RZ 3206 (#93252) 02/14/00
Computer Science/Mathematics 44 pages

Research Report

Secure Reactive Systems

Birgit Pfitzmann[1], Matthias Schunter[1], Michael Waidner[2]

[1] Universität des Saarlandes
Im Stadtwald 45
D-66123 Saarbrücken
Germany
pfitzmann@cs.uni-sb.de, schunter@acm.org

[2] IBM Zurich Research Laboratory
Säumerstrasse 4
CH-8803 Rüschlikon
Switzerland
wmi@zurich.ibm.com

Research Division
Almaden · Austin · Beijing · Delhi · Haifa · T.J. Watson · Tokyo · Zurich

Secure Reactive Systems

Birgit Pfitzmann, Matthias Schunter, Michael Waidner

May 2, 2000

Abstract

We introduce a precise definition of the security of reactive systems following the simulatability approach in the synchronous model. No simulatability definition for reactive systems has been worked out in similar detail and generality before. Particular new aspects are a precise switching model that allows us to discover timing vulnerabilities, a precise treatment of the interaction of users and adversaries, and independence of the trust model.

We present several theorems relating the definition to other possible variants. They substantiate which aspects of such a definition do and do not make a real difference, and are useful in larger proofs. We also have a methodology for defining the security of practical systems by simulation of an ideal system, although they typically have imperfections tolerated for efficiency reasons.

We sketch several examples to show the range of applicability, and present a very detailed proof of one example, secure reactive message transmission. Its main purpose is to validate the model by an example of a class that has also been considered in other models, but we did encounter new problems related to our strict requirements on timing security.[1]

[1]A larger and more novel example is shown in detail in [PfSW2 00, Schu 00]. We also believe to have a proof of a general composition theorem. A short overview of the current paper (without the theorems and the proof of the example), together with discussions on the relation to formal methods, is given in [PfSW 00].

1 Introduction

Most practically relevant cryptographic systems are reactive, i.e., user and system interact multiple times, and both keep state between interactions. In some systems, already the individual operations only make sense in a reactive scenario, e.g., in an electronic cash system where users can perform payments based on withdrawals, and deposits based on payments. But actually *all* systems secure against active attacks (in the sense of chosen-message or chosen-ciphertext) must be defined reactively because such an attack presupposes that the system gets several inputs or makes several outputs.

We introduce definitions which allow us to rigorously specify and prove the security of *general* reactive systems. We follow the simulatability paradigm: A system is specified by means of an ideal system that has all the desired security properties by construction, but typically makes the unrealistic assumption that a machine trusted by all parties is available ("trusted host"). A real system is defined to be as secure as this ideal system if anything an adversary can achieve there can also be achieved by an adversary attacking the ideal system.

There are two main approaches at defining the security of reactive systems under the simulatability paradigm, both based on work on secure function evaluation [Yao 82, GoLe 91, MiRo 92, Beav5 91, Cane 96].

The first, constructive, approach describes the ideal system as a global state-transition machine, and requires the state to be shared among all participants in the real system. Computing the state-transition function by secure multi-party function evaluation yields a general construction for any such reactive system [GMW 87, Gold 98]. However, sharing the entire global state among all participants is not feasible or desirable in scenarios like secure channels or payment systems, where many participants carry out many 2- or 3-party protocol runs at different times.

The second, descriptive, approach only considers the "outside" behaviour of the system. More precisely, we will consider both honest users and an adversary as stateful machines apart from the system, and the definition will be that whatever an adversary can achieve in the real system with respect to any given honest users, another adversary can achieve with respect to the same honest users in the ideal system. This type of definitions was first sketched in [PfWa 94] (presented in [Pfit6 96, PfWa3 98]). A similar sketch, based on a specific definition of multiparty function evaluation [Cane 96], is given in [Cane 00]. The first precise definition—still for a somewhat restricted class of systems in order to achieve general constructions, and with another (but also synchronous) timing model than ours—was given in [HiMa 00]. Such a definition was also worked out in [LMMS 98] for a formal language (π-calculus) as machine model, but without any distinction of user and adversary, which does not allow protocol-independent definitions as we aim at. Protocol independence means in particular that systems like secure message transmission or payment systems, which can be implemented with a variety of cryptographic primitives, have definitions independent of these primitives.

The simulatability paradigm has also been applied to specific problems, such as verifiable secret sharing [GeMi 95], secure key agreement [BeCK1 98, Shou 99] and threshold decryption [CaGo 99]. The first one follows the constructive approach, the others the descriptive approach. Not all issues of the general case arise in these examples. Moreover, we hope that a general definition significantly eases the specification part of specific problems in the future.

We make the definition for a *synchronous* network model. An asynchronous model might appear more general, but it hides security exposures via timing channels.[2] Our notion of security implies that there are no distinguishable timing differences between real and ideal system. Timing channels are well known from operating-systems security, where information that should

[2]Of course, a precise general definition for asynchronous systems is also needed. But it does not exist yet either, and we only argue that neither model comprises the other. We also do not consider dynamic corruptions yet.

be hidden by the access control mechanisms may be leaked by the timing of otherwise harmless events. To our knowledge, this was first mentioned in [Lipn 75], and it is still a research topic for information-flow security also in networks and databases, e.g., [Brow 95, VeWo 95, Tros 98]. For low-level cryptography, timing channels were first treated in [Koch 96]. We are not aware of a treatment of timing behaviour in higher-level cryptographic protocols. We needed a new switching model for it, and we found that typical real-life protocols, even after the cryptography has been adapted to enable simulation, do not correspond to "naive" ideal hosts where every participant in a subprotocol makes an input at the same time and obtains an output at the same time. More motivation, based on an example, follows in Section 6.

Most practical systems also have further *tolerable imperfections*, e.g., they allow traffic analysis, that cannot be avoided or only at a price (e.g., communication overhead) that one is not willing to pay in most applications. Nevertheless, it should be possible to define the achieved security precisely. The consequence for our general definitions is that we allow the ideal system to provide a richer service to the adversary than to the honest users. This is one reason why we need a distinction of two kinds of interfaces already in the system definitions.

In Section 2, we present our primary definitions. In Section 3, we show how to specialize the definitions to the typical trust models of cryptology, and we sketch a few other examples to motivate why we made the definition more general. Section 4 contains some basic lemmas. In Section 5, we define some model variants and prove that most of them are equivalent to our primary definitions. Hence the model appears to capture the notion of secure reactive systems well. Moreover, the equivalent variants are useful in proofs.[3] Finally, in Section 6, we present one cryptographic example in detail, secure reactive message transmission. Its main purpose is to validate the definition by an example that seems simple.[4] Nevertheless, we were not sure in advance what message format would be right, and certain timing problems and other imperfections like traffic analysis had to be dealt with. The proof is rigorous in both the cryptographic and the non-cryptographic aspects.

2 Definitions

We first present the general machine model we use in Section 2.1, and we introduce timing models. Section 2.2 contains definitions of systems with adversaries and honest users. In Section 2.3 we present our primary definition of simulatability for those systems.

2.1 General System Model

Intuitively, when a reactive system is running, there are some connected correct machines, adversaries, and honest users, i.e., an environment to which we want to guarantee a service, see Figure 1. We will call this a configuration. We now define the different parts of such configurations, how different configurations are derived from a "system", and the executions of and views in a configuration.

As machine model, we use normal probabilistic extended finite-state machines (e.g., essentially equivalent to the I/O automata in [Lync 96]). For clarity, we fix one particular notation. We only use point-to-point connections in the basic model. (This is discussed in Section 3.2.3. Hence the connection graph and the "endpoints" of the connections essentially determine each other uniquely. We call these endpoints ports. For instance, a machine that makes outputs to three connections with other machines has three output ports. There are connection-based and

[3] We also have composition theorems enabling modular construction of reactive systems based on ideal versions of their subsystems; this should be presented separately soon.

[4] As a larger example, and of a class that had not previously been defined rigorously or considered under a simulatability approach, we treat certified mail in a separate paper [PfSW2 00, Schu 00].

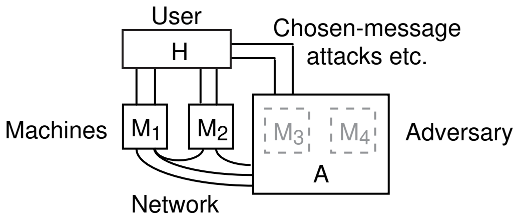

Figure 1: Configuration of a reactive system. The gray part shows that the adversary has typically replaced some machines.

port-based notations. For our purpose, a port-based notation is easiest because we consider the same machines in different settings (e.g., a real and an ideal system) and have to express that they are connected in a similar way (e.g., a user who used a correct machine in the real system must be connected with a trusted host at the same port in the ideal system).

Definition 2.1 (Ports)

a) A name *is a string over a fixed alphabet* Σ.

b) A port *p is a pair $(name_p, dir_p)$ of a name and a Boolean value, called* port name *and* direction. *Concretely, we write output ports $name_p!$ and input ports $name_p?$, inspired by the CSP notation [Hoar2 85].*

c) We write p^c for the complement *of a port p, i.e.*

$$name_p!^c = name_p?$$

and vice versa; also for sets of ports.

d) For a set P of ports, let $\mathsf{In}(P) := \{p \in P | dir_p = ?\}$ denote the input ports and $\mathsf{Out}(P) := \{p \in P | dir_p = !\}$ the output ports.

◇

Definition 2.2 (Machines)

a) A machine M *for a synchronous system is a tuple*

$$\mathsf{M} = (Ports_{\mathsf{M}}, \delta_{\mathsf{M}}, Ini_{\mathsf{M}}, F_{\mathsf{M}})$$

of a finite set of ports, a probabilistic state-transition function, and sets of initial and final states. The states are strings from Σ^*. The inputs are tuples $I = (I_p)_{p \in \mathsf{In}(Ports_{\mathsf{M}})}$ of one input $I_p \in \Sigma^*$ per input port, and the outputs tuples $O = (O_p)_{p \in \mathsf{Out}(Ports_{\mathsf{M}})}$ with $O_p \in \Sigma^*$.

By "probabilistic state-transition function" we mean that $\delta_{\mathsf{M}}(s, I)$ is a finite probability distribution over pairs (s', O) for each pair (s, I).

b) For a set M of machines, let $\mathsf{ports}(M) := \bigcup_{\mathsf{M} \in M} Ports_{\mathsf{M}}$.[5]

c) If we say that a machine M_1 has a machine M_2 as a (blackbox) submachine, *we mean that it gets the state-transition function as a blackbox (or "oracle"), not an object with its own clock. Hence M_1 can "clock" M_2, i.e., decide when to cause state transitions.*[6]

[5]We mostly use a straight font for machines, functions and constants, and italics for sets and other variables.

[6]This model also enables M_1 to reset M_2 to a prior state, but we will not use this. Our type of probabilism does not allow M_1 to set the random inputs of M_2; in other cryptographic proofs where one needs this, δ_{M} must be refined into a deterministic function with an additional random input.

$\diamond$

For computational aspects, we assume that each machine is implemented by a probabilistic interactive Turing machine [GoMR 89] and each port by a communication tape.[7] The complexity of a machine is, as usual, measured in terms of the length of the initial state (often a security parameter; regarded as a distinguished input).[8] We use uniform complexity because the security proofs will be reduction proofs between either two uniform or two non-uniform statements, and a reduction in uniform complexity is also one in non-uniform complexity.

Below, we distinguish correct machines, adversaries and users in particular in how they are clocked w.r.t. each other. However, in proofs we sometimes need different combinations; hence we define collections of machines and their runs with a clocking scheme in general.

Definition 2.3 (Machine Collections, Runs and Views)

a) *A* collection *is a finite set of machines with pairwise disjoint sets of ports.*

b) *Given a collection C, we assume that all complementary ports are connected, i.e., we call each set $c = \{p, p^c\} \subseteq \mathsf{ports}(C)$ a* connection *and the set of these connections the* connection graph $\mathsf{G}(C)$.[9]

c) *By $\mathsf{free}(C)$ we denote the* free *ports, i.e., $p \in \mathsf{ports}(C)$ but $p^c \notin \mathsf{ports}(C)$. The others are called* inner *ports, $\mathsf{inner}(C)$. A collection is* closed *if $\mathsf{free}(C) = \emptyset$.*

d) *A* clocking scheme *is a mapping κ from a set $\{1, \ldots, n\}$ to the powerset of C, i.e., it assigns each number a subset of the machines.*

e) *For a closed collection C, a clocking scheme κ, and a tuple $ini \in Ini := \times_{\mathsf{M} \in C} Ini_{\mathsf{M}}$ of initial states, we define* runs *(or "executions" or "traces"): Each global round i has n subrounds. In Subround $[i.j]$, all machines $\mathsf{M} \in \kappa(j)$ switch simultaneously, i.e., each state-transition function δ_{M} is applied to M's current inputs and state and yields a new state and output according to the distribution. Then the outputs are transported to the corresponding input ports of other machines, where they are available until that machine switches next. If several inputs arrive until that time, they are concatenated. This gives a family of random variables*

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

More precisely $run_{C,ini}$ is a function mapping each triple $(\mathsf{M}, i, j) \in C \times \mathbb{N} \times \{1, \ldots, n\}$ to the quadruple of the old state, inputs, new state, and outputs of machine M in subround $[i.j]$, with a symbol ϵ for machines that do not switch in this subround.

f) *For a subset M of a closed collection C, we define the view of M as the family of random variables*

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

where $(view_{C,ini}(M))$ is the restriction of $run_{C,ini}$ to $M \times \mathbb{N} \times \{1, \ldots, n\}$.[10]

[7]Detailed models for the multi-party case exist in [Beav5 91] and the report version of [MiRo 92]. They are still slightly abstract in assuming that all machines compute as long as they want per round, and then magically the transport function and new round are started.

[8]If only the state-transition function were polynomial-time, two interacting polynomial-time machines could easily gain exponential power, e.g., if each output is twice as long as the previous input.

[9]A connection is intuitively directed, although it is defined as a set, because it consists of one output and one input port.

[10]For the view of a polynomial-time machine in interaction with unrestricted machines, one should only consider the inputs as far as the machine read them. Such a notion of "read" can be defined in the Turing machine realization.

g) For a number $l \in \mathbb{N}$ of rounds, we define l-round prefixes $run_{C,ini,l}$ and $view_{C,ini,l}(M)$ of runs and views in the obvious way. For a function $l : Ini \to \mathbb{N}$, we define a family $run_{C,l} = (run_{C,ini,l(ini)})_{ini \in Ini}$ and similarly $view_{C,l}(M)$.

$\diamond$

Remark 2.1. To avoid that timing differences within a round leak, implementations of synchronous machines have to ensure that input reading and outputting are both clocked. $\circ$

2.2 Specific System Model

We now define more specific collections of machines for security purposes. We begin with the system part and then add an environment, i.e., users and adversaries.

Definition 2.4 (Structures and Systems)

a) A structure *is a pair $struc = (M, S)$ where M is a collection of machines called* correct machines, *and $S \subseteq \mathsf{free}(M)$ is called* specified ports.

We define $\bar{S} := \mathsf{free}(M) \setminus S$ and $\mathsf{forb}(M, S) := \mathsf{ports}(M) \cup \bar{S}^c$.

b) A system *Sys is a set of structures.*

$\diamond$

Essentially, 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, "set up session to *id*" for a key exchange system, "pay amount x to *id*" for a payment system, or "play out card x" in a game. The ports in $\bar{S}$ are additionally available for the adversary. The ports in $\mathsf{forb}(M, S)$ will therefore be forbidden or at least unusual for an honest user to have, those from $\mathsf{ports}(M)$ to avoid name clashes and those from $\bar{S}^c$ because they would give connections to $\bar{S}$.

Such a specification style over services offered at free ports may seem unusual in cryptography, but it is quite normal in other fields, because it is a useful way to offer abstraction and concentrate on those aspects of a system that are relevant for the environment. Moreover, similarity to other specification styles is an advantage in itself, because in the end, cryptographic systems must be employed in larger systems and their specification be usable as a component in a "normal" overall design.

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 one can imagine a wide range of trust models, e.g., with semi-correct machines, we kept the remaining definitions independent of this by the general system model.

The following definition was already illustrated in Figure 1; the clocking scheme is explained after the definition.

Definition 2.5 (Configurations and their Runs and Views)

a) A configuration *conf of a system Sys is a tuple $(M, S, \mathsf{H}, \mathsf{A})$ where $(M, S) \in Sys$ is a structure and H and A are machines modeling the honest users and the adversary. The union $M \cup \{\mathsf{H}, \mathsf{A}\}$ must be a closed collection. We call H and A a* compatible *user and adversary for the structure.*[11]

[11]Hence the conditions are that $Ports_{\mathsf{A}}$ and $Ports_{\mathsf{H}}$ are disjoint with each other and with $\mathsf{ports}(M)$, and that no ports remain free. We could relax the condition by considering only the free ports of M, but that would require defining port visibility, and we can always achieve this condition by prefixes to the names of inner ports.

b) *We denote the set of all configurations of a system by* $\mathsf{Conf}(Sys)$*, and those with polynomial-time adversary and user by* $\mathsf{Conf}_{\mathsf{poly}}(Sys)$*, omitting "*$\mathsf{poly}$*" if it is clear from the context.*

c) *The clocking scheme is* $(M \cup \{\mathsf{H}\}, \{\mathsf{A}\}, \{\mathsf{H}\}, \{\mathsf{A}\})$*, i.e., in Subround* $[i.1]$*, the machines from* M *and* H *switch, in Subround* $[i.2]$ *the adversary* A*, etc.*[12]

d) *Runs and views of a configuration are now given by Definition 2.5. However, for simplicity we now assume that the initial states of all machines are only a security parameter* k *(in unary representation).*[13] *We therefore 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 we identify* Ini' *with* $\mathbb{N}$*.*

   We write run_{conf} *and* $view_{conf}(M)$ *for* run_C *and* $view_C(M)$ *restricted to* Ini' *(although the collection* C *is* $M \cup \{\mathsf{H}, \mathsf{A}\}$*).*

◇

We find such a model with explicit honest users much more intuitive than models without: Honest users should not be modeled as part of the machines in M because they are arbitrary, while the machines have prescribed programs. For example, they may have arbitrary strategies which message to input next to the system after certain outputs. They may also be influenced in these choices by the adversary, e.g., in chosen-message attacks on a signature scheme; therefore they typically have ports connected to the adversary. These strategies and influences are also why we cannot simply quantify over input sequences instead of machines H, at least for the computational case.) Honest users are not a natural part of the adversary either because they are supposed to be protected from the adversary. In particular, they may have secrets, and we want to define that the adversary learns nothing about those except what he learns "legitimately" from the system (this depends on the specification) or what the user tells him directly.[14]

We currently model a single adversary who coordinates all malicious behavior in the configuration. Multiple adversaries only make a difference if we limit the communication between them.[15]

The fact that the adversary is clocked between the correct machines is the well-known model of "rushing adversaries" (the earliest references we are aware of are [BrDo 84] for the concept and [ChDw 89] for the name).[16] Our separate clocking of users is new. For human users, the non-synchrony with the system rounds is clear: Even an honest human user might react on a message as soon as it appears on a display. Application programs may also have their own timing quite different from the rounds of the underlying cryptographic protocol, e.g., in a real-time system. Hence in particular, the users might interact with an adversary several times within one system round. However, we show in Section 5.5 that this "dialogue model" is equivalent to the simpler model above, at least in all cases where security in the dialogue model can be achieved at all.

[12] Letting H switch not only in Subround $[i.3]$, but also in $[i.1]$ is not really necessary: We see in Section 5.5 that arbitrary dialogues can be simulated without. However, it simplifies composition, where machines of one system are users of another system.

[13] See Section 5.3 for definitions with auxiliary inputs and that they do not make much difference.

[14] This is different for pure integrity specifications [Pfit4 93].

[15] This looks fairly straightforward: One also fixes the ports of the different adversaries. But we have not worked out any theorems or examples for this case (e.g., wallets with observers [ChPe1 93]). Mixed adversaries as in [FiHM 99] are still centrally controlled and fit under our standard definition.

[16] The reason is that synchrony is usually not absolute, but implemented by certain bounds in which messages may arrive. Hence an adversary controlling the network can usually base his outputs in one round on the correct machines' outputs of the same round. A standard example where this would harmful is if the machines implemented coin flipping by simply exoring random bits that each of them outputs in the same round.

We make no special definition of ideal systems. Typically, an ideal system *Sys* contains only structures of the type $(\{\mathsf{TH}\}, S)$, i.e., with only one correct machine called *trusted host*. The fact that we allow several such structures in an ideal system and that TH may have specific connections to the adversary is different from other models and allows us to define systems with tolerable imperfections.

2.3 Simulatability

We now define what it means that one system Sys_1 is "at least as secure as" another system Sys_2. Typically, Sys_1 is real and Sys_2 ideal. The essential aspect is that we compare the view of the same honest users once in Sys_1, once in Sys_2. Security means that whatever can happen to the honest users in Sys_1 with some adversary can also happen to them, with another adversary, in Sys_2. (Different choices of what to compare are discussed in Section 5.) A simple case is illustrated in Figure 2.

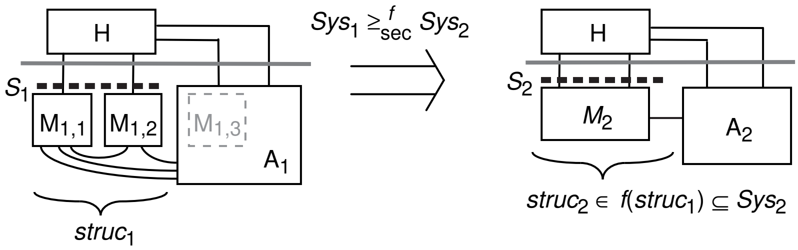

Figure 2: Simulatability definition. The gray line delimits the view of the user H, which must be indistinguishable. The standard case $S_1 = S_2$ is drawn.

However, we only want to compare each structure of Sys_1 with certain "corresponding" ones in Sys_2. Typically these are the ones with the same set of specified ports (see Section 3 for both typical and untypical examples). For the general case, we allow an almost arbitrary mapping f that maps structures $struc_1$ of Sys_1 to sets of "corresponding" structures of Sys_2. The following conditions are mainly naming conventions, and become clear in the remarks at the end of this chapter and the proof of transitivity.

