Project IST-1999-11583

Malicious- and Accidental-Fault Tolerance for Internet Applications

Cryptographic Semantics for the Algebraic Models

André Adelsbach and Michael Steiner (Editors)
Universität des Saarlandes (D)

MAFTIA deliverable D8

Public document

March 5, 2002

Editors

André Adelsbach ..... *Universität des Saarlandes (D)*
Michael Steiner ..... *Universität des Saarlandes (D)*

Contributors

André Adelsbach ..... *Universität des Saarlandes (D)*
Sadie Creese ..... *QinetiQ, Malvern (UK)*
Birgit Pfitzmann ..... *IBM Zurich Research Lab (CH)*
Peter Ryan ..... *University of Newcastle (UK)*
William Simmonds ..... *QinetiQ, Malvern (UK)*
Sandra Steinbrecher ..... *Universität des Saarlandes (D)*
Michael Steiner ..... *Universität des Saarlandes (D)*
Christian Stüble ..... *Universität des Saarlandes (D)*
Michael Waidner ..... *IBM Zurich Research Lab (CH)*

Abstract

MAFTIA's Work-package 6 is pursuing the overall goal of

> "rigorously defining the basic concepts developed by MAFTIA, and verifying results of the work on dependable middle-ware."

In the former MAFTIA deliverable D4, we presented a general rigorous model for the security of reactive systems. This model comprised various types of faults (attacks) and topology as considered in MAFTIA, but was restricted to a synchronous timing model. In this deliverable, we focus on a model-variant for asynchronous reactive systems. This variant is highly important for MAFTIA, since several of the major MAFTIA middle-ware-protocols are asynchronous. To illustrate the use of the asynchronous model a proof of secure message transmission in the asynchronous case is included. We chose this example which delivers a similar service as the example from D4, to illustrate the analogies as well as the differences between the two variants of the secure reactive systems model. As in the synchronous model, we prove a composition theorem for its asynchronous counterpart, which allows modular proofs in this model. Furthermore, we discuss how to model adaptive corruptions in the presented models.

Finally, we discuss the relation between the proposed models and the real world: Every model abstracts in certain ways from the real world and makes assumptions. So do the presented models of secure reactive systems. These abstractions are, on the one hand, necessary to enable reasoning about protocols at all. On the other hand, they can lead to insecure systems in the real world if they are naively implemented. Guided by the goal of secure real world systems, we present an assessment of the model's abstractions and discuss the possible impact on the real world security when implementing reactive systems which are proven secure in this model.

Contents

1 Introduction

The MAFTIA project systematically investigates the tolerance paradigm for building dependable distributed systems. For this, it combines techniques and notions from fault tolerance and various areas of security, such as intrusion detection and cryptographic protocols.

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

Both the cryptography and the formal-methods communities are working on such proofs. The former aims at proofs which rigorously deal with issues such as computational power and success probabilities of adversaries as well as complexity-theoretic assumptions underlying most efficient cryptographic protocols, while the latter aims at proofs in some formal proof system that can be automatically verified or even generated. Unfortunately, both approaches have their limitations. On the one hand, current formal methods in security cannot be applied directly to cryptographic proofs. Instead, they abstract from most cryptographic details, typically following Dolev and Yao's approach [20], and there is no guarantee that a formally proven protocol is actually secure if implemented with a cryptographically secure primitive [3, 43]. On the other hand, cryptographic definitions of complex systems are often sketchy, and even more the proofs, e.g., because every single definition currently has to reconsider active attacks and every single proof has to be a reduction proof to underlying assumptions.

The goal of the Work-package 6 in the MAFTIA project is to enable the verification and assessment of the dependability achieved by protocols and mechanisms developed in the other work-packages. To meet this goal, we developed rigorously defined models that cover the basic concepts identified

in Work-package 1 and try to link the approaches from cryptography and formal-methods known so far to get the best overall results: proofs that allow abstraction and the use of formal methods, but retain a sound cryptographic semantics. Thus we provide a model that allows us to split reactive systems into two layers: The lower layer is a cryptographic system whose security can be rigorously proven using standard cryptographic arguments. To the upper layer it provides an abstract (and typically deterministic) service that hides all cryptographic details. Relative to this abstract service one can verify the upper layer using existing formal methods. Since our model allows secure composition (as shown in Section 2.5, Theorem 2.1) one can conclude that the overall system is secure if the formally verified upper layer is put on top of a cryptographically verified lower layer ([43] provides more motivation for this approach).

Since the underlying synchrony assumptions have major impact on the model's definitions, we developed two model variants. While MAFTIA deliverable D4 [4] concentrates, among other things, on the model variant with strong synchrony assumptions, Chapter 2 of the present deliverable introduces the asynchronous version of the model. It makes only weak synchrony assumptions and allows proofs for protocols with according assumptions. Both models are highly important to the assessment and verification of MAFTIA's basic protocols. Furthermore, we give a model of adaptive adversaries which is a strictly stronger adversary model than that discussed in D4.

In Chapter 3 we illustrate the model and its differences to the synchronous counterpart by presenting a rigorous security proof of a system for secure message transmission which is also one of the most basic building blocks of the MAFTIA middle-ware.

We close this deliverable in Chapter 4 with a discussion of the abstractions from the real world made in the presented models. Especially, the abstractions' impact on real world security of implementations in standard runtime environments is assessed. This assessment is an important aspect, since it calls attention to real world pitfalls which have to be avoided in order to reach the overall goal of MAFTIA: secure, fault tolerant real world systems.

2 Asynchronous Reactive Systems

In this Chapter, we carry out our approach of linking the cryptographic and formal-methods approaches for proving security, specifically for asynchronous reactive systems. Reactive means that the system interacts with its users many times (e.g., multiple sub-protocol executions).

2.1 Overview of the Asynchronous Model

Essentially, we model a system by sets of asynchronously communicating probabilistic state machines, connected via buffered channels. This general system model is presented in Section 2.3.1 and contains, in MAFTIA terms, the synchrony and topological model.

Honest users and the adversary are explicitly represented by two arbitrary machines, H and A, which can interact arbitrarily. A reactive system, Sys_1, is considered *at least as secure as* another system, Sys_2, written as $Sys_1 \geq Sys_2$, if whatever any adversary A_1 can do to any honest user H in Sys_1, some adversary A_2 can do to the same H in Sys_2 essentially with the same probability. System Sys_1 is often a real system using concrete cryptographic primitives, while Sys_2 is an ideal system, i.e., a specification, that does not depend on any specific cryptographic implementation details and is not realistic (e.g., one machine instead of a distributed system) but secure by construction. The resulting generality of the security definition is a highly desirable property of this security model. This is in particular important in the context of MAFTIA since the considered protocols and applications use a wide range of cryptographic primitives and have complex and manifold security goals.

The model is defined in Section 2.3. Section 2.4 shows how to represent typical trust models (or adversary structures), such as static threshold models and adaptive adversaries, with secure, authenticated and insecure channels.

Given the complexity and the size of the systems considered in MAFTIA, it is a necessity that the specification, design and security proofs can be performed in a modular manner. To support this modularity, we state and prove in Section 2.5 a composition theorem. (Note that the possibility of a secure composition was often implicitly but wrongly assumed in past approaches of cryptographic modeling. Here, in contrast, we handle composition explicitly

and put it on firm grounds.)

For concrete specifications, we advocate two main design principles:

1. Abstract interfaces: The ideal system should provide abstract interfaces, hiding all cryptographic details. This keeps the specification independent of the implementation, which is desirable when higher-layer protocols are based on the service. For instance, in order to send messages secretly from one user to another there is no need to ask the user to input cryptographic keys or to output ciphertexts to him; those can be generated, exchanged and processed completely within the system. In an implementation-independent simulatability-based definition, it is problematic to have keys in the interface because keys of different implementations are often distinguishable.

2. Explicit tolerable imperfections: In order to improve efficiency one often accepts certain imperfections. For instance, a typical practical implementation of secure message transmission only conceals the contents of messages, but does not hide who communicates with whom, which would be much more costly to implement. In a simulatability-based approach one has to include all such tolerable imperfections in the specification, i.e., in the ideal system. This must be done in a similar abstract way as the rest of the specification.

In Chapter 3, we apply the model to secure message transmission. We chose this example since it is an important type of cryptographic subsystem for larger MAFTIA systems: Many applications use encryption and signatures algorithms just for this purpose, and our definition of secure message transmission gives a much cleaner interface to these applications than letting them use the cryptographic primitives directly, potentially in insecure ways. Another reason for choosing this example is that it is similar to the example discussed in deliverable D4 [4] for the synchronous model and thus it illustrates the differences between the model variants. Additional to this illustrating example, a (more complex) multi-party key establishment service, a further potential basic MAFTIA protocol, has been defined in this model [46]. The corresponding protocols and related security proofs give also the first example on how to handle adaptive corruption.

For the proof of the real system, we first prove a theorem that extends the security of public-key encryption in multi-user settings to reactive cases, where secret messages may also be decrypted by correct machines. This generalizes a result from [10] and may be of independent interest. It captures

what is often called a "standard hybrid argument" in cryptography, but was not well-defined for the reactive case yet.

Before we go into the details of the model, we give a short overview over the related literature and previous work in this context.

2.2 Related Literature

Several researchers pursue the goal of providing security proofs that allow the use of formal methods, but retain a sound cryptographic semantics: In [37, 38] the cryptographic security of specific systems is directly defined and verified using a formal language (π-calculus), but without providing abstractions (their specifications essentially comprise the actual protocols including all cryptographic details) and without tool support (as even the specifications involve ad-hoc notations, e.g., for generating random primes). [40] has quite a similar motivation to our paper. However, cryptographic systems are restricted to the usual equational specifications (following [20]) and the semantics is not probabilistic. Hence, the abstraction from cryptography is no more faithful than in other papers on formal methods in security. Moreover, only passive adversaries are considered and only one class of users ("environment"). The author actually remarks that the model of what the adversary learns from the environment is not yet general, and that general theorems for the abstraction from probabilism would be useful. Our model solves both these problems. In [3] it is shown that a slight variation of the standard Dolev-Yao abstraction [20] is cryptographically faithful specifically for symmetric encryption, but only under passive attacks.

Our security definitions follow the simulatability approach of modern cryptography. This was first used in secure function evaluation [9, 25, 41, 54], and subsequently also for specific reactive problems (e.g., [11, 19, 21]) and for the construction of generic solutions for large classes of reactive problems [24, 23, 29] (usually yielding inefficient solutions and assuming that all parties take part in all sub-protocols). Somewhat general models for reactive systems have been proposed (after some earlier sketches, in particular in [24, 47, 17]) in [37, 38, 29, 43, 48]. The last three are synchronous, while the first two are in a somewhat simplified timing model with uniform choice among certain classes of unrelated possible events. Among the reactive models, the only composition theorem so far is ours in [48], i.e., we present the

first asynchronous one in the current paper. Our model is based on [48] except for the timing aspects. Those can be seen as extensions of [53, 16, 38]; differences are motivated in Section 2.3.

Independently and concurrently to this work, Ran Canetti has developed a model that roughly corresponds to standard cryptographic systems as discussed in Section 2.4.2, with polynomial-time users and adversaries, and authenticated channels only [18]. The model is less rigorously defined than the model presented here. Security is defined in terms of universal simulatability only (see Definition 2.16), which allows to securely compose a polynomial number of identical systems.

Several specifications for secure message transmissions as examples of general models have been proposed. The specification in [37] is formal but specific for one concrete protocol and comprises all cryptographic details, i.e., it is not abstract and its intuitive correctness is difficult to judge. Our concrete specification is quite close to that in [40], but we had to introduce the tolerable imperfections. Actually, the implementation in [40] has the same imperfections. They do not show up in the proof because the definition of "a system implements another one," used in place of our "as secure as" is weaker: Secrecy is defined as a property that the adversary cannot learn certain messages, but here the information leaked is not entire messages. [2, 1] stress the importance of a secure message transmission (channel) abstraction, but their focus is different from ours: They consider formal calculi that immediately contain secure channels as a primitive, thus the specification comes in indirectly in the semantics. Canetti's parallel work also contains a similar example; however, he assumes underlying authentic channels already, which simplifies both the specification (fewer tolerable imperfections) and the implementation. His technique of proving a system for only one recipient and applying the composition theorem does not apply in our case because we use common signature keys in all executions.

2.3 Asynchronous Reactive Systems

In this section, we present our model for secure reactive systems in an asynchronous network. Subsection 2.3.1 defines the general system model, i.e., machines and executions of collections of machines. Subsection 2.3.2 defines the specific system model, i.e., systems with users and adversaries.

Subsection 2.3.3 defines simulatability, i.e., our basic notion of security. Finally, Subsection 2.3.4 presents important lemmas for this model.

In the first two subsections, the primary differences to the synchronous model as presented in the MAFTIA deliverable D4 [4] will become evident. Naturally, these differences have also some strong implications on the following sections. However, note that despite these differences the two models are closely related and it can be shown that the synchronous model is a special case of the asynchronous model [8].

Our machine model is probabilistic state-transition machines, similar to probabilistic I/O automata as sketched in [39] (more details in [53]). A distinguishing feature in our model of asynchronous executions is distributed scheduling. The standard way to fix the order of events in an asynchronous system of probabilistic I/O automata is a probabilistic scheduler that has full information about the system [53]. The "standard" understanding in cryptology (closest to a rigorous definition in [16]) is that the adversary schedules everything, but only with realistic information. This corresponds to making a certain subclass of schedulers explicit for the model from [53]. However, if one splits a machine into local sub-machines, or defines intermediate systems for the purposes of proof only (as we do in Chapter 3), this may introduce many schedules that do not correspond to a schedule of the original system and, therefore, just complicate the proofs. Our solution is a distributed definition of scheduling which allows machines that have been scheduled to schedule certain (statically fixed) other machines themselves. This does not weaken the adversary's power in real systems, because our definition of standard cryptographic systems in Section 2.4 will not use this feature except for scheduling local sub-machines.

Similar problems with purely adversarial scheduling were already noted in [38]. They distinguish secure channels and schedule all those with uniform probability before adversary-chosen events. However, that introduces a certain amount of global synchrony. Furthermore, we do not require "local" scheduling for all secure channels; they may be blindly scheduled by the adversary (i.e., without even seeing if there are messages on the channel). For instance, this models cases where the adversary has a global influence on relative network speed.

2.3.1 General System Model

Let a finite alphabet Σ be given, let Σ^* denote the strings over it, ϵ the empty string, and $\Sigma^+ := \Sigma^* \setminus \{\epsilon\}$. We assume that $!, ?, ^{\leftrightarrow}, ^{\triangleleft} \notin \Sigma$. For $s \in \Sigma^*$ and $l \in \mathbb{N}_0$, let $s\lceil_l$ denote the l-bit prefix of s.

Definition 2.1 *(Ports)*

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

b) We call a port (n, l, d) an in-port *or* out-port *iff $d = ?$ or $d = !$, respectively. We call it a* simple port*,* buffer port *or* clock port *iff $l = \epsilon, {}^{\leftrightarrow}$, or ${}^{\triangleleft}$, respectively. For a set P of ports let $\mathsf{out}(P) := \{p \in P \mid \mathsf{dir}(p) = !\}$ and $\mathsf{in}(P) := \{p \in P \mid \mathsf{dir}(p) = ?\}$. We use the same notation for sequences of ports, retaining the order.*

*c) By p^c, the (*low-level*)* complement *of a port p, we denote the port with which it connects according to Figure 2.1, i.e., $\mathsf{n}^{\triangleleft}!^c := \mathsf{n}^{\triangleleft}?$, $\mathsf{n}!^c := \mathsf{n}^{\leftrightarrow}?$, $\mathsf{n}^{\leftrightarrow}!^c := \mathsf{n}?$, and vice versa. Accordingly we define the (low-level) complement of a set or sequence of ports.*

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

$\diamond$

Our machine model is an automaton model with Turing machines as a possible implementation. To allow time bounds independent of the environment, we provide bounds on the length of the considered inputs. We are not using Turing machines as the primary model, in contrast to related cryptographic literature, to support abstraction in the specifications.

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

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

of a name $name_{\mathsf{M}} \in \Sigma^+$, a finite sequence $Ports_{\mathsf{M}}$ of ports, a set $States_{\mathsf{M}} \subseteq \Sigma^*$ of states, a probabilistic state-transition function δ_{M}, a length function l_{M} :

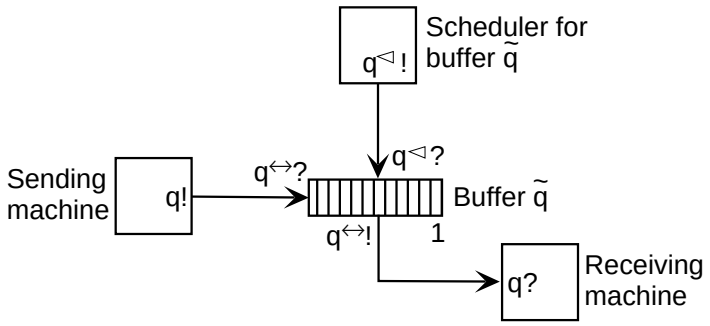

Figure 2.1: Ports and buffers.

$States_{\mathsf{M}} \to (\mathbb{N} \cup \{\infty\})^{|\mathsf{in}(Ports_{\mathsf{M}})|}$, and sets $Ini_{\mathsf{M}}, Fin_{\mathsf{M}} \subseteq States_{\mathsf{M}}$ of initial and final states. Its input set is $\mathcal{I}_{\mathsf{M}} := (\Sigma^*)^{|\mathsf{in}(Ports_{\mathsf{M}})|}$; the i-th element of an input tuple denotes the input at the i-th in-port. Its output set is $\mathcal{O}_{\mathsf{M}} := (\Sigma^*)^{|\mathsf{out}(Ports_{\mathsf{M}})|}$. The empty word, ϵ, denotes no in- or output at a port. δ_{M} maps each pair $(s, I) \in States_{\mathsf{M}} \times \mathcal{I}_{\mathsf{M}}$ to a finite distribution over $States_{\mathsf{M}} \times \mathcal{O}_{\mathsf{M}}$. If $s \in Fin_{\mathsf{M}}$ or $I = (\epsilon, \ldots, \epsilon)$, then $\delta_{\mathsf{M}}(s, I) = (s, (\epsilon, \ldots, \epsilon))$ deterministically. Inputs are ignored beyond the length bounds, i.e., $\delta_{\mathsf{M}}(s, I) = \delta_{\mathsf{M}}(s, I\lceil_{l_{\mathsf{M}}(s)})$ for all $I \in \mathcal{I}_{\mathsf{M}}$, where $(I\lceil_{l_{\mathsf{M}}(s)})_i := I_i\lceil_{l_{\mathsf{M}}(s)_i}$ for all i. ◇

In the text, we often write "M" also for $name_{\mathsf{M}}$.

Remark 2.1. The chosen representation makes δ_{M} independent of the port names. This will also hold for views. Hence we can rename ports in some proofs without changing the views. The requirement for ϵ-inputs means that it does not matter if we switch a machine without inputs or not; we will also omit such steps from the runs. This simplifies combinations below. Inputs "masked" by a length bound 0 are treated in the same way. ○

For computational aspects, a machine M is regarded as implemented by a probabilistic interactive Turing machine as introduced in [27]. We need some refinements of the model.

Definition 2.3 *(Computational Realization) A* probabilistic interactive Turing machine T *is a probabilistic multi-tape Turing machine whose heads see if the head of a partner machine is on the same cell of a common tape. Tapes have a left boundary, and heads start on the left-most cell.* T implements *a machine* M *as in Definition 2.2 if the following holds. Let $i_{\mathsf{M}} := |\mathsf{in}(Ports_{\mathsf{M}})|$.*

We write "finite state" for a state of the finite control of T *and "state" for an element of* $States_{\mathsf{M}}$.

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

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

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

d) The complexity of T *is, unless stated otherwise, measured in terms of the length of its initial state, i.e., the initial worktape content (often a security parameter). In particular, polynomial-time is meant in this sense. A machine is called weakly polynomial-time if its run-time is polynomial in the overall length of its inputs.*

$\diamond$

Definition 2.4 *(Simple Machines and Master Schedulers) A machine* M *is* simple *if it has only simple ports and clock out-ports. A machine* M *is a* master scheduler *if it has only simple ports and clock out-ports and the special* master-clock *in-port* $\mathsf{clk}^{\triangleleft}?$*. Without loss of generality, a master scheduler makes not outputs in a transition that enters the final state.* $\diamond$

Definition 2.5 *(Buffers) For each name* $\mathsf{q} \in \Sigma^+$ *we define a specific machine* $\widetilde{\mathsf{q}}$*, called a* buffer: *It has three ports,* $\mathsf{q}^{\triangleleft}?$*,* $\mathsf{q}^{\leftrightarrow}?$*,* $\mathsf{q}^{\leftrightarrow}!$ *(clock, in, and out) (see Figure 2.1). Its internal state is a queue over* Σ^+ *with random access,*

initially empty. Its set of final states is empty, and all its length bounds are infinite. For each state transition, if the input x at $\mathsf{q}^{\leftrightarrow}?$ is non-empty, then $\delta_{\widetilde{\mathsf{q}}}$ appends x to the queue. A non-empty input at $\mathsf{q}^{\triangleleft}?$ is interpreted as a number $i \in \mathbb{N}$ and the i-th element is retrieved (where 1 indicates the oldest one), removed from the queue, and output at $\mathsf{q}^{\leftrightarrow}!$. (This might be the element just appended.) If there are less than i elements, the output is ϵ. $\diamond$

Buffers are weakly polynomial-time.

Definition 2.6 *(Collections)*

a) For every machine M*, let* $\mathsf{ports}(\mathsf{M})$ *denote the set of ports in* $Ports_{\mathsf{M}}$*, and for a set* $\hat{M}$ *of machines, let* $\mathsf{ports}(\hat{M}) := \bigcup_{\mathsf{M} \in \hat{M}} Ports_{\mathsf{M}}$*.*

b) A collection $\hat{C}$ *is a finite set of machines with pairwise different machine names, disjoint sets of ports, and where all machines are simple, master schedulers, or buffers. It is called* (weakly) polynomial-time *if all its non-buffer machines have a (weakly) polynomial-time implementation.*

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

d) A collection is closed *if* $\mathsf{free}(\hat{C}) = \{\mathsf{clk}^{\triangleleft}?\}$*. (Hence there is exactly one master scheduler, identified by having the port* $\mathsf{clk}^{\triangleleft}?$*.)*

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

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

$\diamond$

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

Definition 2.7 *(Runs) Given a closed collection $\hat{C}$ with master scheduler* X *and a tuple ini* $\in Ini_{\hat{C}} := \times_{\mathsf{M}\in\hat{C}} Ini_{\mathsf{M}}$ *of initial states, the probability space of* runs *is defined inductively by the following algorithm. It has a variable r for the resulting run, an initially empty list, a variable* $\mathsf{M_{CS}}$ *("current scheduler") over machine names, initially* $\mathsf{M_{CS}} := \mathsf{X}$*, and treats each port as a variable over* Σ^**, initialized with* ϵ *except for* $\mathsf{clk}^{\triangleleft}? := 1$*. Probabilistic choices only occur in Phase (1).*

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

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

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

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

5. Scheduled message: *Switch* $\widetilde{\mathsf{q}}$ *with input* $\mathsf{q}^{\triangleleft}? := \mathsf{q}^{\triangleleft}!$*, set* $\mathsf{q}? := \mathsf{q}^{\leftrightarrow}!$ *and then assign* ϵ *to all ports of* $\widetilde{\mathsf{q}}$ *and to* $\mathsf{q}^{\triangleleft}!$*. Let* $\mathsf{M_{CS}} := \mathsf{M}'$ *for the unique machine* M' *with* $\mathsf{q}? \in \mathsf{ports}(\mathsf{M}')$*. Go back to Phase (1).*

Whenever a machine (this may be a buffer) with name $name_{\mathsf{M}}$ *is switched from* (s, I) *to* (s', O)*, we add a* step $(name_{\mathsf{M}}, s, I', s', O)$ *to the run r where* $I' := I\lceil_{l_{\mathsf{M}}(s)}$*, except if s is final or* $I' = (\epsilon, \ldots, \epsilon)$*. This gives a family of random variables*

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

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

The abstract assignments of the run algorithm have a natural realization with the Turing machines from Definition 2.3, where low-level connections

are realized by sharing a tape, except that all clock-ports are connected to an additional Turing machines for the cleanup.

