RZ 3207 (#93253) 02/14/00
Computer Science/Mathematics 39 pages

Research Report

Provably Secure Certified Mail

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

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

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

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

Provably Secure Certified Mail

Birgit Pfitzmann, Matthias Schunter, Michael Waidner*

August 2000

Abstract

With a certified-mail protocol, one fairly exchanges a message for a receipt. No satisfactory protocols without any third party are possible, hence optimistic protocols are the best one can hope for. Here a third party is only involved if one party tries to cheat.

Certified-mail protocols are known in the literature, but there was no rigorous definition yet, in particular for the optimistic case and for many interleaved executions. We provide two such definitions. One defines individual integrity and secrecy requirements. The other defines an ideal system and uses a general simulatability definition. We show the relation between the definitions, present an efficient protocol, and prove its security in detail.

Apart from the intrinsic benefits of provably secure certified mail, this paper serves as an example that a serious-sized practical protocol can be rigorously proven with respect to a general simulatability definition and an abstract specification accessible to formal methods.

1 Introduction

A certified-mail protocol enables the fair exchange of a message for a receipt [B83, R83]. This means that either the recipient gets the message and the sender gets a receipt, i.e., the ability to convince any verifier that the recipient obtained the message, or neither gets anything; in particular the message should then remain entirely secret.

There are three classes of certified-mail protocols. Protocols with an *in-line third party* are quite straightforward [R83]. The older cryptographic literature treats *two-party* protocols, e.g., [B83, R83, VV83, G84], i.e., only the sender and the recipient are involved. These protocols are based on gradual exchange of secrets. However, their error probabilities only decrease linearly in the number of rounds, and one can show that this is unavoidable.[1] Hence they are not used in practice. *Optimistic* protocols use a trusted third party to ensure fairness, but it is not actively involved if both sender and recipient follow the protocol [ASW97, ASW00, M97, BDM98, ZG97].

In this paper, we provide precise definitions of certified mail in two different methodologies. We first define individual properties of the protocols, e.g., the correctness of receipts (an integrity property) and secrecy of the message if no receipt is obtained (a secrecy property). Most cryptographic definitions of individual system classes are of this type. A specific aspect of ours is that to a large extent we use a general system model, and that we formulate at least the integrity properties abstractly using general translations to concrete versions from [PW00]. (Somewhat general reactive system models with interleaved executions were first introduced in [BR94], and [P93] first sketched the use of abstract integrity definitions.) The secrecy definitions follow semantic security (see [Y82a, GM84]) and security against adaptive chosen-message attacks [RS92]. Semantic security is slightly modified because we

*This work was partially supported by the MAFTIA project. A preliminary version is contained in [S00].

[1] In [BGMR90] such a lower bound was proven for two-party contract signing. Contract signing can be reduced to the variant of certified mail defined here; the reduction needs only one additional round of communication and does not increase the error probability significantly [S00]. (The reduction is given for optimistic protocols only, but one can easily see that it also works for 2-party protocols.)

only want it if no receipt is obtained. The most general adaptive attacks for general secrecy properties are not perfectly clear; even for simple encryption systems they are still being extended [BBM00]. This was one reason why we also provide the second definition which, if anything, is too strict, while the first might rather be too lax.

As the second definition, we define an ideal system for certified mail and use a general simulatability definition. Thus the actual definition of the real systems is that they must be at least as secure as the ideal system in a certain sense; essentially, everything an adversary can achieve in the real system must also be achievable in the ideal system. The ideal system is specific to certified mail. We first present a naive version and then show why the actual definition must be more complicated if practical protocols are to fulfil it. The modifications needed can serve as a methodology to define ideal systems for various other protocol classes. General simulatability definitions are based on work for secure function evaluation [Y82, GL91, B91, MR92, C00]. Extensions to reactive systems have, after several sketches, been defined in detail in [HM00, LMMS98, PSW00b]. (Here we only mentioned definitions where only the input-output behavior of the ideal system is to be simulated, as we need it here. A shorter published version of [PSW00b] is [PSW00a].) We use the latter because it is more general than the first and allows more abstraction than the second one (for details see [PSW00b]) and it has the only reactive composition theorem [PW00].

We show that the ideal-system definition is a refinement of the property-based definition, i.e., it has the required properties, while the properties leave some room for different ideal systems. A general theorem from [PW00] then implies that any real system as secure as the ideal system also has the integrity properties. (The same should hold for the secrecy property, but we do not show this here.)

We then present a protocol for certified mail and prove its security. The protocol is based on that in [ASW97], but it needed modifications to enable simulatability. To the best of our knowledge, this is the first rigorous proof of a certified-mail protocol; certainly the first for an optimistic protocol, and the first in a setting with many interleaved runs of the protocol. As we aim at proofs of real-life protocols, we have taken care to rigorously prove also the non-cryptographic aspects, e.g., realistic local message dispatching to different subprotocols.

2 Overview of a Concrete Protocol

Rigorous definitions are long. In this section, we will therefore anticipate the actual protocol that we will prove in an informal description so that it is clear what kind of protocols we are talking about.

The protocol is sketched in Figure 1. More precisely, one run of the main subprotocol "send" of the system is shown. (There is another, simpler subprotocol "show" for showing receipts.) It is a synchronous protocol involving three machines, M_s, M_r, and M_t for a sender, a recipient, and a third party. It is started by inputs (send, r, l, m) to M_s and $(\mathsf{receive}, s, l)$ to M_r. In real life, the inputs are made by two different users (or application programs running on their behalf) and the "machines" are software on each user's device. In the inputs, send and $\mathsf{receive}$ are constants serving as command names, r and s are the indices of recipient and sender serving as addresses, m is the message the sender wants to send, and l ("label") is a subject for the mail.

The use of subjects is a design choice (alternatives are shown in [S00]). The idea in this version, called "labeled certified mail" is that the recipient has the choice whether he wants to receive a message with a particular label. The receipt will unambiguously contain both the label, which the recipient agreed upon, and the message that was then sent under this label.

We use three cryptographic primitives (more formal definitions are given in Section 6.1):

- A signature scheme [GMR 88]. We write $\mathsf{sign}_u(m)$ for the pair of a message m and its signature with the secret key of a participant u.
- A one-way function owf. We use it for one-time signatures, i.e., $\mathsf{owf}(r)$ for a random r is a one-time public key, and later r is revealed as the signature. (This is only an efficiency improvement

over using normal signatures.)

- A non-interactive chameleon commitment scheme [BCC88]. A commitment is denoted by $c = \mathsf{com}_t(m, r)$, where m is the message, r a suitable random value (chosen with an algorithm $\mathsf{gen}_{\mathsf{CR},t}$) and t the party whose public commitment key is used. A commitment keeps m secret, but the committer must be unable to open it in two ways, i.e., to show two pairs (m, r) and (m', r') with $m \neq m'$ and $c = \mathsf{com}_t(m, r) = \mathsf{com}_t(m', r')$. Chameleon means that the party t generating the public key of the scheme can take an opened commitment on a message m and open it to any other m'.

The main ideas underlying the protocol are the following: Messages m_1 and m_2 are promises to send a message under label l and to produce a receipt for it, respectively. The value i is the number of the starting round and k a security parameter. If both parties are honest, M_s reveals the message m in m_3, and M_r sends the one-time signature r_R as a receipt in m_4. The entire receipt is $m_7 = (m_1, m_2, m_3, m_4)$; it can easily be verified by a verifier M_v. If a dishonest recipient does *not* send m_4, the sender uses the recipient's promise m_2 in m_5 to convince the third party that the recipient wanted to receive a message under this label. Thus the third party can safely issue an affidavit, m_6. Then M_s uses m_6 as a receipt, which can again be easily verified by M_v. If a dishonest sender does not send the message in m_3, the recipient waits until Round $i + 6$. If m_6 arrives, the recipient extracts m_3 and thus m from it. Otherwise it knows that the message will never arrive and can safely decide that the transaction failed. For the latter, the third party must honor m_5 only if it arrives in Round $i + 5$.

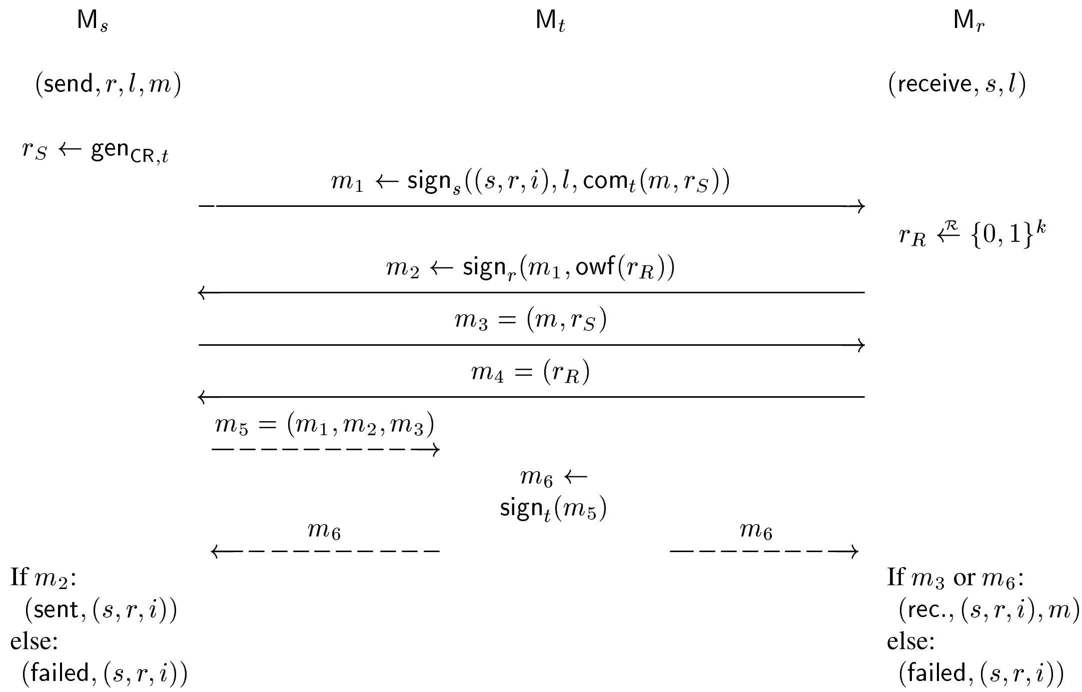

Figure 1: Subprotocol run of labeled certified mail. Dashed flows are only needed if m_4 is missing, i.e., the protocol is optimistic. Some message-type identifiers are omitted.

The construction is based on [ASW97]. There m_1 contained a committing encryption of m, which would not be simulatable. Therefore we used a chameleon commitment instead. Unlike [ASW97], we did not use separate transaction identifiers tid because the triple (s, r, i) already plays this role if we allow each sender to start one protocol run with each recipient in each round. Everything immediately generalizes to any other constant number per round. (Omitting the round number i instead is not possible in this protocol because M_t uses i as included in m_1 to decide whether a complaint m_5 does not come too late.) A translation into more application-oriented tid's can easily be built on top.

In Section 6 we will rigorously define the machines M_s, M_r, M_t, and M_v, including how they handle the simultaneous runs of many subprotocols.

3 Summary of the System Model

In this section, we repeat the basic definitions from [PSW00b] in slightly abbreviated form. They are for a synchronous network model, and the simulatability also includes the timing. Hence security vulnerabilities via timing channels are exposed. To avoid that timing differences within a round leak, implementations of synchronous machines have to ensure that input reading and outputting are both clocked.[2]

The machine model is probabilistic state-transition machines, similar to probabilistic I/O automata as sketched in [L96]. For clarity, one particular notation and semantics is fixed.

Definition 3.1 (Machines and Ports) A *name* is a string over a fixed alphabet Σ.

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

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

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

of a finite set of ports, a probabilistic state-transition function, and sets of initial and final states. The states are strings s from Σ^*. The inputs are tuples $I = (I_p)_{p \in \mathsf{In}(Ports_{\mathsf{M}})}$ of one input $I_p \in \Sigma^*$ per input port, and the outputs analogous tuples O. δ_{M} maps each such pair (s, I) to a finite distribution over pairs (s', O). For a set M of machines, let $\mathsf{ports}(M) = \bigcup_{\mathsf{M} \in M} Ports_{\mathsf{M}}$.[3]

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

For computational aspects, each machine is regarded as implemented by a probabilistic interactive Turing machine [GMR89], and each port by a communication tape. The complexity of a machine is measured in terms of the length of the initial state, represented as initial worktape content (often a security parameter).

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

Definition 3.2 (Machine Collections, Runs and Views) A *collection* C is a finite set of machines with pairwise disjoint sets of ports. Each set of complementary ports $c = \{p, p^c\} \subseteq \mathsf{ports}(C)$ is called a *connection* and the set of these connections the *connection graph* $\mathsf{G}(C)$. By $\mathsf{free}(C)$ we denote the *free* ports, i.e., $p \in \mathsf{ports}(C)$ but $p^c \notin \mathsf{ports}(C)$. A collection is *closed* if $\mathsf{free}(C) = \emptyset$.

A *clocking scheme* is a mapping κ (also written as a tuple) from a set $\{1, \ldots, n\}$ to the powerset of C, i.e., it assigns each number a subset of the machines. Given C and κ and a tuple $ini \in Ini = \times_{\mathsf{M} \in C} Ini_{\mathsf{M}}$ of initial states, *runs* (or "executions" or "traces") are defined: Each global round i has n subrounds. In Subround $[i.j]$, all machines $\mathsf{M} \in \kappa(j)$ switch simultaneously, i.e., each

[2]Of course we do not suggest that one should not consider asynchronous systems, only that one needs a simulatability definition for each model.

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

state-transition function δ_{M} is applied to M's current inputs and state and yields a new state and output (probabilistically). The output at a port p! is available as input at p? until the machine with port p? is clocked next. If several inputs arrive until that time, they are concatenated. This gives a family of random variables

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

More precisely, each run is a function mapping each triple $(\mathsf{M}, i, j) \in C \times \mathbb{N} \times \{1, \ldots, n\}$ to a quadruple (s, I, s', O) of the old state, inputs, new state, and outputs of machine M in subround $[i.j]$, or instead to ϵ if M not clocked in this subround. For a number $l \in \mathbb{N}$ of rounds, l-round prefixes $run_{C,ini,l}$ of runs are defined in the obvious way. For a function $l : Ini \to \mathbb{N}$, this gives a family $run_{C,l} = (run_{C,ini,l(ini)})_{ini \in Ini}$.

The view of a subset M of a closed collection C in a run r is the restriction of r to $M \times \mathbb{N} \times \{1, \ldots, n\}$.[4] This gives a family of random variables

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

and similarly for l-round prefixes.

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

Now we define specific machine collections as we need them in the security definitions, first the system part and then the environment, i.e., users and adversaries. Typically, a cryptographic system is described by an *intended structure*, and the actual structures are derived using a *trust model*: the adversary replaces some machines and taps or completely controls some channels. A concrete derivation is defined in [PSW00b]. However, as a wide range of trust models is possible, it is useful to keep the remaining definitions independent of them by a general system definition.

Definition 3.3 (Structures and Systems) A *structure* is a pair $struc = (M, S)$ where M is a collection of machines called *correct machines*, and $S \subseteq \mathsf{free}(M)$ is called *specified ports*. Let $\bar{S} = \mathsf{free}(M) \setminus S$ and $\mathsf{forb}(M, S) = \mathsf{ports}(M) \cup \bar{S}^c$.

A *system* Sys is a set of structures. ◇

The separation of the free ports into specified ports and others is an important feature of this particular reactive simulatability definition. The specified ports are those where a certain service is guaranteed. Typical examples of inputs at specified ports are "send message m to r" for a message transmission system or "pay amount x to id" for a payment system. The ports in $\bar{S}$ are additionally available for the adversary. The ports in $\mathsf{forb}(M, S)$ will therefore be forbidden or at least unusual for an honest user to have. In the simulatability definition below, only the events at specified ports have to be simulated one by one. This allows *abstract* specification of systems with *tolerable imperfections*. For instance, if the traffic pattern is not hidden (as in almost all cryptographic protocols for efficiency reasons), one can abstractly specify this by giving the adversary one busy-bit per message in transit in the ideal system. Even better, he should only get one busy-bit per subprotocol run (e.g., a payment) and the internal message pattern of the subprotocol should not tell him more. More motivation and an example with just this busy-bit (secure channels) is given in [PSW00b]; we use a similar abstraction in this paper, see Section 5.2.

The following definition contains another important aspect: Both honest users and an adversary are modeled as stateful machines H and A apart from the system. First, honest users should not be modeled as part of the machines in M because they are arbitrary, while the machines have prescribed programs. Secondly, they should not be replaced by a quantifier over input sequences, because they may have arbitrary strategies which message to input next to the system after obtaining certain outputs. They may

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

even be influenced in these choices by the adversary, e.g., in chosen-message attacks on a signature scheme; thus H and A may communicate. At least in the computational case, arbitrary strategies (i.e., adaptive attacks) are not known to be replaceable by arbitrary input sequences. Thirdly, honest users are not a natural part of the adversary because they are supposed to be protected from the adversary. In particular, they may have secrets and we want to define that the adversary learns nothing about those except what he learns "legitimately" from the system (this depends on the specification) or what the user tells him directly.

Definition 3.4 (Configuration) A *configuration conf* of a system Sys is a tuple $(M, S, \mathsf{H}, \mathsf{A})$ where $(M, S) \in Sys$ is a structure and H and A are machines such that $C = M \cup \{\mathsf{H}, \mathsf{A}\}$ a closed collection.

The set of configurations is written $\mathsf{Conf}(Sys)$, and those with polynomial-time user and adversary $\mathsf{Conf}_{\mathsf{poly}}(Sys)$. "poly" is omitted if it is clear from the context. The set of guessing-output configurations $\mathsf{Conf}_{\mathsf{g}}(Sys)$ is defined like $\mathsf{Conf}(Sys)$ except that all adversaries have a free output port guess!.

Runs and views of a configuration are given by Definition 3.2 with the clocking scheme $(M \cup \{\mathsf{H}\}, \{\mathsf{A}\}, \{\mathsf{H}\}, \{\mathsf{A}\})$, except that we end a run if H and A have reached finite states. Typically, the initial states of all machines are only a security parameter k (in unary representation). Then we consider the families of runs and views restricted to the subset $Ini' = \{(1^k)_{\mathsf{M} \in C} | k \in \mathbb{N}\}$ of Ini, and write run_{conf} and $view_{conf}(M)$ for run_C and $view_C(M)$ restricted to Ini', and similar for l-round prefixes. Furthermore, Ini' is identified with $\mathbb{N}$; hence we can write $run_{conf,k}$ etc. ◇

Clocking the adversary between the correct machines is the well-known model of rushing adversaries [BD84]. The given clocking of users is as powerful as clocking them in an arbitrary unsynchronized way [PSW00b].

4 Requirements-Based Definition of Labeled Certified Mail

In this section, we give a definition of labeled certified mail by individual integrity and secrecy requirements. As there are several integrity requirements, it is worthwhile repeating a general definition from [PW00] that translates abstract (perfect) requirements into different types of cryptographic definitions.

4.1 Integrity Requirements in General

Integrity requirements on reactive systems outside cryptography are typically written in a temporal logic or a predicate logic including round numbers. We generalize this by giving a cryptographic semantics to all requirements that can be expressed as sets of allowed traces (sequences of events) at the specified ports of the system. We first need general notation about small functions.

Definition 4.1 (Small Functions) By "a class $SMALL$ of small functions" we mean that $SMALL$ is a set of functions from $\mathbb{N}$ to $\mathbb{R}_{\geq 0}$ which is closed under addition and, with a function g, also contain any function $g' \leq g$.

One typical class is $EXPSMALL$ of all functions g for which a polynomial Q exists with $\forall k : g(k) \leq Q(k) \cdot 2^{-k}$.

Another is the (larger) class $NEGL$ of all functions g where for all positive polynomials Q, $\exists k_0 \forall k \geq k_0 : g(k) \leq 1/Q(k)$. Equivalently, one writes $g(k) \leq 1/\mathsf{poly}(k)$. ◇

Definition 4.2 (Integrity Requirements) An integrity requirement Req for a system Sys is a function that maps each set S with $(M, S) \in Sys$ to a set of traces at the ports in S. More precisely, such a trace contains one value $v_p \in \Sigma^*$ for each port $p \in S$ and Round i, corresponding to the in- or output of a correct machine in Subround $[i.1]$. For the computational and statistical case, the traces must be finite. We say that Sys fulfills Req

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

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

$$P(run_{conf,k,l(k)}\lceil_S \notin Req(S)) \in SMALL.$$

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

$$P(run_{conf,k}\lceil_S \notin Req(S)) \in NEGL.$$

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

4.2 System Class of Certified Mail

In order to express integrity requirements according to Definition 4.2, we need to define the possible sets S with $(M, S) \in Sys$ for any certified-mail system.

