The Reactive Simulatability (RSIM) Framework for Asynchronous Systems*

Michael Backes†, Birgit Pfitzmann, and Michael Waidner‡

Abstract

We define *reactive simulatability* for general asynchronous systems. Roughly, simulatability means that a real system implements an ideal system (specification) in a way that preserves security in a general cryptographic sense. Reactive means that the system can interact with its users multiple times, e.g., in many concurrent protocol runs or a multi-round game. In terms of distributed systems, reactive simulatability is a type of refinement that preserves particularly strong properties, in particular confidentiality. A core feature of reactive simulatability is *composability*, i.e., the real system can be plugged in instead of the ideal system within arbitrary larger systems; this is shown in follow-up papers, and so is the preservation of many classes of individual security properties from the ideal to the real systems.

A large part of this paper defines a suitable system model. It is based on probabilistic IO automata (PIOA) with two main new features: One is *generic distributed scheduling*. Important special cases are realistic adversarial scheduling, procedure-call-type scheduling among colocated system parts, and special schedulers such as for fairness, also in combinations. The other is the definition of the *reactive runtime* via a realization by Turing machines such that notions like polynomial-time are composable. The simple complexity of the transition functions of the automata is not composable.

As specializations of this model we define security-specific concepts, in particular a separation between honest users and adversaries and several trust models.

The benefit of IO automata as the main model, instead of only interactive Turing machines as usual in cryptographic multi-party computation, is that many cryptographic systems can be specified with an ideal system consisting of only one simple, deterministic IO automaton without any cryptographic objects, as many follow-up papers show. This enables the use of classic formal methods and automatic proof tools for proving larger distributed protocols and systems that use these cryptographic systems.

Keywords: security, cryptography, simulatability, formal methods, reactive systems, composability, probabilistic IO automata, distributed polynomial time

1 Introduction

In this paper we present the reactive simulatability (RSIM) framework for general asynchronous systems. More precisely, we present the definition of reactive simulatability and a general reactive asynchronous system model as a basis for this definition and many other general security definitions and theorems.

*A preliminary version of this paper appeared as part of a paper by Pfitzmann and Waidner at IEEE Symposium on Security and Privacy, Oakland, May 2001.

†Saarland University, Saarbrücken, Germany, backes@cs.uni-sb.de. Work done while at IBM Zurich Research Laboratory, Rüschlikon, Switzerland.

‡IBM Zurich Research Laboratory, Rüschlikon, Switzerland, {bpf,wmi}@zurich.ibm.com.

1.1 The Idea of Reactive Simulatability

The basic idea of reactive simulatability, sometimes abbreviated as RSIM, is to define under what conditions one system, typically a real cryptographic system, securely implements another system, typically a much simpler specification called ideal system. Roughly, we define that this is true if everything that can happen to the honest users of the real system, with strong real adversaries, can also happen to the same honest users if they use the ideal system, where adversaries do not occur or at least have far less power. What happens to the users includes the aspect of the adversary's knowledge about the users' behavior and secrets.

Definitions of a real system implementing a specification are well-known in the field of distributed systems and often called refinement; however, normal refinement does not retain confidentiality properties and is therefore not suitable for most security systems, in particular for most cryptographic systems and protocols. For instance, if one defines an ideal secure channel essentially as a black box where messages are put in on one side and come out on the other side, normal notions of correct implementation by a distributed system allow that intermediate parties learn the messages. For security and cryptography, however, this should not happen if the ideal secure channel gives no information to such parties.

In cryptography, a suitable notion of secure implementation, typically called simulatability, was already defined for ideal systems (specifications) that are just functions: each party makes one input at the beginning and obtains one output at the end. Essentially, we extend this notion to reactive systems, i.e., systems where parties may make inputs and obtain outputs at many different times. Examples of reactive cryptographic systems are multi-round auctions, protocols with many concurrent sessions, and untraceable electronic cash systems because whether a payment succeeds depends on prior cash withdrawal actions. For all these systems one requires confidentiality properties, so that the relation between a real system and a specification cannot only be the classical refinement of distributed systems. Reactive simulatability makes the real-and-ideal system specification technique available for such types of systems.

An important property of reactive simulatability is *composability*. Roughly this means that a larger system can be defined based on the specification of a subsystem, in other words with an ideal subsystem, and then the real subsystem can be plugged in instead without causing any significant difference. The notion of "no significant difference" is again reactive simulatability. Composability is generally required of refinement relations in distributed systems. The idea of reactive simulatability has also become known as universal composability (UC) for such properties. (We describe the history of these terms and theorems in Section 1.7.) Reactive simulatability also offers *property preservation*, i.e., if one proves certain important properties of an ideal system, then they also hold for the real system. An example of such a property is that the participants in a payment system cannot spend more money than they put in initially or received. Such properties are not always trivial to show for an ideal system, but usually very much easier than if one had to do it directly for the real system. However, for length reasons of this first journal version with detailed definitions, we do not prove any composition and property preservation theorems here although the first ones were in the corresponding conference publication.

1.2 Link to Formal Methods and Tool-Supported Proofs

As just explained, reactive simulatability with its notion of ideal systems is an important new way of specifying reactive cryptographic systems and protocols, besides the classical way of defining many individual properties, where each property immediately contains details about

adversaries, polynomial-time considerations, and error probabilities.

Besides this general motivation, a specific motivation for defining reactive simulatability was that it offers an important link between cryptography and formal methods, in particular automated proof tools such as model checkers and theorem provers. Interest in such a link can be justified from both prior cryptographic protocol proofs and from prior tool-supported proofs of security protocols: The cryptographic motivation is essentially the limit of human stamina when dealing with the many different sequences of actions occurring in executions of even relatively small protocols. The tool-supported proof motivation is the prior lack of demonstrated soundness with respect to real cryptography. We now discuss this in more detail.

Typical cryptographic proofs are reductions between the security of an overall system under consideration and the security of the cryptographic primitives used: One shows that if one could break the overall system, one could also break one of the primitives with respect to its cryptographic definition, e.g., adaptive chosen-message security for signature schemes. In principle, these proofs are as rigorous as typical proofs in mathematics. In practice, however, human beings are extremely fallible with such proofs when protocols are concerned. This is mainly due to the distributed-systems aspects of the protocols. It is well-known from non-cryptographic distributed systems that many wrong protocols have been published even for very small problems. Hand-made proofs are highly error-prone because following all the different orders of interleaving of the actions of different participants is extremely tedious. Humans tend to take wrong shortcuts and do not want to proof-read such details in proofs by others. If the protocol contains cryptography, the situation is even worse: Already a rigorous definition of the goals and of the protocol itself gets more complicated, and there was previously no general framework for this. Compared with protocol proofs outside security, which are mostly for trace properties, i.e., properties of individual runs, confidentiality properties are more complex because they are properties of entire probability spaces of runs. Moreover, in principle the complexity-theoretic reduction has to be carried out across all the different interleavings, and it is not at all trivial to do this rigorously. In consequence, there are very few real cryptographic proofs of larger protocols, and several times supposedly proven, relatively small systems were later broken. Hence tool support should be very welcome, at least for the tedious distributed-system aspects.

In fact, work on tool-supported proofs of cryptographic protocols started as early as work on computational cryptographic definitions and proofs. Tools mean model checkers and automatic theorem provers; initially mostly special-purpose for security protocols, nowadays mostly specializations of more general tools. However, for a very long time all these proofs were based on idealized abstractions of cryptographic primitives, almost always by representing cryptographic operations as operators of a term algebra with cancellation rules, so-called Dolev-Yao models. For instance, public-key encryption is represented by operators E for encryption and D for decryption with one cancellation rule, $\mathsf{D}(\mathsf{E}(m)) = m$ for all m. Encrypting a message m twice in a Dolev-Yao model does not yield another message from the basic message space but the term $\mathsf{E}(\mathsf{E}(m))$. The models assume that two terms whose equality cannot be derived with the cancellation rules are not equal, and every term that cannot be derived is completely secret. This simplifies proofs of larger protocols considerably. However, originally there was no foundation at all for such idealizations of cryptographic primitives, and thus no guarantee that protocols proved with these tools are secure when implemented with real cryptography. Although no previously proved protocol has been broken when implemented with standard provably secure cryptosystems (if one excludes proofs in formal models that have more semantic problems than the cryptographic ones, in particular those based on logics of belief because of the typically unjustified monotonicity of belief, and only regards the properties that were proved), this was

clearly an unsatisfactory situation, and artificial counterexamples can be constructed.

The main use of reactive simulatability in the context of linking cryptography and formal methods in a sound way is that it can be the gauge for deciding whether an idealization is securely realized by a specific cryptographic implementation, or even realizable by any such implementation. One specific goal, achieved later, was to apply this gauge to Dolev-Yao models. However, the approach is not at all limited to Dolev-Yao models – many possible ideal systems used as specifications for cryptographic systems are quite simple and can thus be encoded into existing proof tools, so that those proof tools can be used when larger systems are proved that use these cryptographic systems.

1.3 Requirements on the System Model

The first obstacle to defining reactive simulatability was that there was no system model, i.e., a model of protocol participants and how they interact, that combined all the features we desire:

- Allowing reactive systems.
- Enabling system definitions that are not encumbered by Turing machine details, in particular for ideal systems and with the aim of encodings into some current proof tools.
- Allowing run-time considerations, because most cryptographic systems are only secure against computationally bounded adversaries. We also desire composability of runtimes; in particular the combination of polynomial-time entities should be polynomial again. This is particularly important in compositions, where the protocol machines of higher layers become the users of the lower-layer systems. Users also have to be polynomially bounded so that an adversary cannot offload computations to them in active attacks.
- Asynchronous systems with a sufficiently flexible scheduling model that all typical cases can be represented. A scheduling model defines how it is determined in which order different actions, in particular of different distributed entities, occur. In particular, we want to allow:
  - scheduling by the adversary with realistic information;
  - procedure-call-style scheduling for colocated components; this occurs in particular in compositions, when a higher-layer entity uses a lower-layer entity in the same location; and
  - restricted scheduling, e.g., fair schedulers for liveness-style considerations (i.e., requirements that something should happen, possibly within restricted time), or for encoding synchronous systems.
- Independence of trust models. This means that the core system model should not fix issues like what computational power an adversary has, how it can corrupt participants, and how it can manipulate messages on channels. There are many variants of this, and we want to be able to express them all.

We address the last requirement by first defining a core system model without any security considerations, i.e., an extension of existing distributed-system models by more generic scheduling and by runtime considerations. Then we specialize this with a small number of general security concepts such as the notion of an adversary and an honest user. Finally, we define some specific

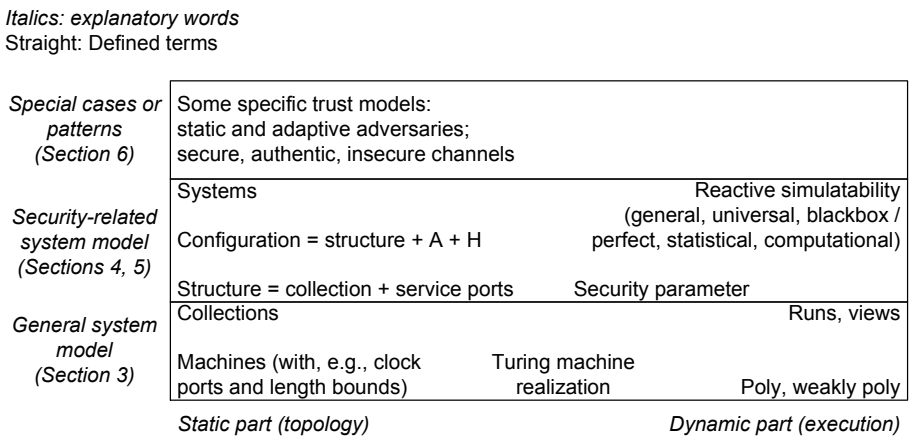

Figure 1: Overview of our layers of definitions

examples of trust models, mainly for real systems, in particular for static and dynamic adversaries and for channels with different types of security. These layers and the main definitions we make on each layer are surveyed in Figure 1. The first four requirement above all refer to the lowest layer. Some details of these requirements may become clearer when we explain how we address them.

1.4 Asynchronous System Model with General Distributed Scheduling and Runtimes

We use an IO automata model as our core model, in other words state-transitions systems that specify the next output and state for every given input and state. Given the importance of probabilism in cryptography we need probabilistic IO automata (PIOAs). We did not choose interactive Turing machines as the core model, although this was normal at that time in cryptography, because of our requirement that ideal systems, if they are simple in principle, should be easy to encode into existing proof tools from our concrete specification. IO automata are the typical basis for encoding distributed systems into the specification languages of standard theorem provers. Furthermore, most formal languages (i.e., with fully fixed syntax) for distributed systems have some state-transition system as their semantics. Informally also cryptography has often used IO automata, because nobody actually specifies cryptographic protocols as Turing machines; the specifications in articles are much closer to IO automata as soon as they go beyond simple arrow pictures.

One novelty is that we allow essentially arbitrary scheduling schemes. The basis of scheduling is message delivery: In an asynchronous system, a message does not arrive immediately at its recipient, but may be held up in the network. In our *generic distributed scheduling*, one can designate for each connection which party decides about the arrival of the messages on this connection; we call this party the *scheduler* of this connection. Thus in particular we can model the special cases required above:

- We can let the adversary schedule everything by making it the scheduler of all connections.
- For procedure-call style interactions between a colocated caller and a service, we let the caller schedule the connection to the service, and let the service schedule the reverse connection. Each of them immediately schedules each of its messages on these connections.

- Or a separate scheduler can schedule everything, or only certain connections (e.g., secure ones) while the adversary schedules others.

While the generic distributed scheduling allows many more variants, these three (in particular combinations of the first and the second) are mostly used in subsequent work.

For computational complexity, the easiest option would be to consider the complexity of the state-transition function of the IO automata. The complexity of functions is well-defined and we would never need to mention the underlying bit-level model like Turing machines, in particular if we concentrate on notions like polynomial-time that are robust against small model variations. However, *polynomial-time transitions* do not even lead to overall polynomial runtime when such a machine runs essentially alone: For instance, such a machine might double the size of its current state in each transition; thus it can use time exponential in its initial state size after a linear number of transitions with one-bit inputs. In a medium notion, which we call *weakly polynomial-time*, the runtime of the machine is polynomial in the overall length of its inputs and its initial state. A weakly polynomial-time machine is a permissible adversary when interacting with a cryptographic system which is in itself polynomially bounded; i.e., this seems to be the weakest useful definition. However, this notion does not compose: Several weakly polynomial-time machines together can become too powerful. E.g., each new output may be twice as long as the inputs so far. Then with a linear number of interactions, these machines can use time exponential in the size of their initial states.

Hence we define *polynomial-time* machines as those that only need time polynomial in their initial state size, independent of all inputs. This notion is composable.

We nevertheless use weakly polynomial-time machines sometimes, because many functionalities are naturally weakly polynomial-time and not naturally polynomial-time.

We made one further addition to individual machines compared with other I/O automata models, in order to enable machines to have polynomial runtime independent of their environment without being automatically vulnerable to denial-of-service attacks by long messages: We allow state-dependent *length bounds* on the inputs that a machine will read from each channel.

1.5 Security-Related System Model

Reactive simulatability is about systems, users, and adversaries: If the same honest users use either the real or the ideal system, they shouldn't notice a difference, even though the real system typically gives an adversary much more power than the ideal system. Hence we have to define users and adversaries. The general system model sketched in Section 1.4 simply allows "open" systems that can interact with some environment. The user-adversary distinction is essential but quite simple: We split the interaction opportunities (later called ports; also think of them as unattached connections) into some for the honest users and others for the adversary. We call the former *service ports*. We call such an open system with a split into service ports and others a *structure*, and if we augment it by an honest user and an adversary in the designated way, we call it a *configuration*.

We also define systems as sets of structures; this is the general provision for trust models which allow that instead of one structure intended by a designer (e.g., with n machines for n users, connected by point-to-point channels), one of many different semi-corrupted structures is actually present (e.g., with fewer machines because some were taken over by the adversary, and with wiretaps on the channels).

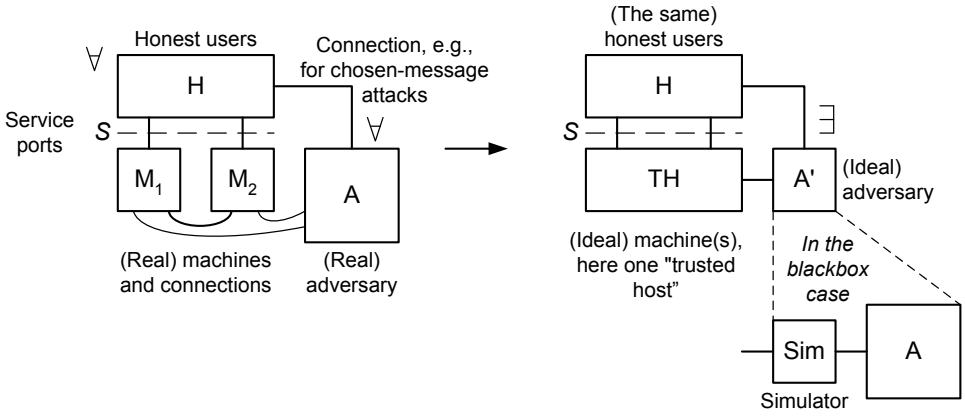

Figure 2: Overview of reactive simulatability

1.6 Reactive Simulatability Variants

We have already introduced reactive simulatability, the main goal of our definitions, in Section 1.1. Figure 2 illustrates it, including typical identifiers that we use for various parts. The left half illustrates a real system. Here it consists of a structure with only two machines (PIOAs) M_1 and M_2. An entirety of honest users H uses it via the service ports, and an adversary A interacts both with the two "normal" machines and the honest users. This is compared with the ideal system on the right side. In this example, the structure in the ideal system consists of just one machine, which we often call TH for "trusted host" (corresponding to the intuition that a trusted host would simply do for the participants what in reality they have to do via a complex cryptographic protocol). Formally there is no difference between ideal and real systems in our model; this is useful in compositions and other multi-part security proofs. The same honest users use the ideal structure via the same service ports, and there may again be an adversary A'.