Definition 2.8 *(Views) The* view *of a subset $\hat{M}$ of a closed collection $\hat{C}$ in a run r is the restriction of r to $\hat{M}$, i.e., the subsequence of all steps $(name, s, I, s', O)$ where $name$ is the name of a machine $\mathsf{M} \in \hat{M}$. This gives a family of random variables*

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

and similarly for l-step prefixes. ◇

2.3.2 Security-specific System Model

Now we define specific collections for security purposes, first the system part and then the environment, i.e., users and adversaries. Typically, a cryptographic system is described by an intended structure, and the actual structures are derived using a trust model, see Section 2.4. However, as a wide range of trust models is possible, we keep the simulatability definition independent of them.

Definition 2.9 *(Structures and Systems)*

a) A structure *is a pair $struc = (\hat{M}, S)$ where $\hat{M}$ is a collection of simple machines called* correct machines*, and $S \subseteq \mathsf{free}([\hat{M}])$ is called* specified ports*. If $\hat{M}$ is clear from the context, let $\bar{S} := \mathsf{free}([\hat{M}]) \setminus S$. We call $\mathsf{forb}(\hat{M}, S) := \mathsf{ports}(\hat{M}) \cup \bar{S}^c$ the* forbidden ports*.*

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

◇

The separation of the free ports into specified ports and others is an important feature of our particular simulatability definitions. The specified ports are those where a certain service is guaranteed. Typical examples of inputs at specified ports are "send message m to id" for a message transmission system or "pay amount x to id" for a payment system. The ports in $\bar{S}$ are

additionally available for the adversary. The ports in $\mathsf{forb}(\hat{M}, S)$ will therefore be forbidden for an honest user to have. In the simulatability definition below, only the events at specified ports have to be simulated one by one. This allows *abstract* specifications of systems with *tolerable imperfections*. This will become clear with the example in Chapter 3.

Definition 2.10 *(Configurations)*

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

b) The initial states of all machines in a configuration are a common security parameter k *in unary representation, denoted by* $\mathsf{1}^k$*. This means that we consider the families of runs and views of the collection* $\hat{C}$ *restricted to the subset* $Ini'_{\hat{C}} := \{(\mathsf{1}^k)_{\mathsf{M} \in \hat{C}} | k \in \mathbb{N}\}$ *of* $Ini_{\hat{C}}$*. We write* run_{conf} *and* $view_{conf}(\hat{M})$ *for the families* $run_{\hat{C}}$ *and* $view_{\hat{C}}(\hat{M})$ *restricted to* $Ini'_{\hat{C}}$*, and similar for* l*-step prefixes. Furthermore, we identify* $Ini'_{\hat{C}}$ *with* $\mathbb{N}$ *and thus write* $run_{conf,k}$ *etc. for the individual random variables.*

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

$\diamond$

2.3.3 Simulatability

We now define the security of a system Sys_1 relative to another system Sys_2. Typically, we only want to compare each structure of Sys_1 with certain corresponding structures in Sys_2. What "corresponding" means can be specified by a mapping f; but we require that only structures with the same set of specified ports correspond.

Definition 2.11 *(Valid Mappings, Suitable Configurations) Let* Sys_1 *and* Sys_2 *be two systems.*

a) *A* valid mapping *for them is a function* $f : Sys_1 \to \mathcal{P}(Sys_2)$ *with* $S_1 = S_2$ *for all structures* $(\hat{M}_1, S_1)$ *and* $(\hat{M}_2, S_2) \in f(\hat{M}_1, S_1)$.

b) *If* Sys_2 *contains exactly one structure* $(\hat{M}_2, S_2)$ *with* $S_2 = S_1$ *for each* $(\hat{M}_1, S_1) \in Sys_1$, *the* canonical mapping f *is defined by* $f(\hat{M}_1, S_1) = \{(\hat{M}_2, S_2)\}$.

c) *Given* f, *the set* $\mathsf{Conf}^f(Sys_1)$ *of* suitable configurations *contains those configurations* $(\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}(Sys_1)$ *where* $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(\hat{M}_2, S) = \emptyset$ *for all* $(\hat{M}_2, S) \in f(\hat{M}_1, S)$.

◇

Remark 2.2. In the synchronous model in [48], we allow more general users and valid mappings. The stronger requirements here simplify the presentation and are sufficient for all cryptographic examples we considered. See [44] for non-cryptographic examples with $S_1 \neq S_2$. ∘

An example of a system where different structures have the same specified ports, and thus a non-canonical mapping is needed, is a protocol with a semi-trusted third-party machine T which needs no user interface (because its behavior is fully prescribed by the protocol) and where different properties can be achieved depending on whether T is correct or not.

Simulatability is based on indistinguishability of views, hence we repeat the definition of indistinguishability, essentially from [55].

Definition 2.12 *(Negligible Functions) A function* $g : \mathbb{N} \to \mathbb{R}_{\geq 0}$ *is* negligible, *written* $g(k) \leq 1/\mathsf{poly}(k)$, *if for all positive polynomials* Q, $\exists k_0 \forall k \geq k_0 : g(k) \leq 1/Q(k)$. *The class of negligible functions is written NEGL.* ◇

Definition 2.13 *(Indistinguishability) Two families* $(\mathsf{var}_k)_{k\in\mathbb{N}}$ *and* $(\mathsf{var}'_k)_{k\in\mathbb{N}}$ *of random variables (or probability distributions) on common domains* D_k *are*

a) perfectly indistinguishable *("=") if for each* k, *the two distributions* var_k *and* var'_k *are identical;*

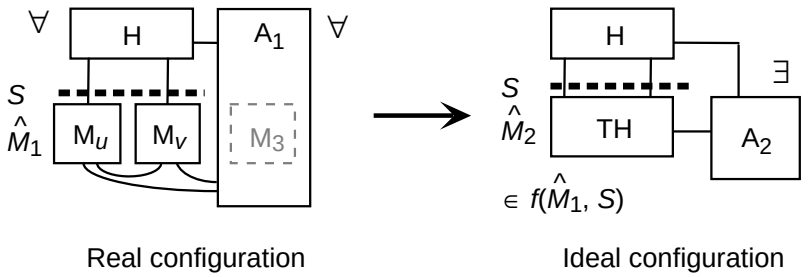

Figure 2.2: Example of simulatability. The view of H is compared.

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 every 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.

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

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

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

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

Now we present the simulatability definition. It captures that whatever an adversary can achieve in the real system against certain honest users, another adversary can achieve against the same honest users in the ideal system. A typical situation is illustrated in Figure 2.2.

Definition 2.14 *(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*) if for every configuration $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^f(Sys_1)$, there exists a configuration $conf_2 = (\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(Sys_2)$ with $(\hat{M}_2, S) \in f(\hat{M}_1, S)$ (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 $view_{conf_1,l}(\mathsf{H}) \approx_{SMALL} view_{conf_2,l}(\mathsf{H})$ for all polynomials l, i.e., statistical indistinguishability of all families of l-step prefixes of the views.*

*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.* ◇

Remark 2.3. Adding a free adversary out-port in the comparison (like the guessing-outputs used to define semantic security [26]) does not make the definition stricter: Any such out-port can be connected to an in-port added to the honest user with sufficiently large length bounds. H does not react on this input, but nevertheless it is included in the view of H, i.e., in the comparison. (In more detail, this can be proved similar to the synchronous case in [44].) ◦

2.3.4 Lemmas and Stronger Simulatability

The main results in this section concern the notion of combining several machines into one, and transitivity of the relations "as secure as". The former is an essential ingredient in the composition and is used to define blackbox simulatability. The remaining lemmas are auxiliary results.

Lemma 2.1 (Properties of Runs and Views) *Let $\hat{C}$ be a closed collection.*

a) Whenever a machine M is switched in a run of $\hat{C}$, there is at most one port $p \in \mathsf{ports}(\hat{C})$ with $p \neq \epsilon$. If it exists, $p \in \mathsf{ports}(\mathsf{M})$.

b) Views of polynomial-time machines are always of polynomial size. If $\hat{C}$ is polynomial-time, the runs are of polynomial size.

□

Proof. Part a) is obviously true before the first iteration, and can easily be seen inductively by following one iteration step by step. Part b) holds because a polynomial-time machine can only make a polynomial number of steps and build up polynomial-size states and in- and outputs. A run of $\hat{C}$ consists of the steps of its polynomial-time machines and the buffers, whose number of steps and queue size are bounded by the inputs received. ■

In the following, by *new name* we always mean one that does not occur in the systems and configurations already under consideration. We can always assume without loss of generality that a new name exists. (Otherwise we can, e.g., extend the alphabet.)

Definition 2.15 *(Combination of Machines) Let $\hat{D}$ be a collection without buffers. For a new name n_D, we define the* combination *of $\hat{D}$ into one machine D with this name, written $\mathsf{comb}(\hat{D})$ in slight abuse of notation.*

a) Its ports are $Ports_\mathsf{D} := \mathsf{ports}(\hat{D})$. (Their order would be an additional parameter of comb, but it never matters in the following.)

b) Its states are $States_\mathsf{D} := \times_{\mathsf{M} \in \hat{D}} States_\mathsf{M}$.

c) Its transition function δ_D is defined by applying the transition function of each submachine to the corresponding sub-states and inputs, unless D has reached a final state (see below). In that case, δ_D does not change the state and produces no output.

d) Its length function l_D is defined by applying the length function of the corresponding submachine for each input.

e) Its initial states are $Ini_{\mathsf{D}} := \times_{\mathsf{M}\in\hat{D}} Ini_{\mathsf{M}}$. *For every* $k \in \mathbb{N}$, *we identify the state* $(1^k)_{\mathsf{M}\in\hat{D}}$ *with* 1^k *(for the conventions in configurations).*

f) If there is a master scheduler $\mathsf{X} \in \hat{D}$, *then* Fin_{D} *is the set of all states of* D *where* X *is in a state from* Fin_{X}. *Otherwise* D *stops as soon as* all *sub-machines have stopped:* $Fin_{\mathsf{D}} := \times_{\mathsf{M}\in\hat{D}} Fin_{\mathsf{M}}$.

$\diamond$

Lemma 2.2 (Combination) *Let* $\hat{C}$ *be a collection without buffers,* $\hat{D} \subseteq \hat{C}$, *and* $\mathsf{D} := \mathsf{comb}(\hat{D})$ *with a name that is new in* $\hat{C}$. *Let* $\hat{C}^* := (\hat{C} \setminus \hat{D}) \cup \{\mathsf{D}\}$.

a) D *is well-defined.*

b) If $[\hat{C}]$ *is a closed collection, then so is* $[\hat{C}^*]$.

c) The view of any set of original machines in $[\hat{C}^*]$ *is the same as in* $[\hat{C}]$. *This includes the views of the sub-machines in* D, *which are well-defined functions of the view of* D.

d) Combination is associative: If $\hat{D} = \hat{D}_1 \cup \hat{D}_2$ *and* $\mathsf{D}_1 := \mathsf{comb}(\hat{D}_1)$, *then* $\mathsf{comb}(\{\mathsf{D}_1\} \cup \hat{D}_2) = \mathsf{D}$, *if one identifies Cartesian products that differ only in the bracket structure.*

e) If all machines in $\hat{D}$ *are polynomial-time, then so is* D.

f) If $\hat{D} = \hat{D}_{\mathsf{p}} \cup \hat{D}_{\mathsf{wp}}$, *where all machines in* $\hat{D}_{\mathsf{p}}$ *are polynomial-time and all those in* $\hat{D}_{\mathsf{wp}}$ *weakly polynomial-time, then* D *is polynomial-time if all inputs to* $\hat{D}_{\mathsf{wp}}$ *are made by* $\hat{D}_{\mathsf{p}}$, *i.e., for every port* $\mathsf{p}? \in \mathsf{ports}(\hat{D}_{\mathsf{wp}})$, *we have* $\mathsf{p}!, \mathsf{p}^{\triangleleft}! \in \mathsf{ports}(\hat{D}_{\mathsf{p}})$.

$\square$

Proof. a) $\hat{C}$ is a collection, thus all machines in $\hat{D} \subseteq \hat{C}$ have unique names and disjoint port sets, and δ_{D} is well-defined. By definition, δ_{D} applied to a final state of D or to an empty input does not change the state and produces no output.

b) D is again simple or a master scheduler because we only combined such machines. Since we selected a new name for D and did not add any port, $\hat{C}^*$ is a collection. By construction $\mathsf{ports}(\hat{C}^*) = \mathsf{ports}(\hat{C})$, which implies

$\mathsf{ports}([\hat{C}^*]) = \mathsf{ports}([\hat{C}])$ and thus $\mathsf{free}([\hat{C}^*]) = \mathsf{free}([\hat{C}])$. Thus $\hat{C}^*$ is still closed.

c) Whenever the run algorithm (Definition 2.7) switches a non-buffer machine M then that machine is the only one that has a non-empty input, and only at one port (Lemma 2.1). Therefore we can identify each step of D with a step of the unique submachine of D that receives a non-empty input in that step, and vice versa. The other sub-machines, although switched by δ_{D}, neither change their states nor produce an output (Definition 2.2). Hence it makes no difference to the variables of the run algorithm that they are not switched in $\hat{C}$. In Phase (3) possibly more buffers switch in $[\hat{C}^*]$ than in $[\hat{C}]$, but they do not receive an input and thus nothing is added to the run. Overall, we have defined a bijection between the runs of the two systems whose projection to the views of any subsets of original machines are identical.

d) This can easily be seen from the associativity of the underlying operations (union of ports, Cartesian product of states with the given identification, transition functions, and initial states), and also for the final states.

e) The running time of D is bounded by the sum of the running times of the machines in $\hat{D}$. (It can be implemented by a Turing machine that has all the tapes of the individual machines and the Cartesian product of the state spaces; then it can simulate each step without overhead.)

f) As the running time of all machines in $\hat{D}_{\mathsf{p}}$ is polynomial, their overall output is of polynomial length. This bounds the overall input length of the machines in $\hat{D}_{\mathsf{wp}}$, as the intermediate buffer only delivers each message once. Thus their running time is also polynomial. ∎

With the notion of combination, we can add the notion of blackbox simulatability to Definition 2.14.

Definition 2.16 *(Universal and Blackbox Simulatability)* Universal simulatability *means that* A_2 *in Definition 2.14 does not depend on* H *(only on* $\hat{M}_1$*,* S*, and* A_1*).* Blackbox simulatability *means that* A_2 *is the combination of a fixed simulator* Sim*, depending at most on* $\hat{M}_1$*,* S *and* $\mathsf{ports}(\mathsf{A}_1)$*, and a machine* A'_1 *that differs from* A_1 *at most in the names and labels of some ports. The partial function* σ *that defines this renaming is tacitly assumed to be given with* Sim*.* A_1 *is then called a* blackbox submachine *of* Sim*.* ◇

Clearly, blackbox simulatability implies universal simulatability, and universal simulatability implies "standard" simulatability.

Lemma 2.3 (Users) *The condition on the ports of* H *in a configuration (Definition 2.10) is equivalent to* $\mathsf{ports}(\mathsf{H}) \cap \mathsf{ports}(\hat{M}) = \emptyset$ (1) *and* $\mathsf{ports}(\mathsf{H})^c \cap \mathsf{ports}([\hat{M}]) \subseteq S$ (2). □

Proof. Let $\mathsf{inner}(\hat{C}) := \mathsf{ports}(\hat{C}) \setminus \mathsf{free}(\hat{C})$ for every collection $\hat{C}$. Clearly $\mathsf{inner}(\hat{C})^c = \mathsf{inner}(\hat{C})$.

The original condition can be written as (1) and $\mathsf{ports}(\mathsf{H})^c \cap (\mathsf{free}([\hat{M}]) \setminus S) = \emptyset$ (3). Now (3) $\Leftrightarrow$ $\mathsf{ports}(\mathsf{H})^c \cap \mathsf{free}([\hat{M}]) \subseteq S$. It remains to be shown that $\mathsf{ports}(\mathsf{H})^c \cap \mathsf{inner}([\hat{M}]) = \emptyset$. This is equivalent to $\mathsf{ports}(\mathsf{H}) \cap \mathsf{inner}([\hat{M}]) = \emptyset$. Now $\mathsf{ports}([\hat{M}])$ only contains additional buffer ports and clock in-ports compared with $\mathsf{ports}(\hat{M})$. Hence (1) even implies $\mathsf{ports}(\mathsf{H}) \cap \mathsf{ports}([\hat{M}]) = \emptyset$.
■

Lemma 2.4 (Valid Mappings and Suitable Configurations) *Let systems* Sys_1 *and* Sys_2 *with a valid mapping* f *be given.*

a) Then $S^c \cap \mathsf{forb}(\hat{M}_i, S) = \emptyset$ *for* $i = 1, 2$*, i.e., the ports that users are intended to use are not at the same time forbidden (not even in the corresponding structures of the other system).*

b) With regard to Sys_1 *alone, the restriction to suitable configurations is without loss of generality in the following sense: For every* $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}(Sys_1) \setminus \mathsf{Conf}^f(Sys_1)$*, there is a configuration* $conf_{\mathsf{f},1} = (\hat{M}_1, S, \mathsf{H}_\mathsf{f}, \mathsf{A}_{\mathsf{f},1}) \in \mathsf{Conf}^f(Sys_1)$ *such that* $view_{conf_{\mathsf{f},1}}(\mathsf{H}_\mathsf{f}) = view_{conf_1}(\mathsf{H})$.

□

Proof. For Part a) recall that $\mathsf{forb}(\hat{M}_i, S) = \mathsf{ports}(\hat{M}_i) \cup (\mathsf{free}([\hat{M}_i]) \setminus S)^c$. Obviously we only have to show $S^c \cap \mathsf{ports}(\hat{M}_i) = \emptyset$. This follows from $S \subseteq \mathsf{free}([\hat{M}_i])$.

For Part b), we want to construct H_f by giving each port $p \in \mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(\hat{M}_2, S)$ a new name. Clearly the runs and views remain the same if we consistently rename all six ports with the same name. The new collection is a configuration $(\hat{M}_1, S, \mathsf{H}_\mathsf{f}, \mathsf{A}_{\mathsf{f},1})$ if none of the renamed ports belongs to $[\hat{M}_1]$.

If it were, then $p^c \in \mathsf{ports}([\hat{M}_1])$ because it is a buffer port, and with any port, the entire buffer belongs to $[\hat{M}_1]$. Now Lemma 2.3 implies $p^c \in S$. As f is a valid mapping, Part a) implies $p \notin \mathsf{forb}(\hat{M}_2, S)$, in contradiction to the original condition on p. ∎

Lemma 2.5 (Indistinguishability) *a) The statistical distance $\Delta(\phi(\mathsf{var}_k), \phi(\mathsf{var}'_k))$ between a function ϕ of two random variables is at most $\Delta(\mathsf{var}_k, \mathsf{var}'_k)$.*

b) Perfect indistinguishability of two families of random variables implies perfect indistinguishability of every function ϕ of them. The same holds for statistical indistinguishability with any class SMALL, and for computational indistinguishability if ϕ is polynomial-time computable and the elements of D_k are of polynomial length in k.

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

d) All three types of indistinguishability are equivalence relations.

□

These are well-known facts; hence we omit the easy proof.

Lemma 2.6 (Types of Security) *If $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{perf}} Sys_2$, then $Sys_1 \geq_{\mathsf{sec}}^{f,SMALL} Sys_2$ for every non-empty class SMALL. Similarly, $Sys_1 \geq_{\mathsf{sec}}^{f,SMALL} Sys_2$ for a class $SMALL \subseteq NEGL$ implies $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{poly}} Sys_2$.* □

Proof. The first part follows immediately from Lemma 2.5 with the fact that equality of possibly infinite views implies equality of all their fixed-length prefixes; the second part with the fact that the view of H in a polynomial-time configuration is of polynomial length and that the distinguisher is a special case of a function ϕ. ∎

Lemma 2.7 (Transitivity) *If $Sys_1 \geq^{f_1} Sys_2$ and $Sys_2 \geq^{f_2} Sys_3$, then $Sys_1 \geq^{f_3} Sys_3$, where $f_3 := f_2 \circ f_1$ is defined in a natural way as follows: $f_3(\hat{M}_1, S)$ is the union of the sets $f_2(\hat{M}_2, S)$ with $(\hat{M}_2, S) \in f_1(\hat{M}_1, S)$. This holds for perfect, statistical and computational security, and also for universal and blackbox simulatability.* □

Proof. Clearly, f_3 is always a valid mapping.

Let a configuration $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^{f_3}(Sys_1)$ be given. Hence $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(\hat{M}_3, S) = \emptyset$ for all $(\hat{M}_3, S) \in f_3(\hat{M}_1, S)$ $(*)$.

If $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(\hat{M}_2, S) \neq \emptyset$ for a $(\hat{M}_2, S) \in f_1(\hat{M}_1, S)$, we give these ports new names: By Lemma 2.4b), we derive a configuration $conf_{\mathsf{f},1} = (\hat{M}_1, S, \mathsf{H}_\mathsf{f}, \mathsf{A}_{\mathsf{f},1}) \in \mathsf{Conf}^{f_1}(Sys_1)$ with $view_{conf_{\mathsf{f},1}}(\mathsf{H}_\mathsf{f}) = view_{conf_1}(\mathsf{H})$.

Because of $Sys_1 \geq^{f_1} Sys_2$, there exists a configuration $conf_{\mathsf{f},2} = (\hat{M}_2, S, \mathsf{H}_\mathsf{f}, \mathsf{A}_{\mathsf{f},2}) \in \mathsf{Conf}(Sys_2)$ with $(\hat{M}_2, S) \in f_1(\hat{M}_1, S)$ such that $view_{conf_{\mathsf{f},1}}(\mathsf{H}_\mathsf{f}) \approx view_{conf_{\mathsf{f},2}}(\mathsf{H}_\mathsf{f})$.

As H_f only has ports from H and new ports, and by $(*)$ and the definition of f_3, it has no ports from $\mathsf{forb}(\hat{M}_3, S)$ for any structure $(\hat{M}_3, S) \in f_2(\hat{M}_2, S)$, i.e., $conf_{\mathsf{f},2} \in \mathsf{Conf}^{f_2}(Sys_2)$. Hence, by $Sys_2 \geq^{f_2} Sys_3$, there exists $conf_{\mathsf{f},3} = (\hat{M}_3, S, \mathsf{H}_\mathsf{f}, \mathsf{A}_{\mathsf{f},3}) \in \mathsf{Conf}(Sys_3)$ with $(\hat{M}_3, S) \in f_2(\hat{M}_2, S)$ and $view_{conf_{\mathsf{f},2}}(\mathsf{H}_\mathsf{f}) \approx view_{conf_{\mathsf{f},3}}(\mathsf{H}_\mathsf{f})$.

Together, we have $(\hat{M}_3, S) \in f_3(\hat{M}_1, S)$ by definition of f_3 and $view_{conf_{\mathsf{f},1}}(\mathsf{H}_\mathsf{f}) \approx view_{conf_{\mathsf{f},3}}(\mathsf{H}_\mathsf{f})$ because indistinguishability is transitive (Lemma 2.5).