Definition 4.3 (Parameters and Notation for Labeled Certified Mail) The parameters of a labeled certified-mail scheme are a message space $Msg \subseteq \Sigma^*$, a label space $L \subseteq \Sigma^*$, numbers $n_S, n_R \in \mathbb{N}$ of intended senders and recipients, and an intended number $\Delta \in \mathbb{N}$ of rounds of the send protocol. Let Par_{CM} be the set of possible tuples of these parameters.

If a tuple $par \in Par_{CM}$ is clear from the context, we simply address its components by the names introduced above. We then also use the notation $n = n_S + n_R + 1$ for the number of parties, $v = n$ for the index of the verifier, and $\mathcal{M} = \{1, \ldots, n\}$, $\mathcal{M}_S = \{1, \ldots, n_S\}$ and $\mathcal{M}_R = \{n_S + 1, \ldots, n_S + n_R\}$ for the sets of (indices of) all parties, senders and recipients.

We write $Slots = \mathcal{M}_S \times \mathcal{M}_R \times \mathbb{N}$. As explained in Section 2, this set is used as the opportunities for protocol runs and thus as transaction identifiers. Let also $Slots_i = \mathcal{M}_S \times \mathcal{M}_R \times \{1, \ldots, i\}$ for all $i \in \mathbb{N}$. $\diamond$

Definition 4.4 (Access Structure and Specified Ports for Labeled Certified Mail) The access structure for certified mail with parameters $par \in Par_{CM}$ is the set

$$\mathcal{ACC}^{\mathsf{CM}}_{par} = \{\mathcal{H} \subseteq \mathcal{M} | v \in \mathcal{H}\},$$

i.e., the verifier is always honest, while all other parties may be dishonest. For each $\mathcal{H} \in \mathcal{ACC}^{\mathsf{CM}}_{par}$, we define a set of specified ports as

$$S_{\mathcal{H}} = \{\mathsf{in}_u?, \mathsf{out}_u! | u \in \mathcal{H}\} \setminus \{\mathsf{in}_v?\},$$

i.e., one port pair for each user except that the verifier does not need an input port.

If a set $\mathcal{H}$ is clear from the context, we abbreviate the (indices of) corrupted parties by $\mathcal{A} = \mathcal{M} \setminus \mathcal{H}$, and correct and corrupted senders and recipients, respectively, by $\mathcal{H}_S = \mathcal{M}_S \cap \mathcal{H}$, $\mathcal{A}_S = \mathcal{M}_S \cap \mathcal{A}$, $\mathcal{H}_R = \mathcal{M}_R \cap \mathcal{H}$, and $\mathcal{A}_R = \mathcal{M}_R \cap \mathcal{A}$. $\diamond$

We also define the desired input format so that we can use it in both definitions. According to Definition 3.1, the machines must be defined for all inputs in Σ^*, but incorrect inputs will be ignored.

Definition 4.5 (Input Domains for Labeled Certified Mail) Let parameters $par \in Par_{CM}$ be given. The desired input set In_S for a sender consists of the vectors $(in_r)_{r \in \mathcal{M}_R \cup \{v\}}$ where each entry in_r for $r \in \mathcal{M}_R$ is ϵ or (send, r, l, m) with $l \in L$ and $m \in Msg$, and in_v is ϵ or $(\mathsf{show}, (s, r, j))$ with $(s, r, j) \in Slots$.[5]

[5]The representation as vectors is for our convenience; in reality, any deterministic shorter encoding can be used.

The desired input set In_R for recipients consists of the vectors $(in_s)_{s\in\mathcal{M}_S}$ where each entry in_s must be ϵ or $(\mathsf{receive}, s, l)$ with $l \in L$. $\diamond$

4.3 Requirements on Labeled Certified Mail

We now present actual requirements on certified mail. Readers unused to formulations in terms of interfaces may already imagine that each port pair $(\mathsf{in}_u?, \mathsf{out}_u!)$ are the in- and outputs of a machine M_u. Requirement a) is not always considered below because, in contrast to the other requirements, it presupposes authentic connections between sender and recipient in real systems.

For finite sequences, "after at most Δ rounds" is defined to be automatically fulfilled if less than Δ further rounds exist.

Definition 4.6 (Integrity Requirements on Labeled Certified Mail) Let parameters $par \in Par_{CM}$ be given. Then the integrity requirements on certified mail are the following functions Req_x with. $x = a, \ldots, h$ (corresponding to the item labels below) mapping each set $S_\mathcal{H}$ with $\mathcal{H} \in \mathcal{ACC}^{\mathsf{CM}}_{par}$ to a predicate $Req_x(S_\mathcal{H})$.

We make the general precondition that only correct inputs are made at the specified ports (intuitively, by honest users), i.e., all predicates $Req_x(S_\mathcal{H})$ are fulfilled in all runs where an input $I \notin In_S$ is made at a port $\mathsf{in}_s?$ with $s \in \mathcal{H}_S$ in any round, or $I \notin In_R$ at a port $\mathsf{in}_r?$ with $r \in \mathcal{H}_\mathcal{R}$.[6] By "an input occurs" we mean that it is an entry of an input vector; similarly, the outputs will actually be members of sets.

For all $s \in \mathcal{H}_S, r \in \mathcal{H}_R$ (i.e., if sender and recipient are correct):

a) *Correct Execution.* If (send, r, l, m) is input at $\mathsf{in}_s?$ and $(\mathsf{receive}, s, l')$ at $\mathsf{in}_r?$ in Round i, then after at most Δ rounds: If $l = l'$, outputs $(\mathsf{sent}, (s, r, i))$ and $(\mathsf{received}, (s, r, i), m)$ occur at $\mathsf{out}_s!$ and $\mathsf{out}_r!$, respectively, otherwise $(\mathsf{failed}, (s, r, i))$ at both these ports.

b) *Unforgeable Messages.* If an output $(\mathsf{received}, (s, r, i), m)$ occurs at $\mathsf{out}_r!$ in a round j after an input $(\mathsf{receive}, s, l)$ at $\mathsf{in}_r?$ in a round $i \leq j$, then an input (send, r, l, m) occurred at $\mathsf{in}_s?$ in a round $i' \leq j$.

For all $r \in \mathcal{H}_R, s \in \mathcal{M}_S$ (i.e., for every correct recipient):

c) *Termination for Recipient.* An input $(\mathsf{receive}, s, l)$ at $\mathsf{in}_r?$ in Round i leads to an output $(\mathsf{received}, (s, r, i), m)$ with $m \in Msg$ or $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out}_r!$ after at most Δ rounds, and no second output of these types with this (s, r, i) occurs at $\mathsf{out}_r!$.

d) *Unforgeable Receipts.* If an output $(\mathsf{received}, (s, r, i), l, m)$ occurs at $\mathsf{out}_v!$ in a round j, then $m \in Msg$ and $i \leq j$, and an input $(\mathsf{receive}, s, l)$ occurred at $\mathsf{in}_r?$ in Round i.

e) *No Surprises for the Recipient.* If an output $(\mathsf{failed}, (s, r, i))$ occurs at $\mathsf{out}_r!$ after an input $(\mathsf{receive}, s, l)$ at $\mathsf{in}_r?$ in Round i, then no output $(\mathsf{received}, (s, r, i), l, m)$ with any $m \in \Sigma^*$ occurs at $\mathsf{out}_v!$ in any round.

f) *Fixed Receipts.* If an output $(\mathsf{received}, (s, r, i), m)$ occurs at $\mathsf{out}_r!$ after an input $(\mathsf{receive}, s, l)$ at $\mathsf{in}_r?$ in Round i, then no output $(\mathsf{received}, (s, r, i), l, m')$ for any different m' occurs at $\mathsf{out}_v!$ in any round.

For all $s \in \mathcal{H}_S$, $r \in \mathcal{M}_R$ (i.e., for all correct senders):

g) *Termination for Sender.* An input (send, r, l, m) at $\mathsf{in}_s?$ in Round i leads to an output $(\mathsf{sent}, (s, r, i))$ or $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out}_s!$ after at most Δ rounds, and no second such output occurs at $\mathsf{out}_s!$.

[6] More specific preconditions would strengthen the requirements. As we will have a strict definition via an ideal system later, we use a weak (and thus very general) variant here.

h) *Verifiability of Valid Receipts.* If an output $(\mathsf{sent}, (s, r, i))$ occurs at $\mathsf{out}_s!$ after an input (send, r, l, m) at $\mathsf{in}_s?$ in Round i, then a later input $(\mathsf{show}, (s, r, i))$ at $\mathsf{in}_s?$ leads to the output $(\mathsf{received}, (s, r, i), l, m)$ at $\mathsf{out}_v!$ within at most Δ rounds.

$\diamond$

By [PW00] (Theorem 3.2), any logical derivations from these requirements are valid also for statistical and computational fulfillment. Hence one can draw conclusions on this abstract level, or join requirements d) to f) into one, etc.

For a relation to formal methods, where avoiding arithmetic is helpful, note that although the current formulation is standard predicate logic using round numbers, most of the requirements could be expressed in temporal logic. Only requirements that mention "after at most Δ rounds" (availability) intrinsically need the round numbers, and this could be extracted into the termination requirements alone.

Finally, we define the secrecy requirement. Here there is no general framework like Definition 4.2, hence we write in a standard cryptographic style.

Definition 4.7 (Secrecy Requirement on Labeled Certified Mail) Let parameters $par \in Par_{CM}$ and a system Sys_{par}^{CM} with the correct user interfaces be given, i.e., for all $(M, S) \in Sys_{par}^{\mathsf{CM}}$ there exists $\mathcal{H} \in \mathcal{ACC}_{par}^{\mathsf{CM}}$ with $S = S_{\mathcal{H}}$.

Roughly, we say that Sys_{par}^{CM} offers *message secrecy* if the message m is kept entirely secret whenever the output $(\mathsf{failed}, (s, r, i))$ occurs at $\mathsf{out}_s!$ on input (send, r, l, m) at $\mathsf{in}_s?$ in Round i.

More precisely, consider the following user machine H_s for any $s \in \mathcal{M}_S$: It only has ports $\mathsf{in}_s!$ and $\mathsf{a\_in}?$ for making inputs to M_s and for being influenced by the adversary. Its transition function is as follows:[7]

1. Initially, in each round $i - 1$, H_s forwards an input vector $(a\_in_r)_{r \in \mathcal{M}_R \cup \{v\}}$ made at $\mathsf{a\_in}?$ to $\mathsf{in}_s!$ if it lies in In_s.
2. If in one round, an element a_in_r with $r \in \mathcal{M}_R$ is $(\mathsf{choice}, r, l, m_0, m_1)$ with $m_0, m_1 \in Msg$ and $l \in L$, then H_s chooses a random bit $b \in \{0, 1\}$ and uses $(\mathsf{send}, r, l, m_b)$ instead of this entry when outputting the vector at $\mathsf{in}_s!$.
3. From then on, H_s acts as before except for not considering a choice input again.

We consider all configurations $conf = (M, S, \mathsf{H}_s, \mathsf{A}) \in \mathsf{Conf_g}(Sys)$ with $s \in \mathcal{H}_S$. Recall that the index g means that A has a free output port guess!; in our case we call the output there b^*; it should be a bit meant as a guess at b.

Let $\mathsf{no\_receipt}$ denote the event that the output $(\mathsf{failed}, (s, r, i))$ occurs at $\mathsf{out}_s!$ where r is the index of the choice input and $i - 1$ the round where H_s handled it, and $\mathsf{adv\_win}$ the event that $\mathsf{no\_receipt}$ is true but the guess is correct, i.e., $b^* = b$. (Both are well-defined in the probability space of the runs of *conf* for any initial input k.) The requirement is that[8]

$$P(\mathsf{adv\_win}(run_{conf,k})) \ \leq \ \frac{1}{2} P(\mathsf{no\_receipt}(run_{conf,k})) + \frac{1}{\mathsf{poly}(k)}.$$

$\diamond$

We have not added an analog of abuse-freeness to the requirements, as sometimes required for contract signing [GJM99], because in our synchronous setting and with moderately short rounds we do not consider such abuse a threat in practice.

[7] It is similar to adaptive chosen-ciphertext attacks on encryption systems except that we only require secrecy if the exchange fails; hence we can allow the adversary to connect to all output ports of the correct machines.

[8] A definition that the conditional probability of $\mathsf{adv\_win}$ given $\mathsf{no\_receipt}$ is at most $1/2 + 1/\mathsf{poly}(k)$ would not work because of cases where the adversary always gives a receipt except if he can break an underlying assumption.

5 Ideal-System-Based Definition of Labeled Certified Mail

Now we present a specific ideal system as a second definition of labeled certified mail. We first summarize the definition of simulatability from [PSW00b]. We then discuss a naive ideal system and motivate why the actual ideal system is a bit more complicated. Then we show that the ideal system fulfils the requirements, i.e., the second definition is stricter, and we discuss what this means for real systems fulfilling the second definition.

5.1 Simulatability in General

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

Definition 5.1 (Valid Mapping, Suitable Configuration) A function f from a system Sys_1 to the powerset of a system Sys_2 is called a *valid mapping* if for all structures with $(M_2, S_2) \in f(M_1, S_1)$

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

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

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

As the definition of computational indistinguishability (originally from [Y82a]) is essential for the simulatability definition, we also present it here.

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

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

b) statistically indistinguishable ("$\approx_{SMALL}$") for a class $SMALL$ of small functions 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).

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

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

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

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

The following definition captures that whatever an adversary can achieve in the real system against certain honest users, another adversary can achieve against the same honest users in the ideal system. Adding an adversary output in the comparison does not make the definition stricter, nor do auxiliary inputs [PSW00b].

Definition 5.3 (Simulatability) Let systems Sys_1 and Sys_2 with a valid mapping f be given.

a) We say $Sys_1 \geq_{\mathsf{sec}}^{f,\mathsf{perf}} Sys_2$ (*perfectly at least as secure as* for f) if for any suitable configuration $conf_1 = (M_1, S_1, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^f(Sys_1)$, there exists a configuration $conf_2 = (M_2, S_2, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(Sys_2)$ with $(M_2, S_2) \in f(M_1, S_1)$ (and the same H) such that

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

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

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

Definition 5.4 (Blackbox and Universal Simulatability) Universal simulatability means that A_2 in Definition 5.3 does not depend on H (only on M_1, S_1, and A_1). Blackbox simulatability means that additionally, A_2 (given M_1, S_1, and the set P of adversary ports) is a fixed simulator Sim with A_1 as a blackbox submachine. $\diamond$

5.2 Naive Specification and Discussion

The simplest trusted host $\mathsf{TH}_{\mathsf{naive}}$ for certified mail can be sketched as follows (omitting verification):

Naive trusted host: Whenever $\mathsf{TH}_{\mathsf{naive}}$ obtains two matching inputs (send, r, l, m) at $\mathsf{in}_s?$ and $(\mathsf{receive}, s, l)$ at $\mathsf{in}_r?$ in a round i, it outputs $(\mathsf{received}, (s, r, i), m)$ at $\mathsf{out}_r!$ and $(\mathsf{sent}, (s, r, i))$ at $\mathsf{out}_s!$ in the next round. For any non-matched input, it outputs $(\mathsf{failed}, (s, r, i))$ to the party concerned.

Unfortunately, one cannot expect a real protocol to satisfy such a specification (the first two issues were also discussed in [PSW00b]):

- $\mathsf{TH}_{\mathsf{naive}}$ hides all interactions between honest parties from the adversary. But in real life, the adversary can eavesdrop on the channels and learn quite a lot in protocols like that in Figure 1. In particular the information "who communicates with whom at what time" would be extremely expensive to hide. As the secure-channel example in [PSW00b] shows how to treat just this minimal information, we decided here to model unencrypted protocols like that in Figure 1. Thus for each slot (s, r, i) the adversary will obtain an output $(\mathsf{busy}, \ldots)$ containing all non-secret information, and (msg, m) when a message is sent in clear.

- $\mathsf{TH}_{\mathsf{naive}}$ guarantees availability, i.e., the adversary cannot prevent honest parties from sending certified mails to each other. But in real systems the adversary can typically do this, e.g., by disrupting a channel. Therefore our trusted host will accept inputs $\mathsf{suppress}$ from the adversary that describe which protocol runs should be killed. (We will need authentic channels to trusted third parties, however.)

- $\mathsf{TH}_{\mathsf{naive}}$ produces outputs within one round, while any real optimistic protocol requires at least four rounds [S00]. Even protocols with more than four rounds should not be considered insecure just because of that. (Recall that our synchronous model requires that the behavior of a real system at the specified ports is indistinguishable from that of the ideal system even in its timing, so that a real system cannot have more timing channels than its specification.) Therefore we will use a number Δ of rounds as a free parameter in the specification. We decided to use only one such parameter here for brevity of the specification, i.e., $\Delta = 6$ for Figure 1, although alternatively M_s could output $(\mathsf{sent}, \ldots)$ after four rounds and M_r could output $(\mathsf{received}, \ldots)$ after three rounds in the all-correct case. Verification will work in one round.

- $\mathsf{TH}_{\mathsf{naive}}$ delivers protocol outputs after the same time Δ even to the adversary. However, an adversary can get these outputs faster. For almost all protocols, this holds by at least one round because the adversary can take a message from the line to him and do all computations still in the same round, while correct machines will only do them in the next round. In addition, in Figure 1, an adversarial recipient can immediately use the message after obtaining m_3, while a correct recipient only gets the output $(\mathsf{received}, (s, r, i), m)$ in Round 6. One way to model this would be to look at a real protocol and to model in the ideal system exactly until what round an adversary can disrupt the protocol, from when on he can obtain a message, etc. However, this would give a fairly large number of parameters in addition to Δ. Instead, our ideal system allows the adversary slightly more than a real adversary can achieve: Roughly, we allow him to disrupt at any time, or else to decide that he will not disrupt and then to use the results. This is represented in the treatment of adversary inputs $\mathsf{suppress}$ and $\mathsf{receive}$.[9]

5.3 An Ideal System for Certified Mail

We now describe an ideal system that formally specifies a particular version of labeled certified mail with the properties introduced in Section 5.2.

Scheme 5.1 (Trusted Host for Labeled Certified Mail) For any parameters $par \in Par_{CM}$ (Definition 4.3), an ideal system for labeled certified mail is defined as

$$Sys_{par}^{\mathsf{CM,id}} = \{(\{\mathsf{TH}_{\mathcal{H}}\}, S_{\mathcal{H}}) | \mathcal{H} \in \mathcal{ACC}_{par}^{\mathsf{CM}}\},$$

where $\mathcal{ACC}_{par}^{\mathsf{CM}}$ and $S_{\mathcal{H}}$ are as in Definition 4.4 and $\mathsf{TH}_{\mathcal{H}}$ is defined as follows.

Ports: Let $Ports_{\mathsf{TH}_{\mathcal{H}}} = S_{\mathcal{H}} \cup \bar{S}_{\mathsf{id},\mathcal{H}}$, where the unspecified ports are simply

$$\bar{S}_{\mathsf{id},\mathcal{H}} = \{\mathsf{out\_a}!, \mathsf{in\_a}?\}.$$

Overall structure of $\mathsf{TH}_{\mathcal{H}}$*:* Each slot $(s, r, i) \in Slots$ is handled by a submachine $\mathtt{th}_{s,r,i}$. For simplicity, we assume that in any round i, submachines $\mathtt{th}_{s,r,j}$ for all $(s, r, j) \in Slots_i$ exist, although most of them will remain in their starting state forever.[10] There are three types of submachines (where "type" means equality except for port renaming), one each for $(s, r) \in \mathcal{H}_S \times \mathcal{A}_R$, $\mathcal{A}_S \times \mathcal{H}_R$ or $\mathcal{H}_S \times \mathcal{H}_R$. Each

[9] This could also be done for the receipts. However, as our protocol has global round numbers, verifiers can simply refuse to accept receipts too early. Otherwise, the specification of the sending protocol and verification become more intertwined.