We define reactive simulatability in several variants: In one dimension, we vary the order of quantifiers in the statement that for all honest users H and all adversaries A on the real system, there should be an adversary A' on the ideal system that achieves the same effects. What we just wrote is *general reactive simulatability (GRSIM)*. If the ideal adversary does not depend on the honest users (only on the real adversary and of course the system), we speak of *universal reactive simulatability (URSIM)*, i.e., then the quantifier order is $\forall\mathsf{A}\exists\mathsf{A}'\forall\mathsf{H}$. If the ideal adversary consists of a fixed part that uses the real adversary as a blackbox, we speak of *blackbox reactive simulatability (BRSIM)* and call the fixed part *simulator*. In another dimension, we have a perfect, a statistical, and a computational variant, depending on computational restrictions and the degree of similarity we require between the real and the ideal system.

1.7 Prior Work

Simulatability, i.e., the notion of using a simple ideal system as a specification for a cryptographic system, was first sketched for secure multi-party function evaluation, i.e., for the computation of one output tuple from one tuple of secret inputs from each participant, in [99]. It was defined (with different degrees of generality and rigorosity) in [58, 30, 81, 38]. Among these, [30] and an unpublished longer version of [81] contain the earliest detailed execution models for cryptographic systems that we are aware of. Both are synchronous models. Problems such as the separation of users and adversaries, or defining runtime restrictions in the

face of continuous external inputs, do not occur in the function-evaluation case, since function evaluation is non-reactive. A composition theorem for non-reactive simulatability was proven in [38].

The idea of simulatability was subsequently also used for specific reactive problems, e.g., [55, 32, 44], without a detailed or general definition. In a similar way it was used for the construction of generic solutions for large classes of reactive problems [57, 56, 61] (usually yielding inefficient solutions and assuming that all parties take part in all subprotocols). A reactive simulatability definition was first proposed (after some earlier sketches, in particular in [57, 90, 38]) in [61]. It is synchronous, covers a restricted class of protocols (straightline programs with restricted operators, in view of the constructive result of this paper), and for the information-theoretic case only, where quantification over input sequences can be used instead of active honest users.

We first presented a synchronous version of a general reactive system model and reactive simulatability in [91]. The report version also contains further variants of the reactive simulatability definitions with proofs of equivalence or non-equivalence that are likely to carry over to the asynchronous case.

After that, and later but independently to the conference version [92] of the current paper, an asynchronous version of a general reactive model and reactive simulatability was also given in [39]. The model parts that were relatively well-defined seem to us a strict subset of our model: The system correspond to our "cryptographic systems with adaptive adversaries" in Section 6.3, always with polynomial-time users and adversaries. The entities are defined as Turing machines only, i.e., there is no explicit abstraction layer like our IO automata. The simulatability definition corresponds to the universal case of ours. Besides the model, the paper contains a composition theorem which was more general than ours at that time, while we had a property preservation theorem. Here the term UC (universal composability) was coined which is nowadays also widely used for the general idea of reactive simulatability or the definitions. (The paper also contains some sketches, while we had decided to only publish parts that we had actually defined and proved.)

The first rigorous model for reactive systems that covers cryptography, i.e., probabilistic and polynomial-time aspects, was presented in [73, 74]. It is based on π-calculus and uses formal language characterizations of polynomial time. The notion of security is observational equivalence. This is even stronger than reactive simulatability because the entire environment (corresponding to our users and adversary together) must not be able to distinguish the implementation and the specification. However, this excludes many abstractions, e.g., because a typical abstract specification is one machine and the real system is distributed with channel manipulation possibilities for the adversary, so that an adversary can already distinguish them by their structure. Correspondingly, the concrete specifications used essentially comprise the actual protocols including all cryptographic details. There was no tool support for the proofs at that time, as even the concrete specifications involved ad-hoc notations, e.g., for generating random primes.

Reductions as cryptographic proofs were introduced for cryptographic primitives in [35, 59, 100]. The best-known application to protocols is the handling of authentication protocols originating in [33]. Examples of later breaks of supposedly proven cryptographic systems are given in [89, 52, 63].

The Dolev-Yao models underlying most proof tools for cryptographic protocols were introduced in [53]. Some important examples of their first use in different types of proof tools are [83, 80, 69, 77, 88, 97, 1]. Soundness of these models with respect to real cryptography was first considered in [2], but only under passive attacks. This corresponds well to the fact

that there wasn't even a definition for such a comparison under active attacks, and only reactive simulatability provided this later. Logics of belief for cryptographic protocol proofs were introduced in [36]. A semantics for such a model (in the sense of an execution semantics, still relying on a Dolev-Yao model for the cryptography) was first given in [3]. Careful study of this semantics shows that one needs hand-proofs of strong protocol properties before the logic applies; this is never done in practice. This is why we did not count breaks of protocols proven in such logics as counter-arguments against the Dolev-Yao models or proof tools relying on normal distributed-systems semantics.

IO automata were already used for security in [79]. There however, cryptographic systems are restricted to the usual equational specifications following a Dolev-Yao model [53], and the semantics is not probabilistic. Only passive adversaries are considered and only one class of users, called environment. The author actually remarks that the model of what the adversary learns from the environment is not yet general, and that general theorems for the abstraction from probabilism would be useful. Our model solves these problems.

So far we looked at prior security definitions. We now consider literature on system models as such. Our IO automata are based on normal finite-state machines; deterministic and non-deterministic versions, also with infinite state, have been used widely throughout the distributed protocol literature. Probabilistic IO-automata and execution models for them are defined in [94, 78, 98]. There the order of events is chosen by a probabilistic scheduler that has full information about the system. However, this can give the scheduler too much power in a cryptographic scenario. In cryptology, the typical understanding of asynchronous systems, closest to a rigorous definition in [37], is that the adversary schedules everything, but only with realistic information, in terms of both observations and computational capabilities. Recall that this is still an important special case in our model. We are sometimes asked why we do not just elaborate this case. However, there are situations where one definitely needs more benign scheduling or even some synchrony, as in our third special case (e.g., one may want to show liveness properties, but can only do so if one can model that messages are eventually delivered). As to local scheduling among system parts modeled as different automata but considered co-located, adversarial scheduling may often do no harm. However, having to prove it when one actually *is* considering a local situation, only defined as a composition, would just introduce unnecessary complications into the proof. Modeling only adversarial scheduling would remove the need to represent who schedules what, while the underlying model of asynchronous connections would remain the same. Problems with purely adversarial scheduling were already noted in [74]; hence they schedule secure channels with uniform probability before adversary-chosen events. However, that introduces a certain amount of global synchrony. Furthermore, we do not require specific scheduling for all secure channels; they may be blindly scheduled by the adversary (i.e., without even seeing whether there are messages on the channel). For instance, this models the case where the adversary has a global influence on the relative network speed.

There are also probabilistic versions of other detailed distributed-systems frameworks than IO automata, but apart from [73, 74] we are not aware of a prior one with polynomial-time considerations or any specific scheduling considerations for security.

We are also not aware of any prior model with a representation of both honest users and adversaries.

1.8 Subsequent Work

As mentioned before, it has been shown that reactive simulatability has at least the properties expected of a refinement notion in distributed systems: First, it is indeed transitive [92].

Secondly, it has a composition theorem that states that substituting a refined system (an implementation) for the original system (a specification) within a larger system is permitted [92]; compositionality of reactive simulatability in different settings has been investigated in [39, 75, 24, 65, 51, 67, 70]. Thirdly, one can define computational versions of various security property classes and prove preservation theorems for them under reactive simulatability, in particular for integrity [91, 11], key and message secrecy [19], transitive and non-transitive non-interference [16, 15], i.e., absence of information flow, and classes of liveness properties [22, 10].

Various concrete cryptographic systems have been proven secure in the sense of reactive simulatability with respect to ideal specifications that are not encumbered with cryptographic details. This comprises secure message transmission [92, 46], key exchange [46], and group key agreement [96]. Under additional assumptions such as the existence of a common random reference string, this was extended to commitment schemes [43], oblivious transfer [48, 54], zero-knowledge proofs [43], and, more generally, any multi-party function evaluation [48]. Reactive simulatability also proved useful for lower-layer proofs, e.g., of reactive encryption and signature security from traditional (non-reactive) encryption and signature security within [92, 40] and [41, 25, 26], respectively, and of reactive Diffie-Hellman security within [96].

A particularly important ideal specification that has been proven to have a cryptographic realization secure in the sense of reactive simulatability is a specific Dolev-Yao model [23, 27, 17, 21], nowadays referred to as the BPW model. The BPW model offers a comprehensive set of Dolev-Yao-style operations for modeling and analyzing security protocols: both symmetric and asymmetric encryption, digital signatures, message authentication codes, as well as nonces, payload data and a list (pairing) operation. Proofs of the Needham-Schroeder-Lowe protocol [14], the Otway-Rees protocol [5], the Yahalom protocol [20], an electronic payment protocol [7], and parts of public-key Kerberos [6] using the BPW model show that one can rigorously prove protocols based on this model in much the same way as with more traditional Dolev-Yao models. Weaker soundness notions for Dolev-Yao models such as integrity only or offline mappings between runs of the two systems, and/or allowing less general protocol classes, e.g., only a specific class of key exchange protocols, have been established in [82, 72, 45]. For these cases, simpler Dolev-Yao models and/or realizations can be used compared to [23].

Establishing reactive simulatability for concrete systems has proven an error-prone task if approached rather informally, as it requires to carefully compare various behaviors of an ideal specification with corresponding behaviors of the concrete system. We emphasize that it is quite easy to guess almost correct ideal specifications of cryptography systems; the major part of the work lies in getting the details right and making a rigorous proof. Several abstractions presented with less detailed proofs have been broken [63, 9] (where the first paper attributes another such attack to Damgård). In order to establish reactive simulatability in a rigorous manner, the notion of a cryptographic bisimulation has been introduced in [23] and further extended in a sightly different framework in [42]. Cryptographic bisimulations are based on the notion of probabilistic bisimulations [71], which capture that two systems that are in related states and receive the same input will yield identically distributed states and outputs after they performed their next transition. Cryptographic bisimulations extend this notion with imperfections and, in the case of the BPW model, an embedded static information-flow analysis.

As far as automation of security proofs in the context of reactive simulatability is concerned, it was first shown that automated proof tools can handle small examples of reactive simulatability proofs, based on the ideal specification of secure message transmission [12, 11]. Later it was shown that the BPW model is accessible to theorem proving techniques using the theorem prover Isabelle/HOL [95]. This constitutes the first tool-supported framework for symbolically verifying security protocols that enjoys the strong cryptographic soundness guarantees provided

by reactive simulatability. Fully automated techniques for proving secrecy properties of security protocols based on the BPW model have been invented in [13] using mechanized flow analysis. In the wider field of linking formal methods and cryptography, there is also work on formulating syntactic calculi for dealing with probabilism and polynomial-time considerations directly, in particular [84, 85, 68, 50, 34]. This is orthogonal to the work of justifying Dolev-Yao models: In situations where Dolev-Yao models are applicable and sound, they are likely to remain important because of the strong simplification they offer to the tools, which enables the tools to treat larger overall systems automatically than with the more detailed models of cryptography.

Finally, the expressiveness of reactive simulatability, i.e., the question which cryptographic tasks can be assigned a suitable ideal functionality under which assumptions, has been investigated in a series of papers. It has been shown that several cryptographic tasks cannot be proven secure in the sense of reactive simulatability unless one makes additional set-up assumptions, e.g., by postulating the existence of a common random reference string. Among these tasks are bit commitment, zero-knowledge proofs, oblivious transfer [43], (authenticated) Byzantine agreement [76], classes of secure multi-party computation protocols [47], classes of functionalities that fulfill certain game-based definitions [49], and Dolev-Yao style abstractions of XOR and hash functions [18, 28]. Other set-up assumptions to circumvent such impossibility results are to augment real and ideal adversaries with oracles that selectively solve certain classes of hard problems [93], to free the ideal adversary from some of its computational restrictions [29], or to impose constraints on the permitted honest users [8]. The price is either a narrower model tailored to a specific problem, sacrificing the transitivity of reactive simulatability and hence significantly complicating the modular construction of larger protocols, or providing only a limited form of compositionality.

1.9 Overview of this Paper

Section 2 introduces notation. Section 3 defines the general system model, i.e., machines, both their abstract version and their computational realization, and executions of collections of machines. Section 4 defines the security-specific system model, i.e., systems with users and adversaries. Section 5 defines reactive simulatability, i.e., our notion of secure refinement. Section 6 shows how to represent typical trust models, i.e., assumptions about the adversary, such as static threshold models and adaptive adversaries, with secure, authenticated and insecure channels. Section 7 concludes the paper.

2 Notation

Let $\mathit{Bool} := \{\mathsf{true}, \mathsf{false}\}$, and let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. For an arbitrary set A, let $\mathcal{P}(A)$ denote its powerset. Furthermore, let A^n denote the set of sequences over A with index set $\mathcal{I} = \{1, \ldots, n\}$ for $n \in \mathbb{N}_0$, $A^* := \bigcup_{n \in \mathbb{N}_0} A^n$, and A^∞ the set of sequences over A with index set $\mathcal{I} = \mathbb{N}$. We write a sequence over A with index set $\mathcal{I}$ as $S = (S_i)_{i \in \mathcal{I}}$, where $\forall i \in \mathcal{I}\colon S_i \in A$. Let $\circ$ denote sequence concatenation, and $()$ the empty sequence. Let $\{()\}$ denote the empty Cartesian product. For a sequence $S \in A^* \cup A^\infty$ we define the following notation:

- For a function $f\colon A \to A'$, let $f(S)$ apply f to each element of S, retaining the order.
- Let $\mathsf{size}(S)$ denote the length of S, i.e., $\mathsf{size}(S) := |\mathcal{I}|$ if $\mathcal{I}$ is finite, and $\mathsf{size}(S) := \infty$ otherwise.

- For $l \in \mathbb{N}_0$, let $S\lceil_l$ (read "S restricted to l elements") denote the l-element prefix of S, and $S[l]$ the l-th element of S with the convention that $S[l] = \epsilon$ if $\mathsf{size}(S) < l$ (for a fixed symbol ϵ). We sometimes write S_l instead of $S[l]$ to increase readability.
- For a predicate $pred\colon A \to Bool$, let $(S[i] \in S \mid pred(S[i]))$ denote the subsequence of S containing those elements $S[i]$ of S with $pred(S[i]) = \mathsf{true}$, retaining the order.

We lift the restriction notation to finite sequences of sequences: For $T = (T_1, \ldots, T_n) \in (A^* \cup A^\infty)^*$ and $L = (L_1, \ldots, L_n) \in \mathbb{N}_0^*$ with the same $n \in \mathbb{N}_0$, let $T\lceil_L := (T_1\lceil_{L_1}, \ldots, T_n\lceil_{L_n})$.

In the following, we assume that a finite alphabet $\Sigma \supseteq \{0, 1, \mathsf{c}, \mathsf{l}, \mathsf{k}\}$ is given, where $\sim, !, ?, \leftrightarrow, \triangleleft \notin \Sigma$. Then Σ^* denotes the strings over Σ. Let ϵ be the empty string and $\Sigma^+ := \Sigma^* \setminus \{\epsilon\}$. All notation for sequences can be used for strings, such as $\circ$ for string concatenation, but $\circ$ is often omitted.

For representing natural numbers and sequences of strings as strings, we assume a surjective function $\mathsf{nat}\colon \Sigma^* \to \mathbb{N}$ with the convention $\mathsf{nat}(1) = 1$, and a bijective function $\iota\colon (\Sigma^*)^* \to \Sigma^*$. We assume that standard operations are efficiently (polynomial-time) computable in these encodings; concretely we need this for inverting the function ι, for appending an element to a sequence of strings, and for retrieving and removing the $\mathsf{nat}(u)$-th element from a sequence of strings.

For an arbitrary set A let $\mathsf{Prob}(A)$ denote the set of all finite probability distributions over A, i.e., those probability distributions D that are actually defined on a finite subset A' of A, augmented by $D(A \setminus A') = 0$. For a probability distribution D over A, the probability of a predicate $pred\colon A \to Bool$ is written $\mathsf{Pr}_D(pred)$. If x is a random variable over A with distribution D, we also write $\mathsf{Pr}_D(pred(x))$. In both cases we omit D if it is clear from the context.

We write $:=$ for deterministic and $\leftarrow$ for probabilistic assignment. The latter means that for a function $f\colon X \to \mathsf{Prob}(Y)$, we write $y \leftarrow f(x)$ to denote that y is chosen according to the distribution $f(x)$. For such a function f we write $y := f(x)$ if there exists $y' \in Y$ with $\mathsf{Pr}_{f(x)}(y') = 1$. If the function f is clear from the context, we also write $x \to_p y$ for $\mathsf{Pr}_{f(x)}(y) = p$, and $\to$ for $\to_1$. Furthermore, we sometimes treat $f(x)$ as a random variable instead of a distribution, e.g., by writing $\mathsf{Pr}(f(x) = y)$ for $\mathsf{Pr}_{f(x)}(y)$.

3 Asynchronous Reactive Systems

In this section, we define our model of interacting probabilistic machines with distributed scheduling and with computational realizations.

3.1 Ports

Machines can exchange messages with each other via *ports*. Intuitively, a port is a possible attachment point for a channel when a machine is considered in isolation. As in many other models, channels in collections of machines are specified implicitly by naming conventions on the ports; hence we define port names carefully. Figure 3 gives an overview of the naming scheme.

Definition 3.1 *(Ports) Let* $\mathcal{P} := \Sigma^+ \times \{\epsilon, \leftrightarrow, \triangleleft\} \times \{!, ?\}$. *Then* $p \in \mathcal{P}$ *is called a* port. *For* $p = (\mathsf{n}, l, d) \in \mathcal{P}$, *we call* $\mathsf{name}(p) := \mathsf{n}$ *its* name, $\mathsf{label}(p) := l$ *its* label, *and* $\mathsf{dir}(p) := d$ *its* direction. $\diamond$

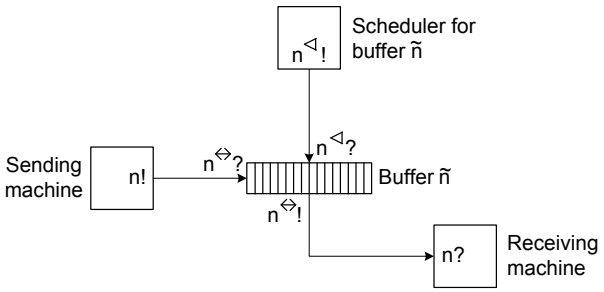

Figure 3: Ports and buffers.