Definition 2.6 (Valid Mapping, Suitable Configuration) *Let two systems Sys_1 and Sys_2 be given. A function f from Sys_1 to the powerset of Sys_2 is called a* valid mapping *(between Sys_1 and Sys_2, but we usually omit this phrase) iff for all corresponding structures $(M_2, S_2) \in f(M_1, S_1)$*

$$p^c \in \mathsf{free}(M_1) \Rightarrow p \notin \mathsf{forb}(M_2, S_2) \quad \wedge \quad p^c \in S_2 \Rightarrow p \notin \mathsf{forb}(M_1, S_1).$$

That is, ports for connecting to (M_1, S_1) are not "forbidden" in (M_2, S_2), nor ports for connecting "legally" to (M_2, S_2) forbidden in (M_1, S_1).[17]

Given Sys_2 and f, we define a set $\mathsf{Conf}^f(Sys_1) \subseteq \mathsf{Conf}(Sys_1)$ of suitable *configurations as follows: $(M_1, S_1, \mathsf{H}, \mathsf{A}_1)$ is suitable if H has no ports from $\mathsf{forb}(M_2, S_2)$ for any $(M_2, S_2) \in f(M_1, S_1)$.* ◇

Remark 2.2. Given a suitable configuration and $p \in Ports_{\mathsf{H}}$, then $p \notin \mathsf{ports}(M_2)$ and $p^c \in \mathsf{ports}(M_2) \Rightarrow p^c \in S_2$. The latter holds because $p^c \in \mathsf{inner}(M_2)$ by the first condition. We will often use suitability in this form. ∘

[17] We could restrict the first precondition to $p^c \in S_1$ if we generally excluded users (here of Sys_1) that connect to unspecified ports, see Section 5.6.

We need the definition of indistinguishability of families of random variables over $\mathbb{N}$, essentially from [Yao1 82].

Definition 2.7 (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 for a class SMALL of "small" functions ("$\approx_{SMALL}$") if the distributions are discrete and their L_1-distance is small. More precisely, SMALL must be a class of non-negative functions from $\mathbb{N}$ to $\mathbb{R}$, and with any function also contain any smaller function. The statistical distance Δ of two discrete random variables with a domain D_k is defined as

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

Then we require that

$$\Delta(\mathsf{var}_k, \mathsf{var}'_k) \in SMALL$$

(as a function of k). Typical classes SMALL are the class EXPSMALL of all functions bounded by $Q(k)\cdot 2^{-k}$ for a polynomial Q,[18] *and the (larger) class NEGL of all negligible functions as in Part c).*

c) computationally indistinguishable ("$\approx_{\mathsf{poly}}$") if for any algorithm Dist *(the distinguisher) that is probabilistic polynomial-time in its first input, the differences of its results on the two distributions are negligible:*

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

Hence Dist *gets the security parameter and an element chosen according to either* var_k *or* var'_k *as inputs and makes a Boolean output. The notation $g(k) \leq 1/\mathsf{poly}(k)$, equivalently $g \in NEGL$, for a function g means that for all positive polynomials Q, $\exists k_0 \forall k \geq k_0 : g(k) \leq 1/Q(k)$.*

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

Definition 2.8 (Simulatability) *Let two systems Sys_1 and Sys_2 and a valid mapping f be given.*

a) We call Sys_1 perfectly at least as secure as *Sys_2 for f and write*

$$Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{perf}} Sys_2$$

if for any suitable configuration $conf_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^f(Sys_1)$, there exists a configuration $conf_2 = (M_2, S_2, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(Sys_2)$ with $(M_2, S_2) \in f(M_1, S_1)$ (and the same H) such that

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

[18] Often a concrete bound 2^{-k} can be shown, but, e.g., the transitivity lemma below requires that *SMALL* is closed under addition.

b) We call Sys_1 statistically at least as secure as Sys_2 *for a class SMALL and write*

$$Sys_1 \geq_{\mathsf{sec}}^{f,SMALL} Sys_2$$

iff 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-round prefixes of the views for polynomials l.*

c) We call Sys_1 computationally at least as secure as Sys_2 *and write*

$$Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{poly}} Sys_2$$

iff 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 instead of equality.*

In all cases, we call $conf_2$ *an indistinguishable configuration for* $conf_1$ *and write*

$$conf_2 \in Indist^f(conf_1).$$

Where the difference between the types of security is irrelevant, we simply write $\geq_{\mathsf{sec}}^f$, *and we omit the indices* f *and* sec *if they are clear from the context.* $\diamond$

The restriction to suitable configurations $\mathsf{Conf}^f(Sys_1)$ in this definition has two purposes; recall that it means that users H that have ports from $\mathsf{forb}(M_2, S_2) = \mathsf{ports}(M_2) \cup \bar{S}_2^c$ for any $(M_2, S_2) \in f(M_1, S_1)$ are not considered. First it excludes users that are incompatible with (M_2, S_2), i.e., those with port names that already occur in M_2. This is a mere naming convention. Without it, at least in the typical case where each image $f(M_1, S_1)$ contains precisely one structure (M_2, S_2), we could not fulfil the definition.[19]

Secondly, it excludes that H communicates with unspecified free ports of (M_2, S_2). This is where the notion of specified ports is really used.[20] The typical example is an ideal system that specifies a system with tolerable imperfections, and thus has channels to the adversary to make certain events visible to him in an abstract way. (Examples are shown in Section 6.) If we would allow H access there, these exact events would have to be indistinguishable from some events in the real system. This would seriously limit the possible abstraction in the specification.

Remark 2.3. With regard to (M_1, S_1), the restriction to suitable configurations is w.l.o.g.: For every $conf_{\mathsf{e},1} = (M_1, S_1, \mathsf{H_e}, \mathsf{A}_{\mathsf{e},1}) \in \mathsf{Conf}(Sys_1) \setminus \mathsf{Conf}^f(Sys_1)$, there is a configuration $conf_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^f(Sys_1)$ such that $view_{conf_1}(\mathsf{H})$ equals $view_{conf_{\mathsf{e},1}}(\mathsf{H_e})$ except for port renaming.

We construct H by giving each port $p \in Ports_{\mathsf{H_e}} \cap \mathsf{forb}(M_2, S_2)$ a new name.[21] As $conf_{\mathsf{e},1}$ is closed, the ports p^c also occurs in it. The first condition on a valid mapping f implies $p^c \notin \mathsf{free}(M_1)$. Hence p^c belongs to $\mathsf{A}_{\mathsf{e},1}$ or $\mathsf{H_e}$ and we can indeed rename it. Then the runs and in particular the user's view are unchanged except for this renaming.

Remark 2.4. At least intuitively, the conditions on valid mappings (which are mainly naming conventions) also do not seem to exclude anything interesting: The only ports that really need to have the same names in two systems are corresponding specified ports. If the names of all

[19]If $|f(M_1, S_1)| > 1$, we have an OR-semantics, i.e., (M_1, S_1) only needs to be as secure as one corresponding structure. Thus it would be inappropriate to require such users to be compared with another $(M_2', S_2') \in f(M_1, S_1)$.

[20]Primarily, however, the distinction between the user and adversary interface of a system is in the definition of f and the port names of the two systems: If structures (M_1, S_1) and $(M_2, S_2) \in f(M_1, S_1)$ have common free ports, and a user H uses them in the first system, the same ports of H are automatically connected to the corresponding ports of (M_2, S_2). Hence the service at these ports must be indistinguishable.

[21]By *new name*, we always mean one that does not occur in the systems and configurations already under consideration. For this, we assume w.l.o.g. that no finite set of systems contains all port names over the given alphabet.

other ports in Sys_i are disjoint from the names of any specified ports in (other structures of) Sys_i and all ports in Sys_j for $j \neq i$ for a set of systems, then the conditions are always fulfilled except that $p \in S_i, p^c \in S_j$ is possible. This cannot be excluded generally, e.g., it is normal in compositions, but it would be an extremely strange case for structures of which one is supposed to replace the other. In particular, it is excluded in the standard case $S_i = S_j$, because $p, p^c \in S_i$ would imply that they are connected, in contradiction to $S_i \subseteq \mathsf{free}(M_i)$. ∘

3 Applications

Before we discuss variants of the model, we now present several specializations and small examples to give a more intuitive understanding. Recall that fully worked-out cryptographic examples can be found in Section 6 and in [PfSW2 00].

3.1 Standard Cryptographic Systems

In this section, we refine the general definitions for a standard class of cryptographic systems. The intuition is that in a real system *Sys*, there is one machine per human user, and each machine is correct if and only if its user is honest. (Typically one calls the machine and its user together a participant, and often Alice or Bob.) The system is derived from an intended structure (M^*, S^*) and a trust model.

Definition 3.1 (Standard Cryptographic Structures and Trust Model)

a) A standard cryptographic structure *is a structure* (M^*, S^*), *where we denote the machines by* $M^* = \{\mathsf{M}_1, \ldots, \mathsf{M}_n\}$. *Each machine* M_u *has two ports* $\mathsf{in}_u?$ *and* $\mathsf{out}_u!$ *intended for its user, i.e., these ports are free and their union is* S^*. *All other ports have complements at other machines, i.e., they correspond to a connection graph* $\mathsf{G}(M^*)$ *among the machines from* M^*.

b) A standard trust model *for such a structure consists of an access structure and a channel model.*

An access structure, *as usual, is a set* $\mathcal{ACC}$ *of subsets* $\mathcal{H}$ *of* $\{1, \ldots, n\}$ *closed under insertion (of more elements into a set) and denotes the possible sets of correct machines.*

A channel model *is a mapping* $\chi : \mathsf{G}(M^*) \to \{\mathsf{s}, \mathsf{r}, \mathsf{i}\}$. *It characterizes each connection as secure (private and authentic), readable (only authentic), or insecure (neither private nor authentic).*

◇

Typical examples of access structures are threshold structures, i.e., all subsets with $|\mathcal{H}| \geq t$ for some t. In the example in Section 6, everybody should be secure on their own, and thus $\mathcal{ACC}$ is the entire powerset of $\{1, \ldots, n\}$.

Definition 3.2 (Standard Cryptographic Systems) *Given a structure* (M^*, S^*) *and a trust model* $(\mathcal{ACC}, \chi)$ *as above, the corresponding system is* $Sys = \{(M(\mathcal{H}), S(\mathcal{H})) | \mathcal{H} \in \mathcal{ACC}\}$ *with the following definitions:*

a) The correct machines are

$$M(\mathcal{H}) = \{\mathsf{M}_{u,\mathcal{H}} | u \in \mathcal{H}\},$$

where each machine $\mathsf{M}_{u,\mathcal{H}}$ *is derived from* M_u *as follows. (Figure 1 was an example with a readable channel between* M_1 *and* M_2*; a more detailed example follows in Figure 11.)*[22]

[22] We rename more ports than necessary here for convenience in Section 6.

- *The ports* $\mathsf{in}_u?$ *and* $\mathsf{out}_u!$ *are unchanged.*
- *Now let any other port* $p \in Ports_{\mathsf{M}_u}$ *be given. Let* v *be the index with* $p^c \in Ports_{\mathsf{M}_v}$ *and* $c = \{p, p^c\}$ *the attached connection.*
  - *If* $v \in \mathcal{H}$ *(i.e., c is a connection between two correct machines):*
    - *If* $\chi(c) = \mathsf{s}$ *(secure), p is unchanged.*
    - *If* $\chi(c) = \mathsf{r}$ *(readable) 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. (This can be done by a trivial blackbox construction.)*[23] *This port automatically remains free, and thus the adversary connects to it. If p is an input port, it is unchanged.*
    - *If* $\chi(c) = \mathsf{i}$ *(insecure) and p is an input port, p is replaced by a new port* p^a*. (Thus both* p^a *and* p^c *become free, i.e., 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.*
  - *If* $v \notin \mathcal{H}$ *(i.e., there is no machine* $\mathsf{M}_{v,\mathcal{H}}$ *in* $M(\mathcal{H})$*), and p is an output port, it is unchanged. If it is an input port, it is renamed into* p^a*. (In both cases it becomes free, and thus the adversary can connect to it.)*

b) *The specified ports are* $S(\mathcal{H}) = \{\mathsf{in}_u?, \mathsf{out}_u! | u \in \mathcal{H}\}$*, i.e., the user ports of the correct machines.*

◇

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 systems Sys_1. In cases without tolerable imperfections, i.e., if adversaries should not have any advantage over honest users (although this seems unlikely with precise timing and a one-machine-per-user system, see Section 6), $\mathsf{TH}(\mathcal{H})$ would be the same in all these structures and only $S(\mathcal{H})$ would vary, i.e., be the same as in Sys_1.[24]

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 $\mathcal{H}$. In the comparison of the real and the ideal system, the intuitive idea is that the honest users use precisely the correct machines and thus the ports in $S(\mathcal{H})$. Corollary 5.4 shows that this is equivalent to the actual definition.[25]

3.2 Other Examples

In this section, we sketch a few examples that are not of the type defined in Section 3.1 and that motivated or tempted us to make the definitions more general.

3.2.1 Graceful Degradation

To see why it makes sense to allow several structures with the same set S in a system, and not to require that $f(M_1, S_1) = \{(M_2, S_2) \in Sys_2 | S_2 = S_1\}$, we consider graceful degradation with a constant interface. Graceful degradation, known from fault tolerance, means that there are

[23] We assume w.l.o.g. that there is a systematic naming scheme for such new ports (e.g., appending "d") that does not clash with prior names.

[24] We cannot use the original S^* in all ideal structures: Then a honest user with $Ports_{\mathsf{H}} \supseteq S^{*c}$ would be suitable for all structures $(M_1, S_1) \in Sys_1$. It would therefore use the ports of incorrect machines in the real system, but there an adversary can behave in a different way than the trusted host would.

[25] For many cryptographic examples, one could carry the specialization further, in particular for systems with two-party transactions. Those would be more similar to concrete examples for which reactive security has been proven before, e.g., [BeRo1 94, Shou 99]. One would define that each input contains the identity of a desired partner and starts a submachine (called "oracle" or "session"). However, at the level of this paper such specializations are not helpful.

different goals depending on the strength of the adversary. Fail-safe properties are the simplest case.

As a cryptographic example consider a third-party service, e.g., time-stamping, where the third-party machine T is fully automated and therefore does not need a user interface, i.e., ports in S. One requires a certain good service if T is correct, but often also makes weaker requirements for the other case, e.g., that T cannot entirely forge messages, only date them incorrectly. Hence we need two types of ideal structures, and the ideal structures defining the weaker requirements are not in the f-image of the real structures where T is correct, although they all have the same set S.[26]

3.2.2 Unmasked Replication

Now we present an example where it is useful to compare structures with different sets S_1 and S_2, both for $S_1 \subset S_2$ and $S_1 \supset S_2$. It is illustrated in Figure 3.

We consider a 3-of-5 system as in fault tolerance, e.g., five devices $\mathsf{M}_1, \ldots, \mathsf{M}_5$ making complicated physical measurements and related computations in a safety-critical control application. They should all make the same output, but they have different programs, hardware etc. The users accept an output if three devices agree. No single other device is trusted to make this majority decision. Hence the human user or the application software reads all five results, and therefore each device has an output port $\mathsf{p_i}!$, and these form the set S^* of the intended structure. (For brevity, we omit all inputs to the devices.) The real structures are derived as in Section 3.1, i.e., any subset of at least three device indices is a possible set $\mathcal{H}$.

Intuitively, the main difference to the cryptographic examples is that a natural honest user now has all five ports $\mathsf{p_i}?$ (i.e., expects to connect to all five ports $\mathsf{p_i}!$) because he does not know which of the devices are correct.

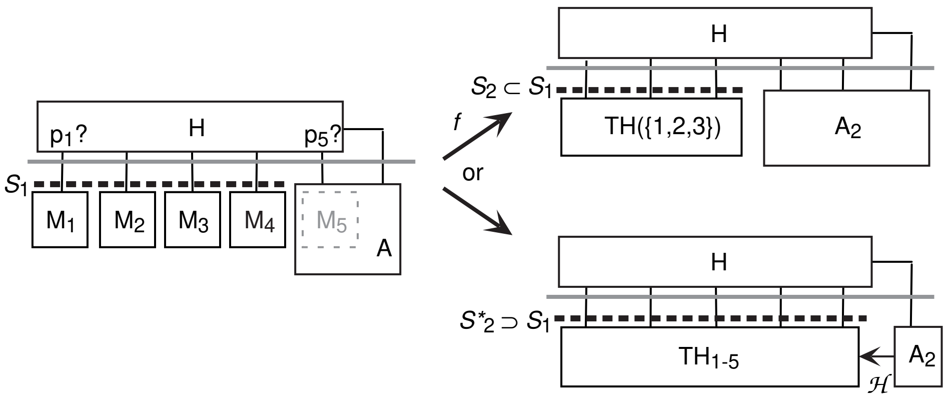

Figure 3: Unmasked replication

The two easiest ways to specify the ideal system are shown on the right side of Figure 3. First, one can define one structure $(\{\mathsf{TH}(\mathcal{H})\}, S_2(\mathcal{H}))$ for each 3-element set $\mathcal{H} \subseteq \{1, \ldots, 5\}$. Here $\mathsf{TH}(\mathcal{H})$ has the three ports $\mathsf{p_i}!$ with $i \in \mathcal{H}$, and simply outputs the correct result at these ports. A real structure with a set $\mathcal{H}_1$ of indices of correct devices is mapped by f onto one or all ideal structures with $\mathcal{H} \subseteq \mathcal{H}_1$.

Secondly, one can define only one structure $(\{\mathsf{TH}_{1-5}\}, S_2^*)$. The trusted host TH_{1-5} allows the adversary to choose a set $\mathcal{H}$ with at least three elements. Then it outputs the correct results at the ports $\mathsf{p_i}!$ with $i \in \mathcal{H}$, and values chosen by the adversary at the other ports.

[26] Clearly, such an example with only two service classes could also be modeled by considering the structures with and without T as different systems.

Of course, one can use additional ideal structures so that one ideal system has trusted hosts with three, four, and five of the ports $\mathsf{p_i}!$. However, as a specification we do not find it as intuitive as the two variants above, and the goal of a general model is to allow as short and intuitive specifications as possible.

3.2.3 Multicast

Many systems in cryptography and other fields use multicast channels. Hence we were tempted to include them as a primitive. However, it is well-known that based on point-to-point channels, arbitrary other channels can be modeled as machines. (However, switching then takes one round.) Multicast from a given machine can also be modeled by replicating the output port, and thus without delay.

Nevertheless, it would be an elegant primitive, e.g., at the user interface. For instance, the fact that a system outputs the same values to many users may be more intuitively specified by multicast from one port, and multicast from the environment would be a useful way to specify the input from the physical environment (the values to be measured) in Section 3.2.2. However, this requires a model where both ports and channels are explicitly specified, and to specify multicast at the user interface, the systems would end with free connections (see [PfSW 00]). Our main reason to give this up was that it necessitates consistency conditions in compositions, which get ugly together with the conditions on valid mappings.

3.2.4 Partially Correct Machines

In all examples so far, the real system was derived from an intended structure by taking different subsets of the intended machines (except for port renaming due to the channel model). A well-known different example is passive adversaries. Here a machine M_u is replaced by a machine M'_u that acts identically, but forwards all received messages and internal states to the adversary.

Another example is a fail-stop machine, e.g., a tamper-detecting device that an adversary may be able to destroy, but not to subvert. Here a machine M_u is replaced by a machine M''_u that acts identically, but has an additional free input port $\mathsf{stop_u}?$ and stops if 1 is input there.

In particular in fault tolerance, a whole range of other models is possible (e.g., independent faults). The same holds for channels. Hence we made the general model independent of any trust models and allow different trust models as specializations, as described in Section 3.1 and here.

4 Basic Lemmas

In this section, we prove a few basic but useful statements. The first lemma considers joining and separating machines within a configuration.

Lemma 4.1 (Combination of Machines) *The following general facts are true about combinations of machines:*

a) For a subset $D \subseteq C$ of a collection, we define two combinations $\mathsf{D_o}$ and $\mathsf{D_h}$ (called open and hiding) of D into one machine: Both have all the original machines as submachines. While $Ports_{D_o} = \mathsf{ports}(D)$, in $\mathsf{D_h}$ all internal connections are hidden, i.e., $Ports_{D_h} = \mathsf{free}(D)$. Both machines are clocked whenever a machine from D is. The transition function is defined by switching the submachines, and in $\mathsf{D_h}$ the internal connections, just as they would be switched externally.

In the resulting collection $C^* = C \setminus D \cup \{\mathsf{D}\}$, where $\mathsf{D} = \mathsf{D_o}$ or $\mathsf{D_h}$, the restriction of the runs to any tuples of the original machines or ports is the same as in C. (The runs as such get a slightly different representation.)

b) Such combinations are associative.

c) Such a combination of polynomial-time machines is polynomial-time.

d) If only D *is clocked in a continuous range of subrounds, i.e.,* $\kappa(i + j \bmod n) = \{\mathsf{D}\}$ *for* $j = 0, \ldots, l$*, one can instead clock it only once in these subrounds. The transition function of the new machine* D^* *is the appropriate concatenation of the individual transition functions together with the internal message transport. We can either retain the now empty subrounds or renumber the non-empty ones.*

The restriction of the runs to any tuples of the original machines or ports is still the same except for the renaming of subrounds.

e) The view of the combined machine D *or* D^* *can be identified with that of the set* D*. This implies that we can also say w.l.o.g. that the view of each* M_i *is a part of the view of* M^**.*

□

Proof. For Part a), it is clear that C^* is again a collection (and closed if C was). The rest would follow immediately from a more formal definition of the global transition functions. This also holds for Part b); recall that all machines belong to a collection C and thus have disjoint port names, hence there is no question of what gets connected first. For Part c), the number of steps of the combination is the sum of the steps of the submachines, plus an overhead for internal switching linear in the length of the messages written. All this is polynomial in the length of the initial states. Part d) would again follows directly from the definition. For Part e), recall that the view of D consists of all the states, inputs, and outputs of its machines. For D_o this is the same, and for D_h the difference is only that the messages on internal connections belong to the state, not to the in- and outputs. ■

In particular, combining machines of the same type in a configuration, e.g., several correct machines, gives a machine of that type again, and correct machines and a user give a correct machine.