[10] As this is only an ideal system, there is no need to implement its internal state efficiently.

submachine $\mathsf{th}_{s,r,i}$ has one input and output port for each party concerned with this run; $\mathsf{in\_s}_{s,r,i}?$ and $\mathsf{out\_s}_{s,r,i}!$ if $s \in \mathcal{H}_S$, $\mathsf{in\_r}_{s,r,i}?$ and $\mathsf{out\_r}_{s,r,i}!$ if $r \in \mathcal{H}_R$, and in any case $\mathsf{out\_v}_{s,r,i}!$, $\mathsf{in\_a}_{s,r,i}?$, $\mathsf{out\_a}_{s,r,i}!$, and finally $\mathsf{in\_th}_{s,r,i}?$ for inputs from $\mathsf{TH}_\mathcal{H}$ itself.

In Rounds 0 and 1, the trusted host does nothing. (This is time reserved for initialization in the real system.) We now define the state transition for any round $i > 1$.

Global inputs and dispatching in Round i: At the port $\mathsf{in\_a}?$, the trusted host $\mathsf{TH}_\mathcal{H}$ expects a matrix $(in\_a_{i,(s,r,j)})_{(s,r,j)\in Slots_i}$ of sets of at most four elements.[11] $\mathsf{TH}_\mathcal{H}$ inputs each element to port $\mathsf{in\_a}_{s,r,j}?$ of $\mathsf{th}_{s,r,j}$ and clocks this submachine once for it.[12]

At each port $\mathsf{in}_s?$ with $s \in \mathcal{H}_S$, $\mathsf{TH}_\mathcal{H}$ expects an input vector $(in_{i,s,r})_{r\in\mathcal{M}_R\cup\{v\}} \in In_S$. It forwards each entry $in_{i,s,r}$ with $r \in \mathcal{M}_R$ to port $\mathsf{in\_s}_{s,r,i}?$ (of $\mathsf{th}_{s,r,i}$), and $in_{i,s,v} = (\mathsf{show}, (s,r,j))$ with $(s,r,j) \in Slot_i$ to port $\mathsf{in\_s}_{s,r,j}?$ (of $\mathsf{th}_{s,r,j}$).[13] At each port $\mathsf{in}_r?$ with $r \in \mathcal{H}_R$, it expects a vector $(in_{i,r,s})_{s\in\mathcal{M}_S} \in In_R$ and forwards each entry $in_{i,r,s}$ to port $\mathsf{in\_r}_{s,r,i}?$ of $\mathsf{th}_{s,r,i}$.

In addition, $\mathsf{TH}_\mathcal{H}$ inputs stop to all machines $\mathsf{th}_{s,r,i-\Delta}$ at port $\mathsf{in\_th}_{s,r,i-\Delta}?$. Then $\mathsf{TH}_\mathcal{H}$ clocks all submachines.

Submachines: The state-transition functions of the machines $\mathsf{th}_{s,r,i}$ are shown in Figures 2 to 4 in standard notation for extended finite-state machines: a circle is a state and an arrow labeled in/out is a transition resulting from an input in and causing an output out. Inputs that are not explicitly shown in a state are ignored. Notation $\mathsf{x}.y$ means that y is input or output at port x, but the subscripts and the appendices "!" and "?" of ports are omitted because they are clear from the context.

The diagrams are complete given the following conventions: Each submachine $\mathsf{th}_{s,r,i}$ obtains its parameters (s, r, i) in its initial state. Any input parameter is assigned to a variable of the same name if the name first occurs in a run; otherwise, the input is only accepted if the value of the parameter and the variable are equal.

The main ideas behind the state-transition functions are the following: Each machine $\mathsf{th}_{s,r,i}$ is started by inputs $(\mathsf{send}, \ldots)$ and/or $(\mathsf{receive}, \ldots)$ from the corresponding correct user(s); if both $s, r \in \mathcal{H}$, the states sr_1, sr_2, and sr_3 occur if there are not two matching inputs. The adversary is immediately told the parameters of the run at the port $\mathsf{out\_a}_{s,r,i}!$. Then the adversary (the partner in Figures 2 and 3 and an outsider in Figure 4) has the opportunity to either disrupt the run with $\mathsf{suppress}$ or to decide that it should end successfully with $\mathsf{receive}$ or (send, m), respectively. In the latter case, he immediately obtains the message if he is not the sender. The outputs for the honest users only occur upon the input stop (from $\mathsf{TH}_\mathcal{H}$ itself), and only then can receipts be shown. Showing by an adversary works within the same round, by an honest user it has one round delay.

Output dispatching in Round i: Machine $\mathsf{TH}_\mathcal{H}$ puts all outputs from submachine ports $\mathsf{out\_s}_{s,r,j}!$ into a set $out_{i,s}$, those at $\mathsf{out\_r}_{s,r,j}!$ into $out_{i,r}$, those at $\mathsf{out\_v}_{s,r,j}!$ into $out_{i,v}$, and those at $\mathsf{out\_a}_{s,r,j}!$ into $out\_a_{i,(s,r,j)}$.[14] Each set is encoded in a deterministic way. It outputs $out_{i,u}$ at $\mathsf{out}_u!$ for each $u \in \mathcal{H}$ and the matrix $(out\_a_{i,(s,r,j)})_{(s,r,j)\in Slots_i}$ at $\mathsf{out\_a}!$. $\diamond$

[11] By "expects" we mean that it replaces any other input by a vector of all ϵ; similarly for other structures in the following. By "for each entry, it does ..." and similar formulations we mean that the entries are treated in lexicographic order of their index tuples in the structure.

[12] Clocking the submachines once for each adversary input and once for honest-user inputs per round helps not to have too many input combinations in the state diagrams. Allowing input sets per slot is not mandatory here (in the state diagrams, never more than one adversary input is considered between two honest-user inputs), but simplifies the simulation, where each of four simulated machines may signal to the trusted host.

[13] The parameter (s, r, j) of $(\mathsf{show}, (s, r, j))$ could be omitted in forwarding; similar for many outputs below because it could be added systematically as the second parameter of all in- and outputs in dispatching. However, for readability we currently let all in- and outputs look the same globally and in submachines.

[14] In state sr_0, two outputs can occur at port $\mathsf{out\_a}_{s,r,j}!$; hence more precisely or in more complicated cases, the dispatching should be seen as set unions.

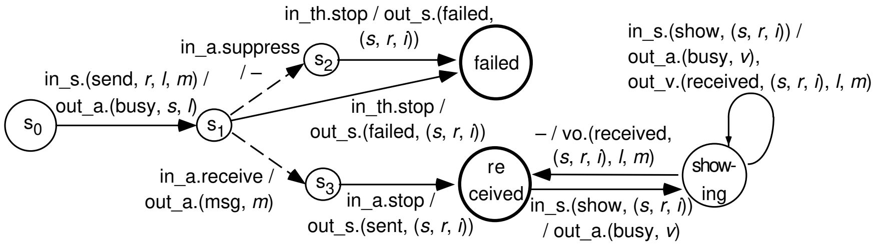

Figure 2: $\mathtt{th}_{s,r,i}$ for $s \in \mathcal{H}_S$ and $r \in \mathcal{A}_R$, i.e., correct sender only. Dashed arrows arise from adversary inputs.

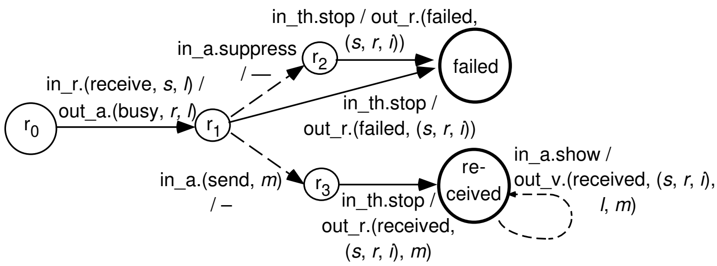

Figure 3: $\mathtt{th}_{s,r,i}$ for $r \in \mathcal{H}_R$ and $s \in \mathcal{A}_S$, i.e., correct recipient only.

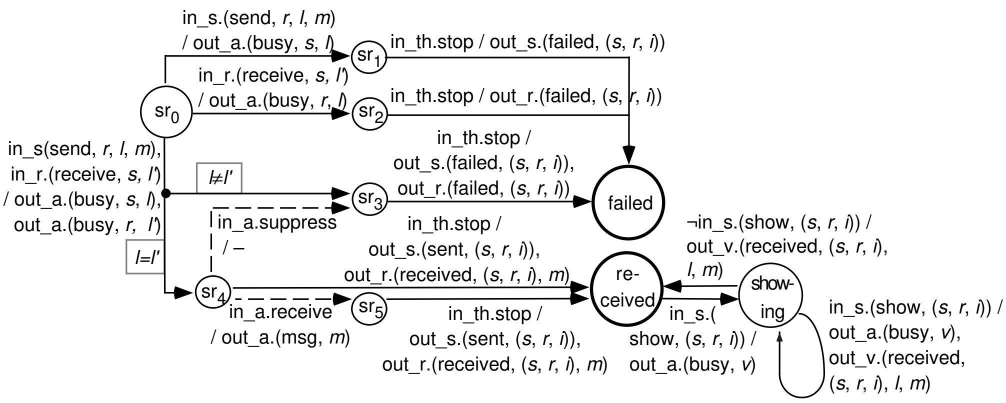

Figure 4: $\mathtt{th}_{s,r,i}$ for $s, r \in \mathcal{H}$, i.e., correct sender and recipient.

5.4 Relation to Requirements-Based Definition

In this section, we show that the ideal system essentially fulfils the integrity requirements from Definition 4.6, and then mention what this means for real systems.

Lemma 5.1 (Integrity of Ideal System) The ideal system from Scheme 5.1 fulfills Definition 4.6b) to h). It also fulfills Requirement a) if submachines $\mathsf{th}_{s,r,i}$ for $s, r \in \mathcal{H}$ ignore inputs $\mathsf{suppress}$. (Intuitively, this means to assume authentic connections between s and r.) □

Proof. Recall that we can assume that only correct inputs are made at the specified ports because of the general preconditions in the requirements. This implies that an input (send, r, l, m) in Round i at $\mathsf{in}_s?$ can only occur as $in_{i,s,r}$ etc. Furthermore, one can easily see from the input dispatching that no input to a machine $\mathsf{th}_{s,r,i}$ is made before Round i.

Case: Sender and recipient correct, i.e., Figure 4 only:

a) Correct Execution, given that $\mathsf{suppress}$ *is ignored:* Let $in_{i,s,r} = (\mathsf{send}, r, l, m)$ and $in_{i,r,s} = (\mathsf{receive}, s, l')$. $\mathsf{TH}_{\mathcal{H}}$ dispatches both to $\mathsf{th}_{s,r,i}$. If $l = l'$, the submachine enters state sr_4. As $\mathsf{suppress}$ is ignored, the input of stop, which $\mathsf{TH}_{\mathcal{H}}$ makes after Δ rounds, leads to outputs $(\mathsf{sent}, (s, r, i))$ and $(\mathsf{received}, (s, r, i), m)$, which are dispatched to the ports $\mathsf{out}_s!$ and $\mathsf{out}_r!$.

If $l \neq l'$, the submachine enters state sr_3 and, on input stop, makes two outputs $(\mathsf{failed}, (s, r, i))$, which are dispatched to $\mathsf{out}_s!$ and $\mathsf{out}_r!$.

b) Unforgeable Messages: An output $(\mathsf{received}, (s, r, i), m) \in out_j$ can only come from $\mathsf{th}_{s,r,i}$. In Figure 4 one easily sees that this requires a local input (send, r, l, m), which, by dispatching, must have been an entry $in_{i',s,r}$.

Case: Only recipient correct, i.e., Figure 3 or 4:

c) Termination for Recipient: If $in_{i,r,s} = (\mathsf{receive}, s, l)$, $\mathsf{TH}_{\mathcal{H}}$ starts $\mathsf{th}_{s,r,i}$, i.e., it leaves state r_0 or sr_0, and then inputs stop to it in Round $i + \Delta$. This leads to exactly one of the required outputs in all states of Figure 3, and in all states reachable without prior stop in Figure 4.

d) Unforgeable Receipts: An output $(\mathsf{received}, (s, r, i), l, m)$ can only come from $\mathsf{th}_{s,r,i}$. One easily sees in Figures 3 and 4 that $\mathsf{th}_{s,r,i}$ does not make this output without a prior input $(\mathsf{receive}, s, l)$, which must have come from $\mathsf{in}_r?$.

e) No Surprises for the Recipient: Both an output $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out}_r!$ and $(\mathsf{received}, (s, r, i), l, m)$ at $\mathsf{out}_v!$ can only come from $\mathsf{th}_{s,r,i}$. However, one easily sees in Figures 3 and 4 that there is no path on which both these outputs occur.

f) Fixed Receipts: Both an output $(\mathsf{received}, (s, r, i), m)$ at $\mathsf{out}_r!$ and $(\mathsf{received}, (s, r, i), l, m')$ at $\mathsf{out}_v!$ can only come from $\mathsf{th}_{s,r,i}$. One easily sees in Figures 3 and 4 that this implies $m' = m$.

Case: Only sender correct, i.e., Figure 2 or 4:

g) Termination for Sender: If $in_{i,s,r} = (\mathsf{send}, r, l, m)$, $\mathsf{TH}_{\mathcal{H}}$ starts $\mathsf{th}_{s,r,i}$, i.e., it leaves state s_0 or sr_0, and then inputs stop to it in Round $i + \Delta$. This leads leads to exactly one of the required outputs in all states reachable without prior stop in Figures 2 and 4.

h) Verifiability of Valid Receipts: An output $(\mathsf{sent}, (s, r, i))$ at $\mathsf{out}_s!$ can only come from $\mathsf{th}_{s,r,i}$. One easily sees in Figures 2 and 4 that $\mathsf{th}_{s,r,i}$ is from then on always in state $\mathsf{received}$ or $\mathsf{showing}$ and will output $(\mathsf{received}, (s, r, i), l, m)$ on input $(\mathsf{show}, (s, r, i))$ within one round. ■

These results together with the integrity-preservation theorem from [PW00] (Theorem 3.1) imply that any real system that is computationally at least as secure as this ideal system also fulfils these requirements computationally.

For secrecy requirements, there is no corresponding general theorem yet, and we do not bother to prove a specific one here. But we sketch a proof that the ideal system fulfils Definition 4.7: An input $in_{i,s,r} = (\mathsf{send}, r, l, m)$ is dispatched to $\mathtt{th}_{s,r,i}$, and an output $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out}_s!$ can only come from the same $\mathtt{th}_{s,r,i}$. One easily sees in Figures 2 and 4 that the same $\mathtt{th}_{s,r,i}$ does not make any output with the parameter m. No other parameter of $\mathtt{th}_{s,r,i}$ depends on m, $\mathsf{TH}_{\mathcal{H}}$ does not use m except in its input to $\mathtt{th}_{s,r,i}$, and $\mathsf{H_s}$ never reuses m. Hence m is perfectly hidden. For relations to formal methods, note that this is a sketch of a typical information-flow proof, see, e.g., [D82].

6 Rigorous Description of the Real System

Now we rigorously define a real system for labeled certified mail. Recall that we already sketched and explained the system in Section 2.

6.1 More Details about the Primitives Used

A signature scheme is a triple of algorithms $(\mathsf{gen_S}, \mathsf{sign}, \mathsf{test})$. We assume w.l.o.g. that the message space is Σ^* [D88] and $\{0,1\} \subseteq \Sigma$. We use slightly abbreviated notation: We write $(\mathsf{sign}_u, \mathsf{test}_u) \leftarrow \mathsf{gen_S}(1^k)$ for the generation of a signing key and a test key based on a security parameter k. By $sig \leftarrow \mathsf{sign}_u(m)$, we denote a signature on the message m, including m itself. More precisely, we assume that sig is a pair (m, s). The scheme may have arbitrary memory. The verification $\mathsf{test}_u(sig)$ returns m if the signature is valid with respect to the included message, else false. Security of a signature scheme means that existential forgery is infeasible even in adaptive chosen-message attacks [GMR 88].

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

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

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

The one-way function owf is simply a function.

Definition 6.2 (Security of One-way Function) A function $\mathsf{owf} : \{0,1\}^* \rightarrow \{0,1\}^*$ is called one-way if for all probabilistic polynomial-time algorithms $\mathsf{A_{owf}}$,

$$P(r^* = r :: r \stackrel{\mathcal{R}}{\leftarrow} \{0,1\}^k; r^* \leftarrow \mathsf{A_{owf}}(1^k, \mathsf{owf}(r)) \in NEGL$$

(as a function of k). The notation means the probability of the event $r^* = r$ in the space defined by the two probabilistic assignments after "::". $\Diamond$

A non-interactive chameleon commitment scheme is a tuple of algorithms $(\mathsf{gen_C}, \mathsf{gen_{CR}}, \mathsf{com}, \mathsf{trans})$. Let its message space be Msg_{C}. (In our case, it must comprise the set Msg of the certified mail scheme.)

- We write $(sk_{\mathsf{C},u}, pk_{\mathsf{C},u}) \leftarrow \mathsf{gen_C}(1^k)$ for the generation of a key pair based on a security parameter k.

- In slightly abbreviated notation, we write $r \leftarrow \mathsf{gen}_{\mathsf{CR},u}$ for the generation of a suitable random value r given $pk_{\mathsf{C},u}$.
- Similarly, we write $c = \mathsf{com}_u(m, r)$ for a commitment on a message $m \in Msg_{\mathsf{C}}$ using a public key $pk_{\mathsf{C},u}$ and a random value r.

  By $(c, r) \leftarrow \mathsf{comr}_u(m)$, we abbreviate the composition $r \leftarrow \mathsf{gen}_{\mathsf{CR},u}; c = \mathsf{com}_u(m, r)$.
- A commitment c is opened by sending (m, r). The recipient compares $\mathsf{com}_u(m, r)$ with c. If they are equal, one says that he accepts m.
- By $r^* \leftarrow \mathsf{trans}_u(c, m, r, m^*)$ we denote the transformation that allows the owner of a secret key $sk_{\mathsf{C},u}$ to take a commitment c, values m, r that open it, and another message $m^* \in Msg_{\mathsf{C}}$ and to derive a value r^* such that c can be opened to m^*. For all correctly generated keys and $c = \mathsf{com}_u(m, r)$, this must give $c = \mathsf{com}_u(m^*, r^*)$.

Definition 6.3 (Security of Commitment Scheme) A non-interactive chameleon commitment scheme is called secure if it has the following three properties.

a) *Computationally binding:* For any probabilistic polynomial-time algorithm A:

$$\begin{aligned} & P(\mathsf{com}_u(m, r) = \mathsf{com}_u(m^*, r^*) \wedge m \neq m^* \\ & \quad :: \quad (sk_{\mathsf{C},u}, pk_{\mathsf{C},u}) \leftarrow \mathsf{gen}_{\mathsf{C}}(1^k); (m, r, m^*, r^*) \leftarrow \mathsf{A}(1^k, pk_{\mathsf{C},u})) \\ & \quad \in \quad NEGL. \end{aligned}$$

b) *Perfectly hiding:* For all $(sk_{\mathsf{C},u}, pk_{\mathsf{C},u}) \in [\mathsf{gen}_{\mathsf{C}}(1^k)]$, all probability distributions Dist on Msg_{C}, all $m \in Msg_{\mathsf{C}}$ and all possible commitments c,

$$P_{\mathsf{Dist}^*}(m|c) = P_{\mathsf{Dist}}(m)$$

where Dist^* is the distribution defined by $m \leftarrow \mathsf{Dist}; (c, r) \leftarrow \mathsf{comr}_u(m)$.[15]

c) *Chameleon:* For all $(sk_{\mathsf{C},u}, pk_{\mathsf{C},u}) \in [\mathsf{gen}_{\mathsf{C}}(1^k)]$ and $m, m^* \in Msg_{\mathsf{C}}$: The probability distribution of the pair (c, r^*) in $(c, r) \leftarrow \mathsf{comr}_u(m)$; $r^* \leftarrow \mathsf{trans}_u(c, m, r, m^*)$ equals that in $(c, r^*) \leftarrow \mathsf{comr}_u(m^*)$.

◇

For example, we can use the commitment scheme from [BCP88, CHP92, P92] with a chameleon extension combined with a family of collision-resistant hash functions [D88]. In the basic scheme, key generation means to randomly choose a k-bit prime q and a $k'(k)$-bit prime p with $q|(p-1)$ (for a function k' determining a suitable second security parameter), a generator g of the unique subgroup G_q of order q in $\mathbb{Z}_p^*$ and $x \xleftarrow{\mathcal{R}} \mathbb{Z}_q^*$, and to set $h = g^x$. The public key is (p, q, g, h) and the secret key x. A random value is then chosen as $r \in \mathbb{Z}_q$ and a commitment on a message $m \in \mathbb{Z}_q$ as $c = \mathsf{com}_u(m, r) = g^m h^r \mod p$. The transformation $r^* = \mathsf{trans}_u(c, m, r, m^*)$ is $r^* = (m - m^*)/x + r$. The scheme is computationally binding under the discrete-logarithm assumption for this family of groups.

