Reactively Simulatable Certified Mail

Birgit Pfitzmann, Matthias Schunter, Michael Waidner

IBM Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland,
e-mail: {bpf,mts,wmi}@zurich.ibm.com

Revision of Sept. 2004 of a journal submission from Dec. 2000.

Abstract Certified mail is the fair exchange of a message for a receipt, i.e., the recipient gets the message if and only if the sender gets a receipt. It is an important primitive for electronic commerce and other atomicity services. Certified-mail protocols are known in the literature, but there was no rigorous definition yet, in particular for optimistic protocols and for many interleaved executions. We provide such a definition via an ideal system and show that a specific real certified-mail protocol is as secure as this ideal system in the sense of reactive simulatability in the standard model of cryptography and under standard assumptions.

As certified mail without any third party is not practical, we consider optimistic protocols, which involve a third party only if one party tries to cheat. The real protocol resembles prior protocols, but we had to use a different cryptographic primitive to achieve simulatability. The communication model is synchronous.

This proof first demonstrated that a cryptographic multi-step protocol can fulfil a general definition of reactive simulatability enabling concurrent composition. We also first showed how formal-method style reasoning can be applied over the ideal system in a cryptographically sound way. Moreover, the treatment of multiple protocol runs and their modular proof in spite of the use of common cryptographic primitives for all runs can be seen as a first example of what is now known as joint-state composition.

Key Words: Certified Mail, Fair Exchange, Reactive Simulatability, Composability, Formal Methods

1 Introduction

A certified-mail protocol enables the fair exchange of a message, i.e., arbitrary data, for a receipt [18, 49]. It ensures that either the recipient gets the message and the sender gets a receipt or neither party gets anything; no knowledge about the message must leak in that case. Certified mail is an important non-repudiation primitive for electronic commerce. A receipt unambiguously characterizes the message received. This is essential in the delivery of high-value digital goods, which are usually post-paid to allow inspection before payment. It is also useful to make sure that an order has been received, in particular with goods whose price changes quickly, or if the buyer must otherwise look for another supplier on short notice. Another example is the provision of information under a non-disclosure agreement. More theoretically, certified mail can be used to provide contract signing, e.g., the fair exchange of two signatures under a contract text [52], and other atomicity services such as fair multi-party exchange of arbitrary items [2].

There are three classes of certified-mail protocols. Protocols with an in-line third party involve a trusted third party in every protocol run [49, 56, 28, 37]. The older cryptographic literature treats two-party protocols that involve only the sender and the recipient, e.g., [18, 49, 53, 31, 19]. These protocols are based on the 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. In order to minimize the involvement of a third party in realistic scenarios, optimistic protocols were introduced [3, 4, 39, 7, 57]. Here a trusted third party is available to ensure fairness, but it is only involved in a protocol run if something goes wrong in this run.

The main value of this paper is not that we propose a new optimistic certified-mail protocol, although we do. More importantly, we provide the first rigorous definition of certified mail, in particular for optimistic protocols and for many concurrent protocol runs. However, the main value in hindsight—the paper was originally written in 2000—is that it was the first example that cryptographic multi-step protocols can fulfil a definition of reactive simulatability that enables concurrent composition. This approach is nowadays better known as "universal composability" [47, 48, 23]. Our ideal certified-mail system is abstract, i.e., the specification for one protocol run is a simple, deterministic state-transition machine. Essentially it gets two inputs, a payload message from a sender and an OK to receive a message from the recipient. If these inputs fit together, it outputs the payload message to the recipient and a success indication to the sender. At any time after a success indication, the sender can input the wish to convince a verifier of the sending, and the verifier will get the output that the sender in fact sent the message to the recipient. While we will add some details below to allow efficient protocols to fulfil the specification, the ideal system remains without probabilism and cryptographic objects. Hence it is easy to encode in state-of-the-art automated proof tools for distributed systems, in order to prove larger electronic-commerce or agreement protocols based on certified mail.

Many later publications adapted other, often smaller, primitives or protocols to the reactive simulatability / universal composability setting, following either similar lines of defining abstract ideal systems as here, or working on lower levels of abstraction where there are still cryptographic objects. While it may thus seem normal nowadays that such idealizations can be made and real protocols can fulfil them, this was not at all clear in 2000. It was known from zero-knowledge proofs that some interesting systems cannot be composed concurrently in spite of fulfilling normal cryptographic definitions [30], and it was known from multi-party function evaluation that committing primitives are problematic in simulation proofs. As reactive simulatability implies concurrent composability, and all known certified-mail protocols contain somewhat committing primitives, we were very suspicious of our potential results. This was also a reason for us to make a very detailed proof. It is typically not hard to define the simulator for a simulatability proof, i.e., a machine that interacts with the ideal system and uses an arbitrary adversary against the real system as a subroutine, and that produces behavior that the honest users cannot distinguish from what would happen in the real system. One may be tempted to stop at this point and say that the simulator obviously simulates correctly. However, comparing two such systems, the ideal system plus simulator with the real interacting machines, is a distributed-system problem of significant size, and humans often misjudge the correctness of distributed systems even for much smaller examples. For instance, even for a very small system like signatures certain ideal systems and proofs had to be repaired several times [24, 8]. Thus the hard part of the proof is to rigorously show that the simulator simulates correctly. We still believe that every simulation proof should contain this part, but certainly the first large proof with respect to a new definition had to.

[1] In [17] such a lower bound was proven for two-party contract signing. Contract signing can be reduced to the type of certified mail defined here; the reduction needs only one additional round of communication and does not increase the error probability significantly [52]. The reduction is given for optimistic protocols only, but one can easily see that it also works for 2-party protocols.

Our certified-mail system is also the first example of a technique that later became known as joint-state composition [25], although the generalization was only sketched in the outlook of the original report [46]. These aspects occur on two layers: First, the certified-mail system is defined as a composition of many submachines, each handling one transaction with a specific transaction identifier. Higher protocols can therefore essentially work with independent certified-mail transactions even though joint state is present underneath. Secondly, the proof is performed modularly for the transaction machines, although they jointly use three cryptographic primitives, signatures with arbitrary memory, a one-way function, and chameleon commitments. For these, we did not use the technique that one would use for higher-level protocols using certified mail, i.e., we did not abstract them as ideal systems with their own transaction identifiers. Instead, we use a similar separation technique directly for the original definitions.

An interesting aspect is that one-way functions and commitments by themselves do not have simple idealizations in the standard model of cryptography [23]. Nevertheless, our overall result is in the standard model. This is due to restrictions in the usage of the primitives: The one-way functions are only used for one-time signatures. The commitments are used in a scenario with a trusted third party—this party already exists in optimistic certified mail to guarantee fairness, but now we also use it to choose the commitment key. Thus in the simulation the simulator can choose the commitment key, and it can later use the chameleon property to open commitments to reveal arbitrary messages, here the payload certified-mail message that the simulator did not know when simulating a commitment on this message. A definition of primitives via ideal systems with usage restrictions was recently made for symmetric encryption in [11]; similar ideas might be applicable here.

Our certified-mail protocol is synchronous, i.e., there is a notion of rounds. In practice, rounds can be realized if suitable bounds on clock deviation and message delay are given. While recent cryptographic advances focus on asynchronous protocols, in particular in the simulatability area following [48,23], synchronous protocols remain important in practice because one almost always needs timeouts anyway; e.g., recall the example of time-critical buying. Simply combining an asynchronous protocol with a timeout will typically not retain the security. It was later shown in [5] how the synchronous systems and simulatability definitions from [47] can be represented as special cases of the asynchronous versions from [48]. However, this embedding does not make concrete protocol definitions and proofs easier. Further, a synchronous proof implies the absence of timing vulnerabilities, e.g., covert channels, on the round level, while an asynchronous proof cannot do this.

Finally, our certified-mail protocol is labeled. This means that the recipient initially agrees to receive a mail under a specific subject, called label, and this label becomes part of the receipt. Labels are important for fixing a protocol context.

1.1 Further Related Literature

The underlying security definition of reactive simulatability and its composition properties were introduced in [47] and extended to asynchronous systems in [48,23]. It extends the security notions of multi-party (one-step) function evaluation [54,34,32,40,15,22] and the observational equivalence of [38]. The idea of simulatability was already used earlier for specific reactive problems, to our knowledge first in [29], and when constructing generic solutions for large classes of reactive problems [34,35], usually yielding inefficient solutions and assuming that all parties take part in all subprotocols. None of these prior reactive papers proposed a general definition or a composition theorem.

Before certified mail, secure message transmission by a combination of signatures and encryption was the only reactive (multi-step) system defined and proved with respect to a simulatability definition with known concurrent compositionality, and also only in grey literature [45]. Many difficulties that occur in certified mail do not occur there yet, e.g., multi-round subprotocols, multiple sub-protocols per transaction identifier (here sending a mail and showing the receipt), and committing primitives.