In the following we usually write (n, l, d) as $\mathsf{n}ld$, i.e., as string concatenation. This is possible without ambiguity, since the mapping $\varphi\colon \mathcal{P} \to \Sigma^+ \circ \{\epsilon, {}^{\leftrightarrow}, {}^{\triangleleft}\} \circ \{!, ?\}$ with $\varphi((\mathsf{n}, l, d)) := \mathsf{n}ld$ is bijective because of the precondition $!, ?, {}^{\leftrightarrow}, {}^{\triangleleft} \notin \Sigma$.

The name of a port serves as an identifier and will later be used to define which ports are connected to each other. The direction of a port determines whether it is a port where inputs occur or where outputs are made. Inspired by the CSP [62] notation, this is represented by the symbols ? and !, respectively. The label becomes clear in Definition 3.3.

Definition 3.2 *(In-Ports, Out-Ports) A port* (n, l, d) *is called an* in-port *or* out-port *iff* $d = ?$ *or* $d = !$*, respectively. For a set* P *of ports let* $\mathsf{out}(P) := \{p \in P \mid \mathsf{dir}(p) = !\}$ *and* $\mathsf{in}(P) := \{p \in P \mid \mathsf{dir}(p) = ?\}$*. For a sequence* P *of ports let* $\mathsf{out}(P) := (p \in P \mid \mathsf{dir}(p) = !)$ *and* $\mathsf{in}(P) := (p \in P \mid \mathsf{dir}(p) = ?)$*.* ◇

The label of a port determines the port's role in the upcoming scheduling model. Roughly, ports p with $\mathsf{label}(p) \in \{{}^{\leftrightarrow}, {}^{\triangleleft}\}$ are used for scheduling whereas ports p with $\mathsf{label}(p) = \epsilon$ are used for "usual" message transmission.

Definition 3.3 *A port* $p = (\mathsf{n}, l, d)$ *is called a* simple port*,* buffer port *or* clock port *iff* $l = \epsilon$*,* ${}^{\leftrightarrow}$*, or* ${}^{\triangleleft}$*, respectively.* ◇

After introducing ports on their own, we now define the *low-level complement* of a port. Later each port and its low-level complement will be regarded as directly connected. Two connected ports have identical names and different directions. The relationship of their labels l and l' is visible in Figure 3, i.e., $l = l' = {}^{\triangleleft}$ or $\{l, l'\} = \{\epsilon, {}^{\leftrightarrow}\}$. The remaining notation of Figure 3 is explained below. In particular, "Buffer $\widetilde{\mathsf{n}}$" represents the network between the two simple ports $\mathsf{n}!$ and $\mathsf{n}?$. If we are not interested in the network details then we regard the ports $\mathsf{n}!$ and $\mathsf{n}?$ as connected; thus we call them *high-level complements* of each other.

Definition 3.4 *(Complement Operators) Let* $p = (\mathsf{n}, l, d)$ *be a port.*

a) The low-level complement p^c *of* p *is defined as* $p^c := (\mathsf{n}, l', d')$ *such that* $\{d, d'\} = \{!, ?\}$*, and* $l = l' = {}^{\triangleleft}$ *or* $\{l, l'\} = \{\epsilon, {}^{\leftrightarrow}\}$*.*

b) If p *is simple, the* high-level complement p^C *of* p *is defined as* $p^C := (\mathsf{n}, l, d')$ *with* $\{d, d'\} = \{!, ?\}$*.*

◇

3.2 Machines

After introducing ports, we now define *machines*. Our primary machine model is probabilistic state-transition machines, similar to probabilistic I/O automata as in [94, 78]. A machine has a *sequence of ports*, containing both in-ports and out-ports, and a set of *states*, comprising sets of *initial* and *final states*. When a machine is switched, it receives an input tuple at its input ports and performs its *transition function* yielding a new state and an output tuple in the deterministic case, or a finite distribution over the set of states and possible outputs in the probabilistic case. Furthermore, each machine has state-dependent bounds on the length of the inputs accepted at each port to enable flexible enforcement of runtime bounds, as motivated in Section 1. The part of each input that is beyond the corresponding length bound is ignored. The value ∞ denotes that arbitrarily long inputs are accepted.

Definition 3.5 *(Machines) A* machine *is a tuple*

$$\mathsf{M} = (name, Ports, States, \delta, l, Ini, Fin)$$

where

- $name \in \Sigma^+ \circ \{\sim, \epsilon\}$ *is called the* name *of* M*,*
- $Ports$ *is a* finite sequence of ports *with pairwise distinct elements,*
- $States \subseteq \Sigma^*$ *is called a set of* states*,*
- $Ini, Fin \subseteq States$ *are called the sets of* initial *and* final states*.*
- $l\colon States \to (\mathbb{N}_0 \cup \{\infty\})^{|\mathsf{in}(Ports)|}$ *is called a* length function*; we require* $l(s) = (0, \ldots, 0)$ *for all* $s \in Fin$*,*
- δ *is called a* probabilistic state-transition function *and defined as follows:*
  Let $\mathcal{I} := (\Sigma^*)^{|\mathsf{in}(Ports)|}$ *and* $\mathcal{O} := (\Sigma^*)^{|\mathsf{out}(Ports)|}$ *denote the* input set *and* output set *of* M*, respectively. Then* $\delta\colon States \times \mathcal{I} \to \mathsf{Prob}(States \times \mathcal{O})$ *with the following restrictions:*
  - *If* $I = (\epsilon, \ldots, \epsilon)$*, then* $\delta(s, I) := (s, (\epsilon, \ldots, \epsilon))$ *deterministically.*
  - $\delta(s, I) = \delta(s, I\lceil_{l(s)})$ *for all* $I \in \mathcal{I}$*. (The parts of each input beyond the length bound is ignored.)*

$\diamond$

In the following, we write $name_{\mathsf{M}}$ for the name of machine M, $Ports_{\mathsf{M}}$ for its sequence of ports, $States_{\mathsf{M}}$, Ini_{M}, Fin_{M} for its respective sets of states, $\mathcal{I}_{\mathsf{M}}$, $\mathcal{O}_{\mathsf{M}}$ for its input and output set, l_{M} for its length function and δ_{M} for its transition function.

The chosen representation makes the transition function δ independent of the port names; this enables port renaming in our proofs. The requirement for ϵ-inputs, i.e., the first restriction on δ, means that it does not matter if we switch a machine without inputs or not, i.e., there are no spontaneous transitions. The second restriction means that the part of each input beyond the current length bound for its port is ignored. In particular one can mask an input by a length bound 0 for a port. The restriction on l means that a machine ignores all inputs if it is in a final state, and hence it no longer switches.

We will often need the *port set* of a machine instead of its port sequence as well as the port set of an entire set of machines.

Definition 3.6 *(Port Set) The* port set $\mathsf{ports}(\mathsf{M})$ *of a machine* M *is the set of ports in the sequence* $Ports_{\mathsf{M}}$*. For a set* $\hat{M}$ *of machines, let* $\mathsf{ports}(\hat{M}) := \bigcup_{\mathsf{M}\in\hat{M}} \mathsf{ports}(\mathsf{M})$. ◇

In the following, we define three disjoint types (subsets) of machines. Whether a machine is of one of these types depends only on its name and ports. All machines that occur in the following will belong to one of these types.

Simple machines only have simple ports and clock out-ports, and their names are contained in Σ^+. We do not make any restrictions on their internal behavior.

Definition 3.7 *(Simple Machines) A machine* M *is* simple *iff* $name_{\mathsf{M}} \in \Sigma^+$ *and for all* $p = (\mathsf{n}, l, d) \in \mathsf{ports}(\mathsf{M})$ *we have* $l = \epsilon$ *or* $(l, d) = (^{\triangleleft}, !)$. ◇

Similar to simple machines, *default schedulers* only have simple ports and clock out-ports, except that they have one special clock in-port $\mathsf{clk}^{\triangleleft}?$, called the *default-clock* in-port. For reasons of compatibility with existing papers based on the RSIM framework, we also introduce the terminology *master scheduler* and *master-clock* in-port to denote the default scheduler and the default-clock in-port, respectively. When we define the interaction of several machines, the default-clock in-port will be used to resolve situations where the interaction cannot proceed otherwise. A default scheduler makes no outputs (i.e., formally only empty outputs) in a transition that enters a final state. This will simplify the later definition that the entire interaction between machines stops if a default scheduler enters a final state.

Definition 3.8 *(Default Schedulers) A machine* M *is a* default scheduler *iff*

- $name_{\mathsf{M}} \in \Sigma^+$,
- $\mathsf{clk}^{\triangleleft}? \in \mathsf{ports}(\mathsf{M})$,
- *for all* $p = (\mathsf{n}, l, d) \in \mathsf{ports}(\mathsf{M}) \setminus \{\mathsf{clk}^{\triangleleft}?\}$*, we have* $l = \epsilon$ *or* $(l, d) = (^{\triangleleft}, !)$*, and*
- *if* $\mathsf{Pr}(\delta_{\mathsf{M}}(s, I) = (s', O)) > 0$ *with* $s' \in Fin_{\mathsf{M}}$ *and arbitrary* $s \in States_{\mathsf{M}}$ *and* $I \in \mathcal{I}_{\mathsf{M}}$*, then* $O = (\epsilon, \ldots, \epsilon)$.

◇

If a simple machine or a default scheduler M has an out-port $\mathsf{n}!$ or an in-port $\mathsf{n}?$ we say that M is the *sending machine* or *receiving machine* for n, as shown in Figure 3.

As the third machine set, we define *buffers*. All buffers have the same predefined transition function. They model the asynchronous channels between other machines, and will later be inserted between two ports $\mathsf{n}!$ and $\mathsf{n}?$ as shown in Figure 3. More precisely, for each port name n, we define a buffer denoted as $\widetilde{\mathsf{n}}$ with three ports $\mathsf{n}^{\triangleleft}?, \mathsf{n}^{\leftrightarrow}?$, and $\mathsf{n}^{\leftrightarrow}!$. When a value is input at $\mathsf{n}^{\leftrightarrow}?$, the transition function of the buffer appends this value to an internal queue over Σ^*. An input $u \neq \epsilon$ at $\mathsf{n}^{\triangleleft}?$ is interpreted as a natural number, captured by the function nat, and the $\mathsf{nat}(u)$-th element of the internal queue is removed and output at $\mathsf{n}^{\leftrightarrow}!$. If there are less than $\mathsf{nat}(u)$ elements, the output is ϵ. As the two inputs never occur together in the upcoming run algorithm, we define that the buffer only evaluates its first non-empty input. Since the states have to be a subset of Σ^* by Definition 3.5, we embed the queue into Σ^* using the embedding function ι for sequences (see Section 2).

Definition 3.9 *(Buffers) For every* $\mathsf{n} \in \Sigma^+$ *we define a machine* $\widetilde{\mathsf{n}}$ *called a* buffer*:*

$$\widetilde{\mathsf{n}} := (\mathsf{n}\sim, (\mathsf{n}^{\triangleleft}?, \mathsf{n}^{\leftrightarrow}?, \mathsf{n}^{\leftrightarrow}!), States_{\widetilde{\mathsf{n}}}, \delta_{\widetilde{\mathsf{n}}}, l_{\widetilde{\mathsf{n}}}, Ini_{\widetilde{\mathsf{n}}}, \emptyset)$$

with

- $States_{\widetilde{\mathsf{n}}} := \{\iota(P) \mid P \in (\Sigma^*)^*\}$,
- $Ini_{\widetilde{\mathsf{n}}} := \{\iota(\,(\,)\,)\}$,
- $l_{\widetilde{\mathsf{n}}}(\iota(P)) := (\infty, \infty)$ *for all* $\iota(P) \in States_{\widetilde{\mathsf{n}}}$, *and*
- $\delta_{\widetilde{\mathsf{n}}}(\iota(P), (u, v)) := (\iota(P'), (o))$ *deterministically as follows:*

  if $u \neq \epsilon$ ***then***
  if $\mathsf{nat}(u) \leq \mathsf{size}(P)$ ***then***
  $P' := (P[i] \in P \mid i \neq \mathsf{nat}(u))$
  $o := P[\mathsf{nat}(u)]$
  else
  $P' := P$ *and* $o = \epsilon$
  end if
  else if $v \neq \epsilon$ ***then***
  $P' := P \circ (v)$ *and* $o := \epsilon$
  else
  $P' := P$ *and* $o := \epsilon$
  end if

$\diamond$

In the following, a machine with a tilde such as $\widetilde{\mathsf{n}}$ always means the unique buffer for $\mathsf{n} \in \Sigma^+$ according to Definition 3.9.

3.3 Computational Realization

For computational aspects, a machine M is regarded as implemented by a probabilistic interactive Turing machine as introduced in [60]. We need some extensions of this model of probabilistic interactive Turing machines. The main feature of interactive Turing machines is that they have communication tapes where one machine can write and one other machine can read. Thus we will use one communication tape to model each low-level connection. Probabilism is modeled by giving each Turing machine a read-only random tape containing an infinite sequence of independent, uniformly random bits. To make each Turing configuration finite, we can instead newly choose such a bit whenever a cell of the random tape is first accessed. Each Turing machine has one distinguished work tape; it may or may not have further local tapes, which are initially empty.

Our first extension concerns how the heads move on communication tapes; our choice guarantees that a machine can ignore the ends of long messages as defined by the length functions in our I/O machine model, and nevertheless read the following message. This helps machines to guarantee certain runtimes without becoming vulnerable to denial-of-service attacks by an adversary sending a message longer than this runtime. This enables liveness properties, although those are not considered in this paper. The second extension concerns restarts of machines in a multi-machine scenario. We guarantee that a switching step with only empty inputs is equivalent to no step at all, as in the I/O machine model.

By a Turing machine whose heads recognize partner heads we mean a Turing machine whose transition function has the following inputs: of course the finite state of the machine, and for each head of the machine, the content of the cell under this head and a bit denoting whether another head is on the same cell.

Definition 3.10 *(Computational Realization of Machines) A* probabilistic interactive Turing machine T *is a probabilistic multi-tape Turing machine whose heads recognize partner heads. Tapes have a left boundary, and heads start on the left-most cell.* T implements *a machine* M *as defined in Definition 3.5 if the following holds. Let* $i_{\mathsf{M}} := |\mathsf{in}(Ports_{\mathsf{M}})|$. *We write "finite state" for a state of the finite control of* T *and "*M*-state" for an element of* $States_{\mathsf{M}}$.

a) T *has a read-only tape for each in-port of* M. *Here the head never moves left, nor to the right of the other head on that tape. For each out-port of* M, T *has a write-only tape where the head never moves left of the other head on that tape.*

b) T *has special finite states* $\mathsf{restart}_{int}$ *(where "int" is similar to an interrupt vector) with* $int \in \mathcal{P}(\{1, \ldots, i_{\mathsf{M}}\})$ *for waking up asynchronously with inputs at a certain set of ports,* sleep *denoting the end of an* M*-transition, and* end *for termination. Here* $\mathsf{restart}_{\emptyset} = \mathsf{sleep}$, *i.e.,* T *needs no time for "empty" transitions.*

c) T *realizes* $\delta_{\mathsf{M}}(s, I)$ *as follows for all* $s \in States_{\mathsf{M}}$ *and* $I \in \mathcal{I}_{\mathsf{M}}$*: Let* T *start in finite state* $\mathsf{restart}_{int}$ *where* $int := \{i \mid I[i]\lceil_{l_{\mathsf{M}}(s)[i]} \neq \epsilon\} \neq \emptyset$, *with worktape content* s, *and with* I_i *on the* i*-th input tape from (including)* T*'s head to (excluding) the other head on this tape for all* i. *Let* s' *be the worktape content in the next finite state* sleep *or* end, *and* $O[i]$ *the content of the* i*-th output tape from (including) the other head to (excluding)* T*'s head in that state. Then the pairs* (s', O) *are distributed according to* $\delta_{\mathsf{M}}(s, I)$, *and the finite state is* end *iff* $s' \in Fin_{\mathsf{M}}$.

$\diamond$

The main reason to introduce a Turing-machine realization of the machine model is to define complexity notions. The interesting question is how we handle the inputs on communication tapes in the complexity definition, in particular for the notion of polynomial time, which is the maximum complexity allowed to adversaries against typical cryptographic systems.

One can imagine three degrees of an interactive machine being polynomial-time. The weakest would be that each M-transition only needs time polynomial in the current inputs and the current state, i.e., the current content of the local tapes. However, such a machine might double the size of its current state in each M-transition; then it would be allowed time exponential in an initial security parameter after a linear number of M-transitions. Hence we do not use this notion.

In the medium notion, which we call *weakly polynomial-time*, the runtime of the machine is polynomial in the overall length of its inputs, including the initial worktape content. Equivalently, the runtime for each M-transition is polynomial in the overall length of the inputs received so far. This makes the machine a permissible adversary when interacting with a cryptographic system which is in itself polynomially bounded. However, several weakly polynomial-time machines together (or even one with a self-connection) can become too powerful. E.g., each new output may be twice as long as the inputs so far. Then after a linear number of M-transitions, these weakly polynomial-time machines are allowed time exponential in an initial security parameter. We nevertheless use weakly polynomial-time machines sometimes, because many functionalities are naturally weakly polynomial-time and not naturally polynomial-time in the following strong sense. However, one always has to keep in mind that this notion does not compose as we just saw.

Finally, polynomial-time machines are those who only need time polynomial in their initial worktape content, independent of all inputs on communication tapes.

A run of a probabilistic, interactive Turing machine is a valid sequence of configurations of T (defined as for other Turing machines), where the finite state end can only occur in the last configuration of the run.

Definition 3.11 *(Complexity of Machines)*

a) A probabilistic interactive Turing machine T *is* polynomial-time *iff there exists a polynomial* P *such that all possible runs of* T *are of length at most* $P(k)$*, i.e, take at most* $P(k)$ *Turing steps, where* k *is the length of the initial worktape content.*

b) T *is called weakly polynomial-time iff there exists a polynomial* P *such that for every fixed initial worktape content and fixed contents of all input tapes with overall length* k'*, all possible runs of* T *are of length at most* $P(k')$*.*

c) A machine M *according to Definition 3.5 is (weakly) polynomial-time iff there exists a (weakly) polynomial-time probabilistic interactive Turing machine that implements* M *according to Definition 3.10.*

More generally, if we say without further qualification that T *fulfills some complexity measure, we mean that all possible runs of the machine fulfill this measure as a function of the length* k *of the initial worktape content.* ◇

Besides the deterministic runtime bounds that we have defined and that we will use in the following, one could define bounds for the expected runtime. Alternative definitions of polynomial-time specifically for reactive systems have recently been proposed [64, 70], see [70] for a comparison. These definitions can be easily incorporated in the Reactive Simulatability framework.