Finally, we derive a configuration $conf_3 = (\hat{M}_3, S, \mathsf{H}, \mathsf{A}_3)$ with the original user H. For each changed port $p \in \mathsf{ports}(\mathsf{H})$, no port with the same name occurs in $\mathsf{ports}([\hat{M}_3])$ because the name was new. Thus we can change the name back (in all six ports that have it) iff the old name also does not occur in $\mathsf{ports}([\hat{M}_3])$. If this were not true, then as in the proof of Lemma 2.4b), in particular $p^c \in \mathsf{ports}([\hat{M}_3])$, and Lemma 2.3 and $(*)$ imply $p^c \in S$. As f_2 is a valid mapping, Lemma 2.4a) implies $p \notin \mathsf{forb}(\hat{M}_2, S)$, in contradiction to the condition for renaming p.

Hence we have $view_{conf_3}(\mathsf{H}) = view_{conf_{\mathsf{f},3}}(\mathsf{H}_\mathsf{f})$ and thus $view_{conf_1}(\mathsf{H}) \approx view_{conf_3}(\mathsf{H})$. This finishes the proof for "standard" simulatability.

Now we show universal simulatability. First, the renaming from A_1 to $\mathsf{A}_{\mathsf{f},1}$ can be described in terms of the ports of A_1 and $(\hat{M}_1, S)$ (they uniquely define the ports of H with the same name as a port of A_1). For $\mathsf{A}_{\mathsf{f},2}$ and $\mathsf{A}_{\mathsf{f},3}$ we use the given universality, and the last renaming into A_3 is the reverse of the first. For blackbox simulatability, renaming can be done as a blackbox construction, and $\mathsf{A}_{\mathsf{f},2}$ uses $\mathsf{A}_{\mathsf{f},1}$, and $\mathsf{A}_{\mathsf{f},3}$ uses $\mathsf{A}_{\mathsf{f},2}$, as a blackbox by the preconditions. Associativity of combinations (Lemma 2.2) implies that this is equivalent to one simulator with a blackbox. ∎

Clearly, $\geq^f$ is also reflexive with the identity function $f = \mathsf{id}$.

2.4 Standard Cryptographic Systems

In this section, we define an important specialization of the general model which targets common cryptographic settings. In Section 2.4.1, we define such standard cryptographic systems restricted to static adversaries, similar to the synchronous model defined in the deliverable D4 [4]. In Section 2.4.2, we show how to model also adaptive adversaries. Adaptive adversaries have not been addressed in the synchronous model presented in the deliverable D4 [4] yet they represent an important class of adversaries. However, the approach chosen here is also adaptable to the synchronous case.

2.4.1 Static Adversaries

The intuition behind this class of systems is that in a real system *Sys*, there is one machine per human owner, and each machine is correct if and only if its owner is honest. Furthermore, a correct machine is assumed to stay correct during the complete lifetime of the system. The system is derived from an intended structure $(\hat{M}^*, S^*)$ and a trust model.

We define that all buffers that connect different machines are scheduled by the adversary. We only allow a machine M_u to schedule buffers that transport messages from itself to itself, and require all these connections to be secure: this allows us to define a machine M_u as a combination of (local) sub-machines. The case where the user in- and outputs are also treated in this way is called localized.

Definition 2.17 *(Standard Cryptographic Structures and Trust Models) A* standard cryptographic structure *is a structure* $(\hat{M}^*, S^*)$ *where* $\hat{M}^* = \{\mathsf{M}_1, \ldots, \mathsf{M}_n\}$ *with* $n \in \mathbb{N}$ *and* $S^{*c} = \{\mathsf{in}_u!, \mathsf{out}_u?|u = 1, \ldots, n\}$*, where* $\mathsf{in}_u?$ *and* $\mathsf{out}_u!$ *are ports of machine* M_u*. (We have specified the complement of* S^* *because that is independent of the buffer notation.) Each machine* M_u *is simple, and for all names* p*, if* $p^{\triangleleft}! \in \mathsf{ports}(\mathsf{M}_u)$ *then* $p?, p! \in \mathsf{ports}(\mathsf{M}_u)$*.*

A localized cryptographic structure *is the same except that for all* $u = 1, \ldots, n$*,* $\mathsf{in}_u{}^{\triangleleft}!$ *also belongs to* S^{*c} *and* $\mathsf{out}_u{}^{\triangleleft}!$ *to* $\mathsf{ports}(\mathsf{M}_u)$*.*

A standard trust model *for such a structure is a pair* $(\mathcal{ACC}, \chi)$ *of an*

access structure *and a* channel model. *Here* $\mathcal{ACC} \subseteq \mathcal{P}(\{1,\ldots,n\})$ *is closed under insertion (of more elements) and denotes the possible sets of correct machines.* χ *is a mapping* $\chi : \mathsf{Gr}(\hat{M}^*) \to \{\mathsf{s},\mathsf{a},\mathsf{i}\}$. *It characterizes each high-level connection as secure (private and authentic), authenticated (only authentic), or insecure (neither private nor authentic). If a connection* c *connects a machine* M_u *with itself, we require* $\chi(c) = \mathsf{s}$. $\diamond$

Typical examples are threshold structures $\mathcal{ACC}_t := \{\mathcal{H} \subseteq \{1,\ldots,n\} \mid |\mathcal{H}| \geq t\}$ with $t \leq n$.

Definition 2.18 *(Standard Static Cryptographic Systems) Given a standard (or localized) cryptographic structure and trust model, the corresponding* standard (or localized) cryptographic system with static adversary

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

is $Sys := \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) | \mathcal{H} \in \mathcal{ACC}\}$ *with* $S^c_{\mathcal{H}} := \{\mathsf{in}_u!, \mathsf{out}_u? | u \in \mathcal{H}\}$, *and* $\mathsf{in}_u{}^{\triangleleft}!$ *in the localized case, and* $\hat{M}_{\mathcal{H}} := \{\mathsf{M}_{u,\mathcal{H}} | u \in \mathcal{H}\}$, *where* $\mathsf{M}_{u,\mathcal{H}}$ *is derived from* M_u *as follows:*

- *The ports* $\mathsf{in}_u?$ *and* $\mathsf{out}_u!$ *and all clock ports are unchanged.*
- *Consider a simple port* $p \in \mathsf{ports}(\mathsf{M}_u) \setminus \{\mathsf{in}_u?, \mathsf{out}_u!\}$, *where* $p^C \in \mathsf{ports}(\mathsf{M}_v)$ *with* $v \in \mathcal{H}$, *i.e.,* $c = \{p, p^C\}$ *is a high-level connection between two correct machines:*
  - *If* $\chi(c) = \mathsf{s}$ *(secure),* p *is unchanged.*
  - *If* $\chi(c) = \mathsf{a}$ *(authenticated) and* p *is an output port,* $\mathsf{M}_{u,\mathcal{H}}$ *gets an additional new port* p^d, *where it duplicates the outputs at* p. *This can be done by a trivial blackbox construction. We assume without loss of generality that there is a systematic naming scheme for such new ports (e.g., appending* d*) that does not clash with prior names. The new 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 the adversary can get the outputs from* p^C *and make the inputs to* p^a *and thus completely control the connection.) If* p *is an output port, it is unchanged.*

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

For localized systems, the same definition holds with the obvious modifications: Ports $\mathsf{in}_u{}^{\triangleleft}!$ *with* $u \in \mathcal{H}$ *also belong to* $S^c_{\mathcal{H}}$*, and* p *is only chosen in* $\mathsf{ports}(\mathsf{M}_u) \setminus \{\mathsf{in}_u?, \mathsf{out}_u!, \mathsf{out}_u{}^{\triangleleft}!\}$. ◇

Definition 2.19 *(Standard Static Ideal Systems) A* standard (or localized) static ideal system *is of the form* $Sys_2 = \{(\{\mathsf{TH}_{\mathcal{H}}\}, S_{\mathcal{H}}) | \mathcal{H} \in \mathcal{ACC}\}$ *for an access structure* $\mathcal{ACC} \subseteq \{1, \ldots, n\}$ *for some* $n \in \mathbb{N}$ *and the same sets of specified ports as in corresponding real systems, i.e.,* $S^c_{\mathcal{H}} := \{\mathsf{in}_u!, \mathsf{out}_u?, (\mathsf{in}_u{}^{\triangleleft}!) | u \in \mathcal{H}\}$. ◇

One then compares a standard or localized static real system with a standard or localized static ideal system with the same access structure, using the canonical mapping (Definition 2.11).

2.4.2 Adaptive Adversaries

Standard cryptographic systems as defined in the previous section are based on the intuition that it is a priori clear who are the "bad guys" and who are the "good guys." However, real-world adversaries often can also corrupt honest machines during the lifetime of a system, e.g., by exploiting buffer overflows or badly set security policies with a trojan horse hidden in an e-mail. This corresponds to malicious faults. In the following *adaptive* (or *dynamic*) adversary model, the set of corrupted machines can increase over time.

Adaptive adversary models are strictly more powerful than static ones, i.e., there are examples of systems secure against static adversaries that are insecure against adaptive adversaries who can corrupt the same sets of machines [17].

Definition 2.20 *(Adaptive Standard Cryptographic Systems) Let a standard (or localized) cryptographic structure* $(\hat{M}^*, S^*)$ *with a channel model* χ *be*

given. The corresponding standard cryptographic system with adaptive adversary *has only one structure,* $(\hat{M}, S)$ *with* $\hat{M} := \{\mathsf{M}'_u | u \in \{1, \ldots, n\}\}$. *Here*

$$S^c := S^{*c} \cup \{\mathsf{corrupt}_u? | u \in \{1, \ldots, n\}\}.$$

The new ports are for corruption requests. (Those must be made via specified ports because the service will change at least at the corresponding ports $\mathsf{in}_u?$ *and* $\mathsf{out}_u!$ *also in the ideal system.)*

For the machines, let $\mathsf{M}_{u,\mathcal{H}}$ *for* $u := 1, \ldots, n$ *denote the machines derived as in the static case (Definition 2.18) with* $\mathcal{H} := \{1, \ldots, n\}$ *(all intended machines are present) and* χ. *Then* M'_u *is derived from* $\mathsf{M}_{u,\mathcal{H}}$ *as follows: It gets the new port* $\mathsf{corrupt}_u?$, *and two new ports* $\mathsf{cor\_out}_u!$, $\mathsf{cor\_in}_u?$ *for communication with* A *after corruption. On input* 1 *at* $\mathsf{corrupt}_u?$, *it sends a predefined "corruption response"* $(\mathsf{corruption}, \sigma)$ *to* A *via* $\mathsf{cor\_out}_u!$, *and from then on becomes "transparent:" Every input* m *at a port* $\mathsf{p}?$ *is translated into the output* $(\mathsf{p}?, m)$ *at* $\mathsf{cor\_out}_u!$, *and every input* $(\mathsf{p}!, m)$ *at* $\mathsf{cor\_in}_u?$ *is translated into the output* m *at* $\mathsf{p}!$.

There are two main types of corruption responses σ*:*

a) With erasure: σ *is the current state of* M'_u.

b) Without erasure: σ *is the entire view of* M'_u. *This corresponds to the assumption that nothing can be erased reliably. Thus every transition of* δ_{M_u} *is modified in* $\delta_{\mathsf{M}'_u}$ *to store the current step.*

◇

Definition 2.21 *(Standard Adaptive Ideal Systems) A* standard (or localized) adaptive ideal system *only has one structure* $(\{\mathsf{TH}\}, S)$ *with* S *as in Definition 2.20. Tolerable sets of corrupted machines are defined by an access structure* $\mathcal{ACC}^*$ *within* TH*: If the set of received corruption requests is no longer in* $\mathcal{ACC}^*$, *i.e., there were "too many," then* TH *sends its state to* A *and gives all control to* A. *(Thus after this, the ideal system no longer guarantees anything and simulation becomes trivial.)* ◇

Several extensions are possible: One may extend the corruption responses to two classes of storage, an erasable and a non-erasable one, e.g., to model the different vulnerability of session keys and long-term keys. This means to refine the state spaces of each machine as a Cartesian product. In- and outputs

would be treated like erasable storage. One can also model non-binary corruption requests, e.g., stop requests and requests to corrupt different classes of storage. To model proactive systems [42], one needs repair requests in addition to corruption requests, and appropriate repair responses, e.g., returning to an initial state with only a certain class of storage still intact.

2.5 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 2.3.

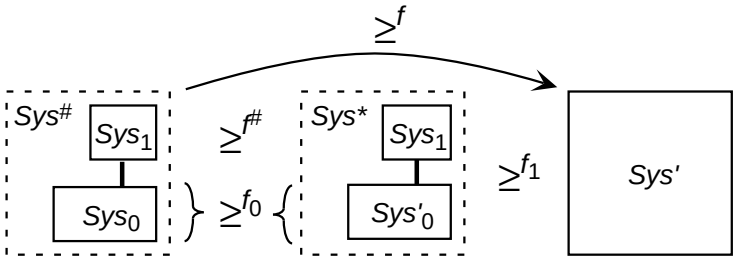

Figure 2.3: Composition theorem and its use in a modular proof: The left and middle part show the statement of Theorem 2.1, the right part Corollary 2.1.

The remainder of this section is quite similar to the corresponding section of [48] for the synchronous case.

We define composition for every number n of systems $Sys_1, \ldots, Sys_n$. We do not provide a composition operator that produces one specific composition. The reason is that one typically does not want to compose every

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

Definition 2.22 *(Composition) The composition of structures and of systems is defined as follows:*

a) Structures $(\hat{M}_1, S_1), \ldots, (\hat{M}_n, S_n)$ *are* composable *if* $\mathsf{ports}(\hat{M}_i) \cap \mathsf{forb}(\hat{M}_j, S_j) = \emptyset$ *and* $S_i \cap \mathsf{free}([\hat{M}_j]) = S_j \cap \mathsf{free}([\hat{M}_i])$ *for all* $i \neq j$. *Their* composition *is then* $(\hat{M}_1, S_1)|| \ldots ||(\hat{M}_n, S_n) := (\hat{M}, S)$ *with* $\hat{M} = \hat{M}_1 \cup \ldots \cup \hat{M}_n$ *and* $S = (S_1 \cup \ldots \cup S_n) \cap \mathsf{free}([\hat{M}])$.

b) A system Sys a composition *of* $Sys_1, \ldots, Sys_n$, *written* $Sys \in Sys_1 \times \cdots \times Sys_n$, *if each structure* $(\hat{M}, S) \in Sys$ *has a unique representation* $(\hat{M}, S) = (\hat{M}_1, S_1)|| \ldots ||(\hat{M}_n, S_n)$ *with composable structures* $(\hat{M}_i, S_i) \in Sys_i$ *for* $i = 1, \ldots, n$.

c) We then call $(\hat{M}_i, S_i)$ *the* restriction *of* $(\hat{M}, S)$ *to* Sys_i *and write* $(\hat{M}_i, S_i) = (\hat{M}, S)\lceil_{Sys_i}$.

$\diamond$

The first condition for composability makes one structure a valid user of another. The second one excludes ambiguities for S, specifically the case where $p \in \mathsf{free}([\hat{M}_i]) \cap \mathsf{free}([\hat{M}_j])$ (e.g., a clock port for a high-level connection between these systems) and $p \in S_i$ but $p \notin S_j$.

Lemma 2.8 (Composition Technicalities) *a) For all compositions of structures, we have* $[\hat{M}] = [\hat{M}_1] \cup \ldots \cup [\hat{M}_n]$ *and* $\mathsf{free}([\hat{M}]) \subseteq \mathsf{free}([\hat{M}_1]) \cup \ldots \cup \mathsf{free}([\hat{M}_n])$.

b) We also have $S = \mathsf{free}([\hat{M}]) \setminus (\bar{S}_1 \cup \ldots \cup \bar{S}_n)$. *(By* $\bar{S}_i$ *we still mean* $\mathsf{free}([\hat{M}_i]) \setminus S_i$*.)*

$\square$

Proof. a) Structures consist of simple machines only. Hence both sides of the first equation contain all these simple machines and all the buffers for their

ports. If $p \in \mathsf{free}([\hat{M}])$, we now know $p \in \mathsf{ports}([\hat{M}_i])$ for some i. Furthermore, $p^c \notin \mathsf{ports}([\hat{M}])$ and thus $p^c \notin \mathsf{ports}([\hat{M}_i])$. Thus $p \in \mathsf{free}([\hat{M}_i])$.

b) Let $S' := \mathsf{free}([\hat{M}]) \setminus (\bar{S}_1 \cup \ldots \cup \bar{S}_n)$. If $p \in S'$, then $p \in \mathsf{free}([\hat{M}_i])$ for some i by Part a), and $p \notin \bar{S}_i$ immediately gives $p \in S_i$. Now assume that $p \in S \setminus S'$. Thus $p \in \mathsf{free}([\hat{M}])$, but there exists i with $p \in \bar{S}_i$. However, there also exists j with $p \in S_j$. Together, $p \in S_j \cap \mathsf{free}([\hat{M}_i])$. By composability, this implies $p \in S_i \cap \mathsf{free}([\hat{M}_j])$, contradicting $p \in \bar{S}_i$. ∎

The following theorem shows that modular proofs are indeed possible. Recall that the situation is shown in the left and middle part of Figure 2.3. 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 2.1 (Secure Two-system Composition) *Let Sys_0, Sys'_0, Sys_1 be systems and $Sys_0 \geq^{f_0} Sys'_0$ for a valid mapping f_0.*

Let $Sys^{\#} \in Sys_0 \| Sys_1$ and $Sys^* \in Sys'_0 \| Sys_1$ be compositions that fulfill the following structural conditions: For every structure $(\hat{M}^{\#}, S) \in Sys^{\#}$ with restrictions $(\hat{M}_i, S_i) = (\hat{M}^{\#}, S)\lceil_{Sys_i}$ and every $(\hat{M}'_0, S_0) \in f_0(\hat{M}_0, S_0)$, the composition $(\hat{M}'_0, S_0)||(\hat{M}_1, S_1)$ exists, lies in Sys^*, and fulfills $\mathsf{ports}(\hat{M}'_0) \cap S_1^c = \mathsf{ports}(\hat{M}_0) \cap S_1^c$.

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

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

This holds for perfect, statistical and, if Sys_1 is polynomial-time, for computational security, and also for the universal and blackbox definitions. □

Proof. First we have to show that $f^{\#}$ is a valid mapping. This will be done in Step 0 below. Then let a configuration $conf^{\#} = (\hat{M}^{\#}, S, \mathsf{H}, \mathsf{A}^{\#}) \in \mathsf{Conf}^{f^{\#}}(Sys^{\#})$ be given and $(\hat{M}_i, S_i) := (\hat{M}^{\#}, S)\lceil_{Sys_i}$ for $i = 0, 1$. We have to show that there is an indistinguishable configuration $conf^* \in \mathsf{Conf}(Sys^*)$ for it. The outline of the proof is as follows; it is illustrated in Figure 2.4.

1. We combine H and $\hat{M}_1$ into a user H_0 to obtain a configuration $conf_0 = (\hat{M}_0, S_0, \mathsf{H}, \mathsf{A}_0) \in \mathsf{Conf}(Sys_0)$ where the view of H as a submachine of H_0 is the same as that in $conf^{\#}$.

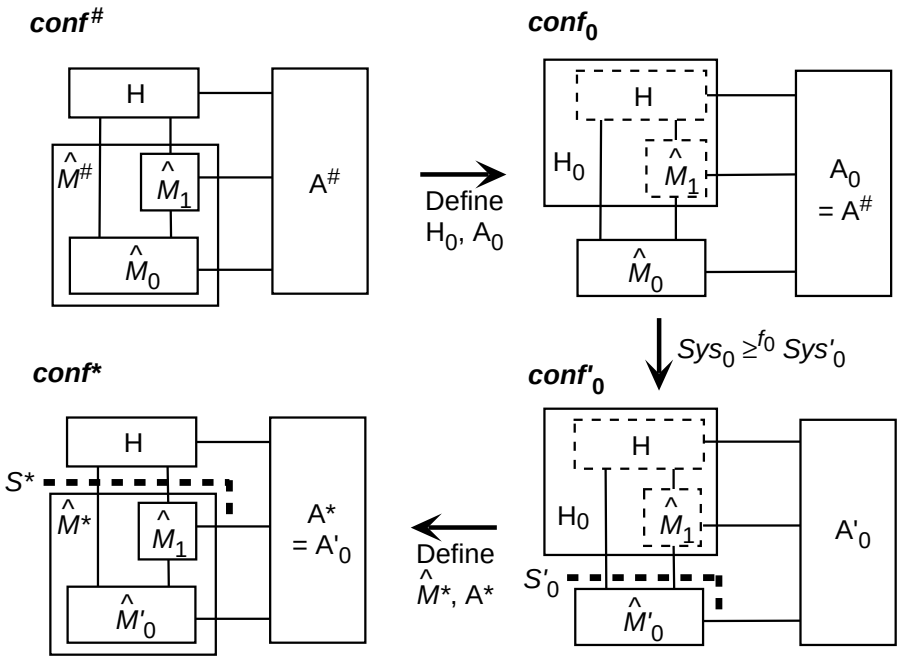

Figure 2.4: Configurations in the composition theorem. Dashed machines are internal sub-machines. (The connections drawn inside H_0 are not dashed because the combination does not hide them.)

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 $conf'_0 = (\hat{M}'_0, S_0, \mathsf{H}_0, \mathsf{A}'_0) \in \mathsf{Conf}(Sys'_0)$ with $(\hat{M}'_0, S_0) \in f_0(\hat{M}_0, S_0)$ where the view of H_0 is indistinguishable from that in $conf_0$.

3. We decompose H_0 into H and $\hat{M}_1$ again and derive a configuration $conf^* = (\hat{M}^*, S, \mathsf{H}, \mathsf{A}^*) \in \mathsf{Conf}(Sys^*)$ 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 these steps in detail.

Step 0: We have to show that $S^* = S$ whenever $(\hat{M}^*, S^*) \in f^\#(\hat{M}^\#, S)$. Let $(\hat{M}_i, S_i) := (\hat{M}^\#, S)\lceil_{Sys_i}$ for $i = 0, 1$. Then $(\hat{M}^*, S^*)$ is a composition $(\hat{M}'_0, S_0)||(\hat{M}_1, S_1)$ with $(\hat{M}'_0, S_0) \in f_0(\hat{M}_0, S_0)$. Hence $S = (S_0 \cup S_1) \cap \mathsf{free}([\hat{M}^\#])$ and $S^* = (S_0 \cup S_1) \cap \mathsf{free}([\hat{M}^*])$.

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

If $p \in S_0$, then $p^c \notin \mathsf{ports}([\hat{M}'_0])$ because $(\hat{M}'_0, S_0)$ is a structure, and $p^c \notin \mathsf{ports}([\hat{M}^\#]) \supseteq \mathsf{ports}([\hat{M}_1])$ because p is free in $[\hat{M}^\#]$. With Lemma 2.8b), $p^c \notin \mathsf{ports}([(\hat{M}'_0 \cup \hat{M}_1)]) = \mathsf{ports}([\hat{M}^*])$. Thus $p \in S^*$.

If $p \in S_1$, then $p^c \notin \mathsf{ports}([\hat{M}_1])$. Hence $p \notin S^*$ would imply $p^c \in \mathsf{ports}([\hat{M}'_0])$ and thus $p^c \in \mathsf{ports}(\hat{M}'_0)$ because it is not a port of a buffer. By the precondition $\mathsf{ports}(\hat{M}'_0) \cap S_1^c = \mathsf{ports}(\hat{M}_0) \cap S_1^c$ of the theorem, this would imply $p^c \in \mathsf{ports}(\hat{M}_0)$ in contradiction to the fact that $p \in \mathsf{free}([\hat{M}^\#])$.