Separation of protocol runs by transaction or session identifiers is a well-known technique in practical distributed computing, although rarely treated systematically. In security, it is known from robust protocol design [1], where protocol designers are advised to add transaction, message, and participant identifiers to authenticated messages unless there is a good reason not to, although this advice is not as explicit as it could be. Not all protocols need transaction identifiers, though. For instance, authentication protocols typically choose nonces internally in the protocol; see, e.g., [16] (whose oracles correspond to transactions).

A comparison of labeled certified mail with protocols where the recipient simply agrees to receive a message, or where he need not agree at all, is given in [52] via reductions and round- and message-optimality results for all classes.

2 State-Machine Notation

We use state-transition machines as the primary system model, in other words probabilistic I/O automata similar to [51]. A Turing-machine realization is defined for the class of automata used in [47]. This allows us to define ideal systems suitable for treatment by formal proof tools when designing higher-level protocols over them, while at the same time the real systems have well-defined complexities. Further, it enables rigorous definitions of our systems. While pure Turing machines also allow this in principle, in reality one typically resorts to informal activity diagrams or state-transition machines there, too.

Each machine may have different connections, e.g., to a user and a network. These are attached at so-called input and output ports. Following CSP notation we denote input ports with $?$ and output ports with !. Ports of different machines that only differ in this "direction" are connected.

We often define state-transition functions graphically in the following standard notation: A circle is a state. An arrow labeled $in[c]/out$ is a transition resulting from an input in, dependent on condition c (which can refer to the input and the internal state) and resulting in an action out, typically an output. Inputs that are not explicitly shown in a state are ignored. No condition means true. The notation $\mathsf{x}.y$ means that y is input or output at port x. We then omit the direction of port x (the $?$ or !) because it is clear from the context. A dash means no in- or output. A state transition without input means that the machine always makes this transition in the next round. An input $\neg\mathsf{x}.y$ means that the machine makes the transition if the input y is not received at port x in the next round.

We further define that every input parameter is assigned to a variable of the same name in the machine if no assignment to such a variable occurred before in the given run of the machine; otherwise the input is only accepted if the values of the parameter and the variable are equal. Finally, T (time) globally denotes the current round number, although not all machines need to know it. The starting state is designated by an incoming arrow without source. The label of this arrow denotes initial variables in this machine.

3 Ideal System for One Labeled Certified-Mail Transaction

In this section, we present an ideal system for labeled certified mail. Defining ideal systems is an engineering discipline, even more so than making requirements-based definitions of cryptographic systems, because an ideal system contains many requirements all in one. There are multiple reasonable ideal systems for a protocol class like labeled certified mail, which allow different subsets of the implementations that one may intuitively all regard as labeled certified mail. In particular, there is often a trade-off between ideality and efficient realizability. By ideality we mean simple, strong definitions that will be most appreciated by higher-level protocols.

Actually, in this section we only define an ideal system for one transaction. Extensions to multiple transactions are discussed in Section 8; they are relatively standard for distributed systems.

3.1 Motivation of the Definition

As discussed in the introduction, a highly ideal system $\mathsf{TH}_{\mathsf{naive}}$ for certified mail might look as follows. This simple description already allows any number of transactions.

- Whenever $\mathsf{TH}_{\mathsf{naive}}$ obtains two matching inputs (send, r, l, m) at port $\mathsf{in}_s?$ 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 every non-matched such input, it outputs $(\mathsf{failed}, (s, r, i))$ to the party concerned.
- On input $(\mathsf{show}, (s, r, i))$ at $\mathsf{in}_s?$, it outputs $(\mathsf{received}, (s, r, i), l, m)$ at $\mathsf{out}_v!$ if it previously got an input (send, r, l, m) at $\mathsf{in}_s?$ and answered with $(\mathsf{sent}, (s, r, i))$.

Here $\mathsf{TH}_{\mathsf{naive}}$ stands for "naive trusted host", s and r are identities of a sender and recipient, $\mathsf{in}_s?$ and $\mathsf{in}_r?$ are input ports for these two parties, and $\mathsf{out}_s!$ and $\mathsf{out}_r!$ are the output ports where these parties get results. We modeled only one verifier with output port $\mathsf{out}_v!$ for simplicity because verifiers are stateless in our protocols and would thus all act identically. The inputs send and receive start the sending protocol, while show starts the showing of a receipt. The values l are the labels; note that the recipient inputs the label he agrees to initially; further, both parties designate the desired partner in their inputs. For simplicity, we treat the combination (s, r, i) of the partner identities and round number as transaction identifier, allowing only one protocol per such combination. The following proofs would work equally well with more such possible protocols; an additional transaction identifier is then needed in many places where now the triple (s, r, i) is used.

Unfortunately, normal certified-mail protocols do not satisfy this simple, very ideal specification.

First, $\mathsf{TH}_{\mathsf{naive}}$ hides all interactions between honest parties from the adversary. In real life, it would be very expensive to hide who communicates with whom at what time. As the secure-channel example in [44] shows how to allow just this minimal information, we decided here to model the usual unencrypted protocols from the fair-exchange literature. Thus for each transaction, our final trusted host initially forwards all non-secret parts of the user inputs to the adversary at a port $\mathsf{out\_a}!$, and later forwards (msg, m) when a payload message is sent in clear.

Secondly, $\mathsf{TH}_{\mathsf{naive}}$ guarantees availability, i.e., it always delivers certified mails between agreeing honest users. However, in usual real systems the adversary can prevent communication. Therefore our final trusted host accepts inputs suppress from the adversary that describe which protocol runs are disrupted. However, we need reliable channels to trusted third parties.

Thirdly, $\mathsf{TH}_{\mathsf{naive}}$ produces outputs in one round, while every real optimistic protocol requires at least four rounds already in the optimistic case [52]. Hence we use a number Δ of rounds as a free parameter in the specification. For brevity, we use only one such parameter for all honest parties and all cases. Showing a receipt will work in one round.

Finally, the trusted host with the additions discussed so far would make outputs to the adversary after the same time Δ as to honest parties. However, a real adversary can usually learn results faster. For simplicity, our final ideal system allows the adversary slightly more than a real adversary can achieve in any specific protocol: He can disrupt at any time, or decide that he will not disrupt and immediately learn the results.

3.2 Ideal Machines for One Transaction

As every transaction may go through several states now, we define the ideal system as a composition of state-transition machines for individual transactions. We first define these

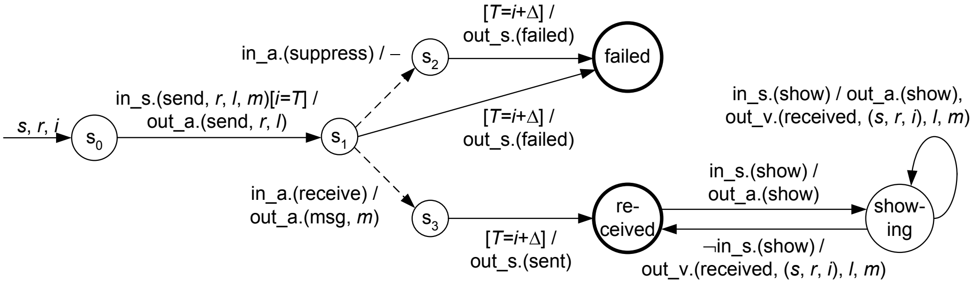
s, r, i
s_0
in_s.(send, r, l, m)[i=T] / out_a.(send, r, l)
s_1
in_a.(suppress) / –
s_2
[T=i+Δ] / out_s.(failed)
failed
[T=i+Δ] / out_s.(failed)
in_a.(receive) / out_a.(msg, m)
s_3
[T=i+Δ] / out_s.(sent)
re-ceived
in_s.(show) / out_a.(show)
¬in_s.(show) / out_v.(received, (s, r, i), l, m)
show-ing
in_s.(show) / out_a.(show), out_v.(received, (s, r, i), l, m)

Fig. 1 Ideal transaction machine ths for correct sender only. Dashed arrows arise from adversary inputs.

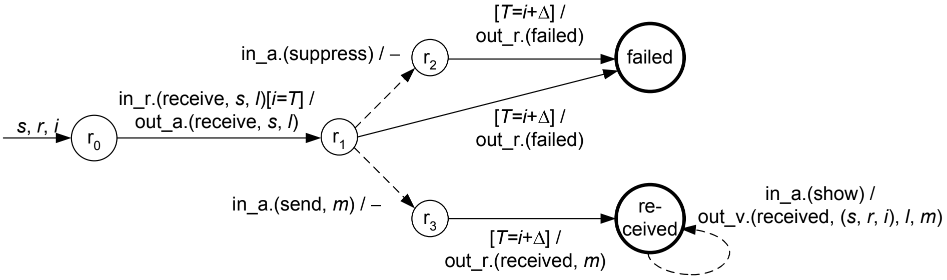

Fig. 2 Ideal transaction machine thr for correct recipient only.