3.4 Collections of Machines

After introducing individual machines, we now focus on *collections* of finitely many machines, with the intuition that these machines interact. Each machine in a collection must be uniquely determined by its name, and their port sets must be pairwise disjoint so that the naming conventions for low- and high-level complements will lead to well-defined one-to-one connections.

Definition 3.12 *(Collections)*

a) A collection $\hat{C}$ *is a finite set of machines with pairwise different machine names, pairwise disjoint port sets, and where each machine is a simple machine, a default scheduler, or a buffer.*

b) A collection is called (weakly) polynomial-time *iff all its non-buffer machines are (weakly) polynomial-time.*

c) If $\widetilde{\mathsf{n}}, \mathsf{M} \in \hat{C}$ *and* $\mathsf{n}^{\triangleleft}! \in \mathsf{ports}(\mathsf{M})$ *then we call* M *the* scheduler for *buffer* $\widetilde{\mathsf{n}}$ *in* $\hat{C}$*, and we omit "in* $\hat{C}$*" if it is clear from the context.*

◇

If a port and its low-level complement are both contained in the port set of the collection, they form a *low-level connection*; recall Definition 3.4. *High-level connections* for simple ports are defined similarly. If a port p is contained in the port set of the collection but its low-level complement is not, p is called *free*. Free ports will later be used to connect external machines to the collection. For instance, a collection may consist of machines that execute a cryptographic protocol, and their free ports can be connected to users and an adversary.

Definition 3.13 *(Connections) Let $\hat{C}$ be a collection.*

a) If $p, p^c \in \mathsf{ports}(\hat{C})$ then $\{p, p^c\}$ is called a low-level connection. *The set $\mathsf{gr}(\hat{C}) := \{\{p, p^c\} \mid p, p^c \in \mathsf{ports}(\hat{C})\}$ is called the* low-level connection graph *of $\hat{C}$.*

b) By $\mathsf{free}(\hat{C}) := \mathsf{ports}(\hat{C}) \setminus \mathsf{ports}(\hat{C})^c$ we denote the free *ports in $\hat{C}$.*

c) If $p, p^C \in \mathsf{ports}(\hat{C})$ then $\{p, p^C\}$ is called a high-level connection. *The set $\mathsf{Gr}(\hat{C}) := \{\{p, p^C\} \mid p, p^C \in \mathsf{ports}(\hat{C})\}$ is called the* high-level connection graph *of $\hat{C}$.*

◇

Given a collection of (usually simple) machines, we want to add buffers for all high-level connections to model asynchronous timing. This is modeled by the *completion* of a collection $\hat{C}$. The completion is the union of $\hat{C}$ and buffers for all existing ports except the default-clock in-port. Note that completion leaves already existing buffers in $\hat{C}$ unchanged. A collection is called *closed* if the only free port of its completion is the default-clock in-port $\mathsf{clk}^{\triangleleft}?$. This implies that a closed collection has precisely one default scheduler, identified by having the unique default-clock in-port.

Definition 3.14 *(Closed Collection, Completion) Let $\hat{C}$ be a collection.*

a) The completion $[\hat{C}]$ *of $\hat{C}$ is defined as*

$$[\hat{C}] := \hat{C} \cup \{\widetilde{\mathsf{n}} \mid \exists l, d\colon (\mathsf{n}, l, d) \in \mathsf{ports}(\hat{C}) \setminus \{\mathsf{clk}^{\triangleleft}?\}\}.$$

b) $\hat{C}$ is closed *iff $\mathsf{free}([\hat{C}]) = \{\mathsf{clk}^{\triangleleft}?\}$, and $\hat{C}$ is* complete *iff $[\hat{C}] = \hat{C}$.*

◇

3.5 Runs and their Probability Spaces

For a closed collection, we now define *runs* (in other terminologies executions or traces).

Informal Description

We start with an informal description. Machines switch sequentially, i.e., we have exactly one active machine M at any time. If this machine has clock out-ports, then besides its "normal" outputs, it can select the next message to be delivered by scheduling a buffer via one of these clock out-ports. If the selected message (i.e., a message at the selected position) exists in the buffer's internal queue, it is delivered by the buffer and the unique receiving machine becomes the next active machine. If M tries to schedule multiple messages, only one is taken, and if it schedules none or the message does not exist, the default scheduler X (which exists since we consider a closed collection) becomes active.

Next we give a more precise, but still only semi-formal definition of runs. Runs and their probability spaces are defined inductively by the following algorithm for each tuple *ini* of initial states of the machines of a closed collection $\hat{C}$. The algorithm maintains variables for the states of all machines of the collection and treats each port as a variable over Σ^*, initialized with ϵ except for $\mathsf{clk}^{\triangleleft}? := 1$. The algorithm further maintains a variable $\mathsf{M_{CS}}$ ("current scheduler") over machine names, initialized with $\mathsf{M_{CS}} := \mathsf{X}$, for the name of the currently active simple machine or default scheduler, and a variable r for the resulting run, an initially empty list. The algorithm operates in five phases, which are illustrated in Figure 4. Probabilistic choices only occur in Phase 1.

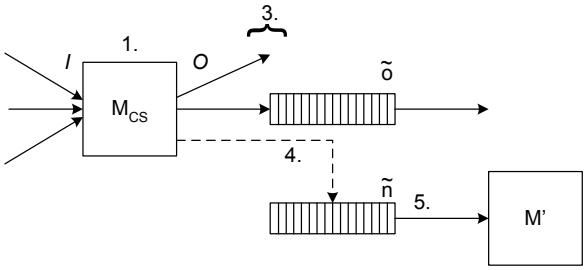

Figure 4: Phases of the run algorithm.

1. *Switch current scheduler:* Switch the current machine $\mathsf{M_{CS}}$, i.e., set $(s', O) \leftarrow \delta_{\mathsf{M_{CS}}}(s, I)$ for its current state s and in-port values I. Then assign ϵ to all in-ports of $\mathsf{M_{CS}}$.

2. *Termination:* If X is in a final state, the run stops. (As X made no outputs in this case, this only prevents repeated inputs at the default-clock in-port.)

3. *Store outputs:* For each simple out-port $\mathsf{o!}$ of $\mathsf{M_{CS}}$ with $\mathsf{o!} \neq \epsilon$, in their given order, switch buffer $\widetilde{\mathsf{o}}$ with input $\mathsf{o}^{\leftrightarrow}? := \mathsf{o!}$. Then assign ϵ to all these ports $\mathsf{o!}$ and $\mathsf{o}^{\leftrightarrow}?$.

4. *Clean up scheduling:* If at least one clock out-port of $\mathsf{M_{CS}}$ has a value $\neq \epsilon$, let $\mathsf{n}^{\triangleleft}!$ denote the first such port according to their given order and assign ϵ to the others. Otherwise let $\mathsf{clk}^{\triangleleft}? := 1$ and $\mathsf{M_{CS}} := \mathsf{X}$ and go to Phase 1.

5. *Deliver scheduled message:* Switch $\widetilde{\mathsf{n}}$ with input $\mathsf{n}^{\triangleleft}? := \mathsf{n}^{\triangleleft}!$, set $\mathsf{n}? := \mathsf{n}^{\leftrightarrow}!$ and then assign ϵ to all ports of $\widetilde{\mathsf{n}}$ and to $\mathsf{n}^{\triangleleft}!$. If $\mathsf{n}? = \epsilon$ let $\mathsf{clk}^{\triangleleft}? := 1$ and $\mathsf{M_{CS}} := \mathsf{X}$. Else let $\mathsf{M_{CS}} := \mathsf{M'}$ for the unique machine $\mathsf{M'}$ with $\mathsf{n}? \in \mathsf{ports}(\mathsf{M'})$. Go to Phase 1.

Whenever a machine (this may be a buffer) with name $name_{\mathsf{M}}$ is switched from (s, I) to (s', O), we add a *step* $(name_{\mathsf{M}}, s, I, s', O)$ to the run r with the following two restrictions. First, we cut each input according to the respective length function, i.e., we replace I by $I' := I\lceil_{l_{\mathsf{M}}(s)}$. Secondly, we do not add the step to the run if $I' = (\epsilon, \ldots, \epsilon)$, i.e., if nothing happens in reality. This gives a family of probability distributions $(run_{\hat{C},ini})$, one for each tuple ini of initial states of the machines of the collection. Moreover, for a set $\hat{M}$ of machines, we define the restriction of runs to those steps where a machine of $\hat{M}$ switches. This is called the *view* of $\hat{M}$. Similar to runs, this gives a family of probability distributions $(view_{\hat{C},ini}(\hat{M}))$, one for each tuple ini of initial states.

Rigorous Definitions

We now define the probability space of runs rigorously. Since the semi-formal description is sufficient to understand our subsequent results and the rigorous definitions are quite technical, this subsection can be skipped at first reading.

We first define a *global state space* and a *global transition function* on these states. The global state space has five parts: the states of all machines of the collection, the current scheduler (currently active machine), a function assigning strings – the current values – to the ports of the collection, the current phase, and a subset of the out-port sequence of one machine for modeling the "for"-loop in the third phase of the informal algorithm. Additionally, the global state space contains a distinguished global final state s_{fin}.

Definition 3.15 *(Global States of a Collection) Let $\hat{C}$ be a complete, closed collection with default scheduler* X. *Let* $\mathcal{P}_{\hat{C}} := \{P \mid \exists \mathsf{M} \in \hat{C} \colon P \subseteq \mathsf{out}(Ports_{\mathsf{M}})\}$ *where* $\subseteq$ *denotes the subsequence relation.*

- *The* set of global states *of* $\hat{C}$ *is defined as*

$$States_{\hat{C}} := \times_{\mathsf{M} \in \hat{C}} States_{\mathsf{M}} \times \hat{C} \times (\Sigma^*)^{\mathsf{ports}(\hat{C})} \times \{1, \ldots, 5\} \times \mathcal{P}_{\hat{C}} \cup \{s_{fin}\}.$$

- *The* set of initial global states *of* $\hat{C}$ *is defined as*

$$Ini_{\hat{C}} := \times_{\mathsf{M} \in \hat{C}} Ini_{\mathsf{M}} \times \{\mathsf{X}\} \times \{f\} \times \{1\} \times \{()\}$$

with $f(\mathsf{clk}^{\triangleleft}?) := 1$ *and* $f(p) := \epsilon$ *for* $p \in \mathsf{ports}(\hat{C}) \setminus \{\mathsf{clk}^{\triangleleft}?\}$.

$\diamond$

On these global states, we define a global transition function. It reflects the informal run algorithm.

Definition 3.16 *(Global Transition Function) Let $\hat{C}$ be a complete, closed collection with default scheduler* X. *We define the* global transition function

$$\delta_{\hat{C}} \colon States_{\hat{C}} \to \mathsf{Prob}(States_{\hat{C}})$$

by $\delta_{\hat{C}}(s_{fin}) := s_{fin}$ *and otherwise by the following rules:*

Phase 1: Switch current scheduler.

$$((s_{\mathsf{M}})_{\mathsf{M} \in \hat{C}}, \mathsf{M}_{\mathsf{CS}}, f, 1, P) \to_p ((s'_{\mathsf{M}})_{\mathsf{M} \in \hat{C}}, \mathsf{M}_{\mathsf{CS}}, f', 2, ()) \tag{1}$$

where, with $I := f(\mathsf{in}(Ports_{\mathsf{M}_{\mathsf{CS}}}))$ *and* $O := f'(\mathsf{out}(Ports_{\mathsf{M}_{\mathsf{CS}}}))$,

- $p = \mathsf{Pr}(\delta_{\mathsf{M}_{\mathsf{CS}}}(s_{\mathsf{M}_{\mathsf{CS}}}, I) = (s'_{\mathsf{M}_{\mathsf{CS}}}, O))$,
- $s_{\mathsf{M}} = s'_{\mathsf{M}}$ *for all* $\mathsf{M} \in \hat{C} \setminus \{\mathsf{M}_{\mathsf{CS}}\}$, *and*
- $f'(\mathsf{in}(Ports_{\mathsf{M}_{\mathsf{CS}}})) = (\epsilon)^{|\mathsf{in}(Ports_{\mathsf{M}_{\mathsf{CS}}})|}$ *and* $f \equiv f'$ *on* $\mathsf{ports}(\hat{C}) \setminus \mathsf{ports}(\mathsf{M}_{\mathsf{CS}})$.

Phase 2: Termination.

$$((s_{\mathsf{M}})_{\mathsf{M} \in \hat{C}}, \mathsf{M}_{\mathsf{CS}}, f, 2, P) \to s_{fin} \qquad \text{if } s_{\mathsf{X}} \in Fin_{\mathsf{X}}; \tag{2}$$

$$((s_{\mathsf{M}})_{\mathsf{M} \in \hat{C}}, \mathsf{M}_{\mathsf{CS}}, f, 2, P) \to ((s_{\mathsf{M}})_{\mathsf{M} \in \hat{C}}, \mathsf{M}_{\mathsf{CS}}, f, 3, P') \qquad \text{if } s_{\mathsf{X}} \notin Fin_{\mathsf{X}} \tag{3}$$

where $P' = (\mathsf{o}! \in Ports_{\mathsf{M}_{\mathsf{CS}}} \mid \mathsf{o} \in \Sigma^+ \wedge f(\mathsf{o}!) \neq \epsilon)$.

Phase 3: Store outputs.

$$((s_{\mathsf{M}})_{\mathsf{M} \in \hat{C}}, \mathsf{M}_{\mathsf{CS}}, f, 3, ()) \to ((s_{\mathsf{M}})_{\mathsf{M} \in \hat{C}}, \mathsf{M}_{\mathsf{CS}}, f, 4, ()); \tag{4}$$

$$((s_{\mathsf{M}})_{\mathsf{M} \in \hat{C}}, \mathsf{M}_{\mathsf{CS}}, f, 3, P) \to ((s'_{\mathsf{M}})_{\mathsf{M} \in \hat{C}}, \mathsf{M}_{\mathsf{CS}}, f', 3, P') \qquad \text{if } P \neq () \tag{5}$$

where there exists $\mathsf{n} \in \Sigma^+$ *with*

- $P = (\mathsf{n}!) \circ P'$,

- $(s'_{\widetilde{\mathsf{n}}}, (\epsilon)) = \delta_{\widetilde{\mathsf{n}}}(s_{\widetilde{\mathsf{n}}}, (\epsilon, f(\mathsf{n}!)))$,
- $s_{\mathsf{M}} = s'_{\mathsf{M}}$ *for all* $\mathsf{M} \in \hat{C} \setminus \{\widetilde{\mathsf{n}}\}$, *and*
- $f'(\mathsf{n}!) = \epsilon$ *and* $f \equiv f'$ *on* $\mathsf{ports}(\hat{C}) \setminus \{\mathsf{n}!\}$.

Phase 4: Clean up scheduling. *Let* $Clks := (\mathsf{n}^{\triangleleft}! \in Ports_{\mathsf{M_{CS}}} \mid f(\mathsf{n}^{\triangleleft}!) \neq \epsilon)$. *Then*

$$((s_{\mathsf{M}})_{\mathsf{M}\in\hat{C}}, \mathsf{M_{CS}}, f, 4, P) \to ((s_{\mathsf{M}})_{\mathsf{M}\in\hat{C}}, \mathsf{X}, f', 1, ()) \qquad \textit{if } Clks = () \tag{6}$$

where $f'(p) = \epsilon$ *for all* $p \in \mathsf{ports}(\hat{C}) \setminus \{\mathsf{clk}^{\triangleleft}?\}$ *and* $f'(\mathsf{clk}^{\triangleleft}?) = 1$, *and*

$$((s_{\mathsf{M}})_{\mathsf{M}\in\hat{C}}, \mathsf{M_{CS}}, f, 4, P) \to ((s_{\mathsf{M}})_{\mathsf{M}\in\hat{C}}, \mathsf{M_{CS}}, f', 5, P') \qquad \textit{if } Clks \neq () \tag{7}$$

where

- $P' = (Clks[1])$, *and*
- $f'(Clks[1]^c) = f(Clks[1])$ *and* $f'(p) = \epsilon$ *for all* $p \in \mathsf{ports}(\hat{C}) \setminus \{Clks[1]^c\}$.

Phase 5: Deliver scheduled message.

$$((s_{\mathsf{M}})_{\mathsf{M}\in\hat{C}}, \mathsf{M_{CS}}, f, 5, P) \to s_{fin} \qquad \textit{if } \nexists \mathsf{n} \in \Sigma^+ \colon P = (\mathsf{n}^{\triangleleft}!); \tag{8}$$

$$((s_{\mathsf{M}})_{\mathsf{M}\in\hat{C}}, \mathsf{M_{CS}}, f, 5, (\mathsf{n}^{\triangleleft}!)) \to ((s'_{\mathsf{M}})_{\mathsf{M}\in\hat{C}}, \mathsf{M'_{CS}}, f', 1, ()) \tag{9}$$

where there exists $o \in \Sigma^+$ *such that*

- $s_{\mathsf{M}} = s'_{\mathsf{M}}$ *for all* $\mathsf{M} \in \hat{C} \setminus \{\widetilde{\mathsf{n}}\}$,
- $(s'_{\widetilde{\mathsf{n}}}, (o)) = \delta_{\widetilde{\mathsf{n}}}(s_{\widetilde{\mathsf{n}}}, (f(\mathsf{n}^{\triangleleft}?), \epsilon))$,

and

- *either* $o = \epsilon$ *and* $\mathsf{M'_{CS}} = \mathsf{X}$ *and* $f'(\mathsf{clk}^{\triangleleft}?) = 1$ *and* $f'(p) = \epsilon$ *for all* $p \in \mathsf{ports}(\hat{C}) \setminus \{\mathsf{clk}^{\triangleleft}?\}$
- *or* $o \neq \epsilon$ *and* $\mathsf{n}? \in Ports_{\mathsf{M'_{CS}}}$ *and* $f'(\mathsf{n}?) = o$ *and* $f'(p) = \epsilon$ *for all* $p \in \mathsf{ports}(\hat{C}) \setminus \{\mathsf{n}?\}$.

