RZ 3234 (#93280) 06/12/00
Computer Science/Mathematics 13 pages

Research Report

Composition and Integrity Preservation of Secure Reactive Systems

Birgit Pfitzmann[1], Michael Waidner[2]

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

[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

Composition and Integrity Preservation of Secure Reactive Systems

Birgit Pfitzmann, Michael Waidner

May 7, 2000

Abstract

We consider compositional properties of reactive systems that are secure in a cryptographic sense. We follow the well-known simulatability approach, i.e., the specification is an ideal system and a real system should in some sense simulate it. We recently presented the first detailed general definition of this concept for *reactive* systems that allows abstraction and enables proofs of efficient real-life systems like secure channels or certified mail.

We prove two important properties of this definition, preservation of integrity and secure composition: First, a secure real system satisfies all integrity requirements (e.g., safety requirements expressed in temporal logic) that are satisfied by the ideal system. Secondly, if a composed system is designed using an ideal subsystem, it will remain secure if a secure real subsystem is used instead. Such a property was so far only known for non-reactive simulatability.

Both properties are important for putting formal verification methods for systems using cryptography on a sound basis.

1 Introduction

Security proofs for systems involving cryptography are getting increasing attention in theory and practice, and they are used for increasingly large systems. While for some time most of the effort concentrated on primitives like encryption and signature schemes, or authentication and key exchange, currently work is under way on medium-sized systems like secure channels, payment systems, and anonymity systems. In the future, one might want to prove even larger systems like entire electronic-commerce architectures or distributed operating systems that use cryptography.

Both the cryptographic and the formal-methods community are working on such proofs, and the techniques are quite disjoint. One of our goals is to link them to get the best overall results: proofs that allow abstraction and the use of formal methods, but retain a sound cryptographic semantics.

1.1 Abstracted Models

In the formal-methods community, one tries to use established specification techniques to specify requirements and actual protocols unambiguously and with a clear semantics. Moreover, most work aims at proofs that are at least automatically verifiable. To make this possible, cryptographic operations are almost always treated as an infinite term algebra where only predefined equations hold (in other terminology, the initial model of an abstract data type) as introduced in [9]. For instance, there is a pair of operators E_X and D_X for asymmetric en- and decryption with a key pair of a participant X. Two encryptions of a message m from a basic message space M do not yield another message from M, but the term $E_X(E_X(m))$. The equation $D_X(E_X(t)) = t$ for all terms t is defined, and the proofs rely on the abstraction that no equations hold unless they can be derived syntactically from the given ones. Early work using this approach for tool-supported proofs was rather specific, e.g., [22, 20]; nowadays most work is based on standard languages and general-purpose verification tools, as initiated, e.g., in [27, 18, 1].

A problem with these models is the lack of a link between the chosen abstractions and the real cryptographic primitives as defined and sometimes proven in cryptography. The main issue is not even that one will somehow need to weaken the statements to polynomial-time adversaries and allow error probabilities; the problem is that the cryptographic definitions say nothing about *all* equations. For instance, the accepted cryptographic definition of secure asymmetric encryption only requires that an adversary in a strong type of attack cannot find out anything about the message (see [5, 8]), but nothing about possible relations on the ciphertexts. One can construct examples, at least artificial ones, where proofs made with the abstractions go wrong with encryption schemes provably secure in the cryptographic sense [26].

1.2 Faithful Abstraction

The problem can be approached from both sides—cryptography can try to offer stronger primitives closer to the typical abstractions, or formal methods can be applied based on weaker abstractions that are easier to fulfil by actual cryptography. (Our examples in [24, 25] belong to the second approach.) Both approaches presuppose that one defines what it means that some abstraction is fulfilled in a cryptographic sense. Both also need proofs that working with the abstractions leads to meaningful results in the real cryptographic sense, i.e., the abstractions should be faithful. This is illustrated in Figure 1. We will show how the theorems proven in this paper help in this program.

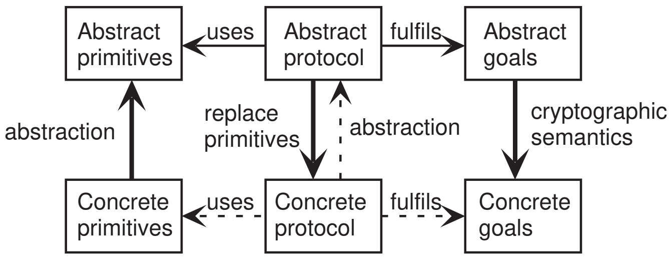

Figure 1: Goals of faithful abstraction. Bold arrows should be defined once and for all, normal arrows once per protocol. It should be proven that dashed arrows follow automatically.

1.3 What does Cryptography Gain from Abstractions?

Cryptographers may ask why one should bother with abstractions: why not continue to make all proofs on the lower layer of Figure 1, i.e., as reduction proofs if asymmetric cryptography is involved? Indeed, abstractions and formal methods neither increase the expressiveness nor make the overall results more rigorous. However, one can hope that the specifications get nicer, the proofs shorter, and tool support easier. (The large number of papers using unfaithful abstractions indicates how alluring these arguments are.) Indeed rigorous cryptographic definitions are long (always involving details of the attack, error probabilities etc.), and many had to be strengthened later. Similarly, most proofs are currently only sketches, and some have contained gaps. This will get worse with larger systems; just imagine proving an entire electronic-commerce framework by a reduction only because it uses signatures in some places.

1.4 Related Literature

There are some approaches at expressing *actual* cryptographic definitions in formal languages. This is almost orthogonal to our goal of providing *abstractions* with a cryptographic semantics. As also none of them captures the entire definitions yet, we do not duplicate the overview from [26].

The main abstraction approach in cryptography is simulatability: One specifies an ideal system that has all the desired properties by construction, but typically makes the unrealistic assumption that there is one machine trusted by all parties ("trusted host"). A real system is defined to be as secure as an ideal system if anything an adversary can achieve can also be achieved by an adversary attacking the ideal system.

This approach was primarily worked out for *function evaluation*, i.e., all parties make one input at the beginning and get one output at the end [28, 12, 3, 21, 6]. For this case, [6] contains a composition theorem. Our Section 4 can be seen as an extension of this to reactive systems.