We now use a family of collision-resistant hash functions to allow commitments to arbitrarily long inputs. A particular hash function hash_u is also (at least in theory) selected by a public key, which becomes part of the public key of the extended commitment scheme. Now $\mathsf{hash}_u(m)$ is committed to in the place of m. One can immediately see that this combination retains all the properties of the commitment scheme.

For proving simulatability of the certified-mail scheme, we need that an adversary cannot open a commitment made by someone else even if he has chosen the content. This is Part b) of the following lemma, and Part a) is a well-known fact used.

[15] The term $P_{\mathsf{Dist}^*}(m|c)$ is the usual abbreviation of $P(m' = m | c' = c :: m' \leftarrow \mathsf{Dist}; (c', r) \leftarrow \mathsf{comr}_u(m'))$.

Lemma 6.1 (Properties of the Commitments) a) For all $(sk_{\mathsf{C},u}, pk_{\mathsf{C},u}) \in [\mathsf{gen}_{\mathsf{C}}(1^k)]$, all $m, m' \in Msg_{\mathsf{C}}$, and all possible commitments c,

$$P(c' = c :: (c', r) \leftarrow \mathsf{comr}_u(m)) = P(c' = c :: (c', r) \leftarrow \mathsf{comr}_u(m')).$$

b) For any probabilistic polynomial-time algorithms A_1, A_2:

$$\begin{aligned} & P(c = \mathsf{com}_u(m^*, r^*) \\ :: \quad & (sk_{\mathsf{C},u}, pk_{\mathsf{C},u}) \leftarrow \mathsf{gen}_{\mathsf{C}}(1^k); (m, aux) \leftarrow \mathsf{A}_1(1^k, pk_{\mathsf{C},u}); \\ & (c, r) \leftarrow \mathsf{comr}_u(m); (m^*, r^*) \leftarrow \mathsf{A}_2(1^k, pk_{\mathsf{C},u}, m, aux, c)) \\ \in \quad & NEGL. \end{aligned}$$

Here aux denotes arbitrary information that A_1 hands to A_2.

□

Proof. If Part a) were not true, then for $P_{\mathsf{Dist}}(m) = P_{\mathsf{Dist}}(m') = 1/2$, we would obtain a contradiction to the hiding property:

$$\begin{aligned} P_{\mathsf{Dist}^*}(m|c) &= P_{\mathsf{Dist}}(m) P(c' = c :: (c', r) \leftarrow \mathsf{comr}_u(m)) / P_{\mathsf{Dist}^*}(c) \\ &= 1/2 P(c' = c :: (c', r) \leftarrow \mathsf{comr}_u(m)) / P_{\mathsf{Dist}^*}(c) \\ &\neq 1/2 P(c' = c :: (c', r) \leftarrow \mathsf{comr}_u(m')) / P_{\mathsf{Dist}^*}(c) \\ &= P_{\mathsf{Dist}^*}(m'|c). \end{aligned}$$

For Part b), assume that A_1, A_2 contradict the lemma. Then either the probability with an additional condition $m^* \neq m$ or with $m^* = m$ is still not negligible. The first case can immediately be seen to contradict the binding property.

In the second case, consider an adversary A_3 that carries out $(m, aux) \leftarrow \mathsf{A}_1(1^k, pk_{\mathsf{C},u})$, then chooses a message $m' \neq m$ in Msg_{C} (e.g., the first possible one out of two fixed ones), sets $(c, r) \leftarrow \mathsf{comr}_u(m')$ and finally $(m^*, r^*) \leftarrow \mathsf{A}_2(1^k, pk_{\mathsf{C},u}, m, auxc)$. By Part a) this does not change the distribution of c compared with the assumption about A_1 and A_2. Hence the success probability of A_2 is unchanged, and as in the first case we immediately get a contradiction to the binding property. ■

6.2 The Real System

We now describe the real system in detail. A comma in messages denotes tuple composition, not concatenation, i.e., it must be implemented such that decomposition is unambiguous. We augment the message formats slightly compared with Figure 1, in particular by adding message type identifiers like m1 in signed messages (as often tacitly assumed when actually implementing a protocol) and repeating the slot identifier in places to simplify dispatching.[16]

Scheme 6.1 (Labeled Certified Mail) For any parameters $par \in Par_{CM}$ with $\Delta = 6$, we define a real system $Sys_{par}^{\mathsf{CM,real}}$ for labeled certified mail. It is (almost) a standard cryptographic system according to Definitions 3.1 and 3.2 in [PSW00b], i.e., it can be derived in a standard way from an intended structure (M^*, S^*) and a trust model (consisting of an access structure and a channel model).[17] However, for independence and for readability of the proof, we also describe the resulting actual structures.

[16]In this protocol we could also do without because each type of machine signs only one type of message.

[17]The difference is that a few user ports are not needed, e.g., $\mathsf{in}_v?$. A larger difference to the rest of that section in [PSW00b] is that the mappings between real and ideal system are not canonical because we need the trusted third party for the optimistic protocols.

Let $\mathcal{M}' = \mathcal{M} \cup \{t, b\}$ with $t = n+1, b = n+2$. Then M^* is a set $\{\mathsf{M}_u \mid u \in \mathcal{M}'\}$, i.e., one machine for each user known from the ideal system, a machine M_t for a third party, and a broadcast machine M_b. The access structure is

$$\mathcal{ACC}'^{\mathsf{CM}}_{par} = \{\mathcal{H}' = \mathcal{H} \cup \{t, b\} | \mathcal{H} \in \mathcal{ACC}^{\mathsf{CM}}_{par}\},$$

i.e., the third-party machine and the broadcast machine are (like the verifier's) always correct. The actual system is

$$Sys^{\mathsf{CM,real}}_{par} = \{(M_{\mathcal{H}}, S_{\mathcal{H}}) | \mathcal{H} \in \mathcal{ACC}^{\mathsf{CM}}_{par}\}$$

where $S_{\mathcal{H}}$ is as in Definition 4.4, while $M_{\mathcal{H}} = \{\mathsf{M}_{u,\mathcal{H}} | u \in \mathcal{H} \cup \{t, b\}\}$ and the machines $\mathsf{M}_{u,\mathcal{H}}$ are now defined. They only slightly differ from the intended machines M_u in the ports because of the channel model. The machines M_u are of five types depending on whether $u \in \mathcal{M}_S$, $\mathcal{M}_R$, or $u = v$, t or b.

Ports and channel model: The ports of a machine M_u with $u \in \mathcal{M} \cup \{t\}$ are $\{\mathsf{in}_u?, \mathsf{out}_u!, \mathsf{broad}_u!\} \cup \{\mathsf{broad}_{w,u}?, \mathsf{net}_{u,w}!, \mathsf{net}_{w,u}? | w \in \mathcal{M} \cup \{t\}\}$, except that M_v does not have (need) the port $\mathsf{in}_v?$ and M_t neither $\mathsf{in}_t?$ nor $\mathsf{out}_t!$. The first two of the six port types are for the user, the second two for broadcast (of public keys) and the last two for communication to and from M_w.

The broadcast machine M_b has the ports $\{\mathsf{broad}_u? | u \in \mathcal{M} \cup \{t\}\} \cup \{\mathsf{broad}_{u,w}! \mid u, w \in \mathcal{M} \cup \{t\}\}$.

The channel model is that all channels involving M_v or M_t are authentic (but not private), the broadcast channel is authentic (and consistent, i.e., even the adversary cannot send different messages to different correct recipients—this is why we need M_b), and the other channels are insecure. This means that each port $\mathsf{net}_{u,w}!$ with $u \in \{v, t\}$ or $w \in \{v, t\}$ gets a duplicate $\mathsf{net}^a_{u,w}?$ in $\mathsf{M}_{u,\mathcal{H}}$ where the machine makes the same outputs. (The adversary will connect to this port.) Similarly $\mathsf{M}_{b,\mathcal{H}}$ gets an additional port $\mathsf{broad}^a_u!$ for each u where it makes the same outputs as at its ports $\mathsf{broad}_{u,w,\cdot}!$ Each port $\mathsf{net}_{w,u}?$ with $u, w \notin \{v, t\}$ is replaced by $\mathsf{net}^a_{w,u,\cdot}?$ (This allows the adversary to connect to both this port and $\mathsf{net}_{w,u}!$, and thus to control the connection.)

We tacitly omit the duplicated ports in the following, while we keep the notation $\mathsf{net}^a_{w,u}?$ for the replaced ports to keep in mind that the inputs there are not necessarily the outputs of another correct machine at $\mathsf{net}_{w,u}!$.

Overall structure of each machine: All machines $\mathsf{M}_{u,\mathcal{H}}$ with $u \neq b$ are defined by submachines per slot. (An overview is given in Figure 12 in the proof.) This is just a convenient way to write the state-transition function of $\mathsf{M}_{u,\mathcal{H}}$; they are not independent machines as in the system model, in particular they can use the keys, parameters par, and current round number as stored in $\mathsf{M}_{u,\mathcal{H}}$.[18]

$\mathsf{M}_{s,\mathcal{H}}$ **for** $s \in \mathcal{H}_S$**.** Let $\mathit{Slots}_S = \mathcal{M}_R \times \mathbb{N}_{>1}$ and $\mathit{Slots}_{S,i} = \mathcal{M}_R \times \{2, \ldots, i\}$ for all $i \in \mathbb{N}$. In $\mathsf{M}_{s,\mathcal{H}}$, each slot $(r, i) \in \mathit{Slots}_S$ is handled by a submachine $\mathsf{cm\_s}_{s,r,i}$. For simplicity, we assume that in any round i, submachines $\mathsf{cm\_s}_{s,r,j}$ for all $(r, j) \in \mathit{Slots}_{S,i}$ exist, although most of them will remain in their starting state forever. Each $\mathsf{cm\_s}_{s,r,i}$ has ports $\mathsf{in\_s}_{s,r,i}?$, $\mathsf{out\_s}_{s,r,i}!$, $\mathsf{r2s}^a_{s,r,i}?$, $\mathsf{s2r}_{s,r,i}!$, $\mathsf{t2s}_{s,r,i}?$, $\mathsf{s2t}_{s,r,i}!$, and $\mathsf{s2v}_{s,r,i}!$ for in- and outputs to its user and messages from and to the recipient, the third party and the verifier.

$\mathsf{M}_{r,\mathcal{H}}$ **for** $r \in \mathcal{H}_R$**.** In exactly the same way, in $\mathsf{M}_{r,\mathcal{H}}$ each slot $(s, i) \in \mathit{Slots}_R = \mathcal{M}_S \times \mathbb{N}_{>1}$ is handled by a submachine $\mathsf{cm\_r}_{s,r,i}$ with ports $\mathsf{in\_r}_{s,r,i}?$, $\mathsf{out\_r}_{s,r,i}!$, $\mathsf{s2r}^a_{s,r,i}?$, $\mathsf{r2s}_{s,r,i}!$, and $\mathsf{t2r}_{s,r,i}?$.

$\mathsf{M}_{v,\mathcal{H}}$ **and** $\mathsf{M}_{t,\mathcal{H}}$**.** In $\mathsf{M}_{v,\mathcal{H}}$, each slot $(s, r, i) \in \mathit{Slots}$ is handled by a submachine $\mathsf{cm\_v}_{s,r,i}$ with ports $\mathsf{s2v}_{s,r,i}?$ and $\mathsf{out\_v}_{s,r,i}!$, and in $\mathsf{M}_{t,\mathcal{H}}$, by a submachine $\mathsf{cm\_t}_{s,r,i}$ with ports $\mathsf{s2t}_{s,r,i}?$, $\mathsf{t2s}_{s,r,i}!$, and $\mathsf{t2r}_{s,r,i}!$.

[18] A real implementation should not reserve resources for submachines as long as they remain in their starting state. Then the submachines of $\mathsf{M}_{v,\mathcal{H}}$ and $\mathsf{M}_{t,\mathcal{H}}$ and all those of unused slots disappear. We uniformly used the submachine notation to unify dispatching and thus to simplify the proof.

Initialization (Rounds 0, 1, and part of 2): In Round 0, each machine $\mathsf{M}_{u,\mathcal{H}}$ with $u \neq v, b$ generates signature keys $(\mathsf{sign}_u, \mathsf{test}_u) \leftarrow \mathsf{gen}_\mathsf{S}(1^k)$. Additionally, $\mathsf{M}_{t,\mathcal{H}}$ generates a key pair $(sk_{\mathsf{C},t}, pk_{\mathsf{C},t}) \leftarrow \mathsf{gen}_\mathsf{C}(1^k)$ of the commitment scheme. Each $\mathsf{M}_{u,\mathcal{H}}$ with $u \in \mathcal{M}_S \cup \mathcal{M}_R$ outputs test_u at $\mathsf{broad}_u!$, and $\mathsf{M}_{t,\mathcal{H}}$ the pair $(pk_{\mathsf{C},t}, \mathsf{test}_t)$ at $\mathsf{broad}_t!$.

In Round 1, all machines do nothing, except that the broadcast machine $\mathsf{M}_{b,\mathcal{H}}$ forwards the input at port $\mathsf{broad}_u?$ to the ports $\mathsf{broad}_{u,w}!$ for all u, w. In all other rounds, $\mathsf{M}_{b,\mathcal{H}}$ does nothing.

In Round 2, each machine $\mathsf{M}_{u,\mathcal{H}}$ with $u \neq b$ considers its broadcast inputs. The first input at $\mathsf{broad}_{t,u}?$ is stored as a commitment key $pk_{\mathsf{C},t}$, the second one as test_t, and the input at $\mathsf{broad}_{w,u}?$ for each $w \in \mathcal{M}_S \cup \mathcal{M}_R$ as test_w. As the broadcast machine guarantees consistency, we do not distinguish the key versions held by different machines.

Now we consider the state-transition functions of machines $\mathsf{M}_{u,\mathcal{H}}$ with $u \neq b$ for Rounds $i \geq 2$. Inputs at the broadcast ports are never considered again after initialization, hence we no longer mention them.

Global inputs and dispatching in Round i:

$\mathsf{M}_{s,\mathcal{H}}$ **for** $s \in \mathcal{H}_S$**.** At its port $\mathsf{in}_s?$, it expects an input vector $in_{i,s} \in In_S$. It forwards each entry $in_{i,s,r}$ with $r \in \mathcal{M}_R$ to port $\mathsf{in\_s}_{s,r,i}?$ (of $\mathsf{cm\_s}_{s,r,i}$), and each entry $in_{i,s,v} = (\mathsf{show}, (s,r,j))$ to $\mathsf{in\_s}_{s,r,j}?$ (of $\mathsf{cm\_s}_{s,r,j}$).

At each network input port $\mathsf{net}^a_{r,s}?$ with $r \in \mathcal{M}_R$, it expects a tuple of at most i entries $((s,r,j), msg)$ with $j \leq i$ and at port $\mathsf{net}_{t,s}?$ a tuple of at most $n_R \cdot i$ entries of this form.[19] It forwards each entry to ports $\mathsf{r2s}^a_{s,r,j}?$ or $\mathsf{t2s}_{s,r,j}?$, respectively, of $\mathsf{cm\_s}_{s,r,j}$.

$\mathsf{M}_{r,\mathcal{H}}$ **for** $r \in \mathcal{H}_R$**.** At its port $\mathsf{in}_r?$, it expects an input vector $in_{i,r} \in In_R$ and forwards each entry $in_{i,r,s}$ to port $\mathsf{in\_r}_{s,r,i}?$ of $\mathsf{cm\_r}_{s,r,i}$.

At each network input port $\mathsf{net}^a_{s,r}?$ with $s \in \mathcal{M}_S$, it expects a tuple of at most i entries of the form $((s,r,j), msg)$ with $j \leq i$, and at port $\mathsf{net}_{t,r}?$ a tuple of a most $n_S \cdot i$ entries of the same form. It forwards each entry to $\mathsf{cm\_r}_{s,r,j}$ at port $\mathsf{s2r}^a_{s,r,j}?$ or $\mathsf{t2r}_{s,r,j}?$, respectively.

$\mathsf{M}_{v,\mathcal{H}}$**.** At each port $\mathsf{net}_{s,v}?$ with $s \in \mathcal{M}_S$, it expects at most one message $((s,r,j), msg)$, which it forwards to port $\mathsf{s2v}_{s,r,j}?$ of $\mathsf{cm\_v}_{s,r,j}$.

$\mathsf{M}_{t,\mathcal{H}}$**.** At each port $\mathsf{net}_{s,t}?$ with $s \in \mathcal{M}_S$, it expects at most one message $((s,r,j), msg)$, which it forwards to port $\mathsf{s2t}_{s,r,j}?$ of $\mathsf{cm\_t}_{s,r,j}$.

Submachines: Recall that a run of the subprotocol send was shown in Figure 1. The detailed behavior of the submachines is illustrated in Figures 5 to 7 in the same notation as for the ideal system, except that these figures are only complete with respect to the states and accepted input classes, while the local variables and computations are defined in detail in the following. State transitions without input in the figure occur at the clock signal, i.e., the submachine counts the number of rounds it waits for some message. Each submachine $\mathsf{cm\_x}_{s,r,i}$ obtains the values (s,r,i) in its initial state, and the fixed index s,r,i of its ports is omitted in the descriptions.

$\mathsf{cm\_s}_{s,r,i}$ **for** $s \in \mathcal{H}_S$**.** On input (send, r, l, m) at port $\mathsf{in\_s}?$, it computes $(c, r_S) \leftarrow \mathsf{comr}_t(m)$ and outputs $m_1 \leftarrow \mathsf{sign}_s((s,r,i), \mathsf{m1}, l, c)$ at $\mathsf{s2r}!$.

Then it waits for an input m_2 at port $\mathsf{r2s}^a?$ in Round $i+2$. It tests whether $\mathsf{test}_r(m_2) = ((s,r,i), \mathsf{m2}, m_1, p_R)$ for some value p_R. (The other occurring variables are already locally defined in $\mathsf{cm\_s}_{s,r,i}$, and a test of ϵ, i.e., no input, automatically fails.) If not, it waits until Round $i+6$, outputs $(\mathsf{failed}, (s,r,i))$ at $\mathsf{out\_s}!$, and enters the final state failed.

[19] In slight abuse of notation, we also consider entries correct where (s,r,j) is the first entry in a nested tuple, i.e., the left-most leaf in the tree. Instead, we could duplicate slot names in some messages below.

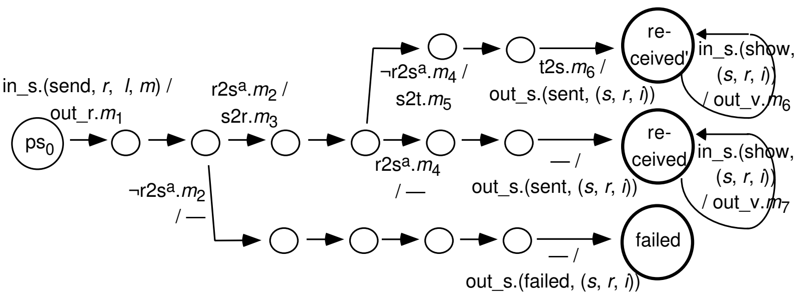

Figure 5: Sender submachine $\mathsf{cm\_s}_{s,r,i}$.

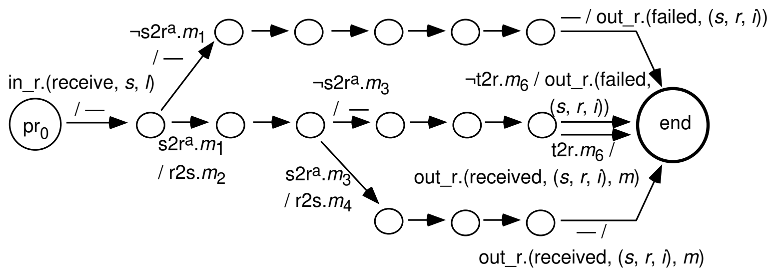

Figure 6: Recipient submachine $\mathsf{cm\_r}_{s,r,i}$.

