := \mathtt{Ver}(vk_A, S)$

Output $ret\_public\_sign(A, V)$
Precondition:
$pk_A = V$
Effect:
$pk_A := \bot$

Output $ret\_corrupt\_sign(A, V, S)$
Precondition:
$ck_A = (V, S)$
Effect:
$ck_A := \bot$

Output $ret\_sign(A, S)$
Precondition:
$sig\_value_A = S$
Effect:
$sig\_value_A := \bot$

Output $ret\_verify\_sign(A, B)$
Precondition:
$ver\_value_A = B$
Effect:
$ver\_value_A := \bot$

Fig. 8.6. Signature automaton $\mathcal{S}_k(\mathbb{A})$

Unlike for the other components, we do not provide the overall representation of the automaton that models both adversary and protocol, but we describe only the transitions it perform. The complete automaton can be reconstructed collecting actions and state variables from the description of each single transition we will list below.

For each state of the automaton, that we denote by $PAdv_k^{adv}(\mathbb{A})$, there is a variable *status* that is used as a program counter to know the next kind of operation to perform. The automaton runs a cycle where the probabilistic polynomial time function *adv* is used to generate the next action to perform. There are only three kind of actions the adversary can execute and that *adv* can choose: *corrupt*, *new* and *send*, that correspond to the corruption of agents, instantiation of a new session, and the sending of a message to a session. Once *adv* has chosen the next action, it is executed, though this may require to invoke the external primitives, such as nonce and keys generation, encryption, signing, and so on. To query the external primitives, we enable the corresponding actions of the automata we have described above in this section.

As first thing, we choose the next action to perform with the function *adv* that uses as input the state variable *history* that keeps the information about all previous events (such as creation of new sessions, corruption of agents, sending of messages, an so on). The *history* variable is a sequence of actions and it is initialized to the empty sequence.

Internal *create_action*
  Precondition:
    $status = \mathsf{create}$
  Effect:
    $(action, message) := adv(history)$
    $status := \mathsf{act}$

The precondition of the *create_action* action is that *status* is create, that is, we have completed the previous action and we can choose the next thing to do. In this way we are sure that the adversary models a sequential algorithm where it is not possible to intermix two different thing but the execution of an action can start only at the end of the previous one. The function *adv* returns a pair of values $(action, message)$. *action* represents the next action to perform and *message* is the argument of it.

One action the adversary can perform is the corruption of a set of agents. This capability is modelled by the action *corrupt* that is possible only at

the beginning of the execution and it induces the generation of the keys of the corrupted agents. In the code below variable *agentsToCorrupt*, which initial value is the empty sequence λ, is updated with the list of agents that should be corrupted. Such list is specified by the *message* variable that was set by the *create_action* action. Then a cycle starts where, for each agent to corrupt, first the signature scheme and then the encryption scheme are queried to generate a pair of keys and return both of them. This is done enabling the output actions $get\_corrupt\_sign(\eta)$ and $get\_corrupt\_encrypt(\eta)$ that activate the corresponding input action of $\mathcal{S}_k(\mathbb{A})$ and $\mathcal{E}_k(\mathbb{A})$. Each request updates the *history* variable adding itself to the history; each pair of keys that are received is stored into the vectors of keys: for example, if we obtain the public encryption key *public* and the private decryption key *private* for agent A, then we store *public* into $ek(A)$ and *private* into $dk(A)$. We operate similarly for private and public signature keys.

Output $corrupt(\eta_1 \dots \eta_l)$
  Precondition:
    $history = \lambda$
    $action = corrupt$
    $message = (\eta_1 \dots \eta_l)$
  Effect:
    $history := corrupt(\eta_1 \dots \eta_l)$
    $action := \bot$
    $agentsToCorrupt := \eta_1 \dots \eta_l$
    $status :=$ get_sig_key

Output $get\_corrupt\_sign(\eta)$
  Precondition:
    $\mathrm{head}(agentsToCorrupt) = \eta$
    $status =$ get_sig_key
  Effect:
    $history := history \vdash get\_corrupt\_sign(\eta)$
    $status :=$ wait_sig_key

Input $ret\_corrupt\_sign(\eta, public, secret)$
  Effect:
    $history := history \vdash ret\_corrupt\_sign(\eta, public, secret)$
    $vk(\eta) := public$
    $sk(\eta) := private$
    $status :=$ get_enc_key

Output $get\_corrupt\_encrypt(\eta)$
Precondition:
$\text{head}(agentsToCorrupt) = \eta$
$status = \mathsf{get\_enc\_key}$
Effect:
$history := history \vdash get\_corrupt\_encrypt(\eta)$
$status := \mathsf{wait\_enc\_key}$

Input $ret\_corrupt\_encrypt(\eta, public, secret)$
Effect:
$history := history \vdash ret\_corrupt\_encrypt(\eta, public, secret)$
$ek(\eta) := public$
$dk(\eta) := private$
$agentsToCorrupt := \text{tail}(agentsToCorrupt)$
$status :=$ **if** $agentsToCorrupt \neq \lambda$ **then** $\mathsf{get\_sig\_key}$ **else** create

Once the keys of all corrupted agents are received, then the *status* variable is set to create and this enables the *create_action* action that chooses the next action to perform. *create_action* can choose again to corrupt another set of agents but this choice blocks the adversary, since the model allows the adversary to corrupt agents only at the beginning. We block the adversary imposing, as precondition of *corrupt*, that $history = \lambda$, that is, *history* is the empty sequence. Now, if the $adv(history)$ function returns again $(corrupt, message)$ for some *message*, then since $history \neq \lambda$ $corrupt(\dots)$ is not enabled. As we will see in the following, only $corrupt(\dots)$ contains the precondition $action = corrupt$, so other actions are not enabled and thus the automaton is in deadlock.

Another possible action the adversary can perform is the instantiation of a new session. This is obtained using the action *new* that creates a new session and makes sure that all necessary keys are generated. In this case a variable *agents_to_init*, which initial value is the empty sequence λ, is updated to the set of new agents for which no key was generated yet. Then a cycle is started where first the signature scheme and then the encryption scheme are queried to generate a pair of keys and return the public one.

Output $new(i, \eta_1 \dots \eta_l)$
Precondition:
$action = new$
$message = (i, \eta_1 \dots \eta_l)$
Effect:
$history := history \vdash new(i, \eta_1, \dots, \eta_l)$
$action := \bot$
$CSId := CSId \cup \{cs\}$
$f := f \cup \{(cs, (\tau, i, 1))\}$
where $cs = (|CSId| + 1, i, (\eta_1 \dots \eta_l))$, $\tau = \{(A_j, \eta_j) \mid 1 \leqslant j \leqslant l\}$
$agents\_to\_init := (\eta_1 \dots \eta_l) - \{\eta \mid ek(\eta) \neq \bot\}$
$status :=$ **if** $agents\_to\_init \neq \lambda$ **then** get_sig_key **else** create

Output $get\_public\_sign(\eta)$
Precondition:
$\mathrm{head}(agents\_to\_init) = \eta$
$status =$ get_sig_key
Effect:
$history := history \vdash get\_public\_sign(\eta)$
$status :=$ wait_sig_key

Input $ret\_public\_sign(\eta, public)$
Effect:
$history := history \vdash ret\_public\_sign(\eta, public)$
$vk(\eta) := public$
$status :=$ get_enc_key

Output $get\_public\_encrypt(\eta)$
Precondition:
$\mathrm{head}(agents\_to\_init) = \eta$
$status =$ get_enc_key
Effect:
$history := history \vdash get\_public\_encrypt(\eta)$
$status :=$ wait_enc_key

Input $ret\_public\_encrypt(\eta, public)$
Effect:
$history := history \vdash ret\_public\_encrypt(\eta, public)$
$ek(\eta) := public$
$agents\_to\_init := \mathrm{tail}(agents\_to\_init)$
$status :=$ **if** $agents\_to\_init \neq \lambda$ **then** get_sig_key **else** create

As for the corruption of agents, once we have obtained all keys of instantiated agents, we can set *status* to create to choose the next action to perform.

The last action that can be chosen by the *adv* function is the sending of a message to a specific session. We model such sending using the action *send* that is much more complex to treat with respect to other actions the adversary can perform. We need to identify the role that receives the message and the rule that should be applied, which is identified by a pair (j, p). If the rule to be applied exists, that is $p \leqslant k_j$ where k_j is the number of rules of the role j, then we have to parse m according to the left-hand side of the rule, possibly invoking external decryption and signature verification primitives, and updating the τ function. τ is a function that maps terms to bitstrings and it is used to keep information about which bitstrings are associated to the terms used in the description of the role. Then we have to compute the right-hand side of the rule, apply the encryption and signing primitives and generating nonces where necessary, updating the τ function along the way.

We found many problems in the original paper while trying to formalize the constructions. As an example, in the left side we may have a subterm like $\{t\}_k^l$ that appears as well on the right side. In such case, the encryption of t should not be recomputed, while according to the informal description in the paper it is recomputed. We can solve the problem only by defining τ for all terms and not only for basic terms like nonce variables, agent identities, and keys.

Output $send(cs, m)$
  Precondition:
    $action = send$
    $message = (cs, m)$
  Effect:
    $history := history \vdash send(cs, m)$
    $action := \bot$
    $csid := cs$
    if $p \leqslant k_j$
      $left := l_p^j$
      $right := r_p^j$
      $(\mathtt{S}, \tau) := EvalLeft(\mathbf{push}(\emptyset, (l_p^j, m)), \rho)$
    else
      $(\mathtt{S}, \tau) := (\emptyset, \bot)$
    $status :=$ parse_left

} where $(\rho, j, p) = f(cs)$

The parsing of the left-hand side of the rule is done via a stack that contains a list of symbolic terms to parse paired with their bitstring value. Function *EvalLeft* consumes elements of the stack S as long as parsing

```
EvalLeft(S, τ)
  if S = ∅ then return(S, τ)
  (t, m) := top(S)
  if t = init return(if m = λ then (∅, τ) else (∅, ⊥))
  if τ(t) ≠ ⊥ return(if τ(t) = m then (∅, τ) else (∅, ⊥))
  if t = X ∧ type(X) = type(m)
    τ(t) := m
    S := pop(S)
    return(EvalLeft(S, τ))
  if t = ⟨t1, t2⟩ ∧ type(m) = pair
    τ(t) := m
    S := pop(S)
    S := push(S, (t2, right(m)))
    S := push(S, (t1, left(m)))
    return(EvalLeft(S, τ))
  if t = {t1}_{t2}^{t3} ∧ type(m) = encrypt
    τ(t) := m
    return(S, τ)
  if t = [t1]_{t2}^{t3} ∧ type(m) = signature
    if τ(t1) = ⊥
      S := push(S, (t1, msg(m)))
      return(EvalLeft(S, τ))
    if τ(t1) = msg(m)
      τ(t) := m
      return(S, τ)
  return(∅, ⊥)
```

Fig. 8.7. The *EvalLeft*$(\mathtt{S}, \tau)$ function

is possible without invoking any primitive. If the top element of S needs the invocation of a cryptographic primitive to be parsed, then *EvalLeft* terminates, so that the appropriate action can be scheduled. In other words, once *EvalLeft* has terminated, either an error was detected ($\tau = \bot$), or the stack is empty, which means that the parsing procedure is completed, or the top element of the stack requires an invocation to an external primitive to be resolved.

Function *EvalLeft* is defined recursively. If the stack is empty it terminates immediately. Otherwise it retrieves the top element and checks whether we are expecting an initiation. In such case the input bitstring should be empty. If we are not expecting an initiation, then the function checks whether the symbolic term was parsed already before, and in such case the bitstrings should coincide, otherwise an error is returned. If the symbolic term was not parsed before, then several cases are distinguished.

In particular in each case it is checked whether the bitstring is of the correct type. If no case applies, then an error is returned. Variables are immediate, while pairs are handled easily by creating two subproblems that are added to the stack. In both cases the evaluation of the stack continues with a recursive call. Encryption requires the invocation of a decryption primitive. Thus, function τ is updated and the function terminates. For signatures it is necessary to parse the signed message before verifying the signature. Thus, if the signed message is not parsed yet, then it is inserted into the stack on top of the whole term, which should be reevaluated later. Otherwise, τ is updated and the procedure terminates.

If we consider the definition of the *EvalLeft* function, we can note that function τ is updated only when the term and the bitstrings are compatible. This means, for example, that they must have the same type (we recall that given a bitstring, it is possible to recover its type: key, agent's identity, pair, and so on) and if the τ assigns a value to the current term t, then such value must be the current bitstring. If the bitstring and the τ do not match, then we are faced to two different bitstrings that should represent the same symbolic term. This means, for example, that we have generated two ciphertexts for the same plaintext instead of copying the first one. On the contrary, if we find some problem during the evaluation of the left-hand side of a rule, then we put τ to $\perp$ denoting in this way that an error occurred.

Once the stack is empty and $\tau \neq \perp$, then we have that for each term t in the left-hand side of the current rule, $\tau(t) \neq \perp$ and, t has been verified successfully and we have associated each term with the actual bitstring that represents t in the message m of $send(cs, m)$.

If we return from *EvalLeft* with a stack that is not empty, then this means that the top element of the stack requires the invocation of a cryptographic primitive to be verified. In particular, we need only two primitives: the decryption and the verification operation. We do not need other cryptographic primitives, since we are not generating nonces, encryptions and signatures. These operations are used only when we build up a message and this can happen only in the right-hand side of the current rule.

The next two actions deal with the decryption primitive, which is invoked when the top of the stack contains an encryption. We know already that τ assigns a value to each agent variable. The result of the decryption is pushed into the stack, and then function *EvalLeft* is invoked again.

Output $get\_decrypt(\eta, m)$
  Precondition:
    $status = \mathsf{parse\_left}$
    $\mathbf{top}(\mathtt{S}) = (\{t_1\}^{t_2}_{\mathsf{ek}(A)}, m)$
    $\tau(A) = \eta$
  Effect:
    $pending := t_1$
    $\mathtt{S} := \mathbf{pop}(\mathtt{S})$
    $status := \mathsf{parse\_left\_wait}$

Input $ret\_decrypt(\eta, m)$
  Effect:
    $(\mathtt{S}, \tau) := EvalLeft(\mathbf{push}(\mathtt{S}, (pending, m)), \tau)$
    $status := \mathsf{parse\_left}$

In particular, the $get\_decrypt(\eta, m)$ action is enabled when we are parsing the left-hand side of a role ($status = \mathsf{parse\_left}$), the top of the stack contains a term that is an encryption ($\{t_1\}^{t_2}_{\mathsf{ek}(A)}$), the actual bitstring associated to the term is exactly m and the encryption key used to generate the encryption belongs to agent η. The effect of the action is to save into the state variable *pending* the term t_1 corresponding to the encrypted message, to remove the top element from the stack, and to update *status* to the value $\mathsf{parse\_left\_wait}$. We change the value of *status* to avoid the enabling of another cryptographic primitive invocation. Once we receive the decryption from the encryption automaton through the $ret\_decrypt(\eta, m)$ action, we push into the stack the received value m associated to the pending term *pending* (that is t_1), we invoke again the *EvalLeft* function on the updated stack and we update *status* to $\mathsf{parse\_left}$ to enable again the parsing of the left-hand side.

The next two actions deal with signature verification. If the verification is unsuccessful, then an error occurs by setting τ to $\perp$.

Output $get\_verify\_sign(\eta, m)$
  Precondition:
    $status = \mathsf{parse\_left}$
    $\mathbf{top}(\mathtt{S}) = ([t_1]^{t_2}_{\mathsf{sk}(A)}, m)$
    $\tau(A) = \eta$
  Effect:
    $\mathtt{S} := \mathbf{pop}(\mathtt{S})$
    $status := \mathsf{parse\_left\_wait}$

Input $ret\_verify\_sign(\eta, b)$
Effect:
if $b = T$
$(\mathtt{S}, \tau) := EvalLeft(\mathtt{S}, \tau)$
else
$\tau := \bot$
$status :=$ parse_left

The verification of a signature is very similar to the decryption of a ciphertext: the $get\_verify\_sign(\eta, m)$ action is enabled when we are parsing the left-hand side of a role ($status =$ parse_left), the top of the stack contains a term that is a signature ($[t_1]^{t_2}_{\mathsf{sk}(A)}$), the actual bitstring associated to the term is exactly m and the signature key used to generate the encryption belongs to agent η. The effect of the action is to remove the top element from the stack and to update *status* to the value parse_left_wait. Again, we change the value of *status* to avoid the enabling of another cryptographic primitive invocation. Once we receive the validity of the signature from the signature automaton through the $ret\_verify\_sign(\eta, b)$ action, we check if the signature is valid or invalid. If it is valid ($b = T$), then we invoke again the *EvalLeft* function on the stack and we update *status* to parse_left to enable again the parsing of the left-hand side. If the signature is not valid ($b = F$), then we set τ to $\bot$ to denote the fact that an error occurred.

The next action *fail_step* is enabled when an error occurs while parsing. In this case the status variable is set to create, thus ignoring the rule under application. On the other hand, if no error occurs and the stack is empty, then the parsing is complete and we can start to work on the right-hand side of the rule.

Internal *fail_step*
Precondition:
$status =$ parse_left
$\tau = \bot$
Effect:
$status =$ create

```
EvalRight(t, τ)
 if t = stop then return(τ)
 if τ(t) ≠ ⊥ then return(τ)
 if t = X then return(τ)
 if t = ⟨t1, t2⟩
  τ := EvalRight(t1, τ)
  τ := EvalRight(t2, τ)
  if τ(t1) ≠ ⊥ ∧ τ(t2) ≠ ⊥ then τ(t) := ⟨τ(t1), τ(t2)⟩
  return(τ)
 if t = {t1}^{t2}_{ek(A)} ∨ t = [t1]^{t2}_{sk(A)}
  τ := EvalRight(t1, τ)
 return(τ)
```

Fig. 8.8. The *EvalRight*(t, τ) function

Internal *parse_right*
Precondition:
status = parse_left
$\tau \neq \bot$
$\mathsf{S} = \emptyset$
Effect:
$\tau :=$ *EvalRight*$(right, \tau)$
status := parse_right

The evaluation of a term t is carried out in two phases. First, the definition of τ is extended on subterms of t, leaving unresolved only those places that require the invocation of an external primitive, and then invoking a primitive for one of the unresolved places. These two phases are alternated until term t is resolved. Function *EvalRight* carries out the first phase. Variables are not evaluated since, in case they are not defined, they require the generation of a nonce. Pairs are resolved completely, while for signatures and encryptions only the subterm corresponding to the message is evaluated since labels are used just to distinguish terms syntactically and agent variables are already defined by construction. Observe that the function *EvalRight* does not operate on t if τ is defined already. This ensures that terms that have appeared already before are not recomputed.

The *FirstUnresolved* function takes the outcome of *EvalRight* and identifies an unresolved place that can be resolved by invoking an external primitive. It is essentially a post-visit of the parse tree of t, searching for an undefined term whose subterms are defined. The next three pairs of actions are the invocations to the three primitives that resolve unresolved

$FirstUnresolved(t, \tau)$
if $t = \mathsf{stop}$ **then return**$(\bot)$
if $\tau(t) \neq \bot$ **then return**$(\bot)$
if $t = X$ **then return**(t)
if $t = \langle t_1, t_2 \rangle$
  if $\tau(t_1) = \bot$
    return$(FirstUnresolved(t_1, \tau))$
  else
    return$(FirstUnresolved(t_2, \tau))$
if $t = \{t_1\}^{t_2}_{\mathsf{ek}(A)} \vee t = [t_1]^{t_2}_{\mathsf{sk}(A)}$
  if $\tau(t_1) = \bot$
    return$(FirstUnresolved(t_1, \tau))$
  else
    return(t)

Fig. 8.9. The $FirstUnresolved(t, \tau)$ function

places. The get action retrieves the arguments from the unresolved term and saves on *pending* the term that should denote the returned value; the return action updates τ.

Output $get\_encrypt(\eta, m)$
  Precondition:
    $status = \mathsf{parse\_right}$
    $FirstUnresolved(right, \tau) = \{t_1\}^{t_2}_{\mathsf{ek}(A)}$
    $(\eta, m) = (\tau(A), \tau(t_1))$
  Effect:
    $pending := \{t_1\}^{t_2}_{\mathsf{ek}(A)}$
    $status := \mathsf{parse\_right\_wait}$

Input $ret\_encrypt(\eta, m)$
  Effect:
    $\tau(pending) := m$
    $\tau := EvalRight(right, \tau)$
    $status := \mathsf{parse\_right}$

Output $get\_sign(\eta, m)$
  Precondition:
    $status = \mathsf{parse\_right}$
    $FirstUnresolved(right, \tau) = [t_1]^{t_2}_{\mathsf{sk}(A)}$
    $(\eta, m) = (\tau(A), \tau(t_1))$
  Effect:
    $pending := [t_1]^{t_2}_{\mathsf{sk}(A)}$
    $status := \mathsf{parse\_right\_wait}$

Input $ret\_sign(\eta, m)$
  Effect:
    $\tau(pending) := m$
    $\tau := EvalRight(right, \tau)$
    $status :=$ parse_right

Output $get\_nonce(\eta)$
  Precondition:
    $status =$ parse_right
    $X_A^j = FirstUnresolved(right, \tau) \in \mathsf{X}_n(A)$
    $\eta = \tau(A)$
  Effect:
    $pending := X_A^j$
    $status :=$ parse_right_wait

Input $ret\_nonce(\eta, m)$
  Effect:
    $\tau(pending) := m$
    $\tau := EvalRight(right, \tau)$
    $status :=$ parse_right

Finally, when the evaluation of the rule is completed successfully, that is the value of the right-hand side of the rule is computed, function f is updated with the result of the computation and we are ready to choose the next action to perform.

Internal end_step
  Precondition:
    $status =$ parse_right $\wedge\ FirstUnresolved(right, \tau) = \bot$
  Effect:
    $history := history \vdash end\_step(\tau(right))$
    $f(csid) := (\tau, j, p+1)$ where $(\rho, j, p) = f(csid)$
    $status :=$ create

This, together with the definition of the encryption automaton $\mathcal{E}_k(\mathbb{A})$, the signature automaton $\mathcal{S}_k(\mathbb{A})$, and the nonce generator $NG_k(\mathbb{A})$, completes the definition of the concrete model. We should now move to the abstract model, though we have several problems to solve.

First, it is evident that Lemma 1 of [41] is not completely correct and that some arguments about non guessing data for which we have no information is not possible. In fact, it seems that there is a missing condition in the hypothesis, that is that the message m should be built out of atomic

elements that are either deducible atomically from S or that are subterms of terms in S. If not, then we could simply guess a nonce or a key of the agents, leading to a non-deducible message that does not satisfy the conclusions of Lemma 1. This argument is completely absent in [41]. However, we can fix the statement of Lemma 1 making the condition of the statement more restrictive: instead of simply requiring $M \not\vdash m$, we can also need that m is built out of atoms of elements of M, that is, m is generated starting from terms that occur in M.

Second, the action names change completely in the symbolic model, so it is likely that somewhere we must hide actions and rely on the actual definition of the simulation relation to state correspondence between symbolic and concrete traces. We have to recover the trick to communicate information to the nonce generator to avoid duplicated nonces, and finally we have to find the correct order of abstraction, or state theorems that allow us to combine separate abstractions into a unique one.

Recovering and sending adversary's generated nonces

One possible way to recover the set of nonces generated by the adversary is to modify the *adv* function in the following way: instead of returning a pair $(action, message)$, it returns a triple $(action, message, extr\_nonces)$ where *extr_nonces* is the set of all nonces created by *adv* during the choice of *action* and *message*. Note that we can expect that *extr_nonces* is meaningful, that is it is not the empty set, only when *action* is *send*. To send the set *extr_nonces* to the nonce generator, we can add to the *PAdv* automaton an output action $used\_nonces(N)$, $N \subseteq \{0,1\}^k$, defined as:

Output $used\_nonces(N)$
  Precondition:
    $N = extr\_nonces$
  Effect:
    $extr\_nonces := \bot$

Moreover, we need to modify *corrupt*, *send* and *new* actions adding the precondition $extr\_nonces = \bot$. In this way, we force the execution of $used\_nonces(N)$, when enabled, before the execution of *corrupt*, *send* and *new*. This is mandatory in particular for the *send* action: if we do not impose that we send nonces before enabling the *send* action, then it is still possible that the nonce generator chooses a nonce that belongs to the set of

extracted nonces *extr_nonces*. We also impose that *extr_nonces* has initial value $\perp$.

This first solution seems to be acceptable, but it presents some problem: in particular, we can not be completely sure that *extr_nonces* contains all nonces that are inside *message*. In fact, it is possible that *adv* generates at random a bitstring that is the encryption of a nonce. In this case, the encrypted nonce is not known to *adv* and thus we can not find it into *extr_nonces*.