Step 1: The precise definition of $conf_0 = (\hat{M}_0, S_0, \mathsf{H}, \mathsf{A}_0)$ is $(\hat{M}_0, S_0) = (\hat{M}^\#, S)\lceil_{Sys_0}$ and $\mathsf{H}_0 := \mathsf{comb}(\hat{M}_1 \cup \{\mathsf{H}\})$ and $\mathsf{A}_0 := \mathsf{A}^\#$. This is a valid configuration from $\mathsf{Conf}(Sys_0)$:

- $(\hat{M}_0, S_0) = (\hat{M}^\#, S)\lceil_{Sys_0}$ is a valid structure by the definition of a composition.

- Closed collection: The overall set of ports is the same as in $conf^\#$. Hence the machines still have pairwise disjoint port sets, and the free ports are unchanged.

- User condition: We have to show $\mathsf{ports}(\mathsf{H}_0) \cap \mathsf{forb}(\hat{M}_0, S_0) = \emptyset$. Composability implies $\mathsf{ports}(\hat{M}_1) \cap \mathsf{forb}(\hat{M}_0, S_0) = \emptyset$, and port disjointness has

just been shown. Thus $\mathsf{ports}(\mathsf{H}) \cap \bar{S}_0^c = \emptyset$ remains to be shown. Assume that a port p contradicts this. Thus $p^c \in \mathsf{free}([\hat{M}_0]) \subseteq \mathsf{ports}([\hat{M}^\#])$, and the validity of $conf^\#$ and Lemma 2.3 imply $p^c \in S$. By Lemma 2.8b), this contradicts $p^c \in \bar{S}_0$.

- For the computational case, H and all machines in $\hat{M}_1$ are polynomial-time by the preconditions. Hence H_0 is polynomial-time by Lemma 2.2.

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 $\mathsf{forb}(\hat{M}_0', S_0)$ for any structure $(\hat{M}_0', S_0) \in f_0(\hat{M}_0, S_0)$. Assume that it had such a port p. By construction of H_0, p is a port of $\hat{M}_1$ or H.

The case $p \in \mathsf{ports}(\hat{M}_1)$ is excluded by the required composability of $(\hat{M}_0', S_0)$ and $(\hat{M}_1, S_1)$.

Thus let $p \in \mathsf{ports}(\mathsf{H})$. We required that $(\hat{M}^*, S) := (\hat{M}_0', S_0)||(\hat{M}_1, S_1)$ exists and lies in $f^\#(\hat{M}^\#, S)$. As $conf^\#$ is suitable, this implies $p \notin \mathsf{forb}(\hat{M}^*, S)$. Hence $p \notin \mathsf{ports}(\hat{M}^*)$ and thus $p \notin \mathsf{ports}(\hat{M}_0')$. Thus the case $p^c \in \mathsf{free}([\hat{M}_0']) \setminus S_0$ remains. Using Lemma 2.8b) for $(\hat{M}^*, S)$, this implies $p^c \notin S$. Furthermore, $p^c \in \mathsf{free}([\hat{M}^*])$ because $p \notin \mathsf{ports}(\hat{M}^*)$ implies $p \notin \mathsf{ports}([\hat{M}^*])$ (since p cannot be a port of a buffer). Hence $p^c \in \mathsf{free}([\hat{M}^*]) \setminus S$ in contradiction to $p \notin \mathsf{forb}(\hat{M}^*, S)$.

Hence $conf_0$ is indeed a suitable configuration. Thus $Sys_0 \geq^{f_0} Sys_0'$ implies that there is a configuration $conf_0' = (\hat{M}_0', S_0, \mathsf{H}_0, \mathsf{A}_0') \in \mathsf{Conf}(Sys_0')$ with $(\hat{M}_0', S_0) \in f_0(\hat{M}_0, S_0)$ 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 2.5 and 2.2).

Step 3: We define $conf^* = (\hat{M}^*, S, \mathsf{H}, \mathsf{A}^*)$ by reversing the combination of H and $\hat{M}_1$ into H_0: The structure is $(\hat{M}^*, S) := (\hat{M}_0', S_0)||(\hat{M}_1, S_1)$, the user the original H, and $\mathsf{A}^* := \mathsf{A}_0'$. We show that $conf^* \in \mathsf{Conf}(Sys^*)$.

- Structure: $(\hat{M}^*, S) \in Sys^*$ follows immediately from the preconditions of the theorem.

- Closed collection: The ports of H and the machines in $\hat{M}_1$ are disjoint because so they were in $conf^\#$, and those of all other pairs of machines because so they were in $conf_0'$. (Recall that $\mathsf{ports}(\mathsf{H}_0) = \mathsf{ports}(\hat{M}_1 \cup \{\mathsf{H}\})$; here we exploit that we did not hide internal connections in the

combination.) As the set of ports is the same as in $conf'_0$, the free ports are unchanged.

- User condition: We have to show $\mathsf{ports}(\mathsf{H}) \cap \mathsf{forb}(\hat{M}^*, S) = \emptyset$. This is simply the precondition that $conf^\#$ is suitable.

We can now see $conf'_0$ as derived from $conf^*$ by taking the combination of $\hat{M}_1 \cup \{\mathsf{H}\}$. 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 $(\hat{M}^*, S) \in f^\#(\hat{M}^\#, S)$ by the construction of $f^\#$. The results about views in Steps 1 to 3 and transitivity (Lemma 2.7) 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 $(\hat{M}_0, S_0)$ and A_0, and thus so does $\mathsf{A}^* = \mathsf{A}'_0$. 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 $\mathsf{A}^* = \mathsf{A}'_0$. ∎

The following corollary finishes the definition and proof of the situation shown in Figure 2.3: We now assume that there is also a specification Sys' for the system Sys^*, as shown in the left part of the figure. The result immediately follows from transitivity.

Corollary 2.1 *Consider five systems satisfying the preconditions of Theorem 2.1, 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.* □

3 Example: Secure Message Transmission

In the following, we present an ideal and a real system for *secure message transmission*. This service enables secure point-to-point channels and is a basic building block of the MAFTIA middle-ware. We chose a similar example as in MAFTIA deliverable D4 [4] to illustrate the influence of the synchrony assumptions of the network on the modeling. Further illustrating applications of the security model in security definitions and proofs can be found in two separate papers: The first paper [7] links the model with the classical notion of non-interference and provides, based on above secure message transmission service, a cryptographic firewall. The second paper [46] models secure multi-party key establishment, a potential MAFTIA middleware service, and shows how to integrate complexity-theoretic assumptions as well as how to handle adaptive corruptions.

The real system sends messages over insecure channels, but for the initial key exchange authenticated channels may be used as well.

To keep the possible real implementations simple in the following bottom-level proof that must be done with cryptographic reductions, we specify an ideal system that tolerates some imperfections: The adversary learns who communicates with whom and the length of the messages. He can delay messages, change their order and suppress or replay them (without, however, seeing their contents). Measures that avoid some imperfections can be defined on top of our ideal system and proven with the composition theorem. In particular, the length of bounded-length messages can be hidden by padding, and delivery out of sequence (in particular replays) can be prevented by nonces or counters. Hiding who communicates with whom requires complicated measures against traffic analysis, and suppression or arbitrary delay of messages cannot be prevented at all.

Notation for data structures. For $m \in \Sigma^*$ let $\mathsf{len}(m)$ denote the length of m. A *list*, $l = (x_1, \ldots, x_j)$, is a sequence of words from Σ^*, itself encoded as a word from Σ^*. We also call fixed-length lists tuples or arrays. The exact encoding does not matter as long as the number $\mathsf{size}(l)$ of elements in l and these elements are efficiently retrievable, and the length of a list is efficiently computable from the length of its elements. For a list $l = (x_1, \ldots, x_j)$ we define $l[i] := x_i$ for $1 \leq i \leq j$, and $l[i] := \downarrow$ for $i > j$, where $\downarrow$ is a distinct error symbol, i.e., $\downarrow \notin \Sigma^*$. The empty list is written $()$. By "adding an

element to a list" and similar formulations we mean appending it at the end. By $\exists x \in l$ we mean that $l[i] = x$ for some i. If we write this in a retrieval operation, the first such x is used.

Conventions about machines. Because of Lemma 2.1a), it suffices to specify the state-transition functions by rules for individual inputs, usually written "On input i with $i \in D$: ...", where D is a domain. Inputs where no rule applies are *ignored*, i.e., the new state equals the old one and no output is made. If an input triggers an output $\neq \epsilon$ or a state change, we say that the machine *accepts* this input.

We do not explicitly write the length functions of the following machines, but assume that they are derived from the domain D. As they are only needed to make machines polynomial-time or weakly polynomial-time, any polynomial upper bound is good enough. We also omit the order of the ports because it does not matter in the proof.

Finally, we omit all outputs at clock out-ports. Each machine we specify only has a port $\mathsf{p}^{\triangleleft}!$ if it also has $\mathsf{p}!$. It makes an output 1 at $\mathsf{p}^{\triangleleft}!$ for each non-empty output at $\mathsf{p}!$, and no others.

3.1 Ideal System

The ideal system is of the standard or localized type from Definition 2.19 (everything holds for both variants), and any number of participants may be faulty. It has a polynomial bound L on the length of messages as a parameter. We also explicitly model initialization (key exchange in real systems) and that the adversary can completely stop machines (by exceeding their runtime bounds in the real system). Most of this could be hidden (because the adversary in the ideal system can suppress messages whenever anything is wrong in the real system), but we find the explicit specification clearer.

Scheme 3.1 (Ideal System for Secure Message Transmission) *Let $n \in \mathbb{N}$ and a polynomial $L \in \mathbb{N}[x]$ be given. Let $\mathcal{M} := \{1, \ldots, n\}$ and $\mathcal{ACC} := \mathcal{P}(\mathcal{M})$. Then*

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

with the standard definition $S_{\mathcal{H}}^{c} := \{\mathsf{in}_u!, \mathsf{out}_u? \mid u \in \mathcal{H}\}$, and $\mathsf{in}_u{}^{\triangleleft}!$ in the localized version, and $\mathsf{TH}_{\mathcal{H}}$ defined as follows. When $\mathcal{H}$ is clear from the context, let $\mathcal{A} := \mathcal{M} \setminus \mathcal{H}$ denote the indices of corrupted machines.

The ports of $\mathsf{TH}_{\mathcal{H}}$ *are* $\{\mathsf{in}_u?, \mathsf{out}_u!, \mathsf{from\_adv}_u?, \mathsf{to\_adv}_u!, \mathsf{to\_adv}_u{}^{\triangleleft}! \mid u \in \mathcal{H}\}$*, and* $\mathsf{out}_u{}^{\triangleleft}!$ *in the localized version.* $\mathsf{TH}_{\mathcal{H}}$ *maintains arrays* $(init^*_{u,v})_{u,v\in\mathcal{M}}$ *and* $(stopped^*_u)_{u\in\mathcal{M}}$ *over* $\{0,1\}$*, both initialized with* 0 *everywhere, and an array* $(deliver^*_{u,v})_{u,v\in\mathcal{M}}$ *of lists, all initially empty. The state-transition function of* $\mathsf{TH}_{\mathcal{H}}$ *is defined by the following rules:*

Send initialization. *On input* (init) *at* $\mathsf{in}_u?$*: If* $stopped^*_u = 0$ *and* $init^*_{u,u} = 0$*, set* $init^*_{u,u} := 1$ *and output* (init) *at* $\mathsf{to\_adv}_u!$*.*

Receive initialization. *On input* (init, u) *at* $\mathsf{from\_adv}_v?$ *with* $u \in \mathcal{M}, v \in \mathcal{H}$*: If* $stopped^*_v = 0$ *and* $init^*_{u,v} = 0$ *and* $[u \in \mathcal{H} \Rightarrow init^*_{u,u} = 1]$*, set* $init^*_{u,v} := 1$ *and output* (init, u) *at* $\mathsf{out}_v!$*.*

Send. *On input* (send, m, v) *at* $\mathsf{in}_u?$ *with* $m \in \Sigma^+$*,* $l := \mathsf{len}(m) \leq L(k)$*, and* $v \in \mathcal{M} \setminus \{u\}$*: If* $stopped^*_u = 0$*,* $init^*_{u,u} = 1$*, and* $init^*_{v,u} = 1$*: If* $v \in \mathcal{A}$ *then* $\{$ *output* (send, m, v) *at* $\mathsf{to\_adv}_u!$ $\}$ *else* $\{i := \mathsf{size}(deliver^*_{u,v}) + 1$*;* $deliver^*_{u,v}[i] := m$*; output* $(\mathsf{send\_blindly}, i, l, v)$ *at* $\mathsf{to\_adv}_u!$ $\}$*.*

Receive from honest party u***.*** *On input* $(\mathsf{receive\_blindly}, u, i)$ *at* $\mathsf{from\_adv}_v?$ *with* $u, v \in \mathcal{H}$*: If* $stopped^*_v = 0$*,* $init^*_{v,v} = 1$*,* $init^*_{u,v} = 1$*, and* $m := deliver^*_{u,v}[i] \neq \downarrow$*, then output* $(\mathsf{receive}, u, m)$ *at* $\mathsf{out}_v!$*.*

Receive from dishonest party u***.*** *On input* $(\mathsf{receive}, u, m)$ *at* $\mathsf{from\_adv}_v?$ *with* $u \in \mathcal{A}$*,* $m \in \Sigma^+$*,* $\mathsf{len}(m) \leq L(k)$*, and* $v \in \mathcal{H}$*: If* $stopped^*_v = 0$*,* $init^*_{v,v} = 1$ *and* $init^*_{u,v} = 1$*, then output* $(\mathsf{receive}, u, m)$ *at* $\mathsf{out}_v!$*.*

Stop. *On input* (stop) *at* $\mathsf{from\_adv}_u?$ *with* $u \in \mathcal{H}$*, set* $stopped^*_u = 1$ *and output* (stop) *at* $\mathsf{out}_u!$*.*

$\diamond$

As the computational costs of each transition is at most linear in the accumulated input size,[1] the following lemma is clearly true:

Lemma 3.1 *Each ideal system* $Sys^{\mathsf{secmsg},\mathsf{ideal}}_{n,L}$ *is weakly polynomial-time.* $\square$

$\mathsf{TH}_{\mathcal{H}}$ is as abstract as we hoped for: It is deterministic and contains no cryptographic objects. Its state-transition function should be easy to express in every formal language for automata provided it allows our data structures, which most such languages do.

[1] At first sight, the computation cost of a transition even seems to be linear only in the *current* input size. However, the growing state makes book-keeping more expensive over time.

3.2 Real System

The real system uses asymmetric encryption and digital signatures as cryptographic primitives; their definitions are briefly repeated in Section 3.2.1. The message transmission scheme itself is described in Section 3.2.2.

3.2.1 Primitives Used

Let, without loss of generality, $\mathsf{false} \notin \Sigma^+$.

Definition 3.1 *(Encryption Schemes) A* public-key encryption scheme *is a triple* $(\mathsf{gen_E}, \mathsf{E}, \mathsf{D})$ *of polynomial-time algorithms, where* $\mathsf{gen_E}$ *and* E *are probabilistic.* $\mathsf{gen_E}$ *takes an input* 1^k *with* $k \in \mathbb{N}$ *and outputs a pair* (ske, pke) *of a secret decryption key and a public encryption key in* Σ^+. E *takes such a public key and a message* $m \in \Sigma^+$ *as inputs and produces a ciphertext in* Σ^+*; we write this* $c \leftarrow \mathsf{E}_{pke}(m)$. *Similarly, we write decryption as* $m := \mathsf{D}_{ske}(c)$. *The result may be* false *for wrong ciphertexts. For a correctly generated key pair and ciphertext, decryption yields the original message.*

We assume without loss of generality that the length of the public key is a function of k*, that of* c *a function of* k *and* $\mathsf{len}(m)$*, and that decryption never increases the length.* ◇

This definition allows arbitrarily long messages to be encrypted. Then, however, the length cannot be hidden. The following security definition means that any two equal-length messages are indistinguishable even in adaptive chosen-ciphertext attacks. Indistinguishability was introduced in [26], chosen-ciphertext security in [50] and formalized as "IND-CCA2" in [12]. It is the accepted definition for general-purpose encryption. We only use our notation for interacting machines.

Definition 3.2 *(Encryption Security) Given an encryption scheme, a* decryptor machine Dec *with one input and one output port, and with variables* ske, pke, c *initialized with* $\downarrow$*, is defined by the following rules:*

- *First set* $(ske, pke) \leftarrow \mathsf{gen_E}(1^k)$ *and output* pke.

- *On input* (enc, m_0, m_1) *(intuitively a pair of messages an adversary hopes to be able to distinguish), and if* $\mathsf{len}(m_0) = \mathsf{len}(m_1)$ *and* $c = \downarrow$*, randomly choose a bit* $b \in_{\mathcal{R}} \{0,1\}$ *and store and output the encryption* $c \leftarrow \mathsf{E}_{pke}(m_b)$*.*

- *On input* (dec, c_j) *and if* $c_j \neq c$*, decrypt* c_j *with ske and return the result.*

The encryption scheme is called indistinguishable under adaptive chosen-ciphertext attack *if for every probabilistic polynomial-time machine* $\mathsf{A}_{\mathsf{enc}}$ *that interacts with* Dec *and finally outputs a bit* b^* *(meant as a guess at* b*), the probability of the event* $b^* = b$ *is bounded by* $1/2 + 1/\mathsf{poly}(k)$*. In our terminology,* $[\{\mathsf{Dec}, \mathsf{A}_{\mathsf{enc}}\}]$ *is a closed collection and the event is a predicate on the runs; hence the probability is well-defined.* ◇

Secure signature schemes often have memory. We model this by a counter. This covers schemes storing a path in a tree or other random values, because all random values needed can be seen as part of the secret key. (The definition in [28] can be interpreted as even allowing that a signature divulges the history of messages signed before; we exclude this because it is unusual and would make our real system more complicated.)

Definition 3.3 *(Signature Schemes) A* signature scheme *is a triple* $(\mathsf{gen}_{\mathsf{S}}, \mathsf{sign}, \mathsf{test})$ *of polynomial-time algorithms, where* $\mathsf{gen}_{\mathsf{S}}$ *and* sign *are probabilistic.* $\mathsf{gen}_{\mathsf{S}}$ *takes an input* $(1^k, 1^s)$ *with* $k, s \in \mathbb{N}$*, where* s *denotes the desired maximum number of signatures, and outputs a pair* (sks, pks) *of a secret signing key and a public test key in* Σ^+*.* sign *takes such a secret key, a counter* $sc \in \{1, \ldots, s\}$*, and a message* $m \in \Sigma^+$ *as inputs and produces a signature in* Σ^+*. We write this* $sig \leftarrow \mathsf{sign}_{sks,sc}(m)$*. We assume without loss of generality that* sig *is of the form* (m, sig')*. Similarly, we write verification as* $m := \mathsf{test}_{pks}(sig)$*. The result may be* false*; if not we say that the signature is* valid*. For a correctly generated key pair and signature, the test yields the original message. It only yields* $m \neq \mathsf{false}$ *if* sig *is of the form* (m, sig')*.*

We assume without loss of generality that the length of the public key is a function of k *and* s*, and that of* sig *a function* $\mathsf{sig\_len}(k, s, \mathsf{len}(m))$*.* ◇

Security of a signature scheme is defined against existential forgery under adaptive chosen-message attacks [28].

Definition 3.4 *(Signature Security) Given a signature scheme and a polynomial $s \in \mathbb{N}[x]$, the* signer machine Sig_s *is defined as follows: It has one input and one output port, variables sks, pks initialized with $\downarrow$ and a counter sc initialized with 0, and the following transition rules:*

- *First generate a key pair, $(sks, pks) \leftarrow \mathsf{gen_S}(1^k, 1^{s(k)})$, and output pks.*
- *On input (sign, m_j), and if $sc < s(k)$, set $sc := sc+1$ and return $sig_j \leftarrow \mathsf{sign}_{sks,sc}(m_j)$.*

The signature scheme is called existentially unforgeable under adaptive chosen-message attack *if for every polynomial s and every probabilistic polynomial-time machine $\mathsf{A_{sig}}$ that interacts with Sig_s and finally outputs a value sig (meant as a forged signature), the probability that $\mathsf{test}_{pks}(sig)$ gives a message $m \neq \mathsf{false}$ with $m \neq m_j$ for all j is negligible (in k). As in Definition 3.2, this probability is well-defined.* ◇

Lemma 3.2 (Skipping signatures) *Without loss of generality, we can assume that a secure signature scheme is* skipping secure, *meaning that $\mathsf{A_{sig}}$ may choose the values sc in the attack, but Sig_s verifies that they are strictly increasing and do not exceed $s(k)$.* □

Proof. "Natural" secure signature schemes will already fulfill this; otherwise encode the messages from Σ^+ into Σ^+ such that there is an unused message m^* (e.g., prepend a bit), and let Sig_s sign m^* for a value sc if $\mathsf{A_{sig}}$ skips it. ■

3.2.2 Real System for Secure Message Transmission

The real system has the same parameters as the ideal one, except that, to make it polynomial-time, it also has a bound on the number of transactions per machine.

Scheme 3.2 (Real System for Secure Message Transmission) *Let parameters n, L, and thus $\mathcal{M}$, be given as in Scheme 3.1, and additionally a polynomial $T \in \mathbb{N}[x]$. Also let an encryption scheme and a signature scheme be given. The system is a standard (or localized) cryptographic system*

$$Sys^{\mathsf{secmsg,real}}_{n,L,T} := StanStat(n, \hat{M}^*, \mathcal{ACC}, \chi)$$

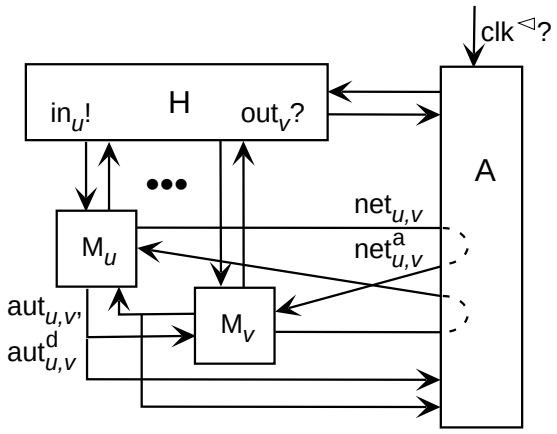

Figure 3.1: Sketch of a derived real structure for secure message transmission. All connections are clocked by A, except those between H and M_u or M_v in the localized case. Indices $\mathcal{H}$ are omitted; also in the following figures.

(Definition 2.18) with $\mathcal{ACC} = \mathcal{P}(\mathcal{M})$ and the following machines M_u (fulfilling Definition 2.17) and channel model χ:

The ports of M_u are $\{\mathsf{in}_u?, \mathsf{out}_u!\} \cup \{\mathsf{net}_{u,v}!, \mathsf{net}_{v,u}?|v \in \mathcal{M}\} \cup \{\mathsf{aut}_{u,v}!, \mathsf{aut}_{v,u}?|v \in \mathcal{M}\}$ and $\mathsf{out}_u{}^{\triangleleft}!$ in the localized case. High-level connections $\{\mathsf{net}_{u,v}!, \mathsf{net}_{u,v}?\}$ are insecure and $\{\mathsf{aut}_{u,v}!, \mathsf{aut}_{u,v}?\}$ are authenticated (for key exchange); this is sketched in Figure 3.1.

