# A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission*

Birgit Pfitzmann
Universität des Saarlandes
Saarbrücken, Germany
pfitzmann@cs.uni-sb.de

Michael Waidner
IBM Research Division
Rüschlikon, Switzerland
wmi@zurich.ibm.com

December 19, 2000

**Abstract**

We present the first rigorous model for secure reactive systems in asynchronous networks with a sound cryptographic semantics, supporting abstract specifications and the composition of secure systems. This enables modular proofs of security, which is essential in bridging the gap between the rigorous proof techniques of cryptography and tool-supported formal proof techniques.

The model follows the general simulatability approach of modern cryptography. A variety of network structures and trust models can be described, such as static and adaptive adversaries.

As an example of our specification methodology we provide the first abstract and complete specification for Secure Message Transmission, improving on recent results by Lynch, and verify one concrete implementation. Our proof is based on a general theorem on the security of encryption in a reactive multi-user setting, generalizing a recent result by Bellare et.al.

## 1 Introduction

In the early days of security research, cryptographic protocols were designed using a simple iterative process: someone proposed a protocol, someone else found an attack, an improved version was proposed, and so on, until no further attacks were found. Today it is commonly accepted that this approach gives no security guarantee. Too many seemingly simple and secure protocols have been found flawed over the years. Moreover, problems like $n$-party key agreement, fair contract signing, anonymous communication, electronic auctions or payments are just too complex for this approach. Secure protocols—or more generally, secure reactive systems—need a proof of security before being acceptable.

Both the cryptographic and the formal-methods communities are working on such proofs. The former aims at complete and mathematically rigorous proofs, while the latter aims at proofs in some formal proof system that can be automatically verified or even generated. Unfortunately, current formal methods in security cannot be applied directly to cryptographic proofs. Instead they abstract from most cryptographic details, and therefore there is no guarantee that a formally proven protocol is actually secure if implemented with a cryptographically secure primitive [1, 26].

One of our goals is to link both approaches to get the best overall results: proofs that allow abstraction and the use of formal methods, but retain a sound cryptographic semantics. Thus we provide a model that allows us to split reactive systems into two layers: The lower layer is a cryptographic system whose security can be rigorously proven using standard cryptographic arguments. To the upper layer it provides an abstract (and typically deterministic) service that hides all cryptographic details. Relative to this abstract service one can verify the upper layer using existing formal methods. Since our model

*Also published as IBM Research Report RZ 3304 (#93350) 12/11/2000, IBM Research Division, Zurich, December 2000

allows secure composition (as shown in Th. 4.1) one can conclude that the overall system is secure if the formally verified upper layer is put on top of a cryptographically verified lower layer ([26] provides more motivation for this approach).

In the following, we carry out this approach specifically for asynchronous reactive systems. Reactive means that the system interacts with its user many times (e.g., multiple subprotocol executions). Essentially, we describe a system by sets of asynchronously communicating probabilistic state machines, connected via buffered channels. Honest users and the adversary are explicitly represented by two arbitrary machines, $\mathsf{H}$ and $\mathsf{A}$, which can interact arbitrarily. A reactive system, $Sys_0$, is considered *at least as secure as* another system, $Sys_1$, written as $Sys_0 \geq Sys_1$, if whatever any adversary $\mathsf{A}_0$ can do to any honest user $\mathsf{H}$ in $Sys_0$, some adversary $\mathsf{A}_1$ can do to the same $\mathsf{H}$ in $Sys_1$ essentially with the same probability. System $Sys_0$ is often a real system using concrete cryptographic primitives, while $Sys_1$ is an ideal system, i.e., a specification, that does not depend on any specific cryptographic implementation details and is not realistic (e.g., one trusted machine), but secure by construction.

The mechanics of our model are defined in Section 2. Section 3 shows how to represent typical trust models (or adversary structures), such as static threshold models and adaptive adversaries, with secure, authenticated or insecure channels. In Section 4 we show that secure systems can be composed, i.e., our model supports modular security proofs as outlined above.

In Section 5 we study Secure Message Transmission as an example. We follow two main design principles:

*1. The ideal system should provide abstract interfaces, hiding all cryptographic details.* This keeps the specification independent of the implementation, which is desirable when higher-layer protocols are based on the service. For instance, in order to send messages secretly from one user to another there is no need to ask the user to input cryptographic keys or to output ciphertexts to him; those can be generated, exchanged and processed completely within the system.[1]

*2. The ideal system needs to explicitly specify all tolerable imperfections.* In order to improve efficiency one often accepts certain imperfections. For instance, a typical practical implementation of secure message transmission only conceals the contents of messages, but does not hide who communicates with whom, which would be much more costly to implement. In a simulatability-based approach one has to include all such tolerable imperfections in the specification of an ideal system, and they should also be abstract.

The proof of the real system in Section 5 uses a theorem (Th. 5.2) that extends the security of public-key encryption in multi-user settings, which might be of independent interest. It captures what is often called a "standard hybrid argument" in cryptography, and generalizes a result from [4].

**Related Literature.** Several researchers pursue the goal of providing security proofs that allow the use of formal methods, but retain a sound cryptographic semantics: In [21, 22] the cryptographic security of specific systems is directly defined and verified using a formal language ($\pi$-calculus), but without providing abstractions (their specifications essentially comprise the actual protocols including all cryptographic details) and without tool support (as even the specifications involve ad-hoc notations, e.g., for generating random primes). [24] has quite a similar motivation to our paper. However, cryptographic systems are restricted to the usual equational specifications (following [12]) and the semantics is not probabilistic. Hence the abstraction from cryptography is no more faithful than in other papers on formal methods in security. Moreover, only passive adversaries are considered and only one class of users ("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 both these problems. In [1] it is shown that a slight variation of the standard Dolev-Yao abstraction [12] is cryptographically faithful specifically for symmetric encryption, but only under passive attacks.

Our security definitions follow the general simulatability approach of modern cryptography, which was first used in secure function evaluation [3, 15, 25, 33], and subsequently also for specific reactive problems (e.g., [5, 10, 13]) and for the construction of generic solutions for large classes of reactive problems [19, 14, 20] (usually yielding inefficient solutions and assuming that all parties take part in all

[1] In a simulatability-based definition it would not even be possible to have keys in the interface and to be implementation-independent because keys of different implementations are distinguishable.

subprotocols). General models for reactive systems have been proposed (after some earlier brief sketches, in particular in [19, 29, 8]) in [21, 22, 20, 27, 30]. The last three are synchronous, while the first two are in a somewhat simplified timing model with uniform choice among certain classes of unrelated possible events. Among the reactive models, the only composition theorem so far is in [30], i.e., we present the first asynchronous one in the current paper. Our model is based on [27, 30], except for the timing aspects. Those can be seen as extensions of [32, 7, 22], see Section 2.[2]

Several specifications for secure message transmissions have been proposed, as examples of general models. The specification in [21] is formal but specific for one concrete protocol and comprises all cryptographic details, i.e., it is not abstract and its intuitive correctness is relatively difficult to verify. Our concrete specification is quite close to that in [24], but we had to introduce the tolerable imperfections. Actually, the implementation in [24] has the same imperfections. They do not show up in the proof because the definition of "a system implements another one," used in place of our "as secure as" is weaker: Secrecy is defined as a property that the adversary cannot learn certain messages, but here the information leaked is not entire messages.

# 2 Reactive Systems in Asynchronous Networks

In this section, we present our model for secure reactive systems in an asynchronous network. Section 2.1 defines the general system model, i.e., machines and executions of collections of machines. Section 2.2 defines the specific system model, i.e., systems with users and adversaries. Section 2.3 defines simulatability, i.e., our basic notion of security. Section 2.4 presents some basic lemmas for this model.

Our machine model is probabilistic state-transition machines, similar to probabilistic I/O automata as sketched in [23] (more details in [32]). A distinguishing feature in our model of asynchronous executions is distributed scheduling.

The standard way to fix the order of events in an asynchronous system of probabilistic I/O automata is a probabilistic scheduler that has full information about the system [32]. The "standard" understanding in cryptology (closest to a rigorous definition in [7]) is that the adversary schedules everything, but only with realistic information. This corresponds to making a certain subclass of schedulers explicit for the model from [32]. However, if one splits up a machine into local submachines, or defines intermediate systems for the purposes of proof only, this may introduce many schedules that do not correspond to a schedule of the original system and therefore just complicate the proofs. (The proof in Section 5 is of this type.) Our solution to this is a distributed definition of scheduling which allows machines that have been scheduled to schedule certain (statically fixed) other machines themselves. This does not weaken the adversary's power in real systems, because our definition of standard cryptographic systems in Section 3.1 will not use this feature.

Similar problems with purely adversarial scheduling were already noted in [22]. They distinguish secure channels and schedule all those with uniform probability before adversary-chosen events. However, that introduces a certain amount of global synchrony. Furthermore, we do not require "local" scheduling for all secure channels; they may be blindly scheduled by the adversary (i.e., without even seeing if there are messages on the channel). For instance, this models cases where the adversary has a global influence on relative network speed.

## 2.1 General System Model

We now define a model of machines and executions of collections of machines. For the port notation, compare Figure 1. (The buffer in the figure is defined later

Let a finite alphabet $\Sigma$ be given, let $\Sigma^*$ denote the strings over it, $\epsilon$ the empty string, and $\Sigma^+ := \Sigma^* \setminus \{\epsilon\}$. We assume that $!, ?, ^{\leftrightarrow}, ^{\triangleleft} \not\in \Sigma$.

[2]Independently and concurrently, Ran Canetti developed a model for asynchronous reactive systems [9]. Roughly, it corresponds to standard cryptographic systems with adaptive adversaries (as described in Section 3.2, using the 2nd type of corruption responses), with polynomial-time users and adversaries, and authenticated channels only. Security is defined in terms of universal simulatability (see Def. 2.12). The model is less rigorously defined than the model presented here. The composition theorem of [9] allows to securely compose a polynomial number of systems, while our theorem applies to a constant number of systems only (which is sufficient for our applications).

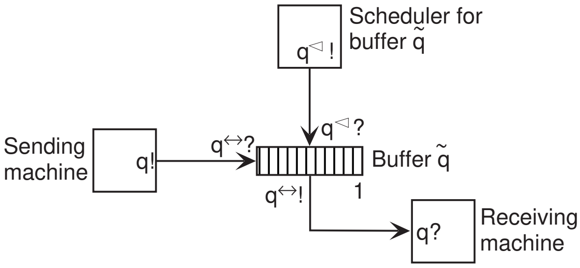


Figure 1: Naming convention for ports and buffers (cf. Definitions 2.1 and 2.3)

**Definition 2.1** *(Ports)* A port *port* $p$ is a triple $(n, l, d) \in \Sigma^+ \times \{\epsilon, ^{\leftrightarrow}, ^{\triangleleft}\} \times \{!, ?\}$. We call $\mathsf{name}(p) := n$ the name, $\mathsf{label}(p) := l$ the label, and $\mathsf{dir}(p) := d$ the direction of $p$. We can write the triples as concatenations without ambiguity.

We call a port $(n, l, d)$ an in-port or out-port iff $d = ?$ or $d = !$, respectively. We call it a simple port, buffer port or clock port iff $l = \epsilon, ^{\leftrightarrow}$, or $^{\triangleleft}$, respectively.[3]

For a set $P$ of ports let $\mathsf{out}(P) := \{p \in P \mid \mathsf{dir}(p) = !\}$ and $\mathsf{in}(P) := \{p \in P \mid \mathsf{dir}(p) = ?\}$. We use the same notation for sequences of ports (retaining the order).

By $p^c$, the (low-level) *complement* of a port $p$, we denote the port with which it connects according to Figure 1, i.e., $\mathsf{n}^{\triangleleft}!^c := \mathsf{n}^{\triangleleft}?$, $\mathsf{n}!^c := \mathsf{n}^{\leftrightarrow}?$, and $\mathsf{n}^{\leftrightarrow}?^c := \mathsf{n}?$ and vice versa. Accordingly we define the complement of a set of ports.

For simple ports, we also define $p^C$, the high-level complement, as the port connected to $p$ without counting the buffer, i.e., $\mathsf{n}?^C := \mathsf{n}!$ and vice versa. ◇

**Definition 2.2** *(Machines and Schedulers)* A *machine* is a tuple

$$\mathsf{M} = (name_{\mathsf{M}}, Ports_{\mathsf{M}}, \delta_{\mathsf{M}}, States_{\mathsf{M}}, Ini_{\mathsf{M}}, Fin_{\mathsf{M}})$$

of a name $name_{\mathsf{M}} \in \Sigma^+$, a finite sequence $Ports_{\mathsf{M}}$ of ports, a probabilistic state-transition function $\delta_{\mathsf{M}}$, a set $States_{\mathsf{M}} \subset \Sigma^*$ of states, and sets $Ini_{\mathsf{M}}, Fin_{\mathsf{M}} \subset States_{\mathsf{M}}$ of initial and final states.[4] The inputs are tuples $I = (I_i)_{i=1,\ldots,|\mathsf{in}(Ports_{\mathsf{M}})|}$, where $I_i \in \Sigma^*$ is the input for the $i$-th in-port. Analogously, the outputs are tuples $O = (O_i)_{i=1,\ldots,|\mathsf{out}(Ports_{\mathsf{M}})|}$. The empty word, $\epsilon$, denotes "no in- or output".[5]

$\delta_{\mathsf{M}}$ maps each pair $(s, I)$ of a state and an input to a finite distribution over pairs $(s', O)$. If $s \in Fin_{\mathsf{M}}$ or $I = (\epsilon, \ldots, \epsilon)$, then deterministically $\delta_{\mathsf{M}}(s, I) = (s, (\epsilon, \ldots, \epsilon))$.

Let $\mathsf{ports}(\mathsf{M})$ denote the set of ports in $Ports_{\mathsf{M}}$, and for a set $M$ of machines, let $\mathsf{ports}(M) = \bigcup_{\mathsf{M} \in M} Ports_{\mathsf{M}}$. [6]A machine $\mathsf{M}$ is *simple* if it has only simple ports and clock out-ports. A machine $\mathsf{M}$ is called *master scheduler* if it has only simple ports and clock out-ports and the special *master-clock* in-port $\mathsf{clk}^{\triangleleft}?$.

For computational aspects, a machine is regarded as implemented by a probabilistic interactive Turing machine [18], where each port is a communication tape. Its complexity is measured in terms of the length of the initial state, represented as initial worktape content (often a security parameter). It is not required to read the entire (new) input on any tape in any step. ◇

**Definition 2.3** *(Buffers)* For each name $q \in \Sigma^+$ we define a specific machine $\widetilde{q}$, called a *buffer:* $\widetilde{q}$ has three ports, $q^{\triangleleft}?$, $q^{\leftrightarrow}?$, $q^{\leftrightarrow}!$ (clock, in, and out) (see Figure 1). The internal state of $\widetilde{q}$ is a queue over $\Sigma^+$ with random access. Initially it is empty, and the set of final states is empty.

For each state transition, if $q^{\leftrightarrow}?$ carries a non-empty input then $\delta_{\widetilde{q}}$ appends this input to the queue. If $q^{\triangleleft}?$ carries a non-empty input, it is interpreted as a number $i \in \mathbb{N}$ and the $i$-th element is retrieved

[3]Double adjectives are then clear, e.g., a clock out-port is a port $\mathsf{n}^{\triangleleft}!$ with $\mathsf{n} \in \Sigma^+$.

[4]We often use $\mathsf{M}$ also as the name $name_{\mathsf{M}}$ of $\mathsf{M}$.

[5]The definition of $\delta_{\mathsf{M}}$ is independent of the port names; it only depends on the number of ports of each direction. Below, we also define the view of a machine independent of the port names. This guarantees that we can later rename ports in some proofs without changing the views.

[6]We use different font shapes to distinguish between machines and sets of machines: $\mathsf{M}$ is a machine, $M$ is a set.

(where 1 indicates the oldest one), removed from the queue, and output at $q^{\leftrightarrow}!$. (This might be the element just appended.) If there are fewer elements, the result is $\epsilon$.[7]

◇

**Definition 2.4** *(Collections)* A *collection* $C$ is a finite set of machines with pairwise different machine names, disjoint sets of ports, and where all machines are simple or master schedulers or buffers. It is called polynomial-time if all its machines (except for the buffers) are.

Each set of low-level complementary ports $\{p, p^c\} \subseteq \mathsf{ports}(C)$ is called a *low-level connection,* and the set of them the *low-level connection graph* $\mathsf{gr}(C)$. By $\mathsf{free}(C)$ we denote the *free* ports in this graph, i.e., $\mathsf{ports}(C) \setminus \mathsf{ports}(C)^c$. A set of high-level complementary simple ports $\{p, p^C\} \subseteq \mathsf{ports}(C)$ is called a *high-level connection,* and the set of them the *high-level connection graph* $\mathsf{Gr}(C)$.

A collection is *closed* if $\mathsf{free}(C) = \{\mathsf{clk}^{\triangleleft}?\}$.[8]

The *completion* $[C]$ of a collection $C$ is the smallest collection that comprises $C$ and the corresponding buffer for each simple or clock out-port $q \in \mathsf{ports}(C)$.

If $\tilde{\mathsf{q}}, \mathsf{M} \in C$ and $\mathsf{q}^{\triangleleft}! \in \mathsf{ports}(\mathsf{M})$ then we call $\mathsf{M}$ the scheduler for buffer $\tilde{\mathsf{q}}$ (in $C$). ◇

Now we define the probability space of runs (or "executions" or "traces") of a closed collection.

**Definition 2.5** *(Runs)* Given a closed collection $C$ with master scheduler $\mathsf{X}$ and a tuple $ini \in Ini := \times_{\mathsf{M} \in C} Ini_{\mathsf{M}}$ of initial states, the probability space of *runs* is defined as follows:

Each run is a sequence of *steps* inductively defined by the following algorithm. The algorithm uses a variable *curr_sched* over names of machines; initially *curr_sched* $:= \mathsf{X}$. It also treats each port like a variable over $\Sigma^*$. All ports are initialized with $\epsilon$, except $\mathsf{clk}^{\triangleleft}? := 1$. Probabilistic choices occur in Phase (2) only and are made by the machines themselves.

1. **Termination:** If $\mathsf{X}$ is in a final state, the run *stops.*
2. **Switch current scheduler:** We switch machine $\mathsf{M} :=$ *curr_sched* and then assign $\epsilon$ to all in-ports of $\mathsf{M}$ (which might include $\mathsf{clk}^{\triangleleft}?$).
3. **Save messages from $\mathsf{M}$ in buffers:** We switch each buffer $\tilde{\mathsf{p}}$ where $\mathsf{p}!$ is a simple out-port of $\mathsf{M}$, in the given order of these ports, with inputs $\mathsf{p}^{\triangleleft}? := \epsilon$ and $\mathsf{p}^{\leftrightarrow}? := \mathsf{p}!$. Then we assign $\epsilon$ to all these ports $\mathsf{p}!$ and $\mathsf{p}^{\leftrightarrow}?$.
4. **Determine next scheduler:** If at least one clock out-port of $\mathsf{M}$ carries a value $\neq \epsilon$, then let $\mathsf{q}^{\triangleleft}!$ denote the first such port, and $\mathsf{M}'$ the unique machine with $\mathsf{q}? \in \mathsf{ports}(\mathsf{M}')$. We set *curr_sched* $:= \mathsf{M}'$ and assign $\epsilon$ to all other clock out-ports of $\mathsf{M}$.