Lemma 4.2 (Indistinguishability) *The following well-known facts are true about indistinguishability:*

a) Perfect indistinguishability of two families of random variables implies perfect indistinguishability of any function ϕ *of them (in particular restrictions). The same holds for statistical indistinguishability with any class SMALL, and for computational indistinguishability if* ϕ *is polynomial-time computable.*

b) Perfect indistinguishability implies statistical indistinguishability for any non-empty class SMALL, and statistical indistinguishability for a class SMALL $\subseteq$ *NEGL implies computational indistinguishability.*

c) Perfect and computational indistinguishability are equivalence relations, and so is statistical indistinguishability for a class SMALL closed under addition (which is true for both EXPSMALL and NEGL).

□

We omit the easy proofs.

Now we consider basic properties of the relation "at least as secure as."

Lemma 4.3 (Types of Security) *If* $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{perf}} Sys_2$*, then also* $Sys_1 \geq_{\mathsf{sec}}^{f,SMALL} Sys_2$ *for any non-empty class SMALL, and* $Sys_1 \geq_{\mathsf{sec}}^{f,SMALL} Sys_2$ *for a class* $SMALL \subseteq NEGL$ *implies* $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{poly}} Sys_2$*.* □

Proof. This follows immediately from Lemma 4.2, the fact that equality of possibly infinite views implies equality of all their fixed-length prefixes, that polynomial configurations only produce polynomial-length runs and that the distinguisher is a special case of a function ϕ. ■

Lemma 4.4 (Transitivity) *If* $Sys_1 \geq^{f_1} Sys_2$ *and* $Sys_2 \geq^{f_2} Sys_3$*, then* $Sys_1 \geq^{f_3} Sys_3$*, unless* f_3 *is not a valid mapping. Here* $f_3 := f_2 \circ f_1$ *is defined in a natural way:* $f_3(M_1, S_1)$ *is the union of the sets* $f_2(M_2, S_2)$ *with* $(M_2, S_2) \in f_1(M_1, S_1)$*.*

This holds for perfect and computational security, and for statistical security if SMALL is closed under addition. □

Proof. Let a configuration $conf_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^{f_3}(Sys_1)$ be given. Hence, by the definition of suitable configurations, if the name of a port p of H also occurs in some $(M_3, S_3) \in f_3(M_1, S_1)$, then $p^c \in S_3$.

If H has forbidden ports w.r.t. a structure $(M_2, S_2) \in f_1(M_1, S_1)$, we give these ports new names. By Remark 2.3, we obtain a configuration $conf_{\mathsf{t},1} = (M_1, S_1, \mathsf{H_t}, \mathsf{A}_{\mathsf{t},1}) \in \mathsf{Conf}^{f_1}(Sys_1)$, where $view_{conf_{\mathsf{t},1}}(\mathsf{H_t})$ equals $view_{conf_1}(\mathsf{H})$ except for the renaming.

Now there exists a configuration $conf_{\mathsf{t},2} = (M_2, S_2, \mathsf{H_t}, \mathsf{A}_{\mathsf{t},2}) \in \mathsf{Conf}(Sys_2)$ with $(M_2, S_2) \in f_1(M_1, S_1)$ such that $view_{conf_{\mathsf{t},1}}(\mathsf{H_t}) \approx view_{conf_{\mathsf{t},2}}(\mathsf{H_t})$.

As $\mathsf{H_t}$ only has ports from H and new ports, it has no forbidden ports w.r.t. any structure $(M_3, S_3) \in f_2(M_2, S_2)$, i.e., $conf_{\mathsf{t},2} \in \mathsf{Conf}^{f_2}(Sys_2)$. Hence there exists $conf_{\mathsf{t},3} = (M_3, S_3, \mathsf{H_t}, \mathsf{A}_{\mathsf{t},3}) \in \mathsf{Conf}(Sys_3)$ with $(M_3, S_3) \in f_2(M_2, S_2)$ and $view_{conf_{\mathsf{t},2}}(\mathsf{H_t}) \approx view_{conf_{\mathsf{t},3}}(\mathsf{H_t})$.

Together, we have $(M_3, S_3) \in f_3(M_1, S_1)$ by definition of f_3 and $view_{conf_{\mathsf{t},1}}(\mathsf{H_t}) \approx view_{conf_{\mathsf{t},3}}(\mathsf{H_t})$ because indistinguishability is transitive.

Finally, we must derive a configuration $conf_3 = (M_3, S_3, \mathsf{H}, \mathsf{A}_3)$ with the original user H. The new names of the changed ports do not occur in (M_3, S_3) by construction. Thus we can change them back iff the old names also do not occur in (M_3, S_3), i.e., in this case $view_{conf_3}(\mathsf{H})$ equals $view_{conf_{\mathsf{t},3}}(\mathsf{H_t})$ except for this renaming. We had assured that the name of a port p of H can only occur in (M_3, S_3) if $p^c \in S_3$. It was changed if p occurred in $\mathsf{forb}(M_2, S_2)$. As f_2 is a valid mapping, this is not possible together.

We conclude that $view_{conf_1}(\mathsf{H}) \approx view_{conf_3}(\mathsf{H})$ because the same renaming transforms these views into $view_{conf_{\mathsf{t},1}}(\mathsf{H_t})$ and $view_{conf_{\mathsf{t},3}}(\mathsf{H_t})$, respectively. ■

Lemma 4.5 (Reflexivity) *The relation* $\geq^f$ *is reflexive with the identity function* $f = \mathsf{id}$ *if one regards all free ports as specified, i.e., strengthens all structures to* $(M, \mathsf{free}(M))$*. Alternatively, one can rename all ports in* $\bar{S}$ *in one copy of the system (and modify the identity function accordingly), or disallow users that connect to unspecified ports.* □

The reason for the restrictions is that id must be a valid mapping, while the actual indistinguishability is clear. The first condition is

$$p^c \in \mathsf{free}(M) \Rightarrow p \notin \mathsf{ports}(M) \wedge p^c \notin \bar{S},$$

and the second is weaker. For $p^c \in S$ this follows directly from the definitions, while for $p^c \in \bar{S}$ it can be guaranteed or the case be excluded by the three possibilities mentioned.

5 Relations Among Model Variants

In this section, we define several variants of the model and investigate whether they are equivalent to our standard model. (By "standard model", "standard user" etc., we now always mean Definitions 2.4 to 2.8.) We have the following results:

- Universality and blackbox simulation make a difference (as far as we know).
- An additional output "guess" of the adversary never makes a difference.
- Auxiliary inputs already give universality except in a case with very powerful honest users; in universal and blackbox definitions they make no further difference.
- For the restricted class of valid mappings with $S_2 \subseteq S_1$ whenever $(M_2, S_2) \in f(M_1, S_1)$:
  - One only needs to consider users and no adversaries on the "left" side of the simulatability definition.
  - Universal and blackbox definitions are equivalent.
  - A "dialogue model" with unsynchronized users is equivalent to the standard model.
- Restricting honest users to the specified ports allows more general valid mappings and nicer reflexivity, but is equivalent for the standard valid mappings.

The reductions are quite efficient except in the proof that auxiliary inputs typically imply universality. This shows that our definition is quite general in the sense that it captures most of the versions that one might naturally consider.

5.1 Universal Variants

Our standard simulatability allows a completely different adversary A_2 for each given configuration $(M_1, S_1, \mathsf{H}, \mathsf{A}_1)$. This corresponds best to the intuition that everything that can happen to a user with a structure from Sys_1 can also happen to him with a corresponding structure of Sys_2. However, all typical examples fulfil stronger definitions where the construction of A_2 is more universal. We first show a definition where A_2 is only independent of H.

Definition 5.1 (Universal Simulatability) *We call Sys_1* universally perfectly *at least as secure as Sys_2 for a valid mapping f and write $\geq^{f,\mathsf{perf}}_{\mathsf{sec,uni}}$, if for any structure $(M_1, S_1) \in Sys_1$ and adversary A_1, there exists a structure $(M_2, S_2) \in f(M_1, S_1)$ and an adversary A_2 such that for all users H for which $conf_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^f(Sys_1)$, we have $conf_2 = (M_2, S_2, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(Sys_2)$ and $view_{conf_1}(\mathsf{H}) = view_{conf_2}(\mathsf{H})$.*

We call Sys_1 universally statistically *at least as secure as Sys_2, $\geq^{f,SMALL}_{\mathsf{sec,uni}}$, if the same holds with statistical indistinguishability of all families of polynomial-length prefixes of the views, and* universally computationally*, $\geq^{f,\mathsf{poly}}_{\mathsf{sec,uni}}$, if it holds only for all polynomial-time adversaries A_1 and users H, with polynomial-time A_2, and with computational indistinguishability. Indices of $\geq$ are omitted if they are clear from the context.* ◇

By this definition, each adversary is only universal for all users with a fixed set of ports. This is no significant restriction: The relevant aspects are only how the ports from $\mathsf{free}(M_1)$ and $\mathsf{free}(M_2)$ are connected. For this, there is only a finite number of choices (and we see in Sections 5.4 and 5.6 that we can usually even fix one choice). All other communication between H and A could be multiplexed over one connection with a fixed name.

Note that, if an adversary A_1 forces users to have forbidden ports, i.e., $\mathsf{free}(M_1 \cup \{\mathsf{A}_1\})^c$ contains ports from $\mathsf{forb}(M_2, S_2)$, then nothing needs to be shown for it because always $conf_1 \notin \mathsf{Conf}^f(Sys_1)$.

Obviously Definition 5.1 implies Definition 2.8, while the reverse is not necessarily true. A possibly even stronger type of universality is achieved by *blackbox* simulations, similar to [GoOr 94]. A reason to make the intermediate universal definition, too, is that it may be realistic that the adversaries do not know the user behaviour, but that they know how they would act in the other system.

Definition 5.2 (Blackbox Simulatability) *We call Sys_1* blackbox perfectly at least as secure as *Sys_2 for a valid mapping f and write $\geq_{\mathsf{sec,b}}^{f,\mathsf{perf}}$, if for each structure $(M_1, S_1) \in Sys_1$ and set P of ports, called adversary ports, there is a structure $(M_2, S_2) \in f(M_1, S_1)$ and a fixed machine Sim called simulator such that for any $conf_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}) \in \mathsf{Conf}^f(Sys_1)$ with $Ports_\mathsf{A} = P$, we have $conf_2 = (M_2, S_2, \mathsf{H}, \mathsf{Sim}(\mathsf{A})) \in \mathsf{Conf}(Sys_2)$ and $view_{conf_1}(\mathsf{H}) = view_{conf_2}(\mathsf{H})$. Here $\mathsf{Sim}(\mathsf{A})$ denotes that Sim gets A as a blackbox submachine.*

We call Sys_1 blackbox statistically *at least as secure as Sys_2 if the same holds with statistical indistinguishability of all families of polynomial-length prefixes of the views, and* blackbox computationally *if it holds with a polynomial-time simulator Sim, for polynomial-time A and H, and with computational indistinguishability.* ◇

The fact that the simulator is only universal for each set P is essentially w.l.o.g., similar to the arguments in the universal case.

Recall that by a blackbox submachine we mean that the state-transition function is given as a blackbox, so that Sim can clock A. It could also reset (typically called "rewind") A to a prior state, but we do not use this. In a reactive scenario a simulator that rewinds back by more than one switching step cannot be indistinguishable unless we make restrictions on the user machines H: We allow adversaries to produce outputs to H and to expect answers from H in each step. Hence any rewinding will be noticed by some H. (One cannot allow Sim to also rewind H, both from a real-life point of view and because Definition 5.2 should be a stronger version of Definition 2.8.)[27]

Corollary 5.1 (Basic Lemmas Universally) *Lemmas 4.3, 4.4 and 4.5 are also true for universal and blackbox simulatability.* □

Proof. The only property for which this is not immediately clear is transitivity. We want to show that the final adversary A_3 only depends on (M_1, S_1) and A_1. For this, note that the renaming from A_1 to $\mathsf{A}_{\mathsf{t},1}$ can also be described using only the ports of A_1 and the structures, i.e., all those ports of A_1 are renamed that are complements of forbidden ports in any corresponding (M_2, S_2). For $\mathsf{A}_{\mathsf{t},2}$ and $\mathsf{A}_{\mathsf{t},3}$ we use the given universality, and the last renaming into A_3 is the reverse of the first. Renaming can also be done as a blackbox construction. ■

5.2 Output of Guess by the Adversary

In the following guessing-output model, we also consider a final output of the adversary, or all outputs the adversary makes to some unconnected result port. This is like the guessing-outputs in semantic security [GoMi 84], and essential in current definitions of multi-party function evaluation. It corresponds to the intuition that privacy means "whatever the *adversary* sees is simulatable," while "whatever the honest users see is simulatable" is integrity. However, we show that it makes no difference in the reactive case. Intuitively, with general active attacks, anything the adversary sees can come back to some honest users.

The definitions are modified as follows: Systems and valid mappings are unchanged. The configurations are denoted $\mathsf{Conf}_\mathsf{g}(Sys)$; the difference to $\mathsf{Conf}(Sys)$ is that for $conf_\mathsf{g} = (M, S,$

[27]The fact that our definitions automatically rule out rewinding seems to be an important reason why we can obtain a general composition theorem.

$\mathsf{H}, \mathsf{A_g}) \in \mathsf{Conf_g}(Sys)$, the adversary $\mathsf{A_g}$ has a distinguished unconnected port guess! (compare Figure 4). By $view_{conf_\mathsf{g}}(\mathsf{H}, \mathsf{guess!})$ we denote the family of restrictions of the runs to the view of H and the outputs at guess!. We obtain the following simulatability definition:

Definition 5.3 (Simulatability with Guessing Output) *Let two systems Sys_1 and Sys_2 and a valid mapping f be given. We call Sys_1 (perfectly, statistically, or computationally) at least as secure as Sys_2 for f in the guessing-output model, $Sys_1 \geq_\mathsf{g}^f Sys_2$, if for any configuration $conf_{\mathsf{g},1} = (M_1, S_1, \mathsf{H}, \mathsf{A}_{\mathsf{g},1}) \in \mathsf{Conf}_\mathsf{g}^f(Sys_1)$, there exists $conf_{\mathsf{g},2} = (M_2, S_2, \mathsf{H}, \mathsf{A}_{\mathsf{g},2}) \in \mathsf{Conf_g}(Sys_2)$ with $(M_2, S_2) \in f(M_1, S_1)$ such that*

$$view_{conf_{\mathsf{g},1}}(\mathsf{H}, \mathsf{guess!}) \approx view_{conf_{\mathsf{g},2}}(\mathsf{H}, \mathsf{guess!}).$$

Recall that $\approx$ means perfect, statistical, or computational indistinguishability, respectively, and for the statistical case refers to all families of polynomial-length prefixes. $\diamond$

We again omit the indices f and sec at $\geq$ if they are clear from the context, and write $conf_{\mathsf{g},2} \in Indist_\mathsf{g}^f(conf_{\mathsf{g},1})$. The universal and blackbox definitions are adapted in the same way.

Theorem 5.1 (Guessing-Output) *For all systems Sys_1, Sys_2 and valid mappings f, we have*

$$Sys_1 \geq_\mathsf{g}^f Sys_2 \quad \Leftrightarrow \quad Sys_1 \geq^f Sys_2.$$

This is true for perfect, statistical and computational security, and also in the universal and blackbox models. $\square$

Proof. We can treat perfect, statistical and computational security together. In the third case all given adversaries and users are polynomial-time; this will imply that so are all the constructed ones, see Lemma 4.1.

"$\geq_\mathsf{g} \Rightarrow \geq$": (This is the easy direction.) Let $Sys_1 \geq_\mathsf{g} Sys_2$, and let $conf_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^f(Sys_1)$ be given. Let $\mathsf{A}_{\mathsf{g},1}$ be A_1 augmented by a port guess! (w.l.o.g., the name guess is new) where it does not make any outputs.[28] Then $conf_{\mathsf{g},1} = (M_1, S_1, \mathsf{H}, \mathsf{A}_{\mathsf{g},1}) \in \mathsf{Conf}_\mathsf{g}^f(Sys_1)$. Clearly, $view_{conf_{\mathsf{g},1}}(\mathsf{H}) = view_{conf_1}(\mathsf{H})$. By the precondition, there exists $conf_{\mathsf{g},2} = (M_2, S_2, \mathsf{H}, \mathsf{A}_{\mathsf{g},2}) \in \mathsf{Conf_g}(Sys_2)$ with $(M_2, S_2) \in f(M_1, S_1)$ and $view_{conf_{\mathsf{g},1}}(\mathsf{H}, \mathsf{guess!}) \approx view_{conf_{\mathsf{g},2}}(\mathsf{H}, \mathsf{guess!})$. This implies $view_{conf_{\mathsf{g},1}}(\mathsf{H}) \approx view_{conf_{\mathsf{g},2}}(\mathsf{H})$ (Lemma 4.2). Let A_2 be $\mathsf{A}_{\mathsf{g},2}$ except that it suppresses the guessing-output. Then $conf_2 = (M_2, S_2, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(Sys_2)$, and clearly $view_{conf_2}(\mathsf{H}) = view_{conf_{\mathsf{g},2}}(\mathsf{H})$. Altogether, this implies $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$.

For the universal case, note that $\mathsf{A}_{\mathsf{g},1}$ is independent of H by construction, then $\mathsf{A}_{\mathsf{g},2}$ by the universal version of $\geq_\mathsf{g}^f$, and thus A_2 again by construction. In the blackbox model, we obtain $\mathsf{A}_{\mathsf{g},2}$ as a simulator Sim with $\mathsf{A}_{\mathsf{g},1}$ as a blackbox, and thus A_2 is Sim with A_1 as a blackbox.

"$\geq \Rightarrow \geq_\mathsf{g}$": Let $Sys_1 \geq Sys_2$, and let $conf_{\mathsf{g},1} = (M_1, S_1, \mathsf{H_g}, \mathsf{A}_{\mathsf{g},1}) \in \mathsf{Conf}_\mathsf{g}^f(Sys_1)$ be given. We construct a related configuration $conf_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1)$ where the former guessing output belongs to the user's view (see Figure 4): The adversary A_1 equals $\mathsf{A}_{\mathsf{g},1}$, and H is like $\mathsf{H_g}$ except for an additional input port guess?, where it ignores all inputs. We show $conf_1 \in \mathsf{Conf}^f(Sys_1)$: It is a closed collection because we required that no port guess? was there before and guess! was the only free port, and H has no forbidden ports because $\mathsf{H_g}$ hasn't.

Now clearly $view_{conf_1}(\mathsf{H}) = view_{conf_{\mathsf{g},1}}(\mathsf{H_g}, \mathsf{guess!})$.

Otherwise, there exists $conf_2 = (M_2, S_2, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(Sys_2)$ with $(M_2, S_2) \in f(M_1, S_1)$ and $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$. As assumed in the figure, A_2 must have a port guess! because H

[28]This and subsequent simple machine modifications can be made by canonical blackbox constructions; we tacitly assume that those are used.

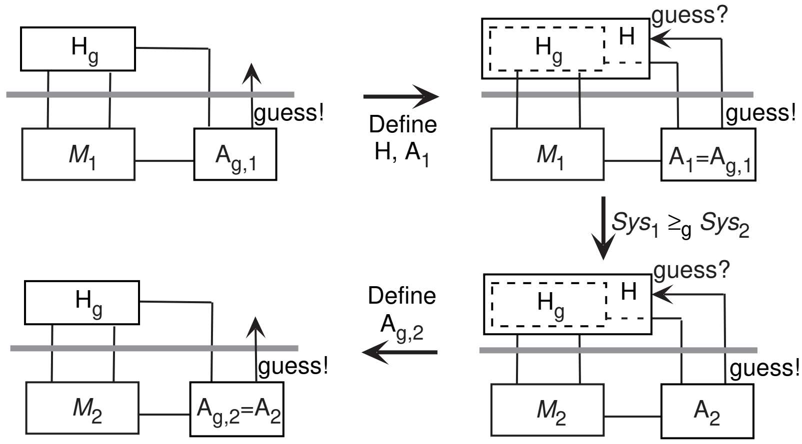

Figure 4: Standard simulatability implies simulatability with guessing outputs

has the port guess?, unconnected ports are not possible in $\mathsf{Conf}(Sys_2)$, and guess! cannot belong to M_2 or H (and thus $\mathsf{H_g}$) because the name guess was new.

Hence we can use $\mathsf{A_{g,2}} = \mathsf{A_2}$ to obtain a configuration $conf_{\mathsf{g},2} = (M_2, S_2, \mathsf{H_g}, \mathsf{A_{g,2}}) \in \mathsf{Conf_g}(Sys_2)$. As H behaves like $\mathsf{H_g}$, ignoring the inputs at guess?, the runs are essentially unchanged, in particular $view_{conf_{\mathsf{g},2}}(\mathsf{H_g}, \mathsf{guess!}) = view_{conf_2}(\mathsf{H})$.

Altogether this implies $view_{conf_{\mathsf{g},2}}(\mathsf{H_g}, \mathsf{guess!}) \approx view_{conf_{\mathsf{g},1}}(\mathsf{H_g}, \mathsf{guess!})$.

For the universal case, $\mathsf{A_1}$ does not depend on H by construction, and hence neither do $\mathsf{A_2}$ or $\mathsf{A_{g,2}}$. In the blackbox case, $\mathsf{A_{g,2}} = \mathsf{A_2}$ is a simulator with $\mathsf{A_1} = \mathsf{A_{g,1}}$ as a blackbox submachine. ■

Corollary 5.2 *The equivalent of Theorem 5.1 also holds in the model where adversaries always output their entire view at the port* guess!. □

The proof is identical to that of Theorem 5.1 except that in "$\geq_\mathsf{g} \Rightarrow \geq$", the adversary $\mathsf{A_{g,1}}$ outputs its view at the new port guess!.