Each M_u maintains an array $(init_{v,u})_{v\in\mathcal{M}}$ of lists, which are initially empty, and counters sc_u, tc_u (for signatures and transitions) initialized with 0. Final states are those with $tc_u = T(k)$. The state-transition function of M_u is defined by the following rules (where automatically nothing happens in a finite state):

General. *The transaction counter tc_u is incremented at the beginning of every transition with non-empty input. If then $tc_u = T(k)$, the transition finishes with the output (stop) at $\mathsf{out}_u!$.*

Send initialization. *On input (init) at $\mathsf{in}_u?$: If $init_{u,u} = ()$, generate $(sks_u, pks_u) \leftarrow \mathsf{gen}_\mathsf{S}(1^k, 1^{T(k)})$ and $(ske_u, pke_u) \leftarrow \mathsf{gen}_\mathsf{E}(1^k)$, set $init_{u,u} := (sks_u, ske_u)$, and output (pks_u, pke_u) at all ports $\mathsf{aut}_{u,v}!$.*

Receive initialization. *On input (pks_v, pke_v) at $\mathsf{aut}_{v,u}?$ with the correct lengths for public keys: If $init_{v,u} = ()$, then set $init_{v,u} := (pks_v, pke_v)$ and output (init, v) at $\mathsf{out}_u!$.*

Send. *On input* (send, m, v) *at* $\mathsf{in}_u?$ *with* $m \in \Sigma^+$, $\mathsf{len}(m) \leq L(k)$, *and* $v \in \mathcal{M} \backslash \{u\}$*: If* $init_{u,u}$ *and* $init_{v,u}$ *are of the form* (sks_u, ske_u) *and* (pks_v, pke_v), *respectively, then set* $sc_u := sc_u + 1$ *and*

$$c \leftarrow \mathsf{E}_{pke_v}(\mathsf{sign}_{sks_u, sc_u}(u, m, v)), \qquad (*)$$

and output c *at* $\mathsf{net}_{u,v}!$.

Receive. *On input* c *at* $\mathsf{net}_{v,u}?$ *within the length bound for a network message as in* $(*)$*: If* $init_{u,u}$ *and* $init_{v,u}$ *are of the form* (sks_u, ske_u) *and* (pks_v, pke_v), *respectively, then parse* c *in the form* $\mathsf{E}_{pke_u}(\mathsf{sign}_{sks_v, sc_v}(v, m, u))$. *More precisely:*

a) $sig := \mathsf{D}_{ske_u}(c)$. *Abort if the result is* false.

b) $m' := \mathsf{test}_{pks_v}(sig)$. *Abort if the result is* false.

c) $(v', m, u') := m'$. *Abort if this fails or* $(v', u') \neq (v, u)$.

If this succeeds and $\mathsf{len}(m) \leq L(k)$, *output* $(\mathsf{receive}, v, m)$ *at* $\mathsf{out}_u!$.

◇

Lemma 3.3 *The real system* $Sys^{\mathsf{secmsg,real}}_{n,L,T}$ *is polynomial-time.* □

Proof. Each machine M_u only makes a polynomial number $T(k)$ of transitions, controlled by tc_u. In each transition, it only accepts polynomial-length inputs (controlled by the length functions implicit in the domain checks) and performs polynomial-time algorithms. ■

The security of this real system with respect to our ideal system is proven in Section 3.4. It is based on a new theorem about the security of encryption in a reactive setting, which we present first.

3.3 Public-key Encryption in a Reactive Multi-user Setting

An essential cryptographic part of the security proof of the real system is captured by Theorem 3.1, which extends the standard notion of chosen-ciphertext security of public-key encryption to a reactive multi-user setting,

using a simulatability definition. A similar multi-user scenario has been considered in [10], but no decryption request for any of the ciphertexts produced by a correct machine is allowed there. However, in a reactive scenario like ours, most secret messages are also decrypted by some correct machine and partial knowledge may leak; hence the theorem is not immediately applicable. We therefore define ideal machines $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ that encrypt simulated messages, but honor *all* decryption requests by table look-up of the intended messages. Then we can show simulatability in our usual sense. (However, this ideal system is not yet "abstract" in our sense, in contrast to that for secure message transmission.)

Remark 3.1. For cryptographers, our theorem can also be seen as a formalization of the notion of "a standard hybrid argument." For a passive setting this was done by Theorem 3.6 of [22]. However, in a reactive setting one has to switch over from a real state to a "corresponding" ideal state, and there is no general definition for this. In particular, it must be made clear how decryption is handled in the hybrids. This is now well-defined at least for systems that use an encryption system only such that they can be rewritten with our real encryption system. ∘

As parameters, the following systems have polynomial bounds s_{keys} on the number of keys to be generated in the system and s_{encs} on the number of encryptions per key.

Scheme 3.3 (Ideal and "Real" Encryption Systems) *Let an encryption scheme* $(\mathsf{gen}_\mathsf{E}, \mathsf{E}, \mathsf{D})$ *and parameters* $n \in \mathbb{N}$ *and* s_{keys}, $s_{encs} \in \mathbb{N}[x]$ *be given. For every* $l \in \mathbb{N}$*, let* $m_{sim,l}$ *denote the fixed message* $\mathsf{0}^l$*. We define two systems*

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

For every $\mathcal{H}$*, the ports are*

- $Ports_{\mathsf{Enc}_\mathcal{H}} := Ports_{\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}} := \{\mathsf{in}_{\mathsf{enc},u}?, \mathsf{out}_{\mathsf{enc},u}!, \mathsf{out}_{\mathsf{enc},u}{}^{\triangleleft}! \mid u \in \mathcal{H}\}$,
- $S^c_{\mathsf{enc},\mathcal{H}} := \{\mathsf{in}_{\mathsf{enc},u}!, \mathsf{in}_{\mathsf{enc},u}{}^{\triangleleft}!, \mathsf{out}_{\mathsf{enc},u}? \mid u \in \mathcal{H}\}$.

Both machines maintain a key counter $kc \in \mathbb{N}$*, initially* $kc := 0$*, and initially empty lists keys and ciphers. The latter is used for the look-up of intended cleartexts in the ideal system. The transition functions are given by the following rules. Let* $\mathsf{in}_{\mathsf{enc},u}?$ *be the port where the current input is made; the resulting output goes to* $\mathsf{out}_{\mathsf{enc},u}!$ *(with* $\mathsf{out}_{\mathsf{enc},u}{}^{\triangleleft}! := 1$ *by the conventions).*

- *On input* $(\mathsf{generate})$ *for* $\mathsf{Enc}_{\mathcal{H}}$ *and* $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$*: If* $kc < s_{keys}(k)$ *then* $\{$ $kc := kc + 1$*;* $(ske, pke) \leftarrow \mathsf{gen}_{\mathsf{E}}(1^k)$*; add* $(u, kc, ske, pke, 0)$ *to keys; output pke* $\}$ *else output* $\downarrow$*.*

- *On input* $(\mathsf{encrypt}, pke, m)$ *with* $pke, m \in \Sigma^+$ *and pke of correct length: If* $\neg\exists v, kc, ske, s_{pke} : (v,\ kc,\ ske,\ pke,\ s_{pke}) \in keys \wedge s_{pke} < s_{encs}(k)$ *then output* $\downarrow$ *else*

  - *for* $\mathsf{Enc}_{\mathcal{H}}$*:* $s_{pke} := s_{pke} + 1$*; output* $c \leftarrow \mathsf{E}_{pke}(m)$*;*
  - *for* $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$*:* $s_{pke} := s_{pke} + 1$*; output* $c \leftarrow \mathsf{E}_{pke}(m_{sim,\mathsf{len}(m)})$*; add* (m, pke, c) *to ciphers.*

- *On input* $(\mathsf{decrypt}, pke, c)$ *with* $pke, c \in \Sigma^+$ *(note that pke is used as a designator of the desired private key): If* $\neg\exists kc,\ ske,\ s_{pke} : (u,\ kc,\ ske,\ pke,\ s_{pke}) \in keys$ *then output* $\downarrow$ *else*

  - *for* $\mathsf{Enc}_{\mathcal{H}}$*: output* $m := \mathsf{D}_{ske}(c)$*;*
  - *for* $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$*: If* $\exists m : (m, pke, c) \in ciphers$ *then output* m *else output* $m := \mathsf{D}_{ske}(c)$*.*

$\diamond$

In this definition, $\bar{S}_{\mathsf{enc},\mathcal{H}} = \emptyset$, i.e., the adversary is not connected with the correct machines except via or in the place of H. Both $\mathsf{Enc}_{\mathcal{H}}$ and $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ limit the capability to decrypt: If $(u, kc, ske, pke, 0)$ was added to *keys* due to an input $(\mathsf{generate})$ at port $\mathsf{in}_{\mathsf{enc},u}?$, then $(\mathsf{decrypt}, pke, c)$ has an effect only if it is entered at the same port.

Similar to the ideal system, we can also state the following lemma:

Lemma 3.4 *The systems* $Sys^{\mathsf{enc},\mathsf{real}}_{n,s_{keys},s_{encs}}$ *and* $Sys^{\mathsf{enc},\mathsf{sim}}_{n,s_{keys},s_{encs}}$ *are weakly polynomial-time.* $\square$

Theorem 3.1 (Reactive Multi-user Public-key Encryption) *For all* $n \in \mathbb{N}$, $s_{keys}, s_{encs} \in \mathbb{N}[x]$, *we have*

$$Sys^{\mathsf{enc,real}}_{n,s_{keys},s_{encs}} \geq^{f,\mathsf{poly}}_{\mathsf{sec}} Sys^{\mathsf{enc,sim}}_{n,s_{keys},s_{encs}}$$

for the canonical mapping f, *provided the encryption scheme used is secure (Definition 3.2). This holds even with blackbox simulatability.* □

Proof. Let n, s_{keys}, s_{encs} and $\mathcal{H}$ and thus one structure of one system be fixed. We always set $\mathsf{A}_2 = \mathsf{A}_1 =: \mathsf{A}$; this is clearly a blackbox construction. The proof is a hybrid argument as first used in [26], i.e., we construct intermediate systems that differ only in one encryption each.

For every $k \in \mathbb{N}$ let $\mathcal{I}_k := (\{1, \ldots, s_{keys}(k)\} \times \{1, \ldots, s_{encs}(k)\}) \cup \{\alpha\}$, let $<_k$ be the lexicographic order on $\mathcal{I}_k \setminus \{\alpha\}$, and $\alpha \leq_k t$ for all $t \in \mathcal{I}_k$. Let $\mathsf{pred}_k(t)$ be the predecessor of $t \in \mathcal{I}_k$ relative to $<_k$ and $\omega(k) := (s_{keys}(k), s_{encs}(k))$.

For every $k \in \mathbb{N}$ and $t \in \mathcal{I}_k$, we define a hybrid machine $\mathsf{Enc}_{k,t,\mathcal{H}}$. It is like $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ with fixed initial input 1^k, except whenever $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ carries out $c \leftarrow \mathsf{E}_{pke}(m_{sim,\mathsf{len}(m)})$: Let $t' := (kc, s_{pke})$ for the values kc, s_{pke} at that moment.

- If $t' \leq_k t$, it sets $c \leftarrow \mathsf{E}_{pke}(m)$ like $\mathsf{Enc}_{\mathcal{H}}$;
- if $t' >_k t$, it sets $c \leftarrow \mathsf{E}_{pke}(m_{sim,\mathsf{len}(m)})$ like $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$.

Clearly, each $\mathsf{Enc}_{k,\alpha,\mathcal{H}}$ works like $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ on input 1^k. Furthermore, $\mathsf{Enc}_{k,\omega(k),\mathcal{H}}$ works like $\mathsf{Enc}_{\mathcal{H}}$ on input 1^k: $\mathsf{Enc}_{\mathcal{H}}$ and $\mathsf{Enc}_{k,\omega(k),\mathcal{H}}$ produce identical outputs for inputs $(\mathsf{generate})$ and $(\mathsf{encrypt}, pke, m)$. Now consider an input $(\mathsf{decrypt}, pke, c)$ at $\mathsf{in}_{\mathsf{enc},u}$? such that $\exists kc, ske, s_{pke} : (u, kc, ske, pke, s_{pke}) \in keys$ (otherwise both output $\downarrow$). If there is no tuple (m, pke, c) in $ciphers$, both output $\mathsf{D}_{ske}(c)$. If there is, then c has been generated by $\mathsf{Enc}_{k,\omega(k),\mathcal{H}}$ as $\mathsf{E}_{pke}(m)$. Thus $\mathsf{D}_{ske}(c) = m$, and both machines output m.

Now assume for contradiction that the theorem is wrong for the given parameters and for polynomial-time machines A and H. Let $conf_{\mathsf{real}} := (\{\mathsf{Enc}_{\mathcal{H}}\}, S_{\mathsf{enc},\mathcal{H}}, \mathsf{H}, \mathsf{A})$ and $conf_{\mathsf{sim}} := (\{\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}\}, S_{\mathsf{enc},\mathcal{H}}, \mathsf{H}, \mathsf{A})$. Furthermore, for all k, t, let $coll_{k,t}$ be the collection $\{\mathsf{Enc}_{k,t,\mathcal{H}}, \mathsf{H}, \mathsf{A}\}$, where the initial inputs are always 1^k. Thus we assume $view_{conf_{\mathsf{real}}}(\mathsf{H}) \not\approx_{\mathsf{poly}} view_{conf_{\mathsf{sim}}}(\mathsf{H})$, and this implies

$$(view_{coll_{k,\omega(k)}}(\mathsf{H}))_{k\in\mathbb{N}} \not\approx_{\mathsf{poly}} (view_{coll_{k,\alpha}}(\mathsf{H}))_{k\in\mathbb{N}}.$$

We abbreviate $view_{k,t} := view_{coll_{k,t}}(\mathsf{H})$. Hence there exists a distinguisher Dis and $p \in \mathbb{N}[x]$ such that for all k in an infinite set $\mathcal{K} \subseteq \mathbb{N}$,

$$|P(\mathsf{Dis}(view_{k,\omega(k)}) = 1) - P(\mathsf{Dis}(view_{k,\alpha}) = 1)| > \frac{1}{p(k)}.$$

(For brevity, we always omit the input 1^k of Dis.)

We construct two almost identical adversaries $\mathsf{A}_{\mathsf{enc}+}$ and $\mathsf{A}_{\mathsf{enc}-}$ on the encryption scheme; we write $\mathsf{A}_{\mathsf{enc}}$ where statements hold for both. On input 1^k, $\mathsf{A}_{\mathsf{enc}}$ randomly selects $t \in_{\mathcal{R}} \mathcal{I}_k \setminus \{\alpha\}$, say, $t = (kc, s')$, and receives a public key pke from the decryptor machine Dec. Then it simulates $coll_{k,t}$ with the following exceptions:

- If H makes the kc-th input ($\mathsf{generate}$), say at port $\mathsf{in}_{\mathsf{enc},u}?$, then $\mathsf{A}_{\mathsf{enc}}$ only adds $(u, kc, 0, pke, 0)$ to $keys$. Instead of operations $m := \mathsf{D}_{ske}(c)$ corresponding to this entry (identified by pke), it queries Dec.
- If H makes an input $(\mathsf{encrypt}, pke, m)$ which gets the index $t = (kc, s')$, i.e., $\mathsf{A}_{\mathsf{enc}}$ finds the entry $(u, kc, 0, pke, s' - 1)$ in $keys$, then $\mathsf{A}_{\mathsf{enc}}$ sends $(\mathsf{enc}, m_0, m_1) := (\mathsf{enc}, m, m_{sim,\mathsf{len}(m)})$ to Dec. Recall that Dec flips a bit $b \in_{\mathcal{R}} \{0, 1\}$ and returns $c \leftarrow \mathsf{E}_{pke}(m_b)$. $\mathsf{A}_{\mathsf{enc}}$ adds (m, pke, c) to $ciphers$.

At the end, $\mathsf{A}_{\mathsf{enc}}$ runs Dis on the resulting view of H, which yields a bit b_k^*. $\mathsf{A}_{\mathsf{enc}+}$ outputs b_k^*, whereas $\mathsf{A}_{\mathsf{enc}-}$ outputs $1 - b_k^*$.

First, $\mathsf{A}_{\mathsf{enc}}$ is indeed polynomial-time. (Given the polynomial runtime bounds on A and H, one easily sees that there is a joint polynomial bound (in k) on the runtime of all collections $coll_{k,t}$.) Secondly, $\mathsf{A}_{\mathsf{enc}}$ never asks Dec to decrypt the ciphertext c from Dec (which Dec would refuse), because $\mathsf{A}_{\mathsf{enc}}$ will find (m, pke, c) in $ciphers$ and output m. Thus the simulation of $coll_{k,t}$ is perfect.

Let $view_k^{(b)}$ denote the random variable of the view of H in $\mathsf{A}_{\mathsf{enc}}$ for parameter k and a specific bit b. This is well-defined because b is chosen independently. For $b = 0$ the simulated run is generated like a run of $coll_{k,t}$ and for $b = 1$ like a run of $coll_{k,\mathsf{pred}_k(t)}$. With $w(k) := s_{keys}(k)s_{encs}(k)$ we get:

$$\begin{aligned} P(\mathsf{Dis}(view_k^{(0)}) = 1) &= \frac{1}{w(k)} \sum_{t \in \mathcal{I}_k \setminus \{\alpha\}} P(\mathsf{Dis}(view_{k,t}) = 1), \\ P(\mathsf{Dis}(view_k^{(1)}) = 1) &= \frac{1}{w(k)} \sum_{t \in \mathcal{I}_k \setminus \{\alpha\}} P(\mathsf{Dis}(view_{k,\mathsf{pred}_k(t)}) = 1). \end{aligned}$$

For all $k \in \mathcal{K}$, this implies

$$\begin{aligned} & |P(\mathsf{Dis}(view_k^{(0)}) = 1) - P(\mathsf{Dis}(view_k^{(1)}) = 1)| \\ = \; & \frac{1}{w(k)} \, |P(\mathsf{Dis}(view_{k,\omega(k)}) = 1) - P(\mathsf{Dis}(view_{k,\alpha}) = 1)| \\ > \; & \frac{1}{w(k)p(k)}. \end{aligned}$$

Thus

$$\begin{aligned} P(b_k^* = b) &= P(b = 0 \wedge \mathsf{Dis}(view_k^{(b)}) = 0) + P(b = 1 \wedge \mathsf{Dis}(view_k^{(b)}) = 1) \\ &= \frac{1}{2} \, (P(\mathsf{Dis}(view_k^{(0)}) = 0) + P(\mathsf{Dis}(view_k^{(1)}) = 1)) \\ &= \frac{1}{2} + \frac{1}{2} \, (P(\mathsf{Dis}(view_k^{(1)}) = 1) - P(\mathsf{Dis}(view_k^{(0)}) = 1)). \end{aligned}$$

This implies for all $k \in \mathcal{K}$

$$P(b_k^* = b) > \frac{1}{2} + \frac{1}{2w(k)p(k)} \quad \text{or} \quad P(b_k^* = b) < \frac{1}{2} - \frac{1}{2w(k)p(k)}.$$

Thus the success probability of either $\mathsf{A}_{\mathsf{enc}+}$ or $\mathsf{A}_{\mathsf{enc}-}$ is larger than $1/2 + 1/(2w(k)p(k))$ for all k in an infinite subset $\mathcal{K}' \subseteq \mathcal{K}$. This is the desired contradiction to the security of the encryption system. ∎

3.4 Security of the Real System

We show that Scheme 3.2 is at least as secure as the ideal Scheme 3.1.

Theorem 3.2 (Secure Message Transmission) *For all $n \in \mathbb{N}$ and $L, T \in \mathbb{N}[x]$, $Sys_{n,L,T}^{\mathsf{secmsg,real}} \geq_{\mathsf{sec}}^{f,\mathsf{poly}} Sys_{n,L}^{\mathsf{secmsg,ideal}}$ for the canonical mapping f from Definition 2.19, provided the encryption and signature schemes used are secure (Definitions 3.2 and 3.4). This holds even with blackbox simulatability.* □

We prove this theorem in the following steps: In Part A we rewrite the real system to use a reactive encryption system $Sys_{n,s_{keys},s_{encs}}^{\mathsf{enc,real}}$ from Scheme 3.3 instead of performing the original encryption algorithms. This gives an intermediate system $Sys_{n,L,T}^{\mathsf{secmsg,Enc}}$, and we show that

$$Sys_{n,L,T}^{\mathsf{secmsg,real}} \geq_{\mathsf{sec}}^{\mathsf{perf}} Sys_{n,L,T}^{\mathsf{secmsg,Enc}}.$$

In Part B, we replace the real reactive encryption system by the ideal one to get a new system $Sys^{\mathsf{secmsg,Encsim}}_{n,L,T}$. With the composition theorem, we obtain

$$Sys^{\mathsf{secmsg,Enc}}_{n,L,T} \geq^{\mathsf{poly}}_{\mathsf{sec}} Sys^{\mathsf{secmsg,Encsim}}_{n,L,T}.$$

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

$$Sys^{\mathsf{secmsg,Encsim}}_{n,L,T} \geq^{\mathsf{poly}}_{\mathsf{sec}} Sys^{\mathsf{secmsg,ideal}}_{n,L}.$$

Theorem 3.2 follows with the transitivity of $\geq_{\mathsf{sec}}$ (Lemma 2.7) and Lemma 2.6.

3.4.1 Rewriting the Real System

We first define the intermediate system $Sys^{\mathsf{secmsg,Enc}}_{n,L,T}$. Its structures are of the form $(\hat{M}'_{\mathcal{H}}, S_{\mathcal{H}})$ with $\hat{M}'_{\mathcal{H}} = \{\mathsf{Enc}_{\mathcal{H}}\} \cup \{\mathsf{M}'_{u,\mathcal{H}} | u \in \mathcal{H}\}$, see Figure 3.2, where $\mathsf{Enc}_{\mathcal{H}}$ is the machine from $Sys^{\mathsf{enc,real}}_{n,s_{keys},s_{encs}}$ with n as in the overall system and $s_{keys}(k) := n$, $s_{encs}(k) := nT(k)$.

Each machine $\mathsf{M}'_{u,\mathcal{H}}$ equals $\mathsf{M}_{u,\mathcal{H}}$ with the following modifications:

- It has additional ports $\mathsf{in}_{\mathsf{enc},u}!$, $\mathsf{in}_{\mathsf{enc},u}{}^{\triangleleft}!$, $\mathsf{out}_{\mathsf{enc},u}?$ to connect to $\mathsf{Enc}_{\mathcal{H}}$.
- In "Send initialization." Instead of calling $\mathsf{gen}_{\mathsf{E}}$, output (generate) at $\mathsf{in}_{\mathsf{enc},u}!$ and wait for a result of the correct length for a public key at $\mathsf{out}_{\mathsf{enc},u}?$. (More precisely enter a state (wait, init) where all other inputs are ignored. As one easily sees from the scheduling that only this input can arrive next, we treat the wait state together with the previous state; also in the following cases. tc_u is not incremented after a wait state.) Use this result as pke_u. In $init_{u,u}$ store pke_u instead of ske_u. As a second change, this transition, except for the final sending of keys, is also carried out if tc_u was $T(k) - 1$. (This gives greater similarity with the state in $\mathsf{TH}_{\mathcal{H}}$ below.)
- In "Send." If $v \in \mathcal{H}$, instead of the computation of c, compute $sig \leftarrow \mathsf{sign}_{sks_u,sc_u}(u, m, v)$ and output (encrypt, pke_v, sig) at $\mathsf{in}_{\mathsf{enc},u}!$. Wait for a result of the correct length for a corresponding ciphertext at $\mathsf{out}_{\mathsf{enc},u}?$

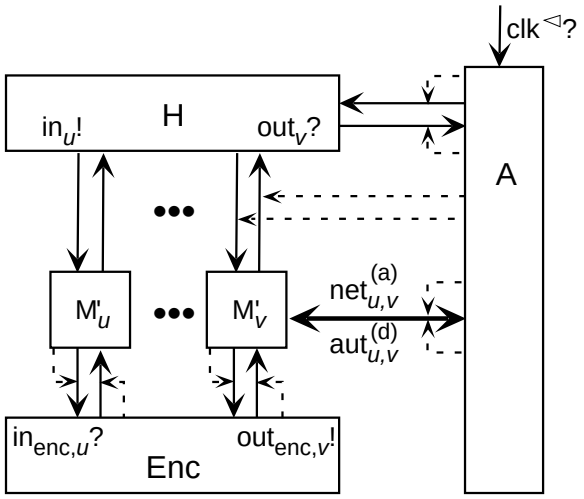

Figure 3.2: Real system rewritten with encryption subsystem (standard case).

and use it as c. This transition, except for the final sending of c, is also carried out if tc_u was $T(k) - 1$.