Another possible way to recover the set of nonces generated by the adversary is to extract it after the evaluation of the left-hand side of the current rule and to send such set to the nonce generator before starting the evaluation of the right-hand side of the rule. We obtain this adding the state variable *extr_nonces* with initial value $\perp$ and replacing the *parse_right* action with two new actions: the output action *used_nonces*(N), where $N \subseteq \{0,1\}^k$, and the internal action *parse_right* defined as:

Output *used_nonces*(N)
  Precondition:
    $N = extr\_nonces$
  Effect:
    $extr\_nonces := \perp$
    $\tau := EvalRight(right, \tau)$
    $status :=$ parse_right

Internal *parse_right*
  Precondition:
    $status =$ parse_left
    $\tau \neq \perp$
    $\mathsf{S} = \emptyset$
    $extr\_nonces = \perp$
  Effect:
    $extr\_nonces := extractNonces(left, \tau)$

The *extractNonces*(t, τ), that is depicted in Figure 8.10, is a function that recursively parses subterms of t collecting all nonces it finds. If t' is a subterm of t that denotes a nonce, the actual value we collect is $\tau(t')$. It is easy to see that the cardinality of the set returned by the *extractNonces*(t, τ) function is bounded by the size of t:

Proposition 8.11. *Let t be a term and τ be a mapping from terms to bitstrings. Let N be the output of extractNonces(t, τ).*

Then, $|N| \leqslant \mathrm{size}(t)$.

```
extractNonces(t, τ)
 if t = X
  return({τ(t)})
 if t = ⟨t1, t2⟩
  return(extractNonces(t1, τ) ∪ extractNonces(t2, τ))
 if t = {t1}^{t2}_{ek(A)} ∨ t = [t1]^{t2}_{sk(A)}
  return(extractNonces(t1, τ))
 return(∅)
```

Fig. 8.10. The $extractNonces(t, \tau)$ function

Proof. We prove the statement using a classical inductive proof on the structure of the term t:

case $t = X$: by definition of $\text{size}(\cdot)$, it follows that $\text{size}(X) = 1$ and by definition of $extractNonces(t, \tau)$, $|N| = |\{\tau(X)\}| = 1$ and thus $|N| \leqslant \text{size}(X) = \text{size}(t)$;

case $t = \langle t_1, t_2 \rangle$: by inductive hypothesis, we have that $|N_1| \leqslant \text{size}(t_1)$ and $|N_2| \leqslant \text{size}(t_2)$ where for $i = 1, 2$, $N_i = extractNonces(t_i, \tau)$. By definition of $extractNonces(t, \tau)$, it follows that $|N| = |extractNonces(t, \tau)| \leqslant |extractNonces(t_1, \tau)| + |extractNonces(t_2, \tau)| \leqslant \text{size}(t_1) + \text{size}(t_2) \leqslant \text{size}(t_1) + \text{size}(t_2) + 1 = \text{size}(t)$;

case $t = \{t_1\}^{t_2}_{\mathsf{ek}(A)}$: by inductive hypothesis, we have that $|N_1| \leqslant \text{size}(t_1)$ and thus, by definition of $extractNonces(t, \tau)$, we have that $|N| = |extractNonces(t, \tau)| = |extractNonces(t_1, \tau)| \leqslant \text{size}(t_1) \leqslant \text{size}(t_1) + 1 = \text{size}(t)$;

case $t = [t_1]^{t_2}_{\mathsf{sk}(A)}$: by inductive hypothesis, we have that $|N_1| \leqslant \text{size}(t_1)$ and thus, by definition of $extractNonces(t, \tau)$, we have that $|N| = |extractNonces(t, \tau)| = |extractNonces(t_1, \tau)| \leqslant \text{size}(t_1) \leqslant \text{size}(t_1) + 1 = \text{size}(t)$;

other cases: by definition of $\text{size}(\cdot)$, it follows that $\text{size}(t) = 1$ and by definition of $extractNonces(t, \tau)$, it follows that $|N| = |extractNonces(t, \tau)| = |\emptyset| \leqslant \text{size}(t)$. □

Recovering and sending adversary's generated ciphertexts

Similarly to the case of nonces, it is possible to recover all adversary's generated ciphertexts and to send them to the encryption oracle. We adopt the same approach of the previous case: we extract the ciphertexts after ending the parsing of the left-hand side of the current rule and we provide them to the encryption oracle before the evaluation of the right-hand side of the

rule. We obtain this adding the state variable *extr_ciphers* with initial value $\perp$ and replacing the *parse_right* and *used_nonces*(N) actions with the following actions: a modified version of *used_nonces*(N) that simply reset the value of *extr_nonces*, a new action *used_ciphers*(C), $C \subseteq \mathcal{C}^{\eta}.c$ that outputs the set C of extracted ciphertexts, and the internal action *parse_right* that recover used nonces and ciphertexts from the current message. The three actions are defined as:

Output *used_nonces*(N)
  Precondition:
    $N = extr\_nonces$
  Effect:
    $extr\_nonces := \perp$

Output *used_ciphers*(C)
  Precondition:
    $extr\_nonces = \perp$
    $C = extr\_ciphers$
  Effect:
    $extr\_ciphers := \perp$
    $\tau := EvalRight(right, \tau)$
    $status :=$ parse_right

Internal *parse_right*
  Precondition:
    $status =$ parse_left
    $\tau \neq \perp$
    $\mathsf{S} = \emptyset$
    $extr\_nonces = \perp$
    $extr\_ciphers = \perp$
  Effect:
    $extr\_nonces := extractNonces(left, \tau)$
    $extr\_ciphers := extractCiphertexts(left, \tau)$

The *extractCiphertexts*(t, τ), that is depicted in Figure 8.11, is a function that recursively parses subterms of t collecting all ciphertexts it finds. If t' is a subterm of t that denotes a ciphertext, the actual value we collect is $\tau(t')$. It is easy to see that the cardinality of the set returned by the *extractCiphertexts*(t, τ) function is bounded by the size of t:

Proposition 8.12. *Let t be a term and τ be a mapping from terms to bitstrings. Let CT be the output of extractCiphertexts(t, τ).*

Then, $|CT| \leqslant \mathrm{size}(t)$.

extractCiphertexts(t, τ)
if $t = \langle t_1, t_2 \rangle$
return(*extractCiphertexts*$(t_1, \tau) \cup$ *extractCiphertexts*(t_2, τ))
if $t = \{t_1\}^{t_2}_{\mathsf{ek}(A)}$
return($\{\tau(t)\} \cup$ *extractCiphertexts*(t_1, τ))
if $t = [t_1]^{t_2}_{\mathsf{ek}(A)}$
return(*extractCiphertexts*(t_1, τ))
return($\emptyset$)

Fig. 8.11. The *extractCiphertexts*(t, τ) function

Proof. We prove the statement using a classical inductive proof on the structure of the term t:

case $t = \langle t_1, t_2 \rangle$: by inductive hypothesis, we have that for $i = 1, 2$, $|CT_i| \leqslant \text{size}(t_i)$ where $CT_i =$ *extractCiphertexts*(t_i, τ). This implies, by definition of *extractCiphertexts*(t, τ), that $|CT| \leqslant |$*extractCiphertexts*$(t_1, \tau)| + |$*extractCiphertexts*$(t_2, \tau)| \leqslant \text{size}(t_1) + \text{size}(t_2) \leqslant \text{size}(t_1) + \text{size}(t_2) + 1 = \text{size}(t)$;

case $t = \{t_1\}^{t_2}_{\mathsf{ek}(A)}$: by inductive hypothesis, we have that $|CT_1| \leqslant \text{size}(t_1)$ and thus, by definition of *extractCiphertexts*(t, τ), we have that $|CT| = |\{\tau(t)\} \cup$ *extractCiphertexts*$(t_1, \tau)| \leqslant \text{size}(t_1) + 1 = \text{size}(t)$;

case $t = [t_1]^{t_2}_{\mathsf{sk}(A)}$: by inductive hypothesis, we have that $|CT_1| \leqslant \text{size}(t_1)$ and thus, by definition of *extractCiphertexts*(t, τ), we have that $|CT| = |$*extractCiphertexts*$(t_1, \tau)| \leqslant \text{size}(t_1) \leqslant \text{size}(t_1) + 1 = \text{size}(t)$;

other cases: by definition of size$(\cdot)$, it follows that $\text{size}(t) = 1$ and by definition of *extractCiphertexts*(t, τ), it follows that $|CT| = |\emptyset| \leqslant \text{size}(t)$. □

Recovering and sending adversary's generated signatures

Similarly to the previous cases, it is possible to recover all adversary's generated signatures and to send them to the signature oracle. We adopt the same approach of the previous cases: we extract the signatures after ending the parsing of the left-hand side of the current rule and we provide them to the signature oracle before the evaluation of the right-hand side of the rule. We obtain this adding the state variable *extr_signatures* with initial value $\perp$, modifying the *used_ciphers*(C) and *parse_right* actions and adding a new action *used_signatures*(S), $S \subseteq \mathcal{C}^{\eta}.s$, that outputs the set S of used signatures:

```
extractSignatures(t, τ)
 if t = ⟨t1, t2⟩
  return(extractSignatures(t1, τ) ∪ extractSignatures(t2, τ))
 if t = {t1}^{t2}_{ek(A)}
  return(extractSignatures(t1, τ))
 if t = [t1]^{t2}_{ek(A)}
  return({τ(t)} ∪ extractSignatures(t1, τ))
 return(∅)
```

Fig. 8.12. The $extractSignatures(t, \tau)$ function

Output $used\_ciphers(C)$
Precondition:
$extr\_nonces = \bot$
$C = extr\_ciphers$
Effect:
$extr\_ciphers := \bot$

Internal $parse\_right$
Precondition:
$status = \mathsf{parse\_left}$
$\tau \neq \bot$
$\mathsf{S} = \emptyset$
$extr\_nonces = \bot$
$extr\_ciphers = \bot$
$extr\_signatures = \bot$
Effect:
$extr\_nonces := extractNonces(left, \tau)$
$extr\_ciphers := extractCiphertexts(left, \tau)$
$extr\_signatures := extractSignatures(left, \tau)$

Output $used\_signatures(S)$
Precondition:
$extr\_nonces = \bot$
$extr\_ciphers = \bot$
$S = extr\_signatures$
Effect:
$extr\_signatures := \bot$
$\tau := EvalRight(right, \tau)$
$status := \mathsf{parse\_right}$

The $extractSignatures(t, \tau)$, that is depicted in Figure 8.12, is a function that recursively parses subterms of t collecting all signatures it finds. If

t' is a subterm of t that denotes a signature, the actual value we collect is $\tau(t')$. It is easy to see that the cardinality of the set returned by the $\mathit{extractSignatures}(t, \tau)$ function is bounded by the size of t:

Proposition 8.13. *Let t be a term and τ be a mapping from terms to bitstrings. Let S be the output of $\mathit{extractSignatures}(t, \tau)$.*

Then, $|S| \leqslant \mathrm{size}(t)$.

Proof. We prove the statement using a classical inductive proof on the structure of the term t:

case $t = \langle t_1, t_2 \rangle$: by inductive hypothesis, we have that for $i = 1, 2$, $|S_i| \leqslant \mathrm{size}(t_i)$ where $S_i = \mathit{extractSignatures}(t_i, \tau)$. This implies, by definition of $\mathit{extractSignatures}(t, \tau)$, that $|S| \leqslant |\mathit{extractSignatures}(t_1, \tau)| + |\mathit{extractSignatures}(t_2, \tau)| \leqslant \mathrm{size}(t_1) + \mathrm{size}(t_2) \leqslant \mathrm{size}(t_1) + \mathrm{size}(t_2) + 1 = \mathrm{size}(t)$;

case $t = \{t_1\}^{t_2}_{\mathsf{ek}(A)}$: by inductive hypothesis, we have that $|S_1| \leqslant \mathrm{size}(t_1)$ and thus, by definition of $\mathit{extractSignatures}(t, \tau)$, we have that $|S| = |\mathit{extractSignatures}(t_1, \tau)| \leqslant \mathrm{size}(t_1) \leqslant \mathrm{size}(t_1) + 1 = \mathrm{size}(t)$;

case $t = [t_1]^{t_2}_{\mathsf{sk}(A)}$: by inductive hypothesis, we have that $|S_1| \leqslant \mathrm{size}(t_1)$ and thus, by definition of $\mathit{extractSignatures}(t, \tau)$, we have that $|S| = |\{\tau(t)\} \cup \mathit{extractSignatures}(t_1, \tau)| \leqslant \mathrm{size}(t_1) + 1 = \mathrm{size}(t)$;

other cases: by definition of $\mathrm{size}(\,\cdot\,)$, it follows that $\mathrm{size}(t) = 1$ and by definition of $\mathit{extractSignatures}(t, \tau)$, it follows that $|S| = |\emptyset| \leqslant \mathrm{size}(t)$. □

Adversary and Protocol: the Formal Model

We provide here a description of the formal model. This model is highly nondeterministic, and therefore does not need to create the next action to perform. Furthermore, there is no need to go through several intermediate operations. Messages are ground terms of the algebra defined in [41] and discussed in Section 8.1. There is a variable H that contains the knowledge of the adversary. Initially the knowledge is given by the public keys of the agents, which is much more than what is known by the adversary in the concrete model. The substitution function of the state of each role, denoted by σ, is just a partial assignment of ground terms to variables. In the description below we compose substitution functions; however, given that variables are associated to ground terms, we could take unions as well, provided that substitution functions are compatible.

There are only three transitions. Corruption occurs only at the beginning, and this is ensured by the state component h, which we should change. A new instance of a role creates the instance and adds the knowledge of all nonces. Sending a message produces an update of the substitution function if the message sent matches what the role is expecting. The message should be derivable from the knowledge of the adversary.

Output $corrupt(a_1 \dots a_l)$
Precondition:
$h = \lambda$

Effect:
$h := corrupt(a_1 \dots a_l)$
$H := H \cup \mathbf{kn}(a_1) \cup \dots \cup \mathbf{kn}(a_l)$

Output $new(i, a_1 \dots a_l)$
Effect:
$H := H \cup \{(i, a_1, \dots, a_l)\}$
$ASId := ASId \cup \{as\}$
$F := F \cup \{as, (\sigma, i, 1)\}$
where $as = (|ASId| + 1, i, (a_1 \dots a_l))$
$\sigma = \{(A_j, a_j) \mid 1 \leqslant j \leqslant l\} \cup \{(X^j_{a_i}, n(a_i, j, as)) \mid j \in \mathbb{N}\}$

Output $send(as, m)$
Precondition:
$H \vdash m$
Effect:
if $p \leqslant k_j \wedge \exists \theta \mid m = (\sigma \circ \theta)(l^j_p)$
$H := H \cup \{(\sigma \circ \theta)(r^j_p)\}$
$F(as) := (\sigma \circ \theta, j, p + 1)$
where $(\sigma, j, p) = F(as)$

Adversary and Protocol: the Mixed Model

We now consider the concrete model and add the fields of the abstract model, while at the same time we keep a mapping that relates formal traces with concrete traces. In a second stage we will need to impose restrictions on the transitions in the sense that we should produce only deducible terms. We will state explicitly what actions and functions are changed.

No change for *create_action*.

```
update_ag_names(c, η_1 ... η_l)
 for i = 1 to l
  if c^{-1}(η_i) = ∅
   c(g_j) := η_i where j = min{k | c(g_k) = ⊥}
 return(c)
```

Fig. 8.13. The *update_ag_names*$(c, \eta_1 \dots \eta_l)$ function

Internal *create_action*
Precondition:
status = create
Effect:
$(action, message) := adv(history)$
status := act

Action *corrupt* needs to construct a symbolic name for each agent. This is done by invoking function *update_ag_names*, which selects a symbolic agent for each new concrete agent. Then the symbolic term is constructed and the database H is updated. The concretization function c is used to know which concrete agents have already an abstract counterpart. The construction is done in such a way that each concrete agent has at most one counterpart and in particular exactly one counterpart whenever c^{-1} is used. The subsequent actions need to update the concretization map c.

Output $corrupt(\eta_1 \dots \eta_l)$
Precondition:
$history = \lambda$
$action = corrupt$
$message = (\eta_1 \dots \eta_l)$
Effect:
$history := corrupt(\eta_1 \dots \eta_l)$
$action := \bot$
$agentsToCorrupt := \eta_1 \dots \eta_l$
status := get_sig_key
$c := update\_ag\_names(c, \eta_1 \dots \eta_l)$
$symbolic := corrupt(c^{-1}(\eta_1) \dots c^{-1}(\eta_l))$
$H := H \cup \mathbf{kn}(c^{-1}(\eta_1)) \cup \dots \cup \mathbf{kn}(c^{-1}(\eta_l))$

Output $get\_corrupt\_sign(\eta)$
  Precondition:
    $\text{head}(agentsToCorrupt) = \eta$
    $status = \mathsf{get\_sig\_key}$
  Effect:
    $history := history \vdash get\_corrupt\_sign(\eta)$
    $status := \mathsf{wait\_sig\_key}$

Input $ret\_corrupt\_sign(\eta, public, secret)$
  Effect:
    $history := history \vdash ret\_corrupt\_sign(\eta, public, secret)$
    $vk(\eta) := public$
    $sk(\eta) := private$
    $status := \mathsf{get\_enc\_key}$
    $c(\mathsf{sk}(c^{-1}(\eta))) := private$
    $c(\mathsf{vk}(c^{-1}(\eta))) := public$

Output $get\_corrupt\_encrypt(\eta)$
  Precondition:
    $\text{head}(agentsToCorrupt) = \eta$
    $status = \mathsf{get\_enc\_key}$
  Effect:
    $history := history \vdash get\_corrupt\_encrypt(\eta)$
    $status := \mathsf{wait\_enc\_key}$

Input $ret\_corrupt\_encrypt(\eta, public, secret)$
  Effect:
    $history := history \vdash ret\_corrupt\_encrypt(\eta, public, secret)$
    $ek(\eta) := public$
    $dk(\eta) := private$
    $agentsToCorrupt := \text{tail}(agentsToCorrupt)$
    $status := \textbf{if } agentsToCorrupt \neq \lambda \textbf{ then } \mathsf{get\_sig\_key} \textbf{ else } \mathsf{create}$
    $c(\mathsf{dk}(c^{-1}(\eta))) := private$
    $c(\mathsf{ek}(c^{-1}(\eta))) := public$

Action *new*, similarly to *corrupt*, needs to construct symbolic names for the new agents. Once again the concretization map is updated for the new keys. We also update the F function.

Output $new(i, \eta_1 \dots \eta_l)$
Precondition:
$action = new$
$message = (i, \eta_1 \dots \eta_l)$
Effect:
$history := history \vdash new(i, \eta_1, \dots, \eta_l)$
$action := \bot$
$c := update\_ag\_names(c, \eta_1 \dots \eta_l)$

$$\left.\begin{array}{l} CSId := CSId \cup \{cs\} \\ f := f \cup \{(cs, (\tau, i, 1))\} \\ ASId := ASId \cup \{as\} \\ F := F \cup \{(as, (\sigma, i, 1))\} \end{array}\right\} \text{where } \begin{array}{rl} cs & = (|CSId| + 1, i, (\eta_1 \dots \eta_l)) \\ \tau & = \{(A_j, \eta_j) \mid 1 \leqslant j \leqslant l\} \\ as & = (|ASId| + 1, i, (c^{-1}(\eta_1) \dots c^{-1}(\eta_l))) \\ \sigma & = \{(A_j, c^{-1}(\eta_j)) \mid 1 \leqslant j \leqslant l\} \cup \\ & \quad \{(X^j_{c^{-1}(\eta_i)}, n(c^{-1}(\eta_i), j, as)) \mid j \in \mathbb{N}\} \end{array}$$

$agents\_to\_init := (\eta_1 \dots \eta_l) - \{\eta \mid ek(\eta) \neq \bot\}$
$status :=$ **if** $agents\_to\_init \neq \lambda$ **then** get_sig_key **else** create
$symbolic := new(i, c^{-1}(\eta_1) \dots c^{-1}(\eta_l))$
$H := H \cup \{(i, c^{-1}(\eta_1) \dots c^{-1}(\eta_l))\}$

Output $get\_public\_sign(\eta)$
Precondition:
$\mathrm{head}(agents\_to\_init) = \eta$
$status =$ get_sig_key
Effect:
$history := history \vdash get\_public\_sign(\eta)$
$status :=$ wait_sig_key

Input $ret\_public\_sign(\eta, public)$
Effect:
$history := history \vdash ret\_public\_sign(\eta, public)$
$vk(\eta) := public$
$status :=$ get_enc_key
$c(\mathsf{vk}(c^{-1}(\eta))) := public$

Output $get\_public\_encrypt(\eta)$
Precondition:
$\mathrm{head}(agents\_to\_init) = \eta$
$status =$ get_enc_key
Effect:
$history := history \vdash get\_public\_encrypt(\eta)$
$status :=$ wait_enc_key

Input $ret\_public\_encrypt(\eta, public)$
  Effect:
    $history := history \vdash ret\_public\_encrypt(\eta, public)$
    $ek(\eta) := public$
    $agents\_to\_init := \mathrm{tail}(agents\_to\_init)$
    $status :=$ **if** $agents\_to\_init \neq \lambda$ **then** get_sig_key **else** create
    $c(\mathsf{ek}(c^{-1}(\eta))) := public$

Action *send*, again, is much more complex to treat. We cannot compute the symbolic input message before completing the elaboration of m. For this reason, we need to keep track of m, of the applied rule, and of the pending symbolic session. We also need to compute σ along the way. We will use the symbolic term $\perp$ whenever the message m cannot be parsed.

Output $send(cs, m)$
  Precondition:
    $action = send$
    $message = (cs, m)$
  Effect:
    $history := history \vdash send(cs, m)$
    $action := \perp$
    $csid := cs$
    if $p \leqslant k_j$
      $left := l_p^j$
      $right := r_p^j$
      $(\mathsf{S}, \tau, \sigma, c) := EvalLeft(\mathbf{push}(\emptyset, (l_p^j, m)), \rho, \phi, c)$
    else
      $(\mathsf{S}, \tau) := (\emptyset, \perp)$
    $status :=$ parse_left

where $(\rho, j, p) = f(cs)$, $(\phi, j, p) = F(c^{-1}(cs))$

In the parsing procedure we need to identify the symbolic nonces that are sent. In particular, either the adversary sends a nonce that exists already, or it generates one. For the purpose we need to add two arguments to the function and two returned values. Once the parsing is finished the concretization map needs to be updates. Note that in handling nonces we use a minimum operator over sets of symbolic nonces. This is to ensure determinism. We assume implicitly an ordering on the set of symbolic nonces. The actual ordering is irrelevant. Later we will ensure that there is at most one selectable nonce.

It is interesting to observe that the two *EvalLeft* functions we use in the concrete and in the mixed model returns the same value on common outputs:

EvalLeft$(\mathtt{S}, \tau, \theta, c)$
 if $\mathtt{S} = \emptyset$ **then return**$(\mathtt{S}, \tau, \theta, c)$
 $(t, m) :=$ **top**$(\mathtt{S})$
 if $t = \mathsf{init}$ **return**(**if** $m = \lambda$ **then** $(\emptyset, \tau, \theta, c)$ **else** $(\emptyset, \bot, \theta, c)$)
 if $\tau(t) \neq \bot$ **return**(**if** $\tau(t) = m$ **then** $(\emptyset, \tau, \theta, c)$ **else** $(\emptyset, \bot, \theta, c)$)
 if $t = X \wedge type(X) = type(m)$
  Note this below works only with nonce variables.
  $\tau(t) := m$
  $\mathtt{S} :=$ **pop**$(\mathtt{S})$
  if $\exists_{a,i,s} c(n(a,i,s)) = m$
   $\theta(X) := \min\{n(a,i,s) \mid c(n(a,i,s)) = m\}$
  else
   $\theta(X) := \min\{n(adv,i,\bot) \mid c(n(adv,i,\bot)) = \bot\}$
  return(*EvalLeft*$(\mathtt{S}, \tau, \theta, c)$)
 if $t = \langle t_1, t_2 \rangle \wedge type(m) = pair$
  $\tau(t) := m$
  $\mathtt{S} :=$ **pop**$(\mathtt{S})$
  $\mathtt{S} :=$ **push**$(\mathtt{S}, (t_2, right(m)))$
  $\mathtt{S} :=$ **push**$(\mathtt{S}, (t_1, left(m)))$
  return(*EvalLeft*$(\mathtt{S}, \tau, \theta, c)$)
 if $t = \{t_1\}_{t_2}^{t_3} \wedge type(m) = encrypt$
  $\tau(t) := m$
  return$(\mathtt{S}, \tau, \theta, c)$
 if $t = [t_1]_{t_2}^{t_3} \wedge type(m) = signature$
  if $\tau(t_1) = \bot$
   $\mathtt{S} :=$ **push**$(\mathtt{S}, (t_1, msg(m)))$
   return(*EvalLeft*$(\mathtt{S}, \tau, \theta, c)$)
  if $\tau(t_1) = msg(m)$
   $\tau(t) := m$
   return$(\mathtt{S}, \tau, \theta, c)$
 return$(\emptyset, \bot, \theta, c)$

Fig. 8.14. The *EvalLeft*$(\mathtt{S}, \tau, \theta, c)$ function

Lemma 8.14. *For each* $\mathtt{S}$, τ, σ, *and* c, *let* $(\mathtt{S}_2, \tau_2)$ *and* $(\mathtt{S}_4, \tau_4, \sigma_4, c_4)$ *be the values returned by* *EvalLeft*$(\mathtt{S}, \tau)$ *and* *EvalLeft*$(\mathtt{S}, \tau, \sigma, c)$, *respectively.*

Then $\mathtt{S}_2 = \mathtt{S}_4$ *and* $\tau_2 = \tau_4$.