   Otherwise we set *curr_sched* $:= \mathsf{X}$ and $\mathsf{clk}^{\triangleleft}? := 1$ and go back to Phase (1).
5. **Retrieve scheduled message for $\mathsf{M}'$:** We switch $\tilde{\mathsf{q}}$ with input $\mathsf{q}^{\triangleleft}? := \mathsf{q}^{\triangleleft}!$ and $\mathsf{q}^{\leftrightarrow}? := \epsilon$, set $\mathsf{q}? := \mathsf{q}^{\leftrightarrow}!$ and assign $\epsilon$ to all ports of $\tilde{\mathsf{q}}$, and to $\mathsf{q}^{\triangleleft}!$. We go back to Phase (1).

Whenever a machine (this may be a buffer) with name $name_{\mathsf{M}}$ is switched from $(s_{\mathsf{M}}, I_{\mathsf{M}})$ to $(s'_{\mathsf{M}}, O_{\mathsf{M}})$ with non-final $s_{\mathsf{M}}$ and non-empty input $I_{\mathsf{M}}$, we add $(name_{\mathsf{M}}, s_{\mathsf{M}}, I_{\mathsf{M}}, s'_{\mathsf{M}}, O_{\mathsf{M}})$ to the run. This gives a family of random variables

$$run_C = (run_{C,ini})_{ini \in Ini}.$$

For a number $l \in \mathbb{N}$, $l$-step prefixes $run_{C,ini,l}$ of runs are defined in the obvious way. For a function $l : Ini \to \mathbb{N}$, this gives a family $run_{C,l} = (run_{C,ini,l(ini)})_{ini \in Ini}$. ◇

*Remark 2.1.* Whenever a non-buffer machine $\mathsf{M}$ is switched, there is at most one port $p \in \mathsf{ports}(C)$ with $p \neq \epsilon$. If it exists, $p \in \mathsf{ports}(\mathsf{M})$.

∘

[7] One can also define *pol*-time buffers for all polynomials *pol*. One can either tacitly assume below that each polynomial-time system uses appropriate buffers, or easily show that unbounded buffers make no difference.

[8] Hence there is exactly one master scheduler, identified by having the port $\mathsf{clk}^{\triangleleft}?$.

**Definition 2.6** *(View)* The view of a subset $M$ of a closed collection $C$ in a run $r$ is the restriction of $r$ to $M$. Restriction means that all steps $(name, s, I, s', O)$ where $name$ is the name of a machine $\mathsf{M} \notin M$ are deleted from $r$.[9] This gives a family of random variables

$$view_C(M) = (view_{C,ini}(M))_{ini \in Ini},$$

and similarly for $l$-step prefixes. $\diamond$

*Remark 2.2.* If all machines in the collection are polynomial-time, the runs are of polynomial size. Views of polynomial-time machines are always of polynomial size. $\circ$

## 2.2 Security-Specific System Model

Now we define specific collections for security purposes, first the system part and then the environment, i.e., users and adversaries. Typically, a cryptographic system is described by an *intended structure*, and the actual structures are derived using a *trust model* (see Section 3.1). However, as a wide range of trust models is possible, we keep the remaining definitions independent of them by a general system definition.

**Definition 2.7** *(Structures and Systems)* A *structure* is a pair $struc = (M, S)$ where $M$ is a collection of simple machines called *correct machines*, and $S \subseteq \mathsf{free}([M])$ is called *specified ports*. Let $\bar{S} := \mathsf{free}([M]) \setminus S$ and $\mathsf{forb}(M, S) := \mathsf{ports}(M) \cup \bar{S}^c$. A *system Sys* is a set of structures. A system is polynomial-time iff all its collections $M$ are. $\diamond$

The separation of the free ports into specified ports and others is an important feature of our particular reactive simulatability definitions. The specified ports are those where a certain service is guaranteed. Typical examples of inputs at specified ports are "send message $m$ to $id$" for a message transmission system or "pay amount $x$ to $id$" for a payment system. The ports in $\bar{S}$ are additionally available for the adversary. The ports in $\mathsf{forb}(M, S)$ will therefore be forbidden for an honest user to have. In the simulatability definition below, only the events at specified ports have to be simulated one by one. This allows *abstract* specification of systems with *tolerable imperfections*. See Section 5 for an example (and [27, 28] for more).

**Definition 2.8** *(Configuration)* A *configuration conf* of a system *Sys* is a tuple $(M, S, \mathsf{H}, \mathsf{A})$ where $(M, S) \in Sys$ is a structure, $\mathsf{H}$ is a simple machine without forbidden ports, $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(M, S) = \emptyset$, and the completion $C := [M \cup \{\mathsf{H}, \mathsf{A}\}]$ is a closed collection with master scheduler $\mathsf{A}$.

The set of all configurations is written $\mathsf{Conf}(Sys)$, and those with polynomial-time user $\mathsf{H}$ and adversary $\mathsf{A}$ are called $\mathsf{Conf}_{\mathsf{poly}}(Sys)$. "poly" is omitted if it is clear from the context. Runs and views of a configuration are given by Definitions 2.5 and 2.6.

Typically, the initial states of all machines are only a security parameter $k$ (in unary representation). Then we consider the families of runs and views restricted to the subset $Ini' = \{(1^k)_{\mathsf{M} \in C} | k \in \mathbb{N}\}$ of $Ini$, and write $run_{conf}$ and $view_{conf}(M)$ for $run_C$ and $view_C(M)$ restricted to $Ini'$, and similar for $l$-step prefixes. Furthermore, $Ini'$ is identified with $\mathbb{N}$; hence one can write $run_{conf,k}$ etc. $\diamond$

*Remark 2.3.* The conditions on the ports of $\mathsf{H}$ in a configuration is equivalent to (1) $\mathsf{ports}(\mathsf{H}) \cap \mathsf{ports}(M) = \emptyset$ and (2) $\mathsf{ports}(\mathsf{H})^c \cap \mathsf{ports}([M]) \subseteq S$.

Proof: The original condition can be written as (1) and (3) $\mathsf{ports}(\mathsf{H})^c \cap (\mathsf{free}([M]) \setminus S) = \emptyset$, and (3) $\Leftrightarrow$ $\mathsf{ports}(\mathsf{H})^c \cap \mathsf{free}([M]) \subseteq S$. Thus it is thus sufficient to show (4) $\mathsf{ports}(\mathsf{H})^c \cap \mathsf{inner}([M]) = \emptyset$, where $\mathsf{inner}(P) := \mathsf{ports}(P) \setminus \mathsf{free}(P)$ for all sets $P$. Clearly $\mathsf{inner}(P)^c = \mathsf{inner}(P)$ for all $P$. Hence (4) $\Leftrightarrow$ $\mathsf{ports}(\mathsf{H}) \cap \mathsf{inner}([M]) = \emptyset$. Now $\mathsf{ports}([M])$ only contains additional buffer ports and clock in-ports compared with $\mathsf{ports}(M)$. Hence (1) even implies $\mathsf{ports}(\mathsf{H}) \cap \mathsf{ports}([M]) = \emptyset$.

*Remark 2.4.* For readers familiar with [27]: Definition 2.8 corresponds to the definition of configurations for the "specified-user-interface model" in [27]. We now used this case because it is somewhat simpler (in the valid mappings below) and sufficient for cryptographic purposes. $\circ$

[9] For the view of a polynomial-time Turing machine in interaction with unrestricted machines, inputs are only considered as far as the machine reads them.

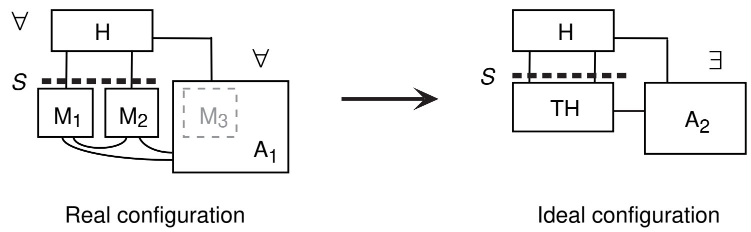


Figure 2: Example of simulatability; the views of H are compared.

## 2.3 Simulatability

We now define the security of a system $Sys_1$ relative to another system $Sys_2$. Typically, we only want to compare each structure of $Sys_1$ with certain corresponding structures in $Sys_2$. An almost arbitrary mapping $f$ is allowed as specification of "corresponding;" we just require that only structures with the same set of specified ports are corresponding. An instantiation of $f$ is usually derived from the trust model, see Section 3.

**Definition 2.9** *(Valid Mapping)* A function $f$ from a system $Sys_1$ to the powerset of a system $Sys_2$ is called a *valid mapping* if $S_1 = S_2$ for all structures $(M_1, S_1)$ and $(M_2, S_2) \in f(M_1, S_1)$.

Given $Sys_2$ and $f$, the set $\mathsf{Conf}^f(Sys_1)$ contains those configurations $(M_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}(Sys_1)$ where $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(M_2, S) = \emptyset$ for all $(M_2, S) \in f(M_1, S)$; we call them *suitable configurations.* $\diamond$

*Remark 2.5.* In most practical systems, $S$ uniquely determines $(M, S)$, and thus $f$ is trivial and could be omitted. But for some problems it is natural to consider several structures with the same set $S$; see [27].

*Remark 2.6.* For a valid mapping $f$, we have $S^c \cap \mathsf{forb}(M_i, S) = \emptyset$ for $i = 1, 2$, i.e., the ports that users are intended to use are not at the same time forbidden (also not in the corresponding structures of the other system).[10]

Proof: Recall that $\mathsf{forb}(M_i, S) = \mathsf{ports}(M_i) \cup (\mathsf{free}([M_i]) \setminus S)^c$. Obviously we only have to show $S^c \cap \mathsf{ports}(M_i) = \emptyset$. This follows from $S \subseteq \mathsf{free}([M_i])$: If $p \in S$, then $p^c \notin \mathsf{ports}([M_i]) \supseteq \mathsf{ports}(M_i)$.

*Remark 2.7.* With regard to $Sys_1$ alone, the restriction to suitable configurations is w.l.o.g.: For every $conf_1 = (M_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}(Sys_1) \setminus \mathsf{Conf}^f(Sys_1)$, there is a configuration $conf_{\mathsf{s},1} = (M_1, S, \mathsf{H}_\mathsf{s}, \mathsf{A}_{\mathsf{s},1}) \in \mathsf{Conf}^f(Sys_1)$ such that $view_{conf_{\mathsf{s},1}}(\mathsf{H}_\mathsf{s}) = view_{conf_1}(\mathsf{H})$.

Proof: We want to construct $\mathsf{H}_\mathsf{s}$ by giving each port $p \in \mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(M_2, S)$ a new name.[11] Clearly the runs and views remain the same if we consistently rename all six ports with the same name. (Recall that the port names are not part of the view.)The new collection is a configuration $(M_1, S, \mathsf{H}_\mathsf{s}, \mathsf{A}_{\mathsf{s},1})$ if none of the renamed ports belongs to $[M_1]$. If it were, then already $p^c \in \mathsf{ports}([M_1])$ (with any port, the entire buffer belongs to $[M_1]$). Now Remark 2.3 implies $p^c \in S$. As $f$ is a valid mapping, Remark 2.6 implies $p \notin \mathsf{forb}(M_2, S)$, in contradiction to the original condition on $p$. $\circ$

For two families $(\mathsf{var}_k)_{k \in \mathbb{N}}$ and $(\mathsf{var}'_k)_{k \in \mathbb{N}}$ of random variables (or probability distributions) let "$=$", "$\approx_{SMALL}$", "$\approx_{\mathsf{poly}}$" denote perfect indistinguishability, statistical indistinguishability (for a class $SMALL$ of functions from $\mathbb{N}$ to $\mathbb{R}_{\geq 0}$), and computational indistinguishability, respectively [34] (see Definition A.2). We write $\approx$ if we want to treat all cases together. The following definition captures that whatever an adversary can achieve in the real system against certain honest users, another adversary can achieve against the same honest users in the ideal system. A typical situation is illustrated in Figure 2.

**Definition 2.10** *(Simulatability)* Let systems $Sys_1$ and $Sys_2$ with a valid mapping $f$ be given.

[10]For readers familiar with [27]: There in Section 5.6, the condition on a valid mapping $f$ was precisely the generalization of this condition to mappings that allow different sets $S_1$ and $S_2$. Here we use the stronger requirement $S_1 = S_2$ as it simplifies the presentation, and is sufficient for all cryptographic examples we considered. See [27] for a non-cryptographic example where $S_1 \neq S_2$ shows up in a natural specification.

[11]By *new name*, we always mean one that does not occur in the systems and configurations already under consideration, here $Sys_1$, $Sys_2$, and $conf_1$. We can assume w.l.o.g. that this is possible. (Otherwise we can, e.g., extend the alphabet.)

a) We say $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{perf}} Sys_2$ (*perfectly at least as secure as* for $f$) if for every configuration $conf_1 = (M_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^f(Sys_1)$, there exists a configuration $conf_2 = (M_2, S, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(Sys_2)$ with $(M_2, S) \in f(M_1, S)$ (and the same $\mathsf{H}$) such that

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

b) We say $Sys_1 \geq_{\mathsf{sec}}^{f,SMALL} Sys_2$ (*statistically at least as secure as*) for a class $SMALL$ if the same as in a) holds with statistical indistinguishability of all families $view_{conf_1,l}(\mathsf{H})$ and $view_{conf_2,l}(\mathsf{H})$ of $l$-step prefixes of the views for polynomials $l$.

c) We say $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{poly}} Sys_2$ (*computationally at least as secure as*) if the same as in a) holds with configurations from $\mathsf{Conf}^f_{\mathsf{poly}}(Sys_1)$ and $\mathsf{Conf}_{\mathsf{poly}}(Sys_2)$ and computational indistinguishability of the families of views.

In all cases, we call $conf_2$ an indistinguishable configuration for $conf_1$. Where the difference between the types of security is irrelevant, we simply write $\geq_{\mathsf{sec}}^f$, and we omit the indices $f$ and $\mathsf{sec}$ if they are clear from the context. ◇

*Remark 2.8.* Adding a free adversary out-port in the comparison (like the guessing-outputs used to define semantic security [16]) does not make the definition stricter: Any such out-port could be connected to an in-port given to the honest user. $\mathsf{H}$ would simply ignore this in-port, but nevertheless it would be included in the view of $\mathsf{H}$, i.e., in the comparison. (This was shown in detail for the synchronous model in [27].) ∘

## 2.4 Basic Lemmas and Blackbox Simulation

An essential ingredient in the composition theorem and other uses of the model is a notion of combining several machines into one, and a lemma that this makes no essential difference in views.

**Definition 2.11** *(Combination of Machines)* Let $C$ be a collection without buffers, and $D \subset C$. We define the *combination* of $D$ into one machine $\hat{\mathsf{D}}$:

- The name of $\hat{\mathsf{D}}$ is chosen arbitrarily, but different from the names of the machines in $C$, and $Ports_{\hat{\mathsf{D}}} := \mathsf{ports}(D)$ (in an arbitrary order).
- $\hat{\mathsf{D}}$ has all machines in $D$ as sub-machines: $States_{\hat{\mathsf{D}}} = \times_{\mathsf{M} \in D} States_{\mathsf{M}}$, and $\delta_{\hat{\mathsf{D}}}$ is defined by applying the transition functions of all sub-machines to the corresponding substates and inputs, unless $\hat{\mathsf{D}}$ has reached a final state (see below). In that case, $\delta_{\hat{\mathsf{D}}}$ does not change the state anymore and produces no output.
- The initial states are $Ini_{\hat{\mathsf{D}}} = \times_{\mathsf{M} \in D} Ini_{\mathsf{M}}$. If there is a master scheduler $\mathsf{X} \in D$ then $Fin_{\hat{\mathsf{D}}}$ is the set of all states of $\hat{\mathsf{D}}$ where $\mathsf{X}$ is in a state from $Fin_{\mathsf{X}}$. Otherwise $\hat{\mathsf{D}}$ stops as soon as *all* sub-machines stopped: $Fin_{\hat{\mathsf{D}}} = \times_{\mathsf{M} \in D} Fin_{\mathsf{M}}$.

◇

*Remark 2.9.* $\hat{\mathsf{D}}$ is well-defined: $C$ is a collection, thus all machines in $D \subset C$ have unique names and disjoint port sets, and $\delta_{\hat{\mathsf{D}}}$ is well-defined. By definition, $\delta_{\hat{\mathsf{D}}}$ applied to a final state of $\hat{\mathsf{D}}$ or to an empty input does not change the state and produces no output. Furthermore, it is again simple or a master scheduler because we only combined such machines.

∘

**Lemma 2.1** *(Combination of Machines)* With the notation of Definition 2.11:

1. If $[C]$ is closed then $[C \setminus D \cup \{\hat{\mathsf{D}}\}]$ is closed as well.
2. The view of any set of original machines in $C \setminus D \cup \{\hat{\mathsf{D}}\}$ is the same as in $C$. This includes the view of the (sub-)machines in $\hat{\mathsf{D}}$, which is well-defined (given $C$ and $D$).

3. If all machines in $D$ are polynomial-time, then so is $\hat{\mathsf{D}}$.

$\square$

*Proof.* (1) Let $C^* := C \setminus D \cup \{\hat{\mathsf{D}}\}$. Since we selected a fresh name for $\hat{\mathsf{D}}$ and did not add any port, $C^*$ is a collection. By construction $\mathsf{ports}(C^*) = \mathsf{ports}(C)$, which implies $\mathsf{ports}([C^*]) = \mathsf{ports}([C])$ and thus $\mathsf{free}([C^*]) = \mathsf{free}([C])$. Thus $C^*$ is still closed.

(2) Whenever the run algorithm (Def. 2.5) switches a non-buffer machine $\mathsf{M}$ then that machine is the only one that has a non-empty input, and only at one port (Remark 2.1). Therefore we can identify each step of $\hat{\mathsf{D}}$ with a step of the unique sub-machine of $\hat{\mathsf{D}}$ that receives a non-empty input in that step, and vice versa. The other submachines, although switched by $\delta_{\hat{\mathsf{D}}}$, neither change their states nor produce an output (Def. 2.2). Hence it makes no difference to the variables of the run algorithm that they are not switched in $C$. In Phase (3) possibly more buffers switch in $[C^*]$ than in $[C]$, but the additional buffers do not receive an input and therefore neither change their states nor produce outputs. Hence nothing is added to the run. Overall, we have defined a bijection between the runs of the two systems whose projection to views of any subsets of original machines are clearly identical.

(3) The running time of $\hat{\mathsf{D}}$ is bounded by the sum of the running times of the machines in $D$. $\blacksquare$

Definition 2.11 allows us to add the notion of blackbox simulatability to Definition 2.10: Here $\mathsf{A}_2$ is given as the combination of a fixed "simulator" $\mathsf{Sim}$ and a machine $\mathsf{A}_1'$ that differs from $\mathsf{A}_1$ at most in the names and labels of some ports. The partial function $\sigma$ that defines this renaming is tacitly assumed to be given with $\mathsf{Sim}$. $\mathsf{A}_1$ is then called a *blackbox sub-machine* of $\mathsf{Sim}$.

**Definition 2.12** *(Blackbox and Universal Simulatability)* Universal simulatability means that $\mathsf{A}_2$ in Definition 2.10 does not depend on $\mathsf{H}$ (only on $M_1$, $S_1$, and $\mathsf{A}_1$).