5.3 Auxiliary Inputs

We now show that auxiliary inputs add only a small intermediate variation to the model. Recall that auxiliary inputs are important in definitions of zero-knowledge [Oren 87, ToW1 87], because they are needed to model the adversary's prior information about the user's secrets from earlier protocol runs. Intuitively, this is not necessary in a general reactive scenario because the adversary can also get such information from H by an active attack via their direct connections at the beginning of the current protocol.

Systems and valid mappings in the auxiliary-input model are as in the standard model. The configurations $conf_\mathsf{x} = (M, S, \mathsf{H_x}, \mathsf{A_x}) \in \mathsf{Conf_x}(Sys)$ are different: $\mathsf{H_x}$ and $\mathsf{A_x}$ have initial states $(1^k, aux_H)$ and $(1^k, aux_A)$ with arbitrary strings aux_H and aux_A.[29] For each such configuration, we obtain a family of views

$$view_{conf_\mathsf{x}}(\mathsf{H_x}) = (view_{conf_\mathsf{x},(k,aux_H,aux_A)}(\mathsf{H_x}))_{k\in\mathbb{N},aux_H\in\Sigma^*,aux_A\in\Sigma^*}$$

[29]The correct machines do not get an auxiliary input: Typical internal results from some history would not be the same in the real and ideal system and therefore no comparison could be made; a collection of related protocols is *one* reactive system. (This is different in the constructive approach to reactive simulatability mentioned in the introduction.) Results that are necessarily the same in the real and ideal system are modeled as outputs to the user and subsequent inputs.

where (k, aux_H, aux_A), written in the place of the initial states *ini* of all machines in Definition 2.3, stands for a vector $(1^k, \ldots, 1^k, (1^k, aux_H), (1^k, aux_A))$.

For perfect security, the families must be equal. For computational security, as we are using uniform complexity, we proceed as in [Gold 93]: We assume that the auxiliary inputs have been generated from k by an arbitrary polynomial-time algorithm $\mathsf{gen_x}$. As usual (see [Gold 93]), we define that one adversary $\mathsf{A_{x,2}}$ must deal with all auxiliary inputs, i.e., there is a certain universality. Hence auxiliary-input simulatability turns out to be closer to universal simulatability than to our standard definition.

Definition 5.4 (Simulatability with Auxiliary Inputs) *Let two systems Sys_1 and Sys_2 and a valid mapping f be given. We call Sys_1 (perfectly, statistically, or computationally) at least as secure as Sys_2 for f in the auxiliary-input model, $Sys_1 \geq_{\mathsf{x}}^f Sys_2$, if for any configuration $conf_{\mathsf{x},1} = (M_1, S_1, \mathsf{H_x}, \mathsf{A_{x,1}}) \in \mathsf{Conf}_{\mathsf{x}}^f(Sys_1)$, there exists $conf_{\mathsf{x},2} = (M_2, S_2, \mathsf{H_x}, \mathsf{A_{x,2}}) \in \mathsf{Conf_x}(Sys_2)$ with $(M_2, S_2) \in f(M_1, S_1)$ such that:*

a) For perfect security:

$$view_{conf_{\mathsf{x},1}}(\mathsf{H_x}) \;=\; view_{conf_{\mathsf{x},2}}(\mathsf{H_x}),$$

i.e., for any k, aux_H, and aux_A, the views in the two configurations have the same distribution.

b) For statistical security: For any polynomial l over $\mathbb{N}$(denoting the length of considered runs), there exists a function $g \in SMALL$ such that for all k, aux_H, and aux_A

$$\Delta(view_{conf_{\mathsf{x},1},(k,aux_H,aux_A),l(k)}(\mathsf{H_x}), view_{conf_{\mathsf{x},2},(k,aux_H,aux_A),l(k)}(\mathsf{H_x})) \leq g(k).$$

c) For computational security (with $\mathsf{Conf_{x,poly}}$ instead of $\mathsf{Conf_x}$ above): for any polynomial-time algorithm $\mathsf{gen_x}$,

$$(view_{conf_{\mathsf{x},1},\mathsf{gen_x}(k)}(\mathsf{H_x}))_{k\in\mathbb{N}} \quad \approx_{\mathsf{poly}} \quad (view_{conf_{\mathsf{x},2},\mathsf{gen_x}(k)}(\mathsf{H_x}))_{k\in\mathbb{N}}.$$

The notation $Indist_{\mathsf{x}}^f$ is defined accordingly. The universal and blackbox definitions are adapted in the same way, i.e., the Parts a)-c) here replace the three types of indistinguishability in Definitions 5.1 and 5.2. We then write $\geq_{\mathsf{x,uni}}^f$ and $\geq_{\mathsf{x,b}}^f$. ◇

Remark 5.1. For any function $\mathsf{gen_x}$ from $\mathbb{N}$ to triples (k, aux_H, aux_A), Definition 5.4a implies $(view_{conf_{\mathsf{x},1},\mathsf{gen_x}(k)}(\mathsf{H_x}))_{k\in\mathbb{N}} = (view_{conf_{\mathsf{x},2},\mathsf{gen_x}(k)}(\mathsf{H_x}))_{k\in\mathbb{N}}$. Similarly, Definition 5.4b implies $(view_{conf_{\mathsf{x},1},\mathsf{gen_x}(k),l(k)}(\mathsf{H_x}))_{k\in\mathbb{N}} \approx^{SMALL} (view_{conf_{\mathsf{x},2},\mathsf{gen_x}(k),l(k)}(\mathsf{H_x}))_{k\in\mathbb{N}}$ for all polynomials l. For perfect security, this is clearly equivalent to the definition. Where we want to treat these three cases together, we write

$$view_{conf_{\mathsf{x},1},\mathsf{gen_x}}(\mathsf{H_x}) \;\approx\; view_{conf_{\mathsf{x},2},\mathsf{gen_x}}(\mathsf{H_x}).$$

∘

The distinguisher in Part c) of the definition sees the auxiliary input aux_H in the view of H. Hence, although the auxiliary inputs are generated separately with $\mathsf{gen_x}$ on both sides of the comparison, the views are not only similar on average, but for the same values aux_H. To show that this also holds for aux_A, we prove the theorem immediately also for the model with guessing outputs, so that the distinguisher can see aux_A in the outputs at $\mathsf{guess!}$.

Theorem 5.2 (Auxiliary Inputs) *Let two systems Sys_1 and Sys_2 and a valid mapping f be given. The following holds for all security types introduced, i.e., perfect, statistical and computational security, and also in the guessing-output model.*

a) If we restrict all configurations to users whose state-transition functions have a Turing-machine representation for each k, then $Sys_1 \geq_{\mathsf{x}}^f Sys_2 \Leftrightarrow Sys_1 \geq_{\mathsf{uni}}^f Sys_2$.

b) The universal model and the blackbox model are not changed by the addition of auxiliary inputs, i.e., $Sys_1 \geq_{\mathsf{x,uni}}^f Sys_2 \Leftrightarrow Sys_1 \geq_{\mathsf{uni}}^f Sys_2$ and $Sys_1 \geq_{\mathsf{x,b}}^f Sys_2 \Leftrightarrow Sys_1 \geq_{\mathsf{b}}^f Sys_2$.

c) If we quantified over an algorithm $\mathsf{gen_x}$ (or even individual auxiliary inputs) before $\mathsf{A_{x,2}}$ in Definition 5.4, we would remain in the standard model.

□

The restriction in Part a) is automatically fulfilled in the computational case, but not in the others: General machines can produce strings of any length for any k after a sufficient number of rounds. However, realistic users will have space or time limitations anyway.[30]

Proof. We treat the models with and without guessing output together. For this, we let "standard model" mean either one in this proof and we omit all indices g, but we always write $view_{conf}(\mathsf{H}, \mathsf{guess!})$. We sometimes distinguish the perfect, statistical and computational case.

Part a) "$\geq_{\mathsf{x}} \Rightarrow \geq_{\mathsf{uni}}$": Let $Sys_1 \geq_{\mathsf{x}}^f Sys_2$, and let a structure $(M_1, S_1) \in Sys_1$ and a compatible adversary A_1 be given. We can assume that $\mathsf{free}(M_1 \cup \{\mathsf{A}_1\})^c$ contains no ports from $\mathsf{forb}(M_2, S_2)$ for a structure $(M_2, S_2) \in f(M_1, S_1)$; otherwise we need not show anything (recall the text below Definition 5.1).

We construct a related suitable configuration $conf_{\mathsf{x},1} = (M_1, S_1, \mathsf{H_x}, \mathsf{A_{x,1}}) \in \mathsf{Conf}_{\mathsf{x}}^f(Sys_1)$, see Figure 5: $\mathsf{A_{x,1}}$ simply ignores its auxiliary input and acts like A_1. $\mathsf{H_x}$ is a universal Turing machine with $Ports_{\mathsf{H_x}} = \mathsf{free}(M_1 \cup \{\mathsf{A}_1\})^c \setminus \{\mathsf{guess?}\}$. It interprets its auxiliary input as a machine H and executes H on its own (i.e., $\mathsf{H_x}$'s) initial state k. For the computational case, $\mathsf{H_x}$ also expects a value k' in unary as part of the auxiliary input and executes at most k' steps of H.[31] The precondition on A_1 guarantees that $conf_{\mathsf{x},1}$ is indeed suitable.

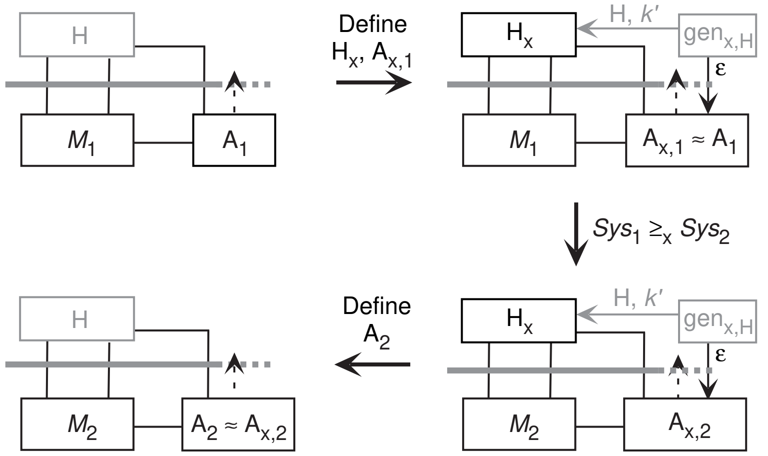

Figure 5: Simulatability with auxiliary inputs implies universal simulatability. Gray machines are fixed later (universality). The dashed adversary output is a possible guessing output.

Now there exists $conf_{\mathsf{x},2} = (M_2, S_2, \mathsf{H_x}, \mathsf{A_{x,2}}) \in Indist_{\mathsf{x}}^f(conf_{\mathsf{x},1})$. By Remark 5.1, this implies $view_{conf_{\mathsf{x},1}, \mathsf{gen_x}}(\mathsf{H_x}, \mathsf{guess!}) \approx view_{conf_{\mathsf{x},2}, \mathsf{gen_x}}(\mathsf{H_x}, \mathsf{guess!})$ for any function or polynomial-time algorithm $\mathsf{gen_x}$, respectively.

[30] However, the proof of Part a) also significantly increases the complexity of the user, hence it is of smaller practical relevance than our other theorems.

[31] Universal Turing machines are polynomial in the run-time of the simulated machines. But $\mathsf{H_x}$ must be polynomial in its initial state alone.

We claim that the machine A_2 that acts like $\mathsf{A}_{\mathsf{x},2}$ with $aux_A = \epsilon$ fulfils the conditions for universal simulatability. Let any machine H be given. We construct functions $\mathsf{gen}_{\mathsf{x},\mathsf{H}}$ from $\mathbb{N}$ to triples (k, aux_H, aux_A). For the perfect or statistical case, let $\mathsf{gen}_{\mathsf{x},\mathsf{H}}(k) = (k, \mathsf{H}(k), \epsilon)$, where H (k) means a Turing program for H on input k. For the computational case, let R be a polynomial bounding the running time of H. Then let $\mathsf{gen}_{\mathsf{x},\mathsf{H}}(1^k) = (k, (\mathsf{H}, 1^{R(k)}), \epsilon)$. (Here the Turing program of H is constant.) This is a polynomial-time algorithm. In both cases, for any k, H_x with the input from $\mathsf{gen}_{\mathsf{x},\mathsf{H}}$ has the same input-output behaviour at its ports as H. Thus the runs of a configuration $conf_{\mathsf{x},i}$ (i = 1, 2) only differ from those of $conf_i$ in the internal states of this machine (and a small change between the internal states of $\mathsf{A}_{\mathsf{x},i}$ and A_i). Furthermore, the internal states of H_x contain those that H would have in the same runs. Hence (with Lemma 4.1) we obtain $view_{conf_1}(\mathsf{H}, \mathsf{guess!}) \approx view_{conf_2}(\mathsf{H}, \mathsf{guess!})$.

"$\geq_{\mathsf{uni}} \Rightarrow \geq_\mathsf{x}$": This follows from Part b), where we even show that $\geq_{\mathsf{uni}}$ implies $\geq_{\mathsf{x},\mathsf{uni}}$.

Parts b) and c) In both directions, we first consider Part c) and then show that the constructions are also valid in the universal and blackbox model.

"$\geq_\mathsf{x} \Rightarrow \geq$": (This is the easy direction.) Let a configuration $conf_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^f(Sys_1)$ be given. Let H_x and $\mathsf{A}_{\mathsf{x},1}$ be equal to H and A_1 except for extra components aux_H or aux_A in the initial states which they ignore. Then $conf_{\mathsf{x},1} = (M_1, S_1, \mathsf{H}_\mathsf{x}, \mathsf{A}_{\mathsf{x},1}) \in \mathsf{Conf}_\mathsf{x}^f(Sys_1)$, and $view_{conf_{\mathsf{x},1}}(\mathsf{H}_\mathsf{x}, \mathsf{guess!})$ consists of $view_{conf_1}(\mathsf{H}, \mathsf{guess!})$ and aux_H. By the precondition there exists $conf_{\mathsf{x},2} = (M_2, S_2, \mathsf{H}_\mathsf{x}, \mathsf{A}_{\mathsf{x},2}) \in Indist_\mathsf{x}^f(conf_{\mathsf{x},1})$. The indistinguishability holds in particular for the generation algorithm $\mathsf{gen}_{\mathsf{eps}}$ that always outputs (ϵ, ϵ) (by Remark 5.1). For this case, we construct a related configuration $conf_2 = (M_2, S_2, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(Sys_2)$ by letting A_2 act like $\mathsf{A}_{\mathsf{x},2}$ with $aux_A = \epsilon$. In particular, $view_{conf_2}(\mathsf{H}, \mathsf{guess!})$ equals $view_{conf_{\mathsf{x},2}}(\mathsf{H}_\mathsf{x}, \mathsf{guess!})$ except for lacking aux_H. Altogether, we therefore have $view_{conf_2}(\mathsf{H}, \mathsf{guess!}) \approx view_{conf_1}(\mathsf{H}, \mathsf{guess!})$.

In the universal case, as $\mathsf{A}_{\mathsf{x},1}$ is independent of H, so is $\mathsf{A}_{\mathsf{x},2}$ and thus A_2. In the blackbox case, $\mathsf{A}_{\mathsf{x},2}$ consists of a simulator Sim and $\mathsf{A}_{\mathsf{x},1}$ as a blackbox, and thus A_2 of Sim and A_1.

"$\geq \Rightarrow \geq_\mathsf{x}$": Let $conf_{\mathsf{x},1} = (M_1, S_1, \mathsf{H}_\mathsf{x}, \mathsf{A}_{\mathsf{x},1}) \in \mathsf{Conf}_\mathsf{x}^f(Sys_1)$ be given and a generation algorithm or function gen_x. We construct a configuration $conf_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^f(Sys_1)$ by letting H perform the generation in Round [0.1], see Figure 6. It sends aux_A to A_1 on a connection between two new (w.l.o.g.) ports auxa! and auxa?. A_1 simply gives its input from auxa? to its submachine $\mathsf{A}_{\mathsf{x},1}$ as auxiliary input aux_A in its initial state before first clocking it in Round [0.2]. Hence $view_{conf_1}(\mathsf{H}, \mathsf{guess!})$ consists of $view_{conf_{\mathsf{x},1}, \mathsf{gen}_\mathsf{x}}(\mathsf{H}_\mathsf{x}, \mathsf{guess!})$ and aux_A.

By the precondition there exists a configuration $conf_2 = (M_2, S_2, \mathsf{H}, \mathsf{A}_2)$ with $(M_2, S_2) \in f(M_1, S_1)$ and $view_{conf_1}(\mathsf{H}, \mathsf{guess!}) \approx view_{conf_2}(\mathsf{H}, \mathsf{guess!})$. Here A_2 must have a port auxa? because H's port auxa! cannot be unconnected, nor connected to H or M_2 because its name was new.

We transform this into a configuration $conf_{\mathsf{x},2} = (M_2, S_2, \mathsf{H}_\mathsf{x}, \mathsf{A}_{\mathsf{x},2}) \in \mathsf{Conf}_\mathsf{x}(Sys_2)$ where $\mathsf{A}_{\mathsf{x},2}$ equals A_2 except that it does not have the port auxa?, but expects an auxiliary input, which it forwards to this ports of its submachine A_2. Then $view_{conf_2}(\mathsf{H}, \mathsf{guess!})$ consists of $view_{conf_{\mathsf{x},2}, \mathsf{gen}_\mathsf{x}}(\mathsf{H}_\mathsf{x}, \mathsf{guess!})$ and aux_A.

Altogether, we have $view_{conf_{\mathsf{x},1}, \mathsf{gen}_\mathsf{x}}(\mathsf{H}_\mathsf{x}, \mathsf{guess!}) \approx view_{conf_{\mathsf{x},2}, \mathsf{gen}_\mathsf{x}}(\mathsf{H}_\mathsf{x}, \mathsf{guess!})$.

For the universal case, note that A_1 is independent of H_x and gen_x. Hence so is A_2, and thus $\mathsf{A}_{\mathsf{x},2}$. In the blackbox case, A_2 consists of a simulator Sim with A_1 as a blackbox. Then $\mathsf{A}_{\mathsf{x},2}$ is Sim with $\mathsf{A}_{\mathsf{x},1}$ as a blackbox without even any renaming. ■

5.4 Users Only in Real Systems

The following model variant, called user-only model, only makes sense if $S_2 \subseteq S_1$ whenever $(M_2, S_2) \in f(M_1, S_1)$. (Recall that the standard case is even $S_2 = S_1$.) We show that in this case, it is also equivalent to our standard model.

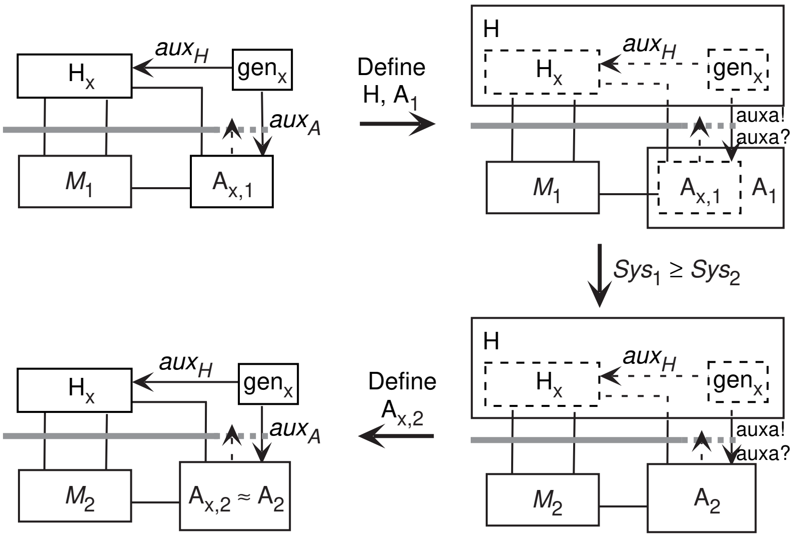

Figure 6: Standard simulatability implies simulatability with auxiliary inputs, computational case. The dashed adversary output is the guessing output.

In the user-only model, we consider users alone on the "left" side of a comparison of two systems, see Figure 7.[32] Formally, the systems are the standard ones, while user-only-valid mappings are valid mappings where additionally $S_2 \subseteq S_1 \Rightarrow (M_2, S_2) \in f(M_1, S_1)$. We define the set $\mathsf{Conf_u}(Sys)$ of user-only configurations as the subset of configurations $(M, S, \mathsf{H}, \mathsf{A_{null}}) \in \mathsf{Conf}(Sys)$, where $\mathsf{A_{null}}$ is a fixed machine without ports that does nothing.[33] User-only simulatability, written $\geq_{\mathsf{u}}^{f}$, is defined just like Definition 2.8 except that $conf_1$ is only chosen from $\mathsf{Conf}_{\mathsf{u}}^{f}(Sys_1)$.

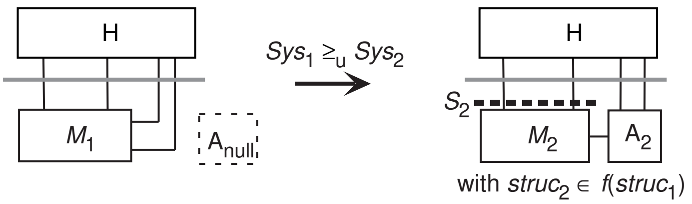

Figure 7: Definition of user-only model

Remark 5.2. We have $\mathsf{Conf}_{\mathsf{u}}^{f}(Sys_1) = \mathsf{Conf_u}(Sys_1)$ for any user-only-valid mapping f: For $conf_{\mathsf{u},1} = (M_1, S_1, \mathsf{H}, \mathsf{A_{null}}) \in \mathsf{Conf_u}(Sys_1)$, we have $Ports_{\mathsf{H}}^{c} = \mathsf{free}(M_1)$ (closed collection). Hence by the first condition on valid mappings, no port $p \in Ports_{\mathsf{H}}$ can lie in $\mathsf{forb}(M_2, S_2)$. Thus $conf_{\mathsf{u},1} \in \mathsf{Conf}_{\mathsf{u}}^{f}(Sys_1)$.