There are two main approaches at extending simulatability to *reactive systems*, i.e., systems where users make inputs and obtain outputs many times. The first, constructive, approach describes the ideal system as a global state-transition machine and requires the global state to be shared among all participants in the real system [14, 11]. This is not feasible or desirable in scenarios like secure channels or payment systems, where many participants carry out many 2- or 3-party subprotocol runs at different times.

The second, descriptive, approach only considers the "outside" behaviour of the system. Several brief sketches have been around for some time (among them ours, hence all are omitted here). Three more detailed definitions have been made [16, 17, 15, 24]. For none of them, theorems about composition or the preservation of integrity properties have been given yet, hence the current paper is novel in any case. The main advantage of [16, 17] is a formal language, π-calculus, as machine model. However, it lacks abstraction: the specifications in both papers essentially comprise the actual protocols and are specific to certain cryptographic primitives used. Hence tool support would also not be possible yet because even the specifications involve ad-hoc notation for generating random primes etc. (Combining some of their language techniques with our abstraction techniques looks promising.) In [15], a somewhat restricted class of systems is considered (straight-line programs and information-theoretic security) because their main goal was general constructions. Particular aspects of [24] (also in comparison with other sketches) are a precise timing model that exposes timing vulnerabilities, a precise treatment of the interaction of users and adversaries, and independence of the trust model. The simulatability approach has also been applied to specific reactive problems [10, 4, 7, 25], but this has not much bearing on composition as studied here.

A formal abstraction specifically for symmetric encryption with cryptographic semantics has been described in [2][1], but without considering its use within a system. An approach to provide integrity properties with a cryptographic semantics was first made in [23], but not as rigorously as here and without a relation to simulatability definitions.

1.5 Organization of this Paper

In this paper, we investigate compositional properties of systems that are secure in the sense of simulatability. We repeat the model we use in Section 2. In Section 3 we define what it means for a system to

[1]They show that if two expressions of a certain class are equivalent in a formal calculus for an adversary, and are interpreted using a symmetric encryption scheme with some special security properties, then the resulting two random variables are computationally indistinguishable.

provide almost arbitrary integrity properties in a cryptographic sense. We then prove that (a) proofs of such properties made for the ideal system also hold for the real system and (b) logic derivations among integrity properties are valid for the real system in the cryptographic sense. In Section 4 we define the composition of systems in our model. We then prove that the specification of one system can be used in the design of another system while preserving statements about the security of the overall systems.

All the derivations mentioned can be abstract, unless the higher layer in a composition uses cryptography itself. In particular, the integrity properties in (a) are accessible to model checkers, those in (b) to theorem provers.

2 Summary of the Definitions

In this section, we repeat the definitions from [24] in slightly abbreviated form. They are for a synchronous network model, and the simulatability also includes the timing. Hence security exposures via timing channels are captured. To avoid that timing differences within a round leak, implementations of synchronous machines have to ensure that input reading and outputting are both clocked.

The machine model is probabilistic state-transition machines, similar to probabilistic I/O automata as sketched in [19]. For clarity, one particular notation and semantics is fixed.

Definition 2.1 (Machines and Ports) A *name* is a string over a fixed alphabet Σ.

A *port* p is a pair $(name_p, dir_p)$ of a name and a Boolean value called direction; we write $name_p?$ and $name_p!$ for in- and output ports, respectively. We write p^c for the *complement* of a port p, i.e., $name_p!^c = name_p?$ and vice versa. For a set P of ports, let $\mathsf{In}(P) := \{p \in P | dir_p = ?\}$ denote the input ports and similarly $\mathsf{Out}(P)$ the output ports.

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 s 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 analogous tuples O. δ_M maps each such pair (s, I) to a finite distribution over pairs (s', O). For a set M of machines, let $\mathsf{ports}(M) := \bigcup_{\mathsf{M} \in M} Ports_\mathsf{M}$.

"Machine M_1 has machine M_2 as a (blackbox) *submachine*" means that it has the state-transition function as a blackbox. Hence M_1 can "clock" M_2, i.e., decide when to cause state transitions. $\diamond$

For computational aspects, each machine is regarded as implemented by a probabilistic interactive Turing machine [13], and each port by a communication tape. The complexity of a machine is measured in terms of the length of the initial state, represented as initial worktape content (often a security parameter).

Below, we distinguish correct machines, adversaries and users in particular in how they are clocked, because one cannot assume adversaries to adhere to synchronization rules. As some proofs need different clocking schemes, general collections of machines and their runs with a clocking scheme are defined.

Definition 2.2 (Machine Collections, Runs and Views) A *collection* C is a finite set of machines with pairwise disjoint sets of ports. Each set of complementary ports $c = \{p, p^c\} \subseteq \mathsf{ports}(C)$ is called a *connection* and the set of these connections the *connection graph* $\mathsf{G}(C)$. By $\mathsf{free}(C)$ we denote the *free* ports, i.e., $p \in \mathsf{ports}(C)$ but $p^c \notin \mathsf{ports}(C)$. A collection is *closed* if $\mathsf{free}(C) = \emptyset$.

A *clocking scheme* is a mapping κ (also written as a tuple) from a set $\{1, \ldots, n\}$ to the powerset of C, i.e., it assigns each number a subset of the machines. Given C and κ and a tuple $ini \in Ini := \times_{\mathsf{M} \in C} Ini_\mathsf{M}$ of initial states, *runs* (or "executions" or "traces") are defined: 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 (probabilistically). The output at a port $\mathsf{p}!$ is available as input at $\mathsf{p}?$ until the machine with port $\mathsf{p}?$ is clocked 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, each run is a function mapping each triple $(\mathsf{M}, i, j) \in C \times \mathbb{N} \times \{1, \ldots, n\}$ to a quadruple (s, I, s', O) of the old state, inputs, new state, and outputs of machine M in subround $[i.j]$, with a symbol

ϵ if M not clocked in this subround. For a number $l \in \mathbb{N}$ of rounds, l-round prefixes $run_{C,ini,l}$ of runs are defined in the obvious way. For a function $l : Ini \to \mathbb{N}$, this gives a family $run_{C,l} = (run_{C,ini,l(ini)})_{ini \in Ini}$.

The view of a subset M of a closed collection C in a run r is the restriction of r to $M \times \mathbb{N} \times \{1, \ldots, n\}$.[2] This gives a family of random variables

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

and similarly for l-round prefixes.