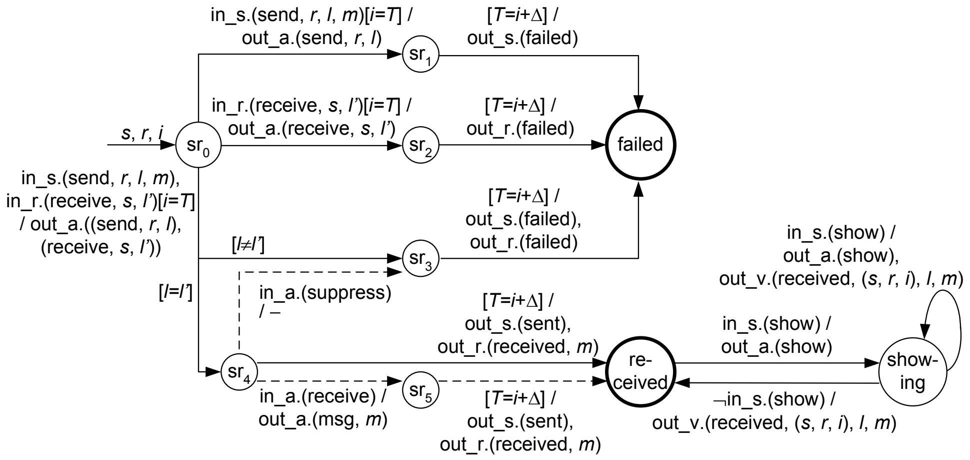
in_s.(send, r, l, m)[i=T] / out_a.(send, r, l)
sr_1
[T=i+Δ] / out_s.(failed)
in_r.(receive, s, l')[i=T] / out_a.(receive, s, l')
s, r, i
sr_0
sr_2
[T=i+Δ] / out_r.(failed)
failed
in_s.(send, r, l, m), in_r.(receive, s, l')[i=T] / out_a.((send, r, l), (receive, s, l'))
[l≠l']
sr_3
[T=i+Δ] / out_s.(failed), out_r.(failed)
[l=l']
in_a.(suppress) / –
[T=i+Δ] / out_s.(sent), out_r.(received, m)
sr_4
in_a.(receive) / out_a.(msg, m)
sr_5
[T=i+Δ] / out_s.(sent), out_r.(received, m)
re-ceived
in_s.(show) / out_a.(show)
¬in_s.(show) / out_v.(received, (s, r, i), l, m)
show-ing
in_s.(show) / out_a.(show), out_v.(received, (s, r, i), l, m)

Fig. 3 Ideal transaction machine thsr for correct sender and recipient.

transaction machines. We distinguish whether only the sender (ths), only the recipient (thr), or both parties (thsr) in the transaction are honest. The trusted third party is not visible in the ideal system; it only has an auxiliary role in real systems. Thus the ideal system is not specific to optimistic protocols. The following is the rigorous definition using the state-machine notation introduced in Section 2.

Definition 1 (Ideal Machines for One Certified-Mail Transaction) *We define machine types* ths, thr, *and* thsr *with ports for each party concerned with this transactions:* in_s? *and* out_s! *for* ths *and* thsr, in_r? *and* out_r! *for* thr *and* thsr, *and always* out_v!, in_a? *and* out_a!. *Their state-transition functions are defined in Figures 1 to 3.*

These state-transition functions correspond to the naive ideal systems with the additions motivated above: Each transaction machine is started by inputs (send, ...) and/or (receive, ...) from the correct users. If both sender and recipient are honest, the states sr_1,

sr_2, and sr_3 occur if there are not two matching inputs. The adversary immediately obtains the parameters of the run at the port $\mathsf{out\_a}!$. Then for Δ rounds he can either disrupt the run with $(\mathsf{suppress})$ or decide that it should end successfully by inputting (send, m) if he is the sender, or else $(\mathsf{receive})$. In the latter case, he immediately obtains the payload message of the honest sender. The outputs for the honest users only occur after Δ rounds. Only then can receipts be shown. Showing by an adversary works within the same round, by an honest user it has one round delay. In most user outputs we now omitted (s, r, i) because it only served as a transaction identifier in the naive ideal system, and for a one-transaction system this is clear by the machine identity.

Parameterizing the Ideal System. The ideal system from Definition 1 accepts inputs of arbitrary length. (We tacitly assume that all inputs are coded as strings over a fixed alphabet.) Further, it is willing to show receipts for one transaction arbitrarily often. One consequence is that it is not polynomial-time in the usual sense for interactive Turing machines, i.e., polynomial-time measured in a security parameter alone. We call this *strictly polynomial-time*. It is, however, *weakly polynomial-time*, i.e., polynomial-time in the overall length of inputs it receives. The system model and the simulatability definition do not require ideal systems to be polynomial-time at all; however, proofs (e.g., of higher protocols using this trusted host and other cryptographic primitives) are often simpler if all machines are strictly polynomial-time, because this is a composable notion, in contrast to weak polynomial-time.

For this purpose, we also define a parameterized version of the ideal system. It has a message space, a label space, and a bound on the number of rounds, which may depend on a security parameter k. Thus they are sequences $(Msg_k)_{k\in\mathbb{N}}$, $(L_k)_{k\in\mathbb{N}}$, and $(B_k)_{k\in\mathbb{N}}$. The message and label spaces must be polynomial-time decidable subsets of strings over the given alphabet, and the bounds $B_k \in \mathbb{N}$ polynomially bounded and polynomial-time computable. Then we modify the machines ths, thr, and thsr as follows: They only run for B_k rounds. Inputs named m and l are typed, i.e., the machines verify that $m \in Msg_k$ and $l \in L_k$ before accepting such an input. Finally, they have length bounds determining how many symbols of any input they read. Polynomial bounds sufficient for all correct inputs can easily be derived from the message formats. While this was well-defined on the Turing-machine level in the original report [46], the corresponding input-bounding functions for our state-transition machine model were only generally defined in [12], called length functions.

Independent of their use for polynomial time, restricted message and label sets may be useful in specific applications of certified mail.

4 Real Certified-Mail Protocol for One Transaction

Like the ideal system, the real system for labeled certified mail, i.e., the actual protocol, is built from machines for individual transactions, and we first define these transaction machines.

4.1 Overview

We first sketch our protocol as activity diagrams, before defining it rigorously via state-transition machines. The subprotocol for sending a mail has three potential participants, the sender's machine $\mathsf{cm\_s}$, the recipient's machine $\mathsf{cm\_r}$, and the third party's machine $\mathsf{cm\_t}$. The protocol for showing a receipt is similarly executed by a sender and a verifier. The inputs starting these subprotocols and the possible outputs to the users are the same as in the ideal system.

The subprotocol for sending is shown in Figure 4. Messages m_1 and m_2 are promises to send a payload message under label l and to give a receipt for it, respectively. The

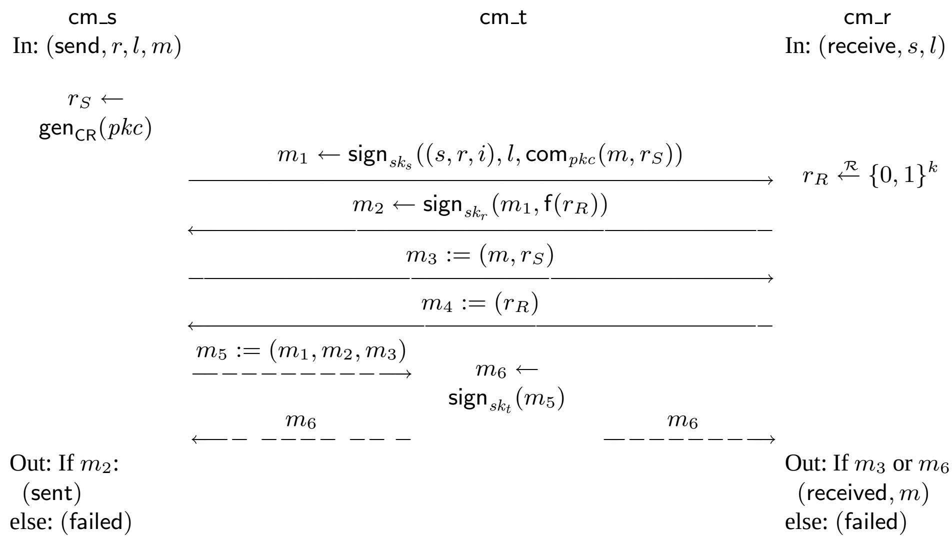
cm_s
In: (send, r, l, m)
cm_t
cm_r
In: $(\mathsf{receive}, s, l)$
$r_S \leftarrow \mathsf{gen}_{\mathsf{CR}}(pkc)$
$m_1 \leftarrow \mathsf{sign}_{sk_s}((s,r,i), l, \mathsf{com}_{pkc}(m, r_S))$
$r_R \overset{\mathcal{R}}{\leftarrow} \{0,1\}^k$
$m_2 \leftarrow \mathsf{sign}_{sk_r}(m_1, \mathsf{f}(r_R))$
$m_3 := (m, r_S)$
$m_4 := (r_R)$
$m_5 := (m_1, m_2, m_3)$
$m_6 \leftarrow \mathsf{sign}_{sk_t}(m_5)$
m_6
m_6
Out: If m_2: (sent) else: (failed)
Out: If m_3 or m_6: $(\mathsf{received}, m)$ else: (failed)

Fig. 4 Run of the sub-protocol "send". Dashed flows are only needed in the non-optimistic recovery, i.e., if m_4 is missing.