- In "Receive." Instead of Step a) of parsing, output $(\mathsf{decrypt}, pke_u, c)$ at $\mathsf{in}_{\mathsf{enc},u}$!. Wait for a result of at most $\mathsf{len}(c)$ (or false) at $\mathsf{out}_{\mathsf{enc},u}$? and use it as sig.

As this intermediate system is only used in the proof, it is no problem that $\mathsf{M}'_{u,\mathcal{H}}$ makes an explicit distinction using $\mathcal{H}$. Each machine $\mathsf{M}'_{u,\mathcal{H}}$ only accepts a fixed polynomial number of inputs, verifies a polynomial length bound for each, and executes polynomial-time algorithms. Hence the following lemma holds.

Lemma 3.5 *The machines $\mathsf{M}'_{u,\mathcal{H}}$ are polynomial-time.* □

The views of A and H in a configuration $(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}, \mathsf{H}, \mathsf{A})$ and $(\hat{M}'_{\mathcal{H}}, S_{\mathcal{H}}, \mathsf{H}, \mathsf{A})$ are identical (perfectly indistinguishable), because the actions of $\mathsf{Enc}_{\mathcal{H}}$ on the given inputs are precisely what $\mathsf{M}_{u,\mathcal{H}}$ would have done at this point, and A can neither observe nor influence the interaction between $\mathsf{Enc}_{\mathcal{H}}$ and $\mathsf{M}'_{u,\mathcal{H}}$. In particular $s_{keys}(k) = n$ is not exceeded because each $\mathsf{M}_{u,\mathcal{H}}$ inputs ($\mathsf{generate}$) at most once, controlled by $init_{u,u}$, and $s_{encs}(k) = nT(k)$ is not

exceeded for any key because each $\mathsf{M}_{u,\mathcal{H}}$ inputs $(\mathsf{encrypt}, \ldots)$ at most $T(k)$ times, controlled by tc_u. Thus we have

$$Sys^{\mathsf{secmsg,real}}_{n,L,T} \geq^{\mathsf{perf}}_{\mathsf{sec}} Sys^{\mathsf{secmsg,Enc}}_{n,L,T}.$$

3.4.2 Replacing the Encryption System

In $Sys^{\mathsf{secmsg,Enc}}_{n,L,T}$, we want to replace each machine $\mathsf{Enc}_{\mathcal{H}}$ by $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ to get a new system $Sys^{\mathsf{secmsg,Encsim}}_{n,L,T}$.

We consider $Sys^{\mathsf{secmsg,Enc}}_{n,L,T}$ as a composition of $Sys_0 := Sys^{\mathsf{enc,real}}_{n,n,nT}$ and a system Sys_1 that is naturally defined as the structures without $\mathsf{Enc}_{\mathcal{H}}$: The specified ports are those of $Sys^{\mathsf{secmsg,real}}_{n,L,T}$ plus the low-level complements of the ports of $\mathsf{Enc}_{\mathcal{H}}$. Then the conditions of Definition 2.22 are fulfilled. As each machine $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ has the same ports as $\mathsf{Enc}_{\mathcal{H}}$, the definition of $Sys^{\mathsf{secmsg,Encsim}}_{n,L,T}$ as a composition of $Sys'_0 := Sys^{\mathsf{enc,sim}}_{n,n,nT}$ and the same Sys_1 is clear and the preconditions of the composition theorem, Theorem 2.1, are fulfilled (in particular by Lemma 3.5). Hence Theorem 3.1 and Theorem 2.1 imply $Sys^{\mathsf{secmsg,Enc}}_{n,L,T} \geq^{\mathsf{poly}}_{\mathsf{sec}} Sys^{\mathsf{secmsg,Encsim}}_{n,L,T}$ (again with blackbox simulatability).

3.4.3 Simulator

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

We construct $\mathsf{Sim}_{\mathcal{H}}$ as the combination of several machines, see Figure 3.3. It uses the given A as a submachine without any port renaming. (Although all figures show H as using all the specified ports, the proof is general.)

- $\mathsf{M}^*_{u,\mathcal{H}}$, for $u \in \mathcal{H}$, equals $\mathsf{M}'_{u,\mathcal{H}}$ with the following modifications:
  - The ports $\mathsf{in}_u?$ and $\mathsf{out}_u!$ are renamed into $\mathsf{to\_adv}_u?$, $\mathsf{from\_adv}_u!$.
  - It has a port $\mathsf{from\_adv}_u{}^{\triangleleft}!$ even if the original system is not localized, and outputs 1 there for every output at $\mathsf{from\_adv}_u!$.

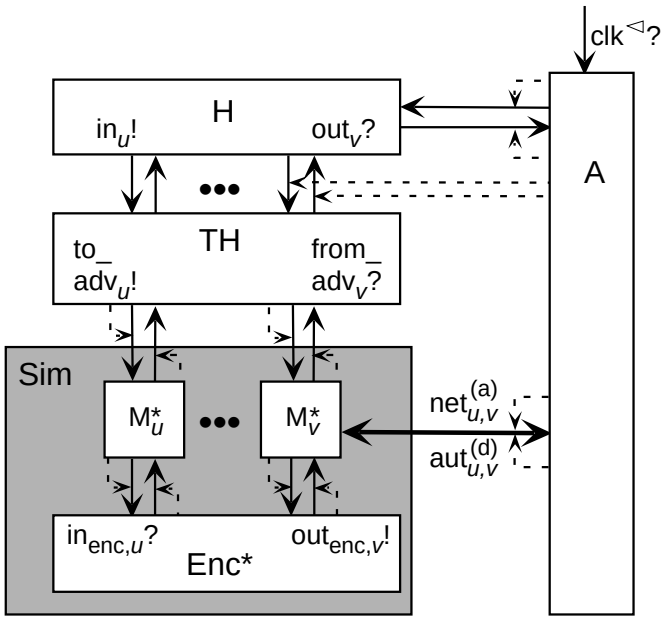

Figure 3.3: Simulator for secure message transmission.

– On input $(\mathsf{send\_blindly}, i, l, v)$ at $\mathsf{to\_adv}_u?$ with $v \in \mathcal{H}$ and $i \leq T(k)$, behave like $\mathsf{M}'_{u,\mathcal{H}}$ on input (send, m, v) for a message m with $\mathsf{len}(m) = l$ (including the remaining domain checks), but instead of computing $sig \leftarrow \mathsf{sign}_{sks_u, sc_u}(u, m, v)$ compute only the length l^* of any such sig using the algorithm $\mathsf{sig\_len}$. Output $(\mathsf{encrypt\_blindly}, pke_v, (u, i, v), l^*)$ at $\mathsf{in}_{\mathsf{enc},u}!$. Then wait and continue like $\mathsf{M}'_{u,\mathcal{H}}$.

– In "receive", when getting a result at $\mathsf{out}_{\mathsf{enc},u}?$: If it is of the form $(\mathsf{decrypted}, sig)$, treat it as $\mathsf{M}'_{u,\mathcal{H}}$ treats sig. If it is of the form $(\mathsf{decrypted\_blindly}, (v', i, u'))$ with $v' = v$, $u' = u$, and $i \leq T(k)$, output $(\mathsf{receive\_blindly}, v, i)$ at $\mathsf{from\_adv}_u!$.

• $\mathsf{Enc}^*_{\mathcal{H}}$ equals $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ with the following modifications:

– An array *blind_ciphers* replaces *ciphers*.

– Instead of inputs $(\mathsf{encrypt}, \ldots)$, it accepts inputs $(\mathsf{encrypt\_blindly}, pke, mid, l^*)$ with $pke, mid \in \Sigma^+$, $l^* \in \mathbb{N}$, and pke of correct length. Here *mid* is a message identifier. If it finds the desired

tuple in *keys* (otherwise the result is $\downarrow$), then { $s_{pke} := s_{pke} + 1$; output $c \leftarrow \mathsf{E}_{pke}(m_{sim,l^*})$; add (mid, pke, c) to *blind_ciphers* }.

– In "decrypt", the special part for $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ is replaced by: If $\exists mid$: $(mid, pke, c) \in$ *blind_ciphers* then output $(\mathsf{decrypted\_blindly}, mid)$ else { $m := \mathsf{D}_{ske}(c)$; output $(\mathsf{decrypted}, m)$ }.

Similar to the previous systems, the following lemma holds:

Lemma 3.6 *All machines* $\mathsf{M}^*_{u,\mathcal{H}}$ *are polynomial-time and* $\mathsf{Enc}^*_{\mathcal{H}}$ *is weakly polynomial-time.* □

3.4.4 Overall Proof of the Correctness of the Simulator

To prove the correctness of this simulator, we have to compare every configuration $conf_{\mathsf{sr}} := (\hat{M}''_{\mathcal{H}}, S_{\mathcal{H}}, \mathsf{H}, \mathsf{A}) \in \mathsf{Conf}(Sys^{\mathsf{secmsg},\mathsf{Encsim}}_{n,L,T})$ with $conf_{\mathsf{id}} := (\{\mathsf{TH}_{\mathcal{H}}\}, S_{\mathcal{H}}, \mathsf{H}, \mathsf{comb}(\{\mathsf{Sim}_{\mathcal{H}}, \mathsf{A}\}))$. We call them semi-real and ideal configuration. The overall idea is to define a mapping ϕ from the runs of $conf_{\mathsf{sr}}$ to the runs of $conf_{\mathsf{id}}$, except for negligible subsets on both sides, which we call *error sets*. ϕ respects probabilities, and the views of A and H in runs ρ and $\phi(\rho)$ are equal. This implies the desired computational indistinguishability.

In our case, we can define ϕ state-wise, and only for states before and after steps of H and A because we only consider the views of these machines. In other words, we consider the combination of H and A, interacting with the combination of all other machines (including $\mathsf{Sim}_{\mathcal{H}}$ in $conf_{\mathsf{id}}$); by Lemma 2.2 this does not change the views. We show that $\phi(\delta_{\mathsf{sr}}(\sigma_{\mathsf{sr}})) = \delta_{\mathsf{id}}(\phi(\sigma_{\mathsf{sr}}))$ for all states σ_{sr} reachable in $conf_{\mathsf{sr}}$, except for the error sets. Here δ_{sr} and δ_{id} denote the overall probabilistic transition functions. The error sets will consist of the runs where the adversary successfully forges a signature.

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

- The states of H and A are mapped identically.
- The states of all buffers with the same name in both systems are mapped identically, and so is the scheduled port.

- The remaining buffers in σ_{id} are always empty and their ports not scheduled. (Recall that we only map states before or after steps of H or A.)
- Security parameters are mapped identically and not mentioned again.

It remains to map the joint states of $\mathsf{M}'_{u,\mathcal{H}}$ for all $u \in \mathcal{H}$ and $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ to states of $\mathsf{TH}_{\mathcal{H}}$ and $\mathsf{Sim}_{\mathcal{H}}$ (which consists of the machines $\mathsf{M}^*_{u,\mathcal{H}}$ and $\mathsf{Enc}^*_{\mathcal{H}}$).

- Transaction counters:
  - Each counter tc_u of a machine $\mathsf{M}^*_{u,\mathcal{H}}$ equals that in $\mathsf{M}'_{u,\mathcal{H}}$.
  - The value $stopped^*_u$ in $\mathsf{TH}_{\mathcal{H}}$ is 1 if $tc_u = T(k)$, else 0.
- Key-related variables:
  - Each array $init_{\bullet,u}$ of a machine $\mathsf{M}^*_{u,\mathcal{H}}$ equals that in $\mathsf{M}'_{u,\mathcal{H}}$.
  - The array $init^*$ of $\mathsf{TH}_{\mathcal{H}}$ is derived from the arrays $init_{\bullet,u}$ of the machines $\mathsf{M}'_{u,\mathcal{H}}$: We set $init^*_{v,u} := 1$ whenever $init_{v,u} \neq ()$, else $init^*_{v,u} := 0$.
  - The counter kc and the list $keys$ of $\mathsf{Enc}^*_{\mathcal{H}}$ equal those in $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$.
- Message-related variables:
  - Each counter sc_u of a machine $\mathsf{M}^*_{u,\mathcal{H}}$ equals that in $\mathsf{M}'_{u,\mathcal{H}}$.
  - The array $deliver^*$ of $\mathsf{TH}_{\mathcal{H}}$ and the list $blind\_ciphers$ of $\mathsf{Enc}^*_{\mathcal{H}}$ are derived from $ciphers$ of $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$: Each entry $e := ciphers[j] \neq \downarrow$ is of the form $e = (sig, pke, c)$ and sig of the form $((u, m, v), sig')$ with $u, v \in \mathcal{H}$. (This is shown as Invariant 4 below.) Given $ciphers$, let $\mathsf{ind}_{u,v}(j)$ denote the number of entries up to (and including) $ciphers[j]$ with the given values u, v. Hence for each such entry e we can set $i := \mathsf{ind}_{u,v}(j)$ and
    - $blind\_ciphers[j] := ((u, i, v), pke, c)$,
    - $deliver^*_{u,v}[i] = m$.

3.5 Detailed Proof of Correct Simulation

We have to show that the mapping ϕ has the properties required above. For this, we first define some invariants. Then we prove them together with the property $\phi(\delta_{\mathsf{sr}}(\sigma_{\mathsf{sr}})) = \delta_{\mathsf{id}}(\phi(\sigma_{\mathsf{sr}}))$ (outside the error sets) by considering each type of transition of the combination of machines without A and H. In that proof, states σ_{sr} of $conf_{\mathsf{sr}}$ and $\sigma_{\mathsf{id}} := \phi(\sigma_{\mathsf{id}})$ are given, and all component names refer to these states. Finally, we show that the error sets are indeed negligible.

3.5.1 Invariants

The first four invariants are formulas valid in all states σ_{sr} reachable in $conf_{\mathsf{sr}}$ (before or after steps of H or A). The last one is a run invariant of $conf_{\mathsf{id}}$.

1. *Buffer emptiness.* All buffers $\widetilde{\mathsf{in}_{\mathsf{enc},u}}$ and $\widetilde{\mathsf{out}_{\mathsf{enc},u}}$ are empty.

2. *Counters.* Let $u \in \mathcal{H}$ and $j := \mathsf{size}(ciphers)$. Then $sc_u \leq tc_u \leq T(k)$, and $\mathsf{ind}_{u,v}(j) \leq tc_u$ for all v.

3. *Key and init consistency.* Each list $init_{u,u}$ with $u \in \mathcal{H}$ is empty or a pair, which we then call (sks_u, pke_u). For $u \neq v$ and $v \in \mathcal{H}$, each list $init_{u,v}$ with $u \in \mathcal{M}$ and each message in a buffer $\widetilde{\mathsf{aut}_{u,v}}$ with $u \in \mathcal{H}$ is empty or a pair, which we then call (pks_u, pke_u). (The names are ambiguous, but the index v of the buffer or machine meant will be clear from the context.) If $u \in \mathcal{H}$, then pks_u forms a correct signature key pair with sks_u (in particular, then $init_{u,u} \neq \emptyset$), and pke_u is the same as in $init_{u,u}$ and also occurs in *keys* of $\mathsf{Enc}_{\mathsf{Sim},\mathcal{H}}$ with the given u and a correct decryption key.

4. *Ciphertext format.* Each entry in *ciphers* is of the form (sig, pke, c), where *sig* is of the form $((u, m, v), sig')$ with $u, v \in \mathcal{H}$ and a correct signature with sks_u (from $init_{u,u}$).

5. *Signatures.* The only usage that $\mathsf{M}^*_{u,\mathcal{H}}$ makes of sks_u is to sign triples (u, m, v) with the given u and $v \in \mathcal{A}$, and with strictly increasing counter values.

3.5.2 Possible inputs and counters

In $conf_{\mathsf{sr}}$, an input to the combination of machines without H and A is always made to a machine $\mathsf{M}'_{u,\mathcal{H}}$ at $\mathsf{in}_u?$ or a network port, i.e., $\mathsf{aut}_{v,u}?$ or $\mathsf{net}^{\mathsf{a}}_{v,u}?$.

If already $tc_u = T(k)$, then $\mathsf{M}'_{u,\mathcal{H}}$ is in a final state and does nothing. In $conf_{\mathsf{id}}$, an input at a network port goes to $\mathsf{M}^*_{u,\mathcal{H}}$, which is in the same state by ϕ and does nothing either. An input at $\mathsf{in}_u?$ goes to $\mathsf{TH}_{\mathcal{H}}$, where $stopped^*_u = 1$ by ϕ. In this case, $\mathsf{TH}_{\mathcal{H}}$ neither changes its state nor makes outputs. Thus ϕ and the invariants remain satisfied. Hence from now on we assume $tc_u < T(k)$ before the transition.

3.5.3 Send Initialization

Upon input (init) at $\mathsf{in}_u?$, $\mathsf{M}'_{u,\mathcal{H}}$ increments tc_u and tests if now $tc_u = T(k)$.

Case 1: Still $tc_u < T(k)$. Then both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{TH}_{\mathcal{H}}$ test whether $init_{u,u}/init^*_{u,u}$ signals a previous initialization. ϕ ensures identical results. If yes, both do nothing. Otherwise, in $conf_{\mathsf{sr}}$, $\mathsf{M}'_{u,\mathcal{H}}$ outputs ($\mathsf{generate}$) at $\mathsf{in}_{\mathsf{enc},u}!$, scheduling it immediately. In $conf_{\mathsf{id}}$, $\mathsf{TH}_{\mathcal{H}}$ sets $init^*_{u,u} := 1$ and sends (init) to $\mathsf{M}^*_{u,\mathcal{H}}$, again scheduling this input at once. As ϕ gives $\mathsf{M}^*_{u,\mathcal{H}}$ the same state as $\mathsf{M}'_{u,\mathcal{H}}$, $\mathsf{M}^*_{u,\mathcal{H}}$ behaves exactly like $\mathsf{M}'_{u,\mathcal{H}}$.

Hence both $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ and $\mathsf{Enc}^*_{\mathcal{H}}$ obtain an input ($\mathsf{generate}$) at $\mathsf{in}_{\mathsf{enc},u}?$. They behave identically for it and always make an output pke and schedule it. (We showed in Part A (Section 3.4.1) that $s_{keys}(k) = n$ is not exceeded.) Hence $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ switch again with this input. Both use it as pke_u, store it within $init_{u,u}$, and output (pks_u, pke_u) at all ports $\mathsf{aut}_{u,v}!$. They make no scheduling output, hence control returns to A.

Case 2: $tc_u = T(k)$ after the increment. Then both configurations proceed as before except that $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$, instead of outputting (pks_u, pke_u), output stop at $\mathsf{out}_u!$ and $\mathsf{from\_adv}_u!$, respectively. Then $\mathsf{TH}_{\mathcal{H}}$ sets $stopped^*_u := 1$ and also outputs stop at $\mathsf{out}_u!$.

Invariants 1, 2 and 3 could be affected, but clearly remain satisfied. For ϕ, only the counters and key-related variables can be affected, and the relations clearly remain satisfied.

3.5.4 Receive Initialization

Upon input (pks_v, pke_v) at $\mathsf{aut}_{v,u}?$, both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ increment tc_u and test if now $tc_u = T(k)$. If yes, both only output stop at $\mathsf{out}_u!$ and $\mathsf{from\_adv}_u!$, respectively. Then $\mathsf{TH}_{\mathcal{H}}$ sets $stopped^*_u := 1$ and also outputs stop at $\mathsf{out}_u!$.

Otherwise, they test that the keys are of correct length and $init_{v,u} = ()$. If not, they do nothing. If yes, they set $init_{v,u} := (pks_v, pke_v)$ and output (init, v). As $\mathsf{M}^*_{u,\mathcal{H}}$ schedules this output, $\mathsf{TH}_{\mathcal{H}}$ tests if $init^*_{v,u} = 0$; this is true by ϕ. If $v \in \mathcal{H}$ it also tests $init^*_{v,v} = 1$. By ϕ, this is equivalent to $init_{v,v} \neq ()$, and by Invariant 3, this follows from the existence of pks_v in the buffer $\widetilde{\mathsf{aut}_{v,u}}$ in σ_{sr}. Thus $\mathsf{TH}_{\mathcal{H}}$ accepts this input, sets $init^*_{v,u} := 1$, and outputs (init, v).

Invariants 2 and 3 could be affected, but remain satisfied. For the case $v \in \mathcal{H}$ this is true because the keys now in $init_{v,u}$ were already in $\widetilde{\mathsf{aut}_{v,u}}$. Only the counter- and key-related parts of ϕ are affected, and they clearly remain satisfied.

3.5.5 Send to Honest Party

Upon input (send, m, v) at $\mathsf{in}_u?$ with $u, v \in \mathcal{H}$, $\mathsf{M}'_{u,\mathcal{H}}$ increments tc_u and tests if now $tc_u = T(k)$.

Case 1: Still $tc_u < T(k)$. Then $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{TH}_{\mathcal{H}}$ first test the domains of m and v and consider $init/init^*$, with identical results (by ϕ and Invariant 3). Now assume they continue.

In $conf_{\mathsf{sr}}$, $\mathsf{M}'_{u,\mathcal{H}}$ increments sc_u and computes a signature sig for the message (u, m, v), outputs $(\mathsf{encrypt}, pke_v, sig)$ at $\mathsf{in}_{\mathsf{enc},u}!$, and schedules it. In $conf_{\mathsf{id}}$, $\mathsf{TH}_{\mathcal{H}}$ adds m to $deliver^*_{u,v}$; let i_{new} denote its index. It outputs and schedules $(\mathsf{send\_blindly}, i_{new}, l, v)$ for $l := \mathsf{len}(m)$. By ϕ, $\mathsf{M}^*_{u,\mathcal{H}}$ starts on this input in the same state as $\mathsf{M}'_{u,\mathcal{H}}$. It makes the same tests and counter updates as $\mathsf{M}'_{u,\mathcal{H}}$ and additionally tests $v \in \mathcal{H}$, which is true here, and $i_{new} \leq T(k)$, which is true by Invariant 2 and the derivation of $deliver^*$ with ϕ. Then it computes a value l^* which by definition equals $\mathsf{len}(sig)$, and outputs and schedules $(\mathsf{encrypt\_blindly}, pke_v, (u, i_{new}, v), l^*)$.

Now $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ and $\mathsf{Enc}^*_{\mathcal{H}}$ switch with these encryption requests. By Invariant 3 and ϕ, both find an entry for pke_v, and by Part A, $s_{encs}(k) = nT(k)$ is not exceeded for it. Hence $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ generates $c \leftarrow \mathsf{E}_{pke_v}(m_{sim,\mathsf{len}(sig)})$ and stores (sig, pke_v, c) in $ciphers$. $\mathsf{Enc}^*_{\mathcal{H}}$ generates $c \leftarrow \mathsf{E}_{pke_v}(m_{sim,l^*})$ and stores

$((u, i_{new}, v), pke_v, c)$ in *blind_ciphers*. Hence c has precisely the same distribution. Both output and schedule c. Finally, $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ output c at $\mathsf{net}_{u,v}!$. Control returns to A.