Blackbox simulatability means that $\mathsf{A}_2$ is a simulator $\mathsf{Sim}$ with $\mathsf{A}_1$ as a blackbox sub-machine, where $\mathsf{Sim}$ depends at most on $M_1$, $S$ and $\mathsf{ports}(\mathsf{A}_1)$.[12] $\diamond$

**Lemma 2.2** *(Types of Security)* If $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{perf}} Sys_2$, then $Sys_1 \geq_{\mathsf{sec}}^{f,SMALL} Sys_2$ for any non-empty class $SMALL$, and $Sys_1 \geq_{\mathsf{sec}}^{f,SMALL} Sys_2$ for a class $SMALL \subseteq NEGL$ implies $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{poly}} Sys_2$. $\square$

The easy proof of this lemma (using Lemma A.1) is omitted. (It is literally identical to the proof of Lemma 4.3 of [27].)

**Lemma 2.3** *(Transitivity)* If $Sys_1 \geq^{f_1} Sys_2$ and $Sys_2 \geq^{f_2} Sys_3$, then $Sys_1 \geq^{f_3} Sys_3$, where $f_3 := f_2 \circ f_1$ is defined in a natural way as follows: $f_3(M_1, S)$ is the union of the sets $f_2(M_2, S)$ with $(M_2, S) \in f_1(M_1, S)$. This holds for perfect, statistical and computational security. It also holds for universal and blackbox simulatability. $\square$

*Proof.* Clearly, $f_3$ is always a valid mapping.

Let a configuration $conf_1 = (M_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^{f_3}(Sys_1)$ be given. Hence $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(M_3, S) = \emptyset$ for all $(M_3, S) \in f_3(M_1, S)$ $(*)$.

If $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(M_2, S) \neq \emptyset$ for any $(M_2, S) \in f_1(M_1, S)$, we give these ports new names: By Remark 2.7, we derive a configuration $conf_{\mathsf{s},1} = (M_1, S, \mathsf{H_s}, \mathsf{A}_{\mathsf{s},1}) \in \mathsf{Conf}^{f_1}(Sys_1)$ with $view_{conf_{\mathsf{s},1}}(\mathsf{H_s}) = view_{conf_1}(\mathsf{H})$.

Now there exists a configuration $conf_{\mathsf{s},2} = (M_2, S, \mathsf{H_s}, \mathsf{A}_{\mathsf{s},2}) \in \mathsf{Conf}(Sys_2)$ with $(M_2, S) \in f_1(M_1, S)$ such that $view_{conf_{\mathsf{s},1}}(\mathsf{H_s}) \approx view_{conf_{\mathsf{s},2}}(\mathsf{H_s})$.

As $\mathsf{H_s}$ only has ports from $\mathsf{H}$ and new ports, it has no ports from $\mathsf{forb}(M_3, S)$ for any structure $(M_3, S) \in f_2(M_2, S)$, i.e., $conf_{\mathsf{s},2} \in \mathsf{Conf}^{f_2}(Sys_2)$. Hence there exists $conf_{\mathsf{s},3} = (M_3, S, \mathsf{H_s}, \mathsf{A}_{\mathsf{s},3}) \in \mathsf{Conf}(Sys_3)$ with $(M_3, S) \in f_2(M_2, S)$ and $view_{conf_{\mathsf{s},2}}(\mathsf{H_s}) \approx view_{conf_{\mathsf{s},3}}(\mathsf{H_s})$.

Together, we have $(M_3, S) \in f_3(M_1, S)$ by definition of $f_3$ and $view_{conf_{\mathsf{s},1}}(\mathsf{H_s}) \approx view_{conf_{\mathsf{s},3}}(\mathsf{H_s})$ because indistinguishability is transitive.

Finally, we derive a configuration $conf_3 = (M_3, S, \mathsf{H}, \mathsf{A}_3)$ with the original user $\mathsf{H}$. For each changed port $p \in \mathsf{ports}(\mathsf{H})$, no port with the same name occurs in $\mathsf{ports}([M_3])$ because the name was new. Thus

[12] A dependence on the adversary ports is shown in Section 3.3 where $\mathsf{Sim}$ completely encloses $\mathsf{A}_1$. In the Secure Message Transmission example, no such dependence occurs.

we can change them back iff their old name also does not occur in $\mathsf{ports}([M_3])$. If this were not true, then as in the proof of Remark 2.7, already $p^c \in \mathsf{ports}([M_3])$, and Remark 2.3 and $(*)$ imply $p^c \in S$. As $f_2$ is a valid mapping, Remark 2.6 implies $p \notin \mathsf{forb}(M_2, S)$, in contradiction to the condition for renaming $p$.

Hence we have $view_{conf_3}(\mathsf{H}) = view_{conf_{\mathsf{s},3}}(\mathsf{H_s})$ and thus $view_{conf_1}(\mathsf{H}) \approx view_{conf_3}(\mathsf{H})$.

Now we consider universal and blackbox simulatability. First, the renaming from $\mathsf{A}_1$ to $\mathsf{A}_{\mathsf{s},1}$ can be described in terms of the ports of $\mathsf{A}_1$ and $(M_1, S)$ (they uniquely define the ports of $\mathsf{H}$ with the same name as a port of $\mathsf{A}_1$). For $\mathsf{A}_{\mathsf{s},2}$ and $\mathsf{A}_{\mathsf{s},3}$ we use the given universality, and the last renaming into $\mathsf{A}_3$ is the reverse of the first. Renaming can also be done as a blackbox construction. ■

**Lemma 2.4** *(Reflexivity)* The relation $\geq^f$ is reflexive with the identity function $f = \mathsf{id}$. □

This is clear because $\mathsf{id}$ is a valid mapping and the indistinguishability is trivial.

# 3 Examples and Adaptive Adversaries

In this section we present a few examples how to use the model. Section 3.1 defines standard cryptographic systems in our model. Section 3.2 shows how to model adaptive adversaries. Section 3.3 shows how one can make more general blackbox simulations than those occurring in Section 5.

## 3.1 Standard Cryptographic Systems with Static Adversary Model

The intuition behind this class of systems is that in a real system *Sys*, there is one machine per human owner, and each machine is correct if and only if its owner is honest. The system is derived from an intended structure $(M^*, S^*)$ and a trust model.

We define that all buffers that connect different machines are scheduled by the adversary. We only allow $\mathsf{M}_u$ to schedule buffers that transport messages from itself to itself, and require all these connections to be secure: this allows us to define a machine $\mathsf{M}_u$ as a combination of (local) submachines.

**Definition 3.1** *(Standard Cryptographic Systems)* A *standard cryptographic structure* is a structure $(M^*, S^*)$ where $M^* = \{\mathsf{M}_1, \ldots, \mathsf{M}_n\}$ and $S^{*c} = \{\mathsf{in}_u!, \mathsf{out}_u?|u = 1, \ldots, n\}$, where $\mathsf{in}_u?$ and $\mathsf{out}_u!$ are ports of machine $\mathsf{M}_u$.[13] Each machine $\mathsf{M}_u$ is simple, and for all names $p$, if $p^{\triangleleft}! \in \mathsf{ports}(\mathsf{M}_u)$ then $p?, p! \in \mathsf{ports}(\mathsf{M}_u)$.

A *standard trust model* for such a structure consists of an access structure, $\mathcal{ACC}$, and a channel model, $\chi$. $\mathcal{ACC}$ is a set of subsets $\mathcal{H}$ of $\{1, \ldots, n\}$, closed under insertion, and denotes the possible sets of correct machines.[14] $\chi$ is a mapping $\chi : \mathsf{Gr}(M^*) \to \{\mathsf{s}, \mathsf{a}, \mathsf{i}\}$. It characterizes each high-level connection as secure (private and authentic), authenticated (only authentic), or insecure (neither private nor authentic). We require that if a connection $c$ connects a machine $\mathsf{M}_u$ with itself then $\chi(c) = \mathsf{s}$.

Given such a structure and trust model, the corresponding *standard cryptographic system* is $Sys = \{(M_\mathcal{H}, S_\mathcal{H})|\mathcal{H} \in \mathcal{ACC}\}$ with $S_\mathcal{H}^c = \{\mathsf{in}_u!, \mathsf{out}_u?|u \in \mathcal{H}\}$ and $M_\mathcal{H} = \{\mathsf{M}_{u,\mathcal{H}}|u \in \mathcal{H}\}$, where $\mathsf{M}_{u,\mathcal{H}}$ is derived from $\mathsf{M}_u$ as follows:

- The ports $\mathsf{in}_u?$ and $\mathsf{out}_u!$ and all clock ports are unchanged.
- Consider a simple port $p \in \mathsf{ports}(\mathsf{M}_u) \setminus \{\mathsf{in}_u?, \mathsf{out}_u!\}$, where $p^C \in \mathsf{ports}(\mathsf{M}_v)$ with $v \in \mathcal{H}$, i.e., $c = \{p, p^C\}$ is a high-level connection between two correct machines:
  - If $\chi(c) = \mathsf{s}$ (secure), $p$ is unchanged.
  - If $\chi(c) = \mathsf{a}$ (authenticated) and $p$ is an output port, $\mathsf{M}_{u,\mathcal{H}}$ gets an additional new port $p^d$, where it duplicates the outputs at $p$.[15] This port automatically remains free, and thus the adversary connects to it. If $p$ is an input port, it is unchanged.

[13] We have specified the complement of $S^*$ because that is independent of the buffer notation.

[14] Typical examples are threshold structures: $\mathcal{ACC} = \{\mathcal{H} \subseteq \{1, \ldots, n\} \mid |\mathcal{H}| \geq t\}$ for some $t$.

[15] This can be done by a trivial blackbox construction. We assume w.l.o.g. that there is a systematic naming scheme for such new ports (e.g., appending $\mathsf{d}$) that does not clash with prior names.

– If $\chi(c) = \mathsf{i}$ (insecure) and $p$ is an input port, $p$ is replaced by a new port $p^a$. (Thus the adversary can get the outputs from $p^C$ and make the inputs to $p^a$ and thus completely control the connection.) If $p$ is an output port, it is unchanged.

- Consider a simple port $p \in \mathsf{ports}(\mathsf{M}_u) \setminus \{\mathsf{in}_u?, \mathsf{out}_u!\}$, where $p^C \notin \mathsf{ports}(\mathsf{M}_v)$ for all $v \in \mathcal{H}$: If $p$ is an output port, it is unchanged. If it is an input port, it is renamed into $p^a$. (In both cases the adversary can connect to it.)

$\diamond$

A typical ideal system is of the form $Sys_2 = \{(\{\mathsf{TH}_\mathcal{H}\}, S_\mathcal{H}) | \mathcal{H} \in \mathcal{ACC}\}$ with the same sets $S_\mathcal{H}$ as in the corresponding real system $Sys_1$. The *canonical mapping* $f$ between such systems is defined by $f(M_\mathcal{H}, S_\mathcal{H}) = \{(\{\mathsf{TH}_\mathcal{H}\}, S_\mathcal{H})\}$ for all $\mathcal{H}$.

## 3.2 Standard Cryptographic Systems with Adaptive Adversary Model

Standard cryptographic systems as defined in the previous section are based on the intuition that it is a priori clear who are the "bad guys" and the "good guys." In *adaptive* 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 [6, 8]. 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 sets of machines (e.g., [8]).

*Standard cryptographic systems with adaptive adversary* can easily be defined within our model: Such a system has only one structure, $(M, S)$. It is derived from an intended structure $(M^*, S^*)$, where $M^*$ is a set $\{\mathsf{M}_1, \ldots, \mathsf{M}_n\}$ and $S^{*c} = \{\mathsf{in}_u!, \mathsf{out}_u?, \mathsf{corrupt}_u! | u = 1, \ldots, n\}$, where the ports $\mathsf{in}_u?$, $\mathsf{out}_u!$, and $\mathsf{corrupt}_u?$ belong to $\mathsf{M}_u$. The derivation is done with the access structure $\mathcal{ACC} = \{\mathcal{M}\}$ for $\mathcal{M} = \{1, \ldots, n\}$ (all intended machines are present) and an arbitrary channel model satisfying that all connections from an $\mathsf{M}_u$ to itself are secure.

The ports $\mathsf{corrupt}_u?$ are for corruption requests. (Those must be made via specified ports because the service will change at least at the corresponding ports $\mathsf{in}_u?$, $\mathsf{out}_u!$ also in the ideal system.) Each $\mathsf{M}_u$ also has two specific ports cor_out!, cor_in? for communication with A after corruption. A newly corrupted machine sends a predefined "corruption response" to A via cor_out!, and from then on simply forwards user inputs to A and inputs from A to the user.

Two main types of corruption responses are natural:

- The current state of $\mathsf{M}_u$.
- The entire view of $\mathsf{M}_u$. This corresponds to the assumption that nothing can be erased reliably. Thus every transition of $\delta_{M^*}$ is modified in $\delta_{\mathsf{M}_u}$ to store the current step.

One may also extend this to different classes of storage, e.g., to model the different vulnerability of session keys and long-term keys.

In the ideal system, the trusted host TH also accepts corruption requests. Tolerable sets of corrupted machines are defined by an access structure $\mathcal{ACC}^*$ within TH: if the set of corruption requests is no longer in $\mathcal{ACC}^*$, i.e., there were "too many", then TH sends its state to A and gives all control to A. Thus after this, the ideal system no longer guarantees anything and simulation becomes trivial.

## 3.3 Toy Example: Transparent Simulator

Definition 2.12 introduced the notion of a blackbox simulation. In the following, we show that this also allows Sim to become master scheduler and determine its own schedule. The main purpose is to make sure that our port naming and run algorithm are powerful enough. Hence we take a simple example, where Sim does nothing except shielding A and the system from each other completely.

This is shown in Figure 3. The left side shows a collection $\{\mathsf{A}, \mathsf{H}, \mathsf{M}\}$ with scheduler A. Solid arrows represent high-level connections, a dashed arrow pointing to a solid arrow represents the corresponding clock out-port and buffer. The right side shows the same machines H and M, the transparent simulator

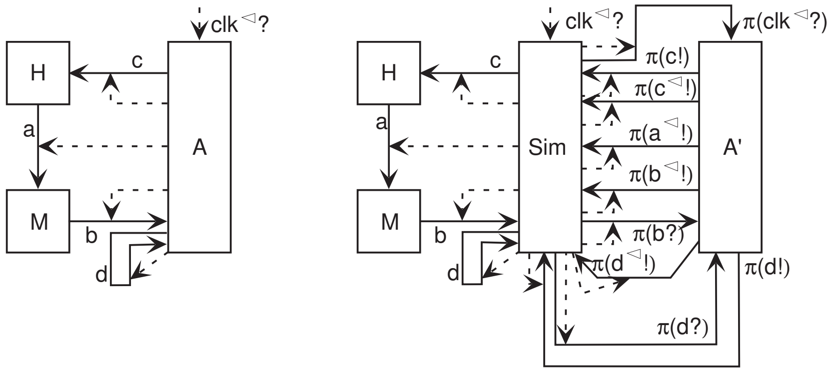


Figure 3: Example of a transparent simulator. The dotted arrows connect clock ports.

$\mathsf{Sim}$ and a machine $\mathsf{A}'$. $\mathsf{A}'$ is derived from $\mathsf{A}$ by giving each port of $\mathsf{A}$ a new name and making it simple. (In particular, ports with the same name are *not* renamed consistently, because $\mathsf{Sim}$ wants to intercept all messages.) One way to do this generically is to extend $\Sigma$ by new symbols in one-to-one correspondence, denoted by $\pi$, with $!, ?, ^{\leftrightarrow}, ^{\triangleleft}$. Then a port $p = (n, l, d)$ becomes $p' = (n\pi(l)\pi(d), \epsilon, d)$. Extending $\pi$ with the identity on $\Sigma$, and to strings, we can write this $p' = (\pi(nld), \epsilon, d)$.

As the number and directions of the ports do not change, $\delta_{\mathsf{A}'} = \delta_{\mathsf{A}}$ is possible. $\mathsf{Sim}$ has the ports of $\mathsf{A}$ and all high-level complements and clock out-ports with the new names. It will always keep all buffers connected to $\mathsf{A}'$ empty.

Our goal is that the views of $\{\mathsf{A}, \mathsf{H}, \mathsf{M}\}$ and of $\{\mathsf{A}', \mathsf{H}, \mathsf{M}\}$ are perfectly indistinguishable, i.e., $\mathsf{Sim}$ is transparent.

Simulating a single step of $\mathsf{A}$ (this starts when $\mathsf{Sim}$ gets an input at the master clock or from a buffer with a name over the old $\Sigma$) requires $\mathsf{Sim}$ to switch $\mathsf{A}'$ once, giving $\mathsf{A}'$ the same inputs that $\mathsf{A}$ would have received at the original ports. Our run algorithm ensures that in each step of $\mathsf{A}$ there is at most one input $\mathsf{q}? \neq \epsilon$. Then $\mathsf{Sim}$ outputs $\pi(\mathsf{q}?)! := \mathsf{q}?$ and triggers the corresponding buffer through $\pi(\mathsf{q}?)^{\triangleleft}! := 1$. Hence $\mathsf{A}'$ is switched with input $\pi(\mathsf{q}?)? = \mathsf{q}?$.

$\mathsf{A}'$ might produce multiple outputs; $\mathsf{Sim}$ needs to learn all of them and finally output them on the original ports. Therefore $\mathsf{Sim}$ successively reads all buffers that connect $\mathsf{A}'$ to $\mathsf{Sim}$. Reading a buffer $\widetilde{\mathsf{q}}$ means that $\mathsf{Sim}$ selects that buffer with $\mathsf{q}^{\triangleleft}! := 1$ (in Phase (2) of the run algorithm), and either receives the first (and always only) element of $\widetilde{\mathsf{q}}$ if that exists or is immediately switched as master scheduler if there is no first element (in the next Phase (2)). As soon as $\mathsf{Sim}$ has read all outputs of $\mathsf{A}'$ it outputs them on the original out-ports.

The views of $\{\mathsf{A}, \mathsf{H}, \mathsf{M}\}$ and of $\{\mathsf{A}', \mathsf{H}, \mathsf{M}\}$ are identical: They have the same transition functions and machine names and receive exactly the same inputs in both collections.

# 4 Composition

In this section, we show that the relation "at least as secure as" is consistent with the composition of systems. The basic idea is the following: Assume that we have proven that a system $Sys_0$ is as secure as another system $Sys_0'$ (typically an ideal system used as a specification). Now we would like to use $Sys_0$ as a secure replacement for $Sys_0'$, i.e., as an implementation of the specification $Sys_0'$.

Usually, replacing $Sys_0'$ means that we have another system $Sys_1$ that uses $Sys_0'$; we call this composition $Sys^*$. Inside $Sys^*$ we want to use $Sys_0$ instead, which gives a composition $Sys^\#$. Hence $Sys^\#$ is typically a completely real system, while $Sys^*$ is partly ideal. Intuitively we expect $Sys^\#$ to be at least as secure as $Sys^*$. The situation is shown in the left and middle part of Figure 4.

The remainder of this section is quite similar to the corresponding section of [30] for the synchronous case.

We define composition for every number $n$ of systems $Sys_1, \ldots, Sys_n$. We do not provide a composition operator that produces one specific composition. The reason is that one typically does not want to compose every structure of one system with every structure of the others, but only with certain matching

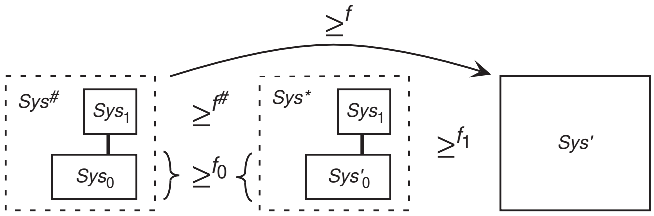


Figure 4: Composition theorem and its use in a modular proof: The left and middle part show the statement of Theorem 4.1, the right part Corollary 4.1.

ones. For instance, if the individual machines of $Sys_2$ are implemented on the same physical devices as those of $Sys_1$, as usual in a layered distributed system, we only compose structures corresponding to the same set of corrupted machines. However, this is not the only conceivable situation. Hence we allow many different compositions.