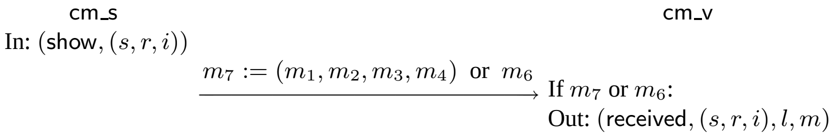
cm_s
In: $(\mathsf{show}, (s,r,i))$
cm_v
$m_7 := (m_1, m_2, m_3, m_4)$ or m_6
If m_7 or m_6:
Out: $(\mathsf{received}, (s,r,i), l, m)$

Fig. 5 Showing a receipt

value i is the number of the starting round and k a security parameter. The message part $\mathsf{com}_{pkc}(m, r_S)$ is a commitment to m, using a suitable random value generated with an algorithm $\mathsf{gen}_{\mathsf{CR}}(pkc)$. The message part $\mathsf{f}(r_R)$ is a one-way function applied to a random value; this is used as the public key for a one-time signature. If both parties are honest, the sender reveals the payload message m in m_3, and the recipient sends the one-time signature r_R as a receipt in m_4. 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. If a dishonest sender does not reveal the payload 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 $i+5$.

If the sender wants to show the receipt to a verifier, it sends (m_1, m_2, m_3, m_4) or m_6, respectively, see Figure 5. The verifier can easily verify both potential receipts, provided it knows the public keys.

4.2 Cryptographic Primitives Used and Probabilistic Notation

In the following, $NEGL$ denotes the set of all negligible functions, i.e., $g\colon \mathbb{N} \to \mathbb{R}_{\geq 0} \in NEGL$ iff for all positive polynomials Q, $\exists k_0 \forall k \geq k_0\colon\ g(k) \leq 1/Q(k)$. The notation $P(\ldots :: \ldots)$ means the probability of the event before "::" in the probability space defined by the probabilistic assignments after "::". By $[\mathsf{alg}(\cdot)]$ we denote the set of possible outcomes of a probabilistic algorithm $\mathsf{alg}(\cdot)$.

For all algorithms of the following cryptographic primitives, we assume that efficiently computable upper bounds on the output length, given the input lengths, are known, as well as for the number of random bits needed in a probabilistic algorithm.

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^+ for an alphabet Σ with $\{0,1\} \subseteq \Sigma$ and $\mathsf{false} \not\in \Sigma^+$ [27]. We write $(sk, pk) \leftarrow \mathsf{gen_S}(1^k, 1^{s^*})$ for the generation of a signing key and a test key based on a security parameter k and the desired maximum number of signatures, $s^* \in \mathbb{N}$. By $sig \leftarrow \mathsf{sign}_{sk}(m)$, we denote the (probabilistic) signing of a message m, where sk may be updated in the process. This is a simple way to represent schemes with memory. We assume that sig is of the form (m, sig'). The deterministic verification $\mathsf{test}_{pk}(sig)$ returns m or false; in the first case we say that the signature is valid. The first s^* correctly generated signatures must be valid. Security of a signature scheme means that existential forgery is infeasible even in adaptive chosen-message attacks [33]:

Definition 2 (Signature Security) *Given a signature scheme and a function s^* over $\mathbb{N}$, a signer machine Sig_{s^*} is defined as follows: It has one input and one output port, variables sk, pk, and the following transition rules:*

1. *First generate a key pair, $(sk, pk) \leftarrow \mathsf{gen_S}(1^k, 1^{s^*})$, and output pk.*
2. *On input (sign, m_j), return $sig_j \leftarrow \mathsf{sign}_{sk}(m_j)$.*

The signature scheme is called existentially unforgeable under adaptive chosen-message attack *if for every efficiently computable, polynomially bounded s^* and every probabilistic polynomial-time machine $\mathsf{A_{sig}}$ that interacts with Sig_{s^*} and outputs a value sig, the probability that $m := \mathsf{test}_{pk}(sig) \neq \mathsf{false}$ and m was not signed by Sig_{s^*} during the interaction is negligible (in k).*

Definition 3 (Security of a One-way Function) *A function $\mathsf{f} : \{0,1\}^* \to \{0,1\}^*$ is called one-way if for all probabilistic polynomial-time algorithms $\mathsf{A_{owf}}$,*

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

(as a function of k).

A non-interactive chameleon commitment scheme [21] is a tuple of algorithms $(\mathsf{gen_C}, \mathsf{gen_{CR}}, \mathsf{com}, \mathsf{trans})$ where com is deterministic, and a message space Msg_C. We write $(skc, pkc) \leftarrow \mathsf{gen_C}(1^k)$ for the generation of a key pair based on a security parameter k, and $r \leftarrow \mathsf{gen_{CR}}(pkc)$ for the generation of a suitable random value r for commitments given the public key pkc. We write $c := \mathsf{com}_{pkc}(m, r)$ for the commitment on a message $m \in Msg_\mathsf{C}$ using the public key pkc and the random value r. By $(c, r) \leftarrow \mathsf{comr}_{pkc}(m)$, we abbreviate the composition $r \leftarrow \mathsf{gen_{CR}}(pkc); c := \mathsf{com}_{pkc}(m, r)$.

A commitment c is opened by sending (m, r). The recipient verifies that $\mathsf{com}_{pkc}(m, r) = c$. By $r^* \leftarrow \mathsf{trans}_{skc}(c, m, r, m^*)$ we denote the transformation that allows the owner of the secret key skc 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^* using r^*. For all correctly generated keys and $c = \mathsf{com}_{pkc}(m, r)$, this must give $c = \mathsf{com}_{pkc}(m^*, r^*)$.

Definition 4 (Security of a Chameleon Commitment Scheme) *A non-interactive chameleon commitment scheme is called secure if it has the following three properties.*

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

$$\begin{aligned} &P(\mathsf{com}_{pkc}(m, r) = \mathsf{com}_{pkc}(m^*, r^*) \wedge m \neq m^* \\ &:: (skc, pkc) \leftarrow \mathsf{gen_C}(1^k); (m, r, m^*, r^*) \leftarrow \mathsf{A}(1^k, pkc)) \\ &\in NEGL. \end{aligned}$$

b) Perfectly hiding: *For all* $(skc, pkc) \in [\mathsf{gen_C}(1^k)]$*, 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* $m \leftarrow \mathsf{Dist}; (c, r) \leftarrow \mathsf{comr}_{pkc}(m)$.[2]

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

For example, we can use the commitment scheme from [20,26,42] with a chameleon extension, combined with a family of collision-resistant hash functions [27]. In the basic scheme, key generation $\mathsf{gen_C}(1^k)$ means to randomly choose a k-bit prime q and an $l(k)$-bit prime p with $q|(p-1)$ for a function l 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 keys are $pkc := (p, q, g, h)$ and $skc := x$. A suitable random value is chosen as $r \xleftarrow{\mathcal{R}} \mathbb{Z}_q$ and a commitment on a message $m \in Msg_\mathsf{C} := \mathbb{Z}_q$ as $c := \mathsf{com}_{pkc}(m, r) := g^m h^r \bmod p$. The transformation is $r^* := \mathsf{trans}_{skc}(c, m, r, m^*) := (m - m^*)/x + r$. The binding property holds 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_{pkh} is also (at least in theory) selected by a public key pkh that becomes part of the public key of the extended commitment scheme. The hash outputs for a security parameter k must belong to Msg_C. Given a message m, we now commit to $\mathsf{hash}_{pkh}(m)$. One easily sees 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 simple fact used.

Lemma 1 (Properties of the Commitments)

a) For all $(skc, pkc) \in [\mathsf{gen_C}(1^k)]$*, all* $m, m' \in Msg_\mathsf{C}$*, and all possible commitments* c*,*

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

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

$$\begin{aligned} &P(c = \mathsf{com}_{pkc}(m^*, r^*) \\ &:: (skc, pkc) \leftarrow \mathsf{gen_C}(1^k); (m, aux) \leftarrow \mathsf{A}_1(1^k, pkc); \\ &\quad (c, r) \leftarrow \mathsf{comr}_{pkc}(m); (m^*, r^*) \leftarrow \mathsf{A}_2(1^k, pkc, m, aux, c)) \\ &\in NEGL. \end{aligned}$$

Here aux *denotes auxiliary information that* A_1 *may hand 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:

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

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 immediately contradicts the binding property.

[2] Thus $P_{\mathsf{Dist}^*}(m|c)$ abbreviates $P(m' = m | c' = c :: m' \leftarrow \mathsf{Dist}; (c', r) \leftarrow \mathsf{comr}_{pkc}(m'))$.