Case 2: $tc_u = T(k)$ after the increment. Both configurations proceed as before except that $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$, instead of outputting c, output stop at $\mathsf{out}_u!$ and $\mathsf{from\_adv}_u!$, respectively. Then $\mathsf{TH}_{\mathcal{H}}$ sets $stopped^*_u := 1$ and also outputs stop at $\mathsf{out}_u!$.

Invariants 1, 2, 4 and 5 could be affected, but 1 and 4 clearly remain true. For Invariant 2, note that at most one entry (sig, pke_v, c) is added to *ciphers*, which only increments $\mathsf{ind}_{u,v}(j)$, while tc_u was also incremented. For Invariant 5, note that $\mathsf{M}^*_{u,\mathcal{H}}$ makes no signature here. As to ϕ, the only non-trivial part is the mapping of the new *ciphers* to *blind_ciphers* and *deliver**: Let $j := \mathsf{size}(ciphers)$ before this step and $i_{old} := \mathsf{ind}_{u,v}[j]$. Then *blind_ciphers* was also of size j and $deliver^*_{u,v}$ of size i_{old} (by ϕ), and thus $i_{new} = i_{old} + 1$. The new entry is $ciphers[j+1]$, and ϕ maps it to $blind\_ciphers[j+1] := ((u, i_{new}, v), pke_v, c)$ and $deliver^*_{u,v}[i_{new}] = m$. These are indeed the new entries in *blind_ciphers* and $deliver^*_{u,v}$.

3.5.6 Send to Dishonest Party

Upon input (send, m, v) at $\mathsf{in}_u?$ with $u \in \mathcal{H}, v \in \mathcal{A}$, $\mathsf{M}'_{u,\mathcal{H}}$ increments tc_u and tests if now $tc_u = T(k)$.

Case 1: Still $tc_u < T(k)$. Then $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{TH}_{\mathcal{H}}$ first test the domains of m and v and consider *init*/*init**, with identical results. Now assume they proceed. $\mathsf{TH}_{\mathcal{H}}$ outputs and schedules (send, m, v). Then $\mathsf{M}^*_{u,\mathcal{H}}$ acts exactly like $\mathsf{M}'_{u,\mathcal{H}}$. Both increase sc_u, compute $c \leftarrow \mathsf{E}_{pke_v}(\mathsf{sign}_{sks_u, sc_u}(u, m, v))$, and output it at $\mathsf{net}_{u,v}!$. Control returns to A.

Case 2: $tc_u = T(k)$ after the increment. Then $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ output stop at $\mathsf{out}_u!$ and $\mathsf{from\_adv}_u!$ immediately. $\mathsf{TH}_{\mathcal{H}}$ sets $stopped^*_u := 1$ and also outputs stop at $\mathsf{out}_u!$.

Invariants 2 and 5 could be affected, but clearly remain satisfied. As to ϕ, only counters are modified and in a consistent way.

3.5.7 Receive from Honest Party

Upon input c at $\mathsf{net}^{\mathsf{a}}_{v,u}?$ with $u, v \in \mathcal{H}$, both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ increment tc_u and test if now $tc_u = T(k)$. If yes, both only output stop at $\mathsf{out}_u!$ and $\mathsf{from\_adv}_u!$, respectively. Then $\mathsf{TH}_{\mathcal{H}}$ sets $stopped^*_u := 1$ and also outputs stop at $\mathsf{out}_u!$.

Otherwise $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ first test the length of c and the contents of *init*, with the same results. If they continue, they both start to parse c by outputting $(\mathsf{decrypt}, pke_u, c)$ at $\mathsf{in}_{\mathsf{enc},u}!$ and scheduling it.

Both $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ and $\mathsf{Enc}^*_{\mathcal{H}}$ first search for an entry $(u, \cdot, ske_u, pke_u, \cdot) \in keys$ with some ske_u; they find it by Invariant 3 (and ϕ). Then they search *ciphers* and *blind_ciphers*, respectively, for an entry (x, pke_u, c), where x is now called *sig* in $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ and *mid* in $\mathsf{Enc}^*_{\mathcal{H}}$. By ϕ, either both succeed with the same index j, or neither.

- If no such entry is found (intuitively, c was generated by the adversary), both set $sig := \mathsf{D}_{ske_u}(c)$. $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ outputs *sig* and $\mathsf{Enc}^*_{\mathcal{H}}$ outputs $(\mathsf{decrypted}, sig)$, and both schedule this output. Now $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ continue parsing in the same way: *sig* does not exceed the length bound by Definition 3.1. Thus they test that $sig \neq \mathsf{false}$, set $m' := \mathsf{test}_{pks_v}(sig)$ and try to write m' as (v, m, u) for the given u, v and a message m with $\mathsf{len}(m) \leq L(k)$. If this does not succeed, they abort and control returns to A. If it succeeds, the simulation would fail, and we put the run of $conf_{\mathsf{id}}$ into a set $Forgeries_{v,k}$ (where k is the security parameter). We call *sig* the "designated signature" for this run. Given a run, one can easily verify whether this case occurs. By Invariant 5, $\mathsf{M}^*_{v,\mathcal{H}}$ never signed m' because $u \notin \mathcal{A}$.

- Now assume that such entries are found. By Invariant 4, *sig* is of the form $sig = ((v', m, u'), sig')$ for some m and $v', u' \in \mathcal{H}$ and a correct signature with $sks_{v'}$. By ϕ, we have $mid = (v', i, u')$ with $i = \mathsf{ind}_{v',u'}(j)$ and $deliver^*_{v',u'}[i] = m$. Finding these entries, $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ outputs *sig* and $\mathsf{Enc}^*_{\mathcal{H}}$ outputs $(\mathsf{decrypted\_blindly}, mid)$. Both schedule these outputs.

  On input *sig*, $\mathsf{M}'_{u,\mathcal{H}}$ continues parsing as in the previous case. Hence it outputs $(\mathsf{receive}, v, m)$ iff $v' = v$ and $u' = u$. (With Invariant 3, one sees that *sig* passes the test with $\mathsf{M}'_{u,\mathcal{H}}$'s variable pks_v.)

On input $(\mathsf{decrypted\_blindly}, mid)$, $\mathsf{M}^*_{u,\mathcal{H}}$ outputs and schedules $(\mathsf{receive\_blindly}, v, i)$ iff $v' = v$ and $u' = u$; the condition $i \leq T(k)$ is fulfilled because of $i = \mathsf{ind}_{v',u'}(j)$ and Invariant 2 (Otherwise both abort and control returns to A.) Now $\mathsf{TH}_{\mathcal{H}}$ verifies $stopped^*_v = 0$, $init^*_{v,v} = 1$, and $init^*_{u,v} = 1$, which is all true by ϕ. It therefore retrieves $deliver^*_{v,u}[i]$, which is m, and also outputs $(\mathsf{receive}, v, m)$.

Only counters are modified, and Invariant 2 and ϕ clearly remain satisfied.

3.5.8 Receive from Dishonest Party

On input c at $\mathsf{net}^{\mathsf{a}}_{v,u}$? for $u \in \mathcal{H}, v \in \mathcal{A}$, everything proceeds as in the case $v \in \mathcal{H}$ until both or neither of $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ and $\mathsf{Enc}^*_{\mathcal{H}}$ have found the desired entry in *ciphers* and *blind_ciphers*.

- If no entry is found, decryption and parsing continues as above. If it is successful, $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ output $(\mathsf{receive}, v, m)$, and $\mathsf{M}^*_{u,\mathcal{H}}$ schedules it. $\mathsf{TH}_{\mathcal{H}}$ verifies that $v \in \mathcal{A}$, $\mathsf{len}(m) \leq L(k)$, which succeeds by the tests in $\mathsf{M}^*_{u,\mathcal{H}}$, and considers $stopped^*$ and $init^*$, which succeeds by ϕ. Then it also outputs $(\mathsf{receive}, v, m)$.
- If such entries are found, Invariant 4 implies that sig is a pair $((v', m, u'), sig')$ with $v' \in \mathcal{H}$, and by ϕ, mid is a triple (v', i, u') with this v'. Thus parsing in both $\mathsf{M}'_{u,\mathcal{H}}$ and $\mathsf{M}^*_{u,\mathcal{H}}$ fails because $v \in \mathcal{A}$ and thus $v' \neq v$. They abort and control returns to A.

Only counters are modified, and Invariant 2 and ϕ clearly remain satisfied.

3.5.9 Final Reduction

It remains to be shown that the error sets are negligible. As ϕ retains probabilities where it is defined, the error sets have the same probabilities in both configurations, and it suffices to consider $conf_{\mathsf{id}}$. There, the error set for each k is the union of the sets $Forgeries_{v,k}$ with $v \in \mathcal{M}$. In each run in $Forgeries_{v,k}$, the adversary has produced a signature with a key sks_v of a correct machine $\mathsf{M}^*_{v,\mathcal{H}}$ under a message that this machine had not signed. Hence the overall statement follows from the security of the signature scheme.

More precisely, the proof is a standard reduction: It suffices to show that the sequence of probabilities of the sets $(Forgeries_{v,k})_{k\in\mathbb{N}}$ is negligible for each $v \in \mathcal{M}$ (Definition 2.12), because *NEGL* is closed under finite addition.

Assume the contrary for a certain v. We then construct an adversary $\mathsf{A}_{\mathsf{sig}}$ against the signer machine Sig_T (recall Definition 3.4). On input a public key *pks*, it simulates $conf_{\mathsf{id}}$ with *pks* as pks_v (i.e., instead of generating pks_v in $\mathsf{M}^*_{v,\mathcal{H}}$) and using the signer machine Sig_T for all signatures with the now unknown sks_v. By Invariant 5, signing is the only usage of sks_v, and Sig_T always answers correctly by Lemma 3.2 (skipping signatures) and because sc_v cannot grow beyond $T(k)$ by Invariant 2 $\mathsf{A}_{\mathsf{sig}}$ verifies whether the run belongs to $Forgeries_{v,k}$ and if yes, outputs the designated signature. By Section 3.5.7 this is efficiently possible, and the designated signature is a successful forgery in the sense of Definition 3.4 because $\mathsf{A}_{\mathsf{sig}}$ did not have to ask Sig_T to sign this message. Hence the success probability of $\mathsf{A}_{\mathsf{sig}}$ for a security parameter k is precisely the probability of $Forgeries_{v,k}$.

Moreover, $\mathsf{A}_{\mathsf{sig}}$ is polynomial-time: By Lemmas 3.6 and 2.2f), the combination of all machines $\mathsf{M}^*_{u,\mathcal{H}}$ and $\mathsf{Enc}^*_{\mathcal{H}}$ is polynomial-time. The combination of this with A and H is also polynomial-time by Lemma 2.2e), and finally the combination with $\mathsf{TH}_{\mathcal{H}}$ by Lemmas 3.1 and 2.2f). By associativity this implies that the simulation of $conf_{\mathsf{id}}$ is polynomial-time.

4 Faithfully Implementing Protocols

4.1 Introduction

General security models such as the ones presented in Chapter 2 and in the prior deliverable D4 [4] offer a rigorously defined foundation for specifying and proving the security of cryptographic protocols. This has been exploited — by providing a corresponding formal security proof — to achieve high confidence in several protocols [45, 7, 46] and more provably secure protocols are expected to follow. It is only a question of time until those protocols will be implemented and deployed.

However, it is by no means an easy task to implement such provably secure protocols in a way which retains their security properties in the real world.[1] Given our limited knowledge about the real world, all models inherently have some abstractions which only approximate and idealize the reality. Furthermore, a model has to make tradeoffs between, on the one hand, being highly detailed and close to reality and, on the other hand, keeping the model's complexity manageable. The resulting idealizations leave a crucial gap between models and the real world: idealizations restrict the capabilities of an attacker and rule out certain classes of attacks in the model which can be serious sources of security flaws in the real world. In fact, most successful attacks against cryptographic systems identify and exploit weaknesses of the implementation.

In this chapter, we identify this gap for the synchronous as well as the asynchronous model defined in the earlier deliverable [4] and the present deliverable, respectively. We start in Section 4.2 by analyzing how the model and the real world differ and assess the impact of each identified abstraction. In Section 4.3 we discuss possible approaches for closing the gap.

4.2 Model Abstractions and Impact on Real-World Security

Let us now identify and inspect the abstractions in these models. In particular, we should compare each identified abstraction with the real-world

[1] In the following, we mean by "real world" the world as ruled by the laws of physics and experienced by human users.

and discuss its impact on the security of a real-world implementation. As we will see, several of these abstractions lead to subtle yet highly relevant security issues and pose special restrictions on an implementation. Thus, an implementor has to take special care to accurately respect these abstractions in an implementation, e.g., in the common runtime environment and operating systems.

We can roughly classify the various abstractions into following categories: (1) computation model, (2) communication model, and (3) synchronization and time. In the following sections, we will take a closer look at each of the aforementioned categories in turn. For each category, we can split the discussion orthogonally into two classes: (a) properties of (supposedly) correct machines, and (b) adversary capabilities. Obviously, as implementors we can only control the class (a) but not (b). However, the discussion of (b) is important as well as it shows us further implicit assumptions and limitations of our model.

As the term "machine" has many connotations, it can be quite misleading. For clarity, we will use the following convention in the sequel: A *machine* always corresponds to a specification in the previously described model. The implementation of a machine is called a *component*. A *host* denotes a runtime environment where several components can be colocated. Limiting the discussion primarily to standard cryptographic systems, we assume that a host is a stand-alone computer under a single administrative domain, i.e., all components in the same host trust each other in respect to the fulfillment of the corresponding specifications.

4.2.1 Computation Model

The computational aspects of machines are represented by I/O Automata and can be easily realized by modern computers. However, there are some properties of (honest) machines which deserve special attention.

4.2.1.1 Atomicity of Transitions

Transitions of automata are modeled as being atomic, i.e., either they are performed completely or not at all. In real-world runtime environments or operating systems with interrupts and multiprocessing, atomicity of tran-

sitions cannot be assumed without further measures. This bears two kinds of problems: First, in case of dynamic corruptions, it is possible that *intermediary states* leak certain information to the attacker that would not have been leaked by states which the machine reaches after complete transitions.[2] Second, a consequence of transitions being not atomic in the real world is that it may change the *timing characteristics* of implementations.

As the implications of potential timing channel are discussed below in Section 4.2.4 in more details, we will focus in the following only on the leakage of intermediary states. As dynamic corruptions model any accidental weaknesses in the implementation and operation of a machine, e.g., the loss of keys in an unprotected backup-tape or the theft of a personal device, there seems to be neither a reasonable way to restrict dynamic corruptions to idle components in the real world nor to rule out dynamic corruptions completely. Therefore, we analyze the impact of leaking intermediary states in more detail: Leakage of an intermediary state to an attacker could be a security thread if it contains information which the attacker cannot compute from the state reached after the transition is completely executed: one example may be a transition in which a corruptible machine randomly chooses a value x from a large domain, computes the image $y = f(x)$ of x under a *one-way function* f and finally deletes x. However, a corruption during a transition is equivalent from the viewpoint of an external observer to an adversary which corrupts the machine immediately before that transition and then simulates the proper operation of the machine up to the actual time of corruption. As the latter case is handled by the model, the leakage of intermediary states does not have to be of deeper concern.

4.2.1.2 Enforcement of Interfaces

The models restrict access to machines to a well-defined interface as given by the machine's definition. In particular, this means that machines have no direct access to the *internal state* of other machines, not even to states of sub-machines.

Implementations based on today's standard programming models and

[2]Recall that dynamic corruptions are modeled as sending the full internal state to the attacker and giving full control to him. Thus, the "implicit" atomicity-assumption implies that in the model, in case of corruption, no intermediary states can leak to the adversary.

operated in the usual runtime-environments do not guarantee this separation in general. Since (local) sub-machines are commonly realized as sub-function calls to routines which lie in the same address space, the resulting components have mutual direct access to their internal states and interfaces are not as strictly enforced as required by security proofs in these models.

To assess the impact of this issue, we have to take a look at the security proofs in the models, especially at the different treatment of correct and incorrect, i.e., adversary-controlled, machines:

1. *Correct machines* are assumed to access other machines only through their interfaces and to be accessible themselves exclusively through their interface too. Thus, this must be guaranteed in the real-world implementation of correct machines. This can be ensured by any combination of physical separation, processor-based memory-protection and language-based separation [52].

2. *Incorrect machines* are modeled as one adversary machine. This way of modeling incorrect machines gives the attacker complete access to all incorrect machines, even to their internal states. Thus, the interface-enforcement for these machines is not crucial for the real-world security. The same holds for *dynamically corruptible* machines after a corruption has taken place. Although not directly vital for retaining the security properties in the real-world,[3] the implementation should ensure interface-enforcement even for components which are allowed to become corrupted. On the one hand, dynamic corruptions are undesirable and the likelihood of them should be minimized. On the other hand, some trust-models, e.g., a (n, t)-threshold model which assumes that out of n servers at most $t-1$ ever get corrupted, implicitly assume a certain resistance to dynamic corruptions.

Another closely related issue is discussed in Section 4.2.4 below: even if interfaces are strictly enforced by the runtime-environment, there can be additional *real-world channels*, which bypass the seemingly enforced interface.

[3] Any accidental violation of the interface enforcement in the real-model can be modeled as a corresponding dynamic corruption.

4.2.1.3 Practicality

A final word to the specification of honest machines and their impact on the practicality of their implementation. While it is possible to model systems which are unrealizable in practice, e.g., because they never terminate or require an unrealistic amount of memory space and computation time, this does not have an impact on the integrity and confidentiality properties studied here. However, when we would extend the model to deal also with (strong) liveness properties, e.g., the guaranteed real-time progress of a system, one would have to consider also this aspect.

4.2.1.4 Adversary

Let us look now at the computational aspects of adversaries. The particular type of I/O Automata is polynomially equivalent to (interactive) Turing machines. According to the classical Church-Turing thesis, this covers the complete class of computable functions and, therefore, represents a realistic upper bound on the adversary's computational capabilities. As a consequence, perfectly secure systems, which guarantee the absence of any adversarial algorithm, do not deserve further attention during implementation.

However, when implementing computationally secure systems, the overwhelming majority of cryptographic systems, two additional aspects have to be considered. First, the security is based on the assumption that only algorithms polynomial in time and space are realizable in practice. While this looks reasonable today, the advent of new computation models such as molecular (biological) computing [34] or quantum computing [13] might prove this assumption to be wrong in the future. However, note that none of these newer computation models violates the Church-Turing thesis. They only have the potential to topple aspects of the polynomiality aspects and are not relevant to perfectly secure systems. Furthermore, the security is often based on unproven complexity-theoretic assumptions which can turn out to be wrong regardless of advances in the computational models. Second and more important in this context, computational proofs are only asymptotic arguments based on some security parameters, i.e., the security is guaranteed a priori only for infinitely large security parameters. Obviously, in practice a particular and finite security parameter must be chosen. It is very important

that the choice is made conservatively under consideration of the lifetime of the system and the current state of the art as well as potential future advances in the cryptanalytical techniques. The interested reader is referred to Lenstra and Verheul [35] for a reasonable approach on how to safely choose a security parameter.

4.2.2 Communication Model

Ports and buffer-machines are the models' abstraction of communication links between components. This has following impact on the implementation of honest components.

4.2.2.1 Atomic Communication

Communication is assumed to be atomic, i.e., receiving machines have no access to parts of the message before it is fully received. In combination with the fact that the models allow the transfer of arbitrarily long messages, this assumption does not hold for standard communication mechanisms available in runtime environments. Fragmentation of large messages into smaller packets is a common technique in today's communication protocols. The adversary may access received fragments of the message before the last fragment was received. It seems that this does not have any negative effect on the overall security and it is tempting not to implement a corresponding atomic communication service in the real world. However, it is necessary to have a closer look at this issue to get firm confidence that non-atomic communication cannot cause any harm.

4.2.2.2 Messages Clipping

The models assume that inputs to machines are automatically clipped to a maximum message length. In particular, the machine's behavior is completely independent from the message part that goes beyond this length. The maximum acceptable message length is specified in the machine's definition and often depends on the internal state of the machine. It is required to model memory limitations of a component, preventing real-world security is-

sues such as *buffer-overflows*, and the polynomial-time aspects of machines. The most critical aspect of this clipping is the fact that this clipping is not necessarily noticeable by the adversary in the model, yet in the real-world such an event, e.g., a host runs out of buffer space or stops, might be detectable. While mostly not of harm, some (arguably contrived) examples are possible where the detection of clipping would lead to insecure implementations.

4.2.2.3 Topology Unawareness

In the model's abstract communication there is no notion of topology, i.e., machines are unaware of the network topology. In the real-world, however, local (software) connections inside a host are different to LAN or WAN connections.

4.2.2.4 Connection Types and Reliability

The channel and clocking models determine the properties of the communication links. If the links are neither secure, authentic nor reliable,[4] no problem should arise when implementing the communication links. However, special care has to be taken when the model assumes one or more of aforementioned properties. For example, one often models communication inside a logical entity as reliable. In this case, one has to ensure that the adversary cannot influence the scheduling of the operating system. Similarly, inter-entity channels labeled authentic or secure require special protection unless the physical properties of the channel prohibit any adversary access.

4.2.2.5 Adversary

The model allows quite realistically to represent all types of intended communication channels. Using the channel and clocking model we can also realistically yet generically model the adversary's control over the network. Using the notation of the honest user machine H which can arbitrarily con-

[4] A channel is considered reliable when the clocking of the channel is performed by an honest machine.

nect to the adversary, the model covers also all unintentional communication between users of a given component and the adversary. However, the real world bears also the risk of additional unintentional information flows between the component itself and the adversary, namely covert channels and side channels. This is not directly covered in the model and is discussed in more detail in Section 4.2.4 below.

4.2.3 Model Semantics — The Notion of Runs

One of the most important notions in the models is that of a *run*, which defines how combinations of several machines act together. The definition of runs differ strongly between the synchronous model, as described in deliverable D4 [4], and the asynchronous model presented in this deliverable. Therefore, the following discussion is split into a synchronous and an asynchronous part:

4.2.3.1 Asynchronous Model

1. **Synchrony Assumptions:** Protocols which were proven in the asynchronous model do not require any synchrony assumptions in order to be secure. Thus, from the security point of view there is no direct need for the runtime environment to offer special synchrony-services. When broadening the focus from security goals to other quality-of-service properties, real-world implementations of asynchronous protocols may make use of synchrony-services offered by the runtime environment. If additional services are used in the implementation, the implementor has to take care that he does not introduce new security flaws. In particular, time-outs should be used only very carefully, e.g., no conclusion of the state of other components can be securely derived from the occurrence of a time-out without making further assumptions.

2. **Scheduling:** One of the most characteristic features of the asynchronous model is its scheduling policy, called *distributed scheduling*: machines which have been scheduled are allowed to schedule other machines, normally local sub-machines, themselves. Machines not modeled as being scheduled by any other machine of the system are assumed

to be scheduled by the attacker.

As long as the scheduling is modeled as being controlled by the attacker, no scheduling policy of the implementation can have negative effects on the system-security. In other words: from a security point of view there is no "wrong" way of implementing them. Thus, they may be implemented arbitrarily, if security (safety) is the only criteria to judge good implementations. However, when looking at availability and liveness properties, additional system-specific requirements on scheduling and activation will arise. This is not covered by the asynchronous model[5] (these ports are just modeled as free clocking ports) and must be specified and verified by the system-designer outside of the current model.