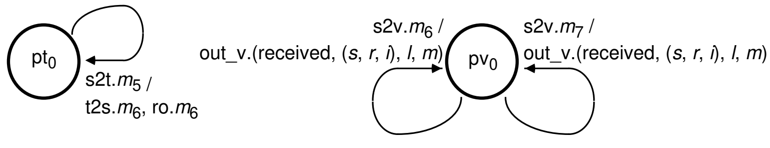

Figure 7: Third-party submachine $\mathsf{cm\_t}_{s,r,i}$ and verifier submachine $\mathsf{cm\_v}_{s,r,i}$.

If it received a correct m_2, it outputs $m_3 = ((s,r,i), m, r_S)$ at $\mathsf{s2r}!$. Then, if it obtains a message $m_4 = ((s,r,i), r_R)$ at $\mathsf{r2s}^a?$ in Round $i+4$ with a value r_R for which $\mathsf{owf}(r_R) = p_R$, it outputs $(\mathsf{sent}, (s,r,i))$ at $\mathsf{out\_s}!$ in Round $i+6$ and enters the state $\mathsf{received}$. Otherwise, it outputs $m_5 = (m_1, m_2, m_3)$ at $\mathsf{s2t}!$. If it obtains any message m_6 at $\mathsf{t2s}?$ in Round $i+6$, it outputs $(\mathsf{sent}, (s,r,i))$ at $\mathsf{out\_s}!$ and enters the state $\mathsf{received}'$.[20]

On input $(\mathsf{show}, (s,r,i))$ at port $\mathsf{in\_s}?$, it outputs $m_7 = ((s,r,i), \mathsf{m7}, m_1, m_2, m_3, m_4)$ at $\mathsf{s2v}!$ if it is in state $\mathsf{received}$, and m_6 if it is in state $\mathsf{received}'$.

$\mathsf{cm\_r}_{s,r,i}$ **for** $r \in \mathcal{H}_R$**.** On input $(\mathsf{receive}, s, l)$ at $\mathsf{in\_r}?$, it waits for an input m_1 at $\mathsf{s2r}^a?$ in Round $i+1$. It then tests whether $\mathsf{test}_s(m_1) = ((s,r,i), \mathsf{m1}, l, c)$ for some value c. If not, it waits until Round $i+6$, outputs $(\mathsf{failed}, (s,r,i))$ at $\mathsf{out\_r}!$, and enters the state end.

If it has received a correct m_1, it selects $r_R \xleftarrow{\mathcal{R}} \{0,1\}^k$ and outputs $m_2 \leftarrow \mathsf{sign}_r((s,r,i), \mathsf{m2}, m_1, \mathsf{owf}(r_R))$ at $\mathsf{r2s}!$.

Then it waits for an input m_3 at $\mathsf{s2r}^a?$ in Round $i+3$. If $m_3 = ((s,r,i), m, r_S)$ for some values $m \in \mathit{Msg}$ and r_S, it tests if $\mathsf{com}_t(m, r_S) = c$. If yes, it outputs $m_4 = ((s,r,i), r_R)$ at $\mathsf{r2s}!$. In Round $i+6$ it then outputs $(\mathsf{received}, (s,r,i), m)$ at $\mathsf{out\_r}!$ and enters the state end.

If it does not receive a correct m_3, it waits until Round $i+6$. If it then obtains an input m_6 at $\mathsf{t2r}?$, it outputs $(\mathsf{received}, (s,r,i), m)$ at $\mathsf{out\_r}!$, where it finds m by retrieving the signed message $((s,r,i), \mathsf{m6}, m_5)$, decomposing m_5 into (m_1, m_2, m_3), and m_3 into $((s,r,i), m, r_S)$. Otherwise it outputs $(\mathsf{failed}, (s,r,i))$, and in both cases it enters the state end.

$\mathsf{cm\_t}_{s,r,i}$**.** It first tests that the current round number is $j = i+5$. If yes, then for each input $\mathit{complaint}_{j,s}$, it tests that it is a correct message m_5. More precisely, it first tests if it is some triple (m_1, m_2, m_3). If yes, it verifies (with its own values i, s) that $\mathsf{test}_s(m_1) = ((s,r,i), \mathsf{m1}, l, c)$ for some values $r \in \mathcal{M}_R$, $l \in L$, and c, and that $\mathsf{test}_r(m_2) = ((s,r,i), \mathsf{m2}, m_1, p_R)$ for some value p_R. Finally, it tests that $m_3 = ((s,r,i), m, r_S)$ for some values $m \in \mathit{Msg}$ and r_S and $\mathsf{com}_t(m, r_S) = c$. If this is true, it outputs $m_6 \leftarrow \mathsf{sign}_t((s,r,i), \mathsf{m6}, m_5)$ at $\mathsf{t2s}!$ and $\mathsf{t2r}!$.

$\mathsf{cm\_v}_{s,r,i}$**.** It first tests that $j \geq i+7$ for the current round number j. For each input $\mathit{receipt}_{j,s}$, it first verifies that it is of the form $(((s,r,i), \mathsf{m6}, m_5), \mathit{sig})$ or $((s,r,i), \mathsf{m7}, m_1, m_2, m_3, m_4)$ for some values m_i and sig. In the first case, it then tests if $\mathsf{test}_t(\mathit{receipt}_{j,s}) = ((s,r,i), \mathsf{m6}, m_5)$ and performs the test of m_5 as defined for $\mathsf{cm\_t}_{s,r,i}$.[21] In the second case, it performs the same tests on $m_5 = (m_1, m_2, m_3)$, and then tests m_4 as defined for $\mathsf{cm\_s}_{s,r,i}$. If all is correct, it outputs $(\mathsf{received}, (s,r,i), l, m)$ at $\mathsf{out\_v}!$.

Output Dispatching in Round i:

$\mathsf{M}_{s,\mathcal{H}}$ **for** $s \in \mathcal{H}_S$**.** It puts all submachine outputs at ports $\mathsf{out\_s}_{s,r,j}!$ into a set $\mathit{out}_{i,s}$, those at $\mathsf{s2r}_{s,r,j}!$ into $\mathit{net}_{i,s,r}$, those at $\mathsf{s2v}_{s,r,j}!$ into $\mathit{net}_{i,s,v}$, and those at $\mathsf{s2r}_{s,r,j}!$ into $\mathit{net}_{i,s,t}$. It outputs these sets at the ports with the corresponding names in some deterministic encoding.

$\mathsf{M}_{r,\mathcal{H}}$ **for** $r \in \mathcal{H}_R$**.** It puts outputs at $\mathsf{out\_r}_{s,r,j}!$ into a set $\mathit{out}_{i,r}$ and those at $\mathsf{r2s}_{s,r,j}!$ into $\mathit{net}_{i,r,s}$ and outputs these sets at the ports $\mathsf{out}_r!$ and $\mathsf{net}_{r,s}!$ for all $s \in \mathcal{M}_S$, respectively.

$\mathsf{M}_{t,\mathcal{H}}$ **and** $\mathsf{M}_{v,\mathcal{H}}$**.** $\mathsf{M}_{t,\mathcal{H}}$ puts all outputs at $\mathsf{t2s}_{s,r,j}!$ into a set $\mathit{net}_{t,s}!$ and those at $\mathsf{t2r}_{s,r,j}!$ into a set $\mathit{net}_{t,r}!$. $\mathsf{M}_{v,\mathcal{H}}$ puts all outputs at $\mathsf{out\_v}_{s,r,j}!$ into a set $\mathit{out}_{i,v}$. They output these sets accordingly.

$\diamond$

[20] Given the program of $\mathsf{M}_{t,\mathcal{H}}$, it will always receive m_6 and m_6 will be correct, because $\mathsf{M}_{t,\mathcal{H}}$ is always correct and the connection to $\mathsf{M}_{t,\mathcal{H}}$ is authentic. In real life, adding a verification here may be wise; similar for $\mathsf{cm\_r}_{s,r,i}$ below.

[21] As $\mathsf{M}_{t,\mathcal{H}}$ is assumed to be correct, this could be omitted except for extracting parameters.

7 Security Proof of the Real System

We now prove that the real system in Scheme 6.1 is computationally at least as secure as the ideal system in Definition 5.1. Our simulation is blackbox (see Definition 5.4).

The main cryptographic aspect is in the simulation of a message m_1: In Round i, where m_1 should be sent, the trusted host only outputs a busy-signal without revealing the actual message. Nevertheless, the simulator has to give A a correct-looking network message, which includes a commitment that is supposed to fix the message m the honest user wants to send. If the protocol run is successful, the simulator has to open this commitment two rounds later. If it then reveals a message $m' \neq m$, the simulation is not correct, e.g., an honest recipient may not get the message an honest sender sent. Here is why we need the chameleon property: It allows the simulator (who also only simulates the machine M_t and thus knows its key) to first make the commitment on an arbitrary message m_{sim}, and later open it to the correct message m.

The main non-cryptographic aspects of the simulation are to verify that the real adversary has no possibilities to disrupt protocol runs in certain states, to show receipts too early, etc., that are not provided in the ideal system.

We structure the simulator as much as possible like the real system, i.e., it simulates the machines M_u and submachines $\mathsf{cm\_s}_{s,r,i}$ and $\mathsf{cm\_r}_{s,r,i}$, see Figure 8. One main difference is that the simulator does not interact directly with the users, but has to communicate with the trusted host for this. The second one is timing. As the simulator is an adversary, it is clocked in Subrounds $[i.2]$ and $[i.4]$, while the correct machines are clocked in Subrounds $[i.1]$. To ease the comparison of the real system, and the simulator together with the trusted host, in the proof below, we use a 6-round clocking scheme from which both 4-round schemes can easily be derived.

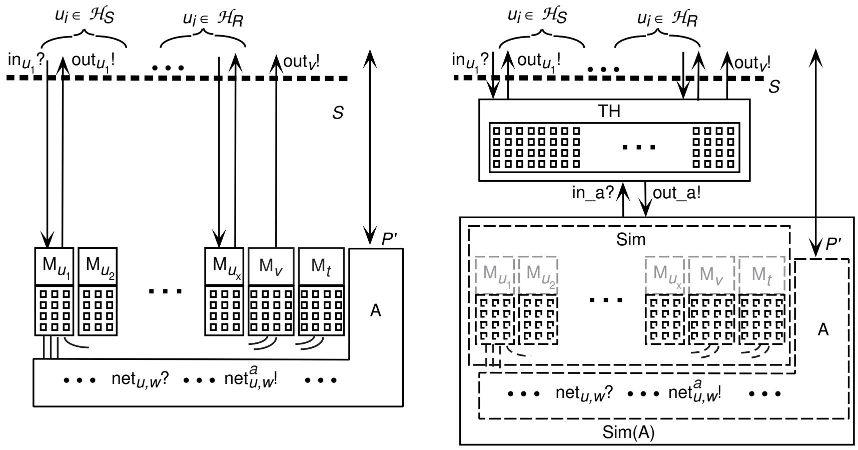

Figure 8: Structure of the real system and the simulation. The tiny squares are submachines. All indices $\mathcal{H}$ are omitted. The ports of A for listening to authentic channels are not shown. Dashed machines are only simulated, dashed gray submachines are not distinguished as such within Sim.

We use Corollary 5.4 of [PSW00b]; it states that we only need to consider users H that use precisely the specified ports of the structures. Hence the set of the ports of the real adversary can be written as $P = \bar{S}^c_{\mathsf{real},\mathcal{H}} \cup P'$, where $\bar{S}_{\mathsf{real},\mathcal{H}}$ are the unspecified free ports of the real system and the ports in P' will be connected to H. Furthermore, by Remark 5.4 in [PSW00b], we can assume that H is not clocked in Subrounds $[i.1]$.

Scheme 7.1 (Simulator for Labeled Certified Mail) Let parameters $par \in Par_{CM}$ with $\Delta = 6$, a set $\mathcal{H} \in \mathcal{ACC}^{\mathsf{CM}}_{par}$ and a set $P = \bar{S}_{\mathsf{real},\mathcal{H}} \cup P'$ of adversary ports be given. We define a simulator $\mathsf{Sim}_{\mathcal{H},P}(\mathsf{A})$ using an arbitrary adversary with the ports P as a blackbox submachine.

In the following, we omit all indices $\mathcal{H}$ and also the index P of $\mathsf{Sim}(\mathsf{A})$. As $\mathsf{Sim}(\mathsf{A})$ leaves the communication between A and H unchanged, we can, in slight abuse of notation, say Sim for the part of $\mathsf{Sim}(\mathsf{A})$ without A.

Ports: The ports of Sim are necessarily $\bar{S}^c_{\mathsf{real}} \cup \{\mathsf{in\_a!}, \mathsf{out\_a?}\}$.

Overall structure and timing: Internally, Sim simulates each submachine $\mathsf{cm\_x}_{s,r,i}$ by a machine $\mathsf{cm\_x}'_{s,r,i}$ with the same ports, except that $\mathsf{in\_x}_{s,r,i}?$ and $\mathsf{out\_x}_{s,r,i}!$ are replaced by $\mathsf{in\_x}'_{s,r,i}?$ and $\mathsf{out\_x}'_{s,r,i}!$, and that $\mathsf{cm\_t}$ has an additional port $\mathsf{out\_t}'_{s,r,i}!$. The dispatching is done by Sim itself. We define the transitions of $\mathsf{Sim}(\mathsf{A})$ as the compressed version of a 6-subround clocking scheme $\kappa_6 = (\mathsf{TH}, \mathsf{Sim}, \mathsf{A}, \mathsf{H}, \mathsf{A}, \mathsf{Sim})$ where Subrounds 2 with 3 and 5 with 6 are joined to get the correct clocking for the ideal system. We call the joined subrounds 2a and 2b, and 4a and 4b.[22] Essentially, in Subround $[i.2a]$, Sim transforms the abstract messages it got from TH into suitable network messages for A, and in Subround $[i.4b]$ it transforms the messages from A into corresponding signals to TH.

Initialization (Rounds 0, 1, and part of 2): The simulated key exchange is identical to that in the real machines M_u. The secret commitment key $sk_{\mathsf{C},t}$ is considered a global variable in Sim, available to all submachines. More precisely, Sim generates keys in Subround $[0.2a]$, switches a simulated broadcast machine in Subround $[1.2a]$, and reads in broadcast inputs in Subround $[1.4b]$.

Now we consider the state-transition function of Sim for Rounds $i \geq 2$, more precisely Subrounds $[i-1.4b]$ and $[i.2a]$, not mentioning the broadcast inputs again.

Global inputs and dispatching in round i: In Subround $[i-1.4b]$, Sim dispatches the network inputs (i.e., those at ports with net in their name) just like the corresponding M_u would. (In particular, we still call the simulated inputs to M_v and M_t in this subround $receipt_{i,s}$ and $complaint_{i,s}$.)

In Subround $[i.2a]$, Sim dispatches the entries of the input matrix out_a_i at port $\mathsf{out\_a?}$ as shown in Figure 9, where m_{sim} is a fixed message from Msg. By definition of TH, only the inputs in the table are possible.

$out\_a_{i,(s,r,j)}$	Input	to port	of
(busy, s, l)	$(\mathsf{send}, r, l, m_{sim})$	$\mathsf{in\_s}'_{s,r,j}?$	$\mathsf{cm\_s}'_{s,r,j}$
(busy, r, l)	$(\mathsf{receive}, s, l)$	$\mathsf{in\_r}'_{s,r,j}?$	$\mathsf{cm\_r}'_{s,r,j}$
(busy, v)	$(\mathsf{show}, (s, r, j))$	$\mathsf{in\_s}'_{s,r,j}?$	$\mathsf{cm\_s}'_{s,r,j}$
(msg, m)	(msg, m)	$\mathsf{in\_s}'_{s,r,j}?$	$\mathsf{cm\_s}'_{s,r,j}$

Figure 9: Dispatching of inputs from TH by Sim. No other inputs can occur.

Sender submachine $\mathsf{cm\_s}'_{s,r,i}$: It acts like $\mathsf{cm\_s}_{s,r,i}$ except for the following changes. Compared with Figure 5, we only consider the states where more than the clock signal is considered.[23] We treat them essentially in the order of the rounds where they occur. A subprotocol run with all the simulated machines is shown in Figures 10 and 11.

[22] This is defined in Lemma 4.1 in [PSW00b]. Essentially, one can first combine arbitrary machines into one (here Sim and A). Then, if only this machine is clocked in several successive subrounds, one can join the subrounds into one and let the machine internally execute the state transitions in the right order.

[23] The other states would be split in two, as Sim is clocked twice per round.

Round	Changes
$[i]$	In Subround $[i.2a]$, $\mathsf{cm\_s}'_{s,r,i}$ computes $m_{1,sim}$ like m_1 with the input message m_{sim} and $(c, r_{sim}) \leftarrow \mathsf{comr}_t(m_{sim})$. It also sets aside a random string r_{trans} of sufficient length for a later call of trans for c with any other message m. (Choosing it so early is helpful in the proof below.)
$[i+2]$	In Subround $[i + 1.4b]$, on input of a correct m_2 (relative to $m_{1,sim}$) at $\mathsf{s2r}^a?$, it outputs $\mathsf{receive}$ at $\mathsf{out\_s}'!$, otherwise $\mathsf{suppress}$. (This signals to the trusted host to reveal the message m or that the run fails.) If it has output $\mathsf{receive}$, then in Subround $[i + 2.2a]$ it expects an input (msg, m) at $\mathsf{in\_s}'?$. If this comes, it transforms the commitment with $r_S \leftarrow \mathsf{trans}_t(c, m_{sim}, r_{sim}, m)$, using r_{trans} if random bits are needed in this process, and outputs $m_3 = ((s, r, i), m, r_S)$ at $\mathsf{s2r}!$. If it does not come, $\mathsf{cm\_s}'_{s,r,i}$ stops prematurely.
$[i+4]$	The transition is made in Subround $[i + 4.2a]$ without changes. (It could also be in $[i + 3.4b]$.)
$[i+6]$	The state transition to a final state is made in Subround $[i + 5.4b]$ without changes, but no output is made.
$[j]$	(With $j > i + 6$.) Reactions on show are made unchanged in Subround $[j.2a]$.

Recipient submachine $\mathsf{cm\_r}'_{s,r,i}$*:* It acts like $\mathsf{cm\_r}_{s,r,i}$ except for the following changes:

$[i+1]$	In Subround $[i.4b]$, if no correct m_1 is input, it additionally outputs $\mathsf{suppress}$ at $\mathsf{out\_r}'!$.
$[i+3]$	In Subround $[i + 2.4b]$, if a correct m_3 is input, it additionally outputs (send, m) at $\mathsf{out\_r}'!$.
$[i+6]$	It does not make an output.

Third-party submachine $\mathsf{cm\_t}'_{s,r,i}$*:* It acts like $\mathsf{cm\_t}_{s,r,i}$ with the following change:

$[j]$ In Subround $[j-1.4b]$, it verifies that $j = i+5$. For each correct input $complaint_{j,s}$, it additionally outputs (send, m) at $\mathsf{out\_t}'!$ for the value m it obtained during the verification.

Verifier submachine $\mathsf{cm\_v}'_{s,r,i}$*:* It acts like $\mathsf{cm\_v}_{s,r,i}$ with the following change:

$[j]$ In Subround $[j - 1.4b]$, it verifies that $j \geq i + 7$. Then, if $\mathsf{cm\_v}_{s,r,i}$ would output $(\mathsf{received}, (s, r, i), l, m)$, it outputs show instead.

Output dispatching in Round i: Sim takes the outputs of all submachine ports $\mathsf{out\_s}'_{s,r,j}!$, $\mathsf{out\_r}'_{s,r,j}!$, $\mathsf{out\_t}'_{s,r,j}!$, and $\mathsf{out\_v}'_{s,r,j}!$, and puts them into the set $in\_a_{i,(s,r,j)}$ of the global output matrix in_a_i.