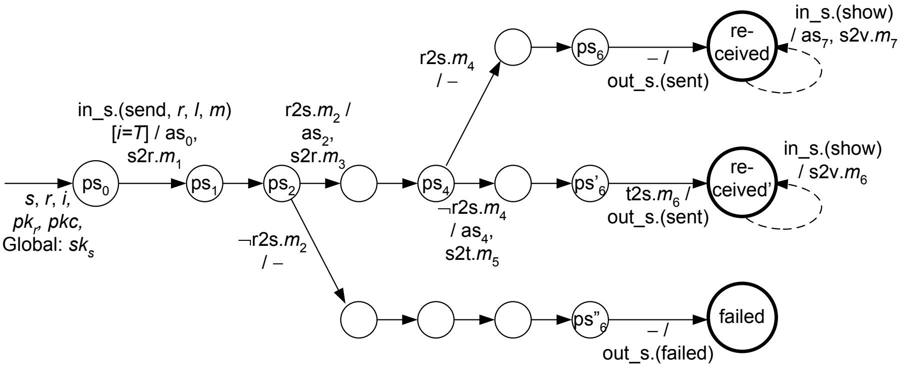

Fig. 6 Sender transaction machine cm_s.

In the second case, consider an adversary A_3 that carries out $(m, aux) \leftarrow \mathsf{A}_1(1^k, pkc)$, then chooses a message $m' \neq m$ in Msg_{C} (e.g., the first possible one out of two fixed ones), sets $(c, r) \leftarrow \mathsf{comr}_{pkc}(m')$ and finally $(m^*, r^*) \leftarrow \mathsf{A}_2(1^k, pkc, m, aux, c)$. 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 get a contradiction to the binding property.

4.3 Real Machines for One Transaction

We now define the real machines for one transaction of our certified-mail protocol in detail. Note that the same sender transaction machine once sends and then, possibly many times, shows the resulting receipt. As we allow the machines for different transactions to use a common signature key that may be updated in each signing, we define it as a global variable, indicated by “Global:” on the start arrow. Further, we define subprograms for message tests and computations that do not fit into the state-transition diagrams, called “ax_j” for party x’s action in state j.

A comma in messages denotes tuple composition, not concatenation. It must be implemented such that decomposition is unambiguous. We slightly augment the message format of Figure 4; in particular we repeat the transaction identifier in places to simplify later dispatching. Further, we add message type identifiers like m1 in signed messages, although in this protocol this is not needed because each type of machine signs only one type of message.

Definition 5 (Real Machines for One Certified-Mail Transaction) *We define machine types* cm_x *with* $x \in \{s, r, t, v\}$. *Each machine has at most one input and output port for its user,* in_x? *for* $x \in \{s, r\}$ *and* out_x! *for* $x \in \{s, r, v\}$. *Further, it has so-called network ports* $x2y$! *for outputs to a machine of type* cm_y *and* $y2x$? *for inputs from that machine. The detailed behavior of these machines is defined in Figures 6 to 8. We now define the subprograms used in the figures; they operate on the variables of the respective machine as defined by our general conventions.*

In the sender cm_s.

Subprogram as$_0$. *Let* $(c, r_S) \leftarrow \mathsf{comr}_{pkc}(m)$ *and* $m_1 \leftarrow \mathsf{sign}_{sk_s}((s, r, i), \mathsf{m1}, l, c)$.

Subprogram as$_2$. *Verify that* $\mathsf{test}_{pk_r}(m_2) = ((s, r, i), \mathsf{m2}, m_1, p_R)$ *for some value* p_R. *If yes, set* $m_3 := ((s, r, i), m, r_S)$.

Subprogram as$_4$. *Test whether* $m_4 = ((s, r, i), r_R)$ *for some value* $p_R \in \{0, 1\}^k$ *with* $\mathsf{f}(r_R) = p_R$. *If not, set* $m_5 := (m_1, m_2, m_3)$.

Subprogram as$_7$. *Set* $m_7 := ((s, r, i), \mathsf{m7}, m_1, m_2, m_3, m_4)$.

In the recipient cm_r.

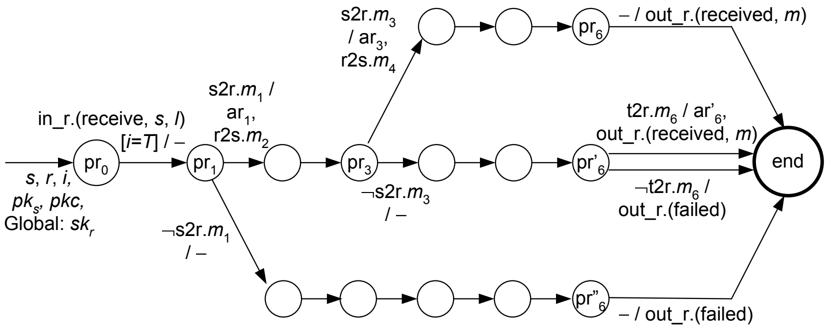

Fig. 7 Recipient transaction machine cm_r.

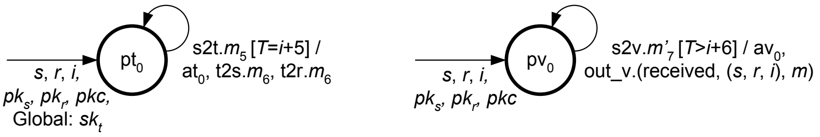

Fig. 8 Transactions machines cm_t and cm_v for third party and verifier.

Subprogram ar_1. *Verify that* $\mathsf{test}_{pk_s}(m_1) = ((s,r,i), \mathsf{m1}, l, c)$ *for some value* c. *If yes, set* $r_R \overset{\mathcal{R}}{\leftarrow} \{0,1\}^k$ *and* $m_2 \leftarrow \mathsf{sign}_{sk_r}((s,r,i), \mathsf{m2}, m_1, \mathsf{f}(r_R))$.

Subprogram ar_3. *Verify that* $m_3 = ((s,r,i), m, r_S)$ *for some values* m *and* r_S *with* $\mathsf{com}_{pkc}(m, r_S) = c$. *If yes, set* $m_4 := ((s,r,i), r_R)$.

Subprogram ar'_6. *Retrieve the signed message* $((s,r,i), \mathsf{m6}, m_5)$ *from* m_6, *decompose* m_5 *into* (m_1, m_2, m_3), *and decompose* m_3 *into* $((s,r,i), m, r_S)$.

In the third-party cm_t, *subprogram* at_0. *Test that* m_5 *is a triple* (m_1, m_2, m_3) *with* $\mathsf{test}_{pk_s}(m_1) = ((s,r,i), \mathsf{m1}, l, c)$ *for some values* l *and* c, *and* $\mathsf{test}_{pk_r}(m_2) = ((s,r,i), \mathsf{m2}, m_1, p_R)$ *for some value* $p_R \in \{0,1\}^k$. *Finally, test that* $m_3 = ((s,r,i), m, r_S)$ *for some values* m *and* r_S *with* $\mathsf{com}_{pkc}(m, r_S) = c$. *Then set* $m_6 \leftarrow \mathsf{sign}_{sk_t}((s,r,i), \mathsf{m6}, m_5)$.

In the verifier cm_v, *subprogram* av_0. *Verify that* m'_7 *is of the form* $m_6 = (((s,r,i), \mathsf{m6}, m_5), sig)$ *or* $m_7 = ((s,r,i), \mathsf{m7}, m_1, m_2, m_3, m_4)$ *for some values* m_i *and* sig. *In the first case, test if* $\mathsf{test}_{pk_t}(m_6) = ((s,r,i), \mathsf{m6}, m_5)$ *and verify and decompose* m_5 *as defined for* cm_t. *In the second case, verify and decompose* $m_5 := (m_1, m_2, m_3)$ *in the same way and then verify* m_4 *as defined for* cm_s.

Part of the trust model for this protocol is that the channels to and from the third party and the verifier are reliable and authentic, i.e., messages arrive as sent, although the adversary may read them. The assumption that other parties can reliably communicate with the third party is essential for optimistic (and in-line) certified mail. For the verifier, it is less essential, but otherwise we would have to allow the adversary to suppress runs of the "show" protocol in the ideal system. The authenticity of these channels could be realized cryptographically, while the reliability has to be assumed.

We can parameterize these real machines with the same message and label spaces and bounds on the number of rounds as the ideal system. Again we then define that the machines stop after B_k rounds, that inputs called m and l are typed, and we define length functions that bound the number of symbols read in each input. We use the same length functions for inputs at the service ports as in the ideal system. The bounds needed for network messages can easily be derived from the message formats and the assumed upper bounds on the outputs of the cryptographic primitives. For polynomial-size messages and labels and a polynomial bound on the rounds, we then obtain strictly polynomial-time real machines, while the machines in Definition 5 are only weakly polynomial-time.