**Definition 4.1** *(Composition)* The composition of structures and of systems is defined as follows: We call structures $(M_1, S_1), \ldots, (M_n, S_n)$ *composable* if $\mathsf{ports}(M_i) \cap \mathsf{forb}(M_j, S_j) = \emptyset$ and $S_i \cap \mathsf{free}([M_j]) = S_j \cap \mathsf{free}([M_i])$ for all $i \neq j$.[16] We then define their composition as $(M_1, S_1)|| \ldots ||(M_n, S_n) := (M, S)$ with $M = M_1 \cup \ldots \cup M_n$ and $S = (S_1 \cup \ldots \cup S_n) \cap \mathsf{free}([M])$.

We call a system $Sys$ a composition of $Sys_1, \ldots, Sys_n$ and write $Sys \in Sys_1 \times \cdots \times Sys_n$ if each structure $(M, S) \in Sys$ has a unique representation $(M, S) = (M_1, S_1)|| \ldots ||(M_n, S_n)$ with composable structures $(M_i, S_i) \in Sys_i$ for $i = 1, \ldots, n$.

We then call $(M_i, S_i)$ the restriction of $(M, S)$ to $Sys_i$ and write $(M_i, S_i) = (M, S)\lceil_{Sys_i}$. $\diamond$

*Remark 4.1.* For all compositions of structures, we have $[M] = [M_1] \cup \ldots \cup [M_n]$ and $\mathsf{free}([M]) \subseteq \mathsf{free}([M_1]) \cup \ldots \cup \mathsf{free}([M_n])$.

*Remark 4.2.* We also have $S = \mathsf{free}([M]) \setminus (\bar{S}_1 \cup \ldots \cup \bar{S}_n)$.

Proof: Let $S' := \mathsf{free}([M]) \setminus (\bar{S}_1 \cup \ldots \cup \bar{S}_n)$. If $p \in S'$, then $p \in \mathsf{free}([M_i])$ for some $i$ by Remark 4.1, and $p \notin \bar{S}_i$ immediately gives $p \in S_i$.

Now assume that $p \in S$, but $p \notin S'$. Thus $p \in \mathsf{free}([M])$, but there exists $i$ with $p \in \bar{S}_i$. However, there also exists $j$ with $p \in S_j$. Together, $p \in S_j \cap \mathsf{free}([M_i])$. By composability, this implies $p \in S_i \cap \mathsf{free}([M_j])$, contradicting $p \in \bar{S}_i$. $\circ$

The following theorem shows that modular proofs are indeed possible. Recall that the situation is shown in the left and middle part of Figure 4. The main issue in formulating the theorem is to characterize $Sys^\#$, i.e., to formulate what it means that $Sys_0$ replaces $Sys'_0$.

**Theorem 4.1** *(Secure Two-System Composition)* Let systems $Sys_0$, $Sys'_0$, $Sys_1$ and a valid mapping $f_0$ be given with $Sys_0 \geq^{f_0} Sys'_0$. Let compositions $Sys^\# \in Sys_0 \times Sys_1$ and $Sys^* \in Sys'_0 \times Sys_1$ be given that fulfil the following structural conditions:

For each structure $(M^\#, S) \in Sys^\#$ with restrictions $(M_i, S_i) = (M^\#, S)\lceil_{Sys_i}$, all compositions $(M'_0, S_0)||(M_1, S_1)$ with $(M'_0, S_0) \in f_0(M_0, S_0)$ exist, lie in $Sys^*$, and fulfil $\mathsf{ports}(M'_0) \cap S_1^c = \mathsf{ports}(M_0) \cap S_1^c$.

Let $f^\#$ denote the function that maps each $(M^\#, S)$ to the set of these compositions. Then we have $Sys^\# \geq^{f^\#} Sys^*$. This holds for perfect, statistical and, if $Sys_1$ is polynomial-time, for computational security, and also for the universal and blackbox definitions. $\Box$

*Proof.* First we have to show that $f^\#$ is a valid mapping. This will be done in Step 0 below.

Then let a configuration $conf^\# = (M^\#, S, \mathsf{H}, \mathsf{A}^\#) \in \mathsf{Conf}^{f^\#}(Sys^\#)$ be given and $(M_i, S_i) := (M^\#, S)\lceil_{Sys_i}$ for $i = 0, 1$. We have to show that there is an indistinguishable configuration $conf^* \in \mathsf{Conf}(Sys^*)$. The outline of the proof is as follows; it is illustrated in Figure 5.

[16] The first condition makes one system a valid user of another. The second one excludes cases where $p \in \mathsf{free}([M_i]) \cap \mathsf{free}([M_j])$ (e.g., a clock port for a high-level connection between them) and $p \in S_i$ but $p \notin S_j$.

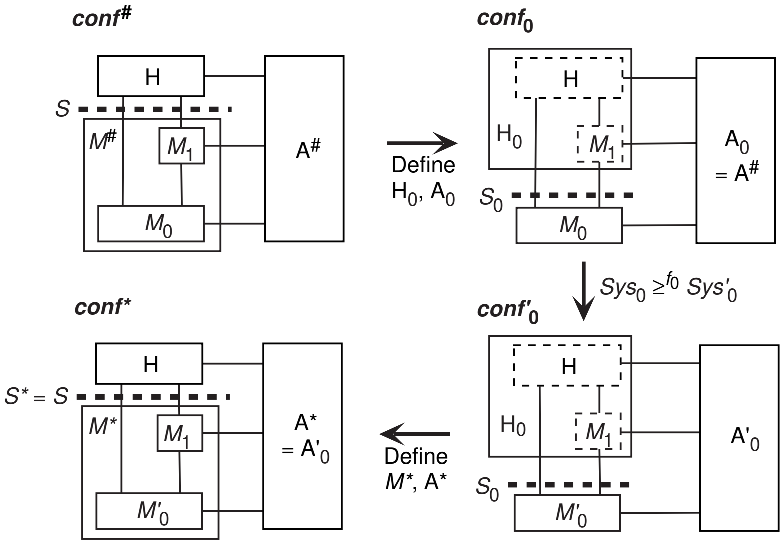


Figure 5: Configurations in the composition theorem. Dashed machines are internal submachines. (The connections drawn inside $\mathsf{H}_0$ are not dashed because the combination does not hide them.)

1. We combine $\mathsf{H}$ and $M_1$ into a user $\mathsf{H}_0$ to obtain a configuration $conf_0 = (M_0, S_0, \mathsf{H}, \mathsf{A}_0) \in \mathsf{Conf}(Sys_0)$ where the view of $\mathsf{H}$ as a submachine of $\mathsf{H}_0$ is the same as that in $conf^{\#}$.
2. We show that $conf_0 \in \mathsf{Conf}^{f_0}(Sys_0)$. Then by the precondition $Sys_0 \geq^{f_0} Sys'_0$, there is a configuration $conf'_0 = (M'_0, S_0, \mathsf{H}_0, \mathsf{A}'_0) \in \mathsf{Conf}(Sys'_0)$ with $(M'_0, S_0) \in f_0(M_0, S_0)$ where the view of $\mathsf{H}_0$ is indistinguishable from that in $conf_0$.
3. We decompose $\mathsf{H}_0$ into $\mathsf{H}$ and $M_1$ again and derive a configuration $conf^* = (M^*, S, \mathsf{H}, \mathsf{A}^*) \in \mathsf{Conf}(Sys^*)$ where the view of $\mathsf{H}$ equals that of $\mathsf{H}$ as a submachine of $\mathsf{H}_0$ in $conf'_0$.
4. We conclude that $conf^*$ is an indistinguishable configuration for $conf^{\#}$.

We now present these steps in detail.

*Step 0:* We have to show that $S^* = S$ whenever $(M^*, S^*) \in f^{\#}(M^{\#}, S)$. Let $(M_i, S_i) := (M^{\#}, S)\lceil_{Sys_i}$ for $i = 0, 1$. Then $(M^*, S^*)$ is a composition $(M'_0, S_0)||(M_1, S_1)$ with $(M'_0, S_0) \in f_0(M_0, S_0)$. Hence $S = (S_0 \cup S_1) \cap \mathsf{free}([M^{\#}])$ and $S^* = (S_0 \cup S_1) \cap \mathsf{free}([M^*])$. (We used that $f_0$ is valid.)

We only show $S \subseteq S^*$; the other direction of the proof is completely symmetric. Hence let $p \in S$.

If $p \in S_0$, then $p^c \notin \mathsf{ports}([M'_0])$ because $(M'_0, S_0)$ is a structure, and $p^c \notin \mathsf{ports}([M^{\#}]) \supseteq \mathsf{ports}([M_1])$ (free in $[M^{\#}]$). Hence $p^c \notin \mathsf{ports}([(M'_0 \cup M_1)]) = \mathsf{ports}([M^*])$. Thus $p \in S^*$.

If $p \in S_1$, then $p^c \notin \mathsf{ports}([M_1])$. Hence $p \notin S^*$ would imply $p^c \in \mathsf{ports}([M'_0])$ and thus $p^c \in \mathsf{ports}(M'_0)$ because it is not a port of a buffer. By the precondition $\mathsf{ports}(M'_0) \cap S_1^c = \mathsf{ports}(M_0) \cap S_1^c$ of the theorem, this would imply $p^c \in \mathsf{ports}(M_0)$ in contradiction to the fact that $p \in \mathsf{free}([M^{\#}])$.

*Step 1:* The precise definition of $conf_0 = (M_0, S_0, \mathsf{H}, \mathsf{A}_0)$ is that $(M_0, S_0) = (M^{\#}, S)\lceil_{Sys_0}$, that $\mathsf{H}_0$ is the combination of $M_1 \cup \{\mathsf{H}\}$ as in Lemma 2.1, and $\mathsf{A}_0 := \mathsf{A}^{\#}$. This is a valid configuration from $\mathsf{Conf}(Sys_0)$:

- $(M_0, S_0) = (M^{\#}, S)\lceil_{Sys_0}$ is a valid structure by the definition of a composition.
- Closed collection: The overall set of ports is the same as in $conf^{\#}$. Hence the machines still have pairwise disjoint port sets, and the free ports are unchanged.
- User condition: We have to show $\mathsf{ports}(\mathsf{H}_0) \cap \mathsf{forb}(M_0, S_0) = \emptyset$. Composability implies $\mathsf{ports}(M_1) \cap \mathsf{forb}(M_0, S_0) = \emptyset$ and port disjointness has just been shown. Thus $\mathsf{ports}(\mathsf{H}) \cap (\mathsf{free}([M_0]) \setminus S_0)^c = \emptyset$ remains to be shown. Assume that a port $p$ contradicts this. Thus $p^c \in \mathsf{free}([M_0]) \subseteq \mathsf{ports}([M^{\#}])$, and the validity of $conf^{\#}$ and Remark 2.3 imply $p^c \in S$. By Remark 4.2, this contradicts $p^c \notin S_0$.

- For the computational case, $\mathsf{H}$ and all machines in $M_1$ are polynomial-time by the preconditions. Hence $\mathsf{H}_0$ is polynomial-time by Lemma 2.1.

Hence Lemma 2.1 implies $\mathit{view}_{\mathit{conf}_0}(\mathsf{H}) = \mathit{view}_{\mathit{conf}^\#}(\mathsf{H})$.

*Step 2:* We now show that $\mathit{conf}_0 \in \mathsf{Conf}^{f_0}(\mathit{Sys}_0)$, i.e., $\mathsf{H}_0$ has no ports from $\mathsf{forb}(M_0', S_0)$ for any structure $(M_0', S_0) \in f_0(M_0, S_0)$. Assume that it had such a port $p$. By construction of $\mathsf{H}_0$, $p$ is a port of $M_1$ or $\mathsf{H}$.

- $p \in \mathsf{ports}(M_1)$: This is immediately excluded by the composability of $(M_0', S_0)$ and $(M_1, S_1)$ (precondition of the theorem).
- $p \in \mathsf{ports}(\mathsf{H})$: By the precondition of the theorem, $(M^*, S) := (M_0', S_0)||(M_1, S_1)$ exists and lies in $f^\#(M^\#, S)$. As $\mathit{conf}^\#$ is suitable, this implies $p \notin \mathsf{forb}(M^*, S)$.

  Hence $p \notin \mathsf{ports}(M^*)$ and thus $p \notin \mathsf{ports}(M_0')$. Thus the case $p^c \in \mathsf{free}([M_0]) \setminus S_0$ remains, and we want to show that it contradicts $p^c \notin \mathsf{free}([M^*]) \setminus S$. First we have $p^c \in \mathsf{free}([M^*])$ because its complement $p$ is a port of $\mathsf{H}$. Secondly, Remark 4.2 implies $p^c \notin S$. This is the desired contradiction.

Hence $\mathit{conf}_0$ is indeed a suitable configuration. Thus $\mathit{Sys}_0 \geq^{f_0} \mathit{Sys}_0'$ implies that there is a configuration $\mathit{conf}_0' = (M_0', S_0, \mathsf{H}_0, \mathsf{A}_0') \in \mathsf{Conf}(\mathit{Sys}_0')$ with $(M_0', S_0) \in f_0(M_0, S_0)$ and $\mathit{view}_{\mathit{conf}_0'}(\mathsf{H}_0) \approx \mathit{view}_{\mathit{conf}_0}(\mathsf{H}_0)$. This implies $\mathit{view}_{\mathit{conf}_0'}(\mathsf{H}) \approx \mathit{view}_{\mathit{conf}_0}(\mathsf{H})$ because the view of a submachine is a function of the larger view (Lemmas A.1 and 2.1).

*Step 3:* We define $\mathit{conf}^* = (M^*, S, \mathsf{H}, \mathsf{A}^*)$ by reversing the combination of $\mathsf{H}$ and $M_1$ into $\mathsf{H}_0$: The structure is $(M^*, S) := (M_0', S_0)||(M_1, S_1)$, the user the original $\mathsf{H}$, and $\mathsf{A}^* := \mathsf{A}_0'$. We show that $\mathit{conf}^* \in \mathsf{Conf}(\mathit{Sys}^*)$.

- Structure: $(M^*, S) \in \mathit{Sys}^*$ follows immediately from the preconditions of the theorem.
- Closed collection: The ports of $\mathsf{H}$ and the machines in $M_1$ are disjoint because so they were in $\mathit{conf}^\#$, and those of all other pairs of machines because so they were in $\mathit{conf}_0'$.[17] As the set of ports is the same as in $\mathit{conf}_0'$, the free ports are unchanged.
- User condition: We have to show $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(M^*, S) = \emptyset$. This is simply the precondition that $\mathit{conf}^\#$ is suitable.

We can now see $\mathit{conf}_0'$ as derived from $\mathit{conf}^*$ by taking the combination of $M_1 \cup \{\mathsf{H}\}$. Hence Lemma 2.1 applies, and we obtain $\mathit{view}_{\mathit{conf}^*}(\mathsf{H}) = \mathit{view}_{\mathit{conf}_0'}(\mathsf{H})$.

*Step 4:* We have shown that $\mathit{conf}^* \in \mathsf{Conf}(\mathit{Sys}^*)$. We also have $(M^*, S) \in f^\#(M^\#, S)$ by the construction of $f^\#$. The results about views in Steps 1 to 3 and transitivity (Lemma 2.3) imply that $\mathit{view}_{\mathit{conf}^*}(\mathsf{H}) \approx \mathit{view}_{\mathit{conf}^\#}(\mathsf{H})$. Hence $\mathit{conf}^*$ is indeed an indistinguishable configuration for $\mathit{conf}^\#$.

*Universal and blackbox:* For the universal case, note that $\mathsf{A}_0 = \mathsf{A}^\#$ does not depend on $\mathsf{H}$. Then $\mathsf{A}_0'$ only depends on $(M_0, S_0)$ and $\mathsf{A}_0$, and thus also $\mathsf{A}^* = \mathsf{A}_0'$. For the blackbox case, $\mathsf{A}_0'$ additionally consists of a simulator $\mathsf{Sim}$ with $\mathsf{A}_0 = \mathsf{A}^\#$ as a blackbox, and thus so does $\mathsf{A}^*$. ■

The following corollary finishes the definition and proof of the situation shown in Figure 4: We now assume that there is also a specification $\mathit{Sys}'$ for the system $\mathit{Sys}^*$, as shown in the left part of the figure.

**Corollary 4.1** Consider five systems satisfying the preconditions of Theorem 4.1, and a sixth one, $\mathit{Sys}'$, with $\mathit{Sys}^* \geq^{f_1} \mathit{Sys}'$. Then $\mathit{Sys}^\# \geq^f \mathit{Sys}'$ where $f := f_1 \circ f^\#$ as in the transitivity lemma. □

[17] Recall that $\mathsf{ports}(\mathsf{H}_0) = \mathsf{ports}(M_1 \cup \{\mathsf{H}\})$; here we exploit that we did not hide internal connections in the combination.

# 5 Secure Message Transmission

In the following we present an ideal and a real system for *secure message transmission.* The real system sends messages over insecure channels, but for the initial key exchange authenticated channels might be used as well.

The specified notion of secure message transmission tolerates some imperfections: The adversary learns who communicates with whom and the length of the messages. He can delay messages, and thus change their order arbitrarily or even suppress them completely. He can replay messages, i.e., if an honest user $u$ received message $m$ from another honest user $v$, the adversary can trick $u$ into accepting $m$ arbitrarily often (although the adversary might have no idea what $m$ looks like).

We allow this to keep the possible real machines simple (stateless except for keys, and without dummy traffic). Clearly, more complicated real systems can avoid some imperfections, although others cannot be avoided in a totally asynchronous scenario. In particular one could add a layer on top of our ideal system that detects replays or even ensures that messages are delivered in the correct sequence (by discarding messages that are out of order). Security of this new, composed system could easily be proven via the composition theorem.

**Notation for data structures.** For $m \in \Sigma^*$ let $\mathsf{len}(m)$ denote the length of $m$. A *list*, $l = (x_1, \ldots, x_k)$, is a sequence of words from $\Sigma^*$, itself encoded as a word from $\Sigma^*$. We also call fixed-length lists tuples. The exact encoding does not matter as long as the number $\mathsf{size}(l)$ of elements in $l$ and these elements are efficiently retrievable. Furthermore, we assume that the length of a list is (efficiently) computable from the length of its elements. For a list $l = (x_1, \ldots, x_j)$ we define $l[i] := x_i$ for $1 \le i \le j$, and $l[i] := \downarrow$ for $i > j$, where $\downarrow$ is a distinct error symbol, i.e., $\notin \Sigma$. () denotes an empty list. By "adding an element to a list" and similar formulations we mean appending it at the end. By $\exists x \in l$ we mean that $l[i] = x$ for some $i$. If we write this in a retrieval operation, the first such $x$ is used.

## 5.1 The Ideal System

The ideal system is of the standard cryptographic type described at the end of Section 3.1.

**Scheme 5.1 (Ideal System for Secure Message Transmission)** Let $n, s \in \mathbb{N}$ and a polynomial $L$ over $\mathbb{N}$ be given. Here $n$ denotes the number of intended participants, $s$ the maximum number of messages each user can send, and $L$ the maximum message length as a function of the security parameter $k$. Let $\mathcal{M} := \{1, \ldots, n\}$. An ideal system for secure message transmission is then defined as

$$Sys^{\mathsf{secmsg,ideal}}_{n,s,L} = \{(\{\mathsf{TH}_\mathcal{H}\}, S_\mathcal{H}) \mid \mathcal{H} \subseteq \mathcal{M}\},$$

where $\mathsf{TH}_\mathcal{H}$ and $S_\mathcal{H}$ are defined as follows. When $\mathcal{H}$ is clear from the context, let $\mathcal{A} := \mathcal{M} \setminus \mathcal{H}$ denote the set of corrupted participant indices.