$\diamond$

Rule (8) has only been included to define the function δ on the entire state space $States_{\hat{C}}$ as claimed at the beginning of the definition. It will not matter in the execution since the previous state has to be in Phase 4, and this ensures that P contains exactly one clock port. Similarly, the other rules make no assumptions about reachable states. Furthermore, $\delta(s)$ is indeed an element of $\mathsf{Prob}(States_{\hat{C}})$ for every $s \in States_{\hat{C}}$: It is deterministic for all states s except those treated in Rule (1). For those, the claim follows immediately from the fact that $\delta_{\mathsf{M_{CS}}}(s_{\mathsf{M_{CS}}}, I)$ is a finite distribution.

Given the global probabilistic transition function δ, we now obtain probability distributions on sequences of global states by canonical constructions as for Markov chains. This even holds for infinite sequences by the theorem of Ionescu-Tulcea; see, e.g., Section V.1 of [86]. More precisely, we obtain one such probability distribution for every global initial state. Applying the theorem of Ionescu-Tulcea to our situation yields the following lemma.

Lemma 3.1 (Probabilities of State Sequences) *Let $\hat{C}$ be a complete, closed collection, and let an initial global state $ini \in Ini_{\hat{C}}$ be given. For each set of fixed-length sequences $States^i_{\hat{C}}$ with $i \in \mathbb{N}$, we can define a finite probability distribution $PStates_{\hat{C},ini,i}$ by*

$$\mathsf{Pr}(S) = \prod_{j=2}^{i} \mathsf{Pr}(\delta_{\hat{C}}(S_{j-1}) = S_j)$$

for every sequence $S = (S_1, \ldots, S_i)$ over $States_{\hat{C}}$ with $S_1 = ini$, and $\mathsf{Pr}(S) = 0$ otherwise.

Furthermore, for every sequence $S \in States^i_{\hat{C}}$, let $\mathsf{Rect}(S)$ denote the corresponding rectangle *of infinite sequences with this prefix, i.e., $\mathsf{Rect}(S) := \{S' \in States^\infty_{\hat{C}} \mid S'\lceil_i = S\}$. Then there exists a unique probability distribution $PStates_{\hat{C},ini,\infty}$ over $States^\infty_{\hat{C}}$ whose value for every rectangle $R := \mathsf{Rect}(S)$ with $S \in States^i_{\hat{C}}$ equals $\mathsf{Pr}(S)$, or more precisely,*

$$\mathsf{Pr}_{PStates_{\hat{C},ini,\infty}}(R) := \mathsf{Pr}_{PStates_{\hat{C},ini,i}}(S).$$

We usually omit the indices "∞" and "i" of these distributions; this cannot lead to confusion. □

Varying *ini* gives a family of probability distributions over $States^\infty_{\hat{C}}$; we write it

$$PStates_{\hat{C}} := (PStates_{\hat{C},ini})_{ini \in Ini_{\hat{C}}}.$$

So far we have defined probabilities for sequences of entire global states. Each step in the runs introduced semi-formally above intuitively corresponds to the difference between two successive states: Moreover, only the switching of machines is considered in a run, i.e., the intermediate phases for termination checks and cleaning up scheduling are omitted, as well as the switching of machines with empty input (after application of the length function) because nothing happens then. We first define the set of possible steps, i.e., five-tuples containing the name of the currently switched machine M, its old state, its input tuple, its new state, and its output tuple. Furthermore, we define an encoding of steps as strings for the sole purpose of defining overall lengths of potential runs.

Definition 3.17 *(Steps) The set of* steps *is defined as*

$$Steps := (\Sigma^+ \circ \{\sim, \epsilon\}) \times \Sigma^* \times (\Sigma^*)^* \times \Sigma^* \times (\Sigma^*)^*.$$

Let ι_{Steps}: $Steps \to \Sigma^+$ denote an efficient encoding from Steps into the non-empty strings over Σ. For all $r = (r_i)_{i \in \mathcal{I}} \in Steps^*$ let $\mathsf{len}(r) := \sum_{i \in \mathcal{I}} \mathsf{size}(\iota_{Steps}(r))$, and for $r \in Steps^\infty$ let $\mathsf{len}(r) := \infty$. ◇

In particular, we assume that the projection to the individual components of a step is efficiently computable. Now we define a mapping that extracts a run from a step sequence. Our definition first extracts a sequence of one potential step from each pair of a global state and the next one. This first part maps intermediate phases to steps with empty input tuples, so that it is then sufficient to restrict the obtained sequence to the elements with a non-empty input tuple for getting rid of both the intermediate phases and of actual switching with empty inputs. When extracting a potential step, we mainly distinguish whether the current scheduler switches or a buffer; such a buffer is always indicated by the first element in the port sequence of outputs still to be handled, the fifth element of the global state.

Definition 3.18 *(Run Extraction) Let $\hat{C}$ be a complete, closed collection. Then the* run extraction *of $\hat{C}$ is the function*

$$run_{\hat{C}} \colon States_{\hat{C}}^{\infty} \to Steps^* \cup Steps^{\infty}$$

defined as follows. (It is independent of $\hat{C}$ except for its domain.) Let $S = (S_i)_{i \in \mathbb{N}} \in States_{\hat{C}}^{\infty}$ where for each $i \in \mathbb{N}$ either $S_i = s_{fin}$ or $S_i = ((s_{\mathsf{M}}^i)_{\mathsf{M} \in \hat{C}}, \mathsf{M}_{\mathsf{CS}}^i, f^i, j^i, P^i)$. If $i^* := \min\{i \in \mathbb{N} \mid s_i = s_{fin}\}$ exists, let $\mathcal{I} := \{1, \ldots, i^* - 2\}$, otherwise $\mathcal{I} := \mathbb{N}$. We define a sequence $pot\_step(S) = ((\mathsf{n}_i, s_i, I_i, s_i', O_i))_{i \in \mathcal{I}}$ as follows. (Thus we already omit the last termination check.)

- *If $P^i = ()$ then, with $\mathsf{M} := \mathsf{M}_{\mathsf{CS}}^i$,*
  - *$\mathsf{n}_i := name_{\mathsf{M}}$,*
  - *$s_i := s_{\mathsf{M}}^i$ and $s_i' := s_{\mathsf{M}}^{i+1}$, and*
  - *if $j^i = 1$ then $I_i := f^i(\mathsf{in}(Ports_{\mathsf{M}}))\lceil_{l_{\mathsf{M}}(s_i)}$ and $O_i := f^{i+1}(\mathsf{out}(Ports_{\mathsf{M}}))$, else $I_i := (\epsilon)$ and $O_i := (\epsilon)$.*
- *If $P^i \neq ()$, let $p := P^i[1]$ and $p \in \mathsf{ports}(\widetilde{\mathsf{n}})$. Then*
  - *$\mathsf{n}_i := \mathsf{n}\!\sim$.*
  - *$s_i := s_{\widetilde{\mathsf{n}}}^i$ and $s_i' := s_{\widetilde{\mathsf{n}}}^{i+1}$,*
  - *$I_i := f^i((\mathsf{n}^{\triangleleft}?, \mathsf{n}^{\leftrightarrow}?))$ and $O_i := f^{i+1}((\mathsf{n}^{\leftrightarrow}!))$.*

Then $run_{\hat{C}}(S) := ((\mathsf{n}_i, s_i, I_i, s_i', O_i) \in pot\_step(S) \mid I_i \neq (\epsilon, \ldots, \epsilon))$. For every number $l \in \mathbb{N}$, let $run_{\hat{C},l}$ denote the extraction of l-step prefixes of runs,

$$run_{\hat{C},l} \colon States_{\hat{C}}^{\infty} \to \bigcup_{i \leq l} Steps^i$$

with $run_{\hat{C},l}(S) := run_{\hat{C}}(S)\lceil_l$. ◇

The run extraction is a random variable on every probability space over $States_{\hat{C}}^{\infty}$. For our particular probability space, this induces a family $run_{\hat{C}} = (run_{\hat{C},ini})_{ini \in Ini_{\hat{C}}}$ of probability distributions over $Steps^* \cup Steps^{\infty}$ via

$$\mathsf{Pr}_{run_{\hat{C},ini}}(r) := \mathsf{Pr}_{PStates_{\hat{C},ini}}(run_{\hat{C}}^{-1}(r))$$

for all $r \in Steps^* \cup Steps^{\infty}$, where $run_{\hat{C}}^{-1}(r)$ is the set of pre-images of r. For a function $l \colon Ini_{\hat{C}} \to \mathbb{N}$, this similarly gives a family of probability distributions

$$run_{\hat{C},l} = (run_{\hat{C},ini,l(ini)})_{ini \in Ini_{\hat{C}}},$$

each over $\bigcup_{i \leq l(ini)} Steps^i$.

We finally introduce the *view* of a set $\hat{M}$ of machines in a collection. It is the restriction of a run (a sequence of steps) to those steps whose machine name belongs to $\hat{M}$. The extraction function is independent of the collection, but we index it with $\hat{C}$ anyway for similarity with the run notation.

Definition 3.19 *(Views) Let $\hat{C}$ be a complete, closed collection and $\hat{M} \subseteq \hat{C}$. The* view *of $\hat{M}$ in $\hat{C}$ is the function $view_{\hat{C}}(\hat{M})$: $Steps^* \cup Steps^\infty \to Steps^* \cup Steps^\infty$ with*

$$view_{\hat{C}}(\hat{M})(r) := (s_i \in r \mid \exists \mathsf{M} \in \hat{M} : s_i[1] = name_{\mathsf{M}}).$$

For every number $l \in \mathbb{N}$ let $view_{\hat{C},l}(\hat{M})$ denote the extraction of l-step prefixes of a view, i.e., $view_{\hat{C},l}(\hat{M})$: $Steps^* \cup Steps^\infty \to \bigcup_{i \leq l} Steps^i$ with $view_{\hat{C},l}(\hat{M})(r) := view_{\hat{C}}(\hat{M})(r)\lceil_l$. ◇

For a singleton $\hat{M} = \{\mathsf{M}\}$, we write $view_{\hat{C}}(\mathsf{M})$ instead of $view_{\hat{C}}(\{\hat{M}\})$, and similar for l-step prefixes. Based on the family $run_{\hat{C}}$ of probability distributions of runs, this induces a family of probability distributions

$$view_{\hat{C}}(\hat{M}) = (view_{\hat{C},ini}(\hat{M}))_{ini \in Ini_{\hat{C}}}$$

over $Steps^* \cup Steps^\infty$. For a function l: $Ini_{\hat{C}} \to \mathbb{N}$, this similarly gives a family $view_{\hat{C},l} = (view_{\hat{C},ini,l(ini)})_{ini \in Ini_{\hat{C}}}$ over $\bigcup_{i \leq l(ini)} Steps^i$.

Finally, we define sets of *state-traces* and of *traces* (or step-traces). Intuitively, they are the possible sequences of global states and of steps, respectively. More precisely, each finite prefix of such a sequence happens with positive probability.

Definition 3.20 *(State-Trace, Trace) Let $\hat{C}$ be a complete, closed collection and $ini \in Ini_{\hat{C}}$ an initial global state.*

- *The set of* state-traces *for ini is*

$$StateTrace_{\hat{C},ini} := \{S \in States_{\hat{C}}^\infty \mid \forall l \in \mathbb{N}\colon \mathsf{Pr}_{PStates_{\hat{C},ini,l}}(S\lceil_l) > 0\}.$$

- *The set of* traces *for ini is*

$$\begin{aligned} StepTrace_{\hat{C},ini} &:= \{tr \in Steps^* \mid \mathsf{Pr}_{run_{\hat{C},ini}}(tr) > 0\} \\ &\cup \{tr \in Steps^\infty \mid \forall l \in \mathbb{N}\colon \mathsf{Pr}_{run_{\hat{C},ini,l}}(tr\lceil_l) > 0\}. \end{aligned}$$

- *The set of* possible views *of a machine set $\hat{M} \subseteq \hat{C}$ for ini is*

$$\begin{aligned} ViewTrace_{\hat{C},ini}(\hat{M}) &:= \{v \in Steps^* \mid \mathsf{Pr}_{view_{\hat{C},ini}}(\hat{M})(v) > 0\} \\ &\cup \{v \in Steps^\infty \mid \forall l \in \mathbb{N}\colon \mathsf{Pr}_{view_{\hat{C},ini,l}}(\hat{M})(v\lceil_l) > 0\}. \end{aligned}$$

Set $StateTrace_{\hat{C}} := \bigcup_{ini \in Ini_{\hat{C}}} StateTrace_{\hat{C},ini}$ and $Trace_{\hat{C}} := \bigcup_{ini \in Ini_{\hat{C}}} Trace_{\hat{C},ini}$ and $ViewTrace_{\hat{C}}(\hat{M}) := \bigcup_{ini \in Ini_{\hat{C}}} ViewTrace_{\hat{C},ini}(\hat{M})$.

◇

We conclude this section with two properties of state-traces and views. The first, technical one states that whenever a non-buffer machine M is switched in a state-trace of $\hat{C}$, which only happens in Phase 1, there is at most one port $p \in \mathsf{ports}(\hat{C})$ with a non-empty value, and it must fulfill $p \in \mathsf{ports}(\mathsf{M})$.

Lemma 3.2 (Unique Inputs in Traces) *Let $\hat{C}$ be a complete, closed collection. Let $S := (S_i)_{i\in\mathbb{N}} \in StateTrace_{\hat{C}}$ where for each $i \in \mathbb{N}$ either $S_i = s_{fin}$ or $S_i = ((s^i_{\mathsf{M}})_{\mathsf{M}\in\hat{C}}, \mathsf{M}^i, f^i, j^i, P^i)$. For all $i \in \mathbb{N}$ with $S_i \neq s_{fin}$:*

$$(j^i = 1 \wedge \exists p \in \mathsf{ports}(\hat{C})\colon (f^i(p) \neq \epsilon)) \Rightarrow (p \in \mathsf{ports}(\mathsf{M}^i) \wedge \forall p' \in \mathsf{ports}(\hat{C}) \setminus \{p\}\colon (f^i(p') = \epsilon)).$$

□

Proof. For $i = 1$ this holds since $S \in StateTrace_{\hat{C}}$ implies $S_1 \in Ini_{\hat{C}}$ and each element of $Ini_{\hat{C}}$ fulfills the claim with $p = \mathsf{clk}^{\triangleleft}!$. Let now $i > 1$ and S_i with $j^i = 1$. Only two cases are possible for the previous state S_{i-1}. The case $j^{i-1} = 4$ and $\mathsf{M}^i = \mathsf{X}$ fulfills the claim for $p = \mathsf{clk}^{\triangleleft}? \in \mathsf{ports}(\mathsf{X})$, and the case $j^{i-1} = 5$ for $p = \mathsf{n}?$. ■

The second property shows that the views of polynomial-time machines are polynomially bounded, and so are runs of polynomial-time collections.

Lemma 3.3 (Polynomial Views and Traces) *Let $\hat{C}$ be a complete, closed collection and let $\hat{M} \subseteq \hat{C}$ be polynomial-time. For all $ini = ((ini_{\mathsf{M}})_{\mathsf{M}\in\hat{C}}, \mathsf{X}, f, 1, ()) \in Ini_{\hat{C}}$ let $\mathsf{size}_{\hat{M}}(ini) := \sum_{\mathsf{M}\in\hat{M}} \mathsf{size}(ini_{\mathsf{M}})$. Then there exists a polynomial P such that for all $ini \in Ini_{\hat{C}}$ and all $v \in ViewTrace_{\hat{C},ini}(\hat{M})$ we have $\mathsf{len}(v) \leq P(\mathsf{size}_{\hat{M}}(ini))$.*

If the entire collection $\hat{C}$ is polynomial-time, then there exists a polynomial P such that for all $ini \in Ini_{\hat{C}}$ and all $r \in Trace_{\hat{C},ini}$ we have $\mathsf{len}(r) \leq P(\mathsf{size}_{\hat{C}}(ini))$. □

Proof. By Definition 3.11, for a polynomial-time machine M, there exists a probabilistic interactive Turing machine T that implements M and only makes a polynomial number of Turing steps, relative to the length $\mathsf{size}(ini_{\mathsf{M}})$ of its own input, which is smaller than the overall input size $\mathsf{size}_{\hat{M}}(ini)$ of the considered machines. Consequently, T and hence M can only build up a polynomial-size state and outputs, and read a polynomial-size part of its inputs. Each step in the view requires at least one Turing step; hence there is also only a polynomial number of these steps. Moreover, only states, outputs, and read parts of the inputs are part of the steps (see Definitions 3.16 and 3.18). This proves the first statement.

A run of a polynomial-time collection $\hat{C}$ consists of the steps of its polynomial-time machines and its buffers. Buffers are not polynomial-time, but weakly polynomial-time; this follows immediately from the assumption we made about the encoding ι of the internal queue. Each buffer obtains all its inputs from polynomial-time machines; hence its overall input is of polynomial length. Thus each buffer only makes a polynomial number of Turing steps. This yields an overall polynomial size of its steps as above. ■

4 Security-Specific System Model

Security-specific *structures* are defined as collections of machines with distinguished service ports for the honest users, as explained in the introduction. Such structures are augmented by arbitrary machines H and A representing the honest users and the adversary, who can interact. We then speak of a *configuration*. For configurations, we also introduce specific families of executions corresponding to different security parameters for cryptographic aspects.

Definition 4.1 *(Structures and Service Ports) A* structure *is a pair $struc = (\hat{M}, S)$ where $\hat{M}$ is a collection of simple machines with $\{1\}^* \subseteq Ini_{\mathsf{M}}$ for all $\mathsf{M} \in \hat{M}$, and $S \subseteq \mathsf{free}([\hat{M}])$. The set S is called* service ports. ◇

Forbidden ports for users of a structure are those that clash with port names of given machines and those that would link the user to a non-service port.

Definition 4.2 *(Forbidden Ports) For a structure* $(\hat{M}, S)$ *let* $\bar{S}_{\hat{M}} := \mathsf{free}([\hat{M}]) \setminus S$. *We call* $\mathsf{forb}(\hat{M}, S) := \mathsf{ports}(\hat{M}) \cup \bar{S}^c_{\hat{M}}$ *the* forbidden ports. ◇

A *system* is a set of structures. The idea behind systems, as motivated in the introduction, is that there may be different actual structures depending on the set of actually malicious participants. Typical derivations of systems from one explicitly defined intended structure and a trust model will be discussed in Section 6.