For a run r and a set P of ports, let $r\lceil_P$ denote its restriction to these ports. This notation is carried over to the random variables. $\diamond$

Now we define specific collections for security purposes, first the system part and then the environment, i.e., users and adversaries. Typically, a cryptographic system is described by an *intended structure*, and the actual structures are derived using a *trust model*: the adversary replaces some machines and taps or completely controls some channels. A concrete derivation is defined in [24]. However, as a wide range of trust models is possible, it is useful to keep the remaining definitions independent of them by a general system definition.

Definition 2.3 (Structures and Systems) 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*. Let $\bar{S} := \mathsf{free}(M) \setminus S$ and $\mathsf{forb}(M, S) := \mathsf{ports}(M) \cup \bar{S}^c$.

A *system Sys* is a set of structures. $\diamond$

The separation of the free ports into specified ports and others is an important feature of this particular reactive simulatability definition. The specified ports are those where a certain service is guaranteed. Typical examples of inputs at specified ports are "send message m to id" for a message transmission system or "pay amount x to id" for a payment system. The ports in $\bar{S}$ are additionally available for the adversary. The ports in $\mathsf{forb}(M, S)$ will therefore be forbidden or at least unusual for an honest user to have. In the simulatability definition below, only the events at specified ports have to be simulated one by one. This allows *abstract* specification of systems with *tolerable imperfections*. For instance, if the traffic pattern is not hidden (as in almost all cryptographic protocols for efficiency reasons), one can abstractly specify this by giving the adversary one busy-bit per message in transit in the ideal system. Even better, he should only get one busy-bit per subprotocol run (e.g., a payment) and the internal message pattern of the subprotocol should not tell him more. Detailed examples and more motivation are given in [24, 25].

The following definition contains another important aspect: Both honest users and an adversary are modeled as stateful machines H and A apart from the system. First, honest users should not be modeled as part of the machines in M because they are arbitrary, while the machines have prescribed programs. Secondly, they should not be replaced by a quantifier over input sequences, because they may have arbitrary strategies which message to input next to the system after obtaining certain outputs. They may even be influenced in these choices by the adversary, e.g., in chosen-message attacks on a signature scheme; thus H and A may communicate. At least in the computational case, arbitrary strategies (i.e., adaptive attacks) are not known to be replaceable by arbitrary input sequences. Thirdly, honest users are not a natural part of the adversary 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.

Definition 2.4 (Configuration) A *configuration conf* of a system *Sys* is a tuple $(M, S, \mathsf{H}, \mathsf{A})$ where $(M, S) \in Sys$ is a structure and $C = M \cup \{\mathsf{H}, \mathsf{A}\}$ a closed collection.

The set of configurations is written $\mathsf{Conf}(Sys)$, and those with polynomial-time user and adversary $\mathsf{Conf}_{\mathsf{poly}}(Sys)$. "poly" is omitted if it is clear from the context.

Runs and views of a configuration are given by Definition 2.4 with the clocking scheme $(M \cup \{\mathsf{H}\}, \{\mathsf{A}\}, \{\mathsf{H}\}, \{\mathsf{A}\})$, except that we end a run if H and A have reached finite state. Typically, the initial states of all machines are only a security parameter k (in unary representation). Then one considers the families of runs and views restricted to the subset $Ini' = \{(1^k)_{\mathsf{M} \in C} | k \in \mathbb{N}\}$ of Ini, and writes run_{conf} and $view_{conf}(M)$ for run_C and $view_C(M)$ restricted to Ini', and similar for l-round prefixes. Furthermore, Ini' is identified with $\mathbb{N}$; hence one can write $run_{conf,k}$ etc. $\diamond$

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

Clocking the adversary between the correct machines is the well-known model of "rushing adversaries". The given clocking of users is as powerful as clocking them in an arbitrary unsynchronized way [24].

In the simulatability definition, one only wants to compare each structure of Sys_1 with certain corresponding structures in Sys_2. An almost arbitrary mapping f is allowed as specification of "corresponding", only certain conventions on the naming of ports are necessary. An instantiation is usually derived from the trust model, and usually only structures with the same set of specified ports are corresponding.

Definition 2.5 (Valid Mapping, Suitable Configuration) A function f from a system Sys_1 to the powerset of a system Sys_2 is called a *valid mapping* if

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

for all structures with $(M_2, S_2) \in f(M_1, S_1)$.

Given Sys_2 and f, the set $\mathsf{Conf}^f(Sys_1) \subseteq \mathsf{Conf}(Sys_1)$ of *suitable* configurations contains all those configurations $(M_1, S_1, \mathsf{H}, \mathsf{A}_1)$ where H has no ports from $\mathsf{forb}(M_2, S_2)$ for any $(M_2, S_2) \in f(M_1, S_1)$. ◇

The restriction to suitable configurations $\mathsf{Conf}^f(Sys_1)$ serves two purposes in simulatability: First it excludes users that are incompatible with (M_2, S_2) simply because of name clashes. Secondly, it excludes that H connects to unspecified free ports of (M_2, S_2). This is necessary for the abstract specification of tolerable imperfections. Recall the example of an ideal system that gives the adversary one busy-bit per subprotocol run. Clearly there is no such bit in the real system; we only need it to capture that whatever the adversary learns in the real system is not more than this bit. As we will require indistinguishability of the views of H, these unspecified ports must only be used by the adversary.

As the definition of computational indistinguishability (originally from [29]) is essential for the simulatability definition, we also present it here.

Definition 2.6 (Indistinguishability) Two families $(\mathsf{var}_k)_{k\in\mathbb{N}}$ and $(\mathsf{var}'_k)_{k\in\mathbb{N}}$ of random variables (or probability distributions) are called

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

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

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

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

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

(Intuitively, Dist, given the security parameter and an element chosen according to either var_k or var'_k, tries to guess which distribution the element came from.) The notation $g(k) \leq 1/\mathsf{poly}(k)$, equivalently $g \in NEGL$, 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 treat all cases together. ◇

The following definition captures that whatever an adversary can achieve in the real system against certain honest users, another adversary can achieve against the same honest users in the ideal system. Adding an adversary output in the comparison does not make the definition stricter, nor do auxiliary inputs [24].