Proof. Suppose, for the sake of contradiction, that there exist $\mathtt{S}$, τ, σ, and c such that, given $(\mathtt{S}_2, \tau_2) =$ *EvalLeft*$(\mathtt{S}, \tau)$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) =$ *EvalLeft*$(\mathtt{S}, \tau, \sigma, c)$, $\mathtt{S}_2 \neq \mathtt{S}_4$ or $\tau_2 \neq \tau_4$. Suppose, without loss of generality, that $\mathtt{S}$ is minimal, where $\mathtt{S}$ is minimal if, denoted by (t, m) the top element of $\mathtt{S}$, for each τ' such that for each $t' \neq t$ $\tau'(t') = \tau(t')$, it holds that

- for each $\mathtt{S}'$ such that there exists (t', m') such that $\mathtt{S} = \mathbf{push}(\mathtt{S}', (t', m'))$, $(\mathtt{S}'_2, \tau'_2) = \mathit{EvalLeft}(\mathtt{S}', \tau')$ and $(\mathtt{S}'_4, \tau'_4, \sigma'_4, c'_4) = \mathit{EvalLeft}(\mathtt{S}', \tau', \sigma, c)$ implies $\mathtt{S}'_2 = \mathtt{S}'_4$ and $\tau'_2 = \tau'_4$;
- for each $\mathtt{S}'$ such that there exist (t_1, m_1) and (t_2, m_2) such that $\mathtt{S}' = \mathbf{push}(\mathbf{push}(\mathbf{pop}(\mathtt{S}), (t_2, m_2)), (t_1, m_1))$ and t_1 and t_2 are subterms of t, denoted by $(\mathtt{S}'_2, \tau'_2)$ and $(\mathtt{S}'_4, \tau'_4, \sigma'_4, c'_4)$ the values returned by functions $\mathit{EvalLeft}(\mathtt{S}', \tau')$ and $\mathit{EvalLeft}(\mathtt{S}', \tau', \sigma, c)$, respectively, it holds that $\mathtt{S}'_2 = \mathtt{S}'_4$ and $\tau'_2 = \tau'_4$.

If $\mathtt{S} = \emptyset$, then by definition of *EvalLeft*, it follows that $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) = \mathit{EvalLeft}(\mathtt{S}, \tau, \sigma, c) = (\mathtt{S}, \tau, \sigma, c)$ and $(\mathtt{S}_2, \tau_2) = \mathit{EvalLeft}(\mathtt{S}, \tau) = (\mathtt{S}, \tau)$, thus $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$. If $\mathtt{S} \neq \emptyset$, then there are the following cases:

- if $t = \mathsf{init}$, then by definition of the function *EvalLeft*, we have two cases: if $m = \lambda$, then $(\mathtt{S}_2, \tau_2) = \mathit{EvalLeft}(\mathtt{S}, \tau) = (\mathtt{S}, \tau)$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) = \mathit{EvalLeft}(\mathtt{S}, \tau, \sigma, c) = (\mathtt{S}, \tau, \sigma, c)$. Otherwise, $(\mathtt{S}_2, \tau_2) = \mathit{EvalLeft}(\mathtt{S}, \tau) = (\mathtt{S}, \bot)$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) = \mathit{EvalLeft}(\mathtt{S}, \tau, \sigma, c) = (\mathtt{S}, \bot, \sigma, c)$. Thus, in both cases, $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$;
- if $\tau(t) \neq \bot$ and $\tau(t) = m$, then $(\mathtt{S}_2, \tau_2) = \mathit{EvalLeft}(\mathtt{S}, \tau) = (\emptyset, \tau)$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) = \mathit{EvalLeft}(\mathtt{S}, \tau, \sigma, c) = (\emptyset, \tau, \sigma, c)$, thus $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$.
- if $\tau(t) \neq \bot$ and $\tau(t) \neq m$, then $(\mathtt{S}_2, \tau_2) = \mathit{EvalLeft}(\mathtt{S}, \tau) = (\emptyset, \bot)$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) = \mathit{EvalLeft}(\mathtt{S}, \tau, \sigma, c) = (\emptyset, \bot, \sigma, c)$, thus $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$.
- if $t = X$ and $\mathit{type}(X) = \mathit{type}(m)$, then given $\mathtt{S}' = \mathbf{pop}(\mathtt{S})$ and τ' such that $\tau'(t) = m$ and for each $t' \neq t$, $\tau'(t) = \tau(t)$, by definition of *EvalLeft* it follows that $(\mathtt{S}''_2, \tau''_2) = \mathit{EvalLeft}(\mathtt{S}', \tau')$ and $(\mathtt{S}''_4, \tau''_4, \sigma''_4, c''_4) = \mathit{EvalLeft}(\mathtt{S}', \tau', \sigma, c)$. Since $\mathtt{S}$ is minimal, it follows that $\mathtt{S}''_2 = \mathtt{S}''_4$ and $\tau''_2 = \tau''_4$. By definition of the function *EvalLeft*, it follows that $(\mathtt{S}_2, \tau_2) = \mathit{EvalLeft}(\mathtt{S}, \tau) = (\mathtt{S}''_2, \tau''_2)$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) = \mathit{EvalLeft}(\mathtt{S}, \tau, \sigma, c) = (\mathtt{S}''_4, \tau''_4, \sigma''_4, c''_4)$, thus $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$.
- if $t = \langle t_1, t_2 \rangle$ and $\mathit{type}(m) = \mathit{pair}$, then by definition of *EvalLeft*, it follows that $(\mathtt{S}_2, \tau_2) = \mathit{EvalLeft}(\mathtt{S}, \tau) = (\mathtt{S}''_2, \tau''_2)$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) = \mathit{EvalLeft}(\mathtt{S}, \tau, \sigma, c) = (\mathtt{S}''_4, \tau''_4, \sigma''_4, c''_4)$ where $(\mathtt{S}''_2, \tau''_2) = \mathit{EvalLeft}(\mathtt{S}', \tau')$, and $(\mathtt{S}''_4, \tau''_4, \sigma''_4, c''_4) = \mathit{EvalLeft}(\mathtt{S}', \tau', \sigma, c)$ with $\mathbf{top}(\mathtt{S}') = (t_1, \mathit{left}(m))$ and $\tau'(t) = m$ and for each $t' \neq t$, $\tau'(t) = \tau(t)$. Since $\mathtt{S}$ is minimal, it follows that $\mathtt{S}''_2 = \mathtt{S}''_4$ and $\tau''_2 = \tau''_4$, thus $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$.
- if $t = \{t_1\}^{t_3}_{t_2}$ and $\mathit{type}(m) = \mathit{encrypt}$, then by definition of *EvalLeft*, it follows that $(\mathtt{S}_2, \tau_2) = \mathit{EvalLeft}(\mathtt{S}, \tau) = (\mathtt{S}, \tau')$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) =$

EvalLeft$(\mathtt{S}, \tau, \sigma, c) = (\mathtt{S}, \tau', \sigma, c)$ where $\tau'(t') = \tau(t')$ for all $t' \neq t$, and $\tau'(t) = m$. Thus $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$.

- if $t = [t_1]_{t_2}^{t_3}$ and *type*(m) = *signature*, then by definition of *EvalLeft*, it follows that there are two cases: if $\tau(t_1) = \bot$, then $(\mathtt{S}_2, \tau_2) =$ *EvalLeft*$(\mathtt{S}, \tau) = (\mathtt{S}'_2, \tau'_2)$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) =$ *EvalLeft*$(\mathtt{S}, \tau, \sigma, c) = (\mathtt{S}'_4, \tau'_4, \sigma'_4, c'_4)$ where, denoted by $\mathtt{S}'$ the stack $\mathbf{push}(\mathtt{S}, (t_1, msg(m)))$, we have $(\mathtt{S}'_2, \tau'_2) =$ *EvalLeft*$(\mathtt{S}', \tau)$, and $(\mathtt{S}'_4, \tau'_4, \sigma'_4, c'_4) =$ *EvalLeft*$(\mathtt{S}', \tau, \sigma, c)$. Since $\mathtt{S}$ is minimal, we have that $\mathtt{S}'_2 = \mathtt{S}'_4$ and $\tau'_2 = \tau'_4$, thus $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$. If $\tau(t_1) = msg(m)$, then $(\mathtt{S}_2, \tau_2) =$ *EvalLeft*$(\mathtt{S}, \tau) = (\mathtt{S}, \tau')$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) =$ *EvalLeft*$(\mathtt{S}, \tau, \sigma, c) = (\mathtt{S}, \tau', \sigma, c)$ where $\tau'(t') = \tau(t')$ for all $t' \neq t$, and $\tau'(t) = m$. Thus $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$.
- in all other cases, we have that $(\mathtt{S}_2, \tau_2) =$ *EvalLeft*$(\mathtt{S}, \tau) = (\mathtt{S}, \bot)$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4) =$ *EvalLeft*$(\mathtt{S}, \tau, \sigma, c) = (\mathtt{S}, \bot, \sigma, c)$, thus $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$.

Since for all possible cases we have that $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$, we have obtained an absurd, thus for each $\mathtt{S}$, τ, σ, and c, denoted by $(\mathtt{S}_2, \tau_2)$ and $(\mathtt{S}_4, \tau_4, \sigma_4, c_4)$ the values returned by *EvalLeft*$(\mathtt{S}, \tau)$ and *EvalLeft*$(\mathtt{S}, \tau, \sigma, c)$, respectively, it holds that $\mathtt{S}_2 = \mathtt{S}_4$ and $\tau_2 = \tau_4$. □

The following four actions (decryption and signature verification) are modified only in the parameters they pass to *EvalLeft*. They work just at the concrete level.

Output *get_decrypt*(η, m)
  Precondition:
    status = parse_left
    $\mathbf{top}(\mathtt{S}) = (\{t_1\}_{\mathsf{ek}(A)}^{t_2}, m)$
    $\tau(A) = \eta$
  Effect:
    pending := t_1
    $\mathtt{S} := \mathbf{pop}(\mathtt{S})$
    status := parse_left_wait

Input *ret_decrypt*(η, m)
  Effect:
    $(\mathtt{S}, \tau, \sigma, c) :=$ *EvalLeft*$(\mathbf{push}(\mathtt{S}, (pending, m)), \tau, \sigma, c)$
    status := parse_left

Output $get\_verify\_sign(\eta, m)$
  Precondition:
    $status = \mathsf{parse\_left}$
    $\mathbf{top}(\mathtt{S}) = ([t_1]^{t_2}_{\mathsf{sk}(A)}, m)$
    $\tau(A) = \eta$
  Effect:
    $\mathtt{S} := \mathbf{pop}(\mathtt{S})$
    $status := \mathsf{parse\_left\_wait}$

Input $ret\_verify\_sign(\eta, b)$
  Effect:
    if b
      $(\mathtt{S}, \tau, \sigma, c) := EvalLeft(\mathtt{S}, \tau, \sigma, c)$
    else
      $\tau := \bot$
    $status := \mathsf{parse\_left}$

The next action *fail_step* is enabled when an error occurs while parsing. In this case the status variable is set to create, thus ignoring the rule under application. In particular the variable σ is not considered at all, while the variable c keeps the modifications. This will not be a problem, though. If a problem will occur, then we can always introduce an intermediary variable to be confirmed only if no error occurs. On the other hand, if no error occurs and the stack is empty, then the parsing is complete and we can start to work on the right-hand side of the rule. Yet, we are in the conditions to compute the symbolic terms and to update the concretization map.

Internal *fail_step*
  Precondition:
    $status = \mathsf{parse\_left}$
    $\tau = \bot$
  Effect:
    $status := \mathsf{create}$
    $symbolic := \bot$

```
update_c_map(c, t, τ, σ)
  c(σ(t)) := τ(t)
  if t = ⟨t1, t2⟩
    c := update_c_map(c, t1, τ, σ)
    c := update_c_map(c, t2, τ, σ)
  if t = {t1}_{t2}^{t3} ∨ t = [t1]_{t2}^{t3}
    c := update_c_map(c, t1, τ, σ)
  return(c)
```

Fig. 8.15. The $update\_c\_map(c, t, \tau, \sigma)$ function

Internal *parse_right*
Precondition:
$status = \mathsf{parse\_left}$
$\tau \neq \bot$
$\mathsf{S} = \emptyset$
Effect:
$(\tau, c) := EvalRight(right, \tau, \sigma, c)$
$status := \mathsf{parse\_right}$
$symbolic := \sigma(left)$
$H := H \cup \{\sigma(right)\}$
$c := update\_c\_map(c, left, \tau, \sigma)$

In the evaluation of the right term we need to update the concretization map c and this is done in both the two phases of the evaluation: in the first phase, the function $update\_c\_map(c, t, \tau, \sigma)$ is used to resolve all terms that do not require the invocation of the cryptographic primitives; in the second phase, we invoke the primitives for one of the unresolved place. The two phases are alternated until the term is completely resolved.

It is interesting to observe that the two *EvalRight* functions we use in the concrete and in the mixed model returns the same value on common outputs:

Lemma 8.15. *For each t, τ, σ, and c, let τ_2 and (τ_4, c_4) be the values returned by $EvalRight(t, \tau)$ and $EvalRight(t, \tau, \sigma, c)$, respectively.*

Then $\tau_2 = \tau_4$.

Proof. Fix t, τ, *send*, and c. If $\tau(t) \neq \bot$, then by definition of function *EvalRight*, it follows that $(\tau_4, c_4) = EvalRight(t, \tau, \sigma, c) = (\tau, c)$ and $\tau_2 = EvalRight(t, \tau) = \tau$, thus $\tau_2 = \tau_4$. If $\tau(t) = \bot$, we prove the statement using a classical inductive proof on the structure of the term t:

```
EvalRight(t, τ, σ, c)
  if t = stop then return(τ, c)
  if t = X ∨ τ(t) ≠ ⊥ then return(τ, c)
  if t = ⟨t1, t2⟩
    (τ, c) := EvalRight(t1, τ, σ, c)
    (τ, c) := EvalRight(t2, τ, σ, c)
    if τ(t1) ≠ ⊥ ∧ τ(t2) ≠ ⊥
      τ(t) := ⟨τ(t1), τ(t1)⟩
      c(σ(t)) := ⟨τ(t1), τ(t1)⟩
    return(τ, c)
  if t = {t1}^{t2}_{ek(A)} ∨ t = [t1]^{t2}_{sk(A)}
    (τ, c) := EvalRight(t1, τ, σ, c)
  return(τ, c)
```

Fig. 8.16. The $EvalRight(t, \tau, \sigma, c)$ function

case $t = \mathsf{stop}$: by definition of function *EvalRight*, we have that $\tau_2 = EvalRight(t, \tau) = \tau$ and $(\tau_4, c_4) = EvalRight(t, \tau, \sigma, c) = (\tau, c)$, thus $\tau_2 = \tau_4$;

case $t = X$: by definition of function *EvalRight*, it follows that $(\tau_4, c_4) = EvalRight(t, \tau, \sigma, c) = (\tau, c)$ and $\tau_2 = EvalRight(t, \tau) = \tau$, thus $\tau_2 = \tau_4$;

case $t = \langle t_1, t_2 \rangle$: let $\tau_2' = EvalRight(t_1, \tau)$, $\tau_2'' = EvalRight(t_2, \tau_2')$, $(\tau_4', c_4') = EvalRight(t_1, \tau, \sigma, c)$, and $(\tau_4'', c_4'') = EvalRight(t_2, \tau_4', \sigma, c_4')$. By inductive hypothesis, we have that $\tau_2' = \tau_4'$ and hence $\tau_2'' = \tau_4''$. This implies that $\tau_2''(t_1) = \tau_4''(t_1)$ and $\tau_2''(t_2) = \tau_4''(t_2)$ and thus either $\tau_2 = \tau_2''$ and $\tau_4 = \tau_4''$ or $\tau_2 = \tau_2''$ and $\tau_4 = \tau_4''$ with $\tau_2(t) = \langle \tau_2''(t_1), \tau_2''(t_2) \rangle$ and $\tau_4(t) = \langle \tau_4''(t_1), \tau_4''(t_2) \rangle$. This implies that $\tau_2 = \tau_4$;

case $t = \{t_1\}^{t_2}_{\mathsf{ek}(A)}$ or $t = [t_1]^{t_2}_{\mathsf{sk}(A)}$: by definition of function *EvalRight*, it follows that $\tau_2 = EvalRight(t, \tau) = EvalRight(t_1, \tau)$ and $(\tau_4, c_4) = EvalRight(t, \tau, \sigma, c) = EvalRight(t_1, \tau, \sigma, c)$ and thus, by inductive hypothesis, $\tau_2 = \tau_4$. □

The $FirstUnresolved(t, \tau)$ function of Figure 8.17 takes the outcome of *EvalRight* and identifies an unresolved place that can be resolved by invoking an external primitive. It is essentially a post-visit of the parse tree of t, searching for an undefined term whose subterms are defined. It is unchanged. The next three pairs of actions are the invocations to the three primitives that resolve unresolved places. The get action retrieves the arguments from the unresolved term and saves on *pending* the term that should

$FirstUnresolved(t, \tau)$
if $t = \mathsf{stop}$ **then return**$(\bot)$
if $\tau(t) \neq \bot$ **then return**$(\bot)$
if $t = X$ **then return**(t)
if $t = \langle t_1, t_2 \rangle$
  if $\tau(t_1) = \bot$
    return$(FirstUnresolved(t_1, \tau))$
  else
    return$(FirstUnresolved(t_2, \tau))$
if $t = \{t_1\}^{t_2}_{\mathsf{ek}(A)} \vee t = [t_1]^{t_2}_{\mathsf{sk}(A)}$
  if $\tau(t_1) = \bot$
    return$(FirstUnresolved(t_1, \tau))$
  else
    return(t)

Fig. 8.17. The $FirstUnresolved(t, \tau)$ function

denote the returned value; the return action updates τ. Upon return the concretization map needs to be updated.

Output $get\_encrypt(\eta, m)$
  Precondition:
    $status = \mathsf{parse\_right}$
    $FirstUnresolved(right, \tau) = \{t_1\}^{t_2}_{\mathsf{ek}(A)}$
    $(\eta, m) = (\tau(A), \tau(t_1))$
  Effect:
    $pending := \{t_1\}^{t_2}_{\mathsf{ek}(A)}$
    $status := \mathsf{parse\_right\_wait}$

Input $ret\_encrypt(\eta, m)$
  Effect:
    $\tau(pending) := m$
    $c(\sigma(pending)) := m$
    $(\tau, c) := EvalRight(right, \tau, \sigma, c)$
    $status := \mathsf{parse\_right}$

Output $get\_sign(\eta, m)$
  Precondition:
    $status = \mathsf{parse\_right}$
    $FirstUnresolved(right, \tau) = [t_1]^{t_2}_{\mathsf{sk}(A)}$
    $(\eta, m) = (\tau(A), \tau(t_1))$
  Effect:
    $pending := [t_1]^{t_2}_{\mathsf{sk}(A)}$
    $status := \mathsf{parse\_right\_wait}$

Input $ret\_sign(\eta, m)$
Effect:
$\tau(pending) := m$
$c(\sigma(pending)) := m$
$(\tau, c) := EvalRight(right, \tau, \sigma, c)$
$status :=$ parse_right

Output $get\_nonce(\eta)$
Precondition:
$status =$ parse_right
$X_A^j = FirstUnresolved(right, \tau) \in \mathsf{X}_n(A)$
$\eta = \tau(A)$
Effect:
$pending := X_A^j$
$status :=$ parse_right_wait

Input $ret\_nonce(\eta, m)$
Effect:
$\tau(pending) := m$
$c(\sigma(pending)) := m$
$(\tau, c) := EvalRight(right, \tau)$
$status :=$ parse_right

Finally, when the evaluation of the rule is completed successfully, that is the value of the right-hand side of the rule is computed, both functions f and F are updated with the result of the computation and we are ready to choose the next action to perform.

Internal end_step
Precondition:
$status =$ parse_right $\wedge\ FirstUnresolved(right, \tau) = \bot$
Effect:
$history := history \vdash end\_step(\tau(right))$
$f(csid) := (\tau, j, p+1)$ where $(\rho, j, p) = f(csid)$
$F(c^{-1}(csid)) := (\sigma, j, p+1)$ where $(\rho, j, p) = F(c^{-1}(csid))$
$status :=$ create

As in the concrete model, we need to send adversary's generated nonces to the nonce generator to avoid the generation of repeated values, as well as for ciphertexts and signatures. We adopt the same approach of the concrete case: to recover the sets of nonces, ciphertexts, and signatures generated by the adversary, we extract them after the evaluation of the left-hand side of the current rule and then we send them to the nonce gen-

erator, the encryption oracle, and the signature oracle before starting the evaluation of the right-hand side of the rule. We obtain this adding the state variables *extr_nonces*, *extr_ciphers*, and *extr_signatures* with initial value $\perp$ and replacing the *parse_right* action with four new actions: the output actions *used_nonces*(N), $N \subseteq \{0,1\}^k$, *used_ciphers*(C), $C \subseteq \mathcal{C}^\eta.c$, *used_signatures*$(S)$, $S \subseteq \mathcal{C}^\eta.s$, and the internal action *parse_right* defined as:

Output *used_signatures*(S)
  Precondition:
    $extr\_nonces = \perp$
    $extr\_ciphers = \perp$
    $S = extr\_signatures$
  Effect:
    $extr\_signatures := \perp$
    $(\tau, c) := EvalRight(right, \tau, \sigma, c)$
    $status :=$ parse_right
    $symbolic := \sigma(left)$
    $H := H \cup \{\sigma(right)\}$
    $c := update\_c\_map(c, left, \tau, \sigma)$

Output *used_nonces*(N)
  Precondition:
    $N = extr\_nonces$
  Effect:
    $extr\_nonces := \perp$

Output *used_ciphers*(C)
  Precondition:
    $extr\_nonces = \perp$
    $C = extr\_ciphers$
  Effect:
    $extr\_ciphers := \perp$

Internal *parse_right*
Precondition:
$status = \mathsf{parse\_left}$
$\tau \neq \bot$
$\mathtt{S} = \emptyset$
$extr\_nonces = \bot$
$extr\_ciphers = \bot$
$extr\_signatures = \bot$
Effect:
$extr\_nonces := extractNonces(left, \tau)$
$extr\_ciphers := extractCiphertexts(left, \tau)$
$extr\_signatures := extractSignatures(left, \tau)$

The three functions $extractNonces(t, \tau)$, $extractCiphertexts(t, \tau)$, and $extractSignatures(t, \tau)$ are the same of the functions we used in the concrete model.

8.4.3 The Proof

We are now able to relate the executions of the concrete model with the ones of the formal model. We obtain such relation defining several levels of abstraction and then showing that there exists a (state) polynomially accurate simulation between each pair of levels. Finally, using the execution correspondence theorem we can relate the executions of the concrete model with the ones of the formal model.

We start giving a brief overview of the abstraction levels and then we provide the formal proofs of the approximated simulations.

An Overview

Let $\mathbb{A}$ be a set of identities of participants of the protocol and k be a security parameter.

The first automaton we consider is the one that model the concrete implementation of a protocol: we define the automaton $\mathcal{A}_k^1(\mathbb{A})$ as the composition of $PAdv_k^{adv}(\mathbb{A})$, $NG_k(\mathbb{A})$, $\mathcal{E}_k(\mathbb{A})$, and $\mathcal{S}_k(\mathbb{A})$ where $PAdv_k^{adv}(\mathbb{A})$ is the automaton that model the protocol plus the concrete adversary (which actions are chosen using the probabilistic polynomial time function adv), $NG_k(\mathbb{A})$, $\mathcal{E}_k(\mathbb{A})$, and $\mathcal{S}_k(\mathbb{A})$ are a real nonce generator, encryption and signature oracles that can generate repeated nonces, ciphertexts and signatures, respectively.