Definition 4.3 *(Systems) A* system *Sys is a set of structures. It is (weakly) polynomial-time iff the machine collections* $\hat{M}$ *of all its structures are (weakly) polynomial-time.* ◇

A *configuration* consists of a structure together with a *user machine* and an *adversary machine* (or *user* and *adversary* for short). The user is restricted to connecting to the service ports. The adversary closes the collection, i.e., it connects to the remaining service ports, to the other free ports $\bar{S}_{\hat{M}}$ of the collection, and to the free ports of the user. Thus, user and adversary can interact, e.g., for modeling active attacks.

Definition 4.4 *(Configurations)*

a) A configuration *of a structure* $(\hat{M}, S)$ *is a tuple* $conf = (\hat{M}, S, \mathsf{H}, \mathsf{A})$ *where*

- H *is a machine called* user *without forbidden ports, i.e.,* $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(\hat{M}, S) = \emptyset$, *and with* $\{1\}^* \subseteq Ini_{\mathsf{H}}$,
- A *is machine called* adversary *with* $\{1\}^* \subseteq Ini_{\mathsf{A}}$,
- *and the completion* $\hat{C} := [\hat{M} \cup \{\mathsf{H}, \mathsf{A}\}]$ *is a closed collection.*

b) The set of configurations of $(\hat{M}, S)$ *is written* $\mathsf{Conf}(\hat{M}, S)$. *The set of configurations of* $(\hat{M}, S)$ *with polynomial-time user* H *and adversary* A *is written* $\mathsf{Conf}_{\mathsf{poly}}(\hat{M}, S)$.

c) The set of configurations of a system Sys is defined as $\mathsf{Conf}(Sys) := \bigcup_{(\hat{M},S)\in Sys} \mathsf{Conf}(\hat{M}, S)$, *and similarly* $\mathsf{Conf}_{\mathsf{poly}}(Sys) := \bigcup_{(\hat{M},S)\in Sys} \mathsf{Conf}_{\mathsf{poly}}(\hat{M}, S)$.

We omit the index "poly*" from* $\mathsf{Conf}_{\mathsf{poly}}(Sys)$ *if it is clear from the context.* ◇

In cryptographic applications, all machines typically start with the same security parameter. An informal description of runs and views of configurations based on the informal description of runs and views of collections from Section 3.5 is thus simply obtained by restricting the tuple *ini* of initial states. For reasons of rigor, we define runs and views of configurations based on the rigorous definitions of global states, runs, and views of collections from Section 3.5 by making a global constraint on the valid global initial states in the family of runs of a configuration that the security parameters are equal.

Definition 4.5 *(Runs and Views of Configurations) Let* $conf = (\hat{M}, S, \mathsf{H}, \mathsf{A})$ *be a configuration and* $\hat{C} := [\hat{M} \cup \{\mathsf{H}, \mathsf{A}\}]$. *We define* $Ini_{conf} := \{((1^k)_{\mathsf{M}\in\hat{M}\cup\{\mathsf{H},\mathsf{A}\}} \circ (\iota(\epsilon))_{\widetilde{\mathsf{n}}\in\hat{C}}, (\mathsf{X}, f, 1, ())) \mid k \in \mathbb{N}\} \subseteq Ini_{\hat{C}}$ *with* X *and* f *as in Definition 3.15. Then we define the* family of probability distributions of runs of the configuration *as*

$$run_{conf} := (run_{\hat{C},ini})_{ini\in Ini_{conf}}$$

and for all sets $\hat{M}' \subseteq \hat{C}$ *the* family of probability distributions of views *similarly*

$$view_{\hat{C}}(\hat{M}') := (view_{\hat{C},ini}(\hat{M}'))_{ini \in Ini_{conf}},$$

and analogously for l-step prefixes. Furthermore, we identify Ini_{conf} *with* $\mathbb{N}$ *and thus write* $run_{conf,k}$ *etc. for the individual probability distributions in the families.* ◇

5 Reactive Simulatability

Reactive simulatability, our notion of secure refinement, is defined for individual structures and lifted to entire systems. Two structures $struc_1$ and $struc_2$ can be compared if they have the same service ports, so that the same honest users can connect to them. In other words, they offer the same interface for the design of a larger system, so that either $struc_1$ or $struc_2$ can be plugged into that system. Now $struc_1$ is considered *at least as secure as* $struc_2$, written $struc_1 \geq struc_2$, if whatever any adversary A_1 can do to any honest user H using $struc_1$, some adversary A_2 can do to the same H using $struc_2$ essentially with the same probability. More precisely, the families of views of H in these two configurations are indistinguishable.

Different variants of indistinguishability are based on different classes of small functions (occurring as differences). The most important class is that of negligible functions; additionally we define closure properties that we require of suitable classes of small functions.

Definition 5.1 *(Small Functions)*

a) The class *NEGL* of *negligible functions* contains all functions $s\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ that decrease faster than the inverse of every polynomial, i.e., for all positive polynomials Q $\exists k_0\ \forall k > k_0 : s(k) < \frac{1}{Q(k)}$.

b) A set *SMALL* of functions $\mathbb{N} \to \mathbb{R}_{\geq 0}$ is a *suitable class of small functions* if it is closed under addition, and with a function g also contains every function $g'\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ with $g' \leq g$.

◇

Typical classes of small functions are *EXPSMALL*, which contains all functions bounded by $Q(k) \cdot 2^{-k}$ for a polynomial Q, and the larger class *NEGL*.

Simulatability is based on indistinguishability of views; hence we repeat the definition of indistinguishability, essentially from [100].

Definition 5.2 *(Indistinguishability) Two families* $(\mathsf{var}_k)_{k\in\mathbb{N}}$ *and* $(\mathsf{var}'_k)_{k\in\mathbb{N}}$ *of probability distributions (or random variables) on common domains* $(D_k)_{k\in\mathbb{N}}$ *are*

a) perfectly indistinguishable *("=") iff* $\forall k \in \mathbb{N}\colon \mathsf{var}_k = \mathsf{var}'_k$.

b) statistically indistinguishable *("*$\approx_{SMALL}$*") for a suitable class SMALL of small functions iff the distributions are discrete and their statistical distances, as a function of k, are small, i.e.,*

$$(\Delta_{\mathsf{stat}}(\mathsf{var}_k, \mathsf{var}'_k))_{k\in\mathbb{N}} := (\sup_{\substack{V_k \subseteq D_k \\ V_k \text{ measurable}}} |\mathsf{Pr}(\mathsf{var}_k \in V_k) - \mathsf{Pr}(\mathsf{var}'_k \in V_k)|)_{k\in\mathbb{N}} \in SMALL.$$

c) computationally indistinguishable *("$\approx_{\mathsf{poly}}$") iff for every algorithm* Dis *(the* distinguisher*) that is probabilistic polynomial-time in its first input,*

$$(|\mathsf{Pr}(\mathsf{Dis}(1^k, \mathsf{var}_k) = 1) - \mathsf{Pr}(\mathsf{Dis}(1^k, \mathsf{var}'_k) = 1)|)_{k\in\mathbb{N}} \in NEGL.$$

(Intuitively, Dis*, given the security parameter and an element chosen according to either* var_k *or* var'_k*, tries to guess which distribution the element came from.)*

We write $\approx$ *if we want to treat all cases together.* ◇

Note that for countable D_k, statistical indistinguishability can be rewritten in the more common form of requiring $(\frac{1}{2}\sum_{d\in D_k} |\mathsf{Pr}(\mathsf{var}_k = d) - \mathsf{Pr}(\mathsf{var}'_k = d)|)_{k\in\mathbb{N}} \in SMALL$.

We now present the reactive simulatability definition. One technical problem is that a user might legitimately connect to the service ports in a configuration of $(\hat{M}_1, S)$, but in a configuration of $(\hat{M}_2, S)$ the same user might have forbidden ports. This is excluded by considering *suitable configurations* only.

Definition 5.3 *(Suitable Configurations for Structures) Let* $(\hat{M}_1, S)$ *and* $(\hat{M}_2, S)$ *be structures with the same set of service ports. The set of* suitable configurations $\mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S) \subseteq \mathsf{Conf}(\hat{M}_1, S)$ *is defined by* $(\hat{M}_1, S, \mathsf{H}, \mathsf{A}) \in \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$ *iff* $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(\hat{M}_2, S) = \emptyset$*. The set of* polynomial-time suitable configurations *is* $\mathsf{Conf}^{\hat{M}_2}_{\mathsf{poly}}(\hat{M}_1, S) := \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S) \cap \mathsf{Conf}_{\mathsf{poly}}(\hat{M}_1, S)$. ◇

As we have three different notions of indistinguishability, our reactive simulatability definition also comes in three flavors. Furthermore, we distinguish the general reactive simulatability (GRSIM) as sketched so far and a stronger *universal* version (URSIM) where one adversary A_2 must be able to work for all users. Note that the term "reactive simulatability" is primarily meant to capture the overall idea of suitably comparing the views of honest users when interacting with reactive protocols; in follow-up papers, reactive simulatability is sometimes also used as a synonym for general reactive simulatability.

Definition 5.4 *(General/Universal Reactive Simulatability for Structures) Let structures* $(\hat{M}_1, S)$ *and* $(\hat{M}_2, S)$ *with identical sets of service ports be given.*

a) $(\hat{M}_1, S) \geq^{\mathsf{perf}}_{\mathsf{sec}} (\hat{M}_2, S)$*, spoken* perfectly at least as secure as*, iff for every configuration* $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$*, there exists a configuration* $conf_2 = (\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(\hat{M}_2, S)$ *(with the same* H*) such that*

$$view_{conf_1}(\mathsf{H}) = view_{conf_2}(\mathsf{H}).$$

b) $(\hat{M}_1, S) \geq^{SMALL}_{\mathsf{sec}} (\hat{M}_2, S)$ *for a suitable class SMALL of small functions, spoken* statistically at least as secure as*, iff for every configuration* $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$*, there exists a configuration* $conf_2 = (\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(\hat{M}_2, S)$ *(with the same* H*) such that*

$$view_{conf_1}(\mathsf{H}) \approx_{SMALL} view_{conf_2}(\mathsf{H}).^1$$

[1] Previous versions of the Reactive Simulatability framework erroneously required statistical indistinguishability to hold only for polynomially bounded prefixes of user views. As pointed out in [66], extending this requirement to user views without additional contraints is necessary to achieve compositionality results in general.

c) $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{poly}} (\hat{M}_2, S)$*, spoken* computationally at least as secure as*, iff for every configuration* $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}_{\mathsf{poly}}^{\hat{M}_2}(\hat{M}_1, S)$*, there exists a configuration* $conf_2 = (\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}_{\mathsf{poly}}(\hat{M}_2, S)$ *(with the same* H*) such that*

$$view_{conf_1}(\mathsf{H}) \approx_{\mathsf{poly}} view_{conf_2}(\mathsf{H}).$$

In all three cases, we speak of universal reactive simulatability (URSIM) *if* A_2 *in* $conf_2$ *does not depend on* H *(only on* $\hat{M}_1$*,* S*, and* A_1*), and we use the notation* $\geq_{\mathsf{sec}}^{\mathsf{uni,perf}}$ *etc. for this. In all cases, we call* $conf_2$ *an* indistinguishable configuration *for* $conf_1$*.* ◇

There is also a notion of blackbox reactive simulatability (BRSIM), where the adversary A_2 consists of a fixed part, called *simulator*, using A_1 as a blackbox submachine. However, its rigorous definition needs the notion of machine combination, which we postpone to the successor paper dealing with composition. If one can simply set $\mathsf{A}_2 := \mathsf{A}_1$, we also say that the structures are *observationally equivalent*, or simply *indistinguishable*; this corresponds to the definition in [73].

Where the difference between the types of security is irrelevant, we simply write $\geq_{\mathsf{sec}}$, and we omit the index sec if it is clear from the context.

Remark 5.1. Adding a free adversary out-port in the comparison (like the guessing-outputs used to define semantic security of encryption systems in [59]), thus taking the view of the adversary into account, does not make the definition stricter since the universal quantification over the honest user and the adversary could always be exploited to provide information that the adversary knows to the honest user: Any such out-port can be connected to an in-port added to the honest user with sufficiently large length bounds. H does not react on inputs at this new in-port, but nevertheless it is included in the view of H, i.e., in the comparison. A rigorous proof can be found in [4]. ◦

The definition of general and universal reactive simulatability can be lifted from structures to systems Sys_1 and Sys_2 by comparing their respective structures. However, we do not want to compare a structure of Sys_1 with arbitrary structures of Sys_2, but only with certain "suitable" ones. What suitable means in a concrete situation can be defined by a mapping f from Sys_1 to the powerset of Sys_2. The mapping f is called *valid* if it maps structures with the same set of service ports, so that the same user can connect.

Definition 5.5 *(Valid Mappings) Let* Sys_1 *and* Sys_2 *be two systems. A* valid mapping *between* Sys_1 *and* Sys_2 *is a function* $f\colon Sys_1 \to \mathcal{P}(Sys_2) \setminus \emptyset$ *with* $S_1 = S_2$ *for all* $(\hat{M}_1, S_1) \in Sys_1$ *and* $(\hat{M}_2, S_2) \in f((\hat{M}_1, S_1))$*. The elements of* $f((\hat{M}_1, S_1))$ *are called the* corresponding structures *for* $(\hat{M}_1, S_1)$*.* ◇

Remark 5.2. In the synchronous model in [91], we allow more general users and valid mappings. The stronger requirements here simplify the presentation and are sufficient for all cryptographic examples we considered. The report version of [91] contains non-cryptographic examples with $S_1 \neq S_2$. ◦

Definition 5.6 *(General/Universal Reactive Simulatability for Systems) Let systems* Sys_1 *and* Sys_2 *be given, and let* f *be a valid mapping between* Sys_1 *and* Sys_2*.*

a) $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{perf}} Sys_2$, *spoken* perfectly at least as secure as, *iff for every* $(\hat{M}_1, S) \in Sys_1$ *there exists* $(\hat{M}_2, S) \in f((\hat{M}_1, S))$ *such that*

$$(\hat{M}_1, S) \geq_{\mathsf{sec}}^{f,\mathsf{perf}} (\hat{M}_2, S).$$

b) $Sys_1 \geq_{\mathsf{sec}}^{f,SMALL} Sys_2$ *for a suitable class SMALL of small functions, spoken* statistically at least as secure as, *iff for every* $(\hat{M}_1, S) \in Sys_1$ *there exists* $(\hat{M}_2, S) \in f((\hat{M}_1, S))$ *such that*

$$(\hat{M}_1, S) \geq_{\mathsf{sec}}^{f,SMALL} (\hat{M}_2, S).$$

c) $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{poly}} Sys_2$, *spoken* computationally at least as secure as, *iff for every* $(\hat{M}_1, S) \in Sys_1$ *there exists* $(\hat{M}_2, S) \in f((\hat{M}_1, S))$ *such that*

$$(\hat{M}_1, S) \geq_{\mathsf{sec}}^{f,\mathsf{poly}} (\hat{M}_2, S).$$

In all three cases, we speak of universal reactive simulatability *if the respective relation on structures fulfills universal simulatability, and we use the notation* $\geq_{\mathsf{sec}}^{f,\mathsf{uni},\mathsf{perf}}$ *etc. for this.* ◇

Where the difference between the types of security is irrelevant, we simply write $\geq_{\mathsf{sec}}$, and we again omit the index sec if it is clear from the context.

We conclude this section with two technical lemmas capturing an equivalent definition of the forbidden ports of a structure and additional results on valid mappings and suitable configurations, which is useful in proofs.

Recall that Definition 4.4 excludes users that would connect to the forbidden ports of a configuration. The following lemma establishes an equivalent condition.

Lemma 5.1 (Users) *Let* $(\hat{M}, S)$ *be a structure. Then for all machines* H, $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(\hat{M}, S) = \emptyset$ *is equivalent to* $\mathsf{ports}(\mathsf{H}) \cap \mathsf{ports}(\hat{M}) = \emptyset$ (1) *and* $\mathsf{ports}(\mathsf{H})^c \cap \mathsf{ports}([\hat{M}]) \subseteq S$ (2). □

Proof. Let $\mathsf{inner}(\hat{C}) := \mathsf{ports}(\hat{C}) \setminus \mathsf{free}(\hat{C})$ for every collection $\hat{C}$. Clearly $\mathsf{inner}(\hat{C})^c = \mathsf{inner}(\hat{C})$. The condition on the left-hand side is equivalent to (1) and $\mathsf{ports}(\mathsf{H})^c \cap (\mathsf{free}([\hat{M}]) \setminus S) = \emptyset$ (3). Now (3) $\Leftrightarrow \mathsf{ports}(\mathsf{H})^c \cap \mathsf{free}([\hat{M}]) \subseteq S$. It remains to be shown that $\mathsf{ports}(\mathsf{H})^c \cap \mathsf{inner}([\hat{M}]) = \emptyset$. This is equivalent to $\mathsf{ports}(\mathsf{H}) \cap \mathsf{inner}([\hat{M}]) = \emptyset$. Now $\mathsf{ports}([\hat{M}])$ only contains additional buffer ports and clock in-ports compared with $\mathsf{ports}(\hat{M})$. Hence (1) even implies $\mathsf{ports}(\mathsf{H}) \cap \mathsf{ports}([\hat{M}]) = \emptyset$. ■

The following lemma shows that the ports of a structure that users are intended to use, i.e., the complement of the service ports, are not at the same time forbidden. Moreover, it shows that a non-suitable configuration can be transformed into a suitable one via port renaming such that this renaming does not affect the view of the user. This means that the restriction to suitable configurations in the definition of reactive simulatability is without loss of generality.

Lemma 5.2 (Valid Mappings and Suitable Configurations) *Let* $(\hat{M}_1, S)$ *and* $(\hat{M}_2, S)$ *be structures.*

a) *Then* $S^c \cap \mathsf{forb}(\hat{M}_1, S) = \emptyset$.

b) *For every* $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}(\hat{M}_1, S) \setminus \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$, *there is a configuration* $conf_{\mathsf{f},1} = (\hat{M}_1, S, \mathsf{H}_\mathsf{f}, \mathsf{A}_{\mathsf{f},1}) \in \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$ *such that* $view_{conf_{\mathsf{f},1}}(\mathsf{H}_\mathsf{f}) = view_{conf_1}(\mathsf{H})$.