All other outputs of submachines (the network messages) are dispatched by Sim just as by the machines M_u. ◇

Theorem 7.1 Let parameters $par \in Par_{CM}$ be given with $\Delta = 6$. Let $f : Sys^{\mathsf{CM,real}}_{par} \to Sys^{\mathsf{CM,id}}_{par}$ be the function with $f(M_\mathcal{H}, S_\mathcal{H}) = (\{\mathsf{TH}_\mathcal{H}\}, S_\mathcal{H})$ for all $\mathcal{H} \in \mathcal{ACC}^{\mathsf{CM}}_{par}$. Then

$$Sys^{\mathsf{CM,real}}_{par} \geq^{f,\mathsf{poly}}_{\mathsf{sec}} Sys^{\mathsf{CM,id}}_{par},$$

and this holds even in the blackbox sense. □

Proof. Let a set $\mathcal{H}$ and thus a structure $(M_\mathcal{H}, S_\mathcal{H})$ and a port set $P = \bar{S}^c_{\mathsf{real},\mathcal{H}} \cup P'$ be given. In the following, we write $\mathsf{Sim(A)}$ for the simulator $\mathsf{Sim}_{\mathcal{H},P}(\mathsf{A})$ from Scheme 7.1, and also omit all other indices $\mathcal{H}$. Now let a configuration $conf_1 = (M, S, \mathsf{H}, \mathsf{A}) \in \mathsf{Conf}^f(Sys^{\mathsf{CM,real}}_{par})$ with $Ports_\mathsf{A} = P$ be given. Recall that we can assume that H is not clocked in Subrounds $[i.1]$. We claim that $conf_2 = (\{\mathsf{TH}\}, S, \mathsf{H}, \mathsf{Sim(A)})$ is an indistinguishable configuration for $conf_1$, i.e., $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$.

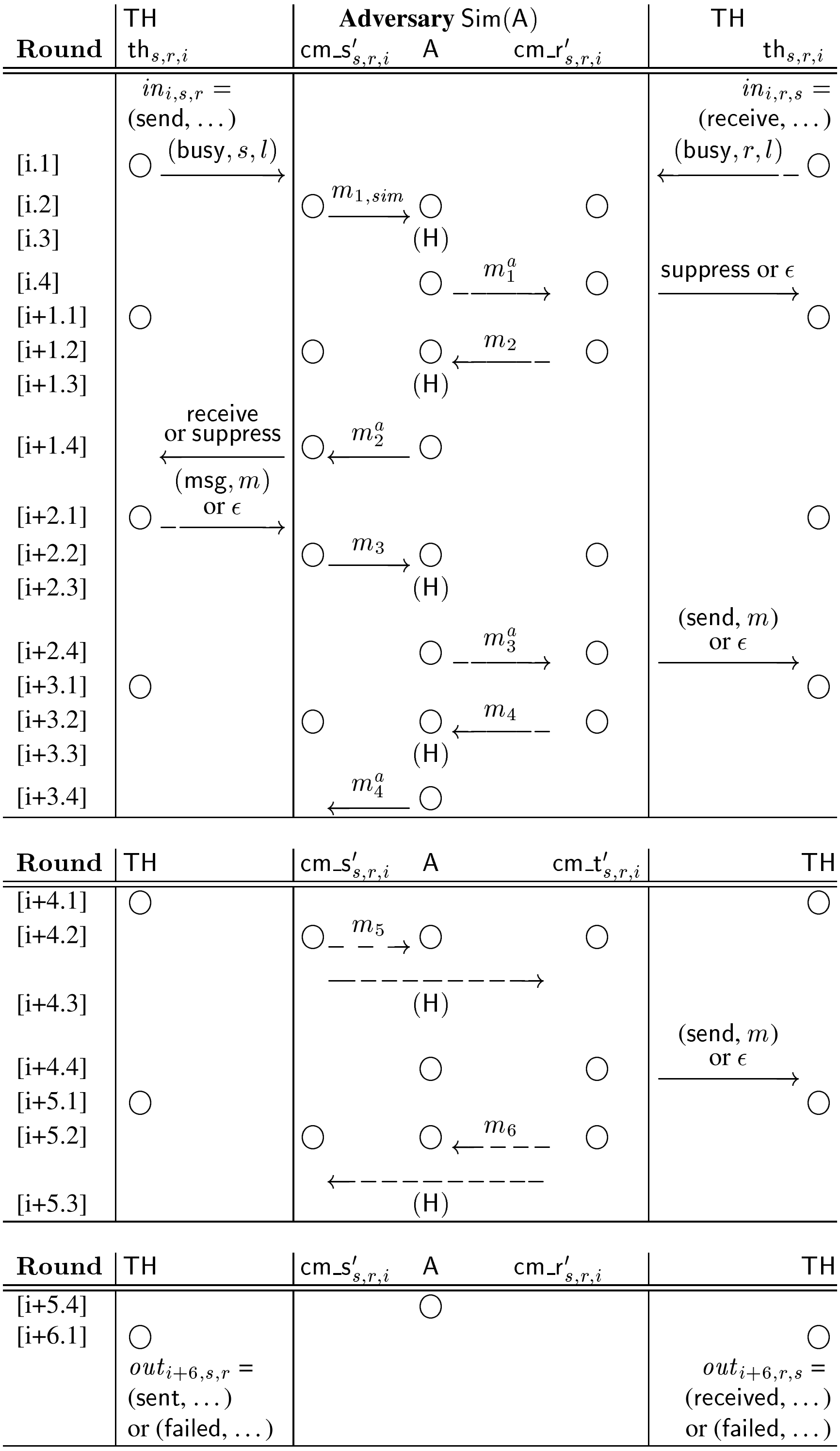

Figure 10: Simulation of a subprotocol run. Mainly the case with $s, r \in \mathcal{H}$ and an unsuppressed run is shown, but also all possible reactions on errors as signaled to TH. The symbol ◯ denotes switching and “(H)” that of H. Messages with superscript a are those arriving at ports $\mathsf{net}^a_{w,u}?$.

	Adversary $\mathsf{Sim}(\mathsf{A})$		TH
Round	A	$\mathsf{cm\_v}'_{s,r,i}$	$\mathsf{th}_{s,r,i}$
[j.1]			○
[j.2]	○		
[j.3]	(H)		
[j.4]	○ $\xrightarrow{m_6 \text{ or } m_7}$	○ $\xrightarrow{\mathsf{show}}$	
[j+1.1]			○
			$(\mathsf{received}, \ldots)$ or ϵ

Figure 11: Simulation of the verification.

A. Overview. We show the stronger statement $view_{conf_1}(\mathsf{H}, \mathsf{A}) \approx view_{conf_2}(\mathsf{H}, \mathsf{A})$. Intuitively this means that we compare M and the combination of TH and Sim. For this, we first show that the clocking makes no difference.

The cryptographic proof ideas should then be fairly clear from the description of the simulator, but we have to translate them into reduction proofs. Whether the suppression possibilities and timing are the same may actually seem less clear; the timing is illustrated in Figures 10 and 11.

More rigorously, we define a partial function ϕ from λ-round prefixes of runs of $conf_2$ for all λ to prefixes of runs of $conf_1$ (for the same k and λ) and show the following properties:

a) On the entire runs, ϕ is defined for all but a negligible subset (in k).

b) It respects prefixes, i.e., if ρ is a prefix of ρ^*, then $\phi(\rho)$ is a prefix of $\phi(\rho^*)$.

c) If $\phi(\rho)$ is defined, the joint view of H and A in ρ equals that in $\phi(\rho)$.

d) As far as ϕ is defined, it retains probabilities: For all k and all prefix lengths λ, the distribution $\phi(run_{conf_2,k,\lambda})$ equals the conditional distribution of the image of ϕ in the distribution $run_{conf_1,k,\lambda}$.

By Properties a), c) and d), the views are even statistically indistinguishable for the class $NEGL$. This implies the desired result (recall Definitions 5.2 and 5.3).[24]

Showing Part c) rigorously for composed automata with a state space this size is not trivial, even without the exceptions arising from cryptographic aspects. First note that a run (or λ-round prefix) ρ is uniquely determined by k and the random values $rand$ chosen by all machines of the configuration (during the λ rounds). Hence we only have to define a partial function ϕ' on those. We then have deterministic configurations and have to prove equality of the views in the resulting runs $\rho_{2,k,rand}$ and $\rho_{1,k,\phi'(rand)}$ (while the internal states are different). For this, we define a partial function ψ_{rand} between reachable global states of the two configurations and show that the same input in corresponding states leads to the same output and corresponding next states as long as ϕ is defined.

We structure this proof by identifying a substructure $struc_{s,r,i}$ of the real structure and $struc'_{s,r,i}$ of the simulation for each slot (s, r, i). We show that different substructures do not interact except via H and A and by using global keys (once they have obtained their different initial states.) One can then define ϕ' and ψ_{rand} and prove their properties more or less separately for each substructure, i.e., they are composed of functions $\phi'_{s,r,i}$ and $\psi_{rand,(s,r,i)}$ of random values used by, and states of, the substructures (and identity functions on global parts).[25]

As the transition diagrams for the submachines are moderately straight-line, we carry out the reachability analysis of global states of the substructures, the definition of $\phi'_{s,r,i}$ and $\psi_{rand,(s,r,i)}$, and the

[24] Nevertheless, we only obtain "$\geq^{f,\mathsf{poly}}_{\mathsf{sec}}$" because the properties only hold for polynomial-time H and A.

[25] "More or less separately" means that, while the compositions are as said, the definitions of $\phi'_{s,r,i}$ and $\psi_{rand,(s,r,i)}$ for different slots are not completely independent.

correctness proof together by a parallel walkthrough through $struc_{s,r,i}$ and $struc'_{s,r,i}$ with a small number of cases. In most places, we map a new random value r_x used in the simulation to the same value r_x in the real system, i.e., ϕ' is defined componentwise on $rand$ and is the identity on most components. After the first time, we will only mention the definition of ϕ' where it is not the identity for a component, and, in slight abuse of notation, we also call the component functions ϕ'. Also, except for the first time, we do not mention $\psi_{rand,(s,r,i)}$ at all; it simply maps the states that we consider at the same time on each other.

In these walkthroughs, we collect the values $rand$ for which we do not define ϕ' (and thus ψ_{rand}) in the following *error sets* (we omit an index k at each set for brevity):

- $Forge_u$ with $u \in \mathcal{H} \cup \{t\}$ (for the forgery of a signature sig_f of participant u),
- $OwfBreak_{s,r,i}$ (for breaking the one-way function) with $(s,r,i) \in Slots_{i_{max}(k)}$, where i_{max} denotes a polynomial bounding the number of rounds with this A and H,
- $BindBreak$ (for breaking the binding property of the commitment scheme), and
- $ComOwfBreak_{s,r,i}$ with $(s,r,i) \in Slots_{i_{max}(k)}$ (for breaking the one-way property of the commitment scheme, Lemma 6.1).

At the end, we show that these error sets are negligible (and thus that Property a) holds) by global reduction proofs with the security of the cryptographic primitives. (Slot-wise reductions are not possible because the keys are common to all substructures.) For use in those reductions, the walkthroughs also show the following properties:

e) The conditions defining the error sets are functions of the views of H and A up to a round where ϕ is still defined.[26] Together with Property c) this implies that the conditions can be verified in one run $\rho_{2,k,rand}$ or $\rho_{1,k,\phi'(rand)}$ alone.

f) Membership in the error sets is efficiently verifiable, i.e., without significant overhead one can simulate a run $\rho_{2,k,rand}$ or $\rho_{1,k,\phi'(rand)}$ and stop at the first occurrence of a condition that puts $rand$ in one (any or a specific one) of the error sets.

g) For each run in an error set, we identify a forged signature sig_f in the first case and similar values for the other cases.

B. Clocking. We defined $\mathsf{Sim}(\mathsf{A})$ in $conf_2$ as a combination based on a collection where Sim and A are separate (we now call it $conf_2^6$) and a 6-subround clocking scheme κ_6. Hence $view_{conf_2}(\mathsf{H},\mathsf{A})$ equals $view_{conf_2^6}(\mathsf{H},\mathsf{A})$ except for subround renaming. In $conf_2^6$, only Sim and TH are clocked in Subrounds 4b, 1, and 2a. Let $\mathsf{TH}+\mathsf{Sim}$ denote the combination of TH and Sim where these subrounds are joined (again according to Lemma 4.1 of [PSW00b]). Then $conf_2^* = (\mathsf{TH}+\mathsf{Sim}, S, \mathsf{H}, \mathsf{A})$ is a configuration with the standard clocking scheme which we can compare with $conf_1$, and $view_{conf_2^*}(\mathsf{H},\mathsf{A})$ equals $view_{conf_2^6}(\mathsf{H},\mathsf{A})$ except for the subround renaming. In both renamings of $conf_2^6$, H ends up in Subround 3 and A in 2 and 4. Hence

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

and in the following, we actually compare $conf_2^*$ and $conf_1$.

[26] In each case, the condition first becomes true by an output of A to $\mathsf{TH}+\mathsf{Sim}$ or M, respectively, and the views would only become unequal when $\mathsf{TH}+\mathsf{Sim}$ and M switch in Subround 1 of the next round.

C. Defining Substructures. We now define substructures containing all submachines that handle a slot (s, r, i). This is illustrated in Figure 12.

By $struc_{s,r,i}$, we denote $\mathsf{cm\_s}_{s,r,i}$, $\mathsf{cm\_r}_{s,r,i}$, $\mathsf{cm\_t}_{s,r,i}$, and $\mathsf{cm\_v}_{s,r,i}$. By $struc'_{s,r,i}$, we denote $\mathsf{th}_{s,r,i}$, $\mathsf{cm\_s}'_{s,r,i}$, $\mathsf{cm\_r}'_{s,r,i}$, $\mathsf{cm\_t}'_{s,r,i}$, $\mathsf{cm\_v}'_{s,r,i}$, a machine $\mathsf{clk}_{s,r,i}$ that only outputs stop in Round $i+6$, and a machine $\mathsf{dis}_{s,r,i}$ that dispatches internally between the other machines. Its ports are as shown in the figure, i.e., $struc'_{s,r,i}$ has the same free ports as $struc_{s,r,i}$. The dashed lines represent authentic channels.[27] If $s \notin \mathcal{H}_S$, $\mathsf{cm\_s}_{s,r,i}$ and $\mathsf{cm\_s}'_{s,r,i}$ are missing and $\mathsf{dis}_{s,r,i}$ and the connections are modified accordingly; and similarly for $r \notin \mathcal{H}_R$.

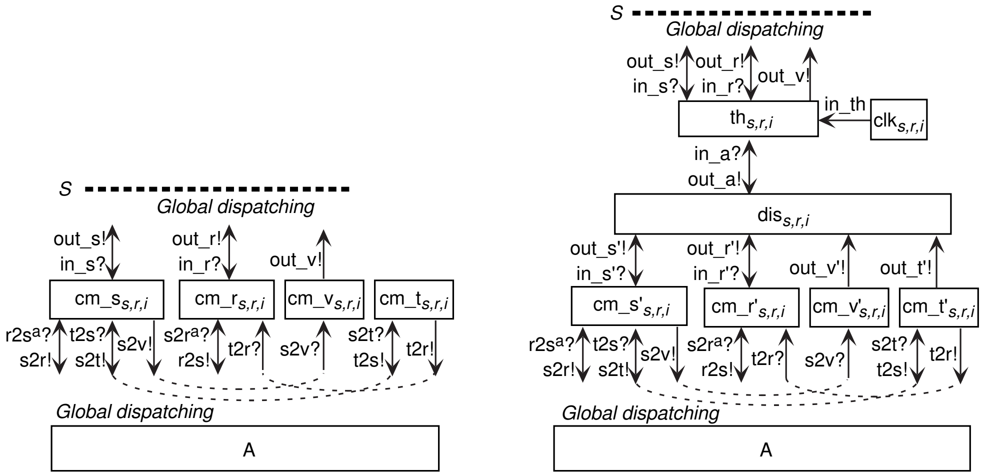

Figure 12: Substructures $struc_{s,r,i}$ and $struc'_{s,r,i}$ for $s, r \in \mathcal{H}$. The index s, r, i of all ports has been omitted. The dashed lines are authentic, but not private channels.

D. Initialization. The initialization is equal in Sim and the machines M_u by construction, and hence in the overall systems because TH and the specified ports are not involved. (The clocking becomes identical by the combination of $\mathsf{TH} + \mathsf{Sim}$ in $conf_2^*$.) Hence on the random values $rand_{glob}$ used here (the only ones outside substructures) we can indeed define ϕ' to be the identity function, and similarly ψ_{rand} on the global part of the state. We can now restrict ourselves to rounds $i > 2$ without considering broadcasts again.

E. Correct Dispatching. We first show that all inputs at the ports $\mathsf{in}_u?$ or at the network lead to identical inputs to each $struc_{s,r,i}$ and $struc'_{s,r,i}$. Similarly, we show that outputs from free ports of $struc_{s,r,i}$ or $struc'_{s,r,i}$ lead to identical global outputs. Finally, we define $\mathsf{dis}_{s,r,i}$ and show that it dispatches correctly as in the global simulation, and that the global dispatching in fact corresponds to the authentic channels.

Inputs at the specified ports: An input $in_{i,s,r}$ with $s \in \mathcal{H}_S, r \in \mathcal{M}_R$ is dispatched to port $\mathsf{in\_s}_{s,r,i}?$ of $\mathsf{cm\_s}_{s,r,i}$ by M_s and of $\mathsf{th}_{s,r,i}$ by TH. An input $in_{i,s,v} = (\mathsf{show}, (s, r, j))$ is forwarded to $\mathsf{in\_s}_{s,r,j}?$ of $\mathsf{cm\_s}_{s,r,j}$ and $\mathsf{th}_{s,r,j}$, respectively. An input $in_{i,r,s}$ with $r \in \mathcal{H}_R$ is input to $\mathsf{in\_r}_{s,r,i}?$ of $\mathsf{cm\_r}_{s,r,i}$ or $\mathsf{th}_{s,r,i}$. Note that this implies that inputs $(\mathsf{send}, \ldots)$ at $\mathsf{in\_s}_{s,r,i}?$ and any inputs at $\mathsf{in\_r}_{s,r,i}?$ are only possible in Round i, and that their syntactic correctness has already been verified.

All other inputs at S are ignored. Furthermore, the only spontaneous action of TH is to make inputs stop; this has been built into $\mathsf{clk}_{s,r,i}$. The machines M_u and Sim make no spontaneous actions.

[27]Recall that channels are no explicit components in our system model; the ports are named correspondingly. Of course we have to show below that these connections have the same effects as the global dispatching when the ports are internal to different machines. Recall also that for brevity we have omitted the replicated ports that our model would use for the authentic channels.

Network in- and outputs: By construction, Sim dispatches network inputs, i.e., those at ports with net in their names, just like the machines M_u. Similarly, it collects outputs to network ports from the submachines just like the machines M_u.

Outputs at the specified ports: TH dispatches outputs from $\mathsf{out\_s}_{s,r,i}!$ of $\mathsf{th}_{s,r,i}$ to $\mathsf{out}_s!$, those from $\mathsf{out\_r}_{s,r,i}!$ to $\mathsf{out}_r!$, and those from $\mathsf{out\_v}_{s,r,i}!$ to $\mathsf{out}_v!$. The machines M_s, M_r, and M_v do the same for the corresponding ports in $struc_{s,r,i}$. No other outputs at the specified ports are made in the real system and by TH.

Definition and correctness of $\mathsf{dis}_{s,r,i}$*:* In the "upward" direction, in Round j Sim takes the outputs of the submachine ports $\mathsf{out\_s}'_{s,r,i}!$, $\mathsf{out\_r}'_{s,r,i}!$, $\mathsf{out\_t}'_{s,r,i}!$, and $\mathsf{out\_v}'_{s,r,i}!$, and puts them into the set $in\_a_{j,(s,r,i)}$. Then TH dispatches precisely this set to $\mathsf{in\_a}_{s,r,i}?$ of $\mathsf{th}_{s,r,i}$. Hence here, $\mathsf{dis}_{s,r,i}$ can simply join the output sets.[28]

In the "downward" direction, TH puts the outputs at $\mathsf{out\_a}_{s,r,i}!$ into $out\_a_{j,(s,r,i)}$. Then Sim dispatches them according to Figure 9; they all remain in $struc'_{s,r,i}$. No other inputs are made at these ports by Sim. Hence here, $\mathsf{dis}_{s,r,i}$ simply implements Figure 9.

Finally, we show that the authentic connections shown by dashed lines in Figure 12 are correct: All outputs at a port $\mathsf{u2w}_{s,r,i}!$ are dispatched to $\mathsf{net}_{u,w}!$ and transported to $\mathsf{net}_{u,w}?$. Then they are dispatched to $\mathsf{u2w}_{s,r,i}?$ because they all have (s,r,i) as their first component (or leftmost leaf).

From now on, we consider the substructures $struc_{s,r,i}$ and $struc'_{s,r,i}$ for one particular slot (s,r,i). Hence we omit the index s,r,i of the submachines and their ports for brevity. By "a machine $\mathsf{cm\_x}$ receives a correct message m_j" we always mean that the message passes the verifications of $\mathsf{cm\_x}$ defined for its type. We have to distinguish three cases.