Definition 2.7 (Simulatability) Let systems Sys_1 and Sys_2 with a valid mapping f be given.

a) We say $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{perf}} Sys_2$ (*perfectly at least as secure as* for f) 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

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

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

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

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

Definition 2.8 (Blackbox and Universal Simulatability) Universal simulatability means that A_2 in Definition 2.7 does not depend on H (only on M_1, S_1, and A_1). Blackbox simulatability means that additionally, A_2 (given M_1, S_1) is a fixed simulator Sim with A_1 as a blackbox submachine. ◇

We need the following lemmas (the first is well-known and easily proved, the other two are from [24]).

Lemma 2.1 (Indistinguishability) Indistinguishability of two families of random variables implies indistinguishability of any function ϕ of them (in particular restrictions). For the computational case, ϕ must be polynomial-time computable.

A step in the proof that we also need separately is that the statistical distance $\Delta(\phi(\mathsf{var}_k), \phi(\mathsf{var}'_k))$ between a function of two random variables is at most $\Delta(\mathsf{var}_k, \mathsf{var}'_k)$. □

Lemma 2.2 (Combination of Machines) The *open* and *hiding combinations*, $\mathsf{D_o}$ and $\mathsf{D_h}$, of a subset $D \subseteq C$ of a collection are machines with all the original machines as submachines. While $Ports_{\mathsf{D_o}} = \mathsf{ports}(D)$, in $\mathsf{D_h}$ internal connections are hidden, i.e., $Ports_{\mathsf{D_h}} = \mathsf{free}(D)$. Both are clocked whenever a machine from D is. The transition function is defined by switching the submachines in the same subrounds where they would be clocked externally, and in $\mathsf{D_h}$ also the internal connections.

In the resulting collection $C^* = C \setminus D \cup \{\mathsf{D_x}\}$, where $\mathsf{x} \in \{\mathsf{o}, \mathsf{h}\}$, the restriction of the runs to any tuple of original machines or ports is the same as in C.

If only $\mathsf{D_x}$ is clocked in a continuous range of subrounds, one can derive a machine $\mathsf{D'_x}$ clocked only once in these subrounds (internally it calls the transition functions of its submachines in the right order) such that the restriction of the runs to any tuple of original machines or ports is still unchanged except for this subround renaming. □

Lemma 2.3 (Transitivity) If $Sys_1 \geq^{f_1} Sys_2$ and $Sys_2 \geq^{f_2} Sys_3$, then $Sys_1 \geq^{f_3} Sys_3$, 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, statistical and computational security. It also holds for universal and blackbox simulatability. □

3 Integrity Requirements

In this section, we show how the relation "at least as secure as" relates to explicit properties required of a system, e.g., safety requirements expressed in temporal logic.

In a modular design approach, one regards the trusted host, i.e., the ideal system used as the specification in simulatability, as a refinement of these properties. Hence one verifies these properties for the trusted host. This may be done by formal and even automatic model checking if the trusted host is simple enough. (The trusted hosts in our two larger examples in [24, 25] are indeed without probabilistic and computational aspects or cryptographic operations.) Now we want to show that the real system also fulfills these requirements in a certain cryptographic sense, i.e., even if parts of the system are under control of an adversary, but possibly only for polynomial-time adversaries and negligible error probabilities. This approach corresponds to the right half of Figure 1: the integrity properties serve as abstract goals, the ideal system as an abstract protocol, and the real system fulfils generally defined concrete versions of these goals.

Clearly this can only hold for requirements formulated in terms of in- and outputs of the trusted host at the specified ports, because the simulatability definition only means that the real and the ideal system interact with their users in an indistinguishable way.

As a rather general version of integrity requirements, independent of the concrete formal language, we consider those that have a linear-time semantics, i.e., that correspond to a set of allowed traces of in- and outputs. We allow different requirements for different sets of specified ports, because requirements of various parties in cryptography are often made for different trust assumptions (typically, every party is assumed to trust only their own computer). To make the translation between the two systems meaningful, we only consider mappings f that keep S constant.

Definition 3.1 (Integrity Requirements) An integrity requirement *Req* for a system *Sys* is a function that assigns a set of traces at the ports in S to each set S with $(M, S) \in \mathit{Sys}$. More precisely, such a trace contains one value $v_p \in \Sigma^*$ for each port $p \in S$ and round i, corresponding to the in- or output of the correct machine in Subround $[i.1]$. For the computational and statistical case, the traces must be finite. We say that *Sys* fulfills *Req*

a) perfectly (written $\mathit{Sys} \models_{\mathsf{perf}} \mathit{Req}$) if for any configuration $\mathit{conf} = (M, S, \mathsf{H}, \mathsf{A}) \in \mathsf{Conf}(\mathit{Sys})$, the restrictions $r\lceil_S$ of all runs of this configuration to the specified ports lie in $\mathit{Req}(S)$. In formulas, $[(\mathit{run}_{\mathit{conf},k}\lceil_S)] \subseteq \mathit{Req}(S)$ for all k, where $[\cdot]$ denotes the carrier set of a probability distribution.

b) statistically for a class *SMALL* ($\mathit{Sys} \models_{\mathit{SMALL}} \mathit{Req}$) if for any configuration $\mathit{conf} = (M, S, \mathsf{H}, \mathsf{A}) \in \mathsf{Conf}(\mathit{Sys})$, the probability that $\mathit{Req}(S)$ is not fulfilled is small, i.e., for all polynomials l (and as a function of k),
$$P(\mathit{run}_{\mathit{conf},k,l(k)}\lceil_S \notin \mathit{Req}(S)) \in \mathit{SMALL}.$$

c) computationally ($\mathit{Sys} \models_{\mathsf{poly}} \mathit{Req}$) if for any configuration $\mathit{conf} = (M, S, \mathsf{H}, \mathsf{A}) \in \mathsf{Conf}_{\mathsf{poly}}(\mathit{Sys})$, the probability that $\mathit{Req}(S)$ is not fulfilled is negligible, i.e.,
$$P(\mathit{run}_{\mathit{conf},k}\lceil_S \notin \mathit{Req}(S)) \in \mathit{NEGL}.$$

Note that a) is normal fulfillment. We write "$\models$" if we want to treat all three cases together. ◇

Theorem 3.1 (Conservation of Integrity Properties) Let a system Sys_2 be given that fulfills an integrity requirement *Req*, and let $\mathit{Sys}_1 \geq^f \mathit{Sys}_2$ for a valid mapping f with $S_1 = S_2$ whenever $(M_2, S_2) \in f(M_1, S_1)$. Then also $\mathit{Sys}_1 \models \mathit{Req}$.