The second automaton $\mathcal{A}_k^2(\mathbb{A})$ is obtained from $\mathcal{A}_k^1(\mathbb{A})$ replacing the automaton $PAdv_k^{adv}(\mathbb{A})$ with $MPAdv_k^{adv}(\mathbb{A})$ that is the automaton that model the protocol plus the mixed adversary. We recall that $MPAdv_k^{adv}(\mathbb{A})$ is defined as $PAdv_k^{adv}(\mathbb{A})$ except for the fact that each state keeps a mapping that relates formal traces with concrete traces.

We define the third automaton $\mathcal{A}_k^3(\mathbb{A})$ as the automaton $\mathcal{A}_k^2(\mathbb{A})$ except for the fact that we replace the nonce generator $NG_k(\mathbb{A})$ with the automaton $NG_k^2(\mathbb{A})$ of Section 6.1.1. This nonce generator ensures that returned nonces are different from all previously generated nonces.

Analogously, we define the $\mathcal{A}_k^4(\mathbb{A})$ and $\mathcal{A}_k^5(\mathbb{A})$ automata replacing the encryption oracle $\mathcal{E}_k(\mathbb{A})$ and the signature oracle $\mathcal{S}_k(\mathbb{A})$ with $\mathcal{E}_k^2(\mathbb{A})$ and $\mathcal{S}_k^2(\mathbb{A})$ of Sections 6.2.1 and 6.3.1, respectively. This means that $\mathcal{A}_k^5(\mathbb{A})$ ensures that nonces, ciphertexts and signatures obtained invoking the corresponding automaton are different from all previously returned values.

The automata $\mathcal{A}_k^6(\mathbb{A})$, $\mathcal{A}_k^7(\mathbb{A})$, and $\mathcal{A}_k^8(\mathbb{A})$ are obtained from the previous one imposing that the $MPAdv_k^{adv}(\mathbb{A})$ automaton extracts the nonces, ciphertexts, and signatures from the received message and that it sends them to the nonce generator, the encryption oracle, and the signature oracle before computing the response message, respectively.

Then we define the automata $\mathcal{A}_k^9(\mathbb{A})$, $\mathcal{A}_k^{10}(\mathbb{A})$, and $\mathcal{A}_k^{11}(\mathbb{A})$ that ensures that nonces, ciphertexts, and signatures generated by the corresponding automaton are different from all previously used values. This means that $\mathcal{A}_k^{10}(\mathbb{A})$ ensures that nonces, ciphertexts, and signatures that are obtained by honest users querying the cryptographic primitives are always different from all values generated previously by cryptographic primitives themselves or by the adversary.

Finally, $\mathcal{A}_k^{12}(\mathbb{A})$ is simply the automaton that represents the formal model. So, it is highly nondeterministic and all messages are ground terms of the algebra defined in Section 8.1.1.

The Simulations

Now we can start to relate the automata we have defined above using the polynomially accurate simulation relation.

Lemma 8.16. *Let $\mathbb{A}$ be a set of agents and for each $k \in \mathbb{N}$, denote by $\mathcal{A}_k^1$ the automaton $NG_k(\mathbb{A})||\mathcal{E}_k(\mathbb{A})||\mathcal{S}_k(\mathbb{A})||PAdv_k^{adv}(\mathbb{A})$ and by $\mathcal{A}_k^2$ the automaton $NG_k(\mathbb{A})||\mathcal{E}_k(\mathbb{A})||\mathcal{S}_k(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A})$.*

Then, $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \leq_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.

Proof. We prove the Lemma showing that $MPAdv_k^{adv}(\mathbb{A})$ is an extension of $PAdv_k^{adv}(\mathbb{A})$, that is $MPAdv_k^{adv}(\mathbb{A}) \in \mathrm{Ext}_B^W(PAdv_k^{adv}(\mathbb{A}))$. Thus, by Lemma 4.6, we have that $PAdv_k^{adv}(\mathbb{A}) \preceq MPAdv_k^{adv}(\mathbb{A})$ and thus, by Proposition 5.6, $PAdv_k^{adv}(\mathbb{A}) \lesssim_s MPAdv_k^{adv}(\mathbb{A})$. Finally, Theorem 5.10 implies that $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$.

So, to prove that $MPAdv_k^{adv}(\mathbb{A}) \in \mathrm{Ext}_B^W(PAdv_k^{adv}(\mathbb{A}))$, let $B = \emptyset$ and W be the set of variables $\{c, symbolic, H, \mathsf{sk}, \mathsf{vk}, \mathsf{ek}, \mathsf{dk}, ASId, F, \sigma\}$. We need to check if the requirements of Definition 4.5 are satisfied:

compatible states: let v be a state variable of $MPAdv_k^{adv}(\mathbb{A})$. Then v is either *history*, *action*, *message*, *agentsToCorrupt*, *status*, *vk*, *sk*, *ek*, *dk*, *CSId*, f, *agents_to_init*, *csid*, *left*, *right*, S, τ, *pending* and thus it is a state variable of $PAdv_k^{adv}(\mathbb{A})$, or it is c, *symbolic*, H, sk, vk, ek, dk, *ASId*, F, σ and thus $v \in W$;

compatible start state: let $\bar{s}_k^1$ be the start state of $PAdv_k^{adv}(\mathbb{A})$ and $\bar{s}_k^2$ be the start state of $MPAdv_k^{adv}(\mathbb{A})$. $\bar{s}_k^2$ is identified by value λ for variables *history*, *agents_to_init*, and *agentsToCorrupt*, by value $\emptyset$ for variables *CSId*, f, τ, c, H, *ASId*, and F and by value $\perp$ for all other variables. Since $\bar{s}_k^1$ is identified by value λ for variables *history*, *agents_to_init*, and *agentsToCorrupt*, by value $\emptyset$ for variables *CSId*, f, τ, and by value $\perp$ for all other variables, then $\bar{s}_k^1 = \bar{s}_k^2{\restriction}_{\bar{s}_k^1}$;

compatible actions: $PAdv_k^{adv}(\mathbb{A})$ and $MPAdv_k^{adv}(\mathbb{A})$ provides the same set of actions, so this condition is trivially verified; and

compatible transitions: let $tr_2 = (s_2, a, \mu_2)$ be a transition of $MPAdv_k^{adv}(\mathbb{A})$. Since the set of actions of $MPAdv_k^{adv}(\mathbb{A})$ is equal to the set of actions of $PAdv_k^{adv}(\mathbb{A})$ by definition of $MPAdv_k^{adv}(\mathbb{A})$, then we must verify that there exists a transition $tr_1 = (s_1, a, \mu_1)$ of $PAdv_k^{adv}(\mathbb{A})$ such that $tr_1 = tr_2{\restriction}_{tr_1}$. There are several cases: since most of them are very similar and the proofs are based on the same argumentation, we prove only a restricted set of actions; other cases can be proved using the same argumentation:

- a = *create_action*: by definition of the action *create_action*, it follows that $s_2.status = \mathsf{create}$ and for each state s_2' of $MPAdv_k^{adv}(\mathbb{A})$, $\mu_2(s_2') = \rho(a, m)$ and s_2' is identified by the same values of s_2 except for the following values: $s_2'.action = a$, $s_2'.message = m$, and $s_2'.status = \mathsf{act}$ where ρ is the probability measure induced by the probabilistic polynomial time function $adv(s_1.history)$. Let s_1 be the state of $PAdv_k^{adv}(\mathbb{A})$ such that $s_1 = s_2{\restriction}_{s_1}$. By definition of action

create_action, also s_1 enables *create_action* that leads to the measure μ_1 such that for each state s'_1 of $PAdv_k^{adv}(\mathbb{A})$, $\mu_1(s'_1) = \rho(a, m)$ and s'_1 is identified by the same values of s_1 except for the following values: $s'_1.action = a$, $s'_1.message = m$, and $s'_1.status = \mathsf{act}$ where ρ is the probability measure induced by the probabilistic polynomial time function $adv(s_1.history)$. Thus $\mu_1 = \mu_2{\restriction_{\mu_1}}$, and hence $tr_1 = tr_2{\restriction_{tr_1}}$.

- $a = corrupt(\eta_1, \ldots, \eta_l)$: by definition of the action $corrupt(\eta_1, \ldots, \eta_l)$, it follows that $s_2.history = \lambda$, $s_2.action = corrupt$, and $s_2.message = (\eta_1, \ldots, \eta_l)$ and $\mu_2 = \delta_{s'_2}$ where s'_2 is the state identified by the same values of s_2 except for the following values: $s'_2.history = corrupt(\eta_1, \ldots, \eta_l)$, $s'_2.action = \bot$, $s'_2.agentsToCorrupt = \eta_1 \ldots \eta_l$, $s'_2.status = \mathsf{get\_sig\_key}$, $s'_2.c = update\_c\_map(s_2.c, \eta_1 \ldots \eta_l)$, $s'_2.H = s_2.H \cup \mathbf{kn}(c^{-1}(\eta_1)) \cup \cdots \cup \mathbf{kn}(c^{-1}(\eta_l))$, and $s'_2.symbolic = corrupt(c^{-1}(\eta_1) \ldots c^{-1}(\eta_l))$ where $c^{-1} = (s_2.c)^{-1}$. Let s_1 be the state of $PAdv_k^{adv}(\mathbb{A})$ such that $s_1 = s_2{\restriction_{s_1}}$. By definition of action $corrupt(\eta_1, \ldots, \eta_l)$, also s_1 enables $corrupt(\eta_1, \ldots, \eta_l)$ that leads to the measure $\mu_1 = \delta_{s'_1}$ where s'_1 is the state of $PAdv_k^{adv}(\mathbb{A})$ that is identified by the same values of s_1 except for the following values: $s'_2.history = corrupt(\eta_1, \ldots, \eta_l)$, $s'_2.action = \bot$, $s'_2.agentsToCorrupt = \eta_1 \ldots \eta_l$, $s'_2.status = \mathsf{get\_sig\_key}$. Thus $s'_1 = s'_2{\restriction_{s'_1}}$, so $\mu_1 = \mu_2{\restriction_{\mu_1}}$, and hence $tr_1 = tr_2{\restriction_{tr_1}}$.
- $a = new(i, \eta_1 \ldots \eta_l)$: by definition of the action, it follows that $s_2.action) = new$ and $s_2.message = (i, \eta_1 \ldots \eta_l)$ and $\mu_2 = \delta_{s'_2}$ where s'_2 is the state identified by the same values of s_2 except for the following values: $s'_2.history = s_2.history \vdash new(i, \eta_1 \ldots \eta_l)$, $s'_2.action = \bot$, $s'_2.c = update\_ag\_names(s_2.c, \eta_1 \ldots \eta_l)$, $s'_2.CSId = s_2.CSId \cup \{cs\}$, $s'_2.f = s_2.f \cup \{(cs, (\tau, i, 1))\}$, $s'_2.ASId = s_2.ASId \cup \{as\}$, $s'_2.F = s_2.F \cup \{(as, (\sigma, i, 1))\}$, $s'_2.status = \mathbf{if}\ s'_2.agentsToCorrupt \neq \lambda\ \mathbf{then}\ \mathsf{get\_sig\_key}\ \mathbf{else}\ \mathsf{create}$, $s'_2.agents\_to\_init = (\eta_1 \ldots \eta_l) - \{\eta \mid s_2.ek(\eta) \neq \bot\}$, $s'_2.symbolic = new(i, c^{-1}(\eta_1) \ldots c^{-1}(\eta_l))$, and $s'_2.H = s_2.H \cup \{(i, c^{-1}(\eta_1) \ldots c^{-1}(\eta_l))\}$ where $c^{-1} = (s_2.c)^{-1}$, $\tau = \{(A_j, \eta_j) \mid 1 \leqslant j \leqslant l\}$, $cs = (|s_2.CSId| + 1, i, (\eta_1 \ldots \eta_l))$, $\sigma = \{(A_j, c^{-1}(\eta_j)) \mid 1 \leqslant j \leqslant l\}$, and $as = (|s_2.ASId| + 1, i, (c^{-1}(\eta_1) \ldots c^{-1}(\eta_l)))$. Let s_1 be the state of $PAdv_k^{adv}(\mathbb{A})$ such that $s_1 = s_2{\restriction_{s_1}}$. By definition of action $get\_corrupt\_sign(\eta)$, also s_1 enables $new(i, \eta_1 \ldots \eta_l)$ that leads to the measure $\mu_1 = \delta_{s'_1}$ where s'_1 is the state of $PAdv_k^{adv}(\mathbb{A})$ that is identified by the same values of s_1 except for the following

values: $s'_1.history = s_1.history \vdash new(i, \eta_1 \dots \eta_l)$, $s'_1.action = \bot$, $s'_1.CSId = s_1.CSId \cup \{cs\}$, $s'_1.f = s_1.f \cup \{(cs, (\tau, i, 1))\}$, $s'_1.status =$ **if** $s'_1.agentsToCorrupt \neq \lambda$ **then** get_sig_key **else** create, and $s'_1.agents\_to\_init = (\eta_1 \dots \eta_l) - \{\eta \mid s_1.ek(\eta) \neq \bot\}$ where $\tau = \{(A_j, \eta_j) \mid 1 \leqslant j \leqslant l\}$ and $cs = (|s_1.CSId| + 1, i, (\eta_1 \dots \eta_l))$. Thus $s'_1 = s'_2{\restriction}_{s'_1}$, so $\mu_1 = \mu_2{\restriction}_{\mu_1}$, and hence $tr_1 = tr_2{\restriction}_{tr_1}$.

- $a = send(cs, m)$: by definition of the action $send(cs, m)$, it follows that $s_2.action = send$ and $s_2.message = (cs, m)$ and $\mu_2 = \delta_{s'_2}$ where s'_2 is the state identified by the same values of s_2 except for the following values: $s'_2.history = s_2.history \vdash send(cs, m)$, $s'_2.csid = cs$, $s'_2.action = \bot$, and $s'_2.status =$ parse_left. Moreover, given $(\rho, j, p) = (s_2.f)(cs)$ and $(\phi, j, p) = (s_2.F)((s_2.c)^{-1}(cs))$, if $p \leqslant k_j$ then $s'_2.left = l^j_p$, $s'_2.right = r^j_p$, $s'_2.\sigma = \phi$, and $(s'_2.\mathtt{S}, s'_2.\tau, s'_2.\sigma, s'_2.c) = EvalLeft(\mathbf{push}(\emptyset, (l^j_p, m)), \rho, \phi, c)$. Otherwise, $(s'_2.\mathtt{S}, s'_2.\tau) = (\emptyset, \bot)$. Let s_1 be the state of $PAdv^{adv}_k(\mathbb{A})$ such that $s_1 = s_2{\restriction}_{s_1}$. By definition of action $get\_corrupt\_sign(\eta)$, also s_1 enables $send(cs, m)$ that leads to the measure $\mu_1 = \delta_{s'_1}$ where s'_1 is the state of $PAdv^{adv}_k(\mathbb{A})$ that is identified by the same values of s_1 except for the following values: $s'_1.history = s_1.history \vdash send(cs, m)$, $s'_1.csid = cs$, $s'_1.action = \bot$, and $s'_1.status =$ parse_left. Moreover, given $(\rho, j, p) = (s_1.f)(cs)$, if $p \leqslant k_j$ then $s'_1.left = l^j_p$, $s'_1.right = r^j_p$, and $(s'_1.\mathtt{S}, s'_1.\tau) = EvalLeft(\mathbf{push}(\emptyset, (l^j_p, m)), \rho)$. Otherwise, $(s'_1.\mathtt{S}, s'_1.\tau) = (\emptyset, \bot)$. Since by Lemma 8.14 we have that $EvalLeft(\mathtt{S}, \tau)$ and $EvalLeft(\mathtt{S}, \tau, \sigma, c)$ return the same values on common outputs, it follows that $s'_1 = s'_2{\restriction}_{s'_1}$, thus $\mu_1 = \mu_2{\restriction}_{\mu_1}$, and hence $tr_1 = tr_2{\restriction}_{tr_1}$.
- $a = parse\_right$: by definition of the action, it follows that $s_2.\mathtt{S} = \emptyset$, $s_2.\tau \neq \bot$, and $s_2.status =$ parse_left and $\mu_2 = \delta_{s'_2}$ where s'_2 is the state identified by the same values of s_2 except for the following values: $(s'_2.\tau, s'_2.c) = EvalRight(s_2.right, s_2.\tau, s_2.\sigma, s_2.c)$, $s'_2.c = update\_c\_map(s'_2.c, s_2.left, s'_2.\tau, s_2.\sigma)$, $s'_2.symbolic = (s_2.\sigma)(s_2.left)$, $s'_2.H = s_2.H \cup \{(s_2.\sigma)(s_2.right)\}$, and $s'_2.status =$ parse_right. Let s_1 be the state of $PAdv^{adv}_k(\mathbb{A})$ such that $s_1 = s_2{\restriction}_{s_1}$. By definition of the action, also s_1 enables $parse\_right$ that leads to the measure $\mu_1 = \delta_{s'_1}$ where s'_1 is the state of $PAdv^{adv}_k(\mathbb{A})$ that is identified by the same values of s_1 except for the following values: $s'_1.\tau = EvalRight(s_1.right, s_1.\tau)$ and $s'_1.status =$ parse_right. Since by Lemma 8.15 we have that $EvalRight(t, \tau)$ and $EvalRight(t, \tau, \sigma, c)$ re-

turn the same values on common outputs, it follows that $s'_1 = s'_2{\restriction}_{s'_1}$, thus $\mu_1 = \mu_2{\restriction}_{\mu_1}$, and hence $tr_1 = tr_2{\restriction}_{tr_1}$.

- $a = end\_step$: by definition of the action $get\_nonce(\eta)$, it follows that $FirstUnresolved(s_2.right, s_2.\tau) = bot$ and $s_2.status = \mathsf{parse\_right}$ and $\mu_2 = \delta_{s'_2}$ where s'_2 is the state identified by the same values of s_2 except for the following values: $s'_2.history = s_2.history \vdash end\_step((s_2.\tau)(s_2.right))$, $(s'_2.f)(s_2.csid) = (s_2.\tau, j, p+1)$ where $(\rho, j, p) = (s_2.f)(s_2.csid)$, $(s'_2.F)((s_2.c)^{-1}(s_2.csid)) = (s_2.\sigma, j, p+1)$ where $(\rho, j, p) = (s_2.F)((s_2.c)^{-1}(s_2.csid))$, and $s'_2.status = \mathsf{create}$. Let s_1 be the state of $PAdv_k^{adv}(\mathbb{A})$ such that $s_1 = s_2{\restriction}_{s_1}$. By definition of action end_step, also s_1 enables end_step that leads to the measure $\mu_1 = \delta_{s'_1}$ where s'_1 is the state of $PAdv_k^{adv}(\mathbb{A})$ that is identified by the same values of s_1 except for the following values: $s'_2.history = s_2.history \vdash end\_step((s_2.\tau)(s_2.right))$, $(s'_2.f)(s_2.csid) = (s_2.\tau, j, p+1)$ where $(\rho, j, p) = (s_2.f)(s_2.csid)$, $s'_2.status = \mathsf{create}$. Thus $s'_1 = s'_2{\restriction}_{s'_1}$, so $\mu_1 = \mu_2{\restriction}_{\mu_1}$, and hence $tr_1 = tr_2{\restriction}_{tr_1}$.
- $a = get\_corrupt\_sign(\eta)$: by definition of the action, it follows that $\mathrm{head}(s_2.agentsToCorrupt) = \eta$ and $s_2.status = \mathsf{get\_sig\_key}$ and $\mu_2 = \delta_{s'_2}$ where s'_2 is the state identified by the same values of s_2 except for the following values: $s'_2.history = s_2.history \vdash get\_corrupt\_sign(\eta)$ and $s'_2.status = \mathsf{wait\_sig\_key}$. Let s_1 be the state of $PAdv_k^{adv}(\mathbb{A})$ such that $s_1 = s_2{\restriction}_{s_1}$. By definition of action $get\_corrupt\_sign(\eta)$, also s_1 enables $get\_corrupt\_sign(\eta)$ that leads to the measure $\mu_1 = \delta_{s'_1}$ where s'_1 is the state of $PAdv_k^{adv}(\mathbb{A})$ that is identified by the same values of s_1 except for the following values: $s'_1.history = s_1.history \vdash get\_corrupt\_sign(\eta)$ and $s'_1.status = \mathsf{wait\_sig\_key}$. Thus $s'_1 = s'_2{\restriction}_{s'_1}$, so $\mu_1 = \mu_2{\restriction}_{\mu_1}$, and hence $tr_1 = tr_2{\restriction}_{tr_1}$.
- all other cases are very similar to the previous one and they are proved using the same argumentation.

Since the requirements of Definition 4.5 are satisfied, for each $k \in \mathbb{N}$, $MPAdv_k^{adv}(\mathbb{A}) \in \mathrm{Ext}_{\emptyset}^{W}(PAdv_k^{adv}(\mathbb{A}))$. □

We define the third automaton $\mathcal{A}_k^3(\mathbb{A})$ as the automaton $\mathcal{A}_k^2(\mathbb{A})$ except for the fact that we replace the nonce generator $NG_k(\mathbb{A})$ with the automaton $NG_k^2(\mathbb{A})$ of Section 6.1.1. This nonce generator ensures that returned nonces are different from all previously generated nonces.

Lemma 8.17. *Let $\mathbb{A}$ be a set of agents and for each $k \in \mathbb{N}$, denote by $\mathcal{A}_k^2$ the automaton $NG_k(\mathbb{A})||\mathcal{E}_k(\mathbb{A})||\mathcal{S}_k(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A})$ and by $\mathcal{A}_k^3$ the automaton $NG_k^2(\mathbb{A})||\mathcal{E}_k(\mathbb{A})||\mathcal{S}_k(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A})$ where $NG_k^2(\mathbb{A})$ is the nonce generator automaton defined as in Section 6.1.1.*

Then there exist $\mathcal{B}_k^1$ and $\mathcal{B}_k^2$ such that

$$\{\mathcal{A}_k^2\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k^1\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k^2\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^3\}_{k\in\mathbb{N}}.$$

Proof. Let $NG_k^1(\mathbb{A})$ be the nonce generator automaton defined as in Section 6.1.1. Let G_k be the set of states s of NG_k^1 such that for all $A \in \mathbb{A}$, $s.is\_fresh_A \neq F$. By Propositions 6.2, 6.5, and 6.4, we know that for each context $\mathcal{C}_k$ compatible with $NG_k^2(\mathbb{A})$ we have the following chain of state approximated simulations: $\{NG_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{NG_k^1(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(NG_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{NG_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$.

To prove the statement of the Lemma, it is sufficient to show that $\bar{\mathcal{C}}_k = \mathcal{E}_k(\mathbb{A})||\mathcal{S}_k(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A})$ is a context compatible with $NG_k^2(\mathbb{A})$.

$\bar{\mathcal{C}}_k$ is trivially compatible with $NG_k^2(\mathbb{A})$ since the set $H_{NG_k^2}$ of internal actions of $NG_k^2(\mathbb{A})$ is empty and thus $H_{NG_k^2} \cap A_{\bar{\mathcal{C}}_k} = \emptyset$ and the set $H_{\bar{\mathcal{C}}_k} = \{create\_action, fail\_step, parse\_right, end\_step\}$ is disjoint from the set $A_{NG_k^2}$ of actions of $NG_k^2(\mathbb{A})$ and thus $A_{NG_k^2} \cap H_{\bar{\mathcal{C}}_k} = \emptyset$, as required. □

Lemma 8.18. *Let $\mathbb{A}$ be a set of agents and for each $k \in \mathbb{N}$, denote by $\mathcal{A}_k^3$ the automaton $NG_k^2(\mathbb{A})||\mathcal{E}_k(\mathbb{A})||\mathcal{S}_k(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A})$ and by $\mathcal{A}_k^4$ the automaton $NG_k^2(\mathbb{A})||\mathcal{E}_k^2(\mathbb{A})||\mathcal{S}_k(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A})$ where $\mathcal{E}_k^2(\mathbb{A})$ is the encryption oracle automaton defined as in Section 6.2.1.*

Then there exist $\mathcal{B}_k^1$ and $\mathcal{B}_k^2$ such that

$$\{\mathcal{A}_k^3\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k^1\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k^2\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^4\}_{k\in\mathbb{N}}.$$

Proof. Let $\mathcal{E}_k^1(\mathbb{A})$ be the encryption oracle automaton defined as in Section 6.2.1. Let G_k be the set of states s of $\mathcal{E}_k^1$ such that for all $A \in \mathbb{A}$, $s.is\_fresh_A \neq F$. By Propositions 6.12, 6.15, and 6.14, we know that for each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k^2(\mathbb{A})$ we have the following chain of simulations: $\{\mathcal{E}_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^1(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{E}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$.