5 Reactive Simulatability

In order to prove that the real system (for one or many transactions) is as secure as the ideal system, we need a summary of the underlying security definitions from [47]. As said in Section 2, the machine model is essentially normal probabilistic state-transition machines and we do not repeat a formal definition here.[3] The machines are connected according to the port naming convention, i.e., messages output at $\mathsf{p}!$ arrive at $\mathsf{p}?$. A *collection* of machines is a set of machines compatible with this connection convention, i.e., no port occurs twice. A collection is called *closed* if all ports are connected.

Runs (executions) are defined for closed collections essentially in the usual way for probabilistic state-transition machines (see [51]), in particular in the synchronous case needed here. Only, to capture security notions like rushing adversaries, we allow different clocking schemes. A clocking scheme with n subrounds for a machine collection $\hat{M}$ is a function $\kappa : \{1, \ldots, n\} \to \mathcal{P}(\hat{M})$; it assigns each subround the set of machines switching in this subround.

For security purposes, we distinguish certain free ports of a collection of machines as *service ports*. This is where the honest users of the protocol (human users or a higher-level protocol) are supposed to make in- and outputs. For instance, in both the real and the ideal certified-mail system for one transaction with honest sender and recipient, the service ports are the set $S := \{\mathsf{in\_s}?, \mathsf{out\_s}!, \mathsf{in\_r}?, \mathsf{out\_r}!, \mathsf{out\_v}!\}$. When comparing systems (e.g., a real one and an ideal one), we restrict ourselves to systems with the same service ports. We call a pair $(\hat{M}, S)$ of a machine collection and distinguished service ports a *structure*. For instance, (thsr, S) and $(\{\mathsf{cm\_s}, \mathsf{cm\_r}, \mathsf{cm\_t}, \mathsf{cm\_v}\}, S)$ are corresponding real and ideal structures for one transaction of certified mail.

A structure becomes a *configuration* $(\hat{M}, S, \mathsf{H}, \mathsf{A})$ by adding an adversary A and honest users H. Both are normal probabilistic state-transition machines, but typically universally quantified, while the machines of the structure are predefined. The honest users H must only link to the machines of the structure by the service ports, while the adversary links to the other free ports, e.g., insecure network connections. Together, a configuration must be a closed collection. We use the clocking scheme $(\hat{M}, \mathsf{A}, \mathsf{H}, \mathsf{A})$. The notation exploits that functions over $\{1, \ldots, n\}$ are equivalent to tuples, and we omitted brackets around one-element sets. Thus not only the adversary, but also the honest users are rushing. The motivation for this clocking scheme and equivalence results can be found in [45]. For configurations, we typically consider the case where the initial state of each machine is just the security parameter k. Then the runs are a family of probability distributions (in other words random variables), one for each value of k.[4]

Views of machines and *traces* of the events at certain ports are defined in a natural way as restrictions (projections) of runs. Details of the run representation and these projections are not needed in the following; they can be found in [47].

Often one defines an *intended structure* for a real system, in other words a protocol and intended connections for a number of parties, plus a *trust model* that defines how many of these parties might be corrupted and how, and how secure the connections are. We may thus obtain a set of possible real structures. To allow many types of trust models, the general security definitions in [47] are for arbitrary sets of possible structures, called *systems*. The certified-mail system for one transaction has three possible structures corresponding to a trust model where the sender, the recipient, or none may be corrupted. As said above, channels to and from $\mathsf{cm\_t}$ and $\mathsf{cm\_v}$ are authentic, the others insecure. The latter means that both machines actually interact with the adversary; this is formally realized by a port

[3] The only special aspect is the length functions that may bound the length of the inputs read at each port in each state, as explained above and first defined precisely in [12].

[4] Our transaction machines have initial variables; nevertheless we measure the complexity in terms of k. Leaving open where the initial variables come from allows different versions of multi-transaction systems later, see Section 8.

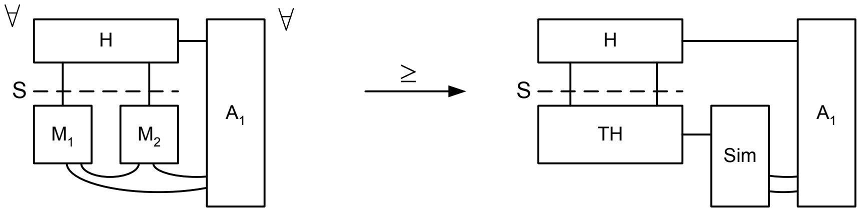

Fig. 9 Example of blackbox simulatability; the two views of the user H are indistinguishable.

renaming convention defined in the derivation of a system from an intended structure and a trust model.

The notion of reactive simulatability essentially means that it makes no difference for the honest users whether they interact with the real system or the ideal system. This is illustrated in Figure 9 (already for the usual stronger case of blackbox simulatability). More precisely, their view is indistinguishable in corresponding structures. Indistinguishability is an important cryptographic concept, first from [55].

Definition 6 (Computational 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 computationally indistinguishable ("*$\approx_{\mathsf{poly}}$*") iff for every algorithm* Dist *(the distinguisher) that is probabilistic polynomial-time in its first input,*

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

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 now denote the random variable of the view of a machine M in a configuration conf by $\mathit{view}_{\mathit{conf}}(\mathsf{M})$. Then reactive simulatability is defined as follows.

Definition 7 (Computational Reactive Simulatability for Structures) *We say that a structure* $(\hat{M}_1, S)$ *is* computationally at least as secure as *a structure* $(\hat{M}_2, S)$ *and write* $(\hat{M}_1, S) \geq^{\mathsf{poly}}_{\mathsf{sec}} (\hat{M}_2, S)$ *iff for every configuration* $\mathit{conf}_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1)$*, there exists a configuration* $\mathit{conf}_2 = (\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2)$ *such that*

$$\mathit{view}_{\mathit{conf}_1}(\mathsf{H}) \approx_{\mathsf{poly}} \mathit{view}_{\mathit{conf}_2}(\mathsf{H}).$$

Blackbox *simulatability, written* $\geq^{\mathsf{poly,bb}}_{\mathsf{sec}}$ *means that, given* $(\hat{M}_1, S)$ *and the set* P *of adversary ports,* A_2 *is a fixed simulator* Sim *with* A_1 *as a blackbox submachine.*

A typical simulator does not modify the communication between A_1 and H (as already shown in Figure 9) and thus only depends on the structure $(\hat{M}_1, S)$. Further, for blackbox simulatability, one typically shows that the joint view of H and A_1 is indistinguishable. Thus the distinction of H and A_1 disappears, while the service ports remain important to decide how Sim connects to the ideal system.

The reactive simulatability definition is lifted from structures to systems (sets of structures) by comparing "corresponding" structures defined by some mapping. Certified mail belongs to the canonical case where a set S of service ports uniquely characterizes a corresponding real and ideal structure, corresponding to a set of correct participants according to the trust model. We also write "$\geq^{\mathsf{poly}}_{\mathsf{sec}}$" for systems.

Note that this is only a brief summary of some definitions from [47] and that the report [45] contains even more variants as well as equivalence proofs. Further note that although we usually talk of real and ideal systems, the definitions do not need such a distinction.

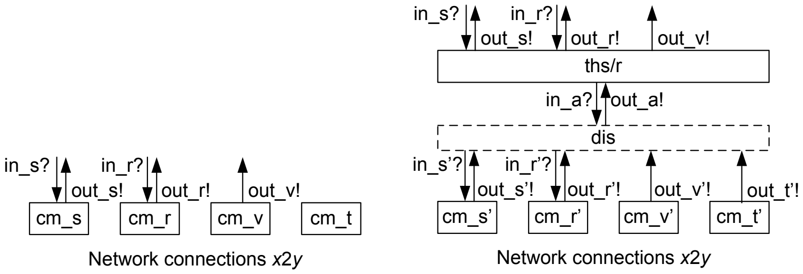

Fig. 10 Machines for one transaction in the real system (left) and the ideal system with simulator (right). Machine ths/r stands for ths, thr, or thsr depending on which parties are correct.

6 Simulator for One Transaction

We will prove that the real certified-mail system (for one or many transactions) is as secure as the ideal system in the sense of blackbox simulatability. For this, we first define a simulator for one transaction.

The main cryptographic aspect is the simulation of the message m_1 of the subprotocol "send" if the sender is honest: In Round i, where m_1 should be sent, the ideal system does not reveal the payload message m the honest user wants to send. Nevertheless, the simulator has to input a correct-looking network message to the adversary A, which includes a commitment that is supposed to fix m. 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. This is why we need the chameleon property: It allows the simulator (which also simulates the machine cm_t and thus knows its secret commitment key skc) to 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 the protocol runs in certain states, to show receipts too early, etc., that are not specified in the ideal system.

We structure the simulator for one transaction similar to the real system, i.e., it explicitly simulates machines cm_s and/or cm_r, cm_t, and cm_v. Additionally it translates signals that the ideal system outputs at port out_a! into user inputs to these machines and vice versa. The machines, instead of making their normal final user outputs, recognize certain intermediate situations and output the decision to suppress or to correctly finish the run as needed for the ideal system. Figure 10 shows the resulting machines in the comparison of the real and ideal system.