This holds in the perfect and statistical sense, and in the computational sense if membership in the set $\mathit{Req}(S)$ is decidable in polynomial time for all S. □

Proof. We first show that *Req* is defined on Sys_1 under the preconditions: Simulatability implies that for each $(M_1, S_1) \in \mathit{Sys}_1$, there exists $(M_2, S_2) \in f(M_1, S_1)$. Then $S_1 = S_2$ by the precondition, and thus $\mathit{Req}(S_1)$ is defined. The idea for the rest of the proof is that if Sys_1 did not fulfill the requirement while Sys_2 does, this would offer a possibility to distinguish the systems.

Assume that a configuration $\mathit{conf}_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}(\mathit{Sys}_1)$ contradicts the theorem. Let $\mathsf{H_h}$ be the hiding combination of H and A_1.[3] By Lemma 2.2, this does not change the probability of the runs restricted to S_1. As all other machines are in M_1 and clocked only in Subround 1, we can even clock $\mathsf{H_h}$ in Subround 3 only and the runs change only by subround renaming. In particular, the traces at the ports in S_1 as considered in the theorem remain the same. We now add an adversary $\mathsf{A_{null}}$ without ports (and doing nothing) to obtain a configuration $\mathit{conf}_{\mathsf{h},1} = (M_1, S_1, \mathsf{H_h}, \mathsf{A_{null}}) \in \mathsf{Conf}(\mathit{Sys}_1)$. We then have $\mathit{run}_{\mathit{conf}_{\mathsf{h},1}}\lceil_{S_1} = \mathit{run}_{\mathit{conf}_1}\lceil_{S_1}$, and thus $\mathit{conf}_{\mathsf{h},1}$ also contradicts the theorem.

Moreover, $\mathit{conf}_{\mathsf{h},1}$ is a suitable configuration, i.e., $\mathsf{H_h}$ has no ports from $\mathsf{forb}(M_2, S_2)$ for any $(M_2, S_2) \in f(M_1, S_1)$ because $\mathit{Ports}_{\mathsf{H_h}} = \mathsf{free}(M_1)^c$ (as the collection is closed and $\mathsf{H_h}$ has no self-connections by construction) and Condition 1 on valid mappings.

[3] Note that we have not required that any user H connects to all ports from S_1, but we need all these ports to be in a user view for exploiting simulatability.

Hence there exists an indistinguishable configuration $conf_{\mathsf{h},2} = (M_2, S_2, \mathsf{H_h}, \mathsf{A_{h,2}}) \in \mathsf{Conf}(Sys_2)$, i.e., $view_{conf_{\mathsf{h},1}}(\mathsf{H_h}) \approx view_{conf_{\mathsf{h},2}}(\mathsf{H_h})$. By the precondition, the requirement is fulfilled for this configuration (perfectly, statistically, or computationally). Furthermore, the view of $\mathsf{H_h}$ in both configurations contains the trace at $S := S_1 = S_2$, i.e., the trace is a function $\lceil_S$ of the view.

In the perfect case, the distribution of the views is identical. This immediately contradicts the assumption that $[(run_{conf_{\mathsf{h},1},k}\lceil_S)] \not\subseteq Req(S)$ while $[(run_{conf_{\mathsf{h},2},k}\lceil_S)] \subseteq Req(S)$.

In the statistical case, let any polynomial l be given. The statistical distance $\Delta(view_{conf_{\mathsf{h},1},k,l(k)}(\mathsf{H_h}), view_{conf_{\mathsf{h},2},k,l(k)}(\mathsf{H_h}))$ is a function $g(k) \in SMALL$. We apply Lemma 2.1 to the characteristic function $1_{v\lceil_S \notin Req(S)}$ on such views v. This gives $|P(run_{conf_{\mathsf{h},1},k,l(k)}\lceil_S \notin Req(S)) - P(run_{conf_{\mathsf{h},2},k,l(k)}\lceil_S \notin Req(S))| \le g(k)$. As $SMALL$ is closed under addition and under making functions smaller, this gives the desired contradiction.

In the computational case, we define a distinguisher Dist: Given a view of machine $\mathsf{H_h}$, it extracts the run restricted to S and verifies if the result lies in $Req(S)$. If yes, it outputs 0, otherwise 1. This distinguisher is polynomial-time (in the security parameter k) because the view of $\mathsf{H_h}$ is of polynomial length, and membership in $Req(S)$ was required to be polynomial-time decidable. Its advantage in distinguishing is $|P(\mathsf{Dist}(1^k, view_{conf_{\mathsf{h},1},k}) = 1) - P(\mathsf{Dist}(1^k, view_{conf_{\mathsf{h},2},k}) = 1)| = |P(run_{conf_{\mathsf{h},1},k}\lceil_S \notin Req(S)) - P(run_{conf_{\mathsf{h},2},k}\lceil_S \notin Req(S))|$. If this difference were negligible, then so would the first term be because the second term is and $NEGL$ is closed under addition. Again this is the desired contradiction. ■

We now show that, if integrity requirements are formulated in a logic (e.g., temporal logic, or first-order logic with round numbers), abstract derivations in the logic are valid in the cryptographic sense. As our definitions are not based on a specific logic, but on the linear-time semantics, the following theorem represents this (see below).

Theorem 3.2 (Using Logics for Integrity Properties)

a) If $Sys \models Req_1$ and $Req_1 \subseteq Req_2$, then also $Sys \models Req_2$.

b) If $Sys \models Req_1$ and $Sys \models Req_2$, then also $Sys \models Req_1 \cap Req_2$.

Here "$\subseteq$" and "'$\cap$" are interpreted pointwise, i.e., for each S. This holds in the perfect and statistical sense, and in the computational sense if for a) membership in $Req_2(S)$ is decidable in polynomial time for all S. □

Proof. Part a) is trivially fulfilled in all three cases. Part b) is trivial in the perfect case. For the statistical case and any $conf = (M, S, \mathsf{H}, \mathsf{A}) \in \mathsf{Conf}(Sys)$,

$$\begin{aligned} & P(run_{conf,k,l(k)}\lceil_S \notin (Req_1(S) \cap Req_2(S)) \\ \le\ & P(run_{conf,k,l(k)}\lceil_S \notin Req_1(S)) + P(run_{conf,k,l(k)}\lceil_S \notin Req_2(S)) \\ \in\ & SMALL \end{aligned}$$

because both summands are in $SMALL$, which is closed under addition. The computational case holds analogously because $NEGL$ is closed under addition. ■

For applying this theorem to concrete logics, the main rule to consider is usually modus ponens, i.e., if one has derived that a and $a \to b$ are valid in a given model, then b is also valid in this model. If Req_a etc. denote the semantics of the formulas, i.e., the trace sets they represent, we have to show that

$$(Sys \models Req_a \wedge Sys \models Req_{a\to b}) \Rightarrow Sys \models Req_b.$$

This follows directly from the theorem with $Req_a \cap Req_{a\to b} = Req_{a\wedge b} \subseteq Req_b$.