Remark 5.3. Universal and blackbox user-only simulatability are identical: As the adversary $\mathsf{A_1} = \mathsf{A_{null}}$ is fixed, both require that for any structure $(M_1, S_1) \in Sys_1$, there is one machine $\mathsf{A_2}$ such that $(M_2, S_2, \mathsf{H}, \mathsf{A_2}) \in Indist^f(M_1, S_1, \mathsf{H}, \mathsf{A_{null}})$ for all H.

Remark 5.4. The reason why the user-only model makes no sense for $S_2 \not\subseteq S_1$ is that we always want to be able to consider users that use all ports from S_2. (Recall the example from Sec-

[32] We could also call H a combined user-adversary machine, but we treat it like a user in the clocking scheme and by keeping it constant in a simulation. We could also call it an adversary and regard $\mathsf{A_2}$ as a blackbox simulator with relatively little power over its blackbox (H) because some connections between correct machines and blackbox are unchanged.

[33] We could also define $\mathsf{Conf_u}(Sys)$ as a set of triples, but some proofs are simplified by having a subset of the standard configurations.

tion 3.2.2 with four correct devices and a 5-port trusted host.) In this case, such users cannot interact with (M_1, S_1) alone. (We will also see that the following proof does not work for them.) $\circ$

Theorem 5.3 *Let systems Sys_1 and Sys_2 and a user-only-valid mapping f be given. Then $Sys_1 \geq_{\mathsf{u}}^f Sys_2 \Leftrightarrow Sys_1 \geq^f Sys_2$. This is true for perfect, statistical and computational security, and also for universal and blackbox definitions.* $\Box$

Proof. **"$\geq \Rightarrow \geq_{\mathsf{u}}$":** This is clear because user-only simulatability is defined as a weaker version of standard simulatability. This is also true for the universal and blackbox case.

"$\geq_{\mathsf{u}} \Rightarrow \geq$": Let a configuration $conf_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}(Sys_1)$ be given. To transform it into a user-only configuration, we combine H and A_1 into a machine $\mathsf{H_u}$, see Figure 8. We define $\mathsf{H_u}$ as a hiding combination according to Lemma 4.1a). Then only $\mathsf{H_u}$ is clocked in subrounds 2 to 4, and by Lemma 4.1d) we can instead clock $\mathsf{H_u}$ once in subround 3. Thus we get the correct clocking scheme for a user machine. (Internally it switches first A_1, then H, and then A_1 again in each subround [i.3].) Hence $conf_{\mathsf{u},1} = (M_1, S_1, \mathsf{H_u}, \mathsf{A_{null}}) \in \mathsf{Conf_u}(Sys_1) = \mathsf{Conf}_{\mathsf{u}}^f(Sys_1)$ and $view_{conf_1}(\mathsf{H})$ equals the subview of H in $view_{conf_{\mathsf{u},1}}(\mathsf{H_u})$ by Lemma 4.1e).

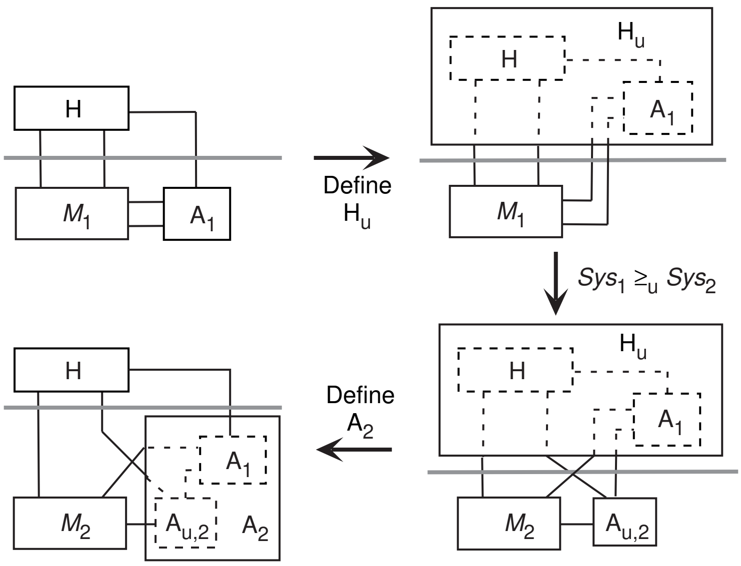

Figure 8: User-only simulatability implies standard simulatability

By the precondition, an indistinguishable configuration $conf_{\mathsf{u},2} = (M_2, S_2, \mathsf{H_u}, \mathsf{A}_{\mathsf{u},2}) \in \mathsf{Conf}(Sys_2)$ exists, i.e., $(M_2, S_2) \in f(M_1, S_1)$ and $view_{conf_{\mathsf{u},1}}(\mathsf{H_u}) \approx view_{conf_{\mathsf{u},2}}(\mathsf{H_u})$.

We want to transform this into a configuration $conf_2 = (M_2, S_2, \mathsf{H}, \mathsf{A}_2)$ with the original user H by splitting $\mathsf{H_u}$ again and then joining A_1 and $\mathsf{A}_{\mathsf{u},2}$. We must show that this gives a valid configuration.[34] We show below that the ports of all machines in M_2 and H, A_1 and $\mathsf{A}_{\mathsf{u},2}$ are disjoint. Then the proof finishes as follows: We consider the intermediate closed collection C where H, A_1 and $\mathsf{A}_{\mathsf{u},2}$ are all separate machines and the clocking scheme is $(M_2 \cup \{\mathsf{H}\}, \{\mathsf{A}_{\mathsf{u},2}\}, \{\mathsf{A}_1\}, \{\mathsf{H}\}, \{\mathsf{A}_1\}, \{\mathsf{A}_{\mathsf{u},2}\})$. Then taking the hiding combination of H and A_1, and joining subrounds 3 to 5 gives $conf_{\mathsf{u},2}$ with its correct clocking scheme, while taking the hiding combination of A_1 and $\mathsf{A}_{\mathsf{u},2}$, and joining subrounds 2 with 3 and 5 with 6 gives a valid

[34]This would not be true for $S_2 \not\subseteq S_1$ and if H had a port p with $p^c \in S_2 \setminus S_1$ and which is connected to A_1 in $conf_1$. The connection would remain hidden in $\mathsf{H_u}$ in $conf_{\mathsf{u},1}$ and $conf_{\mathsf{u},2}$, and we would get a name clash in $conf_2$. Renaming does not help because H is connected to A_1 at p, but should be connected to M_2 because p^c is specified. (These are ports where the user intends to use the trusted host.)

configuration $conf_2 \in \mathsf{Conf}(Sys_2)$. Then, by Lemma 4.1e), $view_{conf_2}(\mathsf{H})$ equals the subview of H in $view_{conf_{\mathsf{u},2}}(\mathsf{H_u})$.

Altogether, this implies $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$.

Disjoint ports. We still have to show that the ports of all machines in M_2 and H, A_1 and $\mathsf{A}_{\mathsf{u},2}$ are disjoint.

- For H and M_2 this holds because $conf_1$ is suitable (Definition 2.6), for A_1 and H because $conf_1$ is a collection, and for $\mathsf{A}_{\mathsf{u},2}$ and M_2 because $conf_{\mathsf{u},2}$ is a collection.

- $\mathsf{A}_{\mathsf{u},2}$ and H: Let $p \in Ports_{\mathsf{A}_{\mathsf{u},2}}$. Then $p^c \in Ports_{\mathsf{H_u}}$ or $p^c \in \mathsf{free}(M_2)$ because $conf_{\mathsf{u},2}$ is closed.

  If $p^c \in Ports_{\mathsf{H_u}}$, then $p \in \mathsf{ports}(M_1)$ because $conf_{\mathsf{u},1}$ is a user-only configuration. Thus $p \notin Ports_{\mathsf{H}}$ because $conf_1$ is a collection. Note that then also $p \notin Ports_{\mathsf{A}_1}$.

  Now let $p^c \in \mathsf{free}(M_2)$, and assume $p \in Ports_{\mathsf{H}}$. Then $p^c \notin \bar{S}_2$ because $conf_1$ is suitable. Thus $p^c \in S_2 \subseteq S_1 \subseteq \mathsf{ports}(M_1)$. Hence p is connected to M_1 in $conf_1$ and becomes a port of $\mathsf{H_u}$. This contradicts the precondition $p \in Ports_{\mathsf{A}_{\mathsf{u},2}}$ because p cannot occur twice in $conf_{\mathsf{u},2}$.

- A_1 and M_2: Let $p \in Ports_{\mathsf{A}_1}$. Then $p^c \in \mathsf{free}(M_1)$ or $p^c \in Ports_{\mathsf{H}}$ because $conf_1$ is closed.

  If $p^c \in \mathsf{free}(M_1)$, then p becomes a port of $\mathsf{H_u}$. Hence $p \notin \mathsf{ports}(M_2)$ because $conf_{\mathsf{u},2}$ is a collection. Note that then also $p \notin Ports_{\mathsf{A}_{\mathsf{u},2}}$.

  If $p^c \in Ports_{\mathsf{H}}$, then $p \in \mathsf{ports}(M_2) \Rightarrow p \in S_2 \subseteq S_1 \subseteq \mathsf{ports}(M_1)$. Then however, p would occur twice in $conf_1$.

- $\mathsf{A}_{\mathsf{u},2}$ and A_1: We have seen in the two previous items that $p \in Ports_{\mathsf{A}_{\mathsf{u},2}} \cap Ports_{\mathsf{A}_1}$ would imply $p^c \in \mathsf{free}(M_2)$ and $p^c \in Ports_{\mathsf{H}}$. However, the ports of M_2 and H are disjoint.

In the universal and blackbox case, the adversary $\mathsf{A}_{\mathsf{u},2}$ is independent of $\mathsf{H_u}$. Thus A_2 is a fixed machine with A_1 as a blackbox submachine. ■

As we have now shown that the original universal and blackbox definitions are both equivalent to their counterparts in the user-only model, which are equal, we have the following corollary:

Corollary 5.3 *The universal model and the blackbox model according to Definitions 5.1 and 5.2 are equivalent if $S_2 \subseteq S_1$ whenever $(M_2, S_2) \in f(M_1, S_1)$.* □

5.5 Dialogues Between Honest Users and Adversary

In Section 2.2 we claimed that splitting rounds into four subrounds is sufficient, even though in reality A and H might engage in a multi-round dialogue within a round of the correct machines. We now show that this is true, at least if $S_2 \subseteq S_1$ whenever $(M_2, S_2) \in f(M_1, S_1)$. Otherwise, a user H of Sys_1 could have a port $p \in (S_2 \setminus S_1)^c$. If H performs multi-round dialogues with the adversary via p in $conf_1$, whereas p is connected to a synchronous correct machine in $conf_2$, it can clearly distinguish the two cases.[35]

We call the model with arbitrary dialogues the dialogue model. Systems are unchanged as always, and dialogue-valid mappings equal user-only-valid mappings. We only allow an arbitrary

[35]Hence specifications in the standard model with $S_2 \not\subseteq S_1$ only make sense if H indeed only uses such ports synchronously. This is justifiable in the example in Section 3.2.2 because the user knows that this is a port to a device from the *system*. In contrast, the user's ports to the adversaries in the cryptographic examples correspond to his talking to other people or communicating with other entities from the application.

but fixed number λ of subrounds, so that we still have a valid clocking scheme according to Definition 2.3: $\kappa_\lambda(1) = M \cup \{\mathsf{H}\}$, and then A is clocked in all even subrounds and H in all odd ones up to λ. (Otherwise we are in a semi-asynchronous model; it would lead too far to define the scheduling for that case here.) Hence the dialogue configurations $\mathsf{Conf_d}(\mathit{Sys})$ are defined as the set of pairs (conf, λ) where conf is a configuration and $\lambda \in \mathbb{N}$, and the runs of conf are defined by the clocking scheme κ_λ.

Dialogue simulatability, $\geq_{\mathsf{d}}^{f}$, is defined like standard simulatability with dialogue configurations with the same λ on both sides.

Theorem 5.4 (Dialogues) *Let systems Sys_1 and Sys_2 and a dialogue-valid mapping f be given. Then $\mathit{Sys}_1 \geq_{\mathsf{d}}^{f} \mathit{Sys}_2 \Leftrightarrow \mathit{Sys}_1 \geq \mathit{Sys}_2$. This is true for perfect, statistical and computational security, and also for universal and blackbox definitions.* □

By Corollary 5.3 this implies that universal and blackbox dialogue simulatability are equivalent.

Proof. We use Theorem 5.3, i.e., that all the variants of the standard model are equivalent to their user-only counterparts for dialogue-valid mappings (which equal user-only-valid mappings). We can treat the perfect, statistical and computational case together.

"$\geq_{\mathsf{d}} \Rightarrow \geq_{\mathsf{u}}$": Let a configuration $\mathit{conf}_1 \in \mathsf{Conf_u}(\mathit{Sys}_1)$ be given. Then $(\mathit{conf}_1, 4) \in \mathsf{Conf_d}(\mathit{Sys}_1)$. Thus there exists a configuration $(\mathit{conf}_2, 4) \in \mathit{Indist}_{\mathsf{d}}^{f}((\mathit{conf}_1, 4))$. This immediately implies $\mathit{conf}_1 \in \mathit{Indist}_{\mathsf{u}}^{f}(\mathit{conf}_1)$. The uniform and blackbox case are proved in the same way.

"$\geq_{\mathsf{u}} \Rightarrow \geq_{\mathsf{d}}$": One can easily see that the part "$\geq_{\mathsf{u}} \Rightarrow \geq$" of the proof of Theorem 5.3 also holds if conf_1 is a dialogue configuration: For machine $\mathsf{H_u}$, we join all subrounds 2 to λ. The clocking scheme κ_C of the intermediate collection C is $\kappa_C(1) = M_2 \cup \{\mathsf{H}\}$; $\kappa_C(2) = \{\mathsf{A}_{\mathsf{u},2}\}$; $\kappa_C(i) = \kappa_\lambda(i-1)$ for $i = 3, \ldots, \lambda+1$; and $\kappa_C(\lambda+2) = \{\mathsf{A}_{\mathsf{u},2}\}$. We join subrounds 3 to $\lambda+1$ to get $\mathit{conf}_{\mathsf{u},2}$; and we join subround 2 with 3 and $\lambda+1$ with $\lambda+2$ to obtain conf_2. ■

Remark 5.5. It makes no difference in the part "$\geq_{\mathsf{u}} \Rightarrow \geq_{\mathsf{d}}$" of this proof whether H also switches in subround $[i.1]$ or not. Hence the clocking scheme $(M, \{\mathsf{A}\}, \{\mathsf{H}\}, \{\mathsf{A}\})$ is also fully general. ∘

5.6 Specified Ports as a Maximum User Interface

In particular in the cryptographic examples, we naturally expect honest users of a structure (M, S) to use precisely the set S of specified ports and to leave the other free ports to the adversary. In the following specified-user-interface model, we consider the slightly more general set $\mathsf{Conf_s}(\mathit{Sys}) \subseteq \mathsf{Conf}(\mathit{Sys})$ of configurations with $\mathsf{ports}(\mathsf{H}) \cap \bar{S}^c = \emptyset$. The conditions on a valid mapping are relaxed: The precondition "$p^c \in \mathsf{free}(M_1)$" of the first condition is replaced by "$p^c \in S_1$".[36]

Theorem 5.5 *Let systems Sys_1 and Sys_2 and a valid mapping f be given. (Thus f is also valid in the specified-user-interface model.) Then $\mathit{Sys}_1 \geq_{\mathsf{s}}^{f} \mathit{Sys}_2 \Leftrightarrow \mathit{Sys}_1 \geq_{\mathsf{s}}^{f} \mathit{Sys}_2$. This is true for perfect, statistical and computational security, and also for the universal and blackbox model.* □

Proof. **"$\geq_{\mathsf{s}} \Rightarrow \geq$":** Let a configuration $\mathit{conf}_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^f(\mathit{Sys}_1)$ be given. We construct a related specified-user-interface configuration $\mathit{conf}_{\mathsf{s},1} = (M_1, S_1, \mathsf{H_s}, \mathsf{A}_{\mathsf{s},1})$ as in Figure 9: $\mathsf{H_s}$ equal H except that any port p of H with $p^c \in \bar{S}_1$ gets a new name p^a. $\mathsf{A}_{\mathsf{s},1}$ equals A_1 except

[36]Recall that reflexivity holds without renaming for this case. Remark 2.3 and the proof of transitivity can also easily be adapted to this case. Hence this extension of valid mappings at the cost of more restricted users is a true alternative to our standard definition.

that it gets additional ports p and p^{ac} for all ports renamed in $\mathsf{H_s}$, and it simply forwards messages between p and port p^{ac}. Clearly $conf_{\mathsf{s},1}$ is still suitable and thus in $\mathsf{Conf}_\mathsf{s}^f(Sys_1)$. There is no significant difference in the executions because $\mathsf{A}_{\mathsf{s},1}$ is clocked in subrounds between $\mathsf{H_s}$ and M_1. Hence $view_{conf_{\mathsf{s},1}}(\mathsf{H_s})$ equals $view_{conf_1}(\mathsf{H})$ except for the renaming.

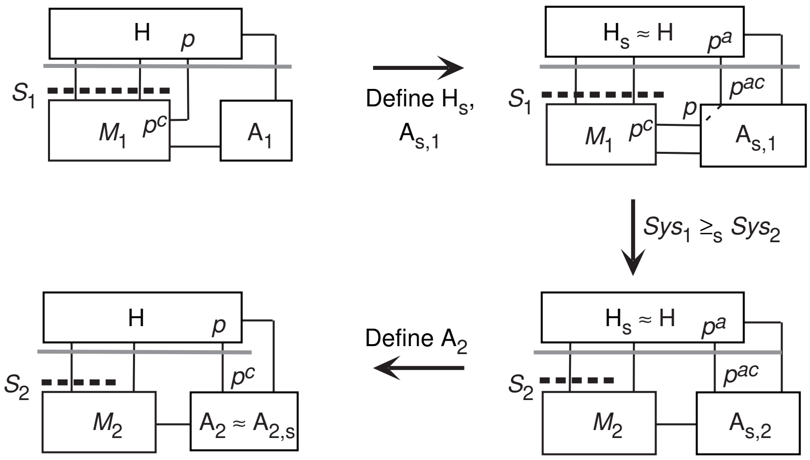

Figure 9: Specified-user-interface simulatability implies standard simulatability

By the precondition there exists $conf_{\mathsf{s},2} = (M_2, S_2, \mathsf{H_s}, \mathsf{A}_{\mathsf{s},2}) \in \mathsf{Conf_s}(Sys_2)$ with $(M_2, S_2) \in f(M_1, S_1)$ and $view_{conf_{\mathsf{s},1}}(\mathsf{H_s}) \approx view_{conf_{\mathsf{s},2}}(\mathsf{H_s})$. As assumed in the figure, $\mathsf{A}_{\mathsf{s},2}$ has a port p^{ac} for any new port p^a of $\mathsf{H_s}$ because the collection is closed and p^{ac} cannot belong to $\mathsf{H_s}$ or M_2 because the name was new.

We want to transform this into a configuration $conf_2 = (M_2, S_2, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(Sys_2)$ with the original H by giving each renamed port p^a of $\mathsf{H_s}$ its old name p. To retain the connection, we also rename p^{ac} of $\mathsf{A}_{\mathsf{s},2}$ into p^c in A_2. We have to make sure that p, p^c do not occur elsewhere in $conf_{\mathsf{s},2}$ (and thus H should be connected differently). By the choice of p, we have $p^c \notin Ports_\mathsf{H}$. As $conf_1$ is suitable, $p \notin \mathsf{ports}(M_2)$ and $p^c \in \mathsf{ports}(M_2) \Rightarrow p^c \in S_2$. The latter and the validity of f would imply $p^c \notin \bar{S}_1$ in contradiction to our choice of p. Hence p and p^c can only occur in $\mathsf{A}_{\mathsf{s},2}$. Then we also give these ports new names in A_2. We have therefore ensured $view_{conf_2}(\mathsf{H})$ $= view_{conf_{\mathsf{s},2}}(\mathsf{H_s})$.

Altogether, this implies $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$.

"$\geq \Rightarrow \geq_\mathsf{s}$": Let $conf_{\mathsf{s},1} = (M_1, S_1, \mathsf{H_s}, \mathsf{A}_{\mathsf{s},1}) \in \mathsf{Conf}_\mathsf{s}^f(Sys_1) \subseteq \mathsf{Conf}^f(Sys_1)$ be given. Hence there exists $conf_2 = (M_2, S_2, \mathsf{H_s}, \mathsf{A}_2)$ with $(M_2, S_2) \in f(M_1, S_1)$ and $view_{conf_{\mathsf{s},1}}(\mathsf{H_s}) \approx view_{conf_2}(\mathsf{H_s})$. Suitability of $conf_{\mathsf{s},1}$ implies $\mathsf{ports}(\mathsf{H_s}) \cap \bar{S}_2^c = \emptyset$, i.e., $conf_2 \in \mathsf{Conf_s}(Sys_2)$. ■

If we have user-only valid mappings, we can restrict ourselves to user-only configurations in the part "$\geq_\mathsf{s} \Rightarrow \geq$" of this proof. Then $\mathsf{H_s}$ uses all ports from S_1. Hence we obtain the following corollary:

Corollary 5.4 *If f is user-only valid, the same theorem also holds for users that use* precisely *the specified ports, i.e., configurations with* $\mathsf{ports}(\mathsf{H})^c \cap \mathsf{free}(M) = S$. □

5.7 Outlook on Model Variants

We have considered several natural model variants and proven that some are completely equivalent to our primary definitions and the others at least in the most typical cases. Further variants are conceivable. For instance, one can let H do its own distinguishing in the computational case, or consider the view of A alone. Or one can omit the honest users and quantify over input sequences. We believe that none of these variants is equivalent to our primary definitions in all cases, although in some cases they are. Given the equivalences shown above, we believe that the primary definitions should stay what they are.