For such a behavior, the simulator switches twice per real round, before and after a switching step of the ideal system. Formally, this would not need to be visible in the simulator definition: The simulator is a synchronous machine and can make one transition in each of its rounds. However, for ease of comparison with the real system, we draw the state-transition diagrams essentially with the global rounds. [5]

Definition 8 (Simulator for One Transaction) *The simulator for one transaction of labeled certified mail consists of submachines* cm_s′ *and/or* cm_r′ *(intuitively depending on whether only the sender, only the recipient, or both are honest),* cm_t′*, and* cm_v′*, and a dispatcher* dis*. The user ports of these machines are named* in_x'? *for* $x \in \{s, r\}$ *and* out_x'! *for* $x \in \{s, r, t, v\}$*. The network ports are named as in the real system.*

[5] Formally we use the clocking scheme (TH, Sim, A, H, A, Sim) for an ideal system with separate simulator. With the combination lemma of [47] (Lemma 2) the system where A and Sim are combined into an adversary A′ and clocked as (TH, A′, H, A′), i.e., the prescribed clocking for synchronous systems, is well-defined, and does not change the views of any (sub-)machines. Similarly, a scheme is defined where TH and Sim are combined into one machine M′ and clocked as (M′, A, H, A), and this also does not change any views. Hence we can use the latter for comparison with the real system.

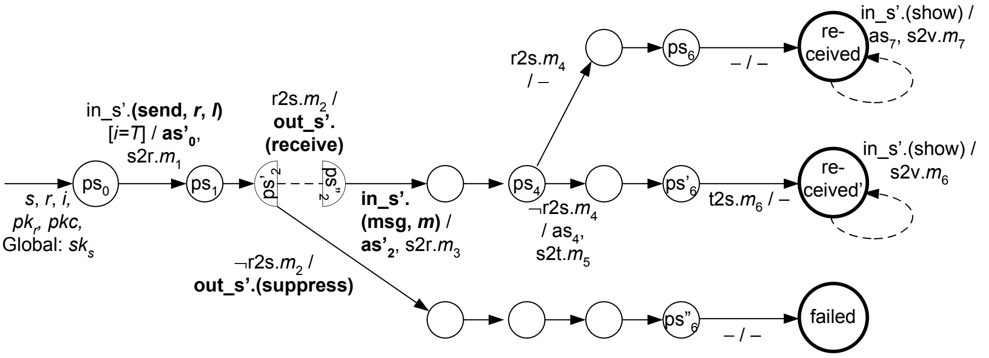

Fig. 11 Simulated sender transaction machine cm_s'. Differences to cm_s are denoted in bold face.

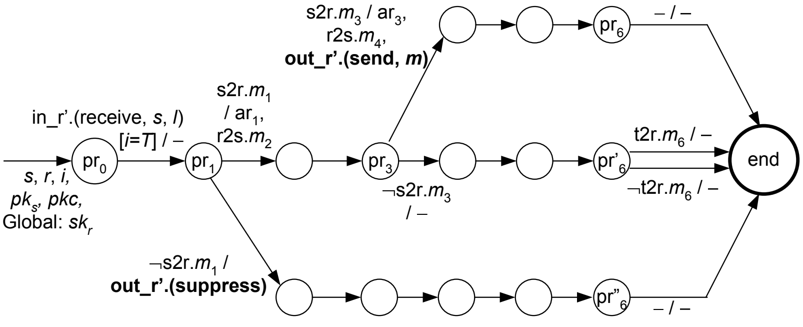

Fig. 12 Simulated recipient transaction machine cm_r'.

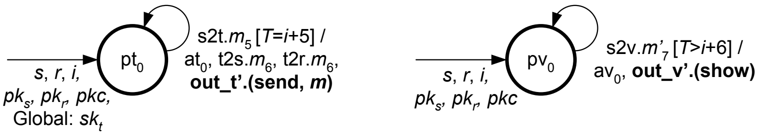

Fig. 13 Simulated transactions machines cm_t' and cm_v' for third party and verifier.

Input at $\mathsf{out_a}?$	to port	of
(send, r, l)	in_s'?	cm_s'
$(\mathsf{receive}, s, l)$	in_r'?	cm_r'
$(\mathsf{show}, (s, r, i))$	in_s'?	cm_s'
(msg, m)	in_s'?	cm_s'

Fig. 14 Dispatching of inputs from the ideal system in the simulator.

The state-transition diagrams of the main machines are defined in Figures 11 to 13. We draw two states of the simulator that envelop one switching step of the ideal system (and thus try to simulate one real switching step) as one circle unless a decision is made in the middle. Inputs at network ports are always expected in the first of these two states; then the machines make outputs at their user ports, and dis *dispatches them to* $\mathsf{in_a}!$. *Inputs at* $\mathsf{out_a}?$ *are expected in the second state, dispatched to the user inputs of the submachines as defined in Figure 14, and then the machines make outputs at the network ports. The global time* T *is now interpreted as an external global variable. Let* m_{sim} *be an arbitrary fixed message.*

Only cm_s' *uses new subprograms:*

In cm_s'.

Subprogram as'_0. *Let* $(c, r_{sim}) \leftarrow \mathsf{comr}_{pkc}(m_{sim})$ *and* $m_1 \leftarrow \mathsf{sign}_{sk_s}((s, r, i), \mathtt{m1}, l, c)$. *Generate a random string* r_{trans} *of sufficient length for a later call of* trans *for* c *with another message* m.[6]

Subprogram as'_2. *Transform the commitment with* $r_S \leftarrow \mathsf{trans}_{skc}(c, m_{sim}, r_{sim}, m)$, *using* r_{trans} *if random bits are needed in this process. Set* $m_3 := ((s, r, i), m, r_S)$.

This was a simulator for the original, weakly polynomial-time systems. If we make the real system strictly polynomial-time, we do the same for the simulator with the same global parameters, the same length functions for network inputs, and easily computable bounds for the inputs from the ideal system.

7 Correctness of the Simulation for One Transaction (Over Joint State)

We now want to show that the simulator for one transaction, i.e., one run of the subprotocol "send" with potentially many runs of "show" for the resulting receipt, simulates correctly. However, we want to do this in a modular way that later lets multiple transactions use common signature and commitment keys, as well as a common one-way function.

7.1 Methods to Prove Correct Simulation

To prove blackbox simulatability with the given simulator (recall Definition 7 and Figure 9), we show that arbitrary polynomial-time users and real adversaries cannot distinguish whether they are interacting with the real transaction or the ideal transaction plus simulator. This is essentially what is called observational equivalence in formal methods for distributed systems. The polynomial-time case was first defined in [38] for a π-calculus system model. Ignoring probabilistic and cryptographic aspects, there are essentially three systematic methods for such proofs when the product state space of each system is too large for ad-hoc treatment, as it is the case with a certified-mail transaction.

- Walkthroughs through both systems with all accepted inputs in each state, showing that they produce identical outputs at the same times. This corresponds to an exploration of the reachable space of corresponding states, typically depth-first. When automated, this approach is called model checking.
- Bisimulation. This means to define a general relation between "corresponding" states of the two systems, and to show that equal inputs in corresponding states lead to equal outputs and to corresponding states again. When automated, this is the typical approach with general-purpose theorem provers.
- Calculi. This means a successive transformation from one system definition into the other by a fixed set of provably secure transformation steps. This is a typical method with process algebras.

For this proof, we chose the walkthrough approach, because our systems for one transaction are relatively straight-line and thus a step-by-step argument seemed easiest for human provers and readers. In the outlook, Section 10.1, we discuss how one could partially automate such a proof, but why it would still be a significant challenge far beyond the scope of this paper.

7.2 Assumptions on the Use of Joint Primitives

In the walkthroughs we will encounter situations that depend on the cryptographic primitives, and sometimes on the correct overall use of those primitives with joint state. We

[6] This is only a technicality that simplifies the proof. It requires that Msg is polynomially bounded or the sufficient length is independent of m, as in the example commitment scheme in Section 4.2.

will later prove that the overall probability of so-called error sets where the observational equivalence is not guaranteed (e.g., due to a forged signature) is negligible even for many transactions. Besides the definitions and lemmas from Section 4.2 we will need one assumption about the joint use of signatures.

Definition 9 (Assumption on the Joint Use of Signatures) *When considering a certified-mail transaction with the transaction identifier (s, r, i) in the initial states of all machines in the real and simulated variant and using a global variable sk, we make the following assumptions.*

a) No other application uses sk to sign messages of the form $((s, r, i), \mathsf{m}x, ...)$ with $x \in 1, 2, 6$.
b) The key sk has been correctly generated as $(sk, pk) \leftarrow \mathsf{gen_S}(1^k, 1^{s^*})$ and no application uses it for anything other than signing.
c) Whenever the real transaction and the simulated one actually sign, then sk has the same value (even if the real transaction and the simulated one are not executed in identical environments) and less than s^* signatures have been made with sk overall so far.