To prove the statement of the Lemma, it is sufficient to show that $\bar{\mathcal{C}}_k = NG_k^2(\mathbb{A})||\mathcal{S}_k(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A})$ is a context compatible with $\mathcal{E}_k^2(\mathbb{A})$.

$\bar{\mathcal{C}}_k$ is trivially compatible with $\mathcal{E}_k^2(\mathbb{A})$ since the set $H_{\mathcal{E}_k^2}$ of internal actions of $\mathcal{E}_k^2(\mathbb{A})$ is empty and thus $H_{\mathcal{E}_k^2} \cap A_{\bar{\mathcal{C}}_k} = \emptyset$ and the set $H_{\bar{\mathcal{C}}_k} = \{create\_action, fail\_step, parse\_right, end\_step\}$ is disjoint from the set $A_{\mathcal{E}_k^2}$ of actions of $\mathcal{E}_k^2(\mathbb{A})$ and thus $A_{\mathcal{E}_k^2} \cap H_{\bar{\mathcal{C}}_k} = \emptyset$, as required. □

Lemma 8.19. *Let $\mathbb{A}$ be a set of agents and for each $k \in \mathbb{N}$, denote by $\mathcal{A}_k^4$ the automaton $NG_k^2(\mathbb{A})||\mathcal{E}_k^2(\mathbb{A})||\mathcal{S}_k(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A})$ and by $\mathcal{A}_k^5$ the automaton $NG_k^2(\mathbb{A})||\mathcal{E}_k^2(\mathbb{A})||\mathcal{S}_k^2(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A})$ where $\mathcal{S}_k^2(\mathbb{A})$ is the signature oracle automaton defined as in Section 6.3.1.*

Then there exist $\mathcal{B}_k^1$ and $\mathcal{B}_k^2$ such that

$$\{\mathcal{A}_k^4\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k^1\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k^2\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^5\}_{k\in\mathbb{N}}.$$

Proof. Let $\mathcal{S}_k^1(\mathbb{A})$ be the signature oracle automaton defined as in Section 6.3.1. Let G_k be the set of states s of $\mathcal{S}_k^1$ such that for all $A \in \mathbb{A}$, $s.is\_fresh_A \neq F$. By Propositions 6.28, 6.31, and 6.30, we know that for each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k^2(\mathbb{A})$ we have the following chain of simulations: $\{\mathcal{S}_k(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}_k^1(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{S}_k^1(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}_k^2(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$.

To prove the statement of the Lemma, it is sufficient to show that $\bar{\mathcal{C}}_k = NG_k^2(\mathbb{A})||\mathcal{E}_k^2(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A})$ is a context compatible with $\mathcal{S}_k^2(\mathbb{A})$.

$\bar{\mathcal{C}}_k$ is trivially compatible with $\mathcal{S}_k^2(\mathbb{A})$ since the set $H_{\mathcal{S}_k^2}$ of internal actions of $\mathcal{S}_k^2(\mathbb{A})$ is empty and thus $H_{\mathcal{S}_k^2} \cap A_{\bar{\mathcal{C}}_k} = \emptyset$ and the set $H_{\bar{\mathcal{C}}_k} = \{create\_action, fail\_step, parse\_right, end\_step\}$ is disjoint from the set $A_{\mathcal{S}_k^2}$ of actions of $\mathcal{S}_k^2(\mathbb{A})$ and thus $A_{\mathcal{S}_k^2} \cap H_{\bar{\mathcal{C}}_k} = \emptyset$, as required. □

Lemma 8.20. *Let $\mathbb{A}$ be a set of agents and for each $k \in \mathbb{N}$, denote by $\mathcal{A}_k^5$ the automaton $NG_k^2(\mathbb{A})||\mathcal{E}_k^2(\mathbb{A})||\mathcal{S}_k^2(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A})$ and by $\mathcal{A}_k^6$ the automaton $NG_k^3(\mathbb{A})||\mathcal{E}_k^2(\mathbb{A})||\mathcal{S}_k^2(\mathbb{A})||MNPAdv_k^{adv}(\mathbb{A})$ where $NG_k^3(\mathbb{A})$ is the nonce generator automaton defined as in Section 6.1.1. Let $UN_k = \{used\_nonces(N) \mid N \subseteq \{0,1\}^k\}$.*

Then, $\{\mathrm{Hide}_{UN_k}(\mathcal{A}_k^5)\}_{k\in\mathbb{N}} \lessapprox_s \{\mathrm{Hide}_{UN_k}(\mathcal{A}_k^6)\}_{k\in\mathbb{N}}$.

Proof. To simplify the proof, denote by $\mathcal{B}_k^2$ and $\mathcal{B}_k^3$ the composed automata $\mathrm{Hide}_{UN_k}(NG_k^2(\mathbb{A})||MPAdv_k^{adv}(\mathbb{A}))$ and $\mathrm{Hide}_{UN_k}(NG_k^3(\mathbb{A})||MNPAdv_k^{adv}(\mathbb{A}))$.

We prove the Lemma showing a stronger result, that is that the automaton $\mathcal{B}_k^2$ is weakly simulated by $\mathcal{B}_k^3$ and that each matching weak transition has length at most 2. Thus, by Proposition 5.16, it follows that $\{\mathcal{B}_k^2\}_{k\in\mathbb{N}} \lessapprox_s \{\mathcal{B}_k^3\}_{k\in\mathbb{N}}$ and hence, by Theorem 5.10, we have that $\{\mathrm{Hide}_{UN_k}(\mathcal{A}_k^5)\}_{k\in\mathbb{N}} \lessapprox_s \{\mathrm{Hide}_{UN_k}(\mathcal{A}_k^6)\}_{k\in\mathbb{N}}$.

Let S_k^2 and S_k^3 be the sets of states of $\mathcal{B}_k^2$ and $\mathcal{B}_k^3$, respectively, and $\mathcal{R}_k \subseteq S_k^2 \times S_k^3$ be the relation defined as $s_2 \; \mathcal{R}_k \; s_3$ if and only if $s_2 = s_3{\restriction}_{s_2}$ and $s_3.extr\_nonces = \bot$.

The condition on start states is trivially verified, since by definition of $\mathcal{B}_k^2$ and $\mathcal{B}_k^3$, it follows that the two automata initialize the common state

variables with the same values and thus $\bar{s}_k^2 = \bar{s}_k^3 \restriction_{\bar{s}_k^2}$. Moreover, by definition of $\mathcal{B}_k^3$, we have that $s_3.extr\_nonces = \bot$ and thus $\bar{s}_k^2 \; \mathcal{R}_k \; \bar{s}_k^3$, as required.

For the step condition, let s_2 and s_3 be two states such that $s_2 \; \mathcal{R}_k \; s_3$ and suppose that $s_2 \xrightarrow{a} \mu_2$. We must find μ_3 such that $s_3 \stackrel{a}{\Longrightarrow}{}_C^2 \; \mu_3$ such that $\mu_2 \; \mathcal{L}(\mathcal{R}_k) \; \mu_3$. There are several cases:

case $a = parse\_right$: by definition of $parse\_right$, it follows that $s_2.status = \mathsf{parse\_left}$, $s_2.\tau \neq \bot$, and $s_2.\mathsf{S} = \emptyset$. Moreover, we have that $\mu_2 = \delta_{s_2'}$ where s_2' is the state of $\mathcal{B}_k^2$ that is identified by the same values of s_2 except for the following values: $s_2'.c = update\_c\_map(c', left', \tau', \sigma')$, $s_2'.\tau = \tau'$, $s_2'.symbolic = \sigma'(left')$, $s_2'.H = s_2.H \cup \{\sigma'(right')\}$, and $s_2'.status = \mathsf{parse\_right}$ where $\sigma' = s_2.\sigma$, $left' = s_2.left$, $right' = s_2.right$, and $(\tau', c') = EvalRight(right', s_2.\tau, \sigma', s_2.c)$. Since $s_2 \; \mathcal{R}_k \; s_3$, it follows that also s_3 satisfies $s_3.status = \mathsf{parse\_left}$, $s_3.\tau \neq \bot$, and $s_3.\mathsf{S} = \emptyset$. Moreover, it satisfies $s_3.extr\_nonces = \bot$ and thus we have that s_3 enables the transition $s_3 \xrightarrow{a} \mu_3'$ where $\mu_3' = \delta_{s_3'}$ where s_3' is the state of $\mathcal{B}_k^3$ that is identified by the same values of s_3 except for the $extr\_nonces$ variable where $s_3'.extr\_nonces = N$ and $N = extractNonces(s_3.left, s_3.\tau)$. This implies that s_3' enables the transition labelled by $used\_nonces(N)$ that leads to $\delta_{s_3''}$ where s_3'' is the state of $\mathcal{B}_k^3$ that is identified by the same values of s_3' except for the following values: $s_3''.used\_nonces = s_3'.used\_nonces \cup N$, $s_3''.extr\_nonces = \bot$, $s_3''.c = update\_c\_map(c', left', \tau', \sigma')$, $s_3''.\tau = \tau'$, $s_3''.symbolic = \sigma'(left')$, $s_3''.H = s_3'.H \cup \{\sigma'(right')\}$, and $s_3''.status = \mathsf{parse\_right}$ where $\sigma' = s_3'.\sigma$, $left' = s_3'.left$, $right' = s_3'.right$, and $(\tau', c') = EvalRight(right', s_3'.\tau, \sigma', s_3'.c)$. Since s_3' and s_3 differ only on the value of the $extr\_nonces$ variable, it follows that $s_3''.extr\_nonces = \bot$, $s_3''.c = update\_c\_map(c', left', \tau', \sigma')$, $s_3''.\tau = \tau'$, $s_3''.symbolic = \sigma'(left')$, $s_3''.H = s_3.H \cup \{\sigma'(right')\}$, and $s_3''.status = \mathsf{parse\_right}$ where $\sigma' = s_3.\sigma$, $left' = s_3.left$, $right' = s_3.right$, and $(\tau', c') = EvalRight(right', s_3.\tau, \sigma', s_3.c)$ and thus $\delta_{s_2'} \; \mathcal{L}(\mathcal{R}_k) \; \delta_{s_3''}$. This implies that there exists a weak combined 2-bounded transition $s_3 \stackrel{a}{\Longrightarrow}{}_C^2 \; \mu_3$ such that $\mu_2 = \delta_{s_2'} \; \mathcal{L}(\mathcal{R}_k) \; \delta_{s_3''} = \mu_3$, as required.

case $a = get\_corrupt\_sign(\eta)$: by definition of $get\_corrupt\_sign(\eta)$ action, it follows that $s_2.status = \mathsf{get\_sig\_key}$ and $\mathrm{head}(s_2.agentsToCorrupt) = \eta$. Moreover, we have that $\mu_2 = \delta_{s_2'}$ where s_2' is the state of $\mathcal{B}_k^2$ that is identified by the same values of s_2 except for the following values: $s_2'.history = s_2.history \vdash get\_corrupt\_sign(\eta)$ and $s_2'.status = \mathsf{wait\_sig\_key}$. Since $s_2 \; \mathcal{R}_k \; s_3$, it follows that also s_3 satisfies $s_3.status = \mathsf{get\_sig\_key}$ and $\mathrm{head}(s_3.agentsToCorrupt) = \eta$. Thus s_3 enables the transition $s_3 \xrightarrow{a} \mu_3$

where $\mu_3 = \delta_{s'_3}$ where s'_3 is the state of $\mathcal{B}^3_k$ that is identified by the same values of s_3 except for the following values: $s'_3.history = s_3.history \vdash get\_corrupt\_sign(\eta)$ and $s'_3.status = \mathsf{wait\_sig\_key}$. This implies that $\delta_{s'_2}\ \mathcal{L}(\mathcal{R}_k)\ \delta_{s'_3}$ that is $\mu_2\ \mathcal{L}(\mathcal{R}_k)\ \mu_3$ and thus $s_2 \xrightarrow{a} \mu_2$ is matched by $s_3 \xrightarrow{a} \mu_3$. Since each transition is also a weak combined 2-bounded transition, it follows that $s_2 \xrightarrow{a} \mu_2$ is matched by $s_3 \stackrel{a}{\Longrightarrow}{}^2_C\ \mu_3$, as required.

all other cases: the argumentation is the same of the previous case.

Since all conditions are satisfied, we have that $\mathcal{B}^2_k \preccurlyeq^2 \mathcal{B}^3_k$. Thus, by Proposition 5.16, it follows that $\{\mathcal{B}^2_k\}_{k\in\mathbb{N}} \lessapprox_s \{\mathcal{B}^3_k\}_{k\in\mathbb{N}}$ and hence, by Theorem 5.10, we have that $\{\mathrm{Hide}_{UN_k}(\mathcal{A}^5_k)\}_{k\in\mathbb{N}} \lessapprox_s \{\mathrm{Hide}_{UN_k}(\mathcal{A}^6_k)\}_{k\in\mathbb{N}}$. □

Lemma 8.21. *Let $\mathbb{A}$ be a set of agents and for each $k \in \mathbb{N}$, denote by $\mathcal{A}^6_k$ the automaton $NG^3_k(\mathbb{A})||\mathcal{E}^2_k(\mathbb{A})||\mathcal{S}^2_k(\mathbb{A})||MNPAdv^{adv}_k(\mathbb{A})$ and by $\mathcal{A}^7_k$ the automaton $NG^3_k(\mathbb{A})||\mathcal{E}^3_k(\mathbb{A})||\mathcal{S}^2_k(\mathbb{A})||MNEPAdv^{adv}_k(\mathbb{A})$ where $\mathcal{E}^3_k(\mathbb{A})$ is the encryption oracle automaton defined as in Section 6.2.1. Let $UC = \{used\_ciphers(C) \mid C \subseteq \mathrm{Ciphertext}\}$.*

Then, $\{\mathrm{Hide}_{UC}(\mathcal{A}^6_k)\}_{k\in\mathbb{N}} \lessapprox_s \{\mathrm{Hide}_{UC}(\mathcal{A}^7_k)\}_{k\in\mathbb{N}}$.

Proof. To simplify the proof, denote by $\mathcal{B}^2_k$ and $\mathcal{B}^3_k$ the composed automata $\mathrm{Hide}_{UC}(\mathcal{E}^2_k(\mathbb{A})||MNPAdv^{adv}_k(\mathbb{A}))$ and $\mathrm{Hide}_{UC}(\mathcal{E}^3_k(\mathbb{A})||MNEPAdv^{adv}_k(\mathbb{A}))$.

We prove the Lemma showing a stronger result, that is that the automaton $\mathcal{B}^2_k$ is weakly simulated by $\mathcal{B}^3_k$ and that each matching weak transition has length at most 2. Thus, by Proposition 5.16, it follows that $\{\mathcal{B}^2_k\}_{k\in\mathbb{N}} \lessapprox_s \{\mathcal{B}^3_k\}_{k\in\mathbb{N}}$ and hence, by Theorem 5.10, we have that $\{\mathrm{Hide}_{UN_k}(\mathcal{A}^5_k)\}_{k\in\mathbb{N}} \lessapprox_s \{\mathrm{Hide}_{UN_k}(\mathcal{A}^6_k)\}_{k\in\mathbb{N}}$.

Let S^2_k and S^3_k be the sets of states of $\mathcal{B}^2_k$ and $\mathcal{B}^3_k$, respectively, and $\mathcal{R}_k \subseteq S^2_k \times S^3_k$ be the relation defined as $s_2\ \mathcal{R}_k\ s_3$ if and only if $s_2 = s_3{\restriction_{s_2}}$ and $s_3.extr\_ciphers = \bot \iff s_2.extr\_nonces = \bot$.

The condition on start states is trivially verified, since by definition of $\mathcal{B}^2_k$ and $\mathcal{B}^3_k$, it follows that the two automata initialize the common state variables with the same values and thus $\bar{s}^2_k = \bar{s}^3_k{\restriction_{\bar{s}^2_k}}$. Moreover, by definition of $\mathcal{B}^3_k$, we have that $s_3.extr\_ciphers = \bot$ and $s_2.extr\_nonces = \bot$ and thus $\bar{s}^2_k\ \mathcal{R}_k\ \bar{s}^3_k$, as required.

For the step condition, let s_2 and s_3 be two states such that $s_2\ \mathcal{R}_k\ s_3$ and suppose that $s_2 \xrightarrow{a} \mu_2$. We must find μ_3 such that $s_3 \stackrel{a}{\Longrightarrow}{}^2_C\ \mu_3$ such that $\mu_2\ \mathcal{L}(\mathcal{R}_k)\ \mu_3$. There are several cases:

case $a = \mathit{used\_nonces}(N)$: by definition of $\mathit{used\_nonces}(N)$, it follows that $s_2.\mathit{extr\_nonces} = N$. Moreover, we have that $\mu_2 = \delta_{s_2'}$ where s_2' is the state of $\mathcal{B}_k^2$ that is identified by the same values of s_2 except for the following values: $s_2'.\mathit{used\_nonces} = s_2.\mathit{used\_nonces} \cup N$, $s_2'.\mathit{extr\_nonces} = \bot$, $s_2'.c = \mathit{update\_c\_map}(c', \mathit{left}', \tau', \sigma')$, $s_2'.\tau = \tau'$, $s_2'.\mathit{symbolic} = \sigma'(\mathit{left}')$, $s_2'.H = s_2.H \cup \{\sigma'(\mathit{right}')\}$, and $s_2'.\mathit{status} = \mathsf{parse\_right}$ where $\sigma' = s_2.\sigma$, $\mathit{left}' = s_2.\mathit{left}$, $\mathit{right}' = s_2.\mathit{right}$, and $(\tau', c') = \mathit{EvalRight}(\mathit{right}', s_2.\tau, \sigma', s_2.c)$. Since $s_2 \ \mathcal{R}_k \ s_3$, it follows that also s_3 satisfies $s_3.\mathit{extr\_nonces} = N$. Moreover, s_3 also satisfies $s_3.\mathit{extr\_ciphers} \neq \bot$ and thus we have that s_3 enables the transition $s_3 \xrightarrow{a} \mu_3'$ where $\mu_3' = \delta_{s_3'}$ where s_3' is the state of $\mathcal{B}_k^3$ that is identified by the same values of s_3 except for the $\mathit{extr\_nonces}$ variable where $s_3'.\mathit{extr\_nonces} = \bot$. This implies that s_3' enables the transition labelled by $\mathit{used\_ciphers}(C)$ (where $C = s_3'.\mathit{extr\_ciphers} \neq \bot$) that leads to $\delta_{s_3''}$ where s_3'' is the state of $\mathcal{B}_k^3$ that is identified by the same values of s_3' except for the following values: $s_3''.\mathit{extr\_ciphers} = \bot$, $s_3''.c = \mathit{update\_c\_map}(c', \mathit{left}', \tau', \sigma')$, $s_3''.\tau = \tau'$, $s_3''.\mathit{symbolic} = \sigma'(\mathit{left}')$, $s_3''.H = s_3'.H \cup \{\sigma'(\mathit{right}')\}$, and $s_3''.\mathit{status} = \mathsf{parse\_right}$ where $\sigma' = s_3'.\sigma$, $\mathit{left}' = s_3'.\mathit{left}$, $\mathit{right}' = s_3'.\mathit{right}$, and $(\tau', c') = \mathit{EvalRight}(\mathit{right}', s_3'.\tau, \sigma', s_3'.c)$. Since s_3' and s_3 differ only on the value of the $\mathit{extr\_nonces}$ variable, it follows that $s_3''.\mathit{extr\_ciphers} = \bot$, $s_3''.c = \mathit{update\_c\_map}(c', \mathit{left}', \tau', \sigma')$, $s_3''.\tau = \tau'$, $s_3''.\mathit{symbolic} = \sigma'(\mathit{left}')$, $s_3''.H = s_3.H \cup \{\sigma'(\mathit{right}')\}$, and $s_3''.\mathit{status} = \mathsf{parse\_right}$ where $\sigma' = s_3.\sigma$, $\mathit{left}' = s_3.\mathit{left}$, $\mathit{right}' = s_3.\mathit{right}$, and $(\tau', c') = \mathit{EvalRight}(\mathit{right}', s_3.\tau, \sigma', s_3.c)$ and thus $\delta_{s_2'} \ \mathcal{L}(\mathcal{R}_k) \ \delta_{s_3''}$. This implies that there exists a weak combined 2-bounded transition $s_3 \stackrel{a}{\Longrightarrow}{}_C^2 \mu_3$ such that $\mu_2 = \delta_{s_2'} \ \mathcal{L}(\mathcal{R}_k) \ \delta_{s_3''} = \mu_3$, as required.

case $a = \mathit{get\_corrupt\_sign}(\eta)$: by definition of $\mathit{get\_corrupt\_sign}(\eta)$ action, it follows that $s_2.\mathit{status} = \mathsf{get\_sig\_key}$ and $\mathrm{head}(s_2.\mathit{agentsToCorrupt}) = \eta$. Moreover, we have that $\mu_2 = \delta_{s_2'}$ where s_2' is the state of $\mathcal{B}_k^2$ that is identified by the same values of s_2 except for the following values: $s_2'.\mathit{history} = s_2.\mathit{history} \vdash \mathit{get\_corrupt\_sign}(\eta)$ and $s_2'.\mathit{status} = \mathsf{wait\_sig\_key}$. Since $s_2 \ \mathcal{R}_k \ s_3$, it follows that also s_3 satisfies $s_3.\mathit{status} = \mathsf{get\_sig\_key}$ and $\mathrm{head}(s_3.\mathit{agentsToCorrupt}) = \eta$. Thus s_3 enables the transition $s_3 \xrightarrow{a} \mu_3$ where $\mu_3 = \delta_{s_3'}$ where s_3' is the state of $\mathcal{B}_k^3$ that is identified by the same values of s_3 except for the following values: $s_3'.\mathit{history} = s_3.\mathit{history} \vdash \mathit{get\_corrupt\_sign}(\eta)$ and $s_3'.\mathit{status} = \mathsf{wait\_sig\_key}$. This implies that $\delta_{s_2'} \ \mathcal{L}(\mathcal{R}_k) \ \delta_{s_3'}$ that is $\mu_2 \ \mathcal{L}(\mathcal{R}_k) \ \mu_3$ and thus $s_2 \xrightarrow{a} \mu_2$ is matched by $s_3 \xrightarrow{a} \mu_3$. Since each transition is also a weak combined 2-bounded

transition, it follows that $s_2 \xrightarrow{a} \mu_2$ is matched by $s_3 \stackrel{a}{\Longrightarrow}{}^2_C \mu_3$, as required.

all other cases: the argumentation is the same of the previous case.

Since all conditions are satisfied, we have that $\mathcal{B}^2_k \preccurlyeq^2 \mathcal{B}^3_k$. Thus, by Proposition 5.16, it follows that $\{\mathcal{B}^2_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}^3_k\}_{k\in\mathbb{N}}$ and hence, by Theorem 5.10, we have that $\{\text{Hide}_{UC}(\mathcal{A}^6_k)\}_{k\in\mathbb{N}} \lesssim_s \{\text{Hide}_{UC}(\mathcal{A}^7_k)\}_{k\in\mathbb{N}}$. □

Lemma 8.22. *Let $\mathbb{A}$ be a set of agents and for each $k \in \mathbb{N}$, denote by $\mathcal{A}^7_k$ the automaton $NG^3_k(\mathbb{A})||\mathcal{E}^3_k(\mathbb{A})||\mathcal{S}^2_k(\mathbb{A})||MNEPAdv^{adv}_k(\mathbb{A})$ and by $\mathcal{A}^8_k$ the automaton $NG^3_k(\mathbb{A})||\mathcal{E}^3_k(\mathbb{A})||\mathcal{S}^3_k(\mathbb{A})||MNESPAdv^{adv}_k(\mathbb{A})$ where $\mathcal{S}^3_k(\mathbb{A})$ is the signature oracle automaton defined as in Section 6.3.1. Let $US = \{used\_signatures(S) \mid S \subseteq \text{Signature}\}$.*

Then, $\{\text{Hide}_{US}(\mathcal{A}^7_k)\}_{k\in\mathbb{N}} \lesssim_s \{\text{Hide}_{US}(\mathcal{A}^8_k)\}_{k\in\mathbb{N}}$.

Proof. To simplify the proof, denote by $\mathcal{B}^2_k$ and $\mathcal{B}^3_k$ the composed automata $\text{Hide}_{US}(\mathcal{S}^2_k(\mathbb{A})||MPAdv^{adv}_k(\mathbb{A}))$ and $\text{Hide}_{US}(\mathcal{S}^3_k(\mathbb{A})||MNEPAdv^{adv}_k(\mathbb{A}))$.

We prove the Lemma showing a stronger result, that is that the automaton $\mathcal{B}^2_k$ is weakly simulated by $\mathcal{B}^3_k$ and that each matching weak transition has length at most 2. Thus, by Proposition 5.16, it follows that $\{\mathcal{B}^2_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}^3_k\}_{k\in\mathbb{N}}$ and hence, by Theorem 5.10, we have that $\{\text{Hide}_{UN_k}(\mathcal{A}^5_k)\}_{k\in\mathbb{N}} \lesssim_s \{\text{Hide}_{UN_k}(\mathcal{A}^7_k)\}_{k\in\mathbb{N}}$.