4 Composition

In this section, we show that the relation "at least as secure as" is consistent with the composition of systems. The basic idea is the following: Assume that we have proven that a system Sys_0 is as secure as another system Sys'_0 (typically an ideal system used as a specification). Now we would like to use Sys_0 as a secure replacement for Sys'_0, i.e., as an implementation of the specification Sys'_0.

Usually, replacing Sys'_0 means that we have another system Sys_1 that uses Sys'_0; we call this composition Sys^*. Inside Sys^* we want to use Sys_0 instead, which gives a composition $Sys^\#$. Hence $Sys^\#$ is typically a completely real system, while Sys^* is partly ideal. Intuitively we expect $Sys^\#$ to be at least as secure as Sys^*. The situation is shown in the left and middle part of Figure 2.

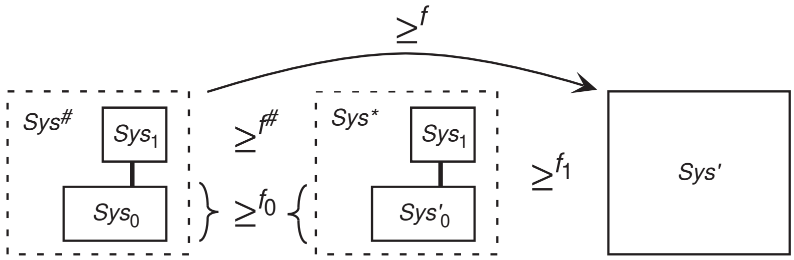

Figure 2: Composition theorem and its use in a modular proof: The left and middle part show the statement of Theorem 4.1, the right part Corollary 4.1.

In terms of Figure 1, once we have proven Sys_0, it serves as a concrete primitive and Sys'_0 as the abstract primitive. The abstract protocol is Sys_1 or, considered together with the abstract primitive, Sys^*. The concrete protocol is again Sys_1 or, together with the concrete primitive, $Sys^\#$. In Theorem 4.1, we show that the arrow "abstraction" in the middle of Figure 1 is correct. (Here "$\geq$" is the concrete version of "abstraction".) Thus we can continue the modular design with Sys^* as an abstract primitive for larger systems. Corollary 4.1 adds the right part of the figure for the case where the abstract goals are given by a specification Sys', i.e., "fulfils" is then also "$\geq$", and the abstract and concrete goals are the same.

We first have to define composition. We do it immediately for n systems $Sys_1, \ldots, Sys_n$. We do not provide a composition operator that takes the individual systems and produces one specific composition. The reason is that one typically does not want to compose every structure of one system with every structure of the others, but only with certain matching ones. E.g., if the individual machines of M_1 are implemented on the same physical devices as those of M_0, as usual with a layered distributed system, we might only want to compose structures corresponding to the same trust model. However, this is not the only conceivable situation. Hence we allow many different compositions.

Definition 4.1 (Composition) The composition of structures and of systems is defined as follows:

1. We call structures $(M_1, S_1), \ldots, (M_n, S_n)$ *composable* if $\mathsf{ports}(M_i) \cap \mathsf{ports}(M_j) = \emptyset$ for all $i \neq j$. We then define their composition as

$$(M_1, S_1)|| \ldots ||(M_n, S_n) := (M, S)$$

with $M = M_1 \cup \ldots \cup M_n$ and $S = (S_1 \cup \ldots \cup S_n) \cap \mathsf{free}(M)$. Clearly, (M, S) is again a structure.

2. We call a system Sys a composition of systems $Sys_1, \ldots, Sys_n$ and write

$$Sys \in Sys_1 \times \cdots \times Sys_n$$

if each structure $(M, S) \in Sys$ has a unique representation $(M, S) = (M_1, S_1)|| \ldots ||(M_n, S_n)$ with composable structures $(M_i, S_i) \in Sys_i$ for $i = 1, \ldots, n$.

3. Under the conditions of 2., we call (M_i, S_i) the restriction of (M, S) to Sys_i and write $(M_i, S_i) = (M, S)\lceil_{Sys_i}$.

◇

Remark 4.1. As compositions are again systems and structures, all further definitions (configurations, runs etc.) apply to them.

Remark 4.2. Restriction is defined relative to all n systems, i.e., uniqueness is only guaranteed if one knows them all. In most cases, however, it is unique even if one only knows (M, S) and Sys_i, e.g., if the n systems have disjoint sets of machines and each set M_i occurs in at most one structure of Sys_i. ◦

The following theorem shows that modular proofs as sketched in the introduction to this section are indeed possible. Recall that the situation is shown in the left and middle part of Figure 2. The main issue in formulating the theorem is to characterize $Sys^{\#}$, i.e., to formulate what it means that Sys_0 replaces Sys'_0.

Theorem 4.1 (Security of Two-System Composition) Let systems Sys_0, Sys'_0, Sys_1 and a valid mapping f_0 be given with

$$Sys_0 \geq^{f_0} Sys'_0.$$

Let compositions $Sys^{\#} \in Sys_0 \times Sys_1$ and $Sys^* \in Sys'_0 \times Sys_1$ be given that fulfil the following structural conditions:

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

   Let $f^{\#}$ denote the function that maps each $(M^{\#}, S^{\#})$ to the set of these compositions.