The ports of $\mathsf{TH}_\mathcal{H}$ are $\{\mathsf{in}_u?, \mathsf{out}_u! \mid u \in \mathcal{H}\} \cup \{\mathsf{in_{sim}}?, \mathsf{out_{sim}}!, \mathsf{out_{sim}}^{\triangleleft}!\}$.[18] The complements of the specified ports are $S^c_\mathcal{H} := \{\mathsf{in}_u!, \mathsf{out}_u? \mid u \in \mathcal{H}\}$.

$\mathsf{TH}_\mathcal{H}$ maintains three data structures: An array $(init^*_{u,v})_{u,v\in\mathcal{M}}$ over $\{0, 1\}$ and an array $(sc^*_u)_{u\in\mathcal{M}}$ over $\{0, \ldots, s\}$, both initialized with 0 everywhere, and an array $(deliver^*_{u,v})_{u,v\in\mathcal{M}}$ of lists, all initially empty.

The state-transition function of $\mathsf{TH}_\mathcal{H}$ is defined by the following rules. Inputs where no rule applies are ignored. If an input triggers an output $\neq \epsilon$ we say that $\mathsf{TH}_\mathcal{H}$ *accepts* this input.

- **Initialization.**
  - **Send initialization:** If $\mathsf{TH}_\mathcal{H}$ receives $(\mathsf{init})$ at $\mathsf{in}_u?$ and $init^*_{u,u} = 0$, it outputs $(\mathsf{initialized}, u)$ at $\mathsf{out_{sim}}!$, 1 at $\mathsf{out_{sim}}^{\triangleleft}!$, and sets $init^*_{u,u} := 1$.
  - **Receive initialization from honest $u$:** If $\mathsf{TH}_\mathcal{H}$ receives $(\mathsf{initialize}, u, v)$ at $\mathsf{in_{sim}}?$, $u, v \in \mathcal{H}$ and if $init^*_{u,v} = 0$, $init^*_{u,u} = 1$, then it sets $init^*_{u,v} := 1$ and outputs $(\mathsf{initialized}, u)$ at $\mathsf{out}_v!$.

[18] The order of the ports in the sequence $Ports_{\mathsf{TH}_\mathcal{H}}$ does not matter below, we can assume any canonical one; similar for the following machines.

– **Receive initialization from dishonest $u$:** If $\mathsf{TH}_{\mathcal{H}}$ receives $(\mathsf{initialize}, u, v)$ at $\mathsf{in}_{\mathsf{sim}}?$, $u \in \mathcal{A}$, $v \in \mathcal{H}$, $init^*_{u,v} = 0$ then it sets $init^*_{u,v} := 1$ and outputs $(\mathsf{initialized}, u)$ at $\mathsf{out}_v!$.

- **Sending and receiving messages.**
  – **Send:** If $\mathsf{TH}_{\mathcal{H}}$ receives an input $(\mathsf{send}, m, v)$ at $\mathsf{in}_u?$ and $m \in \Sigma^+$, $\mathsf{len}(m) \leq L(k)$, $v \in \mathcal{M} \setminus \{u\}$, $init^*_{u,u} = 1$, $init^*_{v,u} = 1$, and $sc^*_u < s$, then it first sets $sc^*_u := sc^*_u + 1$.
  If $v \in \mathcal{H}$, it determines $l := \mathsf{len}(m)$ and $i := \mathsf{size}(deliver^*_{u,v}) + 1$, sets $deliver^*_{u,v}[i] := m$, and outputs $(\mathsf{busy}, u, i, l, v)$ at $\mathsf{out}_{\mathsf{sim}}!$ and 1 at $\mathsf{out}_{\mathsf{sim}}{}^{\triangleleft}!$.
  If $v \in \mathcal{A}$, it outputs $(\mathsf{msg}, u, m, v)$ at $\mathsf{out}_{\mathsf{sim}}!$ and 1 at $\mathsf{out}_{\mathsf{sim}}{}^{\triangleleft}!$.
  – **Receive from honest party $u$:** If $\mathsf{TH}_{\mathcal{H}}$ receives $(\mathsf{select}, u, i, v)$ at $\mathsf{in}_{\mathsf{sim}}?$ and $u, v \in \mathcal{H}$, $init^*_{v,v} = 1$, $init^*_{u,v} = 1$, and $m := deliver^*_{u,v}[i] \neq \downarrow$, then it outputs $(\mathsf{received}, u, m)$ at $\mathsf{out}_v!$.
  – **Receive from dishonest party $u$:** If $\mathsf{TH}_{\mathcal{H}}$ receives $(\mathsf{send}, u, m, v)$ at $\mathsf{in}_{\mathsf{sim}}?$ and $u \in \mathcal{A}$, $m \in \Sigma^+$, $\mathsf{len}(m) \leq L(k)$, $v \in \mathcal{H}$, $init^*_{v,v} = 1$ and $init^*_{u,v} = 1$, then it outputs $(\mathsf{received}, u, m)$ at $\mathsf{out}_v!$.

◇

$\mathsf{TH}_{\mathcal{H}}$ is as abstract as we hoped for: It is deterministic and contains no cryptographic objects at all. Its state-transition function should be easy to express in any formal language for automata provided it allows certain data structures, which most such languages do. We could have been even more abstract by omitting initialization (the abstraction of key exchange) and the bound $s$ (and thus the counters). However, in practice users will have to know about initialization to provide authentic channels, and several cryptographic schemes require upper bounds on the usage. Moreover, one can choose $L$ constant to make $k$ invisible.

## 5.2 The Real System

The real system uses asymmetric encryption and digital signatures as cryptographic primitives; notation for them is briefly introduced in Section 5.2.1. The scheme itself is described in Section 5.2.2.

### 5.2.1 Primitives Used

We will prove a real system based on digital signatures and public-key encryption.

The algorithms $(\mathsf{gen}_\mathsf{S}, \mathsf{sign}, \mathsf{test})$ denote a digital signature scheme secure against existential forgery under adaptive chosen-message attacks [17]. Let the overall message space be $\Sigma^+$ and, w.l.o.g., $\mathsf{false} \notin \Sigma^+$. We write $(sks, pks) \leftarrow \mathsf{gen}_\mathsf{S}(1^k, 1^s)$ for the generation of a secret signing key and a public test key based on a security parameter, $k \in \mathbb{N}$, and the desired maximum number of signatures, $s \in \mathbb{N}$. By $sig \leftarrow \mathsf{sign}_{sks,sc}(m)$ we denote the (probabilistic) signing of a message $m \in \Sigma^+$, where $sc \in \{1, \ldots, s\}$ is a counter value that has to be unique among the signatures generated with $sks$. We assume that $sig$ is of the form $(m, sig')$. Verification $\mathsf{test}_{pks}(sig)$ returns either $m$ (then we say that the signature is valid) or $\mathsf{false}$. We assume that the length of a signature on $m$ is a function $\mathsf{sig\_len}(k, s, \mathsf{len}(m))$.

By $(\mathsf{gen}_\mathsf{E}, \mathsf{E}, \mathsf{D})$, we denote a public-key encryption scheme secure against adaptive chosen-ciphertext attacks, e.g., [11]. We write $(ske, pke) \leftarrow \mathsf{gen}_\mathsf{E}(1^k)$ for the generation of an encryption key and a decryption key. We denote the (probabilistic) encryption of a message $m$ by $c \leftarrow \mathsf{E}_{pke}(m)$, and decryption by $m \leftarrow \mathsf{D}_{ske}(c)$. The result may be $\mathsf{false}$ for wrong ciphertexts. We assume that messages of arbitrary length can be encrypted, but the length need not be hidden, and that the length of $c$ is a function of $k$ and $\mathsf{len}(m)$.

We also need that upper bounds on the length of the results of all the algorithms (e.g., key generation) for any $k$ are efficiently computable.

### 5.2.2 Real System for Secure Message Transmission

**Scheme 5.2 (Real System for Secure Message Transmission)** Let $n, s, L$ and consequently $\mathcal{M}$ be given as in Scheme 5.1. Also let an encryption scheme and a signature scheme be given as above.

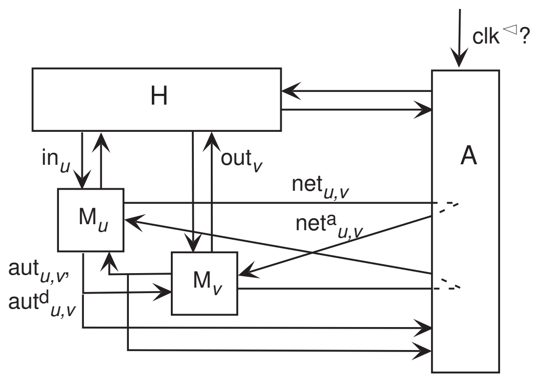


Figure 6: Sketch of real system for secure message transmission. All connections are clocked by A. Indices $\mathcal{H}$ are omitted; also in the following figures.

The system is a standard cryptographic system (Def. 3.1). Thus $M^*$ is a set $\{\mathsf{M}_u | u \in \mathcal{M}\}$ and $S^{*c} := \{\mathsf{in}_u!, \mathsf{out}_u? | u \in \mathcal{M}\}$. Here $\mathcal{ACC}$ is the powerset of $\mathcal{M}$. Hence the system is of the form

$$Sys_{n,s,L}^{\mathsf{secmsg,real}} = \{(M_\mathcal{H}, S_\mathcal{H}) \mid \mathcal{H} \subseteq \mathcal{M}\}$$

with $M_\mathcal{H} = \{\mathsf{M}_{u,\mathcal{H}} | u \in \mathcal{H}\}$.

The ports of machine $\mathsf{M}_u$ are $\{\mathsf{in}_u?, \mathsf{out}_u!\} \cup \{\mathsf{net}_{u,v}!, \mathsf{net}_{v,u}? | v \in \mathcal{M}\} \cup \{\mathsf{aut}_{u,v}!, \mathsf{aut}_{v,u}? | v \in \mathcal{M}\}$. The first ones are for the user, the second ones for normal messages to and from each $\mathsf{M}_v$, and the last ones for key exchange. High-level connections $\{\mathsf{net}_{u,v}!, \mathsf{net}_{u,v}?\}$ are insecure and $\{\mathsf{aut}_{u,v}!, \mathsf{aut}_{u,v}?\}$ are authenticated. The resulting high-level connection graph is sketched in Figure 6.

Each $\mathsf{M}_u$ maintains an array $(init_{v,u})_{v \in \mathcal{M}}$ of lists, which are initially empty, and a counter $sc_u$ initialized with 0. The state-transition function of $\mathsf{M}_u$ is defined by the following rules. Inputs where no rule applies are ignored. If an input triggers an output $\neq \epsilon$ we say that $\mathsf{M}_u$ *accepts* this input.

- **Initialization.**
  - **Send initialization:** If $\mathsf{M}_u$ receives $(\mathsf{init})$ at $\mathsf{in}_u?$ and $init_{u,u} = ()$, then it generates two key pairs, $(sks_u, pks_u) \leftarrow \mathsf{gen_S}(1^k, 1^s)$ and $(ske_u, pke_u) \leftarrow \mathsf{gen_E}(1^k)$, outputs $(pks_u, pke_u)$ at all ports $\mathsf{aut}_{u,v}!$, and sets $init_{u,u} := (sks_u, ske_u)$.
  - **Receive initialization:** If $\mathsf{M}_u$ receives $(pks_v, pke_v)$ (within the maximum length for a pair of public keys) at $\mathsf{aut}_{v,u}?$ and $init_{v,u} = ()$, then it sets $init_{v,u} := (pks_v, pke_v)$ and outputs $(\mathsf{initialized}, v)$ at $\mathsf{out}_u!$.
- **Sending and receiving messages.**
  - **Send:** If $\mathsf{M}_u$ receives an input $(\mathsf{send}, m, v)$ at $\mathsf{in}_u?$, with $m \in \Sigma^+$, $\mathsf{len}(m) \leq L(k)$, and $v \in \mathcal{M} \setminus \{u\}$, and if $init_{u,u} = (sks_u, ske_u)$, $init_{v,u} = (pks_v, pke_v)$, and $sc_u < s$, then it sets $sc_u := sc_u + 1$ and
  $$c \leftarrow \mathsf{E}_{pke_v}(\mathsf{sign}_{sks_u, sc_u}(u, m, v)).$$
  It outputs $c$ at $\mathsf{net}_{u,v}!$.
  - **Receive:** If $\mathsf{M}_u$ receives $c$ at $\mathsf{net}_{v,u}?$ (within the maximum length for a network message as above), and $init_{u,u} = (sks_u, ske_u)$ and $init_{v,u} = (pks_v, pke_v)$, then $\mathsf{M}_u$ tries to parse $c$ in the form $\mathsf{E}_{pke_u}(\mathsf{sign}_{sks_v, sc_v}(v, m, u))$. More precisely:
    a) $sig := \mathsf{D}_{ske_u}(c)$. Abort if the result is $\mathsf{false}$.
    b) $m' := \mathsf{test}_{pks_v}(sig)$. Abort if the result is $\mathsf{false}$.
    c) $(v', m, u') := m'$. Abort if this fails or $u' \neq u$ or $v' \neq v$.

  If this succeeds then $\mathsf{M}_u$ outputs $(\mathsf{received}, v, m)$ at $\mathsf{out}_u!$.

◇

## 5.3 Security of the Real System

We show that Scheme 5.2 is at least as secure as the ideal Scheme 5.1.

**Theorem 5.1** *(Security of Secure Message Transmission)* For all $n, s \in \mathbb{N}$ and $L \in \mathbb{N}[x]$, $Sys_{n,s,L}^{\mathsf{secmsg,real}} \geq_{\mathsf{sec}}^{f,\mathsf{poly}} Sys_{n,s,L}^{\mathsf{secmsg,ideal}}$ for the canonical mapping $f$ from Section 3.1, provided the signature and encryption schemes used are secure. This holds with blackbox simulatability (Def. 2.12). $\square$

The proof of Theorem 5.1 is given in Section 5.3.2. It is based on Theorem 5.2, presented in Section 5.3.1.

### 5.3.1 General Simulatability of Public-key Encryption in a Reactive Multi-user Setting

An essential cryptographic part of the proof of Theorem 5.1 is captured by Theorem 5.2, which extends the standard notion of chosen-ciphertext security of public-key encryption to a reactive multi-user setting, using a simulatability definition. A similar multi-user scenario has been considered in [4], but no decryption request for any of the messages encrypted by a correct machine is allowed there. However, in a reactive scenario like ours, most secret messages are also decrypted by some correct machine and partial knowledge may leak; hence the theorem is not immediately applicable. Hence we define an ideal system $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ that encrypts simulated messages, but honors *all* decryption requests by means of by table look-up of the intended messages. For this we can show simulatability in our usual sense.[19]

**Scheme 5.3 (Ideal and "Real" Encryption Systems)** Let an encryption scheme $(\mathsf{gen_E}, \mathsf{E}, \mathsf{D})$ and parameters $n, s_{keys}, s_{encs} \in \mathbb{N}$ be given, Here $s_{keys}$ denotes the maximum number of keys to be generated in the system and $s_{encs}$ the maximum number of encryptions per key. For every $l \in \mathbb{N}$, let $m_{sim,l}$ denote a fixed message from $\Sigma^l$, say $\mathsf{0}^l$. We define two systems

- $Sys_{n,s_{keys},s_{encs}}^{\mathsf{enc,real}} = \{(\{\mathsf{Enc}_{\mathcal{H}}\}, S_{\mathsf{enc},\mathcal{H}}) \mid \mathcal{H} \subseteq \mathcal{M}\}$,
- $Sys_{n,s_{keys},s_{encs}}^{\mathsf{enc,sim}} = \{(\{\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}\}, S_{\mathsf{enc},\mathcal{H}}) \mid \mathcal{H} \subseteq \mathcal{M}\}$.

For all $\mathcal{H}$, the ports are defined as follows (with an arbitrary order on $Ports_{\mathsf{Enc}_{\mathcal{H}}}$):

- $Ports_{\mathsf{Enc}_{\mathcal{H}}} := Ports_{\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}} := \{\mathsf{in}_{\mathsf{enc},u}?, \mathsf{out}_{\mathsf{enc},u}!, \mathsf{out}_{\mathsf{enc},u}{}^{\triangleleft}! \mid u \in \mathcal{H}\}$,
- $S_{\mathsf{enc},\mathcal{H}}^c := \{\mathsf{in}_{\mathsf{enc},u}!, \mathsf{in}_{\mathsf{enc},u}{}^{\triangleleft}!, \mathsf{out}_{\mathsf{enc},u}? \mid u \in \mathcal{H}\}$.[20]

Both machines have a security parameter, $k$, and maintain a key counter $kc \in \mathbb{N}$, initially $kc = 0$, and initially empty lists *keys* and *ciphers*. The latter is used for the look-up of intended cleartexts in the simulation. The transition functions are given as follows, where we assume that the current input is made at port $\mathsf{in}_{\mathsf{enc,u}}?$. The resulting output goes to $\mathsf{out}_{\mathsf{enc},u}!$, with $\mathsf{out}_{\mathsf{enc},u}{}^{\triangleleft}! := 1$.

- Input $(\mathsf{generate})$ for $\mathsf{Enc}_{\mathcal{H}}$ and $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$: If $kc < s_{keys}$ then $\{$ $kc := kc + 1$; $(ske, pke) \leftarrow \mathsf{gen_E}(1^k)$; add $(u, kc, ske, pke, 0)$ to *keys*; output $pke$ $\}$ else output $\downarrow$.
- Input $(\mathsf{encrypt}, pke, m)$ with $pke, m \in \Sigma^+$ and within the length bounds:
  - for $\mathsf{Enc}_{\mathcal{H}}$: If $\exists v, kc, ske, s_{pke}$: $(v, kc, ske, pke, s_{pke}) \in keys \wedge s_{pke} < s_{encs}$ then $\{$ $s_{pke} := s_{pke}+1$; output $c \leftarrow \mathsf{E}_{pke}(m)$ $\}$ else output $\downarrow$.
  - for $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$: If $\exists v, kc, ske, s_{pke}$: $(v, kc, ske, pke, s_{pke}) \in keys \wedge s_{pke} < s_{encs}$ then $\{$ $s_{pke} := s_{pke} + 1$; output $c \leftarrow \mathsf{E}_{pke}(m_{sim,\mathsf{len}(m)})$; add $(m, pke, c)$ to *ciphers* $\}$ else output $\downarrow$.

[19]For cryptographers, our theorem can also be seen as a formalization of the notion of "a standard hybrid argument." This was not well-defined so far because in an overall reactive system, one has to switch over between a real state and an ideal state, which is not generally defined. In particular, it must be made clear how decryption is handled in the hybrids. This is now well-defined at least for those systems that use an encryption system only such that they can be rewritten with our real encryption system.

[20]Thus $\bar{S}_{\mathsf{enc},\mathcal{H}} = \emptyset$, i.e., the adversary is not connected with the correct machines except via or in the place of $\mathsf{H}$. The fact that inputs are scheduled by $\mathsf{H}$, and outputs for $\mathsf{H}$ are scheduled by the system, allows $\mathsf{H}$ (typically a machine) to use the system like a local subsystem which always returns a result immediately.