One can also ask what one has really gained by allowing the more exotic cases $S_1 \neq S_2$ and users with ports from $\bar{S}_1$; for this one can prove some canonical equivalences between systems with varying sets S_i. Restrictions to the standard cases would allow some proofs to be shortened.

Another open question is whether standard and universal definitions are really different.

6 Secure Message Transmission in Detail

We now define and prove one reactive cryptographic system in detail, a system for secure message transmission. The purpose of this example is to show

- why we model tolerable imperfections, and thus allow ideal systems with different trusted hosts containing specific capabilities for adversaries,
- the timing problems that occur in a synchronous system and why we think they should not always be abstracted from,
- and that some system is actually provable with respect to our definition.

By secure message transmission, we mean the sending and receiving of private and authentic messages, as usually achieved by encryption and authentication.

6.1 Tolerable Imperfections and Timing Problems

Traffic Patterns and Traffic Suppression The most intuitive version of an ideal system for secure message transmission would be a trusted host that simply relays messages. E.g., [Gold 99], Page 8, says that an encryption scheme is considered secure if it simulates an ideal private channel between the parties, and that this means that an eavesdropping adversary gains nothing over a user which does not tap this channel. This is indeed the most desirable service. However, it implies that an adversary does not even see whether a message is sent or not. Thus any real system as secure as this trusted host must also hide this fact, e.g., by sending meaningless ciphertexts at all other times (see [Bara 64] for an early note of this and [Abad 98] for one in a similar context as here). This may indeed be the method of choice in a few applications, but it costs a lot of bandwidth. Usually one is therefore satisfied with encrypting the real messages only. Nevertheless, one may want to have a precise definition of what one has achieved in the form of the simulation of a trusted host. For this, we introduce an abstraction of the tolerable imperfection into the trusted host. For encryption, the trusted host will tell the adversaries the traffic pattern in the form of one "busy" bit per pair of honest participants and round.

Similarly, as cryptography cannot prevent an adversary from destroying a message in transit, the trusted host will allow the adversary to input a bit "suppress" for any message that was signaled to him by "busy". Again, in some applications one will want a more ideal service, and thus need a lot of replication in the underlying protocol, but typically one is satisfied with the much cheaper service, and hence this should also be defined and proved. Furthermore, for availability one would need stronger assumptions on the number of corrupted machines than for privacy and integrity, where only the two parties concerned must be honest.

Observable traffic patterns also occur in many other systems, e.g., all typical key exchange and authentication protocols or payment systems. Our approach enables us to separate vulnerabilities of insecure implementations (such as cryptographic problems or protocol vulnerabilities allowing replay attacks etc.) from such unavoidable imperfections of the service given certain resource limitations.

Timing Imperfections While, once noticed, the traffic analysis imperfections may seem clear, it is less obvious that even after these changes to the initially desired trusted host, no real implementation will be as secure as it. (This would also be true if we had not allowed the imperfections discussed so far.) The problem is that adversaries can react several rounds faster than honest participants.[37] We first present it in the synchronous model and then discuss why it is not only a technical difficulty of this model.

If an honest user sends a message using the real encryption system, the message goes via the sender's machine and the recipient's machine. The corresponding timing must be modeled in the trusted host, i.e., the trusted host delivers messages after two switching steps. Thus, with the trusted host, an honest user cannot get an answer referring to his message earlier than four rounds after his message. However, if the recipient in the real system is corrupted, he can save the two rounds where his own machine would handle the message by taking it immediately from the line, decrypting it (he knows his own decryption key), and composing and encrypting the answer, all in one round. Thus an answer from a dishonest recipient can arrive two rounds faster. We see no realistic way to improve the real system so that it corresponds to the same trusted host for honest and dishonest recipients in this respect. We therefore model this tolerable imperfection in the trusted host by offering a faster service at the ports for the adversary, so that anyone basing an application on the trusted host must be aware of it.

It is reasonable to ask whether this problem would disappear in asynchronous models and whether those aren't more realistic. (Or one can relax the timing requirements in the comparison, i.e., not require that events in the ideal and the real system must happen in the same rounds, as in [HiMa 00].) First, however, many cryptographic protocols are designed for synchronous systems and it should be possible to define their security.[38] Secondly, in real life the problem occurs whenever the users (not the machines of the system!) have access to real time. This seems unavoidable because we cannot define that cryptographic systems must never be used in real-time applications or that human users must not look at their watches.[39] Thus, in a real system an adversary may indeed be faster (e.g., by bridging network delays), and honest users can notice this. This will usually not have a very bad effect, but contradicts real-life indistinguishability of the real and ideal system. Hence we believe that the timing differences must be modeled in the trusted host. One can even construct artificial examples where users tell secrets to someone who can react very fast, e.g., because they believe that the person must have known the secrets before; such a problem might occur in a badly designed synchronous protocol for mutual identification.

Another problem is timing channels in the classical sense (see Section 1). For example, consider a user who is known to answer all good news fast and all bad news slowly. An adversary in the real system can, simply by the timing of the traffic pattern, judge whether this user gets good or bad news, but this imperfection is not visible in an asynchronous model. Concrete examples of this type might be automatic message-processing systems that answer certain types of requests faster than others.

Even more classical timing channels are those where the users are considered dishonest, e.g., a Trojan horse that tries to send secret information out from an infected program to its creator, but without having subverted the operating system. Our model, with a universal quantifier over honest users H, also includes those that are actually two machines H_1 and H_2 without connection, and requires that they cannot communicate better using the real system than in the ideal system.

[37] This is different from the problem of rushing adversaries within a round, which is already taken care of in the switching model.

[38] Of course, the synchrony is typically virtual, i.e., derived from loosely synchronized clocks and bounds on message delays after which a message will be considered lost.

[39] The rounds in a cryptographic protocol used over the Internet may have to be quite long to tolerate temporary congestion or message retransmission. Hence they may indeed be observable with watches.

6.2 An Ideal System

We now define an ideal system that models secure message transmission with three tolerable imperfections: observable traffic patterns, suppressible messages, and faster service for the adversary. As discussed in the previous subsection, this is the optimum that can be achieved without a large increase in the network traffic.

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

Scheme 6.1 (Ideal System for Secure Message Transmission) *Let a polynomial-time decidable set* $Msg \subseteq \{0,1\}^{\leq len}$ *of messages of length at most len be given with* $\epsilon \in Msg$*, where* ϵ *is the empty word and stands for "no message". Let a number* $n \in \mathbb{N}$ *of intended participants be given and* $\mathcal{M} := \{1, \ldots, n\}$*, and let* $R \in \mathbb{N}$ *be the intended number of rounds. An ideal system for secure message transmission is then defined as*

$$Sys^{\mathsf{secmsg,id}}_{Msg,n,R} = \{(\{\mathsf{TH}(\mathcal{H})\}, S(\mathcal{H}))|\mathcal{H} \subseteq \mathcal{M}\},$$

(i.e., the access structure is the powerset of $\mathcal{M}$*), where* $\mathsf{TH}(\mathcal{H})$ *and* $S(\mathcal{H})$ *are defined as follows. Let* $\mathcal{A} = \mathcal{M} \setminus \mathcal{H}$ *be the set of corrupted participant indices. We often write "user u" or "participant u" although u is only the index.*

Ports: *The specified ports of* $\mathsf{TH}(\mathcal{H})$ *are* $S(\mathcal{H}) := \{\mathsf{in}_u?, \mathsf{out}_u!|u \in \mathcal{H}\}$ *and the unspecified ports* $\bar{S}_2(\mathcal{H}) = \{\mathsf{adv\_out!}, \mathsf{busy!}, \mathsf{adv\_in?}, \mathsf{suppress?}\}$*; thus* $Ports_{\mathsf{TH}(\mathcal{H})} = S(\mathcal{H}) \cup \bar{S}_2(\mathcal{H})$ *(see Figure 10).*

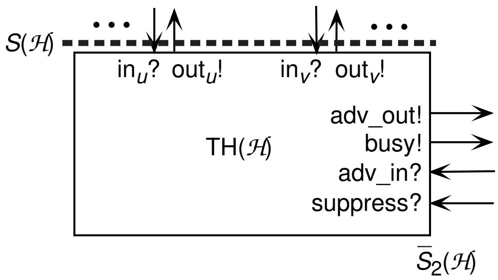

Figure 10: A structure of the ideal system for secure message transmission

The internal state of $\mathsf{TH}(\mathcal{H})$ *consists of the current round number i, which it updates with every transition, and a matrix* $in_{i-1} \in Msg^{\mathcal{M}^2}$*. In Round 0,* $\mathsf{TH}(\mathcal{H})$ *does nothing. (This is time reserved for initialization in the real system.) Now we consider the state transition for any Round* $i > 0$*.*

Inputs: *At each port* $\mathsf{in}_s?$*,* $\mathsf{TH}(\mathcal{H})$ *expects an input vector* $(in_{i,s,r})_{r\in\mathcal{M}} \in Msg^{\mathcal{M}}$*. By "expects" we mean that it replaces any other input by a vector of all* ϵ*; similarly for other structures in the following. (Hence every user s may send one message to every other user r in each round.) At the ports* $\mathsf{adv\_in}?$ *and* $\mathsf{suppress}?$*, it expects matrices* $adv\_in_i \in Msg^{\mathcal{A}\times\mathcal{H}}$ *and* $suppress_i \in \{0,1\}^{\mathcal{H}^2}$*.*

Computation: *For all* $s, r \in \mathcal{H}$*, the trusted host sets*

$$out_{i,s,r} \quad := \quad \begin{cases} \epsilon & \text{if } suppress_{i,s,r} = 1 \\ in_{i-1,s,r} & \text{else;} \end{cases} \tag{1}$$

$$busy_{i,s,r} \quad := \quad \begin{cases} 1 & \text{if } in_{i,s,r} \neq \epsilon \\ 0 & \text{else.} \end{cases} \tag{2}$$

Hence it delays messages between honest users for one round, but immediately tells the adversary where such messages are in transit. For $a \in \mathcal{A}$, $u \in \mathcal{H}$, it sets

$$\begin{aligned} out_{i,a,u} &:= adv\_in_{i,a,u}; && (3) \\ adv\_out_{i,u,a} &:= in_{i,u,a}. && (4) \end{aligned}$$

Hence messages to and from the adversary are delivered immediately.

Outputs: *The matrices $adv\_out_i \in Msg^{\mathcal{H}\times\mathcal{A}}$ and $busy_i \in \{0,1\}^{\mathcal{H}^2}$ are output at the ports* $\mathsf{adv\_out!}$ *and* busy, *and each tuple $(out_{i,s,r})_{s\in\mathcal{M}}$ at the port* $\mathsf{out}_r!$.

◇

Fixing a finite message space is needed because otherwise the length of a message cannot be secret. The bound on the number of rounds is not essential, but simplifies the representation and is no serious restriction. The input and output vectors at the specified ports, i.e., for honest users, could be implemented with a (fixed) compressed representation because they will usually be sparse. The representation of the adversary in- and outputs is irrelevant because they only occur in the ideal system.

6.3 A Real System

We now define a real system for secure message transmission. The main part, composition and decomposition of network messages, is Equations 5 and 6.

We use asymmetric encryption and digital signatures. Let Msg be the message space, R the number of rounds, and $\mathcal{M} = \{1, \ldots, n\}$ as in Scheme 6.1.

The algorithms $(\mathsf{gen_S}, \mathsf{sign}, \mathsf{test})$ denote a secure digital signature scheme [DiHe 76, GoMR 88] whose message space includes $Msgs_S := Msg \times \mathcal{M} \times \{1, \ldots, R\}$ (for all security parameters).[40] We use slightly abbreviated notation: We write $(\mathsf{sign}_u, \mathsf{test}_u) \leftarrow \mathsf{gen_S}(1^k)$ for the generation of a signing key and a test key based on a security parameter k. By $sig \leftarrow \mathsf{sign}_u(m)$, we denote a signature on the message $m \in Msgs_S$, including m itself.[41] We denote the resulting signature space by $Sigs_S(k)$. The length of the signatures is polynomially bounded in k. The verification $\mathsf{test}_u(sig)$ returns m if the signature is valid with respect to the included message, else false.

By $(\mathsf{gen_E}, \mathsf{E}, \mathsf{D})$, we denote an encryption scheme secure against adaptive chosen-ciphertext attacks, e.g., [CrSh1 98]. We write $(\mathsf{E}_u, \mathsf{D}_u) \leftarrow \mathsf{gen_E}(1^{k^*(k)})$ for the generation of an encryption key and a decryption key. Here, $k^*(k)$ denotes a security parameter that allows us to securely encrypt the message space $\mathcal{M} \times Sigs_S(k)$ for the given k.[42] We denote the (probabilistic) encryption of a message m by $c \leftarrow \mathsf{E}_u(m)$, and decryption by $m \leftarrow \mathsf{D}_u(c)$.

The comma denotes tuple composition, not concatenation, and its implementation must guarantee unambiguous decomposition.

Scheme 6.2 (Real System for Secure Message Transmission) *Let a message set Msg, a set $\mathcal{M} = \{1, \ldots, n\}$ of participant indices, and a round number R be given as in Scheme 6.1. The system is a standard cryptographic system according to Definitions 3.1 and 3.2. Hence we only have to define the intended structure (M^*, S^*) and the trust model, but for readability of*

[40] We repeat the security definitions in more detail in Section 6.5. If the original scheme has a too small message space, a combination with a collision-free hash functions retains security [Damg 88].

[41] We even assume that it is a pair of the message and the actual signature which can be uniquely decomposed. Then sig uniquely determines m independently of the key. Otherwise more complicated message formats would be needed below, which is a waste of bandwidth in the standard case.

[42] We do not require all input messages to be of the same length. However, security for this message space implies that also the message length is hidden by the encryption.

the proof, we also describe the resulting actual structures. M^* is a set $\{\mathsf{M}_u | u \in \mathcal{M}\}$ and the access structure is the powerset of $\mathcal{M}$. Hence the system is of the form

$$Sys^{\mathsf{secmsg,real}}_{Msg,n,R} = \{(M(\mathcal{H}), S(\mathcal{H})) | \mathcal{H} \subseteq \mathcal{M}\}$$

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

Ports: *The ports of machine 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 M_v, and the last ones for key exchange, only used in Round 0.*

The specified ports are $S^* := \{\mathsf{in}_u?, \mathsf{out}_u! | u \in \mathcal{M}\}$.

Channel model χ: *Connections $\{\mathsf{net}_{u,v}!, \mathsf{net}_{u,v}?\}$ are insecure and connections $\{\mathsf{aut}_{u,v}!, \mathsf{aut}_{u,v}?\}$ authentic. The resulting ports in a real structure are illustrated in Figure 11.*

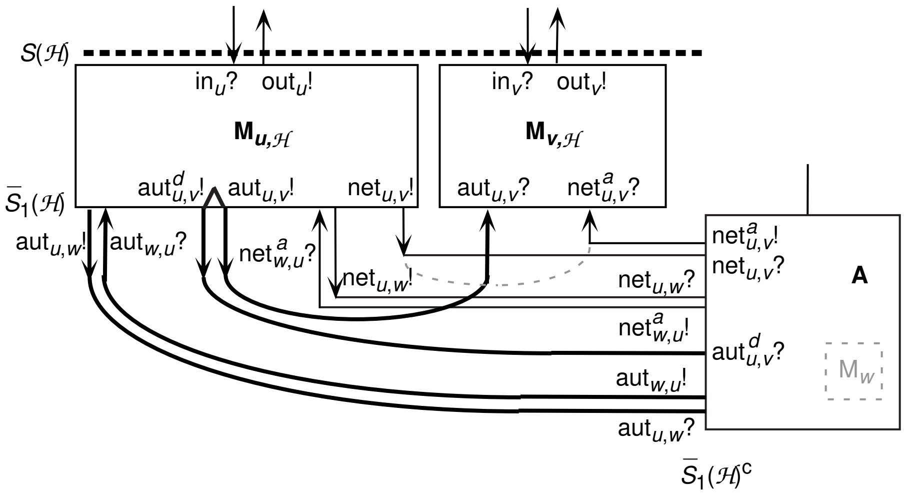

Figure 11: Classes of ports in the real system for secure message transmission. Two correct machines are shown and what became of the intended connections from $\mathsf{M}_{u,\mathcal{H}}$ to $\mathsf{M}_{v,\mathcal{H}}$ (port names in the machines) and between $\mathsf{M}_{u,\mathcal{H}}$ and a machine M_w with $w \notin \mathcal{H}$ (port names outside). Authentic connections are bold, others normal, and the original of a modifiable connection is dashed.

The state-transition function of a machine $\mathsf{M}_{u,\mathcal{H}}$ is defined as follows:

Round 0 (Initialization) *It generates two key pairs, $(\mathsf{sign}_u, \mathsf{test}_u) \leftarrow \mathsf{gen}_\mathsf{S}(1^k)$ and $(\mathsf{E}_u, \mathsf{D}_u) \leftarrow \mathsf{gen}_\mathsf{E}(1^{k^*(k)})$, and outputs $(\mathsf{test}_u, \mathsf{E}_u)$ at $\mathsf{aut}_{u,v}!$ and $\mathsf{aut}^d_{u,v}!$ for all $v \in \mathcal{M}$.*

Now we consider $\mathsf{M}_{u,\mathcal{H}}$ in an arbitrary round $i > 0$.

Inputs: *It expects an input vector $(in_{i,u,r})_{r\in\mathcal{M}} \in Msg^{\mathcal{M}}$ at $\mathsf{in}_u?$. If $i = 1$, it also expects a pair $(\mathsf{test}_v, \mathsf{E}_v)$ (of public keys) at each port $\mathsf{aut}_{v,u}?$. If $i > 1$, it reads a value $net^a_{i-1,s,u}$ from each port $\mathsf{net}^a_{s,u}?$.*

Computation: *It composes a* network message *for each recipient $r \in \mathcal{M}$ as*

$$net_{i,u,r} \leftarrow \begin{cases} \mathsf{E}_r(u, \mathsf{sign}_u(in_{i,u,r}, i, r)) & \text{if } in_{i,u,r} \neq \epsilon, \\ \epsilon & \text{else.} \end{cases} \tag{5}$$

We abbreviate this function by $net_{i,u,r} \leftarrow \mathsf{comp}(i, u, r, in_{i,u,r})$.

If $i > 1$, it decomposes each received message $net^a_{i-1,s,u}$ as

$$(s_{i-1,s,u}, sig_{i-1,s,u}) \leftarrow \mathsf{D}_u(net^a_{i-1,s,u}).$$

If the decryption or decomposition fails, let both components be ϵ. Then

$$out_{i,s,u} := \begin{cases} m & \text{if } s_{i-1,s,u} = s \wedge \mathsf{test}_s(sig_{i-1,s,u}) = (m, i-1, u) \\ \epsilon & \text{else.} \end{cases} \tag{6}$$

We abbreviate this function by $out_{i,s,u} \leftarrow \mathsf{decomp}(i, s, u, net^a_{i-1,s,u})$.

Outputs: *It outputs the vector $(out_{i,s,u})_{s\in\mathcal{M}}$ at $\mathsf{out}_u!$, and each value $net_{i,u,r}$ at $\mathsf{net}_{u,r}!$.*

◇

6.4 Remarks on the Message Format

The format of the network message may look complicated. However, simpler formats are not always secure. In particular, omitting the identity u in the encryption, i.e.,

$$net_{i,u,r} \leftarrow \mathsf{E}_r(\mathsf{sign}_u(in_{i,u,r}, i, r))$$

is insecure although one may feel that the signature inside determines the identity: An adversary can choose one of his own keys test_a equal to one of a correct machine, say test_s, because M_s switches in Round [0.1] and A in Round [0.2]. Later it can take a network message $net_{i,s,r}$ sent between two honest participants and use it as its own network message $net^a_{i,a,r}$ (again because A switches after M_s in Round i). This message passes the test, and thus $in_{i,s,r}$ is output to H as $out_{i+1,a,r}$. An adversary in the ideal system cannot achieve this effect. It is even dangerous in practice, because H, believing that it obtained this message from M_a, might freely send parts of it back to M_a in a reply.

This attack on the simplified network messages could be avoided by verifying that all public keys are different. However, this would not imply provable security given the normal definition of a signature system (see Definition 6.1): It is not excluded that an adversary can choose a key related to the key of a correct machine such that signatures made with sign_s are also acceptable with respect to test_a.

First encrypting and then signing does not automatically work either, e.g.,

$$net_{i,u,r} \leftarrow \mathsf{sign}_u(i, r, \mathsf{E}_r(in_{i,u,r}))$$

has the same problem even more obviously: The adversary can take the ciphertext $c = \mathsf{E}_r(in_{i,u,r})$ from such a message and also send it in a message of his own as $net^a_{j,a,r} \leftarrow \mathsf{sign}_a(j, r, c)$.

The inclusion of the round number i is necessary for freshness. An alternative would be nonces, but this requires a multi-round protocol for each message.

6.5 Security Proof

We now show that the real system is as secure as the ideal system. Our simulation is blackbox, see Definition 5.2. We use Corollary 5.4, i.e., we only consider users that use precisely the specified ports. Hence the set P of ports of the real adversary always comprises $\bar{S}_1(\mathcal{H})^c$.

Scheme 6.3 (Simulator) *Let a set $\mathcal{H} \subseteq \mathcal{M}$ of (indices of) correct participants be given and $\mathcal{A} := \mathcal{M} \setminus \mathcal{H}$. Let $P = \bar{S}_1(\mathcal{H})^c \cup P'$ be a set of adversary ports.*

The definition of the simulator $\mathsf{Sim}_{\mathcal{H},P}(\mathsf{A})$ is illustrated in Figure 12. If $\mathcal{H}$ and P are clear from the context, we write $\mathsf{Sim}(\mathsf{A})$. Its ports must be as in Figure 10, here $Ports_{\mathsf{Sim(A)}}$ =

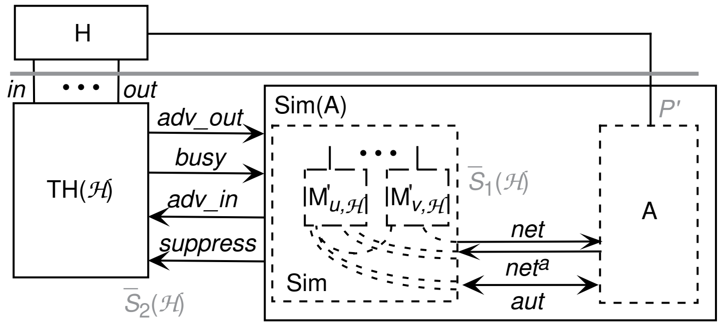

Figure 12: Set-up of the simulation. The names on the arrows are names of message classes (slightly summarizing the port names).