2. If $(M_1, S_1) \in Sys_1$ then $\mathsf{ports}(M_1) \cap \mathsf{forb}(M'_0, S'_0) = \emptyset$.

Then we have

$$Sys^{\#} \geq^{f^{\#}} Sys^*.$$

This holds for perfect, statistical and computational security, and also for the universal and blackbox definitions. □

Remark 4.3. Condition (2) implies that the machines in M_1 can be considered part of the user for any structure (M'_0, S'_0). We could weaken the condition by only comparing structures $(M_1, S_1) = (M^{\#}, S^{\#})\lceil_{Sys_1}$ with structures $(M'_0, S'_0) \in f_0((M^{\#}, S^{\#})\lceil_{Sys_0})$. The simpler but stronger condition is intuitively w.l.o.g.: The only ports that really need to have the same names in different systems are the specified ones. All others can be given a specific prefix for each system. ◦

Proof. (Of Theorem 4.1.) Let a configuration $conf^{\#} = (M^{\#}, S^{\#}, \mathsf{H}, \mathsf{A}^{\#}) \in \mathsf{Conf}^{f^{\#}}(Sys^{\#})$ be given and $(M_i, S_i) := (M^{\#}, S^{\#})\lceil_{Sys_i}$ for $i = 0, 1$. We have to show that there is an indistinguishable configuration $conf^* \in \mathsf{Conf}(Sys^*)$. The outline of the proof is as follows; it is illustrated in Figure 3.

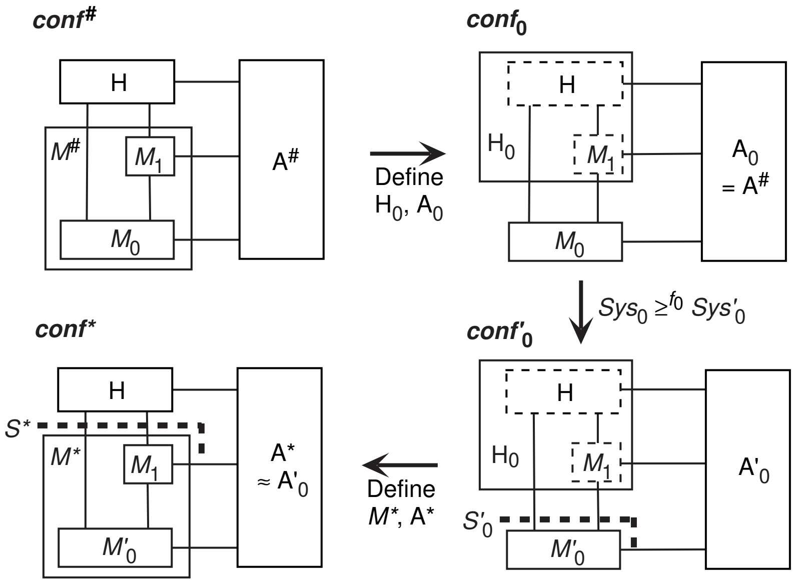

Figure 3: Configurations in the composition theorem. Dashed machines are internal submachines. (The connections drawn inside H_0 are not dashed because the combination is open.)

1. We combine H and M_1 into a user H_0 to obtain a configuration $conf_0 = (M_0, S_0, \mathsf{H}, \mathsf{A}_0) \in \mathsf{Conf}(Sys_0)$ where the view of H as a submachine of H_0 is the same as that in $conf^{\#}$.

2. We show that $conf_0 \in \mathsf{Conf}^{f_0}(Sys_0)$. Then by the precondition $Sys_0 \geq^{f_0} Sys'_0$, there is a configuration $conf'_0 = (M'_0, S'_0, \mathsf{H}_0, \mathsf{A}'_0) \in \mathsf{Conf}(Sys'_0)$ with $(M'_0, S'_0) \in f_0(M_0, S_0)$ where the view of H_0 is indistinguishable from that in $conf_0$.

3. We decompose H_0 into H and M_1 again and derive a configuration $conf^* = (M^*, S^*, \mathsf{H}, \mathsf{A}^*) \in \mathsf{Conf}(Sys^*)$ where the view of H equals that of H as a submachine of H_0 in $conf'_0$.

4. We conclude that $conf^*$ is an indistinguishable configuration for $conf^{\#}$.

We now present the four steps in detail.

Step 1: The precise definition of $conf_0 = (M_0, S_0, \mathsf{H}, \mathsf{A}_0)$ is that $(M_0, S_0) = (M^{\#}, S^{\#})\lceil_{Sys_0}$, that H_0 is the open combination of $M_1 \cup \{\mathsf{H}\}$ as in Lemma 2.2, and $\mathsf{A}_0 := \mathsf{A}^{\#}$. This is a valid configuration from $\mathsf{Conf}(Sys_0)$:

- $(M_0, S_0) = (M^{\#}, S^{\#})\lceil_{Sys_0}$ is a valid structure by the definition of a composition.
- Closed collection: The overall set of ports is the same as in $conf^{\#}$. Hence the machines still have pairwise disjoint port sets, and all ports still have complements in the set.

Hence Lemma 2.2 implies $view_{conf_0}(\mathsf{H}) = view_{conf^{\#}}(\mathsf{H})$.

Step 2: We now show that $conf_0 \in \mathsf{Conf}^{f_0}(Sys_0)$, i.e., H_0 has no ports from $\mathsf{forb}(M'_0, S'_0) = \mathsf{ports}(M'_0) \cup \bar{S}'^{c}_0$ for any structure $(M'_0, S'_0) \in f_0(M_0, S_0)$.

Assume that it had such a port p. By construction of H_0, p is also a port of M_1 or H. The first case is excluded in Precondition 2 of the theorem. Thus $p \in Ports_{\mathsf{H}}$. We use that $conf^{\#}$ is suitable, i.e., H has no ports from $\mathsf{forb}(M^*, S^*) = \mathsf{ports}(M^*) \cup \bar{S}^{*c}$ for any $(M^*, S^*) \in f^{\#}(M^{\#}, S^{\#})$. By Precondition (1) of the Theorem, $(M'_0, S'_0)||(M_1, S_1)$ exists and lies in $f^{\#}(M^{\#}, S^{\#})$, hence we use this as (M^*, S^*).

Then $M'_0 \subseteq M^*$ implies $p \notin \mathsf{ports}(M'_0)$. Thus only $p^c \in \bar{S}'_0$ remains, and we want to show that it contradicts $p^c \notin \bar{S}^*$. We first show $p^c \in \mathsf{free}(M^*)$: We have $p^c \in \bar{S}'_0 \subseteq \mathsf{ports}(M'_0)$, and thus $p \notin \mathsf{ports}(M'_0)$, and we have shown above that $p \notin \mathsf{ports}(M_1)$. Secondly, disjointness of the part names of M'_0 and M_1 implies $p^c \notin S_1$. Hence $p^c \in \mathsf{free}(M^*) \setminus (S'_0 \cup S_1) = \bar{S}^*$. This is the desired contradiction.