Let S^2_k and S^3_k be the sets of states of $\mathcal{B}^2_k$ and $\mathcal{B}^3_k$, respectively, and $\mathcal{R}_k \subseteq S^2_k \times S^3_k$ be the relation defined as $s_2\ \mathcal{R}_k\ s_3$ if and only if $s_2 = s_3{\restriction_{s_2}}$ and $s_3.extr\_signatures = \bot \iff s_2.extr\_ciphers = \bot$.

The condition on start states is trivially verified, since by definition of $\mathcal{B}^2_k$ and $\mathcal{B}^3_k$, it follows that the two automata initialize the common state variables with the same values and thus $\bar{s}^2_k = \bar{s}^3_k{\restriction_{\bar{s}^2_k}}$. Moreover, by definition of $\mathcal{B}^3_k$, we have that $s_3.extr\_signatures = \bot$ and $s_2.extr\_ciphers = \bot$ and thus $\bar{s}^2_k\ \mathcal{R}_k\ \bar{s}^3_k$, as required.

For the step condition, let s_2 and s_3 be two states such that $s_2\ \mathcal{R}_k\ s_3$ and suppose that $s_2 \xrightarrow{a} \mu_2$. We must find μ_3 such that $s_3 \stackrel{a}{\Longrightarrow}{}^2_C \mu_3$ such that $\mu_2\ \mathcal{L}(\mathcal{R}_k)\ \mu_3$. There are several cases:

case $a = used\_ciphers(C)$: by definition of $used\_ciphers(C)$, it follows that $s_2.extr\_ciphers = C$. Moreover, we have that $\mu_2 = \delta_{s'_2}$ where s'_2 is the state of $\mathcal{B}^2_k$ that is identified by the same values of s_2 except for the following values: $s'_2.extr\_ciphers = \bot$, $s'_2.c = update\_c\_map(c', left', \tau', \sigma')$, $s'_2.\tau = \tau'$, $s'_2.symbolic = \sigma'(left')$, $s'_2.H = s_2.H \cup \{\sigma'(right')\}$, and $s'_2.status = \mathsf{parse\_right}$ where $\sigma' = s_2.\sigma$, $left' = s_2.left$, $right' =$

$s_2.right$, and $(\tau', c') = EvalRight(right', s_2.\tau, \sigma', s_2.c)$. Since $s_2 \ \mathcal{R}_k \ s_3$, it follows that also s_3 satisfies $s_3.extr\_ciphers = C$. Moreover, it satisfies $s_3.extr\_signatures \neq \bot$ and thus we have that s_3 enables the transition $s_3 \xrightarrow{a} \mu'_3$ where $\mu'_3 = \delta_{s'_3}$ where s'_3 is the state of $\mathcal{B}^3_k$ that is identified by the same values of s_3 except for the $extr\_ciphers$ variable where $s'_3.extr\_ciphers = \bot$. This implies that s'_3 enables the transition labelled by $used\_signatures(S)$ (where $S = s'_3.extr\_signatures \neq \bot$) that leads to $\delta_{s''_3}$ where s''_3 is the state of $\mathcal{B}^3_k$ that is identified by the same values of s'_3 except for the following values: $s''_3.used\_signatures = s'_3.used\_signatures \cup S$, $s''_3.extr\_signatures = \bot$, $s''_3.c = update\_c\_map(c', left', \tau', \sigma')$, $s''_3.\tau = \tau'$, $s''_3.symbolic = \sigma'(left')$, $s''_3.H = s'_3.H \cup \{\sigma'(right')\}$, and $s''_3.status = \mathsf{parse\_right}$ where $\sigma' = s'_3.\sigma$, $left' = s'_3.left$, $right' = s'_3.right$, and (τ', c') is the output of $EvalRight(right', s'_3.\tau, \sigma', s'_3.c)$. Since s'_3 and s_3 differ only on the value of the $extr\_signatures$ variable, it follows that $s''_3.extr\_signatures = \bot$, $s''_3.c = update\_c\_map(c', left', \tau', \sigma')$, $s''_3.\tau = \tau'$, $s''_3.symbolic = \sigma'(left')$, $s''_3.H = s_3.H \cup \{\sigma'(right')\}$, and $s''_3.status = \mathsf{parse\_right}$ where $\sigma' = s_3.\sigma$, $left' = s_3.left$, $right' = s_3.right$, and (τ', c') is the output of $EvalRight(right', s_3.\tau, \sigma', s_3.c)$ and thus $\delta_{s'_2} \ \mathcal{L}(\mathcal{R}_k) \ \delta_{s''_3}$. This implies that there exists a weak combined 2-bounded transition $s_3 \stackrel{a}{\Longrightarrow}^2_C \mu_3$ such that $\mu_2 = \delta_{s'_2} \ \mathcal{L}(\mathcal{R}_k) \ \delta_{s''_3} = \mu_3$, as required.

case $a = get\_corrupt\_sign(\eta)$: by definition of $get\_corrupt\_sign(\eta)$ action, it follows that $s_2.status = \mathsf{get\_sig\_key}$ and $\mathrm{head}(s_2.agentsToCorrupt) = \eta$. Moreover, we have that $\mu_2 = \delta_{s'_2}$ where s'_2 is the state of $\mathcal{B}^2_k$ that is identified by the same values of s_2 except for the following values: $s'_2.history = s_2.history \vdash get\_corrupt\_sign(\eta)$ and $s'_2.status = \mathsf{wait\_sig\_key}$. Since $s_2 \ \mathcal{R}_k \ s_3$, it follows that also s_3 satisfies $s_3.status = \mathsf{get\_sig\_key}$ and $\mathrm{head}(s_3.agentsToCorrupt) = \eta$. Thus s_3 enables the transition $s_3 \xrightarrow{a} \mu_3$ where $\mu_3 = \delta_{s'_3}$ where s'_3 is the state of $\mathcal{B}^3_k$ that is identified by the same values of s_3 except for the following values: $s'_3.history = s_3.history \vdash get\_corrupt\_sign(\eta)$ and $s'_3.status = \mathsf{wait\_sig\_key}$. This implies that $\delta_{s'_2} \ \mathcal{L}(\mathcal{R}_k) \ \delta_{s'_3}$ that is $\mu_2 \ \mathcal{L}(\mathcal{R}_k) \ \mu_3$ and thus $s_2 \xrightarrow{a} \mu_2$ is matched by $s_3 \xrightarrow{a} \mu_3$. Since each transition is also a weak combined 2-bounded transition, it follows that $s_2 \xrightarrow{a} \mu_2$ is matched by $s_3 \stackrel{a}{\Longrightarrow}^2_C \mu_3$, as required.

all other cases: the argumentation is the same of the previous case.

Since all conditions are satisfied, we have that $\mathcal{B}_k^2 \preccurlyeq^2 \mathcal{B}_k^3$. Thus, by Proposition 5.16, it follows that $\{\mathcal{B}_k^2\}_{k\in\mathbb{N}} \lessapprox_s \{\mathcal{B}_k^3\}_{k\in\mathbb{N}}$ and hence, by Theorem 5.10, we have that $\{\text{Hide}_{US}(\mathcal{A}_k^7)\}_{k\in\mathbb{N}} \lessapprox_s \{\text{Hide}_{US}(\mathcal{A}_k^8)\}_{k\in\mathbb{N}}$. □

Lemma 8.23. *Let $\mathbb{A}$ be a set of agents and for each $k \in \mathbb{N}$, denote by $\mathcal{A}_k^8$ the automaton $NG_k^3(\mathbb{A})||\mathcal{E}_k^3(\mathbb{A})||\mathcal{S}_k^3(\mathbb{A})||MNESPAdv_k^{adv}(\mathbb{A})$ and by $\mathcal{A}_k^9$ the automaton $NG_k^4(\mathbb{A})||\mathcal{E}_k^3(\mathbb{A})||\mathcal{S}_k^3(\mathbb{A})||MNESPAdv_k^{adv}(\mathbb{A})$ where $NG_k^4(\mathbb{A})$ is the nonce generator automaton defined in Section 6.1.1.*

Then there exists $\mathcal{B}_k$ such that $\{\mathcal{A}_k^8\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^9\}_{k\in\mathbb{N}}$.

Proof. Let G_k be the set of states s of NG_k^3 such that for all $A \in \mathbb{A}$, $s.is\_not\_used_A \neq F$. By Propositions 6.10 and 6.9, we know that for each context $\mathcal{C}_k$ compatible with $NG_k^3(\mathbb{A})$ we have the following chain of simulations: $\{NG_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(NG_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{NG_k^4(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$, provided that there exists $q \in Poly$ such that for each $N \subseteq \{0,1\}^k$ such that $used\_nonces(N)$ is an action of $\mathcal{C}_k$, $|N| \leqslant q(k)$.

To prove the statement of the Lemma, it is sufficient to show that $\bar{\mathcal{C}}_k = \mathcal{E}_k^3(\mathbb{A})||\mathcal{S}_k^3(\mathbb{A})||MNESPAdv_k^{adv}(\mathbb{A})$ is a context compatible with $NG_k^4(\mathbb{A})$ such that there exists $q \in Poly$ such that for each $used\_nonces(N) \in A_{\bar{\mathcal{C}}_k}$, $N \subseteq \{0,1\}^k$, $|N| \leqslant q(k)$ holds.

$\bar{\mathcal{C}}_k$ is trivially compatible with $NG_k^4(\mathbb{A})$ since the set $H_{NG_k^4}$ of internal actions of $NG_k^4(\mathbb{A})$ is empty and thus $H_{NG_k^I} \cap A_{\bar{\mathcal{C}}_k} = \emptyset$ and the set $H_{\bar{\mathcal{C}}_k}$ is disjoint from the set $A_{NG_k^4}$ of actions of $NG_k^4(\mathbb{A})$ and thus $A_{NG_k^4} \cap H_{\bar{\mathcal{C}}_k} = \emptyset$. The last thing to check is that there exists $q \in Poly$ such that for each action $used\_nonces(N)$ of $\bar{\mathcal{C}}_k$, we have $|N| < q(k)$. By the definition of $used\_nonces(N)$, it follows that the cardinality of the set of nonces N depends on the output of the function $extractNonces(t,\tau)$ where t is a term and τ a mapping terms to bitstrings. By Proposition 8.11, we know that $|N| \leqslant \text{size}(t)$. In particular, the terms t that are used as input of $extractNonces(t,\tau)$ are the left-hand side of the roles that describe the protocol. This means that given a n-party protocol, we are able to find $r \in \mathbb{N}$ such that for each left-hand side term t of the roles of the protocol, $\text{size}(t) < r$. Let $q \in Poly$ be the constant polynomial $q(k) = r$. Then $|N| \leqslant \text{size}(t) < r = q(k)$ and thus $|N| < q(k)$, as required. □

Lemma 8.24. *Let E be an IND-CCA encryption scheme and $\mathbb{A}$ be a set of agents. For each $k \in \mathbb{N}$, denote by $\mathcal{A}_k^9$ the composed automaton $NG_k^4(\mathbb{A})||\mathcal{E}_k^3(\mathbb{A})||\mathcal{S}_k^3(\mathbb{A})||MNESPAdv_k^{adv}(\mathbb{A})$ and by $\mathcal{A}_k^{10}$ the automaton $NG_k^4(\mathbb{A})||\mathcal{E}_k^4(\mathbb{A})||\mathcal{S}_k^3(\mathbb{A})||MNESPAdv_k^{adv}(\mathbb{A})$ where $\mathcal{E}_k^4(\mathbb{A})$ is the encryption oracle automaton defined in Section 6.2.1.*

Then there exists $\mathcal{B}_k$ *such that* $\{\mathcal{A}_k^9\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^{10}\}_{k\in\mathbb{N}}$.

Proof. Let G_k be the set of states s of $\mathcal{E}_k^3$ such that for all $A \in \mathbb{A}$, $s.is\_not\_used_A \neq F$. By Propositions 6.20 and 6.19, we know that for each context $\mathcal{C}_k$ compatible with $\mathcal{E}_k^3(\mathbb{A})$ we have the following chain of simulations: $\{\mathcal{E}_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{E}_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{E}_k^4(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$, provided that there exists $q \in Poly$ such that for each action $used\_ciphers(C)$ of $\mathcal{C}_k$, $|C| \leqslant q(k)$.

To prove the statement of the Lemma, it is sufficient to show that $\bar{\mathcal{C}}_k = NG_k^4(\mathbb{A})||\mathcal{S}_k^3(\mathbb{A})||MNESPAdv_k^{adv}(\mathbb{A})$ is a context compatible with $\mathcal{E}_k^4(\mathbb{A})$ such that there exists $q \in Poly$ such that for each $used\_ciphers(C) \in A_{\bar{\mathcal{C}}_k}$, $C \subseteq$ Ciphertext, $|C| \leqslant q(k)$ holds.

$\bar{\mathcal{C}}_k$ is trivially compatible with $\mathcal{E}_k^4(\mathbb{A})$ since the set $H_{\mathcal{E}_k^4}$ of internal actions of $\mathcal{E}_k^4(\mathbb{A})$ is empty and thus $H_{\mathcal{E}_k^I} \cap A_{\bar{\mathcal{C}}_k} = \emptyset$ and the set $H_{\bar{\mathcal{C}}_k}$ is disjoint from the set $A_{\mathcal{E}_k^4}$ of actions of $\mathcal{E}_k^4(\mathbb{A})$ and thus $A_{\mathcal{E}_k^4} \cap H_{\bar{\mathcal{C}}_k} = \emptyset$. The last thing to check is that there exists $q \in Poly$ such that for each action $used\_ciphers(C)$ of $\bar{\mathcal{C}}_k$, we have $|C| < q(k)$. By the definition of $used\_ciphers(C)$, it follows that the cardinality of the set of ciphertexts C depends on the output of the function $extractCiphertexts(t, \tau)$ where t is a term and τ a mapping terms to bitstrings. By Proposition 8.12, we know that $|C| \leqslant \text{size}(t)$. In particular, the terms t that are used as input of $extractCiphertexts(t, \tau)$ are the left-hand side of the roles that describe the protocol. This means that given a n-party protocol, we are able to find $r \in \mathbb{N}$ such that for each left-hand side term t of the roles of the protocol, $\text{size}(t) < r$. Let $q \in Poly$ be the constant polynomial $q(k) = r$. Then $|C| \leqslant \text{size}(t) < r = q(k)$ and thus $|C| < q(k)$, as required. □

Lemma 8.25. *Let* S *be a non-repeating unforgeable signature scheme and* $\mathbb{A}$ *be a set of agents and for each* $k \in \mathbb{N}$*, denote by* $\mathcal{A}_k^{10}$ *the composed automaton* $NG_k^4(\mathbb{A})||\mathcal{E}_k^4(\mathbb{A})||\mathcal{S}_k^3(\mathbb{A})||MNESPAdv_k^{adv}(\mathbb{A})$ *and by* $\mathcal{A}_k^{11}$ *the automaton* $NG_k^4(\mathbb{A})||\mathcal{E}_k^4(\mathbb{A})||\mathcal{S}_k^4(\mathbb{A})||MNESPAdv_k^{adv}(\mathbb{A})$ *where* $\mathcal{S}_k^4(\mathbb{A})$ *is the signature oracle automaton defined in Section 6.3.1.*

Then there exists $\mathcal{B}_k$ *such that* $\{\mathcal{A}_k^{10}\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{B}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{A}_k^{11}\}_{k\in\mathbb{N}}$.

Proof. Let G_k be the set of states s of $\mathcal{S}_k^3$ such that for all $A \in \mathbb{A}$, $s.is\_not\_used_A \neq F$. By Propositions 6.36 and 6.35, we know that for each context $\mathcal{C}_k$ compatible with $\mathcal{S}_k^3(\mathbb{A})$ we have the following chain of simulations: $\{\mathcal{S}_k^3(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{(\mathcal{S}_k^3(\mathbb{A})|G_k)||\mathcal{C}_k\}_{k\in\mathbb{N}} \lesssim_s \{\mathcal{S}_k^4(\mathbb{A})||\mathcal{C}_k\}_{k\in\mathbb{N}}$, provided that there exists $q \in Poly$ such that for each action $used\_signatures(S)$ of $\mathcal{C}_k$, $|S| \leqslant q(k)$.

To prove the statement of the Lemma, it is sufficient to show that $\bar{\mathcal{C}}_k = NG_k^4(\mathbb{A})||\mathcal{E}_k^3(\mathbb{A})||MNESPAdv_k^{adv}(\mathbb{A})$ is a context compatible with $\mathcal{S}_k^4(\mathbb{A})$ such that there exists $q \in Poly$ such that for each $used\_signatures(S) \in A_{\bar{\mathcal{C}}_k}$, $C \subseteq$ Ciphertext, $|S| \leqslant q(k)$ holds.

$\bar{\mathcal{C}}_k$ is trivially compatible with $\mathcal{S}_k^4(\mathbb{A})$ since the set $H_{\mathcal{S}_k^4}$ of internal actions of $\mathcal{S}_k^4(\mathbb{A})$ is empty and thus $H_{\mathcal{S}_k^I} \cap A_{\bar{\mathcal{C}}_k} = \emptyset$ and the set $H_{\bar{\mathcal{C}}_k}$ is disjoint from the set $A_{\mathcal{S}_k^4}$ of actions of $\mathcal{S}_k^4(\mathbb{A})$ and thus $A_{\mathcal{S}_k^4} \cap H_{\bar{\mathcal{C}}_k} = \emptyset$. The last thing to check is that there exists $q \in Poly$ such that for each action $used\_signatures(S)$ of $\bar{\mathcal{C}}_k$, we have $|S| < q(k)$. By the definition of $used\_signatures(S)$, it follows that the cardinality of the set of signatures C depends on the output of the function $extractSignatures(t, \tau)$ where t is a term and τ a mapping terms to bitstrings. By Proposition 8.13, we know that $|S| \leqslant \text{size}(t)$. In particular, the terms t that are used as input of $extractSignatures(t, \tau)$ are the left-hand side of the roles that describe the protocol. This means that given a n-party protocol, we are able to find $r \in \mathbb{N}$ such that for each left-hand side term t of the roles of the protocol, $\text{size}(t) < r$. Let $q \in Poly$ be the constant polynomial $q(k) = r$. Then $|S| \leqslant \text{size}(t) < r = q(k)$ and thus $|S| < q(k)$, as required. □

Lemma 8.26. *Let $\mathbb{A}$ be a set of agents. For each $k \in \mathbb{N}$, denote by $\mathcal{A}_k^{11}$ the automaton $NG_k^4(\mathbb{A})||\mathcal{E}_k^4(\mathbb{A})||\mathcal{S}_k^4(\mathbb{A})||MNESPAdv_k^{adv}(\mathbb{A})$. Let $\bar{E}_k$ be the set $\{corrupt(\eta_1 \dots \eta_l), new(i, \eta_1 \dots \eta_l), send(cs, m)\}$ and $\bar{A}_k$ be the set $A_k^{11} \setminus \bar{E}_k$.*

If E is an IND-CCA encryption scheme and S is a non-repeating unforgeable signature scheme, then $\{\text{Hide}_{\bar{A}_k}(\mathcal{A}_k^{11})\}_{k\in\mathbb{N}} \leq_s^{\approx} \{SAdv\}_{k\in\mathbb{N}}$.

Before proving the Lemma, we need of a preliminary result. The first thing we need to define is a function that allows us to abstract from the actual value of the bitstring but that takes into account the structure of the input. Let $class(\,\cdot\,)$ be a function defined inductively as follows:

- if $x \in \mathbb{N}$, then $class(x) = \mathbb{N}$;
- if $x \in$ Message and $type(x) = id$, then $class(x) = x$;
- if $x \in$ Message and $type(x) = nonce$, then $class(x) = nonce$;
- if $x \in$ Message and $type(x) = ek$, then $class(x) = ek$;
- if $x \in$ Message and $type(x) = dk$, then $class(x) = dk$;
- if $x \in$ Message and $type(x) = sk$, then $class(x) = sk$;
- if $x \in$ Message and $type(x) = vk$, then $class(x) = vk$;
- if $x \in$ Message and $type(x) = ek$, then $class(x) = ek$;
- if $x_1, x_2 \in$ Message, $type(x_1) = t_1$, and $type(x_2) = t_2$, then $class(\langle x_1, x_2\rangle) = \langle t_1, t_2\rangle$;

- if $x \in$ Message, $type(x) = t$, and $type(e) = ek$, then $class(\{x\}_e) = \{t\}_{ek}$;
- if $x \in$ Message, $type(x) = t$, and $type(s) = sk$, then $class([x]_s) = [t]_{sk}$;
- $class((x_1, x_2)) = (class(x_1), class(x_2))$;
- $class(\lambda) = \lambda$ and $class(x \vdash S) = class(x) \vdash class(S)$;
- $class(x_1 x_2 \dots x_n) = class(x_1) class(x_2 \dots x_n)$;
- $class(\emptyset) = \emptyset$ and $class(\mathbf{push}(\mathtt{S}, x)) = \mathbf{push}(class(\mathtt{S}), class(x))$;
- $class(action(S)) = action(class(S))$;
- $class(s) = s'$ where for each state variable v of s, $s'.v = class(s.v)$;
- for all other cases, $class(x) = x$.

Lemma 8.27. *Let $\mathcal{A}_k^{11}$ and $\bar{E}_k$ be the automaton and the set of actions defined as in Lemma 8.26. Let $\xi = s_0 a_1 s_1 \dots a_n s_n$ be an execution fragment of $\mathcal{A}_k^{11}$ such that for each $0 \leqslant i < n$, $s_i.action = \bot$. An execution fragment $\xi = s_0 a_1 s_1 \dots a_n s_n$ is maximal if $s_n.action \neq \bot$. Let $\Xi \subseteq Frags^*(\mathcal{A}_k^{11})$ be the set of finite maximal execution fragments ξ defined as above.*

For each pair of transitions $s \xrightarrow{a_1} \mu$ and $s' \xrightarrow{a'_1} \mu'$ such that $class(s) = class(s')$ and $class(a_1) = class(a'_1)$ and for each pair of states s_1 and s'_1 such that $s_1, s'_1 \in Supp(\mu)$ and $s_1, s'_1 \in Supp(\mu)$, if $\xi = s a_1 s_1 \dots a_n s_n \in \Xi$, then there exists $\xi' = s' a'_1 s'_1 \dots a'_n s'_n \in \Xi$ such that $class(\xi) = class(\xi')$.

Proof. The proof is a classical inductive proof on the length of ξ. □

Lemma 8.28. *Let $\mathcal{A}_k^{11}$ and $\bar{E}_k$ be the automaton and the set of actions defined as in Lemma 8.26. Let $\xi = s_0 a_1 s_1 \dots a_n s_n$ be an execution fragment of $\mathcal{A}_k^{11}$ such that for each $0 \leqslant i < n$, $s_i.action = \bot$. An execution fragment $\xi = s_0 a_1 s_1 \dots a_n s_n$ is maximal if $s_n.action \neq \bot$. Let $\Xi \subseteq Frags^*(\mathcal{A}_k^{11})$ be the set of finite maximal execution fragments ξ defined as above.*

For each state s such that $s.agentsToCorrupt \neq \lambda$ or $s.agents\_to\_init \neq \lambda$, there exists $\xi = s a_1 s_1 \dots a_n s_n \in \Xi$ such that for each $1 \leqslant i \leqslant n$, $s_i.H = s.H$, $s_i.ASId = s.ASId$, and $s_i.F = s.F$.

Proof. The proof is a classical inductive proof on the length of ξ. □

Lemma 8.29. *Let $\mathcal{A}_k^{11}$ and $\bar{E}_k$ be the automaton and the set of actions defined as in Lemma 8.26. Let $\xi = s_0 a_1 s_1 \dots a_n s_n$ be an execution fragment of $\mathcal{A}_k^{11}$ such that for each $0 \leqslant i < n$, $s_i.action = \bot$. An execution fragment $\xi = s_0 a_1 s_1 \dots a_n s_n$ is maximal if $s_n.action \neq \bot$. Let $\Xi \subseteq Frags^*(\mathcal{A}_k^{11})$ be the set of finite maximal execution fragments ξ defined as above.*

For each state s such that there exists s' such that $(s', send(cs, m), \delta_s)$ is a transition of $\mathcal{A}_k^{11}$, if there does not exist θ such that $m = (\sigma \circ \theta)(l_p^j)$

where $(\sigma, j, p) = (s.F)((s.c)^{-1}(cs))$ *then there exists* $\xi = sa_1s_1 \dots a_ns_n \in \Xi$ *such that* $a_{n-1} = fail\_step$ *and for each* $1 \leqslant i \leqslant n$, $s_i.H = s.H$, $s_i.ASId = s.ASId$, *and* $s_i.F = s.F$.