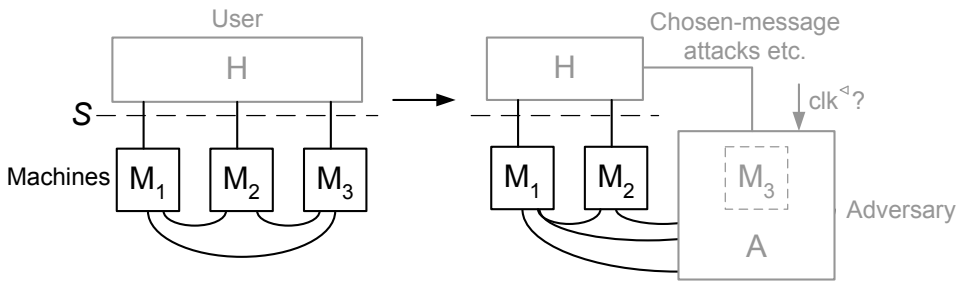

Figure 5: Derivation of one possible structure from an intended structure. The structures are the black parts.

□

Proof. For Part a) recall that $\mathsf{forb}(\hat{M}, S) = \mathsf{ports}(\hat{M}_1) \cup (\mathsf{free}([\hat{M}_1]) \setminus S)^c$. The part $S^c \cap (\mathsf{free}([\hat{M}_1]) \setminus S)^c = \emptyset$ is clear, and $S^c \cap \mathsf{ports}(\hat{M}_1) = \emptyset$ follows from $S \subseteq \mathsf{free}([\hat{M}_1])$.

For Part b), we want to construct $\mathsf{H_f}$ by giving each port $p = \mathsf{n}ld \in \mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(\hat{M}_2, S)$ a new name. Since runs and views do not depend on port names, cf. Definition 3.5, they remain the same if we consistently rename all other ports q (at most five) with $\mathsf{name}(q) = \mathsf{n}$. The new collection is a configuration $(\hat{M}_1, S, \mathsf{H_f}, \mathsf{A_{f,1}})$ if no renamed port belongs to $\hat{M}_1$. Assume that $q = \mathsf{n}l'd' \in \mathsf{ports}(\hat{M}_1)$ is such a renamed port, then $\tilde{\mathsf{n}} \in [\hat{M}_1]$ and hence $p^c \in \mathsf{ports}([\hat{M}_1])$. Now Lemma 5.1 implies $p^c \in S$, hence Part a) applied to the structure $(\hat{M}_2, S)$ implies $p \notin \mathsf{forb}(\hat{M}_2, S)$, in contradiction to the original condition on p. ■

6 Special Cases for Cryptographic Purposes

In the presence of adversaries, the structure of correct machines running may not be the *intended structure* that the designer originally planned. For instance, some machines might have been corrupted; hence they are missing from the actual structure and the adversary took over their connections. We model this by defining a *system* as a set of possible actual structures. A system is typically derived automatically from an intended structure and a *trust model*. We define this for static and adaptive adversaries, arbitrary access structures limiting the corruption capabilities of an adversary, and different channel types. While one typically considers all basic channels insecure in a security protocol, secure or authentic channels are useful to model initialization phases, e.g., the assumption that a public-key infrastructure exists. In contrast to some formal methods which immediately abstract from cryptography, we cannot represent this by a fixed initial key set because we need probabilities over the key generation for security; moreover, this would not allow a polynomial number of keys to be chosen reactively.

Cryptographic systems come in two flavors, depending on how the adversary takes control over certain machines. If malicious machines are malicious from the beginning, we call the system a *static standard cryptographic system*. If the adversary may corrupt machines during the protocol executions, depending on the knowledge that he has already collected, we speak of an *adaptive standard cryptographic system*.

6.1 Trust Models and Intended Structures

We start with the definition of trust models for a structure. Trust models consist of two parts: an *access structure* and a *channel model*. Access structures will later be used to denote the

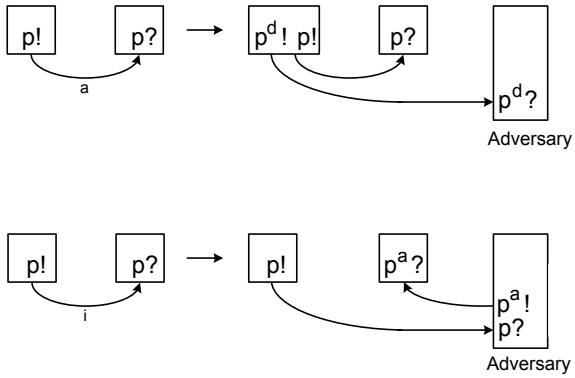

Figure 6: Channel modifications for authenticated and insecure channels.

possible sets of *correct* machines in an intended structure. Access structures have to be closed under insertion, i.e., with every such set, every larger set is contained as well.

Definition 6.1 *(Access Structure) Let A be an arbitrary set. Then $\mathcal{ACC} \subseteq \mathcal{P}(A)$ is called an* access structure *for A iff $(B \in \mathcal{ACC} \wedge C \in \mathcal{P}(A) \wedge B \subseteq C) \Rightarrow C \in \mathcal{ACC}$ for all sets B and C.* $\diamond$

Typical examples of access structures are threshold structures $\mathcal{ACC}_{t,n} := \{\mathcal{H} \subseteq A \mid |\mathcal{H}| \geq t\}$ with $t \leq n$.

A channel model for a structure classifies each internal high-level connection as *secure* (private and authentic), *authenticated* (only authentic), or *insecure* (neither private nor authentic), represented as elements of the set $\{\mathsf{s}, \mathsf{a}, \mathsf{i}\}$. What this means will become clear in Section 6.2.

Definition 6.2 *(Channel Model) A* channel model *for a structure $(\hat{M}, S)$ is a mapping $\chi \colon \mathsf{Gr}(\hat{M}) \to \{\mathsf{s}, \mathsf{a}, \mathsf{i}\}$.* $\diamond$

Definition 6.3 *(Trust Model) A* trust model *for a structure $(\hat{M}, S)$ is a pair $(\mathcal{ACC}, \chi)$ where $\mathcal{ACC}$ is an access structure for $\hat{M}$, and χ is a channel model for $(\hat{M}, S)$.* $\diamond$

We proceed with the definition of intended structures, i.e., structures a designer of a security protocol would typically design. An intended structure is a structure that is benign in the sense that it does not offer any free simple ports for the adversary. Moreover, we demand that a machine of a structure may only schedule those connections for which it owns the corresponding input or output port, i.e., it does not intend to schedule users and adversaries. We distinguish three different kinds of intended structures, depending on how the channels between the system and the user are clocked. We call the structure *localized* if every output is clocked by the outputting machine itself. Structures of this kind are typically used as local submachines that can be clocked by the overall protocol and then immediately deliver a result. If the adversary clocks the communication between the structure and the user, we call the structure *stand-alone*. Finally, if the user and the system have to fetch the outputs of the other, we call the structure *fetching*.

Remark 6.1. We could as well distinguish channels from the user to the system and vice versa, e.g., to define that users schedule their outputs and fetch their inputs. This would give nine different combinations. Modifying the upcoming definition of intended structures in such a way is trivial. $\circ$

The derivation of the remaining structures based on the intended structure will rely on modifying the connections in an adequate way, e.g., by duplicating output ports that output on authentic channels so that one output port is connected as usual and the duplicated port is connected to the adversary. Hence derived structures need an extended set of possible port names. Moreover, we will need a distinguished state and distinguished ports to model adaptive corruptions of machines. Technically, we therefore parameterize intended structures by an additional alphabet $\Gamma \subsetneq \Sigma$ with $|\Gamma| = |\Sigma| - 2$ and a state s_{corr}, and we restrict port names in the intended structure to Γ^+ and possible states to $\Sigma^* \setminus \{s_{\mathsf{corr}}\}$.

Definition 6.4 *(Intended Structure) A structure $(\hat{M}^*, S^*)$ is called an* intended structure *for $s_{\mathsf{corr}} \in \Sigma^*$ and $\Gamma \subset \Sigma$ with $|\Gamma| = |\Sigma| - 2$ iff*

- *all $\mathsf{M} \in \hat{M}^*$ are simple, $\mathsf{name}(p) \in \Gamma^+$ for all $p \in \mathsf{ports}(\hat{M}^*)$, and $s_{\mathsf{corr}} \notin States_{\mathsf{M}}$ for all $\mathsf{M} \in \hat{M}^*$,*
- *for all $\mathsf{M} \in \hat{M}^*$: $(\mathsf{n}^{\triangleleft}! \in \mathsf{ports}(\mathsf{M}) \Rightarrow (\mathsf{n}? \in \mathsf{ports}(\mathsf{M})) \vee (\mathsf{n}! \in \mathsf{ports}(\mathsf{M})))$, and*
- *it has the following properties. Let $S' := \{p \in \mathsf{free}([\hat{M}^*]) \mid \mathsf{label}(p^c) = \epsilon\}$.*
  - *The structure is called* localized *iff*

    $$S^* = S' \cup \{\mathsf{n}^{\triangleleft}? \mid \mathsf{n}! \in \mathsf{free}([\hat{M}^*])^c\}.$$

    and the following condition on the port set of $\hat{M}^*$ holds:

    $$\mathsf{n}? \in \mathsf{free}([\hat{M}^*])^c \Rightarrow \mathsf{n}^{\triangleleft}! \in \mathsf{ports}(\hat{M}^*) \quad \# \ \hat{M}^* \text{ schedules its outputs to the user}$$

  - *The structure is called* stand-alone *iff $S^* = S'$ and*

    $$(\mathsf{n}, \epsilon, d) \in \mathsf{free}([\hat{M}^*])^c \Rightarrow \mathsf{n}^{\triangleleft}! \notin \mathsf{ports}(\hat{M}^*) \quad \# \ \hat{M}^* \text{ does not schedule any connection between the user and } \hat{M}^*$$

  - *The structure is called* fetching *iff*

    $$S^* = S' \cup \{\mathsf{n}^{\triangleleft}? \mid \mathsf{n}? \in \mathsf{free}([\hat{M}^*])^c\}$$

    and the following condition on the port set of $\hat{M}^*$ holds:

    $$\mathsf{n}! \in \mathsf{free}([\hat{M}^*])^c \Rightarrow \mathsf{n}^{\triangleleft}! \in \mathsf{ports}(\hat{M}^*) \quad \# \ \hat{M}^* \text{ schedules the inputs from the user}$$

$\diamond$

6.2 Standard Static Cryptographic Systems

Standard static cryptographic systems are derived from an intended structure and a trust model as follows. Each system contains one structure for each element of the considered access structure, i.e., for each set $\mathcal{H}$ of potential correct machines.

The channel model is taken into account as follows. Intuitively, we change the connections such that the adversary receives messages sent on authenticated and insecure channels, and we enable him to arbitrarily modify messages sent on insecure channels. This is modeled as depicted in Figure 6 for authenticated and insecure channels; secure channels remain unchanged.

Authenticated channels are modeled by additional out-ports $\mathsf{p}^{\mathsf{d}}!$ where outputs at $\mathsf{p}!$ are duplicated. The port $\mathsf{p}^{\mathsf{d}}!$ remains free; hence the adversary can connect to it. Insecure channels are modeled by renaming the input port. This breaks the existing connection and places the adversary in between.

Moreover, if $\mathsf{p}?$ belongs to a correct machine and $\mathsf{p}!$ does not, we also rename $\mathsf{p}?$ into $\mathsf{p}^{\mathsf{a}}?$ so that all inputs from the adversary have superscript a. By applying these changes to a machine M, we get a modified machine $\mathsf{M}_{\mathcal{H},\chi}$.

Formally, let $\{\mathsf{a},\mathsf{d}\} := \Sigma \setminus \Gamma$. Then we define two mappings φ^{a} and φ^{d} that assign each port $p = (\mathsf{n}, l, d)$ with $\mathsf{n} \in \Gamma^+$ ports (an, l, d) and (dn, l, d) respectively. We write p^{a} and p^{d} instead of $\varphi^{\mathsf{a}}(p)$ and $\varphi^{\mathsf{d}}(p)$.

We now define the derivation of static cryptographic systems from a given intended structure and a trust model rigorously. Similar to the definition of runs, the semi-formal description given above is sufficient to understand our subsequent results, so the following technical definition can be skipped at first reading.

Definition 6.5 *(Derivation of Standard Static Cryptographic Systems) Let $(\hat{M}^*, S^*)$ be an intended structure for a state s_{corr} and a set Γ, and let $(\mathcal{ACC}, \chi)$ be a trust model for $(\hat{M}^*, S^*)$. Then the corresponding* cryptographic system with static adversary

$$Sys = StanStat((\hat{M}^*, S^*), (\mathcal{ACC}, \chi))$$

is $Sys := \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}$ where for all $\mathcal{H} \in \mathcal{ACC}$:

- $S_{\mathcal{H}} := S^* \cap \mathsf{free}([\mathcal{H}])$.
- $\hat{M}_{\mathcal{H}} := \{\mathsf{M}_{\mathcal{H},\chi} | \mathsf{M} \in \mathcal{H}\}$, *where*

$$\mathsf{M}_{\mathcal{H},\chi} = (\mathsf{name}_{\mathsf{M}}, Ports_{\mathsf{M}_{\mathcal{H},\chi}}, States_{\mathsf{M}}, \delta_{\mathsf{M}_{\mathcal{H},\chi}}, l_{\mathsf{M}}, Ini_{\mathsf{M}}, Fin_{\mathsf{M}})$$

  is defined as follows:

  - *The sequence $Ports_{\mathsf{M}_{\mathcal{H},\chi}}$ is derived by the following algorithm.*

    $Ports_{\mathsf{M}_{\mathcal{H},\chi}} := ()$.
    for $p \in Ports_{\mathsf{M}}$ *(in the given order)* ***do***
      if $c := \{p, p^C\} \in \mathsf{Gr}(\mathcal{H}) \wedge \chi(c) = \mathsf{a} \wedge \mathsf{dir}(p) = !$ ***then***
        $Ports_{\mathsf{M}_{\mathcal{H},\chi}} := Ports_{\mathsf{M}_{\mathcal{H},\chi}} \circ (p, p^{\mathsf{d}})$
      else if $c := \{p, p^C\} \in \mathsf{Gr}(\mathcal{H}) \wedge \chi(c) = \mathsf{i} \wedge \mathsf{dir}(p) = ?$ ***then***
        $Ports_{\mathsf{M}_{\mathcal{H},\chi}} := Ports_{\mathsf{M}_{\mathcal{H},\chi}} \circ (p^{\mathsf{a}})$
      else if $c := \{p, p^C\} \notin \mathsf{Gr}(\mathcal{H}) \wedge \mathsf{dir}(p) = ?$ ***then***
        $Ports_{\mathsf{M}_{\mathcal{H},\chi}} := Ports_{\mathsf{M}_{\mathcal{H},\chi}} \circ (p^{\mathsf{a}})$
      else
        $Ports_{\mathsf{M}_{\mathcal{H},\chi}} := Ports_{\mathsf{M}_{\mathcal{H},\chi}} \circ (p)$
      end if
    end for

  - *Let $s, s' \in States_{\mathsf{M}}$, $I \in (\Sigma^*)^{|\mathsf{in}(\mathsf{ports}(\mathsf{M}))|}$, $O^1 \in (\Sigma^*)^{|\mathsf{out}(\mathsf{ports}(\mathsf{M}))|}$ and $O^2 \in (\Sigma^*)^{|\mathsf{out}(\mathsf{ports}(\mathsf{M}_{\mathcal{H},\chi}))|}$. Then $\mathsf{Pr}(\delta_{\mathsf{M}_{\mathcal{H},\chi}}(s, I) = (s', O^2)) := \mathsf{Pr}(\delta_{\mathsf{M}}(s, I) = (s', O^1))$ if O^2 is derived from O^1 by the following algorithm, and zero otherwise.*[2]

[2]Note that $|\mathsf{in}(\mathsf{ports}(\mathsf{M}))| = |\mathsf{in}(\mathsf{ports}(\mathsf{M}_{\mathcal{H},\chi}))|$ by definition; hence I is also a valid input tuple for the machine $\mathsf{M}_{\mathcal{H},\chi}$.

$i := 1$
for $j := 1, \ldots, |\mathsf{out}(\mathsf{ports}(\mathsf{M}_{\mathcal{H},\chi}))|$ ***do***
  if $(\mathsf{out}(Ports_{\mathsf{M}_{\mathcal{H},\chi}}))[j] = p^{\mathsf{d}}$ *for a port* p ***then***
    $O^2[j] := O^2[j-1]$
  else
    $O^2[j] := O^1[i]$
    $i := i + 1$
  end if
end for

$\diamond$

6.3 Standard Cryptographic Systems with Adaptive Adversaries

Standard static cryptographic systems as defined in the previous section are based on the intuition that corrupted machines are corrupted right from the start, e.g., they belong to untrusted owners. In *adaptive* (or *dynamic*) adversary models the set of corrupted machines can increase over time, e.g., because there is a "master adversary" who has to hack into machines in order to corrupt them [31, 38]. Adaptive adversary models are more powerful than static ones, i.e., there are examples of systems secure against static adversaries that are insecure against adaptive adversaries who can corrupt the same number of machines [38].

For a given intended structure and a channel model, the corresponding *cryptographic system with adaptive adversary* has only one structure. Similar to the derivation of static cryptographic systems from an intended structure, we define the derivation of a machine $\mathsf{M}_{\mathsf{corr},\chi}$ from each machine M. This derivation is used to grant the adversary the possibility to corrupt machines during the global execution. This is modeled by giving $\mathsf{M}_{\mathsf{corr},\chi}$ a *corruption port* $\mathsf{corrupt}_{\mathsf{M}_{\mathsf{corr},\chi}}?$, which is used for corruption requests, and two new ports $\mathsf{cor\_out}_{\mathsf{M}_{\mathsf{corr},\chi}}!$, $\mathsf{cor\_in}_{\mathsf{M}_{\mathsf{corr},\chi}}?$ for communication with the adversary after corruption. We assume that these ports must neither occur in an intended structure nor after port renaming as defined for the static case; this can be achieved by encoding the names of these ports into $\Sigma^+ \setminus (\{\epsilon, \mathsf{a}, \mathsf{d}\} \circ \Gamma^+)$ where $\Sigma = \Gamma \cup \{\mathsf{a}, \mathsf{d}\}$. The corruption port must connect to the service ports. Upon a non-empty input at the corruption port, $\mathsf{M}_{\mathsf{corr}}$ sends its current state to the adversary via $\mathsf{cor\_out}_{\mathsf{M}_{\mathsf{corr},\chi}}!$, and from now on acts *transparently*, i.e., every input $(I_1, \ldots, I_s)$ is translated into the output $\iota(I_1, \ldots, I_s)$ at $\mathsf{cor\_out}_{\mathsf{M}_{\mathsf{corr},\chi}}!$, and every input (b) at $\mathsf{cor\_in}_{\mathsf{M}_{\mathsf{corr},\chi}}?$ is first decomposed as $(O_1, \ldots, O_t) := \iota^{-1}(b)$ and then output at the respective output ports.