- Input $(\mathsf{decrypt}, pke, c)$ with $pke, c \in \Sigma^+$ and within the (efficiently computable) length bounds (note that $pke$ is used as a designator of the desired private key):
  - for $\mathsf{Enc}_{\mathcal{H}}$: If $\exists kc, ske, s_{pke}$: $(u, kc, ske, pke, s_{pke}) \in keys$ then $\{\ m \leftarrow \mathsf{D}_{ske}(c)$; output $m\ \}$ else output $\downarrow$.
  - for $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$: If $\exists kc, ske, s_{pke}$: $(u, kc, ske, pke, s_{pke}) \in keys$ then $\{$ If $\exists m$: $(m, pke, c) \in ciphers$ then output $m$ else output $m \leftarrow \mathsf{D}_{ske}(c)\ \}$ else output $\downarrow$.

◇

Note that $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ limits the capability to decrypt: If $(u, kc, ske, pke, 0)$ was added to $keys$ due to an input $(\mathsf{generate})$ at port $\mathsf{in}_{\mathsf{enc},u}?$, then $(\mathsf{decrypt}, pke, c)$ has an effect only if it is entered at the same port.

**Theorem 5.2** *(General Simulatability of Public-key Encryption)* For all $n, s_{keys}, s_{encs} \in \mathbb{N}$, we have $Sys^{\mathsf{enc,real}}_{n,s_{keys},s_{encs}} \geq^{f,\mathsf{poly}}_{\mathsf{sec}} Sys^{\mathsf{enc,sim}}_{n,s_{keys},s_{encs}}$ for the canonical mapping $f$, provided the encryption scheme used is secure against adaptive chosen-ciphertext attacks. This holds with blackbox simulatability. □

*Proof.* From now on, let $n, s_{keys}, s_{encs}, n$ be fixed, and we omit them as system indices. Recall that the definition of encryption security is repeated (in our notation) in Appendix B. In particular, it concerns only one ciphertext not generated by the adversary. The proof is a hybrid argument, as first used in [16], i.e., we construct intermediate systems that differ only in one encryption each. Let $\mathcal{I} := (\{1, \ldots, s_{keys}\} \times \{1, \ldots, s_{encs}\}) \cup \{\alpha\}$, let $<_{\mathcal{I}}$ be the lexicographic order on $\mathcal{I} \setminus \{\alpha\}$, and $\alpha \leq_{\mathcal{I}} t$ for all $t \in \mathcal{I}$. Let $\mathsf{pred}(t)$ be the predecessor of $t \in \mathcal{I}$ relative to $<_{\mathcal{I}}$, and let $\omega = (s_{keys}, s_{encs})$.

We can now define the hybrid systems $Sys_t = \{(\{\mathsf{Enc}_{t,\mathcal{H}}\}, S_{\mathsf{enc},\mathcal{H}}) \mid \mathcal{H} \subseteq \mathcal{M}\}$ for each $t \in \mathcal{I}$. Each $\mathsf{Enc}_{t,\mathcal{H}}$ is like $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$, except whenever $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ encrypts $m_{sim,\mathsf{len}(m)}$: Let $t' := (kc, s_{pke})$ for the values $kc, s_{pke}$ at that moment (where $s_{pke} \leq s_{encs}$).

- If $t' \leq_{\mathcal{I}} t$, it encrypts $m$ like $\mathsf{Enc}_{\mathcal{H}}$,
- if $t' >_{\mathcal{I}} t$, it encrypts $m_{sim,\mathsf{len}(m)}$ like $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$.

Obviously, $\mathsf{Enc}_{\alpha,\mathcal{H}} = \mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$.

We will now prove that $Sys^{\mathsf{enc,real}} \geq^{\mathsf{perf}}_{\mathsf{sec}} Sys_\omega$ and that $Sys_t \geq^{\mathsf{poly}}_{\mathsf{sec}} Sys_{\mathsf{pred}(t)}$ for all $t >_{\mathcal{I}} \alpha$ (with canonical mappings). By transitivity, this implies Theorem 5.2. (Note that $\mathcal{I}$ is independent of $k$.)

$Sys^{\mathsf{enc,real}} \geq^{\mathsf{perf}}_{\mathsf{sec}} Sys_\omega$**:** Obviously, $\mathsf{Enc}_{\mathcal{H}}$ and $\mathsf{Enc}_{\omega,\mathcal{H}}$ produce identical outputs for inputs $(\mathsf{generate})$ and $(\mathsf{encrypt}, pke, m)$. Consider an input $(\mathsf{decrypt}, pke, c)$ at $\mathsf{in}_{\mathsf{enc},u}?$ such that $\exists kc, ske, s_{pke}$: $(u, kc, ske, pke, s_{pke}) \in keys$ (otherwise both output $\downarrow$). If there is no tuple $(m, pke, c)$ in $ciphers$, both output $\mathsf{D}_{ske}(c)$. If such a triple exists then $c$ has been generated by $\mathsf{Enc}_{\omega,\mathcal{H}}$ as $\mathsf{E}_{pke}(m)$. Thus $m = \mathsf{D}_{ske}(c)$, and both machines output $m$.

$Sys_t \geq^{\mathsf{poly}}_{\mathsf{sec}} Sys_{\mathsf{pred}(t)}$**:** Assume that this is not true for a certain $t >_{\mathcal{I}} \alpha$. Let $t =: (kc, s')$. Let $\mathcal{H}$, $\mathsf{H}$ and $\mathsf{A}$ be such that the views of $\mathsf{H}$ in $(\mathsf{Enc}_{\mathsf{pred}(t),\mathcal{H}}, S_{\mathsf{enc},\mathcal{H}}, \mathsf{H}, \mathsf{A})$ and $(\mathsf{Enc}_{t,\mathcal{H}}, S_{\mathsf{enc},\mathcal{H}}, \mathsf{H}, \mathsf{A})$ are distinguishable. (We use the same adversary $\mathsf{A}$ on both sides, which is a trivial blackbox construction.)

From this we construct an adversary $\mathsf{A}_{\mathsf{enc}}$ on the encryption scheme: $\mathsf{A}_{\mathsf{enc}}$ receives a correctly chosen public key $pke$ and interacts with a decryption oracle $\mathsf{Dec}$. Basically, $\mathsf{A}_{\mathsf{enc}}$ simulates $(\mathsf{Enc}_{t,\mathcal{H}}, S_{\mathsf{enc},\mathcal{H}}, \mathsf{H}, \mathsf{A})$, but with a few exceptions:

- If $\mathsf{H}$ makes the $kc$-th input $(\mathsf{generate})$, and via $\mathsf{in}_{\mathsf{enc},u}?$, then $\mathsf{A}_{\mathsf{enc}}$ only adds $(u, kc, 0, pke, 0)$ to $keys$. For all decryptions corresponding to this entry, i.e., $pke$, the decryption oracle $\mathsf{Dec}$ is used.
- If $\mathsf{H}$ makes an input $(\mathsf{encrypt}, pke, m)$ which gets the index $t = (kc, s')$, i.e., it finds the entry $(u, s, 0, pke, s' - 1)$ in $keys$, then $\mathsf{A}_{\mathsf{enc}}$ sends $(m_0, m_1) := (m, m_{sim,\mathsf{len}(m)})$ to $\mathsf{Dec}$. This oracle flips a bit $b \in_{\mathcal{R}} \{0, 1\}$ and returns $c \leftarrow \mathsf{E}_{pke}(m_b)$. $\mathsf{A}_{\mathsf{enc}}$ adds $(m, pke, c)$ to $ciphers$.

Note that $\mathsf{Dec}$ is never asked to decrypt $c$, as $\mathsf{A}_{\mathsf{enc}}$ will find $(m, pke, c)$ in $ciphers$ and output $m$. For $b = 0$ the simulated run is generated like a run of $(\mathsf{Enc}_{t,\mathcal{H}}, S_{\mathsf{enc},\mathcal{H}}, \mathsf{H}, \mathsf{A})$, and for $b = 1$ like a run of $(\mathsf{Enc}_{\mathsf{pred}(t),\mathcal{H}}, S_{\mathsf{enc},\mathcal{H}}, \mathsf{H}, \mathsf{A})$. By assumption the views of $\mathsf{H}$ in both structures are distinguishable. This can immediately be converted into an advantage for $\mathsf{A}_{\mathsf{enc}}$ in guessing $b$. ■

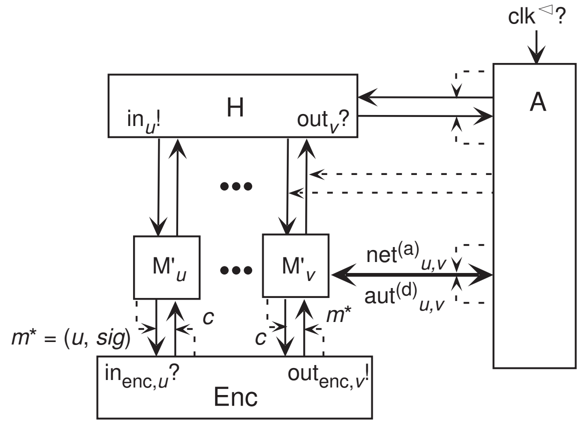


Figure 7: Real system rewritten with encryption subsystem

### 5.3.2 Overall Proof of Theorem 5.1

We prove Theorem 5.1 in the following steps: In Part A we rewrite the real system to use a reactive encryption system $Sys^{\mathsf{enc,real}}_{n,s_{keys},s_{encs}}$ from Scheme 5.3 instead of performing the original encryption algorithms. This gives an intermediate system $Sys^{\mathsf{secmsg,Enc}}_{n,s,L}$, and we show that

$$Sys^{\mathsf{secmsg,real}}_{n,s,L} \geq_{\mathsf{sec}} Sys^{\mathsf{secmsg,Enc}}_{n,s,L}.$$

In Part B, we replace each machine $\mathsf{Enc}_{\mathcal{H}}$ by $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ to get a new system $Sys^{\mathsf{secmsg,Encsim}}_{n,s,L}$. We show that

$$Sys^{\mathsf{secmsg,Enc}}_{n,s,L} \geq_{\mathsf{sec}} Sys^{\mathsf{secmsg,Encsim}}_{n,s,L}.$$

In Part C we define a simulator $\mathsf{Sim}_{\mathcal{H}}$ that uses an adversary from a configuration of $Sys^{\mathsf{secmsg,Encsim}}_{n,s,L}$ as a blackbox, resulting in a configuration of the ideal system $Sys^{\mathsf{secmsg,ideal}}_{n,s,L}$. We show that this simulation is correct, i.e., that

$$Sys^{\mathsf{secmsg,Encsim}}_{n,s,L} \geq_{\mathsf{sec}} Sys^{\mathsf{secmsg,ideal}}_{n,s,L}.$$

Theorem 5.1 follows by transitivity of $\geq_{\mathsf{sec}}$.

**Part A, Intermediate System: Real System Using Encryption Subsystem.** We first define the intermediate system $Sys^{\mathsf{secmsg,Enc}}_{n,s,L}$. Its structures are of the form $(M'_{\mathcal{H}}, S_{\mathcal{H}})$ with $M'_{\mathcal{H}} = \{\mathsf{Enc}_{\mathcal{H}}\} \cup \{\mathsf{M}'_{u,\mathcal{H}} | u \in \mathcal{H}\}$, see Fig. 7.[21] The parameter $n$ of the encryption subsystem is as in the overall system and $s_{keys} := n$, $s_{encs} := ns$. Each machine $\mathsf{M}'_{u,\mathcal{H}}$ equals $\mathsf{M}_{u,\mathcal{H}}$ with the following modifications:

- It has additional ports $\mathsf{in}_{\mathsf{enc},u}!$, $\mathsf{in}_{\mathsf{enc},u}{}^{\triangleleft}!$, $\mathsf{out}_{\mathsf{enc},u}?$ to connect to $\mathsf{Enc}_{\mathcal{H}}$.
- In "Send initialization." Instead of calling $\mathsf{gen}_{\mathsf{E}}$, $\mathsf{M}'_{u,\mathcal{H}}$ outputs ($\mathsf{generate}$) at $\mathsf{in}_{\mathsf{enc},u}!$ and 1 at $\mathsf{in}_{\mathsf{enc},u}{}^{\triangleleft}!$ and waits until it receives a result at $\mathsf{out}_{\mathsf{enc},u}?$.[22] It uses this result as $pke_u$. In $init_{u,u}$ it stores $pke_u$ instead of $ske_u$.
- In "Send." If $v \in \mathcal{H}$, instead of the computation of $c$, it computes $sig \leftarrow \mathsf{sign}_{sks_u,sc_u}(u, m, v)$ and outputs $(\mathsf{encrypt}, pke_v, sig)$ at $\mathsf{in}_{\mathsf{enc},u}!$ and 1 at $\mathsf{in}_{\mathsf{enc},u}{}^{\triangleleft}!$. It waits until it receives a result at $\mathsf{out}_{\mathsf{enc},u}?$, which it uses as $c$.
- In "Receive." $\mathsf{M}'_{u,\mathcal{H}}$ replaces Step a) of parsing by an output $(\mathsf{decrypt}, pke_u, c)$ at $\mathsf{in}_{\mathsf{enc},u}!$ and 1 at $\mathsf{in}_{\mathsf{enc},u}{}^{\triangleleft}!$ . It waits until it receives a result at $\mathsf{out}_{\mathsf{enc},u}?$, which it uses as $sig$.

[21] This intermediate system is not "real" and only used in our proof; hence it is no problem that $\mathsf{M}'_{u,\mathcal{H}}$ makes explicit distinctions using $\mathcal{H}$ below.

[22] More precisely it enters a state $(\mathsf{wait}, \mathsf{init})$ where it only reacts on this input. As one easily sees from the scheduling that only this input can arrive next, we treat the wait state together with the previous state; also in the following cases.

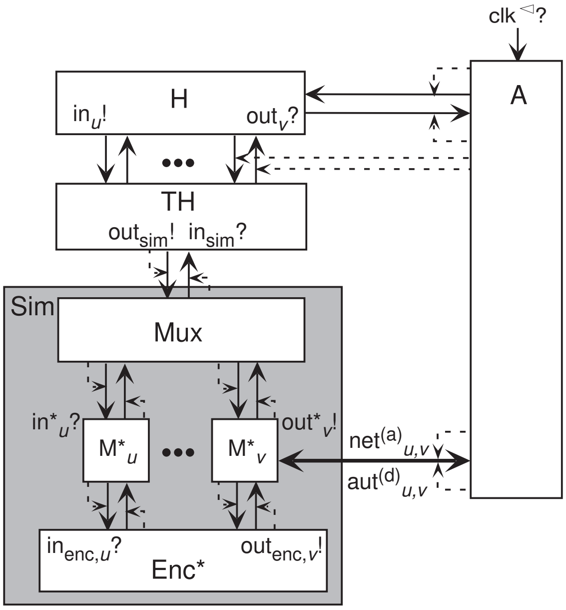


Figure 8: Simulator for secure message transmission

The views of $\mathsf{A}$ and $\mathsf{H}$ in a configuration $(M_{\mathcal{H}}, S_{\mathcal{H}}, \mathsf{H}, \mathsf{A})$ and $(M'_{\mathcal{H}}, S_{\mathcal{H}}, \mathsf{H}, \mathsf{A})$ are clearly identical (perfectly indistinguishable), because the actions of $\mathsf{Enc}_{\mathcal{H}}$ on the given inputs are precisely what $\mathsf{M}_{u,\mathcal{H}}$ would have done at this point. (In particular $s_{keys}$ is not exceeded because each $\mathsf{M}_{u,\mathcal{H}}$ inputs $(\mathsf{generate})$ at most once, controlled by $init_{u,u}$, and $s_{encs}$ is not exceeded for any key because each $\mathsf{M}_{u,\mathcal{H}}$ inputs $(\mathsf{encrypt}, \ldots)$ at most $s$ times.) The interactions between $\mathsf{Enc}_{\mathcal{H}}$ and the $\mathsf{M}'_{u,\mathcal{H}}$ are invisible for $\mathsf{A}$. Thus we have

$$Sys^{\mathsf{secmsg,real}}_{n,s,L} \geq_{\mathsf{sec}} Sys^{\mathsf{secmsg,Enc}}_{n,s,L}.$$

**Part B, Replacing the Encryption System.** In $Sys^{\mathsf{secmsg,Enc}}_{n,s,L}$, we want to replace each machine $\mathsf{Enc}_{\mathcal{H}}$ by $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ to get a new system $Sys^{\mathsf{secmsg,Encsim}}_{n,s,L}$.

We consider $Sys^{\mathsf{secmsg,Enc}}_{n,s,L}$ as a composition of $Sys_0 := Sys^{\mathsf{enc,real}}_{n,s_{keys},s_{encs}}$ and a system $Sys_1$ that is naturally defined as the structures without $\mathsf{Enc}_{\mathcal{H}}$: The specified ports are those of $Sys^{\mathsf{secmsg,real}}_{n,s,L}$ plus the complements of the ports of $\mathsf{Enc}_{\mathcal{H}}$. Then the conditions of Definition 4.1 are fulfilled. As each machine $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ has the same ports as $\mathsf{Enc}_{\mathcal{H}}$, the definition of $Sys^{\mathsf{secmsg,Encsim}}_{n,s,L}$ as a composition of $Sys'_0 := Sys^{\mathsf{enc,sim}}_{n,s_{keys},s_{encs}}$ and the same $Sys_1$ is clear and the preconditions of the composition theorem, Theorem 4.1, are fulfilled. In particular, all machines $\mathsf{M}'_{u,\mathcal{H}}$ are polynomial-time in $k$. Hence Theorem 5.2 and Theorem 4.1 imply $Sys^{\mathsf{secmsg,Enc}}_{n,s,L} \geq_{\mathsf{sec}} Sys^{\mathsf{secmsg,Encsim}}_{n,s,L}$ (again with blackbox simulatability).

**Part C, Simulator.** It remains to be shown that $Sys^{\mathsf{secmsg,Encsim}}_{n,s,L} \geq_{\mathsf{sec}} Sys^{\mathsf{secmsg,ideal}}_{n,s,L}$. Intuitively, one remaining aspect is to show that the real messages $m$, which are still inputs to $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$, but which are not output by $\mathsf{TH}_{\mathcal{H}}$, are indeed not needed. This is a perfectly indistinguishable rewriting. The other aspect is to show that the use of signatures guarantees authenticity as specified in the ideal system.

**Simulator $\mathsf{Sim}_{\mathcal{H}}$:** We construct $\mathsf{Sim}_{\mathcal{H}}$ as the combination of several machines, see Figure 8. It uses the given $\mathsf{A}$ as a submachine, without any port renaming. (Although all figures show $\mathsf{H}$ as using all the specified ports, the proof is general.)

- $\mathsf{Mux}$ acts as a multiplexer/demultiplexer between $\mathsf{TH}_{\mathcal{H}}$ and other machines. It has ports $\mathsf{in_{sim}}!$, $\mathsf{in_{sim}}^{\triangleleft}!$, $\mathsf{out_{sim}}?$, and $\mathsf{in}^*_u!$, $\mathsf{in}^{*\triangleleft}_u!$ and $\mathsf{out}^*_u?$ for each $u \in \mathcal{H}$. Basically, it translates from the "top" to the "bottom" and vice versa, immediately scheduling its outputs so that the values simply "pass through":

- Input $(\mathsf{initialized}, u)$ at $\mathsf{out_{sim}}?$ / Output $(\mathsf{init})$ at $\mathsf{in}_u^*!$ and 1 at $\mathsf{in}_u^{*\triangleleft}!$.
- Input $(\mathsf{busy}, u, i, l, v)$ at $\mathsf{out_{sim}}?$ / Output $(\mathsf{send\_blindly}, i, l, v)$ at $\mathsf{in}_u^*!$ and 1 at $\mathsf{in}_u^{*\triangleleft}!$.
- Input $(\mathsf{msg}, u, m, v)$ at $\mathsf{out_{sim}}?$ / Output $(\mathsf{send}, m, v)$ at $\mathsf{in}_u^*!$ and 1 at $\mathsf{in}_u^{*\triangleleft}!$.
- Input $(\mathsf{initialized}, u)$ at $\mathsf{out}_v^*?$ / Output $(\mathsf{initialize}, u, v)$ at $\mathsf{in_{sim}}!$ and 1 at $\mathsf{in_{sim}}^{\triangleleft}!$.
- Input $(\mathsf{received\_blindly}, u, i)$ at $\mathsf{out}_v^*?$ / Output $(\mathsf{select}, u, i, v)$ at $\mathsf{in_{sim}}!$ and 1 at $\mathsf{in_{sim}}^{\triangleleft}!$.
- Input $(\mathsf{received}, u, m)$ at $\mathsf{out}_v^*?$ / Output $(\mathsf{send}, u, m, v)$ at $\mathsf{in_{sim}}!$ and 1 at $\mathsf{in_{sim}}^{\triangleleft}!$.