$\bar{S}_2(\mathcal{H})^c \cup P'$. *It leaves the communication between* A *and* H *unchanged. In slight abuse of notation we therefore say* Sim *for the part of* Sim(A) *without* A*; then* $Ports_{\mathsf{Sim}} = \bar{S}_1(\mathcal{H}) \cup \bar{S}_2(\mathcal{H})^c$ *corresponding to Figure 11.*

Internally, Sim *more or less simulates the real machines; we illustrate this by* $\mathsf{M}'_{u,\mathcal{H}}$ *for* $u \in \mathcal{H}$*. The timing of the main message sequence and its simulation is illustrated in Figures 13 and 14. We define the transitions of* Sim(A) *as the compressed version of a 6-subround clocking scheme* $\kappa_6 = (\{\mathsf{TH}(\mathcal{H}), \mathsf{H}\}, \mathsf{Sim}, \mathsf{A}, \mathsf{H}, \mathsf{A}, \mathsf{Sim})$ *where Subrounds 2 with 3 and 5 with 6 are joined to get the correct clocking for the real system (see Lemma 4.1d). We call the joined subrounds 2a and 2b, and 4a and 4b.*

Essentially, in Subround $[i.2a]$*,* Sim *transforms the abstract messages it got from* $\mathsf{TH}(\mathcal{H})$ *into suitable network messages for* A*, and in Subround* $[i.4b]$ *it transforms the messages from* A *into corresponding signals to* $\mathsf{TH}(\mathcal{H})$*.*

Round $[0.2a]$ Sim *simulates the key generation of the machines* $\mathsf{M}_{u,\mathcal{H}}$ *for* $u \in \mathcal{H}$ *without changes.*

Round $[0.4b]$ Sim *stores received keys as in Round* $[1.1]$ *of the real system.*

Round $[i.2a]$ **for** $i > 0$**:** Sim *obtains matrices* adv_out_i *and* $busy_i$ *from* $\mathsf{TH}(\mathcal{H})$*.*

Each element $adv\_out_{i,s,r}$ *represents a message from a correct sender* $s \in \mathcal{H}$ *for a corrupted recipient* $r \in \mathcal{A}$*. Here* Sim *computes the corresponding network message as* $net_{i,s,r} \leftarrow \mathsf{comp}(i, s, r, adv\_out_{i,s,r})$*.*

Each element $busy_{i,s,r} = 1$ *signals a message between correct machines* $s, r \in \mathcal{H}$*. Here* Sim *computes* $net_{i,s,r} \leftarrow \mathsf{comp}(i, s, r, m_{sim})$ *with a fixed message* $m_{sim} \in Msg \setminus \{\epsilon\}$*. (It cannot obtain the actual input made at the port* $\mathsf{in}_s?$ *because* $\mathsf{TH}(\mathcal{H})$ *keeps it secret.)*

Round $[i.4b]$ **for** $i > 0$**:** *Now* Sim *converts the messages* $net^a_{i,s,r}$ *received from* A *(in Rounds* $[i.2b]$ *and* $[i.4a]$*) into inputs to* $\mathsf{TH}(\mathcal{H})$*:*

For $s \in \mathcal{A}$ *and* $r \in \mathcal{H}$*, it sets* $adv\_in_{i+1,s,r} := \mathsf{decomp}(i+1, s, r, net^a_{i,s,r})$*.*

For $s, r \in \mathcal{H}$ *and if* $busy_{i,s,r} = 1$*, the trusted host expects to receive* $suppress_{i+1,s,r}$ *indicating whether the message is destroyed in transit. Therefore* Sim *sets* $m := \mathsf{decomp}(i+1, s, r, net^a_{i,s,r})$*. If* $m = \epsilon$*, it sets* $suppress_{i+1,s,r} := 1$*, otherwise* $suppress_{i+1,s,r} := 0$*. If* $m \neq m_{sim} \wedge m \neq \epsilon$*, or* $m \neq \epsilon \wedge busy_{i,s,r} = 0$*, the simulator stops.*[43]

[43]In the real system, m would be output at $\mathsf{out}_r!$, but the simulator has no way of making $\mathsf{TH}(\mathcal{H})$ do this; this is where message integrity is expressed in $\mathsf{TH}(\mathcal{H})$. Hence the simulation would become distinguishable and can as well be stopped. The proof will show that this case only occurs if A has forged a signature, and thus it is negligible.

◇

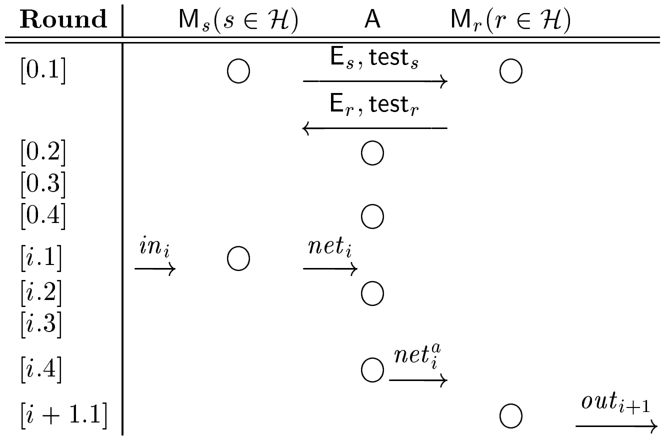

Figure 13: Timing of Scheme 6.2 for key exchange and a message between two correct machines. ◯ denotes relevant switching of the machine in this column; H switches in [i.1] and [i.3].

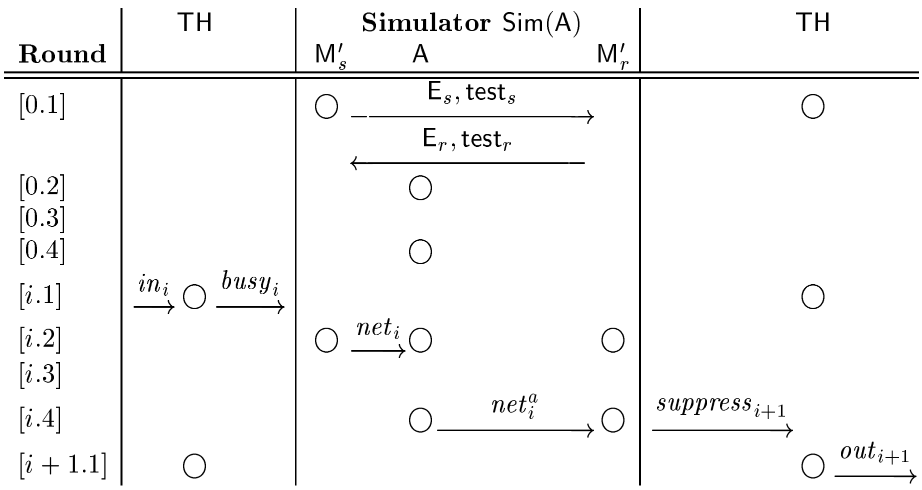

Figure 14: Timing of Scheme 6.3 for the same situation as in Figure 13.

For the following security proof, we need the underlying definitions of secure encryption and signature schemes. Security of a signature scheme means that existential forgery is infeasible even in adaptive chosen-message attacks [GoMR 88].

Definition 6.1 (Security of Signature Schemes) *An arbitrary (probabilistic) polynomial-time machine* $\mathsf{A_{sig}}$ *interacts with a signer machine* Sig *(also called signing oracle) defined as follows:*

1. Sig *generates a key pair,* $(\mathsf{sign}, \mathsf{test}) \leftarrow \mathsf{gen_S}(1^k)$*, and sends* test *to* $\mathsf{A_{sig}}$.
2. *In each round,* Sig *signs an arbitrary message* m_j *it receives from* $\mathsf{A_{sig}}$ *(automatically only a polynomial number if* $\mathsf{A_{sig}}$ *is polynomial).*
3. *Finally,* $\mathsf{A_{sig}}$ *should output a value sig.*

$\mathsf{A_{sig}}$ *has won if* $\mathsf{test}(sig)$ *gives a message* m *with* $m \neq m_j$ *for all* j*, i.e., sig is a valid signature on a message that* Sig *did not sign. The probability of this event must be negligible in* k*. (In* *our terminology, we have a closed collection of 2 machines clocked alternately, and the event is a predicate on the runs; hence the probability is well-defined.)* ◇

Security of an encryption scheme means that any two messages must be indistinguishable even in adaptive chosen-ciphertext attacks. This was introduced in [RaSi 92], used for an efficient construction in [CrSh1 98] and formalized as "IND-CCA2" in [BDPR 98].

Definition 6.2 (Security of Encryption Schemes) *An arbitrary (probabilistic) polynomial-time machine* $\mathsf{A_{enc}}$ *interacts with a decryptor machine* Dec *(also called decryption oracle) defined as follows:*

1. Dec *generates a key pair,* $(\mathsf{E}, \mathsf{D}) \leftarrow \mathsf{gen_E}(1^k)$*, and sends* E *to* $\mathsf{A_{enc}}$*.*
2. *In each round,* Dec *decrypts an arbitrary ciphertext* c_j *received from* $\mathsf{A_{enc}}$*.*
3. *At some point,* $\mathsf{A_{enc}}$ *sends a pair* (m_0, m_1) *of messages to* Dec*. Then* Dec *randomly chooses a bit* b *and returns an encryption* $c \leftarrow \mathsf{E}(m_b)$*.*
4. $\mathsf{A_{enc}}$ *may again ask* Dec *to decrypt ciphertexts* c_j*, but now* Dec *refuses to decrypt if* c_j *equals its own ciphertext* c*.*
5. *Finally,* $\mathsf{A_{enc}}$ *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* $1/2 + 1/\mathsf{poly}(k)$*. (Again this is a 2-machine collection and thus the probability is well-defined.)* ◇

Theorem 6.1 (Security of the Message Transmission System) *For any message set* Msg*, index set* $\mathcal{M} = \{1, \ldots, n\}$ *and round number* R *as in Scheme 6.1,*

$$Sys^{\mathsf{secmsg,real}}_{Msg,n,R} \geq^{f,\mathsf{poly}}_{\mathsf{sec}} Sys^{\mathsf{secmsg,id}}_{Msg,n,R}$$

for the canonical mapping f *from Section 3.1 if the signature and encryption schemes used in Scheme 6.2 are secure.* □

Recall that the canonical mapping f maps each real structure $(M(\mathcal{H}), S(\mathcal{H}))$ to the ideal structure $(\{\mathsf{TH}(\mathcal{H})\}, S(\mathcal{H}))$ for the same set $\mathcal{H}$.

Proof. Let a set $\mathcal{H}$ and thus a structure $(M(\mathcal{H}), S(\mathcal{H}))$ and a port set $P = \bar{S}_1(\mathcal{H})^c \cup P'$ be given. In the following, we omit the parameter $\mathcal{H}$ of all machines and write $\mathsf{Sim(A)}$ for $\mathsf{Sim}_P(\mathsf{A})$. Now let a configuration $conf_1 = (M, S, \mathsf{H}, \mathsf{A}) \in \mathsf{Conf}^f(Sys^{\mathsf{secmsg,real}}_{Msg,n,R})$ with $Ports_{\mathsf{A}} = P$ be given. By Remark 5.4, we can assume that H is not clocked in Subrounds $[i.1]$. We claim that $conf_2 = (\{\mathsf{TH}\}, S, \mathsf{H}, \mathsf{Sim(A)}) \in Indist^f(conf_1)$, i.e., $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$.

We show the stronger statement $view_{conf_1}(\mathsf{H}, \mathsf{A}) \approx view_{conf_2}(\mathsf{H}, \mathsf{A})$. Intuitively this means that we compare M and the combination of TH and Sim. For this, we first have to show that the clocking makes no difference. Then, there are intuitively two aspects in the simulation that make it only computationally indistinguishable: a message m_{sim} is often encrypted instead of a real message, and a simulation may stop prematurely due to what should be a successful signature forgery by A. We therefore show by two reduction proofs that we could break one of the underlying schemes if the views were distinguishable.

Clocking: We defined $\mathsf{Sim(A)}$ in $conf_2$ as a combination based on a collection where Sim and A are separate (we now call it $conf_2^6$) and a 6-subround clocking scheme κ_6. Hence $view_{conf_2}(\mathsf{H}, \mathsf{A})$ equals $view_{conf_2^6}(\mathsf{H}, \mathsf{A})$ except for subround renaming. In $conf_2^6$, only Sim and TH are clocked in Subrounds 4b, 1, and 2a. Let $\mathsf{TH}+\mathsf{Sim}$ denote the hiding combination of TH and Sim according to Lemma 4.1 and where these subrounds are joined. Then $conf_2^* = (\mathsf{TH} + \mathsf{Sim}, S, \mathsf{H}, \mathsf{A})$ is a configuration with the standard clocking scheme, and $view_{conf_2^*}(\mathsf{H}, \mathsf{A})$ equals $view_{conf_2^6}(\mathsf{H}, \mathsf{A})$ except for the subround renaming. In both renamings of $conf_2^6$, H ends up in Subround 3 and A in 2 and 4. Hence

$$view_{conf_2^*}(\mathsf{H}, \mathsf{A}) = view_{conf_2}(\mathsf{H}, \mathsf{A}).$$

Signatures: We first show that the probability that the simulation stops prematurely is negligible. Assume the contrary. We then construct an adversary $\mathsf{A_{sig}}$ against the signature scheme using H and A as blackboxes (recall Definition 6.1) as shown in Figure 15. $\mathsf{A_{sig}}$ randomly chooses $s^* \in \mathcal{H}$ and starts simulating TH + Sim interacting with H and A. It does everything as TH + Sim would except that it uses the public key test from Sig as test_{s^*}. Thus whenever it has to execute $sig_j \leftarrow \mathsf{sign}_{s^*}(m_j)$ for some message m_j, it sends m_j to Sig instead and uses the answer as sig_j.[44] If the simulation stops prematurely in a subround $[i + 1.1]$, this corresponds to Subround $[i.4b]$ in the original clocking of Sim, and A has sent a network message $net^a_{i,s,r}$ for some $s, r \in \mathcal{H}$ that is correct according to Equation (6) but contains a message $m' \neq m_{sim}$, or any $m' \neq \epsilon$ although $busy_{i,s,r} = 0$. Then if $s = s^*$, the machine $\mathsf{A_{sig}}$ outputs the second component sig of $\mathsf{D}_r(net^a_{i,s,r})$.

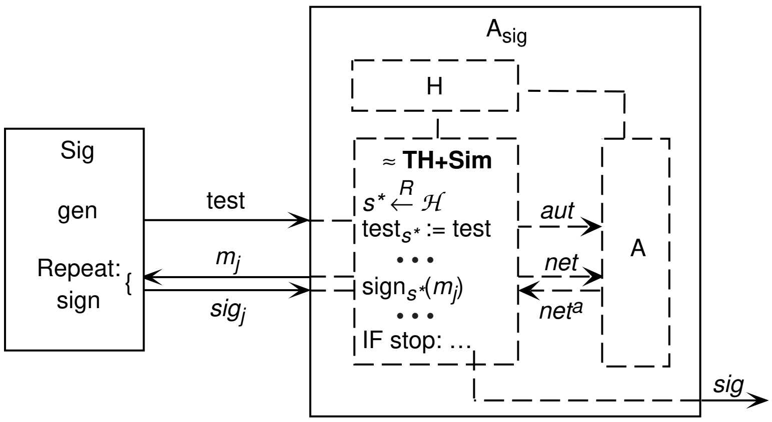

Figure 15: Reduction to an adversary on the signature scheme

We first show that if $\mathsf{A_{sig}}$ makes any output, it is a successful forgery. By Equation (6), $\mathsf{test}_{s^*}(sig) = (m', i, r) =: m^*$. We have to show that $m^* \neq m_j$ for all messages m_j that $\mathsf{A_{sig}}$ asked Sig to sign. Assume the contrary. By construction, messages m_j only arise when Sim applies Equation (5) to construct a message $net_{i',s^*,r'}$ in an original subround $[i'.2a]$, now $[i'.1]$, and then $m_j = (in', i', r')$ for some in'. By the required unambiguous decomposition of the representation of message triples, $m^* = m_j$ implies $in' = m' \wedge i' = i \wedge r' = r$. As $r \in \mathcal{H}$, the construction of Sim implies $m' = m_{sim}$, and actually m_j is only signed if $busy_{i,s^*,r} = 1$. This contradicts the preconditions on m' (as a message leading to premature stopping).

Secondly, we show that the probability that $\mathsf{A_{sig}}$ makes an output is not negligible. $\mathsf{A_{sig}}$ combined with Sig behaves exactly like TH + Sim until the stopping time. Hence the probability of stopping in the simulation equals that in $conf_2^*$. If stopping occurs, the stopping conditions are fulfilled for one $s \in \mathcal{H}$. As $\mathsf{A_{sig}}$ chose s^* randomly and no information about s^* is visible to H and A, the probability that $s^* = s$ is $|\mathcal{H}|^{-1}$.[45] Hence the success probability of $\mathsf{A_{sig}}$ is a fixed fraction $|\mathcal{H}|^{-1}$ of the probability that Sim stops prematurely, and therefore also not negligible. This is the desired contradiction with the security of the signature scheme.

[44]There is no clocking problem although we defined that Sig signs only one message per round because $\mathsf{A_{sig}}$ can clock its submachines itself, i.e., the rounds indexed j and those indexed i are different.

[45]More formally, one transforms the given probability space, where s^* is chosen at the start, such that s^* is chosen independently at the end, exploiting that the rest of the runs is compatible with any s^*.

As a consequence, we can modify the behaviour of $\mathsf{TH} + \mathsf{Sim}$ in the case that Sim stops prematurely: We define a machine $\mathsf{TH}' + \mathsf{Sim}'$ that, instead of stopping under the given conditions, outputs the forged message m' as $out_{i+1,s,r}$ (as the real machine M_r would do in $conf_1$). Let $conf'_2 = (\mathsf{TH}' + \mathsf{Sim}', S, \mathsf{H}, \mathsf{A})$. As a difference only occur with negligible probability, we have shown that[46]

$$view_{conf'_2}(\mathsf{A}, \mathsf{H}) \approx view_{conf^*_2}(\mathsf{A}, \mathsf{H}).$$

As a technicality for the following proof, we need a similar statement for the correct machines. Let M' be a machine that acts like the hiding combination of M with one exception: If $\mathsf{decomp}(i, s, r, net^a_{i-1,s,r})$ yields $m = m_{sim} \neq in_{i-1,s,r}$, then M' sets $out_{i,s,r} = in_{i-1,s,r}$ instead of m. The proof is a reduction just as above: We have a forged signature on $(m_{sim}, i-1, r)$ because only $(in_{i-1,s,r}, i-1, r)$ was signed with the components $i-1$ and r. Let $conf'_1 = (\mathsf{M}', S, \mathsf{H}, \mathsf{A})$. Hence we have

$$view_{conf'_1}(\mathsf{A}, \mathsf{H}) \approx view_{conf_1}(\mathsf{A}, \mathsf{H}).$$

Encryption, hybrid argument: The encryption part is proven with a reactive version of the typical "hybrid arguments", e.g., known from [GoMi 84]. Intuitively, we show that to distinguish the overall views, one has to distinguish at least one particular encryption. (Hence this part of the proof also shows that all other aspects of the simulation are correct.) For this, we define a hybrid machine Hyb_t for each triple $t = (i, s, r) \in \{1, \ldots, R\} \times \mathcal{H}^2$. Roughly, it treats the inputs $in_{i',s',r'}$ for $s', r' \in \mathcal{H}$ as in the real system up to the triple (i, s, r) and afterwards as in the simulation.[47] We write "$\leq$" for the lexicographic order on these triples, t_{max} for their maximum, and $\mathsf{pred}(t)$ for the predecessor of t in this order. Furthermore, let $0 < t$ for all these triples.

We define $\mathsf{Hyb}_0 := \mathsf{TH}' + \mathsf{Sim}'$. For $t = (i, s, r)$, the key exchange of Hyb_t is as in both M' and $\mathsf{TH}' + \mathsf{Sim}'$. We now have to show how Hyb_t computes all values $net_{i',s',r'}$ and $out_{i',s',r'}$.

1. For $(i', s', r') \leq t$, it computes $net_{i',s',r'} \leftarrow \mathsf{comp}(i', s', r', in_{i',s',r'})$ like M and thus M'.
2. For $(i', s', r') > t$, it computes $net_{i',s',r'}$ like $\mathsf{TH}' + \mathsf{Sim}'$:
   a) For $r' \in \mathcal{H}$, this means $net_{i',s',r'} = \epsilon$ if $in_{i',s',r'} = \epsilon$ and thus $busy_{i',s',r'} = \epsilon$; otherwise $net_{i',s',r'} \leftarrow \mathsf{comp}(i', s', r', m_{sim})$.
   b) For $r' \in \mathcal{A}$, it means that $adv\_out_{i',s',r'} = in_{i',s',r'}$ and thus $net_{i',s',r'} \leftarrow \mathsf{comp}(i', s', r', in_{i',s',r'})$ (as in M').
3. For $(i'-1, s', r') \leq t$, it computes $out_{i',s',r'}$ like M', i.e., as $\mathsf{decomp}(i', s', r', net^a_{i'-1,s',r'})$ except that $in_{i'-1,s',r'}$ is output instead if the result is m_{sim}.
4. For $(i'-1, s', r') > t$, it computes $out_{i',s',r'}$ like $\mathsf{TH}' + \mathsf{Sim}'$: It first sets $m = \mathsf{decomp}(i', s', r', net^a_{i'-1,s',r'})$.
   a) If $s' \in \mathcal{H}$:
      - If $m \neq \epsilon$ and $m \neq m_{sim} \vee busy_{i'-1,s',r'} = 0$, then $out_{i',s',r'} = m$ (here $\mathsf{TH}' + \mathsf{Sim}'$ differs from $\mathsf{TH} + \mathsf{Sim}$).
      - If $m = m_{sim} \wedge busy_{i'-1,s',r'} = 1$, then $suppress_{i',s',r'} = 0$ and thus $out_{i',s',r'} = in_{i'-1,s',r'}$.