F. Comparison of Substructures

Case S: Correct Sender; Incorrect Recipient. We first compare $struc_{s,r,i}$ and $struc'_{s,r,i}$ for $s \in \mathcal{H}_S$ and $r \in \mathcal{A}_R$. In this case, we have Figure 12 without $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$.

1. States reached without input send*:* By the global dispatching, the only round where $\mathsf{cm\_s}$ and th (with the index (s,r,i)) can obtain an input (send, r, l, m) at port $\mathsf{in\_s}?$ is Round i. If they do not, both remain in their starting state forever and do not make any outputs; hence $\mathsf{cm\_s}'$ does not get an input $(\mathsf{send}, \ldots)$ via dis either and also remains in its starting state. As the only inputs to $\mathsf{cm\_t}$, $\mathsf{cm\_t}'$, $\mathsf{cm\_v}$, and $\mathsf{cm\_v}'$ are on authentic connections from $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$, respectively, they also never get inputs and make outputs.

2. States reached on input send*:* If (send, r, l, m) is input at $\mathsf{in\_s}?$ of th in Round i, it changes to State s_1 and outputs (busy, s, l) at $\mathsf{out\_a}!$, which dis dispatches as $(\mathsf{send}, r, l, m_{sim})$ to $\mathsf{cm\_s}'$. Thus $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ send m_1 and $m_{1,sim}$, respectively. These messages only differ in the commitments c and c_{sim}. We therefore define $\phi'(r_{sim}, r_{trans}) = r_S \leftarrow \mathsf{trans}_t(c_{sim}, m_{sim}, r_{sim}, m)$, where r_{trans} is used if random bits are needed in this process.[29] By the chameleon property, the resulting c equals c_{sim}, and thus $m_1 = m_{1,sim}$ in the corresponding states. Furthermore, the chameleon property implies that the distribution of (c, r_S) in $\phi(run_{conf_2,k})$ equals that in $run_{conf_1,k}$, as required for Property d) of ϕ.[30]

[28] As in the ideal system, $\mathsf{th}_{s,r,i}$ must be clocked once for each such input. Hence it must be considered a submachine of $\mathsf{dis}_{s,r,i}$.

[29] Recall that we let $\mathsf{cm\_s}'$ choose a value r_{trans} already in this round. Also note that r_S may depend on other slots via m, which was chosen by H and A.

[30] Let us once mention how this first walkthrough step implies a definition of $\psi_{rand,(s,r,i)}$: If we call the first state after ps_0 in Figure 5 ps_1, then we have shown that only states $(\mathsf{th}.(\mathsf{s}_1, l, m), \mathsf{cm\_s}'.(\mathsf{ps}_1, l, m_{sim}, r_{sim}, r_{trans}))$ are reachable from the initial state in $struc'_{s,r,i}$, and defined their $\psi_{rand,(s,r,i)}$-image to be $\mathsf{cm\_s}.(\mathsf{ps}_1, l, m)$ (with the same l, m). There may be one more type of state component: the internal memory of the signature system for each secret key; these states are mapped identically. This is possible because identical messages are signed.

No m_2**.** If the adversary does not respond with a correct m_2 (the test is equal for $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ in the given states), then $\mathsf{cm\_s}$ outputs $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out\_s!}$ in Round $i+6$, while $\mathsf{cm\_s}'$ immediately sends $\mathsf{suppress}$ via dis to th, which changes from State $\mathsf{s_1}$ to $\mathsf{s_2}$. Thus, when clk inputs stop in Round $i + 6$, th outputs $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out\_s!}$ as well. No network outputs are made in this process, and all three machines end up in State failed.

Correct m_2**.** If $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ receive a correct m_2, $\mathsf{cm\_s}$ sends m_3, while $\mathsf{cm\_s}'$ inputs $\mathsf{receive}$ to th, which changes from State $\mathsf{s_1}$ to $\mathsf{s_3}$ and outputs (msg, m). Thus $\mathsf{cm\_s}'$ never stops at this point, and it sends m_3 still in Round $i + 2$ using the value $r_S \leftarrow \mathsf{trans}_t(c, m_{sim}, r_{sim}, m)$, using r_{trans} in the process. By definition of ϕ', this r_S is also used by $\mathsf{cm\_s}$.

No further actions of $\mathsf{cm\_s}'$ refer to r_{sim} and m_{sim}. Hence after sending m_3, the behavior of $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ with respect to the network is identical. If they send a message m_5, it arrives at $\mathsf{cm\_t}$ and $\mathsf{cm\_t}'$ because these connections are authentic, and it passes the test by definition, so that $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ will obtain a correct m_6. Thus in Round $i + 6$, $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ either both enter the state $\mathsf{received}$ or both $\mathsf{received}'$, and th enters $\mathsf{received}$ and remains in this state and in $\mathsf{showing}$. In this round, both $\mathsf{cm\_s}$ and th also output $(\mathsf{sent}, (s, r, i))$ at $\mathsf{out\_s!}$.

This covers all states reachable and inputs accepted by $\mathsf{cm\_s}$, $\mathsf{cm\_s}'$, and th as long as the only input at $\mathsf{in\_s?}$ is $(\mathsf{send}, \ldots)$. As the only inputs to $\mathsf{cm\_t}$, $\mathsf{cm\_t}'$, $\mathsf{cm\_v}$, and $\mathsf{cm\_v}'$ are on authentic connections from $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$, respectively, the same holds for those machines.

3. Reactions on input show*:* Now we consider an input $(\mathsf{show}, (s, r, i))$ in Round j at $\mathsf{in\_s?}$. Machine $\mathsf{cm\_s}$ considers this input if it is in State $\mathsf{received}$ or $\mathsf{received}'$, while th considers it in State $\mathsf{received}$ or $\mathsf{showing}$. Above we showed that the machines enter these states under the same conditions in Round $i + 6$, and that $\mathsf{cm\_s}'$ is then in the same state as $\mathsf{cm\_s}$. Now th goes into State $\mathsf{showing}$ and outputs (busy, v), which dis dispatches as $(\mathsf{show}, (s, r, i))$ to $\mathsf{cm\_s}'$. Hence $\mathsf{cm\_s}'$ sends the same message m_6 or m_7 to $\mathsf{cm\_v}'$ as $\mathsf{cm\_s}$ to $\mathsf{cm\_v}$ on authentic connections. By construction, these messages are accepted. Thus $\mathsf{cm\_v}$ outputs $(\mathsf{received}, (s, r, i), l, m)$ in Round $j + 1$, just like th does because it is in State $\mathsf{showing}$.[31]

Case R: Correct Recipient; Incorrect Sender. We now compare $struc_{s,r,i}$ and $struc'_{s,r,i}$ for $r \in \mathcal{H}_R$ and $s \in \mathcal{A}_S$. In this case, we have Figure 12 without $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$.

In Parts 1 and 2 we consider all states reachable and inputs accepted except by $\mathsf{cm\_v}$ and $\mathsf{cm\_v}'$.

1. States reached without input $\mathsf{receive}$*:* By the global dispatching, the only round where $\mathsf{cm\_r}$ and th can obtain an input $(\mathsf{receive}, s, l)$ at port $\mathsf{in\_r?}$ is Round i. If they do not, both remain in their starting state forever and do not make any outputs; hence $\mathsf{cm\_r}'$ does not get an input $(\mathsf{receive}, \ldots)$ via dis either and also remains in its starting state and does not make outputs.

If the adversary nevertheless inputs a correct m_5 at $\mathsf{s2t?}$ of $\mathsf{cm\_t}$ and $\mathsf{cm\_t}'$ in Round $i + 5$ (the only round where these machines accept inputs), this must contain a correct message m_2, i.e. $\mathsf{test}_r(m_2) = ((s, r, i), \mathsf{m2}, \ldots)$. Then we let *rand* be in the set $Forge_r$ and $sig_f = m_2$. Note that no other submachine signs a message starting $((s, r, i), \mathsf{m2}, \ldots)$ with sign_r, and recall that tuple decomposition is unambiguous. Hence $\mathsf{cm\_t}$ and $\mathsf{cm\_t}'$ also do not make any outputs. (And recall that the verifier submachines are considered separately below.)

2. States reached on input $\mathsf{receive}$*:* If $(\mathsf{receive}, s, l)$ is input at $\mathsf{in\_r?}$, then th changes to State $\mathsf{r_1}$ and outputs (busy, r, l), which dis dispatches as $(\mathsf{receive}, s, l)$ to $\mathsf{cm\_r}'$. Thus $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ wait for an input m_1 in Round $i + 1$.

[31] In addition, $\mathsf{cm\_v}'$ outputs show, which is dispatched to th, but th ignores it, being in State $\mathsf{showing}$. This corresponds to the fact that the trusted host specifies that showing receipts by a correct sender must *always* work.

No m_1. If the adversary does not send a correct m_1, then $\mathsf{cm\_r}$ outputs $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out\_r}!$ in Round $i + 6$, while $\mathsf{cm\_r}'$ immediately outputs $\mathsf{suppress}$ to th, which changes from State r_1 to r_2. Thus, when clk inputs stop in Round $i + 6$, th outputs $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out\_r}!$ as well and changes into State failed. No network outputs are made in this process, and $\mathsf{cm\_r}$, $\mathsf{cm\_r}'$ and th never accept any other inputs.

Again, if the adversary nevertheless inputs a correct m_5 to $\mathsf{cm\_t}$ and $\mathsf{cm\_t}'$ in Round $i + 5$, we let $rand$ be in $Forge_r$ and $sig_f = m_2$ for the message m_2 contained in m_5.

m_1 and m_3. If $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ receive a correct m_1, both send m_2. If they receive a correct $m_3 = ((s, r, i), m, r_S)$ in Round $i + 3$, both send m_4 and $\mathsf{cm\_r}'$ outputs (send, m) at $\mathsf{out\_r}'!$. This is dispatched to th, which changes into State r_3. Thus both $\mathsf{cm\_r}$ and th output $(\mathsf{received}, (s, r, i), m)$ at $\mathsf{out\_r}!$ in Round $i + 6$, and all three make no further network outputs or accept other inputs.

If the adversary additionally inputs a correct m_5 to $\mathsf{cm\_t}$ and $\mathsf{cm\_t}'$ in Round $i+5$, they both output m_6 at $\mathsf{t2r}!$. Additionally, $\mathsf{cm\_t}'$ outputs $(\mathsf{send}, \ldots)$ at $\mathsf{out\_t}'!$, but th ignores it, being already in State r_3.

m_1, no m_3, but m_5. If $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ do not receive a correct m_3 (after sending m_2), both wait until Round $i + 6$.

If $\mathsf{cm\_t}$ and $\mathsf{cm\_t}'$ obtain a correct m_5 in Round $i + 5$, they both send m_6, and $\mathsf{cm\_t}'$ outputs (send, m) at $\mathsf{out\_t}'!$ for the m contained in m_5. Thus th changes to State r_3 and outputs $(\mathsf{received}, (s, r, i), m)$ at $\mathsf{out\_r}!$ in Round $i + 6$. As the channel from $\mathsf{cm\_t}$ to $\mathsf{cm\_r}$ is authentic, $\mathsf{cm\_r}$ obtains m_6 as sent by $\mathsf{cm\_t}$ and also outputs $(\mathsf{received}, (s, r, i), m)$ at $\mathsf{out\_r}!$.

m_1, no m_3, no m_5. In this case, if $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ obtain a correct m_6 in Round $i + 6$, we let $rand$ be in $Forge_t$ and $sig_f = m_6$. Note that the message signed in m_6 starts $((s, r, i), \mathsf{m6}, \ldots)$ and no other submachine signs such a message with sign_t.

Otherwise, $\mathsf{cm\_r}$ outputs $(\mathsf{failed}, (s, r, i))$ in Round $i + 6$, and so does th (changing to State failed) because we saw that it is still in State r_1.

3. Inputs to $\mathsf{cm\_v}$ *and* $\mathsf{cm\_v}'$*:* The remaining accepted inputs are at the ports $\mathsf{s2v}?$ of $\mathsf{cm\_v}$ and $\mathsf{cm\_v}'$; they must be made in a round $j \geq i + 7$ and must be correct receipts. (Thus primarily, this part of the proof proves that receipts are unforgeable and fixed.) We now denote receipts and their parts by m_i', and messages handled by the other submachines by m_i.

If a correct message m_6' or m_7' arrives, $\mathsf{cm\_v}$ outputs $(\mathsf{received}, (s, r, i), l', m')$ at $\mathsf{out\_v}!$, while $\mathsf{cm\_v}'$ outputs show at $\mathsf{out\_v}'!$, which is dispatched to $\mathsf{in\_a}?$. Then th also outputs $(\mathsf{received}, (s, r, i), l', m')$ at $\mathsf{out\_v}!$ if it is in State $\mathsf{received}$ with the parameters $l = l'$ and $m = m'$. We show that this is true except in certain (rare) cases.

$l' = l$. Both a correct m_6' and m_7' must contain correct messages m_1' and m_2'. If $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ did not send a message m_2 with the same content $((s, r, i), \mathsf{m2}, m_1, p_R)$, let $rand$ be in $Forge_r$ and $sig_f = m_2'$. Note that no other submachine signs a message starting $((s, r, i), \mathsf{m2}, \ldots)$ with sign_r.

From now on, we consider that they sent m_2 and thus $m_1' = m_1$. The verifications in $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ imply that the value l' in m_1 equals l as it was input to $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$, and thus to th. Furthermore, $\mathsf{cm\_r}'$ makes no output $\mathsf{suppress}$ and thus th never changes to State r_2.

State $\mathsf{received}$**.**

i. If a correct m_7' is shown, it contains a correct m_4', in particular a value r_R' with $\mathsf{owf}(r_R') = p_R$. If $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ did not send m_4, let $rand$ be in the set $OwfBreak_{s,r,i}$. Note that the original r_R is internal to $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ and only used in the assignment $p_R \leftarrow \mathsf{owf}(r_R)$.

If $\mathsf{cm\_r}'$ sent m_4, it must have received a correct m_3, i.e., $((s, r, i), m, r_S)$ with $m \in \mathit{Msg}$ and $\mathsf{com}_t(m, r_S) = c$ for the fourth component, c, of m_1. Then it output (send, m) at $\mathsf{out\_r}'!$, which caused th to change to State r_3, and thus in Round $i + 6$ to $\mathsf{received}$, with this parameter m.

ii. If a correct m'_6 is shown, but $\mathsf{cm\_t}$ and $\mathsf{cm\_t}'$ did not send m_6, let rand be in Forge_t and $\mathit{sig}_f = m'_6$. Note that no other submachine signs a message starting $((s, r, i), \mathsf{m6}, \ldots)$ with sign_t.

If $\mathsf{cm\_t}$ and $\mathsf{cm\_t}'$ sent m_6, they must have received a correct triple (m''_1, m''_2, m''_3). If the content of m''_2 is unequal to that of m_2 sent by $\mathsf{cm\_r}$, we let rand be in Forge_r with $\mathit{sig}_f = m''_2$. Otherwise, we have $m''_1 = m_1$ and $m''_3 = ((s, r, i), m, r_S)$ with $m \in \mathit{Msg}$ and $\mathsf{com}_t(m, r_S) = c$ for the fourth component, c, of m_1. Then $\mathsf{cm\_t}'$ output (send, m). This caused th to change to State r_3, and thus in Round $i + 6$ to $\mathsf{received}$, with this parameter m, except if it is already in State r_3 with a parameter $m'' \neq m$. This would imply that $\mathsf{cm\_r}'$ output (send, m''), which it does only after receiving a correct m_3 containing m'', r''_S with $\mathsf{com}_t(m'', r''_S) = c = \mathsf{com}_t(m, r_S)$. We then let rand be in $\mathit{BindBreak}$ and $\mathit{bindbreak} = (m, r_S, m'', r''_S)$.

$m' = m$**.** Both m'_6 and m'_7 must also contain a correct m'_3, i.e., $((s, r, i), m', r'_S)$ where $m' \in \mathit{Msg}$ and $\mathsf{com}_t(m', r'_S) = c$ for the fourth component, c, of m_1. This m' is indeed the one that $\mathsf{cm\_v}$ outputs.

If $m' \neq m$, let rand be in the set $\mathit{BindBreak}$ and $\mathit{bindbreak} = (m, r_S, m', r'_S)$ with m, r_S from m_3 or m'_3 as derived under "State received".

Case SR: Correct Sender and Recipient. Finally, we compare $\mathit{struc}_{s,r,i}$ and $\mathit{struc}'_{s,r,i}$ for $s \in \mathcal{H}_S$ and $r \in \mathcal{H}_R$. In this case, we have Figure 12 with all machines. As in the first case, inputs to $\mathsf{cm\_t}$, $\mathsf{cm\_v}$ and $\mathsf{cm\_t}'$, $\mathsf{cm\_v}'$ can only come from $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$, respectively.

1. States reached without input send*:* As in the first two cases, $\mathsf{cm\_s}$, $\mathsf{cm\_r}$ and th can only obtain inputs (send, r, l, m) and $(\mathsf{receive}, s, l')$ at ports $\mathsf{in\_s}?$ and $\mathsf{in\_r}?$ in Round i. If neither of these inputs occurs, they remain in their starting state forever without making any outputs, and so do $\mathsf{cm\_s}'$ and $\mathsf{cm\_r}'$.

2a. Input send *alone:* If (send, r, l, m) is input, but $(\mathsf{receive}, s, l')$ is not, then th changes to State sr_1 and outputs (busy, s, l), which dis dispatches as $(\mathsf{send}, r, l, m_{sim})$ to $\mathsf{cm\_s}'$, while $\mathsf{cm\_r}'$ obtains no input. $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ then send m_1 and $m_{1,sim}$, respectively. Precisely as in Case S, we define ϕ' such that $m_1 = m_{1,sim}$. $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ never leave their starting state and send nothing. If the adversary now inputs a correct m_2 to $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$, let rand be in Forge_r and $\mathit{sig}_f = m_2$. (As before, note that no other submachine signs a message $((s, r, i), \mathsf{m2}, \ldots)$ with sign_r.) Otherwise, $\mathsf{cm\_s}$ outputs $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out\_s}!$ in Round $i + 6$, and so does th, being in State sr_1. No further outputs are made, or inputs accepted, in this process, and $\mathsf{cm\_s}$, $\mathsf{cm\_s}'$, and th are in State failed.

2b. Input $\mathsf{receive}$ *alone:* If $(\mathsf{receive}, s, l)$ is input, but $(\mathsf{send}, \ldots)$ is not, th changes to State sr_2 and outputs (busy, r, l), which dis dispatches as $(\mathsf{receive}, s, l)$ to $\mathsf{cm\_r}'$. Then $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ wait for m_1, while $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ remain in their starting states without making any outputs. If the adversary inputs a correct m_1 to $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$, let rand be in Forge_s and $\mathit{sig}_f = m_1$. Note that no other submachine signs a message $((s, r, i), \mathsf{m1}, \ldots)$ with sign_s. Otherwise, $\mathsf{cm\_r}$ outputs $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out\_r}!$ in Round $i + 6$ and so does th, being in State sr_2. No further outputs are made, or inputs accepted, in this process.

2c. Different labels: If inputs (send, r, l, m) and $(\mathsf{receive}, s, l')$ with $l \neq l'$ are made, th changes to State sr_3 and makes outputs (busy, s, l) and (busy, r, l'), which are dispatched as $(\mathsf{send}, r, l, m_{sim})$ and $(\mathsf{receive}, s, l')$. Hence $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ send m_1 and $m_{1,sim}$. Precisely as in Case S, we define ϕ' such that $m_1 = m_{1,sim}$. If the adversary now inputs an m'_1 to $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ that passes their test with l'

(while m_1 contains l), let the run be in $Forge_s$ and $sig_f = m_1'$. Otherwise, $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ do not send any messages. If the adversary can then input a correct m_2 to $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$, let $rand$ be in $Forge_r$ and $sig_f = m_2$. Otherwise, $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ do not send further messages either, and $\mathsf{cm\_s}$ and $\mathsf{cm\_r}$ output $(\mathsf{failed}, (s, r, i))$ in Round $i + 6$. So does th, being in State sr_3. $\mathsf{cm\_s}$, $\mathsf{cm\_s}'$, and th are in State failed.

2d. Two matching inputs: Finally, let inputs (send, r, l, m) and $(\mathsf{receive}, s, l)$ be made. Then th goes to State sr_4 and (after outputs busy and $\mathsf{dispatching}$), $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ send $m_1 = m_{1,sim}$, respectively.