- $\mathsf{M}^*_{u,\mathcal{H}}$, for $u \in \mathcal{H}$, equals $\mathsf{M}'_{u,\mathcal{H}}$ with the following modifications:
  - It has an additional port $\mathsf{out}_u^{*\triangleleft}!$ which it sets to 1 for every output at $\mathsf{out}_u^*!$ (so that the outputs "pass up" through $\mathsf{Mux}$).
  - Upon input $(\mathsf{send\_blindly}, i, l, v)$ at $\mathsf{in}_u^*?$, it behaves like $\mathsf{M}'_{u,\mathcal{H}}$ on input $(\mathsf{send}, m, v)$ with $\mathsf{len}(m) = l$, but instead of computing $sig \leftarrow \mathsf{sign}_{sks_u, sc_u}(u, m, v)$ it just computes the length $l^*$ of any such $sig$ using the algorithm $\mathsf{sig\_len}$. It outputs $(\mathsf{encrypt\_blindly}, pke_v, (u, i, v), l^*)$ at $\mathsf{in}_{\mathsf{enc},u}!$ and 1 at $\mathsf{in}_{\mathsf{enc},u}{}^{\triangleleft}!$. Then it waits and continues like $\mathsf{M}'_{u,\mathcal{H}}$.
  - In "receive", it acts like $\mathsf{M}'_{u,\mathcal{H}}$ until it receives a result at $\mathsf{out}_{\mathsf{enc},u}?$. If this is of the form $(\mathsf{decrypted}, sig)$, then $\mathsf{M}^*_{u,\mathcal{H}}$ treats it as $\mathsf{M}'_{u,\mathcal{H}}$ treats $sig$. If it is of the form $(\mathsf{decrypted\_blindly}, (v', i, u'))$, then $\mathsf{M}^*_{u,\mathcal{H}}$ verifies that $v' = v$ and $u' = u$. If yes, it outputs $(\mathsf{received\_blindly}, v, i)$ at $\mathsf{out}_u^*!$.

- $\mathsf{Enc}^*_{\mathcal{H}}$ equals $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ with the following modification:
  - An array *blind_ciphers* replaces *ciphers*.
  - Instead of inputs $(\mathsf{encrypt}, \ldots)$, it accepts inputs $(\mathsf{encrypt\_blindly}, pke, mid, l^*)$. Here $mid \in \Sigma^*$ is a message identifier. If it finds the desired tuple in *keys* (otherwise the result is $\downarrow$), it behaves like $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ on input $(\mathsf{encrypt}, pke, m)$ with $\mathsf{len}(m) = l^*$, and it stores $(mid, pke, c)$ in *blind_ciphers*.
  - In "decrypt", it looks for a tuple $(mid, pke, c)$ in *blind_ciphers*. If it finds one, it outputs $(\mathsf{decrypted\_blindly}, mid)$. Otherwise, it decrypts and outputs the result $m$ as $(\mathsf{decrypted}, m)$.

To prove the correctness of this simulator we have to compare configurations $conf_{\mathsf{sr}} := (M'_{\mathcal{H}}, S_{\mathcal{H}}, \mathsf{H}, \mathsf{A})$ and $conf_{\mathsf{id}} := (\{\mathsf{TH}_{\mathcal{H}}\}, S_{\mathcal{H}}, \mathsf{H}, \mathsf{Sim}_{\mathcal{H}}(\mathsf{A}))$, called semi-real and ideal configuration. The overall idea of this proof is to define a mapping $\phi$ from runs of $conf_{\mathsf{sr}}$ to runs of $conf_{\mathsf{id}}$, except for negligible subsets on both sides; we call them "error sets". $\phi$ must respect probabilities and the views of $\mathsf{A}$ and $\mathsf{H}$ in runs $\rho$ and $\phi(\rho)$ must be equal. In our case, we can simply define $\phi$ state-wise, and only for states before and after switching steps of $\mathsf{H}$ and $\mathsf{A}$; this is sufficient because only the views of $\mathsf{H}$ and $\mathsf{A}$ must be identical.

We show that $\phi(\delta_{\mathsf{sr}}(\sigma_{\mathsf{sr}})) = \delta_{\mathsf{id}}(\phi(\sigma_{\mathsf{sr}}))$ for all states $\sigma_{\mathsf{sr}}$ reachable in $conf_{\mathsf{sr}}$, except for the error sets. Here $\delta_{\mathsf{sr}}$ and $\delta_{\mathsf{id}}$ denote the overall probabilistic transition functions. The error sets will consist of the runs where the adversary successfully forges a signature.

**Mapping $\phi$:** Let a state $\sigma_{\mathsf{sr}}$ of $conf_{\mathsf{sr}}$ be given; we define the components of $\sigma_{\mathsf{id}} := \phi(\sigma_{\mathsf{sr}})$. Large parts of the mapping are trivial:

- The states of $\mathsf{H}$ and $\mathsf{A}$ are mapped identically.
- The states of all buffers with the same name in both systems are mapped identically, and so is the scheduled port.[23]
- The remaining buffers in $\sigma_{\mathsf{id}}$ are always empty.[24]
- Security parameters are mapped identically and not mentioned again.

[23] These are $\widetilde{\mathsf{in}_u}$, $\widetilde{\mathsf{out}_u}$, $\widetilde{\mathsf{in}_{\mathsf{enc},u}}$, $\widetilde{\mathsf{out}_{\mathsf{enc},u}}$, $\widetilde{\mathsf{net}_{u,v}}$, $\widetilde{\mathsf{net}^{\mathsf{a}}_{v,u}}$, $\widetilde{\mathsf{aut}_{u,v}}$, $\widetilde{\mathsf{aut}^{\mathsf{d}}_{u,v}}$, $\widetilde{\mathsf{aut}_{v,u}}$ for all $u \in \mathcal{H}, v \in \mathcal{M}$, and the buffers between $\mathsf{H}$ and $\mathsf{A}$.

[24] Recall that we only map states before or after $\mathsf{H}$ or $\mathsf{A}$ switches. The buffers are $\widetilde{\mathsf{in_{sim}}}$, $\widetilde{\mathsf{out_{sim}}}$ and $\widetilde{\mathsf{in}_u^*}$, $\widetilde{\mathsf{out}_u^*}$ for all $u \in \mathcal{H}$.

Now we map the joint states of $\mathsf{M}'_{u,\mathcal{H}}$ for all $u \in \mathcal{H}$ and $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ to states of $\mathsf{TH}_{\mathcal{H}}$ and $\mathsf{Sim}_{\mathcal{H}}$. The state of $\mathsf{Sim}_{\mathcal{H}}$ consists of those of $\mathsf{Mux}$, the machines $\mathsf{M}^*_{u,\mathcal{H}}$, and $\mathsf{Enc}^*_{\mathcal{H}}$.

- $\mathsf{Mux}$ is state-less.
- Key-related variables:
  - The array $init^*$ of $\mathsf{TH}_{\mathcal{H}}$ is derived from the arrays $init_{\bullet,u}$ of the machines $\mathsf{M}'_{u,\mathcal{H}}$: We set $init^*_{v,u} := 1$ whenever $init_{v,u} \neq ()$, else $init^*_{v,u} := 0$.
  - Each array $init_{\bullet,u}$ of a machine $\mathsf{M}^*_{u,\mathcal{H}}$ equals that in $\mathsf{M}'_{u,\mathcal{H}}$.
  - The counter $kc$ and the list $keys$ of $\mathsf{Enc}^*_{\mathcal{H}}$ equal those in $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$.
- Message-related variables: In the semi-real configuration, $\mathsf{M}'_{u,\mathcal{H}}$ contains a counter $sc_u$, and $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ the list $ciphers$. In the ideal configuration, $\mathsf{TH}_{\mathcal{H}}$ contains counters $sc^*_u$ and an array $deliver^*$, while $\mathsf{M}^*_{u,\mathcal{H}}$ contains counters $sc_u$, and $\mathsf{Enc}^*_{\mathcal{H}}$ the list $blind\_ciphers$.
  - The counters $sc_u$ in $\mathsf{M}^*_{u,\mathcal{H}}$ and $sc^*_u$ in $\mathsf{TH}_{\mathcal{H}}$ equal $sc_u$ in $\mathsf{M}'_{u,\mathcal{H}}$.
  - Each entry $e = ciphers[j] \neq \downarrow$ is of the form $e = (sig, pke, c)$ and $sig$ of the form $((u, m, v), sig')$ with $u, v \in \mathcal{H}$. (This is shown as Invariant 3 below.) Given $ciphers$, let $\mathsf{ind}_{u,v}(j)$ denote the number of entries up to (and including) $ciphers[j]$ with the given values $u, v$. Hence for each such entry we can set $i := \mathsf{ind}_{u,v}(j)$ and
    - $blind\_ciphers[j] := ((u, i, v), pke, c)$ and
    - $deliver^*_{u,v}[i] = m$.

The proof of the correctness of this mapping is a relatively straightforward but tedious exercise.

### 5.3.3 Detailed Proof of the Simulation in Part C

We have to show that the mapping $\phi$ has the properties required above. For this, we define some invariants. The first three are formulas valid in all states $\sigma_{\mathsf{sr}}$ reachable in $conf_{\mathsf{sr}}$ (before or after $\mathsf{H}$ or $\mathsf{A}$ switch). The last one is a run invariant of $conf_{\mathsf{id}}$. All invariants are proved below together with the correctness of $\phi$.

**Invariants:**

1. *Buffer emptyness.* All buffers $\widetilde{\mathsf{in}_{\mathsf{enc},u}}$ and $\widetilde{\mathsf{out}_{\mathsf{enc},u}}$ are empty.
2. *Key and init consistency.* Each list $init_{u,u}$ with $u \in \mathcal{H}$ is empty or a pair, whose entries we then denote by $(sks_u, pke_u)$. For $u \neq v, v \in \mathcal{H}$, each list $init_{u,v}$ with $u \in \mathcal{M}$ and each message in a buffer $\widetilde{\mathsf{aut}_{u,v}}$ with $u \in \mathcal{H}$ is empty or a pair, which we denote by $(pks_u, pke_u)$.[25] If $u \in \mathcal{H}$, then $pks_u$ forms a correct signature key pair with $sks_u$ (in particular, then $init_{u,u} \neq \emptyset$), and $pke_u$ also occurs in $keys$ of $\mathsf{Enc}_{\mathsf{Sim},\mathcal{H}}$ (with the given $u$ and a correct decryption key).
3. *Ciphertext format.* Each entry in $ciphers$ (of $\mathsf{Enc}_{\mathsf{Sim},\mathcal{H}}$) is of the form $(sig, pke, c)$, where $sig$ of the form $((u, m, v), sig')$ with $u, v \in \mathcal{H}$ and a valid signature with $sks_u$ (from $init_{u,u}$).
4. *Signatures.* The only usage that $\mathsf{M}^*_{u,\mathcal{H}}$ makes of $sks_u$ is to sign triples $(u, m, v)$ with the given $u$ and $v \in \mathcal{A}$.

[25]The multiple use of these variables for all $v$ is consistent if $u \in \mathcal{H}$, but for the present purpose this does not matter and we can see $v$ as clear from the context.

**Send initialization.** Upon input (init) at $\mathsf{in}_u?$, both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{TH}_\mathcal{H}$ first check whether $init^*_{u,u}/init_{u,u}$ signals a previous initialization. $\phi$ ensures identical results. If yes, both do nothing.

Otherwise, in $conf_{\mathsf{sr}}$, $\mathsf{M}'_{u,\mathcal{H}}$ outputs (generate) at $\mathsf{in}_{\mathsf{enc},u}!$, scheduling it immediately.

In $conf_{\mathsf{id}}$, $\mathsf{TH}_\mathcal{H}$ sets $init^*_{u,u} := 1$ and sends $(\mathsf{initialized}, u)$ to Mux, scheduling it at once. (As the buffer was empty before by $\phi$, this is indeed the delivered message.) Mux sends (init) to $\mathsf{M}^*_{u,\mathcal{H}}$, again scheduling this input at once. As $\phi$ gives $\mathsf{M}^*_{u,\mathcal{H}}$ the same state as $\mathsf{M}'_{u,\mathcal{H}}$, $\mathsf{M}^*_{u,\mathcal{H}}$ behaves exactly like $\mathsf{M}'_{u,\mathcal{H}}$.

Hence now both $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ and $\mathsf{Enc}^*_\mathcal{H}$ obtain an input (generate) at $\mathsf{in}_{\mathsf{enc},u}?$. They behave identically for it. They always make an output $pke$ (we already showed in Part A that $s_{keys}$ is not exceeded) and schedule it.

Hence $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ switch again with this input. Both use it as $pke_u$, store it within $init_{u,u}$, and output $(pks_u, pke_u)$ at all ports $\mathsf{aut}_{u,v}!$. They make no scheduling output, hence control returns to A.

Invariants 1 and 2 could be affected, but clearly remain satisfied. For $\phi$, only the key-related variables can be affected, and the relations also clearly remain satisfied.

**Receive initialization.** Upon input $(pks_v, pke_v)$ at $\mathsf{aut}_{v,u}?$, both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ check whether $init_{v,u} = ()$. If not, they do nothing.

If yes, they set $init_{v,u} := (pks_v, pke_v)$ and output $(\mathsf{initialized}, v)$. Additionally, $\mathsf{M}^*_{u,\mathcal{H}}$ schedules this output. Mux translates it into $(\mathsf{initialize}, v, u)$ and also schedules it. $\mathsf{TH}_\mathcal{H}$ first tests if $init^*_{v,u} = 0$; this is true by $\phi$. If $v \in \mathcal{H}$ it also tests $init^*_{v,v} = 1$. By $\phi$, this is equivalent to $init_{v,v} \neq ()$, and by Invariant 2, this follows from the existence of $pks_v$ in the buffer $\widetilde{\mathsf{aut}_{v,u}}$ before this step.

Thus $\mathsf{TH}_\mathcal{H}$ accepts this input, sets $init^*_{v,u} := 1$, and outputs $(\mathsf{initialized}, v)$.

Invariant 2 could be affected, but remains satisfied. For the case $v \in \mathcal{H}$ this is true because the keys now in $init_{v,u}$ were already in $\widetilde{\mathsf{aut}_{v,u}}$. Only the key-related parts of $\phi$ are affected, and they clearly remain satisfied.

**Send to honest party.** Upon input $(\mathsf{send}, m, v)$ at $\mathsf{in}_u?$ with $u, v \in \mathcal{H}$, both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{TH}_\mathcal{H}$ first consider $init/init^*$, with identical results (by $\phi$). Then they check and possibly increase the counters $sc_u/sc^*_u$, also with the same result. Now assume they continue.

In $conf_{\mathsf{sr}}$, $\mathsf{M}'_{u,\mathcal{H}}$ generates a real signature $sig$ for the message $(u, m, v)$, outputs $(\mathsf{encrypt}, pke_v, sig)$ at $\mathsf{in}_{\mathsf{enc},u}!$, and schedules it.

In $conf_{\mathsf{id}}$, $\mathsf{TH}_\mathcal{H}$ adds $m$ to $deliver^*_{u,v}$; let $i_{new}$ denote its index. It outputs and schedules $(\mathsf{busy}, u, i_{new}, l, v)$ for $l := \mathsf{len}(m)$. Mux transforms it into $(\mathsf{send\_blindly}, i_{new}, l, v)$ for $\mathsf{M}^*_{u,\mathcal{H}}$ and schedules this. By $\phi$, $\mathsf{M}^*_{u,\mathcal{H}}$ starts with the same state as $\mathsf{M}'_{u,\mathcal{H}}$. It makes the same checks and counter updates as $\mathsf{M}'_{u,\mathcal{H}}$. Then it computes a value $l^*$ which by definition equals $\mathsf{len}(sig)$, and outputs and schedules $(\mathsf{encrypt\_blindly}, pke_v, (u, i_{new}, v), l^*)$.

Now $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ and $\mathsf{Enc}^*_\mathcal{H}$ switch with these encryption requests. By Invariant 2 and $\phi$, both find an entry for $pke_v$, and by Part A, $s_{encs}$ is not exceeded for it. Hence $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ generates $c \leftarrow \mathsf{E}_{pke_v}(m_{sim,\mathsf{len}(sig)})$ and stores $(sig, pke_v, c)$ in $ciphers$. $\mathsf{Enc}^*_\mathcal{H}$ generates $c \leftarrow \mathsf{E}_{pke_v}(m_{sim,l^*})$ and stores $((u, i_{new}, v), pke_v, c)$ in $blind\_ciphers$. Hence $c$ has precisely the same distribution. Both output and schedule $c$.

Finally, $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ output $c$ at $\mathsf{net}_{u,v}!$. Control returns to A.

Invariants 1, 3 and 4 could be affected, but clearly remain satisfied. (In particular, $\mathsf{M}^*_{u,\mathcal{H}}$ makes no signature at all here.) As to $\phi$, the only non-trivial part is the mapping of the new $ciphers$ to $blind\_ciphers$ and $deliver^*$: Let $j := \mathsf{size}(ciphers)$ before this step, $i_{old} := \mathsf{ind}_{u,v}[j]$, and $i_{new} := i_{old} + 1$. Then $blind\_ciphers$ was also of size $j$ and $deliver^*_{u,v}$ of length $i_{old}$ (by $\phi$). The new entry is $ciphers[j+1]$, and $\phi$ maps it to $blind\_ciphers[j+1] := ((u, i_{new}, v), pke_v, c)$ and $deliver^*_{u,v}[i_{new}] = m$. These are indeed the new entries in $blind\_ciphers$ and $deliver^*_{u,v}$.

**Send to dishonest party.** Upon input $(\mathsf{send}, m, v)$ at $\mathsf{in}_u?$ with $u \in \mathcal{H}, v \in \mathcal{A}$, both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{TH}_\mathcal{H}$ first consider $init/init^*$, with identical results. Then they check and possibly increase the counters $sc_u/sc^*_u$, also with the same result.

In $conf_{\mathsf{id}}$, $\mathsf{TH}_\mathcal{H}$ outputs and schedules $(\mathsf{msg}, u, m, v)$, and Mux transforms this into $(\mathsf{send}, m, v)$ and schedules it. $\mathsf{M}^*_{u,\mathcal{H}}$ also treats $init$ and $sc_u$ as $\mathsf{M}'_{u,\mathcal{H}}$.

Now both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ compute $c \leftarrow \mathsf{E}_{pke_v}(\mathsf{sign}_{sks_u,sc_u}(u, m, v))$ and output it at $\mathsf{net}_{u,v}!$. Control returns to A.