Proof. The proof is a classical inductive proof on the length of ξ. □

Lemma 8.30. *Let* $\mathcal{A}_k^{11}$ *and* $\bar{E}_k$ *be the automaton and the set of actions defined as in Lemma 8.26. Let* $\xi = s_0a_1s_1 \dots a_ns_n$ *be an execution fragment of* $\mathcal{A}_k^{11}$ *such that for each* $0 \leqslant i < n$, $s_i.action = \perp$. *An execution fragment* $\xi = s_0a_1s_1 \dots a_ns_n$ *is maximal if* $s_n.action \neq \perp$. *Let* $\Xi \subseteq Frags^*(\mathcal{A}_k^{11})$ *be the set of finite maximal execution fragments* ξ *defined as above.*

For each state s *such that there exists* s' *such that* $(s', send(cs, m), \delta_s)$ *is a transition of* $\mathcal{A}_k^{11}$, *if there exists* θ *such that* $m = (\sigma \circ \theta)(l_p^j)$ *where* $(\sigma, j, p) = (s.F)((s.c)^{-1}(cs))$ *then there exists* $\xi = sa_1s_1 \dots a_ns_n \in \Xi$ *such that* $a_{n-1} = end\_step$, $s_n.H = s.H \cup \{(\sigma \circ \theta)(r_p^j)\}$, *and* $s_n.F((s'.c)^{-1}(cs)) = (\sigma \circ \theta, j, p+1)$ *where* $(\sigma, i, p) = (s'.F)((s'.c^{-1})(cs))$.

Proof. The proof is a classical inductive proof on the length of ξ. □

Now we are able to prove the statement of the Lemma 8.26:

Proof of Lemma 8.26. Let Ξ be the set of executions fragments of $\mathcal{A}_k^{11}$ defined as in Lemma 8.27. Let $\mathcal{R}_k \subseteq S_k^{11} \times S_k^{12}$ be the relation defined as: $s_k^{11}\ \mathcal{R}_k\ s_k^{12}$ if and only if one of the following conditions holds:

- $s_k^{11}.action \in \{new, send\}$ and $s_k^{12} = s_k^{11}\lceil_{s_k^{12}}$;
- $s_k^{11}.action = corrupt$, $s_k^{11}.history = \lambda$, $s_k^{12}.h = \lambda$, and $s_k^{12} = s_k^{11}\lceil_{s_k^{12}}$; or
- for each transition enabled by s_k^{11} and labelled by a, there exist $\xi = s_0as_1 \dots a_ns_n \in \Xi$ such that $s_k^{11} = s_0$, $s_k^{12} = s_n\lceil_{s_k^{12}}$.

Then $\{\mathcal{R}_k\}_{k\in\mathbb{N}}$ is a state weak polynomially accurate simulation from $\{\text{Hide}_{\bar{A}_k}(\mathcal{A}_k^{11})\}_{k\in\mathbb{N}}$ to $\{SAdv\}_{k\in\mathbb{N}}$.

Start condition is trivially true, since by definition of $\mathcal{A}_k^{11}$ and of $\mathcal{A}_k^{12}$ it follows that $s.action \neq \perp$ $s.F = \emptyset$, $\bar{s}_k^{12}.F = \emptyset$, $s.ASId = \emptyset$, $\bar{s}_k^{12}.ASId = \emptyset$, $s.H = \emptyset$, and $\bar{s}_k^{12}.H = \emptyset$ where $s \in Supp(\mu)$ and $(\bar{s}_k^{11}, create\_action, \mu)$ is a transition of $\mathcal{A}_k^{11}$ and thus $\bar{s}_k^{11} create\_actions \in \Xi$.

Suppose, for the sake of contradiction, that the step condition does not hold. This means that there exists $c \in \mathbb{N}$ and $p \in Poly$ such that for each $l \in Poly$ and $\bar{k} \in \mathbb{N}$ there exist $k > \bar{k}$, μ_{11}, μ_{12} and $\gamma \geqslant 0$ such that μ_{11} is reached within $p(k)$ steps in $\mathcal{A}_k^{11}$ and for each ε-weighting function w_γ for μ_{11} $\mathcal{L}_w(\mathcal{R}_k, \gamma)$ μ_{12} we have that $\sum\{w_\gamma(s_{11}, s_{12}) \mid s_{11} \neg\ \mathcal{R}_k^{l(k)}(k^{-c})$

$s_{12}\} \geqslant k^{-c}$. Let s_{11} and s_{12} be two states such that $s_{11} \neg\ \mathcal{R}_k^{l(k)}(k^{-c})\ s_{12}$. This implies that either $s_{11} \neg\ \mathcal{R}_k\ s_{12}$ or $s_{11}\ \mathcal{R}_k\ s_{12}$ and there exists $tr_{11} = (s_{11}, a, \rho_{11}) \in D_k^{11}$ such that for each weak $l(k)$-bounded combined transition $tr_{12} = (s_{12}, a, \rho_{12})$, $\rho_{11} \neg\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \rho_{12}$. In the former case, $s_{11} \neg\ \mathcal{R}_k\ s_{12}$ implies that $w_\gamma(s_{11}, s_{12}) = 0$. Otherwise, if $w_\gamma(s_{11}, s_{12}) > 0$, then by property 1 of the definition of ε-weighting function, we have that $s_{11}\ \mathcal{R}_k\ s_{12}$. So, $\sum\{w_\gamma(s_{11}, s_{12}) \mid s_{11} \neg\ \mathcal{R}_k^{l(k)}(k^{-c})\ s_{12}\} \geqslant k^{-c}$ is due to pair of states (s_{11}, s_{12}) such that $s_{11}\ \mathcal{R}_k\ s_{12}$ and there exists $tr_{11} = (s_{11}, a, \rho_{11}) \in D_k^{11}$ such that for each weak $l(k)$-bounded combined transition $tr_{12} = (s_{12}, a, \rho_{12})$, $\rho_{11} \neg\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \rho_{12}$. Take a pair of such states, say (s_{11}, s_{12}), and suppose that $s_{11} \xrightarrow{a} \rho_{11}$. Now there are two cases: either $s_{11}.action \notin \{corrupt, new, send\}$ or $s_{11}.action \in \{corrupt, new, send\}$. In the former case, by definition of $\mathcal{R}_k$ it follows that there exists a maximal execution fragment $\xi = q_0 a q_1 \ldots a_n q_n \in \Xi$ such that $s_{11} = q_0$ and $s_k^{12} = q_n{\restriction}_{s_k^{12}}$. This implies, by Lemma 8.27, that for each $q_1' \in Supp(\rho_{11})$ there exists $\xi' = s_{11} a q_1' \ldots a_n' q_n' \in \Xi$ such that $class(\xi) = class(\xi')$ and thus $s_k^{12} = q_n'{\restriction}_{s_k^{12}}$. By definition of $\mathcal{R}_k$, it follows that $q_1'\ \mathcal{R}_k\ s_{12}$ and since this happens for all $q_1' \in Supp(\rho_1)$, it follows that $\rho_1\ \mathcal{L}_w(\mathcal{R}_k, 0)\ \delta_{s_{12}}$ and thus $\rho_1\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s_{12}}$. Since $s_{11}.action \notin \{corrupt, new, send\}$, by definition of *corrupt*, *new*, and *send* actions it follows that $a \in \bar{A}_k$ and thus a is an internal action of $\mathrm{Hide}_{\bar{A}_k}(\mathcal{A}_k^{11})$ thus the transition $s_{12} \stackrel{\tau}{\Longrightarrow}_C \delta_{s_{12}}$ is a valid weak $l(k)$-bounded combined transition such that $\rho_{11} \neg\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \rho_{12}$.

Now, suppose that $s_{11}.action \in \{corrupt, new, send\}$. This implies that s_{11} enables a transition labelled by either $corrupt(\eta_1 \ldots \eta_l)$, $new(i, \eta_1 \ldots \eta_l)$, or $send(cs, m)$ provided that $s_{11}.action = corrupt$, *new*, and *send*, respectively:

case $a = corrupt(\eta_1 \ldots \eta_l)$: by definition of action $corrupt(\eta_1 \ldots \eta_l)$ it follows that $s_{11}.history = \lambda$, $s_{11}.action = corrupt$ and $\rho_1 = \delta_{s_{11}'}$ where s_{11}' is the state of $\mathcal{A}_k^{11}$ that is identified by the same values of s_{11} except for the following values: $s_{11}'.history = corrupt(\eta_1 \ldots \eta_l)$, $s_{11}'.action = \bot$, $s_{11}'.agentsToCorrupt = \eta_1 \ldots \eta_l$, $s_{11}'.status = \mathsf{get\_sig\_key}$, $s_{11}'.c = c'$, $s_{11}'.symbolic = corrupt(\bar{c}(\eta_1) \ldots \bar{c}(\eta_l))$, $s_{11}'.H = s_{11}.H \cup \mathbf{kn}(\bar{c}(\eta_1)) \cup \cdots \cup \mathbf{kn}(\bar{c}(\eta_l))$, where $c' = update\_ag\_names(s_{11}.c, \eta_1 \ldots \eta_l)$ and $\bar{c} = (c')^{-1}$. This implies, by definition of $\mathcal{R}_k$, that s_{12} satisfies $s_{12}.h = \lambda$ and thus it enables the transition $(s_{12}, corrupt(\bar{c}(\eta_1) \ldots \bar{c}(\eta_l)), \delta_{s_{12}'})$ where s_{12}' is the state of $\mathcal{A}_k^{12}$ that is identified by the same values of s_{12} except for the following values: $s_{12}'.h = corrupt(\bar{c}(\eta_1) \ldots \bar{c}(\eta_l))$ and

$s'_{12}.H = s_{12}.H \cup \mathbf{kn}(\bar{c}(\eta_1)) \cup \cdots \cup \mathbf{kn}(\bar{c}(\eta_l))$ and thus $s'_{12} = s'_{11}\restriction_{s'_{12}}$. By Lemma 8.28 it follows that there exists $\xi = s_0 a_1 s_1 \ldots a_n s_n \in \Xi$ such that $s_0 = s'_{11}$ and $s'_{12} = s'_{11}\restriction_{s'_{12}}$ and thus $s'_{11}\ \mathcal{R}_k\ s'_{12}$, so $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, 0)\ \delta_{s'_{12}}$ that implies $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s'_{12}}$ and thus the transition $(s_{12}, \mathit{corrupt}(\bar{c}(\eta_1)\ldots\bar{c}(\eta_l)), \delta_{s'_{12}})$ is a valid weak $l(k)$-bounded combined transition such that $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s'_{12}}$.

case $a = \mathit{new}(i, \eta_1 \ldots \eta_l)$: by definition of action $\mathit{new}(i, \eta_1 \ldots \eta_l)$ it follows that $s_{11}.\mathit{action} = \mathit{new}$ and $\rho_1 = \delta_{s'_{11}}$ where s'_{11} is the state of $\mathcal{A}_k^{11}$ that is identified by the same values of s_{11} except for the following values: $s'_{11}.\mathit{history} = s_{11}.\mathit{history} \vdash \mathit{new}(i, \eta_1 \ldots \eta_l)$, $s'_{11}.\mathit{action} = \bot$, $s'_{11}.c = c'$, $s'_{11}.\mathit{CSId} = s_{11}.\mathit{CSId} \cup \{cs\}$, $s'_{11}.f = s_{11}.f \cup \{(cs, (\tau, i, 1))\}$, $s'_{11}.\mathit{ASId} = s_{11}.\mathit{ASId} \cup \{as\}$, $s'_{11}.F = s_{11}.F \cup \{(as, (\sigma, i, 1))\}$, $s'_{11}.\mathit{agents\_to\_init} = (\eta_1 \ldots \eta_l) - \{\eta \mid \mathit{ek}(\eta) \neq \bot\}$, $s'_{11}.\mathit{status} = \mathbf{if}\ s'_{11}.\mathit{agents\_to\_init} \neq \lambda\ \mathbf{then}\ \mathsf{get\_sig\_key}\ \mathbf{else}\ \mathsf{create}$, $s'_{11}.\mathit{symbolic} = \mathit{new}(i, \bar{c}(\eta_1)\ldots\bar{c}(\eta_l))$, $s'_{11}.H = s_{11}.H \cup \{(i, \bar{c}(\eta_1)\ldots\bar{c}(\eta_l))\}$ where $\tau = \{(A_j, \eta_j) \mid 1 \leqslant j \leqslant l\}$, $cs = (|s_{11}.\mathit{CSId}|+1, i, (\eta_1 \ldots \eta_l))$, $c' = \mathit{update\_ag\_names}(s_{11}.c, \eta_1 \ldots \eta_l)$, $\bar{c} = (c')^{-1}$, $as = (|s_{11}.\mathit{ASId}|+1, i, (\bar{c}(\eta_1)\ldots\bar{c}(\eta_l)))$, and $\sigma = \{(A_j, \bar{c}(\eta_j)) \mid 1 \leqslant j \leqslant l\} \cup \{(X^j_{\bar{c}(\eta_i)}, n(\bar{c}(\eta_i), j, as)) \mid j \in \mathbb{N}\}$. s_{12} enables the transition $(s_{12}, \mathit{new}(i, \bar{c}(\eta_1)\ldots\bar{c}(\eta_l)), \delta_{s'_{12}})$ where s'_{12} is the state of $\mathcal{A}_k^{12}$ that is identified by the same values of s_{12} except for the following values: $s'_{11}.H = s_{11}.H \cup \{(i, \bar{c}(\eta_1)\ldots\bar{c}(\eta_l))\}$, $s'_{12}.\mathit{ASId} = s_{12}.\mathit{ASId} \cup \{as\}$, $s'_{12}.F = s_{12}.F \cup \{(as, (\sigma, i, 1))\}$ where $as = (|s_{12}.\mathit{ASId}| + 1, i, (\bar{c}(\eta_1)\ldots\bar{c}(\eta_l)))$ and $\sigma = \{(A_j, \bar{c}(\eta_j)) \mid 1 \leqslant j \leqslant l\} \cup \{(X^j_{\bar{c}(\eta_i)}, n(\bar{c}(\eta_i), j, as)) \mid j \in \mathbb{N}\}$ and thus $s'_{12} = s'_{11}\restriction_{s'_{12}}$. If $s'_{11}.\mathit{agents\_to\_init} \neq \bot$, then by Lemma 8.28 it follows that there exists $\xi = s_0 a_1 s_1 \ldots a_n s_n \in \Xi$ such that $s_0 = s'_{11}$ and $s'_{12} = s'_{11}\restriction_{s'_{12}}$ and thus $s'_{11}\ \mathcal{R}_k\ s'_{12}$, so $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, 0)\ \delta_{s'_{12}}$ that implies $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s'_{12}}$ and thus $(s_{12}, \mathit{new}(i, \bar{c}(\eta_1)\ldots\bar{c}(\eta_l)), \delta_{s'_{12}})$ is a valid weak $l(k)$-bounded combined transition such that $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s'_{12}}$. If $s'_{11}.\mathit{agents\_to\_init} = \bot$, then $s'_{11}.\mathit{status} = \mathsf{create}$ and thus s'_{11} enables a transition labelled by *create_action* that by definition of the action reaches only states s''_{11} where only variables *action*, *message*, and *status* are modified. This implies that $s'_{12} = s''_{11}\restriction_{s'_{12}}$, thus $s'_{11}\mathit{create\_actions}''_{11} \in \Xi$ and hence $s'_{11}\ \mathcal{R}_k\ s'_{12}$. This implies that $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, 0)\ \delta_{s'_{12}}$, hence $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s'_{12}}$ and thus $(s_{12}, \mathit{new}(i, \bar{c}(\eta_1)\ldots\bar{c}(\eta_l)), \delta_{s'_{12}})$ is a valid weak $l(k)$-bounded combined transition such that $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s'_{12}}$.

case $a = \mathit{send}(cs, m)$: by definition of action $\mathit{send}(cs, m)$ it follows that $s_{11}.\mathit{action} = \mathit{send}$ and $\rho_1 = \delta_{s'_{11}}$ where s'_{11} is the state of $\mathcal{A}_k^{11}$ that is identified by the same values of s_{11} except for the following values:

$s'_{11}.history = s_{11}.history \vdash send(cs, m)$, $s'_{11}.action = \bot$, $s'_{11}.csid = cs$, $s'_{11}.status = \mathsf{parse\_left}$, and if $p \leqslant k_j$, then $s'_{11}.left = l_p^j$, $s'_{11}.right = r_p^j$, $(s'_{11}.\mathsf{S}, s'_{11}.\tau, s'_{11}.\sigma, s'_{11}.c) = EvalLeft(\mathbf{push}(\emptyset, (l_p^j, m)), \rho, \phi, s_{11}.c)$, else $(s'_{11}.\mathsf{S}, s'_{11}.\tau) = (\emptyset, \bot)$, where $(\rho, j, p) = s_{11}.f(cs)$ and $(\phi, j, p) = s_{11}.F((s_{11}.c)^{-1}(cs))$. If $s_{12}.H \not\vdash m$, then $\mathcal{A}_k^{12}$ is not able to simulate the transition performed by $\mathcal{A}_k^{11}$ from the state s_{11}. This implies that the message m, by Lemma 8.6, that either m contains a forged signature, or m contains a message that $\mathcal{A}_k^{11}$ has obtained decrypting a ciphertext which decryption key is not known by $\mathcal{A}_k^{11}$. Since this happens with non-negligible probability, it follows that $\mathcal{A}_k^{11}$ is either a forger for the signature scheme S or a distinguisher for the encryption scheme E. But this violates the hypothesis that E is an IND-CCA encryption scheme and S is a non-repeating unforgeable signature scheme. So, suppose that $s_{12}.H \vdash m$. If $p > k_j$, then it follows that $s'_{11}.\tau = \bot$ and $s'_{11}.status = \mathsf{parse\_left}$. This implies that s'_{11} enables a transition labelled by *fail_step* that leads to the measure $\delta_{s''_{11}}$ that is the same of s'_{11} except for the following values: $s''_{11}.status = \mathsf{create}$ and $s''_{11}.symbolic = \bot$. Also s_{12} enables the transition $(s_{12}, send((s_{11}.c)^{-1}(cs), m), \delta_{s'_{12}})$ where $s'_{12} = s_{12}$, since $p > k_j$. This implies that $s_{12} = s_{11}\lceil_{s_{12}} = s'_{11}\lceil_{s_{12}} = s''_{11}\lceil_{s_{12}} = s''_{11}\lceil_{s'_{12}} = s'_{12}$. This implies, by definition of $\mathcal{R}_k$, that $s'_{11}\ \mathcal{R}_k\ s'_{12}$, thus $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, 0)\ \delta_{s'_{12}}$, hence $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s'_{12}}$ and thus $(s_{12}, send((s_{11}.c)^{-1}(cs), m), \delta_{s'_{12}})$ is a valid weak $l(k)$-bounded combined transition such that $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s'_{12}}$. Now, suppose that $p \leqslant k_j$. If there does not exist θ such that $m = (\sigma \circ \theta)(l_p^j)$ where $(\sigma, j, p) = (s_{12}.F)((s_{11}.c)^{-1}(cs))$, then this implies that there exists some subterm t of l_p^j such that $\sigma(t)$ is different from the actual value of t in m. This implies, by definition of *EvalLeft* function, that the test $\tau(t) \neq \bot$ of **if** $\tau(t) \neq \bot$ **return**(**if** $\tau(t) = m$ **then** $(\emptyset, \tau, \theta, c)$ **else** $(\emptyset, \bot, \theta, c)$) is satisfied while the test $\tau(t) = m$ fails. By Lemma 8.29, it follows that there exists an execution $s'_{11}a_1s_1 \ldots a_ns_n \in \Xi$ such that $a_{n-1} = fail\_step$ and for each $1 \leqslant i \leqslant n$, $s_i.H = s.H$, $s_i.ASId = s.ASId$, and $s_i.F = s.F$. Also s_{12} enables the transition $(s_{12}, send((s_{11}.c)^{-1}(cs), m), \delta_{s'_{12}})$ where $s'_{12} = s_{12}$, since there does not exist θ such that $m = (\sigma \circ \theta)(l_p^j)$. This implies that $s_{12} = s_{11}\lceil_{s_{12}} = s'_{11}\lceil_{s_{12}} = s''_{11}\lceil_{s_{12}} = s''_{11}\lceil_{s'_{12}} = s'_{12}$. This implies, by definition of $\mathcal{R}_k$, that $s'_{11}\ \mathcal{R}_k\ s'_{12}$, thus $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, 0)\ \delta_{s'_{12}}$, hence $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s'_{12}}$ and thus $(s_{12}, send((s_{11}.c)^{-1}(cs), m), \delta_{s'_{12}})$ is a valid weak $l(k)$-bounded combined transition such that $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})$

$\delta_{s'_{12}}$. Finally, suppose that there exists θ such that $m = (\sigma \circ \theta)(l_p^j)$ where $(\sigma, j, p) = (s_{12}.F)((s_{11}.c)^{-1}(cs))$. By Lemma 8.30, it follows that there exists an execution $s'_{11}a_1s_1 \dots a_ns_n \in \Xi$ such that $a_{n-1} = end\_step$, $s_n.H = s.H \cup \{(\sigma \circ \theta)(r_p^j)\}$, and $s_n.F((s'.c)^{-1}(cs)) = (\sigma \circ \theta, j, p+1)$ where $(\sigma, i, p) = (s'.F)((s'.c^{-1})(cs))$. Also s_{12} enables the transition $(s_{12}, send((s_{11}.c)^{-1}(cs), m), \delta_{s'_{12}})$ where s'_{12} is that state that is identified by the same values of s_{12} except for the following values: $s'_{12}.H = s_{12}.H \cup \{(\sigma \circ \theta)(r_p^j)\}$, and $(s'_{12}.F)((s_{11}.c)^{-1}(cs)) = (\sigma \circ \theta, j, p+1)$. This implies that $s'_{12} = s_n{\restriction}_{s'_{12}}$, hence by definition of $\mathcal{R}_k$, that $s'_{11}\ \mathcal{R}_k\ s'_{12}$, thus $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, 0)\ \delta_{s'_{12}}$, hence $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s'_{12}}$ and thus $(s_{12}, send((s_{11}.c)^{-1}(cs), m), \delta_{s'_{12}})$ is a valid weak $l(k)$-bounded combined transition such that $\delta_{s'_{11}}\ \mathcal{L}_w(\mathcal{R}_k, k^{-c})\ \delta_{s'_{12}}$.

Since all cases lead to a contradiction, it follows that the step condition is satisfied, hence $\{\mathrm{Hide}_{\bar{A}_k}(\mathcal{A}_k^{11})\}_{k\in\mathbb{N}} \lessapprox_s \{SAdv\}_{k\in\mathbb{N}}$. □

Completing the Proof

Now we are able to complete the proof showing that there exists a chain of simulations between the concrete model and the symbolic model.

Theorem 8.31. *Let $\mathbb{A}$ be a set of agents, E be an IND-CCA encryption scheme and S be a non-repeating unforgeable signature scheme. For each $k \in \mathbb{N}$, denote by $\mathcal{A}_k$ the automaton $NG_k(\mathbb{A})||\mathcal{E}_k(\mathbb{A})||\mathcal{S}_k(\mathbb{A})||PAdv_k^{adv}(\mathbb{A})$. Let $\bar{A}_k$ be the set $A_k \setminus \{corrupt(\eta_1 \dots \eta_l), new(i, \eta_1 \dots \eta_l), send(cs, m)\}$.*

Then there exists a sequence of automata $\mathcal{B}_k^1, \dots, \mathcal{B}_k^{21}$ such that $\mathcal{B}_k^1 = \mathcal{A}_k$, $\mathcal{B}_k^{21} = SAdv$ and for each $1 \leqslant i \leqslant 20$, it holds that $\{\mathrm{Hide}_{\bar{A}_k}(\mathcal{B}_k^i)\}_{k\in\mathbb{N}} \lessapprox_s \{\mathrm{Hide}_{\bar{A}_k}(\mathcal{B}_k^{i+1})\}_{k\in\mathbb{N}}$.

Proof. The result is the immediate consequence of Lemmas 8.16 to 8.26 and Propositions 5.13, 5.17, and 5.21. □

The above result allows us to invoke the correspondence execution theorem (Theorem 5.20) and thus we have that for each probability measure μ_1 that is reachable within a polynomial number of steps in $\mathcal{A}_k$, there exists a probability measure μ_2 that is reached in $SAdv$ such that μ_1 and μ_2 are related up to a negligible error. In particular, if we consider the relations we used in the chain from $\mathcal{A}_k$ to $SAdv$, we note that they are the identity relation and thus with overwhelming probability for each execution of $\mathcal{A}_k$ (that is stored into the *history* variable), there exists an execution of $SAdv$

that simulates it performing the same sequence of actions and each message is derivable from the previous knowledge, that is all actions of the sequence are Dolev-Yao transitions.