No correct m_1^a. If the adversary does not forward a correct m_1^a to $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$, then $\mathsf{cm\_r}$ outputs $(\mathsf{failed}, (s, r, i))$ in Round $i + 6$, while $\mathsf{cm\_r}'$ outputs $\mathsf{suppress}$, which causes th to change to State sr_3 and thus to output $(\mathsf{failed}, (s, r, i))$ at both $\mathsf{out\_s}!$ and $\mathsf{out\_r}!$ in Round $i + 6$. If the adversary inputs a correct m_2 to $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ in Round $i + 2$, let $rand$ be in $Forge_r$ and $sig_f = m_2$. Otherwise, $\mathsf{cm\_s}$ also outputs $(\mathsf{failed}, (s, r, i))$ in Round $i + 6$, and no machine makes any further output or considers inputs. $\mathsf{cm\_s}$, $\mathsf{cm\_s}'$, and th are in State failed.

Correct m_1^a, no correct m_2^a. If the adversary forwards a correct m_1^a to $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$, both send m_2. If the value c^a in m_1^a differs from c in m_1, let $rand$ be in $Forge_s$ and $sig_f = m_1^a$. Thus from now on, we can assume $c^a = c$.

If no correct m_2^a is forwarded by the adversary, $\mathsf{cm\_s}$ outputs $(\mathsf{failed}, (s, r, i))$ in Round $i + 6$, while $\mathsf{cm\_s}'$ inputs $\mathsf{suppress}$ to th, which changes to State sr_3 and thus to failed in Round $i + 6$, outputting $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out\_s}!$ and $\mathsf{out\_r}!$. They do not make further outputs or consider inputs.

If the adversary nevertheless inputs a correct m_3^a to $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$, let $rand$ be in $ComOwfBreak_{s,r,i}$. Note that m_3^a must contain values (m^a, r_S^a) with $c = \mathsf{com}_t(m^a, r_S^a)$, and that the value r_{sim} from the assignment $(c, r_{sim}) \leftarrow \mathsf{comr}_t(m_{sim})$ in $\mathsf{cm\_s}'$, and similarly r_S in $\mathsf{cm\_s}$, has not yet been used anywhere else.

Otherwise, $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ wait for m_6, but this does not come: It could only come over an authentic connection from $\mathsf{cm\_t}$ and $\mathsf{cm\_t}'$, and those only react on a message over an authentic connection from $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$, respectively. Hence $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ also do not send further messages, and $\mathsf{cm\_r}$ outputs $(\mathsf{failed}, (s, r, i))$ at $\mathsf{out\_r}!$ as well.

Correct m_1^a and m_2^a. If the adversary forwards a correct m_2^a, $\mathsf{cm\_s}$ sends m_3, while $\mathsf{cm\_s}'$ first only outputs $\mathsf{receive}$ at $\mathsf{out\_s}'!$, which is dispatched to th. Thus th changes from State sr_4 to sr_5 and outputs (msg, m), which is dispatched to $\mathsf{cm\_s}'$. Then $\mathsf{cm\_s}'$ sends $m_3 = ((s, r, i), m, r_S)$ as well (still in the same round). If m_2^a contains a value $p_R^a \neq p_R$, let $rand$ be in $Forge_r$ and $sig_f = m_2^a$. Otherwise, we can now speak of one fixed p_R.

Now th will output $(\mathsf{sent}, (s, r, i))$ at $\mathsf{out\_s}!$ and $(\mathsf{received}, (s, r, i), m)$ at $\mathsf{out\_r}!$ in Round $i + 6$ and from then on always be in State $\mathsf{received}$ or $\mathsf{showing}$. Furthermore, the behavior of all corresponding machines with respect to the network is clearly identical from now on, and $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ enter the same final state. Hence only the final states and outputs of $struc_{s,r,i}$ remain to be derived.

i. If the adversary does not forward a correct m_3^a, then $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ do not send m_4 and wait for m_6. If the adversary nevertheless inputs a correct m_4^a to $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$, it must contain an r_R^a with $\mathsf{owf}(r_R^a) = p_R^a = p_R$. Then let $rand$ be in $OwfBreak_{s,r,i}$ and note that the original r_R is internal to $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ and only used in the one assignment $p_R \leftarrow \mathsf{owf}(r_R)$. Otherwise, $\mathsf{cm\_s}$ sends m_5. It arrives at $\mathsf{cm\_t}$ because the connection is authentic and passes the test by construction. Hence $\mathsf{cm\_t}$ sends m_6 to $\mathsf{cm\_s}$ and $\mathsf{cm\_r}$, again over authentic connections. Hence they make outputs $(\mathsf{sent}, (s, r, i))$ at $\mathsf{out\_s}!$ and $(\mathsf{received}, (s, r, i), m)$ at $\mathsf{out\_r}!$ in Round $i + 6$ as desired, and $\mathsf{cm\_s}$ is in State $\mathsf{received}'$.

ii. Now let the adversary input a correct message $m_3^a = ((s,r,i), m^a, r_S^a)$ to $\mathsf{cm\_r}$. If $m^a \neq m$, then $\mathsf{com}_t(m^a, r_S^a) = \mathsf{com}_t(m, r_S) = c^a = c$. Then let $rand$ be in $BindBreak$ and $bindbreak = (m, r_S, m^a, r_S^a)$. Otherwise, $\mathsf{cm\_r}$ now stores m and outputs $(\mathsf{received}, (s,r,i), m)$ in Round $i+6$. Then $\mathsf{cm\_s}$ either obtains a correct m_4^a, or it sends m_5 and gets m_6 as in Case i. In both cases, it outputs $(\mathsf{sent}, (s,r,i))$ in Round $i+6$ and changes to State $\mathsf{received}$ or $\mathsf{received}'$.

3. Reactions on input show. The argument for an input $(\mathsf{show}, (s,r,i))$ is identical to Case S, except that State sr_5 plays the role of State s_3.

G. Final Reductions. We have now carried out the program described in Part A, except that it remains to be shown that the sequence of unions of the error sets (for which the mappings are not defined) has negligible probability (in k). Written with the index k, which was so far omitted for brevity, these sets are $Forge_{u,k}$ with $u \in \mathcal{H} \cup \{t\}$, $OwfBreak_k = \bigcup_{(s,r,i) \in Slots_{i_{max}(k)}} OwfBreak_{s,r,i,k}$, $BindBreak_k$, and $ComOwfBreak_k = \bigcup_{(s,r,i) \in Slots_{i_{max}(k)}} ComOwfBreak_{s,r,i,k}$. This is a constant number of sets (independent of k; this is why we took the two unions). Hence if each sequence has a negligible probability, then so has the sequence of unions.

Hence we now assume for contradiction that one sequence has a larger probability.

1. $(Forge_{u,k})_{k \in \mathbb{N}}$ *for a certain* $u \in \mathcal{H} \cup \{t\}$. We construct an adversary $\mathsf{A_{sig}}$ against the signer machine Sig from Definition 6.1. It simulates the configuration $conf_2^*$ using the public key test obtained from Sig as test_u. I.e., it runs this configuration with k as the initial state of each machine and a random value $rand$ chosen during the run as usual, except that it sends every message m_j to be signed with sign_u to Sig instead and uses the result as the signature.[32] In addition, it keeps track of the conditions for putting $rand$ in $Forge_{u,k}$. (By Properties e) and f) from Part A they can be verified efficiently on $conf_2^*$ alone.) If one of them is fulfilled, $\mathsf{A_{sig}}$ outputs the designated value sig_f as its forged signature. In each case, it was already shown in the walkthrough that sig_f is indeed a valid signature for test_u and that the contained message m_f was not signed by the given submachine or any other simulated machine, i.e., $\mathsf{A_{sig}}$ did not ask Sig to sign m_f.

Hence the success probability of $\mathsf{A_{sig}}$ for each k is at least the probability of the set $Forge_{u,k}$, which is the desired contradiction.

2. $(OwfBreak_k)_{k \in \mathbb{N}}$. We construct an adversary $\mathsf{A_{owf}}$ as in Definition 6.2. On input $(1^k, p)$, it first chooses a slot $(s,r,i) \in Slots_{i_{max}(k)}$ randomly. Then it simulates $conf_2^*$ using the given p as p_R in the submachine $\mathsf{cm\_r}'_{s,r,i}$, instead of setting $p_R = \mathsf{owf}(r_R)$ for random r_R. It then checks if the condition for putting $rand$ into $OwfBreak_{s,r,i,k}$ is fulfilled. (There is one such condition each in Item i. of Case R and Case SR, and (s,r,i) fixes which of them applies, if any.) If yes, a value r'_R (called r_R^a in the second case) with $\mathsf{owf}(r'_R) = p_R = p$ is obtained, and $\mathsf{A_{owf}}$ outputs it. It was already shown in the walkthroughs that the (now unknown) value r_R was not used outside the replaced assignment $p_R = \mathsf{owf}(r_R)$ up to this point; hence the simulation is possible.

Hence for any $rand \in OwfBreak_{s',r',i',k}$ for a slot $(s',r',i') \in Slots_{i_{max}(k)}$, the probability that $\mathsf{A_{owf}}$ is successful in the sense of Definition 6.2 for this value $rand$ is at least $|Slots_{i_{max}(k)}|^{-1}$. (Because $\mathsf{A_{owf}}$ is certainly successful if $(s,r,i) = (s',r',i')$.) The overall success probability of $\mathsf{A_{owf}}$ is therefore at least $(n_S n_R i_{max}(k))^{-1}$ times the probability of the set $OwfBreak_k$. As $i_{max}(k)$ is polynomial, this is still not negligible.

3. $(BindBreak_k)_{k \in \mathbb{N}}$. We construct an adversary $\mathsf{A_{bind}}$ as in Definition 6.3a. It obtains a public commitment key $pk_{\mathsf{C},u}$, which it uses as $pk_{\mathsf{C},t}$ in a simulation. However, without the secret key, it cannot execute trans_t. Hence we let $\mathsf{A_{bind}}$ simulate the real configuration $conf_1$ directly, choosing the random

[32] There is no clocking problem although we defined that Sig signs only one message per round because $\mathsf{A_{sig}}$ can clock its submachines itself, i.e., the rounds of the signature attack and of the simulated certified mail system are different.

values $rand'$ when needed. By Property d) from Part A, this gives the same probability distribution as simulating $\rho_{1,k,\phi'(rand)}$ as far as it is defined. $\mathsf{A_{bind}}$ keeps track whether the conditions for $BindBreak_k$ are fulfilled; this can be done efficiently on a run $\rho_{1,k,\phi'(rand)}$ by Properties e) and f). Recall that there were two conditions at the end of Case R and one at the end of Case SR. Each time, $\mathsf{A_{bind}}$ outputs the designated tuple $bindbreak$, for which we have already shown that it fulfils the condition from Definition 6.3a.

Hence the success probability of $\mathsf{A_{bind}}$ for all k is at least the probability of the set $BindBreak_k$, which is the desired contradiction.

4. $(ComOwfBreak_k)_{k\in\mathbb{N}}$. We construct adversary algorithms A_1, A_2 as in Lemma 6.1b. On input $(1^k, pk_{\mathsf{C},u})$, A_1 first chooses a slot $(s,r,i) \in Slots_{i_{max}(k)}$ randomly. Then it starts simulating $conf_1$, using the given commitment key $pk_{\mathsf{C},u}$ as $pk_{\mathsf{C},t}$. If the (simulated) H inputs (send, r, l, m) at $\mathsf{in}_s?$ in Round i, then A_1 outputs this m as its own m and its entire state as aux. Thus A_2 can continue the simulation, using its additional input c as the commitment c in m_1 for this slot (s,r,i), instead of choosing it as $(c, r_S) \leftarrow \mathsf{comr}_t(m)$. If the condition for $ComOwfBreak_{s,r,i,k}$ (in Case SR under "Correct m_1^a, no correct m_2^a") is fulfilled, A_2 outputs (m^a, r_S^a). We have already shown that this fulfils the condition from Lemma 6.1b and that r_S (now unknown) is not used so that the simulation is possible.

Hence, similar to Case 2, the overall success probability of A_1 and A_2 is at least $(n_S n_R i_{max}(k))^{-1}$ times the probability of $ComOwfBreak_k$ and thus still not negligible.

Hence we have shown that if any sequence of error sets had more than negligible probability, we could break one of the underlying cryptographic primitives. This finishes the proof. ■

8 Conclusion

We have proven the security of an efficient certified-mail system in the framework of a general simulatability definition. Apart from the value for certified mail, we believe that this is convincing evidence that general simulatability definitions, in particular that from [PSW00b], are a useful basis for specifying and proving the security of practical reactive systems.

We have also shown how further properties of the protocol can then be derived from the ideal system used as a specification by applying a theorem on the preservation of integrity properties from [PW00]. This is one step in a program to link cryptographic systems and abstract models accessible to formal methods. However, actual formal methods remain to be applied. The first step is to express the ideal system in a standard specification language.

Orthogonally to the primary goals of the paper, we have tried to make the proofs really rigorous. The typical cryptographic proof sketches are often not convincing to the formal-methods community (and some not at all), so that one even sometimes meets the misconception that cryptography cannot provide any strict protocol proofs. This rigorosity implies that large parts concern "boring" details like dispatching and walkthroughs through many cases. The dispatching could easily be defined and proven once for a large class of systems.[33] However, the walkthroughs are system-specific, and they are typical proof parts that machines should do better than humans. Hence we hope that the abstraction (i.e., definition of corresponding ideal systems) can in the future already be applied at a lower level, so that the walkthroughs can be made with the abstractions.[34] The composition theorem from [PW00] would then imply that the real protocol is secure whenever the version with the abstract primitives is secure.

[33]These would be systems with subprotocols with a finite number of participants per subprotocol, subprotocol runs identified by some transaction IDs (here the slot numbers), and transaction IDs in all network messages. The subprotocol runs are similar to the "oracles" in [BR94].

[34]An approach of this type is [AR00], although not in the context of simulatability definitions.

Acknowledgments

We thank *Victor Shoup* and *Michael Steiner* for interesting discussions.

References

[AR00] Martín Abadi, Phillip Rogaway: Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption); to appear at IFIP International Conferences on Theoretical Computer Science (IFIP TCS2000), Sendai, Japan, August 2000.

[ASW97] N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Fair Exchange; 4th Conference on Computer and Communications Security, ACM, New York 1997, 6–17.

[ASW00] N. Asokan, Victor Shoup, Michael Waidner: Optimistic Fair Exchange of Digital Signatures; IEEE Journal on Selected Areas in Communications 18/4 (2000) 593–610.

[B83] Manuel Blum: How to Exchange (Secret) Keys; ACM Transactions on Computer Systems 1/2 (1983) 175–193.

[B91] Donald Beaver: Secure Multiparty Protocols and Zero Knowledge Proof Systems Tolerating a Faulty Minority; Journal of Cryptology 4/2 (1991) 75–122.

[BBM00] Mihir Bellare, Alexandra Boldyreva, Silvio Micali: Public-Key Encryption in a Multi-user Setting: Security Proofs and Improvements; Eurocrypt 2000, LNCS 1807, Springer-Verlag, Berlin 2000, 259–274.

[BCC88] Gilles Brassard, David Chaum, Claude Crépeau: Minimum Disclosure Proofs of Knowledge; Journal of Computer and System Sciences 37 (1988) 156–189.

[BCP88] Jurjen Bos, David Chaum, George Purdy: A Voting Scheme; unpublished manuscript, presented at the rump session of Crypto '88.

[BD84] Andrei Z. Broder, Danny Dolev: Flipping coins in many pockets (Byzantine agreement on uniformly random values); 25th Symposium on Foundations of Computer Science (FOCS), IEEE, 1984, 157–170.

[BDM98] Feng Bao, Robert Deng, Wenbo Mao: Efficient and Practical Fair Exchange Protocols with Off-Line TTP; Symposium on Research in Security and Privacy, IEEE, 1998, 77–85.

[BGMR90] Michael Ben-Or, Oded Goldreich, Silvio Micali, Ronald L. Rivest: A Fair Protocol for Signing Contracts; IEEE Transactions on Information Theory 36/1 (1990) 40–46.

[BR94] Mihir Bellare, Phillip Rogaway: Entity Authentication and Key Distribution; Crypto '93, LNCS 773, Springer-Verlag, Berlin 1994, 232–249.

[C00] Ran Canetti: Security and Composition of Multiparty Cryptographic Protocols; Journal of Cryptology 13/1 (2000) 143–202.

[CHP92] David Chaum, Eugène van Heijst, Birgit Pfitzmann: Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 470–484.

[D82] Dorothy Denning: Cryptography and Data Security; Addison-Wesley, Reading 1982; reprinted with corrections, January 1983.

[D88] Ivan Bjerre Damgård: Collision free hash functions and public key signature schemes; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 203–216.

[G84] Oded Goldreich: Sending Certified Mail using Oblivious Transfer and a Threshold Scheme; Technion - Israel Institute of Technology, Computer Science Department, Technical Report, 1984.

[GJM99] Juan A. Garay, Markus Jakobsson, Philip MacKenzie: Abuse-Free Optimistic Contract Signing; Crypto '99, LNCS 1666, Springer-Verlag, Berlin 1999, 449–466.

[GL91] Shafi Goldwasser, Leonid Levin: Fair Computation of General Functions in Presence of Immoral Majority; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 77–93.

[GM84] Shafi Goldwasser, Silvio Micali: Probabilistic Encryption; Journal of Computer and System Sciences 28 (1984) 270–299.

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

[GMR89] Shafi Goldwasser, Silvio Micali, Charles Rackoff: The Knowledge Complexity of Interactive Proof Systems; SIAM Journal on Computing 18/1 (1989) 186–207.

[HM00] Martin Hirt, Ueli Maurer: Player Simulation and General Adversary Structures in Perfect Multiparty Computation; Journal of Cryptology 13/1 (2000) 31–60.

[L96] Nancy Lynch: Distributed Algorithms, Morgan Kaufmann, San Francisco 1996.

[LMMS98] P. Lincoln, J. Mitchell, M. Mitchell, A. Scedrov: A Probabilistic Poly-Time Framework for Protocol Analysis; 5th Conference on Computer and Communications Security, ACM, New York 1998, 112–121.

[M97] Silvio Micali: Certified E-Mail with Invisible Post Offices—or—A Low-Cost, Low-Congestion, and Low-Liability Certified E-Mail System; presented at RSA 97.

[MR92] Silvio Micali, Phillip Rogaway: Secure Computation; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 392–404.

[P92] Torben Pryds Pedersen: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 129–140.

[P93] Birgit Pfitzmann: Sorting Out Signature Schemes; 1st Conference on Computer and Communications Security, ACM, New York 1993, 74–85.

[PSW00a] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Cryptographic Security of Reactive Systems; Workshop on Secure Architectures and Information Flow, Electronic Notes in Theoretical Computer Science (ENTCS), March 2000, `http://www.elsevier.nl/locate/entcs/volume32.html`.

[PSW00b] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Secure Reactive Systems; IBM Research Report RZ 3206 (#93252) 02/14/2000, IBM Research Division, Zurich, May 2000.

[PW00] Birgit Pfitzmann, Michael Waidner: Composition and Integrity Preservation of Secure Reactive Systems; accepted for 7th ACM Conference on Computer and Communication Security, Athens, November 2000. Preliminary version as IBM Research Report RZ 3234 (#93280) 06/12/00, IBM Research Division, Zurich, June 2000.

[R83] Michael O. Rabin: Transaction Protection by Beacons; Journal of Computer and System Sciences 27/ (1983) 256–267.

[RS92] Charles Rackoff, Daniel R. Simon: Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 433–444.

[S00] Matthias Schunter: Optimistic Fair Exchange; PhD thesis (submitted); Universität des Saarlandes, Saarbrücken, February 2000.

[VV83] Umesh V. Vazirani, Vijay V. Vazirani: Trapdoor Pseudo-random Number Generators, with Applications to Protocol Design; 24th Symposium on Foundations of Computer Science (FOCS), IEEE, 1983, 23–30.

[Y82] Andrew C. Yao: Protocols for Secure Computations; 23rd Symposium on Foundations of Computer Science (FOCS), IEEE, 1982, 160–164.

[Y82a] Andrew C. Yao: Theory and Applications of Trapdoor Functions; 23rd Symposium on Foundations of Computer Science (FOCS), IEEE, 1982, 80–91.

[ZG97] JianYing Zhou, Dieter Gollmann: An Efficient Non-repudiation Protocol; 10th Computer Security Foundations Workshop, IEEE, Los Alamitos 1997, 126–132.