Hence $conf_0$ is indeed a suitable configuration. Thus $Sys_0 \geq^{f_0} Sys'_0$ implies that there is a configuration $conf'_0 = (M'_0, S'_0, \mathsf{H}_0, \mathsf{A}'_0) \in \mathsf{Conf}(Sys'_0)$ with $(M'_0, S'_0) \in f_0(M_0, S_0)$ and $view_{conf'_0}(\mathsf{H}_0) \approx view_{conf_0}(\mathsf{H}_0)$. This implies $view_{conf'_0}(\mathsf{H}) \approx view_{conf_0}(\mathsf{H})$ because the view of a submachine is a function of the larger view (Lemmas 2.1 and 2.2).

Step 3: We define $conf^* = (M^*, S^*, \mathsf{H}, \mathsf{A}^*)$ by reversing the combination of H and M_1 into H_0: The structure is $(M^*, S^*) := (M'_0, S'_0)||(M_1, S_1)$, the user the original H, and $\mathsf{A}^* := \mathsf{A}'_0$. We show that $conf^* \in \mathsf{Conf}(Sys^*)$.

- Structure: $(M^*, S^*) \in Sys^*$ follows immediately from Precondition 1 of the theorem.
- Closed collection: The ports of H and the machines in M_1 are disjoint because so they were in $conf^{\#}$, and those of all other pairs of machines because so they were in $conf'_0$.[4] As the set of ports is the same as in $conf'_0$, they also still all have complements in the set.

We can now see $conf'_0$ as derived from $conf^*$ by taking the open combination of $M_1 \cup \{\mathsf{H}\}$. Hence Lemma 2.2 applies, and we obtain $view_{conf^*}(\mathsf{H}) = view_{conf'_0}(\mathsf{H})$.

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

Universal and blackbox: For the universal case, note that $\mathsf{A}_0 = \mathsf{A}^{\#}$ does not depend on H. Then A'_0 only depends on (M_0, S_0) and A_0, and thus also $\mathsf{A}^* = \mathsf{A}'_0$. For the blackbox case, A'_0 additionally consists of a simulator Sim with $\mathsf{A}_0 = \mathsf{A}^{\#}$ as a blackbox, and thus so does A^*. ■

The following corollary finishes the formalization and proof of the situation shown in Figure 2: We now assume that there is also a specification Sys' for the system Sys^*, as shown in the left part of the figure.

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

Corollary 4.1 *Consider five systems satisfying the preconditions of Theorem 4.1, and a sixth one, Sys', with $Sys^* \geq^{f_1} Sys'$. Then*

$$Sys^{\#} \geq^{f} Sys'$$

where $f := f_1 \circ f^{\#}$ as in the transitivity lemma, except if f is not a valid mapping. □

The restriction that f must be a valid mapping is not serious (as for transitivity generally); it means that the naming conventions must be fulfilled for the two systems $Sys^{\#}$ and Sys' that we finally want to compare.

Proof. Theorem 4.1 implies that $Sys^{\#} \geq^{f^{\#}} Sys^*$. Then we immediately obtain $Sys^{\#} \geq^{f} Sys'$ using transitivity (Lemma 2.3). ■

Remark 4.4. An alternative to the corollary would be to consider *two* specifications Sys'_0 and Sys'_1 for the two real systems Sys_0 and Sys_1, and to show that $Sys^{\#} \in Sys_0 \times Sys_1$ is as secure as some $Sys' \in Sys'_0 \times Sys'_1$. For this, our composition theorem could be applied to internally structured systems Sys'. However, typically a specification should not prescribe that the implementation must have two subsystems; e.g., in specifying a payment system it should be irrelevant whether the implementation uses secret channels as a subsystem. Hence the overall specification Sys' will typically be monolithic as in Figure 2. ∘

5 Outlook

We have proven two important properties of a simulatability definition for reactive cryptographic systems, composability and preservation of integrity properties. There are many possible next steps. One is to apply the composition theorem to concrete systems, e.g., systems using secure channels based on the specification used in [24]. For actual tool support, the specifications should be translated from our I/O automata to a concrete formal language, i.e., a restricted syntax. Preservation of privacy properties is also desirable; a general class (besides simulatability) is harder to define, but we have sketches that simple properties like non-interference are preserved.

Acknowledgments

We thank *Matthias Schunter* for interesting discussions.

References

[1] M. Abadi, A. D. Gordon, *A Calculus for Cryptographic Protocols: The Spi Calculus*, 4th Conf. on Computer and Communications Security, ACM, 1997, 36–47

[2] M. Abadi, P. Rogaway, *Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption)*, IFIP International Conferences on Theoretical Computer Science (IFIP TCS2000), Sendai, Japan, August 2000 (to appear)

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

[4] M. Bellare, R. Canetti, H. Krawczyk, *A modular approach to the design and analysis of authentication and key exchange protocols*, 13th Symp. on Theory of Computing (STOC), ACM, 1998, 419–428

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

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

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

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

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

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

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

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

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

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

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

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

[17] P. Lincoln, J. Mitchell, M. Mitchell, A. Scedrov, *Probabilistic Polynomial-Time Equivalence and Security Analysis*, Formal Methods 1999, available at ftp://theory.stanford.edu/pub/jcm/papers/fm-99.ps

[18] G. Lowe, *Breaking and fixing the Needham-Schroeder public-key protocol using FDR*, Tools and Algorithms for the Construction and Analysis of Systems, LNCS 1055, Springer-Verlag, 1996, 147–166

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

[20] C. Meadows, *Using Narrowing in the Analysis of Key Management Protocols*, Symp. on Security and Privacy, IEEE, 1989, 138–147

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

[22] J. K. Millen, *The Interrogator: A Tool for Cryptographic Protocol Security*, Symp. on Security and Privacy, IEEE, 1984, 134–141

[23] B. Pfitzmann, *Sorting Out Signature Schemes*, 1st Conf. on Computer and Communications Security, ACM, 1993, 74–85

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

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

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

[27] A. W. Roscoe, *Modelling and Verifying Key-Exchange Protocols Using CSP and FDR*, 8th Computer Security Foundations Workshop, IEEE, 1995, 98–107

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

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