Definition 6.7 *(Derivation of Standard Adaptive Cryptographic Systems) Let* $(\hat{M}^*, S^*)$ *be an intended structure for a state* s_{corr} *and a set* Γ*, and let* χ *be a channel model* χ *for* $(\hat{M}^*, S^*)$*. Then the corresponding* cryptographic system with adaptive adversary

$$Sys = StanAdap((\hat{M}^*, S^*), \chi)$$

is $Sys := \{(\hat{M}, S)\}$ *where*

- $S := S^* \cup \{\mathsf{corrupt}_{\mathsf{M}}{}^{\leftrightarrow}? \mid \mathsf{M} \in \hat{M}^*\}$.
- $\hat{M} := \{\mathsf{M}_{\mathsf{corr},\chi} \mid \mathsf{M} \in \hat{M}^*\}$*, where* $\mathsf{M}_{\mathsf{corr},\chi}$ *is derived from* M *with* $\mathcal{H} = \hat{M}$ *as follows: Let* $\mathsf{M}_{\mathcal{H},\chi}$ *be the machine defined in the static case (Definition 6.5). Then*

  $$\mathsf{M}_{\mathsf{corr},\chi} = (\mathsf{name}_{\mathsf{M}}, Ports_{\mathsf{M}_{\mathsf{corr},\chi}}, States_{\mathsf{M}_{\mathsf{corr},\chi}}, \delta_{\mathsf{M}_{\mathsf{corr},\chi}}, l_{\mathsf{M}_{\mathsf{corr},\chi}}, Ini_{\mathsf{M}}, Fin_{\mathsf{M}})$$

  is defined as follows:

– $Ports_{\mathsf{M}_{\mathrm{corr},\chi}} := Ports_{\mathsf{M}_{\mathcal{H},\chi}} \circ (\mathsf{corrupt}_{\mathsf{M}_{\mathrm{corr},\chi}}?, \mathsf{cor\_out}_{\mathsf{M}_{\mathrm{corr},\chi}}!, \mathsf{cor\_in}_{\mathsf{M}_{\mathrm{corr},\chi}}?)$.
– $States_{\mathsf{M}_{\mathrm{corr},\chi}} := States_{\mathsf{M}} \cup \{s_{\mathsf{corr}}\}$.
– $l_{\mathsf{M}_{\mathrm{corr},\chi}}(s) := l_{\mathsf{M}}(s) \circ (1,0)$ *for* $s \in States_{\mathsf{M}}$, *and* $l_{\mathsf{M}_{\mathrm{corr},\chi}}(s_{\mathsf{corr}}) := (\infty,\ldots,\infty) \circ (0,\infty)$.
– *Let* $I =: I' \circ (a,b)$. *If* $(a,b) = (\epsilon,\epsilon)$ *and* $s \neq s_{\mathsf{corr}}$, *then* $\delta_{\mathsf{M}_{\mathrm{corr},\chi}}(s,I) := \delta_{\mathsf{M}_{\mathcal{H},\chi}}(s,I') \circ (\epsilon)$. *Otherwise* $\delta_{\mathsf{M}_{\mathrm{corr},\chi}}(s,I) := (s',O)$ *with* $O := O' \circ (o)$ *is defined as follows:*

if $a \neq \epsilon$ *and* $s \neq s_{\mathsf{corr}}$ ***then***
  $s' := s_{\mathsf{corr}}$, $o := s$, *and* $O' := (\epsilon,\ldots,\epsilon)$
else if $s = s_{\mathsf{corr}}$ ***then***
  $o := \iota(I)$
  if $b = \epsilon$ ***then***
    $O' = (\epsilon,\ldots,\epsilon)$
  else
    $(o_1,\ldots,o_t) := \iota^{-1}(b)$ *and* $t' := \mathsf{size}(\mathsf{out}(Ports_{\mathsf{M}_{\mathcal{H},\chi}}))$
    $O'[j] := o_j$ *for* $j = 1,\ldots,\mathsf{min}(t,t')$ *and* $O'[j] := \epsilon$ *for* $j = t+1,\ldots,t'$
  end if
end if

◇

Note that $\mathsf{M}_{\mathrm{corr},\chi}$ is not polynomial-time. Explicitly bounding the number of bits read in a corrupted state independent of the adversary does not capture our intuition of a machine that acts transparently for any adversary, and an explicit bound would severely limit the adversary's capabilities. The fundamental problem is that transparent machines are by definition not polynomial-time but only weakly polynomial-time.

Several extensions are possible: One may extend the corruption responses to two classes of storage, an erasable and a non-erasable one, e.g., to model the different vulnerability of session keys and long-term keys. This means to refine the state spaces of each machine as a Cartesian product. Inputs and outputs would be treated like erasable storage. One can also model non-binary corruption requests, e.g., stop requests and requests to corrupt different classes of storage. To model proactive systems [87], one needs repair requests in addition to corruption requests, and appropriate repair responses, e.g., returning to an initial state with only a certain class of storage still intact.

7 Summary

We have presented a rigorous model for secure reactive systems with cryptographic parts in asynchronous networks. Common types of cryptographic systems and their trust models were expressed as special cases of this model, in particular systems with static or adaptive adversaries and with different types of underlying channels. We have defined reactive simulatability as a notion of refinement that retains not only integrity properties but also confidentiality.

As design principles based on this model, we propose to keep specifications of cryptographic systems abstract, i.e., free of all implementation details and deterministic unless the desired functionality is probabilistic by nature, e.g., a common coin-flipping protocol. This allows the cryptographically verified abstractions to be used as building blocks for systems that can be subjected to formal verification. Suitable abstractions sometimes have to explicitly include tolerable imperfections of systems, e.g., leakage of the length of encrypted messages or abilities of the adversary to disrupt protocol runs. In subsequent work, we already showed multiple desired

properties of reactive simulatability, in particular composition and property-preservation theorems. We also proved several building blocks like secure message transmission and a Dolev-Yao-style cryptographic library with nested operations. Moreover, we demonstrated the applicability of formal methods over our model by small examples.

Acknowledgment

We thank *Anupam Datta, Martin Hirt, Dennis Hofheinz, Paul Karger, Ralf Küsters, John Mitchell, Jörn Müller-Quade, Phil Rogaway, Andre Scedrov, Matthias Schunter, Victor Shoup, Michael Steiner, Rainer Steinwandt, Dominique Unruh* and an anonymous reviewer for interesting discussions. In particular, joint work with Matthias on examples of synchronous reactive systems also influenced this asynchronous model, and so did joint work with Michael on liveness and adaptiveness. Dennis, Jörn, Rainer, and Dominique pointed out some subtle problems. Phil Rogaway provided enough motivation for making the model much more rigorous than it was before.

This work was partially supported by the European IST Project MAFTIA. However, it represents the view of the authors. The MAFTIA project was partially funded by the European Commission and the Swiss Federal Office for Education and Science (BBW).

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[2] M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.

[3] M. Abadi and M. R. Tuttle. A semantics for a logic of authentication. In *Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 201–216, 1991.

[4] M. Backes. *Cryptographically Sound Analysis of Security Protocols.* PhD thesis, Universität des Saarlandes, 2002. `http://www.zurich.ibm.com/~mbc/papers/PhDthesis.ps.gz`.

[5] M. Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In *Proc. 9th European Symposium on Research in Computer Security (ESORICS)*, volume 3193 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2004.

[6] M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Cryptographically sound security proofs for basic and public-key Kerberos. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS)*. Springer, 2006.

[7] M. Backes and M. Dürmuth. A cryptographically sound Dolev-Yao style security proof of an electronic payment system. In *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 78–93, 2005.

[8] M. Backes, M. Dürmuth, D. Hofheinz, and R. Küsters. Conditional reactive simulatability. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS)*, number 4189 in Lecture Notes in Computer Science, pages 424–443. Springer, 2006.

[9] M. Backes and D. Hofheinz. How to break and repair a universally composable signature functionality. In *Proc. 6th Information Security Conference (ISC)*, volume 3225 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2004. Extended version in IACR Cryptology ePrint Archive 2003/240, September 2003, `http://eprint.iacr.org/`.

[10] M. Backes, D. Hofheinz, J. Mueller-Quade, and D. Unruh. On fairness in simulatability-based cryptographic systems. In *Proc. 3rd ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 13–22, 2005.

[11] M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 675–686. Springer, 2003.

[12] M. Backes, C. Jacobi, and B. Pfitzmann. Deriving cryptographically sound implementations using composition and formally verified bisimulation. In *Proc. 11th Symposium on Formal Methods Europe (FME 2002)*, volume 2391 of *Lecture Notes in Computer Science*, pages 310–329. Springer, 2002.

[13] M. Backes and P. Laud. Computationally sound secrecy proofs by mechanized flow analysis. In *Proc. 13th ACM Conference on Computer and Communications Security*, 2006.

[14] M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 1–12, 2003. Full version in IACR Cryptology ePrint Archive 2003/121, Jun. 2003, `http://eprint.iacr.org/`.

[15] M. Backes and B. Pfitzmann. Intransitive non-interference for cryptographic purposes. In *Proc. 24th IEEE Symposium on Security & Privacy*, pages 140–152, 2003.

[16] M. Backes and B. Pfitzmann. Computational probabilistic non-interference. *International Journal of Information Security*, 3(1):42–60, 2004.

[17] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004.

[18] M. Backes and B. Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style XOR. In *Proc. 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 178–196. Springer, 2005.

[19] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on Dependable and Secure Computing*, 2(2):109–123, 2005.

[20] M. Backes and B. Pfitzmann. On the cryptographic key secrecy of the strengthened Yahalom protocol. In *21st IFIP TC-11 International Information Security Conference (SEC'2006)*, May 2006. To appear.

[21] M. Backes, B. Pfitzmann, and A. Scedrov. Key-dependent message security under active attacks – BRSIM/UC-soundness of symbolic encryption with key cycles. In *Proc. 20th IEEE Computer Security Foundations Workshop (CSFW)*, 2007.

[22] M. Backes, B. Pfitzmann, M. Steiner, and M. Waidner. Polynomial liveness. *Journal of Computer Security*, 12(3-4):589–618, 2004.

[23] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, `http://eprint.iacr.org/`.

[24] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2004.

[25] M. Backes, B. Pfitzmann, and M. Waidner. Low-level ideal signatures and general integrity idealization. In *Proc. 7th Information Security Conference (ISC)*, volume 3225 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2004.

[26] M. Backes, B. Pfitzmann, and M. Waidner. Reactively secure signature schemes. *International Journal of Information Security*, 4(4):242–252, 2005.

[27] M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. *International Journal of Information Security*, 4(3):135–154, 2005.

[28] M. Backes, B. Pfitzmann, and M. Waidner. Limits of the Reactive Simulatability/UC of Dolev-Yao models with hashes. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS)*, Lecture Notes in Computer Science. Springer, 2006. Preliminary version in IACR Cryptology ePrint Archive 2006/068, Feb. 2006, `http://eprint.iacr.org/`.

[29] B. Barak and A. Sahai. How to play almost any mental game over the net — concurrent composition via super-polynomial simulation. In *Proc. 46th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 543–552, 2005.

[30] D. Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

[31] D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Advances in Cryptology: EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 307–323. Springer, 1992.

[32] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 419–428, 1998.

[33] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology: CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1994.

[34] B. Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symposium on Security & Privacy*, pages 140–154, 2006.

[35] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[36] M. Burrows, M. Abadi, and R. Needham. A logic for authentication. Technical Report 39, SRC DIGITAL, 1990.

[37] R. Canetti. Studies in secure multiparty computation and applications. Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, June 1995. Revised March 1996.

[38] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.

[39] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version available at `http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps`.

[40] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67, `http://eprint.iacr.org/`.

[41] R. Canetti. Universally composable signatures, certification and authorization. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, 2004. Extended version in Cryptology ePrint Archive, Report 2003/239, `http://eprint.iacr.org/`.

[42] R. Canetti, L. Cheung, D. K. Kaynar, M. Liskov, N. A. Lynch, O. Pereira, and R. Segala. Time-bounded task-PIOAs: A framework for analyzing security protocols. In *Proc. 20th International Symposium on Distributed Computing (DISC)*, pages 238–253, 2006.

[43] R. Canetti and M. Fischlin. Universally composable commitments. In *Advances in Cryptology: CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

[44] R. Canetti and S. Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *Advances in Cryptology: EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 90–106. Springer, 1999.

[45] R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. 3rd Theory of Cryptography Conference (TCC)*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2006.

[46] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels (extended abstract). In *Advances in Cryptology: EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002. Extended version in IACR Cryptology ePrint Archive 2002/059, `http://eprint.iacr.org/`.

[47] R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *Advances in Cryptology: EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86. Springer, 2003.

[48] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Proc. 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 494–503, 2002.

[49] A. Datta, A. Derek, J. Mitchell, A. Ramanathan, and A. Scedrov. Games and the impossibility of realizable ideal functionality. In *Proc. 3rd Theory of Cryptography Conference (TCC)*. Springer, 2006.

[50] A. Datta, A. Derek, J. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2005.

[51] A. Datta, R. Küsters, J. C. Mitchell, and A. Ramanathan. On the relationships between notions of simulation-based security. In *Proc. 2nd Theory of Cryptography Conference (TCC)*, volume 3378 of *Lecture Notes in Computer Science*, pages 476–494. Springer, 2005.

[52] Y. Desmedt and K. 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. Springer, 2000.

[53] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[54] J. Garay, P. MacKenzie, and K. Yang. Efficient and universally composable committed oblivious transfer and applications. In *Proc. 1st Theory of Cryptography Conference (TCC)*, pages 171–190, 2004.

[55] R. Gennaro and S. Micali. Verifiable secret sharing as secure computation. In *Advances in Cryptology: EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 1995.

[56] O. Goldreich. Secure multi-party computation. Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, June 1998, revised Version 1.4 October 2002. `http://www.wisdom.weizmann.ac.il/users/oded/pp.htm`.

[57] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[58] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology: CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 1990.

[59] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[60] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–207, 1989.

[61] M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.

[62] C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science, Prentice Hall, Hemel Hempstead, 1985.

[63] D. Hofheinz, J. Müller-Quade, and R. Steinwandt. Initiator-resilient universally composable key exchange. In *Proc. 8th European Symposium on Research in Computer Security (ESORICS)*, volume 2808 of *Lecture Notes in Computer Science*, pages 61–84. Springer, 2003.

[64] D. Hofheinz, J. Müller-Quade, and D. Unruh. Polynomial runtime in simulatability definitions. In *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 156–169, 2005.

[65] D. Hofheinz and D. Unruh. Comparing two notions of simulatability. In *Proc. 2nd Theory of Cryptography Conference (TCC)*, volume 3378 of *Lecture Notes in Computer Science*, pages 86–103. Springer, 2005.

[66] D. Hofheinz and D. Unruh. On the notion of statistical security in simulatability definitions. In *Proc. 8th Information Security Conference (ISC)*, volume 3650 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2005.

[67] D. Hofheinz and D. Unruh. Simulatable security and polynomially bounded concurrent composability. In *Proc. 27th IEEE Symposium on Security & Privacy*, pages 169–183, 2006.

[68] R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–381, 2003.

[69] R. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, 1989.

[70] R. Küsters. Simulation-based security with inexhaustible interactive turing machines. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 309–320, 2006.

[71] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[72] P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.

[73] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[74] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security analysis. In *Proc. 8th Symposium on Formal Methods Europe (FME 1999)*, volume 1708 of *Lecture Notes in Computer Science*, pages 776–793. Springer, 1999.

[75] Y. Lindell. General composition and universal composability in secure multi-party computation. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 394–403, 2003.

[76] Y. Lindell, A. Lysyanskaya, and T. Rabin. On the composition of authenticated byzantine agreement. In *Proc. 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 514–523, 2002.

[77] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[78] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Francisco, 1996.

[79] N. Lynch. I/O automaton models and proofs for shared-key communication systems. In *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW)*, pages 14–29, 1999.

[80] C. Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.

[81] S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1991.

[82] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.

[83] J. K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141, 1984.

[84] J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 725–733, 1998.

[85] J. Mitchell, M. Mitchell, A. Scedrov, and V. Teague. A probabilistic polynominal-time process calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.

[86] J. Neveu. *Mathematical Foundations of the Calculus of Probability*. Holden-Day, 1965.

[87] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 51–59, 1991.

[88] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.

[89] B. Pfitzmann and M. Waidner. How to break and repair a "provably secure" untraceable payment system. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 338–350. Springer, 1992.

[90] B. Pfitzmann and M. Waidner. A general framework for formal notions of "secure" systems. Research Report 11/94, University of Hildesheim, Apr. 1994. `http://www.semper.org/sirene/lit/abstr94.html#PfWa_94`.

[91] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000. Extended version (with Matthias Schunter) IBM Research Report RZ 3206, May 2000, `http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz`.

[92] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001. Extended version of the model (with Michael Backes) IACR Cryptology ePrint Archive 2004/082, `http://eprint.iacr.org/`.

[93] M. Prabhakaran and A. Sahai. New notions of security: Achieving universal composability without trusted setup. In *Proc. 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 242–251, 2004.

[94] R. Segala and N. Lynch. Probabilistic simulation for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[95] C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner. Cryptographically sound theorem proving. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 153–166, 2006.

[96] M. Steiner. *Secure Group Key Agreement*. PhD thesis, Universität des Saarlandes, 2002. `http://www.semper.org/sirene/publ/Stei_02.thesis-final.pdf`.

[97] F. J. Thayer Fabrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. 19th IEEE Symposium on Security & Privacy*, pages 160–171, 1998.

[98] S.-H. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176(1–2):1–38, 1997.

[99] A. C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

[100] A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.