Invariant 4 could be affected, but clearly remains satisfied. As to $\phi$, only counters are modified and clearly in a consistent way.

**Receive from honest party.** Upon input $c$ at $\mathsf{net}^{\mathsf{a}}_{v,u}?$ with $u, v \in \mathcal{H}$, both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ first check *init*, with the same results. If they continue, they both start to parse $c$ by outputting $(\mathsf{decrypt}, \mathit{pke}_u, c)$ at $\mathsf{in}_{\mathsf{enc},u}!$ and scheduling it.

Both $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ and $\mathsf{Enc}^*_{\mathcal{H}}$ first search for an entry $(u, \cdot, \mathit{ske}_u, \mathit{pke}_u, \cdot) \in \mathit{keys}$ with some $\mathit{ske}_u$; they find it by Invariant 2 (and $\phi$). Then they search *ciphers* and *blind_ciphers*, respectively, for an entry $(x, \mathit{pke}_u, c)$, where $x$ is now called *sig* in $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ and *mid* in $\mathsf{Enc}^*_{\mathcal{H}}$. By $\phi$, either both succeed or none.

- If no such entry is found (intuitively, $c$ was generated by the adversary), both set $\mathit{sig} \leftarrow \mathsf{D}_{\mathit{ske}_u}(m')$. $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ outputs *sig* and $\mathsf{Enc}^*_{\mathcal{H}}$ outputs $(\mathsf{decrypted}, \mathit{sig})$, and both schedule this output.

  Now $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ continue parsing in the same way: They set $m' := \mathsf{test}_{\mathit{pks}_v}(\mathit{sig})$ and try to write $m'$ as $(v, m, u)$ for the given $u, v$ and arbitrary $m$. If this does not succeed, they abort and control returns to $\mathsf{A}$.

  If it succeeds, the simulation would fail, and we put the run of $\mathit{conf}_{\mathsf{id}}$ into a set $\mathit{Forgeries}_{v,k}$ (where $k$ is the security parameter). We call *sig* the "designated signature" for this run. One can easily verify during a run whether this case occurs. By Invariant 4, $\mathsf{M}^*_{v,\mathcal{H}}$ never signed $m'$ because $u \notin \mathcal{A}$.

- Now assume that such entries are found with index $j$. By Invariant 3, *sig* is of the form $\mathit{sig} = ((v', m, u'), \mathit{sig}')$ for some $m$ and $v', u' \in \mathcal{H}$ and a valid signature with $\mathit{sks}_{v'}$. By $\phi$, we have $\mathit{mid} = (v', i, u')$ with $i = \mathsf{ind}_{v',u'}(j)$ and $\mathit{deliver}^*_{v',u'}[i] = m$.

  Finding these entries, $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ outputs *sig* and $\mathsf{Enc}^*_{\mathcal{H}}$ outputs $(\mathsf{decrypted\_blindly}, \mathit{mid})$. Both schedule these outputs.

  On input *sig*, $\mathsf{M}'_{u,\mathcal{H}}$ continues parsing as in the previous case. Hence it outputs $(\mathsf{received}, v, m)$ iff $v' = v$ and $u' = u$. (Invariant 2 is used to derive that *sig* passes the test with $\mathsf{M}'_{u,\mathcal{H}}$'s variable $\mathit{pks}_v$.)

  On input $(\mathsf{decrypted\_blindly}, \mathit{mid})$, $\mathsf{M}^*_{u,\mathcal{H}}$ outputs and schedules $(\mathsf{received\_blindly},\ v, i)$ iff $v' = v$ and $u' = u$. (Otherwise both abort and control returns to $\mathsf{A}$.) $\mathsf{Mux}$ then outputs and schedules $(\mathsf{select}, v, i, u)$. Thus $\mathsf{TH}_{\mathcal{H}}$ retrieves $\mathit{deliver}^*_{v,u}[i]$, which is $m$, and also outputs $(\mathsf{received}, v, m)$.

The invariants and $\phi$ remain satisfied because nothing is changed.

**Receive from dishonest party.** Upon input $c$ at $\mathsf{net}^{\mathsf{a}}_{v,u}?$ for $u \in \mathcal{H}, v \in \mathcal{A}$, everything proceeds as in the case $v \in \mathcal{H}$ until $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ and $\mathsf{Enc}^*_{\mathcal{H}}$ have either both or none found the desired entry in *ciphers* and *blind_ciphers*, respectively.

- If no entries are found, decryption and parsing continues as above and both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ output $(\mathsf{received}, v, m)$, and $\mathsf{M}^*_{u,\mathcal{H}}$ schedules it. $\mathsf{Mux}$ then outputs and schedules $(\mathsf{send}, v, m, u)$. $\mathsf{TH}_{\mathcal{H}}$ checks $\mathit{init}^*$, which succeeds because of the consistency with *init*, and also outputs $(\mathsf{received}, v, m)$.

- If such entries are found, Invariant 3 implies that *sig* is a pair $((v', m, u'), \mathit{sig}')$ with $v' \in \mathcal{H}$, and by $\phi$, *mid* is a triple $(v', i, u')$ with this $v'$. Thus parsing in both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ fails because $v \in \mathcal{A}$ and thus $v' \neq v$.

The invariants and $\phi$ remain satisfied because nothing is changed.

**Runs with forged signatures.** We have shown that the ideal and semi-real configuration produce runs where all steps are matched by $\phi$, and thus the views of $\mathsf{A}$ and $\mathsf{H}$ are identical, except if the run of the semi-real system belongs to a set $\mathit{Forgeries}_{v,k}$. In each such run the adversary has produced a signature with a key $\mathit{sks}_v$ of a correct machine $\mathsf{M}^*_{v,\mathcal{H}}$ under a message that this machine had not signed. Hence the overall statement follows from the security of the signature scheme.

The proof is a standard reduction: We show that each sequence $(\mathit{Forgeries}_{v,k})_{k\in\mathbb{N}}$ has negligible probability (see Definition A.1). Then the sequence of the overall sets of exceptions, $(\mathit{Forgeries}_k)_{k\in\mathbb{N}}$ with $\mathit{Forgeries}_k := \cup_{v\in\mathcal{H}} \mathit{Forgeries}_{v,k}$ is also negligble.

Assume the contrary for some $v$. We then construct an adversary $\mathsf{A}_{\mathsf{sig}}$ against the signature scheme (recall Definition B.1 in Appendix B): On input a public key *pks*, it runs $conf_{\mathsf{id}}$ with *pks* as $pks_v$, using the signature oracle for all signatures with the now unknown $sks_v$. It verifies on-line whether the run belongs to $Forgeries_{v,k}$ (recall that this is easy). If yes, it outputs the designated signature. The success probability of $\mathsf{A}_{\mathsf{sig}}$ for a security parameter $k$ is precisely the probability of $Forgeries_{v,k}$.

# 6 Summary

We presented the first rigorous model for secure reactive systems with cryptographic parts in asynchronous networks (Section 2) and a composition theorem for it (Section 4). Common types of cryptographic systems and their trust models can be expressed in this model, in particular systems with static or adaptive adversaries (Section 3).

As design principles, we propose to keep specifications *abstract*, i.e., free of all implementation details, and to explicitly include all *tolerable imperfections*. This allows the cryptographically verified systems to be used as building blocks for systems that are then subject to formal verification. As an example of this specification methodology we provided the first abstract and complete specification for Secure Message Transmission, and verified a concrete implementation (Section 5).

Future work will include actually using a formal language to express the abstract specifications and examples where the composition theorem is applied based on such specifications. We already used it in the example, but based on Th. 5.2 with a non-abstract ideal system; we only regard the final ideal system of Th. 5.1 as abstract. Nevertheless, it might be interesting to see how far even the lower-level proofs could be supported by tools.

# Acknowledgments

We thank *Ran Canetti, Martin Hirt, Paul Karger, Matthias Schunter, Victor Shoup* and *Michael Steiner* for interesting discussions.

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

## References

[1] M. Abadi, P. Rogaway, *Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption)*, IFIP International Conference on Theoretical Computer Science (TCS 2000), LNCS 1872, Springer-Verlag, 2000, 3–22

[2] M. Bellare, A. Desai, D. Pointcheval, P. Rogaway, *Relations Among Notions of Security for Public-Key Encryption Schemes*, Crypto '98, LNCS 1462, Springer-Verlag, 1998, 26–45

[3] D. Beaver, *Secure Multiparty Protocols and Zero Knowledge Proof Systems Tolerating a Faulty Minority*, J. of Cryptology 4/2 (1991) 75–122

[4] M. Bellare, A. Boldyreva, S. Micali, *Public-Key Encryption in a Multi-user Setting: Security Proofs and Improvements*, Eurocrypt 2000, LNCS 1807, Springer-Verlag, 2000, 259–274

[5] M. Bellare, R. Canetti, H. Krawczyk, *A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols*, 13th Symp. on Theory of Computing (STOC), ACM, 1998, 419–428

[6] D. Beaver, S. Haber, *Cryptographic Protocols Provably Secure Against Dynamic Adversaries*, Eurocrypt '92, LNCS 658, Springer-Verlag, 1993, 307–323

[7] 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

[8] R. Canetti, *Security and Composition of Multiparty Cryptographic Protocols*, J. of Cryptology 13/1 (2000) 143–202

[9] R. Canetti, *A Unified Framework for Analyzing Security of Protocls (Preliminary Version)*, received from author, December 3rd, 2000.

[10] R. Canetti, S. Goldwasser, *An Efficient Threshold Public Key Cryptosystem Secure Against Adaptive Chosen Ciphertext Attack*, Eurocrypt '99, LNCS 1592, Springer-Verlag, 1999, 90–106

[11] R. Cramer, V. Shoup, *A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack*, Crypto '98, LNCS 1462, Springer-Verlag, 1998, 13–25

[12] D. Dolev, A. C. Yao, *On the Security of Public Key Protocols*, IEEE Transactions on Information Theory 29/2 (1983) 198–208

[13] R. Gennaro, S. Micali, *Verifiable Secret Sharing as Secure Computation*, Eurocrypt '95, LNCS 921, Springer-Verlag, 1995, 168–182

[14] O. Goldreich, *Secure Multi-Party Computation*, Working Draft, Version 1.1, September 21, 1998, available from `http://www.wisdom.weizmann.ac.il/users/oded/pp.htm`

[15] S. Goldwasser, L. Levin, *Fair Computation of General Functions in Presence of Immoral Majority*, Crypto '90, LNCS 537, Springer-Verlag, 1991, 77–93

[16] S. Goldwasser, S. Micali, *Probabilistic Encryption*, J. of Computer and System Sciences 28 (1984) 270–299

[17] S. Goldwasser, S. Micali, R. L. Rivest, *A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks*, SIAM J. on Computing 17/2 (1988) 281–308

[18] S. Goldwasser, S. Micali, C. Rackoff, *The Knowledge Complexity of Interactive Proof Systems*, SIAM J. on Computing 18/1 (1989) 186–207

[19] O. Goldreich, S. Micali, A. Wigderson, *How to Play any Mental Game—Or—A Completeness Theorem for Protocols with Honest Majority*, 19th Symp. on Theory of Computing (STOC), ACM, 1987, 218–229

[20] M. Hirt, U. Maurer, *Player Simulation and General Adversary Structures in Perfect Multiparty Computation*, J. of Cryptology 13/1 (2000) 31–60.

[21] P. Lincoln, J. Mitchell, M. Mitchell, A. Scedrov, *A Probabilistic Poly-Time Framework for Protocol Analysis*, 5th Conf. on Computer and Communications Security, ACM, 1998, 112–121

[22] P. Lincoln, J. Mitchell, M. Mitchell, A. Scedrov, *Probabilistic Polynomial-Time Equivalence and Security Analysis*, Formal Methods '99, LNCS 1708, Springer-Verlag, 1999, 776–793

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

[24] N. Lynch, *I/O Automaton Models and Proofs for Shared-Key Communication Systems*, 12th Computer Security Foundations Workshop (CSFW), IEEE, 1999, 14–29.

[25] S. Micali, P. Rogaway, *Secure Computation*, Crypto '91, LNCS 576, Springer-Verlag, 1992, 392–404

[26] B. Pfitzmann, M. Schunter, M. Waidner, *Cryptographic Security of Reactive Systems*, Electronic Notes in Theoretical Computer Science (ENTCS) 32 (2000), `http://www.elsevier.nl/cas/tree/store/tcs/free/noncas/pc/menu.htm`

[27] B. Pfitzmann, M. Schunter, M. Waidner, *Secure Reactive Systems*, IBM Research Report RZ 3206 (#93252), IBM Research Division, Zürich, May 2000

[28] B. Pfitzmann, M. Schunter, M. Waidner, *Provably Secure Certified Mail*, IBM Research Report RZ 3207 (#93253), IBM Research Division, Zürich, August 2000

[29] B. Pfitzmann, M. Waidner, *A General Framework for Formal Notions of "Secure" System*, Hildesheimer Informatik-Berichte 11/94, Universität Hildesheim, April 1994, available at `http://www.semper.org/sirene/lit/abstr94.html#PfWa_94`

[30] B. Pfitzmann, M. Waidner, *Composition and Integrity Preservation of Secure Reactive Systems*, 7th Conf. on Computer and Communications Security, ACM, 2000, 245–254

[31] C. Rackoff, D. R. Simon, *Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack*, Crypto '91, LNCS 576, Springer-Verlag, 1992, 433–444

[32] R. Segala, N. Lynch, *Probabilistic Simulations for Probabilistic Processes*, Concur '94, LNCS 836, Springer-Verlag, 1994, 481–497

[33] A. C. Yao, *Protocols for Secure Computations*, 23rd Symp. on Foundations of Computer Science (FOCS), IEEE, 1982, 160–164

[34] A. C. Yao, *Theory and Applications of Trapdoor Functions*, 23rd Symp. on Foundations of Computer Science (FOCS), IEEE, 1982, 80–91

# A Negligible Functions and Indistinguishability

Definition A.2 is based on [34].

**Definition A.1** *(Negligible Functions)* The notation $g(k) \leq 1/\mathsf{poly}(k)$ for a function $g : \mathbb{N} \to \mathbb{N}$, also written $g \in \mathit{NEGL}$, means that for all positive polynomials $Q$, $\exists k_0 \forall k \geq k_0$: $g(k) \leq 1/Q(k)$. $\diamond$

**Definition A.2** *(Indistinguishability)* Two families $(\mathsf{var}_k)_{k\in\mathbb{N}}$ and $(\mathsf{var}'_k)_{k\in\mathbb{N}}$ of random variables (or probability distributions) are called

a) perfectly indistinguishable ("=") if for each $k$, the two distributions are identical;

b) statistically indistinguishable ("$\approx_{\mathit{SMALL}}$") for a class $\mathit{SMALL}$ of functions from $\mathbb{N}$ to $\mathbb{R}_{\geq 0}$ if the distributions are discrete and their statistical distances

$$\Delta(\mathsf{var}_k, \mathsf{var}'_k)$$
$$= \frac{1}{2} \sum_{d \in D_k} |P(\mathsf{var}_k = d) - P(\mathsf{var}'_k = d)| \in \mathit{SMALL}$$

(as a function of $k$). $\mathit{SMALL}$ should be closed under addition, and with a function $g$ also contain every function $g' \leq g$. Typical classes are $\mathit{EXPSMALL}$ containing all functions bounded by $Q(k) \cdot 2^{-k}$ for a polynomial $Q$, and the (larger) class $\mathit{NEGL}$.

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

$$|P(\mathsf{Dis}(1^k, \mathsf{var}_k) = 1) - P(\mathsf{Dis}(1^k, \mathsf{var}'_k) = 1)| \leq \frac{1}{\mathsf{poly}(k)}.$$

(Intuitively, $\mathsf{Dis}$, given the security parameter and an element chosen according to either $\mathsf{var}_k$ or $\mathsf{var}'_k$, tries to guess which distribution the element came from.)

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

The following lemma about indistinguishability can easily be proven.

**Lemma A.1** *(Indistinguishability)*

- Indistinguishability of two families of random variables implies indistinguishability of every function $\phi$ of them (in particular restrictions). For the computational case, $\phi$ must be polynomial-time computable.
- The statistical distance $\Delta(\phi(\mathsf{var}_k), \phi(\mathsf{var}'_k))$ between a function of two random variables is at most $\Delta(\mathsf{var}_k, \mathsf{var}'_k)$.

$\square$

# B Secure Signature and Encryption Schemes

**Definition B.1** *(Security of Signature Schemes)* An arbitrary (probabilistic) polynomial-time machine $\mathsf{A}_{\mathsf{sig}}$ interacts with a signer machine $\mathsf{Sig}$ (also called signing oracle) defined as follows:

1. $\mathsf{Sig}$ generates a key pair, $(sks, pks) \leftarrow \mathsf{gen}_{\mathsf{S}}(1^k, 1^s)$, and sends $pks$ to $\mathsf{A}_{\mathsf{sig}}$.
2. In each step, $\mathsf{Sig}$ signs an arbitrary message $m_j$ it receives from $\mathsf{A}_{\mathsf{sig}}$ (automatically only a polynomial number if $\mathsf{A}_{\mathsf{sig}}$ is polynomial).
3. Finally, $\mathsf{A}_{\mathsf{sig}}$ should output a value $sig$.

$\mathsf{A}_{\mathsf{sig}}$ has won if $\mathsf{test}_{pks}(sig)$ gives a message $m$ with $m \neq m_j$ for all $j$, i.e., $sig$ is a valid signature on a message that $\mathsf{Sig}$ did not sign. The probability of this event must be negligible in $k$. (In our terminology, $[\{\mathsf{Sig}, \mathsf{A}_{\mathsf{sig}}\}]$ is a closed collection, and the event is a predicate on the runs; hence the probability is well-defined.) $\diamond$

Security of an encryption scheme means that any two messages must be indistinguishable even in adaptive chosen-ciphertext attacks. This was introduced in [31], used for an efficient construction in [11] and formalized as "IND-CCA2" in [2].

**Definition B.2** *(Security of Encryption Schemes)* An arbitrary (probabilistic) polynomial-time machine $\mathsf{A}_{\mathsf{enc}}$ interacts with a decryptor machine $\mathsf{Dec}$ (also called decryption oracle) defined as follows:

1. $\mathsf{Dec}$ generates a key pair, $(ske, pke) \leftarrow \mathsf{gen}_{\mathsf{E}}(1^k)$, and sends $pke$ to $\mathsf{A}_{\mathsf{enc}}$.
2. In each step, $\mathsf{Dec}$ decrypts an arbitrary ciphertext $c_j$ received from $\mathsf{A}_{\mathsf{enc}}$.
3. At some point, $\mathsf{A}_{\mathsf{enc}}$ sends a pair $(m_0, m_1)$ of messages to $\mathsf{Dec}$. Then $\mathsf{Dec}$ randomly chooses a bit $b$ and returns an encryption $c \leftarrow \mathsf{E}_{pke}(m_b)$.
4. $\mathsf{A}_{\mathsf{enc}}$ may again ask $\mathsf{Dec}$ to decrypt ciphertexts $c_j$, but now $\mathsf{Dec}$ refuses to decrypt if $c_j$ equals its own ciphertext $c$.
5. Finally, $\mathsf{A}_{\mathsf{enc}}$ should output a bit $b^*$.

This bit is meant as a guess at $b$, i.e., which of the two messages is contained in $c$. The probability of the event $b^* = b$ must be bounded by $1/2 + 1/\mathsf{poly}(k)$. (In our terminology, $[\{\mathsf{Dec}, \mathsf{A}_{\mathsf{enc}}\}]$ is a closed collection and the event is a predicate on the runs; hence the probability is well-defined.) $\diamond$