Mostly for convenience of modeling and proofs, some machines are scheduled also by honest machines. In this case, contrary to the case discussed above, particular scheduling strategies might lead to insecure systems. Therefore, the implementor has to guarantee that his implementation complies with such scheduling assumptions encoded in the model.

3. **Concurrency:** *Concurrency* of machines is not part of the asynchronous model: At any time exactly one machine of a configuration is switched by the run-algorithm. However, concurrency is inherent in the real world and can only be partly masked by the runtime-environment: at least concurrency of real world users, e.g., humans, and real world attackers cannot be prevented by implementations. Possible impacts of this inherent concurrency deserve further studies in the future.

4.2.3.2 Synchronous Model

1. **Synchrony Assumptions:** In the synchronous model, one assumes that all machines are perfectly synchronized and operate in lock step, i.e., they read inputs and make outputs at fixed points in (absolute!) time. In fact, the security of many protocols strongly depends on this

[5]Note that this is not a fault of the system model, but rather inherent due to the asynchronous nature of the system model, see Chapter 2.

and break when some honest component receives and processes messages of a round i after other components have already sent messages for round $i + 1$. Therefore, an implementation has to synchronize the whole distributed system. Achieving this efficiently with quality-of-service guarantees regarding network latency as well as with security guarantees seems to be a very difficult task in the face of powerful attackers, arbitrary clock variations and network delays. Also note that the *whole* real-world implementation, compromising all modularly composed components of dependent applications, has to be lock-stepped, not only a simple system implementing a service such as the secure message transmission shown in the deliverable D4 [4]. As a (severe) consequence the most dragging component will determine the pace of the system.

Note that the synchronous model does not allow arbitrary concurrency (all honest machines are strictly parallel inside a round) and restricts the scheduling of honest machines in respect to the adversary. However, as the deliverable D4 [4] showed that the standard synchronous model is equivalent to arbitrary interleaving of an arbitrary number of sub-rounds, concurrency and scheduling seems less of an issue for the synchronous system and is by far dominated by the impact of the synchrony assumption.

4.2.4 Additional Information-Flows in the Real-World

Security proofs in the models assume a certain connection-structure. The connection-structure is the explicit model of all channels between components which the prover considered when proving the system secure. Thus, the security of systems is only retained if there are no additional channels between components that should be unconnected. However, it is a well known fact that in real-world systems there are many possible sources of unintentional information flows. Secure real-world implementations have to take care that no additional information-flow becomes possible.

One can distinguish two general types of real-world channel, which may endanger the practical security of systems, even to those that were proven secure in one of the models:

1. A *covert channel* denotes an information flow mechanism within a system that is based on system resources not normally intended for communication. Both sender and receiver *deliberately* use these system resources to exchange information. The problem of covert channels was introduced by Lampson in [36] and has been extensively studied since then. Covert channels can be divided into two classes, namely *storage channels* and *timing channels*. Storage channels misuse shared (system) variables or attributes to transmit information from the sender to the receiver. Timing channels transmit information by varying the amount of time required for certain tasks.

   Generally, covert channels may have a great impact on system security. However, in implementations of systems that were proven secure in the models considered here, their impact is limited. This is quite obvious: covert channels are not established accidentally by the sender, but only *on purpose*. When proving security of a system in the general models, the prover assumes that all incorrect machines are combined in one attacker machine. This is a strictly stronger attacker model[6] and implies that even an implementation of a secure system in a runtime-environment which enables covert channels is secure. Thus, we will not go into the details of how to eliminate covert channels in real-world implementations.

   However, using components which cannot be fully trusted to be correct in the implementation, e.g., third party "off-the-shelf" components (GUIs, etc.), may establish covert channels and compromise the security of the implementation.

2. The other type of real-world channels is called *side channel*. The crucial difference to covert channels is that the component which "sends" information over these channels does not do this intentionally, but is rather leaking information. This type subsumes channels which allow for information flow from a correct component to an attacker or vice versa. In the following we discus different kinds of side channels. Note that this list cannot be assumed to be complete, because it depends on

[6] One can regard the incorrect machines which are fully connected with each other to be a special case of one attacker machine that combines all incorrect machines and over which one all-quantifies in security proofs.

the perception of potential attackers and perception may improve with any progress in the field of natural sciences.[7]

(a) **Timing-Channels:** Both, the synchronous and the asynchronous model abstract partially or completely from time aspects. Thus, timing-channels are at least assumed to be restricted or even non-existent in the models. Timing-channels are a priori only a security-risk for confidentiality-properties. However, by leaking secrets arbitrary security-properties may be violated. For implementors, the question is how to implement "absence of (real)-time" in the real world, or at least hide it from the implementation: The presence of time in the real world is an inherent difference to the models discussed here.[8] The absence or idealization of time in the models abstract away the possibility of (certain) timing-channels, which an attacker can use in the real world to gain information about the internal state of components [32].

If we assume strict interface enforcement, a realistic channel model and statistical separation of unrelated components, timing information can leave the system only via specified ports and free ports of a system, i.e., the ports where potential adversaries can gather information from. For the attacker A being able to measure the behavior and timing-properties of a collection of correct and causally related, i.e., connected, components, there must be a directed cycle of channels leading from A or H [9] via an input port into this collection and eventually back to H or A. This kind of "syntactic criteria" is necessary for the leakage of information via timing channels, but obviously not sufficient: the timing characteristics has additionally to depend on the internal state of the components in the collection.

[7] Progress in natural sciences can be understood as a refinement of the "model" called the real-world, opening new kinds of attacks and real-world channels.

[8] In the synchronous model there is only a discrete notion of time (each machine needs one standard unit of time, a so called sub-round, to switch). Thus, time has not been completely abstracted away. Yet even this idealization of real-time in the real world has to be emulated securely.

[9] In the model, we allow arbitrary behavior by H. Therefore, we have to assume the worst, i.e., that H does whatever helps A to get timing information.

(b) **Simple and Differential Power Channels:** An attacker might be able to derive information from measuring the power consumption of devices [5, 31, 30, 6]. This is an attack which is particularly successful on trusted devices which are potentially in the complete (external) control of an adversary, such as smart-cards in an e-cash system which should protect the bank from the payment system's customer. Note that the supposed tamper-resistance of a smart-card in this example is of no help for this kind of attack!

(c) **Electro-magnetic Channels:** Similarly to above, an adversary can also derive information from the electro-magnetic signals emanated by usual devices [33, 6, 49, 51], e.g., a display. This is in particular of concern as it can happen over considerable distance and does not require the attacker to have control over the device as usually required in the previous case.

(d) **Fault-Injection Channels:** Fault injection can be used by an attacker to try to negatively influence the behavior of a component. Even more surprisingly, fault injection can also lead to information flows through supposedly innocent error-treatment such as error-messages on illegal input [15]. A very prominent example is the (very realistic) attack on SSL described by Bleichenbacher [14].

All of above side and covert channels can be modeled by adaptive adversaries. However, some systems require in their trust model that some components are unconditionally correct, e.g., by considering only static adversaries. Furthermore, even when tolerating arbitrary adaptive adversaries it should be our goal to keep the opportunities for corruption to a minimum such as ones which cannot be excluded by an implementation, e.g., corruptions due to pure luck in unavoidable cryptanalytic attacks or attacks using "social engineering" of involved humans. Therefore, an implementor has to exclude such additional information flows as much as possible.

4.3 Closing the Gap: Possible Approaches

In the previous section we have seen that many of the model's current abstractions bear subtle security risks for real-world implementations. Let us now discuss the possible ways to tackle this problem. There are basically two

main approaches: The first approach is to extend the generality of the models to make them closer to reality. This will give correspondingly additional confidence in security proofs. The second approach is to address the issues in the implementation of the protocols, a less formal yet more practical way to get faithful abstractions. Let us look now at the different options in more detail.

4.3.1 Closing the Gap when Proving the Security of Protocols

This approach can be subdivided again into two different strategies:

1. **Extend the generality of the underlying system model:** This is the most rigorous and general approach. We may extend the underlying system model, i.e., the definition of machines and their interaction, by including additional details of the real world, e.g., real-time clocks and the various physical means which allow propagation of information. That is, we drop abstractions and idealizations such that the models resemble the real world even more accurately. Consider for example the abstraction that transitions of machines have no duration. This hides timing attacks and timing channels in the models. One may eliminate this abstraction by extending the machine model as follows: Instead of defining machine-transitions "only" by their input/output behavior, one could include additional information about their real world timing behavior.[10] This extension would *enforce* additional reasoning about the effects of timing channels on the system security in *any* security proof in this extended model. This is quite clear, since it is a change in the model's semantics. Thus, a prover using this extended model has two possibilities: Either he proves that these extensions have no effects on the security of his protocols, or he explicitly includes the arising weaknesses into the service specification, i.e., the trusted host. The latter forces everyone using this protocol in designing higher protocols to consider the effects of these weaknesses on the security of the higher protocols.

   The advantage of this approach is that it *forces* protocol designers to

[10] For buffer machines this may be used to model networks with certain quality-of-service guarantees regarding their bandwidth and latency.

reason more about real world threats on the security of their protocols, thus, simultaneously making secure implementations afterwards easier. Another advantage is that general theorems such as the composition theorems provided by the models — assuming one could derive them despite the extensions of the model — ensure the consistency of resulting systems also when based on modular construction.

The considerable drawback of this approach is quite clear: proofs become arbitrarily complex when including more and more real world features into the model. This holds not only for general theorems like the composition theorem, but also for every single security proof made in this model. Today's lack of adequate tool support makes such proofs in highly detailed models hardly practicable. A possible way out would be the following. We ensure that generalization of the model maintains the current model as a special case, similar to the synchronous model which is a special case of the asynchronous model [8]. Based on this, one might be able to design general tools which transform a system, which is secure in the current model, into a corresponding system, which is secure in the general model. This would be similar to the compiler techniques proposed by Bellare et al. [11] which transform protocols secure when assuming authenticated channels to protocols secure also over unauthenticated channels.

2. **Model certain real-world features in the standard models:** A more pragmatic approach is to include additional real world characteristics in a more ad-hoc and on-demand manner in the standard models. One example for this approach is the way *adaptive adversaries* are modeled into the *attacker model* of the standard models as shown in Section 2.4.2.

   To pick up the timing-attack example from above, one may think of explicitly extending the *outputs* of machines by runtime information to include timing-attacks in the security proofs. General side channels between system components may be modeled by including additional dedicated buffer machines which allow information flow from system components to the attacker. These buffer machines can be defined quite freely to resemble the relevant characteristics of the correspondent real-world side channels in common runtime environments.

The advantage of this approach is that one could just take these aspects into account which seem appropriate in a given context and limit the increase of the models complexity. However, the ad-hoc nature brings the drawback that the omission of real world characteristics may not only happen intentionally, but also unintentionally in case the prover was unaware of certain type of attacks or certain real-world features crucial to the overall security. In the previous approach the proofs just would not go through.

4.3.2 Closing the Gap from the Implementation Side

The most pragmatic, yet informal approach, is to ensure that the implementation respects the assumptions made in the security proofs and meets the model-inherent abstractions in the real world. This requires that all security-critical real world characteristics such as time or side channels, not visible in the model, are properly taken care of in the implementation.

To assist the implementors, there should be a general implementation architecture on how to safely and faithfully implement the abstractions such that the implementation of a system proven secure in the standard model retains the security properties in the real world. This implementation architecture must be based on a careful study of each abstraction and possible related real-world attacks. It should also enclose common engineering principles such as the use of strictly typed and modularity enforcing programming languages with automatic memory management. Ideally, it should also be extended by supporting tools and tightly linked to compilers and operating systems. Picking up the timing-attack example again, the runtime environment together with a static analysis of the code might ensure that transitions of components do not vary in their timing behavior. This would prevent timing channels in the implementation.

The main advantage of this approach is that security proofs in the models stay manageable. Additionally, many of the implementation guidelines are general, i.e., independent of the implemented systems, and relatively straightforward. The main drawback is the informality of the approach. Furthermore, the performance of the adapted implementation will most likely degrade and the implementation will become more complex. However, the aforementioned alternatives require considerable changes to the systems nec-

essary such that the proofs would go through. It is very likely that these changes result in similar performance drawbacks.

As this approach is currently the most, if not only, viable, it will be one focus of our ongoing work.

5 Conclusion

We presented a rigorous model for secure reactive systems with cryptographic parts in asynchronous networks. This model comprises several concepts important to MAFTIA such as faults/attacks, topologies and failures under the weak synchrony assumption of an asynchronous network. We also presented a composition theorem for this model which allows for modular design and security proofs of complex systems. Furthermore, common types of cryptographic systems and their trust models were expressed as special cases of this model, in particular systems with static or adaptive adversaries, and systems with different types of underlying channels. The new model of adaptive adversaries is highly relevant to MAFTIA since it is a more realistic attack model in open networks such as the Internet. The model of adaptive adversaries can be easily adapted to the synchronous model of secure reactive systems as described in the prior Work-package 6 deliverable D4.

We also presented a detailed, rigorous proof of a system for secure message transmission in the asynchronous model. This example was chosen, because it is one of the most basic building blocks of the MAFTIA middle-ware. In addition, it illustrates the influence of different synchrony assumptions on the models of secure reactive systems, when compared with the related example in D4.

Finally, we discussed issues that may arise when implementing reactive systems which have been proven secure in the models defined in [4] and in the present deliverable. We focused particularly on security weaknesses which may arise from the model's idealizations which do not hold in the real world in general. In this deliverable, we mainly highlighted the potential problems and provided only limited solutions to address them. It is the focus of our ongoing research to develop a corresponding implementation architecture including tools and guidelines which will support an implementor in constructing a secure real-world system.

Acknowledgment

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

Bibliography

[1] M. Abadi, C. Fournet and G. Gonthier, *Secure Implementation of Channel Abstractions*, 13th Symp. on Logic in Computer Science (LICS), IEEE, 1998, 105–116 6

[2] M. Abadi and A. Gordon, *A Calculus for Cryptographic Protocols: The Spi Calculus*, 4th Conf. on Computer and Communications Security (CCS), ACM, 1997, 36–47 6

[3] M. Abadi and P. Rogaway, *Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption)*, IFIP Intern. Conf. on Theoretical Computer Science (TCS 2000), LNCS 1872, Springer-Verlag, 2000, 3–22 1, 5

[4] A. Adelsbach and B. Pfitzmann Eds., *Formal Model of Basic Concepts*, MAFTIA Project, Deliverable D4, 2001 2, 4, 7, 24, 35, 61, 68, 70, 78

[5] M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart, *Power analysis, what is now possible...*, In Advances in Cryptology – ASIACRYPT'2000, LNCS 1976, Springer-Verlag, 2000, 489–502 73

[6] R. J. Anderson, *Security Engineering*, John Wiley & Sons, 2001 73

[7] M. Backes and B. Pfitzmann, *Computational probabilistic noninterference*, In submission, Nov. 2001. 35, 61

[8] M. Backes and B. Pfitzmann, *Embedding Synchronous in Asynchronous Reactive Systems*, Personal communication, 2001. 7, 75

[9] D. Beaver, *Secure Multiparty Protocols and Zero Knowledge Proof Systems Tolerating a Faulty Minority*, J. of Cryptology 4/2 (1991) 75–122 5

[10] M. Bellare, A. Boldyreva and S. Micali, *Public-Key Encryption in a Multi-user Setting: Security Proofs and Improvements*, Eurocrypt 2000, LNCS 1807, Springer-Verlag, 2000, 259–274 4, 43

[11] M. Bellare, R. Canetti and 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, 75

[12] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway, *Relations Among Notions of Security for Public-Key Encryption Schemes*, Crypto '98, LNCS 1462, Springer-Verlag, 1998, 26–45 38

[13] A. Berthiaume. *Quantum computation*, In L. Hemaspaandra and A. Selman (Eds.), Complexity Theory Retrospective II, pages 23–51. Springer-Verlag, Berlin Germany, 1997. 65

[14] D. Bleichenbacher, *Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS*, In Advances in Cryptology – CRYPTO '98, LNCS 1462, Springer-Verlag, 1–12 73

[15] D. Boneh, R. A. DeMillo, and R. J. Lipton, *On the importance of eliminating errors in cryptographic computations*, Journal of Cryptology, 14(2):101–119, 2001 73

[16] R. Canetti, *Studies in Secure Multiparty Computation and Applications,* Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, June 1995, revised March 1996 6, 7

[17] R. Canetti, *Security and Composition of Multiparty Cryptographic Protocols*, J. of Cryptology 13/1 (2000) 143–202 5, 26

[18] R. Canetti, *A Unified Framework for Analyzing Security of Protocols*, IACR Cryptology ePrint Archive 2000/067, December 2000, http://eprint.iacr.org/ 6

[19] R. Canetti and S. Goldwasser, *An Efficient Threshold Public Key Cryptosystem Secure Against Adaptive Chosen Ciphertext Attack*, Eurocrypt '99, LNCS 1592, Springer-Verlag, 1999, 90–106 5

[20] D. Dolev and A. C. Yao, *On the Security of Public Key Protocols*, IEEE Transactions on Information Theory 29/2 (1983) 198–208 1, 5

[21] R. Gennaro and S. Micali, *Verifiable Secret Sharing as Secure Computation*, Eurocrypt '95, LNCS 921, Springer-Verlag, 1995, 168–182 5

[22] O. Goldreich, *Foundations of Cryptography (Fragments of a Book)*, Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, February 1995, available with updates at http://theory.lcs.mit.edu/~oded/ 43

[23] O. Goldreich, *Secure Multi-Party Computation*, Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Version 1.1, September 1998, available from http://www.wisdom.weizmann.ac.il/users/oded/pp.htm 5

[24] O. Goldreich, S. Micali and 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 5

[25] S. Goldwasser and L. Levin, *Fair Computation of General Functions in Presence of Immoral Majority*, Crypto '90, LNCS 537, Springer-Verlag, 1991, 77–93 5

[26] S. Goldwasser and S. Micali, *Probabilistic Encryption*, J. of Computer and System Sciences 28 (1984) 270–299 17, 38, 45

[27] S. Goldwasser, S. Micali and C. Rackoff, *The Knowledge Complexity of Interactive Proof Systems*, SIAM J. on Computing 18/1 (1989) 186–207 9

[28] S. Goldwasser, S. Micali and R. L. Rivest, *A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks*, SIAM J. on Computing 17/2 (1988) 281–308 39

[29] M. Hirt and U. Maurer, *Player Simulation and General Adversary Structures in Perfect Multiparty Computation*, J. of Cryptology 13/1 (2000) 31–60 5

[30] S. Chari, C. Jutla, J. R. Rao, and P. Rohatgi, *Towards sound approaches to counteract power-analysis attacks*, In Advances in Cryptology – CRYPTO '99, LNCS 1666, Springer-Verlag, 398–412 73

[31] P. Kocher, J. Jaffe and B. Jun, *Differential power analysis*, In Advances in Cryptology – CRYPTO '99, LNCS 1666, Springer-Verlag, 399–397 73

[32] P. C. Kocher, *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, In N. Koblitz, editor, Advances in Cryptology – CRYPTO '96, LNCS 1109, Springer-Verlag, 1996, 104–113 72

[33] M. G. Kuhn and R. J. Anderson, *Soft tempest: Hidden data transmission using electromagnetic emanations*, In Information Hiding—Second International Workshop, LNCS 1525, Springer-Verlag, 1998 73

[34] S. A. Kurtz, S. R. Mahaney, J. S. Royer and J. Simon, *Biological computing*, In L. Hemaspaandra and A. Selman, editors, Complexity Theory Retrospective II, Springer-Verlag, 1996, 179–195 65

[35] A. K. Lenstra and E. R. Verheul, *Selecting cryptographic key sizes*, Journal of Cryptology, 14(4), 2001. 66

[36] B. Lampson, *A note on the confinement problem*, Communications of the ACM, 16(10):613–615, 1973. 71

[37] P. Lincoln, J. Mitchell, M. Mitchell and A. Scedrov, *A Probabilistic Poly-Time Framework for Protocol Analysis*, 5th Conf. on Computer and Communications Security (CCS), ACM, 1998, 112–121 5, 6

[38] P. Lincoln, J. Mitchell, M. Mitchelland A. Scedrov, *Probabilistic Polynomial-Time Equivalence and Security Analysis*, Formal Methods '99, LNCS 1708, Springer-Verlag, 1999, 776–793 5, 6, 7

[39] N. Lynch, *Distributed Algorithms*, Morgan Kaufmann, San Francisco 1996 7

[40] N. Lynch, *I/O Automaton Models and Proofs for Shared-Key Communication Systems*, 12th Computer Security Foundations Workshop (CSFW), IEEE, 1999, 14–29 5, 6

[41] S. Micali and P. Rogaway, *Secure Computation*, Crypto '91, LNCS 576, Springer-Verlag, 1992, 392–404 5

[42] R. Ostrovsky and M. Yung, *How To Withstand Mobile Virus Attacks*, 10th Symp. on Principles of Distributed Computing (PODC), ACM, 1991, 51–59 28

[43] B. Pfitzmann, M. Schunter and M. Waidner, *Cryptographic Security of Reactive Systems*, Electronic Notes in Theoretical Computer Science (ENTCS) 32 (2000), http://www.elsevier.nl/cas/tree/store/tcs/free/noncas/pc/menu.htm 1, 2, 5

[44] B. Pfitzmann, M. Schunter and M. Waidner, *Secure Reactive Systems*, IBM Research Report RZ 3206 (#93252), IBM Research Division, Zürich, May 2000, available at http://www.semper.org/sirene/lit/abstrA0.html#PfSW1_00 15, 17

[45] B. Pfitzmann, M. Schunter and M. Waidner, *Provably Secure Certified Mail*, IBM Research Report RZ 3207 (#93253), IBM Research Division, Zürich, August 2000 available at http://www.semper.org/sirene/lit/abstrA0.html#PfSW2_00 61

[46] B. Pfitzmann, M. Steiner and M. Waidner, *A Formal Model for Multiparty Group Key Agreement*, IBM Research Report RZ 3383 (# 93419), IBM Research Division, Zürich, Feb. 2002 4, 35, 61

[47] B. Pfitzmann and M. Waidner, *A General Framework for Formal Notions of "Secure" System*, Hildesheimer Informatik-Berichte 11/94, Universität Hildesheim, April 1994, available at http://www.semper.org/sirene/lit/abstr94.html#PfWa_94 5

[48] B. Pfitzmann and M. Waidner, *Composition and Integrity Preservation of Secure Reactive Systems*, 7th Conf. on Computer and Communications Security (CCS), ACM, 2000, 245–254 5, 6, 15, 28

[49] J.-J. Quisquater and D. Samyde, *ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards*, E-smart 2001, LNCS 2140, Springer-Verlag, 2001, 200–210 73

[50] C. Rackoff and D. R. Simon, *Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack*, Crypto '91, LNCS 576, Springer-Verlag, 1992, 433–444 38

[51] J. R. Rao and P. Rohatgi, *EMpowering side-channel attacks*, Report 2001/037, Cryptology ePrint Archive, May 2001 73

[52] F. Schneider, *Enforceable Security Policies*, ACM Transactions on Information Security, 3/1 (2001), 30–50 64

[53] R. Segala and N. Lynch, *Probabilistic Simulations for Probabilistic Processes*, Concur '94, LNCS 836, Springer-Verlag, 1994, 481–497 6, 7

[54] A. C. Yao, *Protocols for Secure Computations*, 23rd Symp. on Foundations of Computer Science (FOCS), IEEE, 1982, 160–164 5

[55] A. C. Yao, *Theory and Applications of Trapdoor Functions*, 23rd Symp. on Foundations of Computer Science (FOCS), IEEE, 1982, 80–91 15