9

Conclusion

In this thesis we have considered the problem of the verification of the security of cryprographic protocols. Starting from the work we find in literature, we have identified two main approaches that are used to prove the security: in the first approach, also called the *formal model* or *symbolic model*, the protocol is modelled combining terms of an algebra where each term models a message. This means that we have terms that denote nonces, other terms that represent agent's identities, and so on. Complex terms, such as pairs of terms, encryptions and signatures are modelled by operations that combine terms and produce a new term. The security of the protocol is proved considering an adversary that interacts with the modelled protocol: the aim of the adversary is to retreive reserved information exchanged by the participants of the protocol or to induce a participant to complete a session of the protocol with the adversary. In this last case, the adversary attacks the protocol if it is able to convince the participant that it has completed the session with another agent and not with the adversary.

The adversary during its interaction with the protocol can not send all messages it desires. Specifically, each message it sends must be deducible from the adversary's knowledge. A message is deducible if it can be derived from the knowledge using a fixed set of deduction rules. These rules characterize the properties of the cryptographic primitives we are considering: for example, an usual rule states that whatever the knowledge is, it is possible to derive the public encryption key of each participant; another rule is the one that permits to deduce the pairing of two messages provided that we are able to derive both the messages. When the protocol involves the encryption as cryptographic operation, there are the following rules that characterize it: for the encryption, the first rule states that if we are able

to derive a message m and a public encryption key pe, then we are able to derive the encryption $\{m\}_{pe}$ of m under the key pe while the second rules states that if we are able to derive the encryption $\{m\}_{pe}$ and the secret decryption key pe^{-1}, then we are able to derive m. These two rules model the widely accepted assumption that everyone can encrypt a message but only the owner of the decryption key can decrypt a ciphertext. Analogously, we have rules that model the generation of signatures and other cryptographic operations.

The proof of security of a cryptographic protocol is very rigorous and often quite simple to understand, but it does not model exactly what happens in the real world. In fact, in the real world the adversary can guess the decryption key and thus it can decrypt a message, even if this happens with very small probability.

The aim of the second approach we have identified in the literature is to model also these cases. In this approach, that is known as the *concrete model* or *computational model*, all elements are sequences of bits, also called bitstrings: nonces, agent's identities, keys; cryptographic operations are functions that receive bitstrings and return a bitstring. The length of the bitstrings is based on a security parameter k: for example, given k, all identities are bitstrings of length k as well as the length of keys. Also the properties of cryptographic primitives are defined considering the real world: for example, we require that the encryption scheme ensures that we can recover the plaintext from an encrypted message only with negligible probability if we do not know the decryption key associated to the encryption key used to encrypt the message.

The security of the protocol is proved considering an adversary that interacts with the modelled protocol: as in the first approach, the aim of the adversary is to retreive reserved information exchanged by the participants of the protocol or to induce a participant to complete a session of the protocol with the adversary. In this last case, the adversary attacks the protocol if it is able to convince the agent that it has completed the session with another participant and not with the adversary. In this approach, the adversary can interact with the protocol sending all messages it desires. The only restriction we impose is that the adversary can perform at most a polynomial number of steps during the generation of the message. The security of the protocol is not absolute but depends on the probability that the adversary performs a successful attack. This means that we do not require that the attack occurs with probability 0 to say that the protocol

is correct, but we say that it is secure if the probability of an attack is negligible with respect to the security parameter k.

This second approach permits to study the security of protocols in the real world but it presents some problems: we must consider the probability of attacks and thus we should take into account all sources of randomness in our analysis; the proofs are usually very long and with a lot of technical details; proofs are usually tedious and hence it is quite common to perform some step of the proof in an informal way. This lack of formality is usually safe but sometimes it can hide some error that invalidates the proof.

A similar situation can be found in the context of distributed systems, where there are several probabilistic components that interact with each other implementing a distributed algorithm. In this context, the analysis of the correctness of a complex system is very rigorous and it is based on tools from information theory such as the *simulation method* that allows us to decompose large problems into smaller problems and to verify systems hierarchically and compositionally. The simulation method consists of establishing relations between the states of two automata, called *simulation relations*, and to verify that such relations satisfy appropriate *step conditions*: each transition of the simulated system can be matched by the simulating system up to the given relation. Using a compositional approach we can study the properties of each small problem independently from the each other, deriving the properties of the overall system. Furthermore, the hierarchical verification allows us to build several intermediate refinements between specification and implementation. Often hierarchical and compositional verification is simpler and cleaner than direct one-step verification, since each refinement may focus on specific homogeneous aspects of the implementation.

In this thesis we have investigates if it is possible to extend tools and results from distributed systems to the verification of cryptographic protocols: we base our modeling on probabilistic automata and we represent each actor (cryptographic primitive, protocol, adversary, and so on) with an automaton. In particular, we model them as in the computational model: exchanged messages are bitstrings which length depends on a security parameter k that parameterize the automaton; the probabilistic aspects are considered directly by the transitions of the involved automata. Once we have the automata that model all actors, their composition models the concrete protocol that interacts with the concrete adversary. The security proof is obtained relating this concrete automaton with another automaton

that implements actors ideally. This means that in this abstract automaton, encryptions satisfy the properties as in the symbolic model, the adversary can generate a message m only if it is able to derive m from its knowledge, and so on.

Standard relations defined on probabilistic automata are not suitable for our purpose, since they do not consider the computational constraints we impose to the adversary. For this reason, we have defined a new simulation relation that takes into account the length of the execution but it is still too restrictive for our aims. In fact, we have that in the symbolic model, the probability to decrypt an encrypted message is zero if we do not know the decryption key; in the computational model, the same event has negligible probability. So we can not use an exact matching, but we need to match up to an error.

This consideration leads us to define the *polynomially accurate simulation* that takes into account the security parameter that characterizes the concrete primitives, the computational aspects of the system and the admitted error. We have defined both strong and weak version of the simulations, since we want to be able to abstract from the internal computations of the automata, provided that such computations are polynomially bounded.

Besides the polynomially accurate simulations, we have provided other tools that can simplify the analysis of cryptographic protocols: the first one is the concept of *conditional automaton*, that permits to safely remove events that occur with negligible probability. Starting from a machine that is attackable with negligible probability, if we build an automaton that is conditional to the absence of these attacks, then there exists a simulation. And this allows us to work with the simulation relations all the time and in particular we can also prove in a compositional way that the elimination of negligible events from an automaton is safe. This property is justified by the *conditional automaton theorem* that states that events are negligible if and only if the identity relation is an approximated simulation from the automaton and its conditional counterpart. Another tool is the *execution correspondence theorem*, that extends the one of the distributed systems context, that allows us to use the hierarchical approach. In fact, the theorem states that if we have several automata and a chain of simulations between them, then with overwhelming probability each execution of the first automaton is related to an execution of the last automaton. In other words, we have that the probability that the last automaton is not able to simulate an execution of the first one is negligible.

Finally, we have provided a library of families of automata that implement commonly used cryptographic primitives and that can be used each time during the verification of a protocol we want to replace a concrete primitive with its ideal counterpart.

To show the usefulness of our polynomially accurate simulation, we have considered two different case studies: in the first one we have recast the security proof of the mutual authentication protocol of Bellare and Rogaway [22] showing how our new simulation can be used to relate the concrete implementation of the protocol with its idealized version, obtaining a proof that involves compositionality and hierarchical verification; in the second case study we used our polynomially accurate simulation to provide a more rigorous proof of the soundness result of Cortier and Warinschi [41] showing off the problems hidden by the not so formal argumentation used in some step of the original proof.

The results of this thesis can be expanded to several directions. One of them is that currently our simulation relates the probability measures that can reached within a polynomial number of steps. It is our intention to relax such restriction allowing the probability measures to be reached in an expected polynomial number of steps and then to study the effects of such modification. Similarly, we want to investigate the properties of the weak simulation defined as the one of Section 5.2 except for the weak transition, where we require that the expected length of the weak transitions is polynomially bounded.

Another direction we would like to extend the results of this thesis is the application of the polynomially accurate simulation to other cryptographic primitives, such as symmetric encryptions and hash functions. The scope of this extension is to add other primitives to the library of results of the Chapter 6.

Moreover, we would like to search for a logical characterization of the proposed simulations and how it can be used together with automatic verification tools or other proposal of literature.

Another research direction can be to study if and how to use our simulation in other research fields, like hybrid systems.

References

1. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages*, pages 104–115, 2001.
2. Martín Abadi and Cédric Fournet. Hiding names: Private authentication in the applied pi calculus. In *Software Security - Theories and Systems*, number 2609 in Lecture Notes in Computer Science, pages 317–338, 2003.
3. Martín Abadi and Andrew G. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, 1999.
4. Martín Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. In *Proceedings of the Fourth International Simposium on Theoretical Aspects of Computer Software (TACS)*, volume 2215 of *Lecture Notes in Computer Science*, pages 82–94, 2001.
5. Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.
6. Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP TCS*, volume 2000 of *Lecture Notes in Computer Science*, pages 3–22, 2000.
7. Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
8. Michael Backes. Unifying simulatability definitions in cryptographic systems under different timing assumptions. Cryptology ePrint Archive, Report 2003/114, 2003.
9. Michael Backes. Unifying simulatability definitions in cryptographic systems under different timing assumptions. In *CONCUR 2003 - Concurrency Theory, 14th International Conference, Proceedings*, volume 2761 of *Lecture Notes in Computer Science*, pages 346–360, 2003.
10. Michael Backes. A cryptographically sound dolev-yao style security proof of the otway-rees protocol. In *Computer Security - ESORICS 2004*, volume 3139 of *Lecture Notes in Computer Science*, pages 89–108, 2004.
11. Michael Backes and Christian II Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *STACS '03: Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2607 of *Lecture Notes in Computer Science*, pages 675–686, 2003.

12. Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable dolev-yao style cryptographic library. In *CSFW '04: Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, pages 204–218, 2004.
13. Michael Backes and Birgit Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on Dependable and Secure Computing*, 2(2):109–123, 2005.
14. Michael Backes and Birgit Pfitzmann. Relating symbolic and cryptographic secrecy. In *IEEE Symposium on Security and Privacy*, pages 171–182, 2005.
15. Michael Backes, Birgit Pfitzmann, Michael Steiner, and Michael Waidner. Polynomial fairness and liveness. In *15th IEEE Computer Security Foundations Workshop*, pages 160–174, 2002.
16. Michael Backes, Birgit Pfitzmann, Michael Steiner, and Michael Waidner. Polynomial liveness. *Journal of Computer Security*, 12(3-4):589–618, 2004.
17. Michael Backes, Birgit Pfitzmann, and Michael Waidner. A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, 2003.
18. Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In *Theory of Cryptography (TCC 2004)*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354, 2004.
19. Michael Backes, Birgit Pfitzmann, and Michael Waidner. Symmetric authentication in a simulatable dolev-yao-style cryptographic library. *International Journal of Information Security*, 4(3):135–154, 2005.
20. Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 394–403, 1997.
21. Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.
22. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO '93: Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, pages 232–249, 1994.
23. Bruno Blanchet. Abstracting cryptographic protocols by Prolog rules. In *Static Analysis Symposium*, volume 2126 of *Lecture Notes in Computer Science*, pages 433–436, 2001.
24. Bruno Blanchet. A computationally sound mechanized prover for security protocols. Cryptology ePrint Archive, Report 2005/401, 2005.
25. Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. Cryptology ePrint Archive, Report 2006/069, 2006.
26. Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 112–117, 1982.
27. Liana Bozga, Yassine Lakhnech, and Michaël Périn. Hermes: An automatic tool for verification of secrecy in security protocols. In *Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 219–222, 2003.
28. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. Technical Report 39, Digital Equipment Corporation Systems Research Center, 1989.
29. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems (TOCS)*, 8(1):18–36, 1990.

30. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000.
31. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *The 42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, 2001.
32. Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Using Probabilistic I/O Automata to analyze an oblivious transfer protocol. Technical Report 2005/452, Cryptology ePrint Archive, 2005.
33. Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Task-structured Probabilistic I/O Automata. In *Proceedings the 8th International Workshop on Discrete Event Systems (WODES'06)*, pages 207–214, 2006.
34. Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Time-bounded task-PIOAs: A framework for analyzing security protocols. In *20th International Symposium on Distributed Computing (DISC 2006)*, volume 4167 of *Lecture Notes in Computer Science*, pages 238–253, 2006.
35. Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Using task-structured Probabilistic I/O Automata to analyze cryptographic protocols. In *Workshop on Formal and Computational Cryptography - FCC 2006*, pages 34–39, 2006.
36. Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40, 2001.
37. Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *Advances in Cryptology - EUROCRYPT 2003*, number 2656 in Lecture Notes in Computer Science, pages 68–86, 2003.
38. Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *Journal of Cryptology*, 19(2):135–167, 2006.
39. David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.
40. Véronique Cortier, Steve Kremer, Ralf Küsters, and Bogdan Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, number 4337 in Lecture Notes in Computer Science, pages 176–187, 2006.
41. Véronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for security protocols. Technical Report RR-5341, INRIA, 2004.
42. Véronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for security protocols. In *Programming Languages and Systems*, volume 3444 of *Lecture Notes in Computer Science*, pages 157–171, 2005.
43. Richard A. DeMillo, Nancy A. Lynch, and Michael Merritt. Cryptographic protocols. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC)*, pages 383–400, 1982.

44. Yvo Desmedt and Kaoru Kurosawa. How to break a practical mix and design a new one. In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 557–572, 2000.
45. Danny Dolev, Shimon Even, and Richard M. Karp. On the security of ping-pong protocols. *Information and Control*, 55(1-3):57–68, 1982.
46. Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.
47. Shimon Even and Oded Goldreich. On the security of multi-party ping-pong protocols. In *Proc. 24th IEEE Symposium Foundations Computer Science. (FOCS)*, pages 34–39, 1983.
48. Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In *Selected Areas in Cryptography: 8th Annual International Workshop*, volume 2259 of *Lecture Notes in Computer Science*, 2001.
49. Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2001.
50. Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.
51. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play a mental game, or completeness theorem for protocols with honest. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 218–229, 1987.
52. Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991.
53. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
54. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
55. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):231–308, 1988.
56. Jean Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *Parallel and Distributed Processing: 15 IPDPS 2000 Workshops*, volume 1800 of *Lecture Notes in Computer Science*, pages 977–984, 2000.
57. James W. III Gray and John McLean. Using temporal logic to specify and verify cryptographic protocols (progress report). In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 108–116, 1995.
58. Joshua D. Guttman, F. Javier Thayer Fábrega, and Lenore D. Zuck. The faithfulness of abstract protocol analysis: Message authentication. In *ACM Conference on Computer and Communications Security*, pages 186–195, 2001.
59. Jonathan C. Herzog. *Computational Soundness for Standard Assumptions of Formal Cryptography*. PhD thesis, Massachusetts Institute of Technology, 2004.
60. Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
61. Dennis Hofheinz and Jörn Müller-Quade. A synchronous model for multi-party computation and the incompleteness of oblivious transfer. Cryptology ePrint Archive, Report 2004/016, 2004.

62. Romain Janvier, Yassine Lakhnech, and Laurent Mazare. Completing the picture: Soundness of formal encryption in the presence of active adversaries. Technical Report TR-2004-19, Verimag, 2005.
63. Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Computational soundness of symbolic analysis for protocols using hash functions. In *Proceedings of the Workshop on Information and Computer Security (ICS'06)*, volume 186 of *Electronic Notes in Theoretical Computer Science*, pages 121–139, 2006.
64. Richard A. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal of Selected Areas in Communications*, 7(4):448–457, 1989.
65. Richard A. Kemmerer, Catherine A. Meadows, and Jonathan K. Millen. Three system for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.
66. Butler W. Lampson. A note on the confinement problem. *Communication of the ACM*, 16(10):613–615, 1973.
67. Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proceeding of the 2004 IEEE Symposium on Security and Privacy*, pages 71–85, 2004.
68. Patrick Lincoln, John C. Mitchell, Mark Mitchell, and Andre Scedrov. A probabilistic poly-time framework for protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 112–121, 1998.
69. Patrick Lincoln, John C. Mitchell, Mark Mitchell, and Andre Scedrov. Probabilistic polynomial-time equivalence and security analysis. In *FM '99: Proceedings of the World Congress on Formal Methods in the Development of Computing Systems-Volume I*, volume 1708 of *Lecture Notes in Computer Science*, pages 776–793, 1999.
70. Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.
71. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS '96: Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 147–166, 1996.
72. Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.
73. Nancy Lynch, Roberto Segala, and Frits Vaandrager. Compositionality for probabilistic automata. In *Concur 2003*, volume 2761 of *Lecture Notes in Computer Science*, pages 208–221, 2003.
74. Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *PODC '87: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, 1987.
75. Paulo Mateus, John C. Mitchell, and Andre Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time calculus. In *Concur 2003*, volume 2761 of *Lecture Notes in Computer Science*, pages 327–349, 2003.
76. Catherine A. Meadows. Using narrowing in the analysis of key management protocols. In *IEEE Symposium on Security and Privacy*, pages 138–147, 1989.
77. Catherine A. Meadows. A system for the specification and verification of key management protocols. In *IEEE Symposium on Security and Privacy*, pages 182–197, 1991.
78. Catherine A. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–36, 1992.
79. Michael John Merritt. *Cryptographic protocols*. PhD thesis, Georgia Institute of Technology, 1983.

80. Daniele Micciancio and Bogdan Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004.
81. Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography Conference – TCC'04*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151, 2004.
82. Jonathan K. Millen. The interrogator: A tool for cryptographic protocol security. In *IEEE Symposium on Security and Privacy*, pages 134–141, 1984.
83. Jonathan K. Millen, Sidney C. Clark, and Sheryl B. Freedman. The interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, 13(2):274–288, 1987.
84. John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 725–733, 1998.
85. John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Mur-ϕ. In *IEEE Symposium on Security and Privacy*, pages 141–151, 1997.
86. John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theoretical Computer Science*, 353(1):118–164, 2006.
87. Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
88. Jesper Buus Nielsen. *On Protocol Security in the Cryptographic Model*. PhD thesis, Department of Computer Science, University of Aarhus, 2003.
89. David J. Otway and Owen Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.
90. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.
91. Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Cryptographic security of reactive systems. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 32, 2000.
92. Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Secure reactive systems. Technical Report Technical Report RZ 3206 (#93252), IBM, Research Division, Zurich, 2000.
93. Birgit Pfitzmann and Michael Waidner. How to break and repair a "provably secure" untraceable payment system. In *Advances in Cryptology - CRYPTO '91: Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 338–350, 1992.
94. Birgit Pfitzmann and Michael Waidner. A general framework for formal notions of secure systems. Technical Report Hildesheimer Informatik-Berichte 11/94, Universitat Hildesheim, 1994.
95. Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *CCS '00: Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000.
96. Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. Cryptology ePrint Archive, Report 2000/066, 2000.

97. Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 184–200, 2001.
98. Michael O. Rabin. Probabilistic algorithms. In *Algorithms and Complexity: New Directions and Results*, pages 21–39, 1976.
99. Michael O. Rabin. N-process mutual exclusion with bounded waiting by $4 \log_2 N$ valued shared variables. *Journal of Computer and Systems Sciences*, 25(1):66–75, 1982.
100. Ronald L. Rivest, Adi Shamir, and Leonard Adleman. On a method for obtain digital signatures and public key cryptosystems. *Communication of the ACM*, 21:120–126, 1978.
101. Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. In *Proceedings, 14th IEEE Computer Security Foundations Workshop*, pages 174–190, 2001.
102. Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1995. Also appears as technical report MIT/LCS/TR-676.
103. Roberto Segala. Verification of randomized distributed algorithms. *Lectures on Formal Methods and Performance Analysis, First EEF/Euro Summer School on Trends in Computer Science*, 2090:232–260, 2001.
104. Roberto Segala. Probability and nondeterminism in operational models of concurrency. In *CONCUR 2006 - Concurrency Theory*, volume 4137 of *Lecture Notes in Computer Science*, pages 64–78, 2006.
105. Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. In *Concur 1994*, volume 836 of *Lecture Notes in Computer Science*, pages 481–496, 1994.
106. Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
107. Victor Shoup. Sequences of games: A tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.
108. Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. Using the Fluhrer, Mantin, and Shamir attack to break WEP. Technical Report TD-4ZCPZZ, AT&T Labs Research, 2001.
109. Paul F. Syverson and Paul C. van Oorschot. On unifying some cryptographic protocols logics. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 14–28, 1994.
110. F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *IEEE Symposium on Security and Privacy*, pages 160–171, 1998.
111. F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.
112. Andrew C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

Sommario

Nella verifica dei protocolli di sicurezza ci seguono due importanti approcci che sono conosciuti sotto il nome di approccio *simbolico* e *computazionale*, rispettivamente. Nell'approccio simbolico i messaggi sono termini di un'algebra e le primitive crittografiche sono idealmente sicure; nell'approccio computazionale i messaggi sono sequenze di bit e le primitive crittografiche sono sicure con elevata probabilità. Questo significa, per esempio, che nell'approccio simbolico solo chi conosce la chiave di decifratura può decifrare un messaggio cifrato, mentre nell'approccio computazionale la probabilità di decifrare un testo cifrato senza conoscere la chiave di decifratura è trascurabile.

Di solito, i protocolli crittografici sono il risultato dell'interazione di molte componenti: alcune sono basate su primitive crittografiche, altre su altri principi. In generale, quello che risulta è un sistema complesso che vorremmo poter analizzare in modo modulare invece che doverlo studiare come un singolo sistema.

Una situazione simile può essere trovata nel contesto dei sistemi distribuiti, dove ci sono molti componenti probabilistici che interagiscono tra loro implementando un algoritmo distribuito. In questo contesto l'analisi della correttezza di un sistema complesso è molto rigorosa ed è basata su strumenti che derivano dalla teoria dell'informazione, strumenti come il *metodo di simulazione* che permette di decomporre grossi problemi in problemi più piccoli e di verificare i sistemi in modo gerarchico e composizionale. Il metodo di simulazione consiste nello stabilire delle relazioni tra gli stati di due automi, chiamate *relazioni di simulazione*, e nel verificare che tali relazioni soddisfano delle *condizioni di passo* appropriate, come che ogni transizione del sistema simulato può essere imitata dal sistema simu-

lante nel rispetto della relazione data. Usando un approccio composizionale possiamo studiare le proprietà di ogni singolo sotto-problema indipendentemente dagli altri per poi derivare le proprietà del sistema complessivo. Inoltre, la verifica gerarchica ci permette di definire molti raffinamenti intermedi tra la specifica e l'implementazione. Spesso la verifica gerarchica e composizionale è più semplice e chiara che l'intera verifica fatta in una volta sola.

In questa tesi introduciamo una nuova relazione di simulazione, che chiamiamo *simulazione polinomialmente accurata* o *simulazione approssimata*, che è composizionale e che permette di usare l'approccio gerarchico nelle nostre analisi. Le simulazioni polinomialmente accurate estendono le relazioni di simulazione definite nel contesto dei sistemi distribuiti sia nel caso forte sia in quello debole tenendo conto delle lunghezze delle esecuzioni e delle proprietà computazionali delle primitive crittografiche.

Oltre alle simulazioni polinomialmente accurate, forniamo altri strumenti che possono semplificare l'analisi dei protocolli crittografici: il primo è il concetto di *automa condizionale* che permette di rimuovere eventi che occorrono con probabilità trascurabile in modo sicuro. Data una macchina che è attaccabile con probabilità trascurabile, se costruiamo un automa che è condizionale all'assenza di questi attacchi, allora esiste una simulazione tra i due. Questo ci permette, tra l'altro, di lavorare con le relazioni di simulazione tutto il tempo e in particolare possiamo anche dimostrare in modo composizionale che l'eliminazione di eventi trascurabili è sicura. Questa proprietà è giustificata dal *teorema dell'automa condizionale* che afferma che gli eventi sono trascurabili se e solo se la relazione identità è una simulazione approssimata dall'automa alla sua controparte condizionale. Un altro strumento è il *teorema della corrispondenza delle esecuzioni*, che estende quello del contesto dei sistemi distribuiti, che giustifica l'approccio gerarchico. Infatti, il teorema afferma che se abbiamo molti automi e una catena di simulazioni tra di essi, allora con elevata probabilità ogni esecuzione del primo automa della catena è in relazione con un'esecuzione dell'ultimo automa della catena. In altre parole, abbiamo che la probabilitèche l'ultimo automa non sia in grado di simulare un'esecuzione del primo è trascurabile.

Infine, usiamo il framework delle simulazioni polinomialmente accurate per fornire delle famiglie di automi che implementano le primitive crittografiche comunemente usate e per dimostrare che l'approccio simbolico è corretto rispetto all'approccio computazionale.