[46] In more detail, we have shown that the change only concerns a negligible fraction of the runs. Hence the runs of the two configurations are statistically indistinguishable for the class $NEGL$. (Note that they are only of polynomial length.) This implies statistical and thus also computational indistinguishability of all functions of the runs, in particular the views, by Lemma 4.2.

[47] It is not trivial to define hybrid reactive systems generally: If the configurations are probabilistic and have memory, it may not be clear how to initialize the memory of configuration 2 such that it is consistent with the execution of configuration 1 so far. Hence we make the particular construction explicit.

– If $m = \epsilon$, then $suppress_{i',s',r'} = 1$ and therefore $out_{i',s',r'} = \epsilon$.

b) If $s' \in \mathcal{A}$, then m becomes $adv\_in_{i',s',r'}$ and thus $out_{i',s',r'}$ (as in M').

Clearly, $\mathsf{Hyb}_{t_{max}} = \mathsf{M}'$. As Cases 2b and 4b are as in M', each Hyb_t only differs from $\mathsf{Hyb}_{\mathsf{pred}(t)}$ in the computation of $net_{i,s,r}$ and $out_{i+1,s,r}$. Let $conf_{\mathsf{hyb},t} = (\mathsf{Hyb}_t, S, \mathsf{H}, \mathsf{A})$ for all t.

Assume that a distinguisher Dist can distinguish $view_{conf_{\mathsf{hyb},0}}(\mathsf{A},\mathsf{H})$ and $view_{conf_{\mathsf{hyb},t_{max}}}(\mathsf{A},\mathsf{H})$ in contradiction to Definition 2.7, i.e., the function of absolute probability differences is not negligible; we call this function $\Delta(k)$. For all t, let $p_t(k) := P(\mathsf{Dist}(1^k, view_{conf_{\mathsf{hyb},t},k}(\mathsf{A},\mathsf{H}) = 1)$. Then

$$\left|\sum_{t=1}^{t_{max}} (p_t(k) - p_{\mathsf{pred}(t)}(k))\right| = |p_{t_{max}}(k) - p_0(k)| = \Delta(k).$$

Hence for at least one t, the sequence $(|p_t(k) - p_{\mathsf{pred}(t)}(k)|)_{k\in\mathbb{N}}$ is not negligible.[48] Let $t = (i, s, r)$ be such a triple. We assume w.l.o.g. that

$$p_t(k) - p_{\mathsf{pred}(t)}(k) > Q(k)^{-1}$$

for a polynomial Q and infinitely many k, i.e., intuitively that Dist outputs 1 to identify the real system and 0 for the simulation.[49]

Encryption, reduction: We now construct an adversary $\mathsf{A}_{\mathsf{enc}}$ against the encryption scheme (recall Definition 6.2). It uses H, A and Dist as blackboxes. The basic idea is that it simulates either Hyb_t or $\mathsf{Hyb}_{\mathsf{pred}(t)}$ depending on the bit b of the decryption oracle, i.e., without knowing which. It then uses the distinguisher to distinguish the cases, thus obtaining a guess at b. An overview is given in Figure 16.

1. First $\mathsf{A}_{\mathsf{enc}}$ obtains a public key E from Dec. It uses this key in the place of E_r, and generates the other keys of correct participants itself.

2. It simulates M' in interaction with the blackboxes H and A until directly before constructing $net_{i,s,r}$. (So far, M' equals both $\mathsf{Hyb}_{\mathsf{pred}(t)}$ and Hyb_t.) Where decryptions with the unknown key D_r are needed, it asks the decryption oracle Dec.

3. Now, if $in_{i,s,r} \neq \epsilon$, it sends the two messages $m_0 := (s, \mathsf{sign}_s(m_{sim}, i, r))$ and $m_1 := (s, \mathsf{sign}_s(in_{i,s,r}, i, r))$ to Dec. (Otherwise it sends nothing and sets $net_{i,s,r} = \epsilon$.) The former is the message encrypted in $\mathsf{Hyb}_{\mathsf{pred}(t)}$, the latter in Hyb_t. Hence Dec chooses $b \xleftarrow{\mathcal{R}} \{0,1\}$ and sends a ciphertext $c \leftarrow \mathsf{E}_r(m_b)$. Then $\mathsf{A}_{\mathsf{enc}}$ uses c as $net_{i,s,r}$.

4. When the corresponding network message $net^a_{i,s,r}$ (this is how A modified $net_{i,s,r} = c$ in transit) is handled in Round $[i + 1.1]$, $\mathsf{Hyb}_{\mathsf{pred}(t)}$ and Hyb_t also differ.

   - If $net^a_{i,s,r} = c$ (unchanged), Dec will not decrypt it, but $\mathsf{A}_{\mathsf{enc}}$ simply sets $out_{i+1,s,r} = in_{i,s,r}$. We claim that for $b = 0$ and 1, this is what $\mathsf{Hyb}_{\mathsf{pred}(t)}$ and Hyb_t, respectively, would do.
     - $b = 0$: $\mathsf{Hyb}_{\mathsf{pred}(t)}$ acts like $\mathsf{TH}' + \mathsf{Sim}'$: Sim' decrypts c to m_0, which passes all tests. Thus it obtains $m = m_{sim}$. As $in_{i,s,r} \neq \epsilon$, we had $busy_{i,s,r} = 1$ and therefore the output is $in_{i,s,r}$.
     - $b = 1$: Hyb_t acts like M'. It decrypts c to m_1 and, as this passes all tests, output its content $in_{i,s,r}$. (Even if $in_{i,s,r} = m_{sim}$, the effect is the same.)
   - If $net^a_{i,s,r} \neq c$, $\mathsf{A}_{\mathsf{enc}}$ sets $m = \mathsf{decomp}(i, s, r, net^a_{i,s,r})$, using Dec for the decryption.

[48]The standard argument is: There is a polynomial Q with $\Delta(k) > Q(k)^{-1}$ for infinitely many k. For each of them, an index t_k with $|p_{t_k}(k) - p_{\mathsf{pred}(t_k)}(k)| > t_{max}^{-1} Q(k)$ exists. One t must occur infinitely often as t_k.

[49]Otherwise, consider the distinguisher Dist' that outputs 1 iff Dist does not.

– If $m = \epsilon$, it sets $out_{i+1,s,r} = \epsilon$. This is correct for both cases.
– If $m = m_{sim}$, it sets $out_{i+1,s,r} = in_{i,s,r}$. This is correct for M' and thus Hyb_t by definition (here M' differs from M), and for $\mathsf{Hyb}_{\mathsf{pred}(t)}$ because $busy_{i,s,r} = 0$.
– Otherwise, it sets $out_{i+1,s,r} = m$. This is correct for M' and thus Hyb_t by definition of M, and for $\mathsf{Hyb}_{\mathsf{pred}(t)}$ by construction.

5. For all other messages, $\mathsf{Hyb}_{\mathsf{pred}(t)}$ and Hyb_t are the same; for concreteness let $\mathsf{A}_{\mathsf{enc}}$ simulate Hyb_t. Again, if it needs to decrypt a ciphertext c_j with D_r, it asks Dec, except if $c_j = c$. (Intuitively, this is a replay attack.) Then $\mathsf{A}_{\mathsf{enc}}$ must act on its own. The only ciphertexts Hyb_t decrypts with D_r are network messages $net^a_{i'-1,s',r}$, and it only uses the result to compute $out_{i',s',r}$. In these cases, $\mathsf{A}_{\mathsf{enc}}$ simply sets $out_{i',s',r} := \epsilon$. (Correctness is shown below.)

6. At the end, $\mathsf{A}_{\mathsf{enc}}$ inputs the view of the simulated H and A to Dist. It uses the output bit b^* of Dist as its own output.

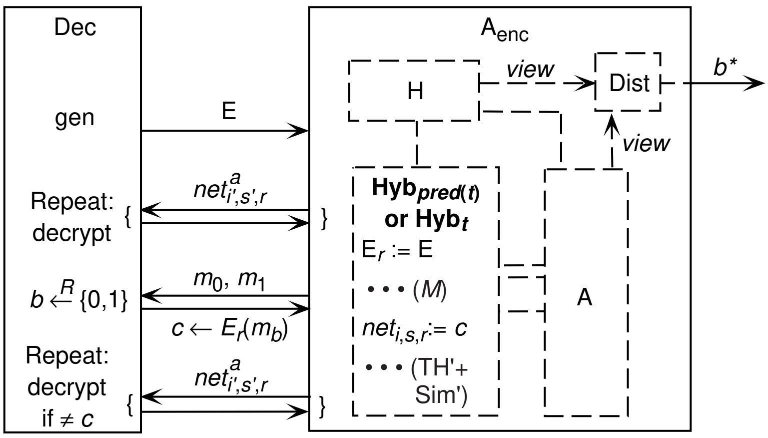

Figure 16: Reduction to an adversary on the encryption scheme

To prove that $\mathsf{A}_{\mathsf{enc}}$, depending on b, simulates either $\mathsf{Hyb}_{\mathsf{pred}(t)}$ or Hyb_t correctly, only the correctness of $out_{i',s',r} = \epsilon$ for $net^a_{i'-1,s',r} = c$ and $(i'-1, s') \neq (i, s)$ remains to be shown.

- $b = 0$: Then $\mathsf{Hyb}_{\mathsf{pred}(t)}$ would decrypt c to $m_0 = (s, sig)$ with $sig = \mathsf{sign}_s(m_{sim}, i, r)$. It first tests that $s = s'$. If yes, it verifies that $\mathsf{test}_s(sig) = (m', i'-1, r)$ for some $m' \in Msg$ and the given (i', r). As we assumed that the message in clear is part of sig, this implies $i'-1 = i$. As $(i'-1, s') \neq (i, s)$, the verifications fail. Hence $\mathsf{Hyb}_{\mathsf{pred}(t)}$ obtains $m = \epsilon$ and sets $out_{i',s',r} = \epsilon$.

- $b = 1$: Then Hyb_t would decrypt c to $m_1 = (s, sig)$ with $sig = \mathsf{sign}_s(in_{i,s,r}, i, r)$. It first tests that $s = s'$. If yes, it verifies that $\mathsf{test}_s(sig) = (m', i'-1, r)$ for any $m' \in Msg$ and the given (i', r). As above, this implies $i'-1 = i$; hence the verifications fail and $out_{i',s',r} = \epsilon$.

Hence the success probability of Dist within $\mathsf{A}_{\mathsf{enc}}$ is the same as in its normal setting. Since $\mathsf{A}_{\mathsf{enc}}$ outputs the same value b^* as Dist, we can compute the success probability $p_{enc}(k)$ of $\mathsf{A}_{\mathsf{enc}}$

as follows:

$$
\begin{aligned}
p_{enc}(k) &= \frac{1}{2}P(b^* = 1|b = 1) + \frac{1}{2}P(b^* = 0|b = 0) \\
&= \frac{1}{2}p_t(k) + \frac{1}{2}(1 - p_{\mathsf{pred}(t)}(k)) \\
&= \frac{1}{2} + \frac{1}{2}(p_t(k) - p_{\mathsf{pred}(t)}(k)) \\
&> \frac{1}{2} + \frac{1}{2Q(k)}
\end{aligned}
$$

for infinitely many k. This contradicts the security of the encryption scheme, and thus the assumption that a distinguisher Dist exists as described. Hence $view_{conf_{\mathsf{hyb},0}}(\mathsf{A},\mathsf{H}) \approx view_{conf_{\mathsf{hyb},t_{max}}}(\mathsf{A},\mathsf{H})$. As $\mathsf{Hyb}_0 = \mathsf{TH}' + \mathsf{Sim}'$ and $\mathsf{Hyb}_{t_{max}} = \mathsf{M}'$, this means

$$view_{conf'_2}(\mathsf{A},\mathsf{H}) \approx view_{conf'_1}(\mathsf{A},\mathsf{H}).$$

All parts of the proof together and transitivity therefore imply the desired result

$$view_{conf_1}(\mathsf{A},\mathsf{H}) \approx view_{conf_2}(\mathsf{A},\mathsf{H}).$$

■

References

[Abad 98] Martín Abadi: Protection in Programming-Language Translations; 25th International Colloquium on Automata, Languages and Programming (ICALP), LNCS 1443, Springer-Verlag, Berlin 1998, 868-883.

[Bara 64] Paul Baran: On Distributed Communications: IX. Security, Secrecy, and Tamper-Free Considerations; Memorandum RM-3765-PR, August 1964, The Rand Corporation. Reprinted in: Lance J. Hoffman (ed.): Security and Privacy in Computer Systems; Melville Publishing Company, Los Angeles, 1973, 99-123.

[BDPR 98] Mihir Bellare, Anand Desai, David Pointcheval, Phillip Rogaway: Relations Among Notions of Security for Public-Key Encryption Schemes; Crypto '98, LNCS 1462, Springer-Verlag, Berlin 1998, 26-45.

[Beav5 91] Donald Beaver: Secure Multiparty Protocols and Zero Knowledge Proof Systems Tolerating a Faulty Minority; Journal of Cryptology 4/2 (1991) 75-122.

[BeCK1 98] Mihir Bellare, Ran Canetti, Hugo Krawczyk: A modular approach to the design and analysis of authentication and key exchange protocols; 13th Symposium on Theory of Computing (STOC), ACM, New York 1998, 419-428.

[BeRo1 94] Mihir Bellare, Phillip Rogaway: Entity Authentication and Key Distribution; Crypto '93, LNCS 773, Springer-Verlag, Berlin 1994, 232-249.

[BrDo 84] Andrei Z. Broder, Danny Dolev: Flipping coins in many pockets (Byzantine agreement on uniformly random values); 25th Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 1984, 157-170.

[Brow 95] Randy Browne: An Architecture for Covert Channel Control in RealTime Networks and MultiProcessors; IEEE Symposium on Research in Security and Privacy, IEEE Computer Society, 1995, 155-168.

[CaGo 99] Ran Canetti, Shafi Goldwasser: An Efficient Threshold Public Key Cryptosystem Secure Against Adaptive Chosen Ciphertext Attack; Eurocrypt '99, LNCS 1592, Springer-Verlag, Berlin 1999, 90-106.

[Cane 96] Ran Canetti: Studies in Secure Multiparty Computation and Applications; Thesis, Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, June 1995, revised March 1996.

[Cane 00] Ran Canetti: Security and Composition of Multiparty Cryptographic Protocols; Journal of Cryptology 13/1 (2000) 143-202. (Previously Theory of Cryptography Library 98-18, `http://philby.ucsd.edu/cryptolib.html`.)

[ChDw 89] Benny Chor, Cynthia Dwork: Randomization in Byzantine Agreement; JAI Press, Advances in Computing Research Vol. 5, Greenwich 1989, 443-497.

[ChPe1 93] David Chaum, Torben Pryds Pedersen: Wallet Databases with Observers; Crypto '92, LNCS 740, Springer-Verlag, Berlin 1993, 89-105.

[CrSh1 98] Ronald Cramer, Victor Shoup: A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack; Crypto '98, LNCS 1462, Springer-Verlag, Berlin 1998, 13-25.

[Damg 88] Ivan Bjerre Damgård: Collision free hash functions and public key signature schemes; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 203-216.

[DiHe 76] Whitfield Diffie, Martin E. Hellman: New Directions in Cryptography; IEEE Transactions on Information Theory 22/6 (1976) 644-654.

[FiHM 99] Matthias Fitzi, Martin Hirt, Ueli Maurer: General Adversaries in Unconditional Multi-Party Computation; Asiacrypt '99, LNCS 1716, Springer-Verlag, Berlin 1999, 232-246.

[GeMi 95] Rosario Gennaro, Silvio Micali: Verifiable Secret Sharing as Secure Computation; Eurocrypt '95, LNCS 921, Springer-Verlag, Berlin 1995, 168-182.

[GMW 87] Oded Goldreich, Silvio Micali, Avi Wigderson: How to play any mental game or a completeness theorem for protocols with honest majority; 19th Symposium on Theory of Computing (STOC), ACM, New York 1987, 218-229.

[Gold 93] Oded Goldreich: A Uniform-Complexity Treatment of Encryption and Zero-Knowledge; Journal of Cryptology 6/1 (1993) 21-53.

[Gold 98] Oded Goldreich: Secure Multi-Party Computation; Working Draft, Version 1.1, September 21, 1998, `http://www.wisdom.weizmann.ac.il/users/oded/pp.htm`.

[Gold 99] Oded Goldreich: Modern Cryptography, Probabilistic Proofs and Pseudorandomness; Algorithms and Combinatorics 17, Springer-Verlag, Berlin 1999.

[Gold1 95] Oded Goldreich: Foundations of Cryptography (Fragments of a Book); Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 1995, `http://theory.lcs.mit.edu/~oded/` (with updates).

[GoLe 91] Shafi Goldwasser, Leonid Levin: Fair Computation of General Functions in Presence of Immoral Majority; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 77-93.

[GoMi 84] Shafi Goldwasser, Silvio Micali: Probabilistic Encryption; Journal of Computer and System Sciences 28 (1984) 270-299.

[GoMR 88] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM Journal on Computing 17/2 (1988) 281-308.

[GoMR 89] Shafi Goldwasser, Silvio Micali, Charles Rackoff: The Knowledge Complexity of Interactive Proof Systems; SIAM Journal on Computing 18/1 (1989) 186-207.

[GoOr 94] Oded Goldreich, Yair Oren: Definitions and Properties of Zero-Knowledge Proof Systems; Journal of Cryptology 7/1 (1994) 1-32.

[HiMa 00] Martin Hirt, Ueli Maurer: Player Simulation and General Adversary Structures in Perfect Multiparty Computation; Journal of Cryptology 13/1 (2000) 31-60. (Previously `http://www.inf.ethz.ch/personal/hirt/publications/journal.ps.gz`, 1997.)

[Hoar2 85] Charles Anthony Richard Hoare: Communicating Sequential Processes; International Series in Computer Science, Prentice Hall, Hemel Hempstead 1985.

[Koch 96] Paul Kocher: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems; Crypto '96, LNCS 1109, Springer-Verlag, Berlin 1996, 104-113.

[Lipn 75] Steven B. Lipner: A Comment on the Confinement Problem; 5th Symposium on Operating Systems Principles (SOSP), Operating Systems Review 9/5 (1975) 192-196.

[LMMS 98] P. Lincoln, J. Mitchell, M. Mitchell, A. Scedrov: A Probabilistic Poly-Time Framework for Protocol Analysis; 5th ACM Conference on Computer and Communications Security, ACM Press, New York 1998, 112-121.

[Lync 96] Nancy Lynch: Distributed Algorithms; Morgan Kaufmann Publishers, San Francisco 1996.

[MiRo 92] Silvio Micali, Phillip Rogaway: Secure Computation; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 392-404. (Chaptes 1-3 of longer report version distributed at Crypto '91.)

[Oren 87] Yair Oren: On the cunning power of cheating verifiers: some observations about zero-knowledge proofs; 28th Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 1987, 462-471.

[Pfit4 93] Birgit Pfitzmann: Sorting Out Signature Schemes; 1st ACM Conference on Computer and Communications Security, ACM Press, New York 1993, 74-85.

[Pfit6 96] Birgit Pfitzmann: Cryptographic Semantics of Formal Specifications; presented at the Colloquium on Formal Methods and Security, Isaac Newton Institute, University of Cambridge, 23.4.1996.

[PfSW 00] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Cryptographic Security of Reactive Systems; Workshop on Secure Architectures and Information Flow, Dec. 1999, Electronic Notes in Theoretical Computer Science (ENTCS), `http://www.elsevier.nl/locate/entcs/volume32.html`, March 2000.

[PfSW2 00] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Provably Secure Certified Mail; IBM Research Report RZ 3207 (#93253) 02/14/2000, IBM Research Division, Zürich, Feb. 2000.

[PfWa 94] Birgit Pfitzmann, Michael Waidner: A General Framework for Formal Notions of "Secure" Systems; Hildesheimer Informatik-Berichte 11/94, University of Hildesheim, April 1994, `http://www.semper.org/sirene/lit/abstr94.html#PfWa_94`.

[PfWa3 98] Birgit Pfitzmann, Michael Waidner: Extensions to Multi-Party Computations; Slides, presented at: The 1998 Weizmann Workshop on Cryptography, June 16-18th, Weizmann Institute of Science, Rehovot, Israel, `http://www.semper.org/sirene/lit/abstr98.html#PfWa3_98`.

[RaSi 92] Charles Rackoff, Daniel R. Simon: Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 433-444.

[Schu 00] Matthias Schunter: Optimistic Fair Exchange; submitted as Ph.D. thesis, Saarland University, Feb. 2000, `http://www.semper.org/sirene/publ/Schu_00.thesis.ps.gz`.

[Shou 99] Victor Shoup: On Formal Models for Secure Key Exchange; IBM Research Report RZ 3076 (##93122), IBM Research Division, Zürich, November 1998. Also as Theory of Cryptography Library 99-12, last revised November 1999, `http://philby.ucsd.edu/cryptolib/`.

[ToW1 87] Martin Tompa, Heather Woll: Random self-reducibility and zero knowledge proofs of possession of information; 28th Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 1987, 472-482.

[Tros 98] Jonathon T. Trostle: Timing Attacks Against Trusted Path; IEEE Symposium on Research in Security and Privacy, IEEE Computer Society, 1998, 125-134.

[VeWo 95] Balaji R. Venkatraman, R. E. Newman-Wolfe: Capacity Estimation and Auditability of Network Covert Channels; IEEE Symposium on Research in Security and Privacy, IEEE Computer Society, 1995, 186-198.

[Yao 82] Andrew C. Yao: Protocols for Secure Computations; 23rd Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 1982, 160-164.

[Yao1 82] Andrew C. Yao: Theory and Applications of Trapdoor Functions; 23rd Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 1982, 80-91.