We immediately see from the state-transition functions that each certified mail transaction fulfils Assumption a) of the others. Hence this assumption is automatically fulfilled if the many transactions of a certified-mail system are the only application that uses this key. This assumption generalizes to other types of transaction identifiers than (s, r, i) if each signed message is again a tuple starting with this transaction identifier.

7.3 The Walkthrough Proof

We summarized the machines that handle one transaction in the two systems in Figure 10. We call the real structure $struc$ and the simulated structure $struc'$. Recall that network channels involving the third party or the verifier are authentic and reliable, while those between senders and recipients are insecure. By the formal derivations of real structures from intended structures sketched in Section 5, this means that inputs supposedly from sender to recipient and vice versa are actually inputs from the adversary, and thus outer inputs for us, while the authentic channels only give additional outer outputs because the adversary can listen there. We want to show that these two structures essentially have the same outer input-output behavior by walkthroughs with all acceptable inputs in all reachable states.

Lemma 2 (Walkthrough Security of One Certified-Mail Transaction) *Let the two structures summarized in Figure 10 be given, where all machines are started in Round i with the same security parameter k and transaction identifier (s, r, i), the keys in the start states have been chosen correctly and consistently, the signature assumption (Definition 9) holds, and the ideal protocol length parameter is $\Delta = 6$. Then they have the same probabilistic input-output behavior and make the same use of the global signature keys for all input sequences except for a set of runs called* error sets *that we define within the proof. This holds both for the original weakly polynomial-time versions and the strictly polynomial-time parameterized versions.*

Each run in an error set corresponds in an obvious way to a situation where a cryptographic primitive has been broken. It will thus not be hard to show later that the error sets are negligible; however, because of the joint use of primitives, we have to postpone this proof to the consideration of multiple transactions.

Proof In the walkthroughs, corresponding machines $\mathsf{cm\_}x$ and $\mathsf{cm\_}x'$ will essentially be in the same state. The only exception is that the choice of the commitments is done differently in $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$. For this, we will define an exact mapping of corresponding random values that lead to the same future input-output behavior. Hence in all other places in the walkthroughs, we only have to remember the current state of the real machines and of the trusted host in the system with simulator. We distinguish the three cases from our trust model.

For the strictly polynomial-time case, note that the simulated and the real system have the same bounds on the length of accepted messages from A and H, and that the bounds for messages between the trusted host and the simulator were assumed to be chosen sufficiently large for the simple message format used there, so that messages from either of these machines will always be completely read by the other.

Case S: Correct Sender only. In this case, we have Figure 10 without cm_r and cm_r′, and with ths.

1. States reached without input send*:* The only round where cm_s and ths accept an input (send, r, l, m) at port in_s? is Round i. If they do not, both remain in their starting state forever and do not make any outputs. Hence cm_s′ does not get an input $(\mathsf{send}, \ldots)$ via dis and also remains in its starting state. As cm_t, cm_t′, cm_v, and cm_v′ only get inputs on authentic connections from cm_s and 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 in_s? of ths in Round i, it changes to State s_1 and outputs (send, r, l) at out_a!, which dis dispatches to cm_s′. Thus cm_s and cm_s′ send m_1. These messages only differ in the computation of the commitment c. (Here the signature assumption is used; we do not mention this again.) We therefore define a mapping of random values of cm_s′ to corresponding ones of cm_s as $\phi(r_{sim}, r_{trans}) := r_S \leftarrow \mathsf{trans}_{skc}(c, m_{sim}, r_{sim}, m)$, where r_{trans} is used if random bits are needed in this process; recall that cm_s′ chooses r_{trans} already in this round. By the chameleon property, the resulting commitments c are equal, and thus so are the messages m_1. Furthermore, the chameleon property implies that the resulting distribution of (c, r_S) equals that in cm_s.

No m_2. If the adversary does not respond with a correct m_2 (the test is equal for cm_s and cm_s′ in the given states), then cm_s outputs (failed) at out_s! in Round $i + 6$, while cm_s′ immediately sends (suppress) via dis to ths. Thus ths changes from State s_1 to s_2, and in Round $i + 6$ it outputs (failed) at out_s! as well. No network outputs are made in this process, and all three machines finish in State failed.

Correct m_2. If cm_s and cm_s′ receive a correct m_2, then cm_s sends $m_3 = ((s, r, i), m, r_S)$, while cm_s′ inputs (receive) to ths via dis. Thus ths changes from State s_1 to s_3 and reveals the payload message as (msg, m). Thus cm_s′ also sends m_3 still in Round $i + 2$, using the value $r_S \leftarrow \mathsf{trans}_{skc}(c, m_{sim}, r_{sim}, m)$, computed deterministically with r_{trans}. This is the same r_S as used by cm_s.

After sending m_3, the behavior of cm_s and cm_s′ with respect to the network is identical. If they send a message m_5, it arrives at cm_t and cm_t′ because these connections are authentic, and it passes the test by definition, so that cm_s and cm_s′ will obtain a correct m_6. Thus in Round $i + 6$, cm_s and cm_s′ either both enter the state received or both received′, and ths enters received and remains in this state and in showing. In this round, both cm_s and ths also output (sent) at out_s!.

This covers all states reachable and inputs accepted by cm_s, cm_s′, and ths as long as the only input at in_s? is $(\mathsf{send}, \ldots)$. As cm_t, cm_t′, cm_v, and cm_v′ only accept inputs on authentic connections from cm_s and cm_s′, respectively, the same holds for them.

3. Reactions on input show*:* Now we consider an input (show) in Round j at in_s?. Machine cm_s accepts this input if it is in State received or received′, while ths considers it in State received or showing. Above we showed that the machines enter these states under the same conditions in Round $i + 6$, and that cm_s′ is then in the same state as cm_s. Now ths goes into State showing and outputs (show), which dis dispatches to cm_s′. Hence 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 ths does because it is in State $\mathsf{showing}$. In addition, $\mathsf{cm\_v}'$ outputs (show), which is dispatched to ths, but ths is in State $\mathsf{showing}$ and ignores it.

Case R: Correct Recipient Only. As the walkthrough proofs are tedious and follow the same pattern, we postpone the remaining two to the appendix. Here we only summarize the situations where we put runs into error sets. (We do not omit the walkthroughs entirely because they are the part of the proof that is unique to certified mail, and they show that the following situations are really the only ones where the real and the simulated systems deviate.)

- If no input $(\mathsf{receive}, s, l)$ is made to $\mathsf{cm\_r}$ and thr (always in the correct round), or $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ do not obtain a correct message m_1, and thus they do not send m_2, and the adversary nevertheless inputs a correct message m_5 to $\mathsf{cm\_t}$ and $\mathsf{cm\_t}'$ later, or m_7' to $\mathsf{cm\_v}$ and $\mathsf{cm\_v}'$, this must contain a correct message m_2, i.e., $\mathsf{test}_{pk_r}(m_2) = ((s,r,i), \mathsf{m2}, \ldots)$. Then we let the run be in the error set Forge_k with the forged signature $\mathit{sig}_f := m_2$ and attacked participant index $u := r$. Recall that by Assumption 9 no other application signs a message starting $((s,r,i), \mathsf{m2}, \ldots)$ with sk_r and that tuple decomposition is unambiguous, i.e., this is indeed a forgery.
- Similarly, if $\mathsf{cm\_t}$ and $\mathsf{cm\_t}'$ do not obtain a correct message m_5, but $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ or $\mathsf{cm\_v}$ and $\mathsf{cm\_v}'$ obtain a correct m_6, then $\mathit{sig}_f := m_6$ is signature forgery for a message starting $((s,r,i), \mathsf{m6}, \ldots)$, and we put the run in the error set Forge_k with attacked participant $u := t$.
- If $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ sent m_2 and the adversary later shows a receipt m_7' containing a different m_2'', we put the run in the error set Forge_k with $\mathit{sig}_f := m_2''$ and $u := r$.
- If $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ sent m_2 but not m_4, and the adversary can nevertheless show a receipt of the form m_7, it must contain a one-way preimage r_R' with $\mathsf{f}(r_R') = p_R$ for the one-time public key p_R from m_2, while $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ never used the original preimage r_R except in the assignment $p_R \leftarrow \mathsf{f}(r_R)$. This contradicts the one-way property of f. We put the run in the error set $\mathit{OwfBreak}_k$.
- If $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ obtained m_3 with a payload m and the adversary can later show a receipt m_7' for a different payload m', then the receipt must contain a message m_3' where the same commitment c is opened in a different way $\mathsf{com}_{pkc}(m', r_S') = c = \mathsf{com}_{pkc}(m, r_S)$. This contradicts the binding property of the commitments. We put the run in the error set $\mathit{BindBreak}_k$ and set $\mathit{bindbreak} := (m, r_S, m', r_S')$.

Case SR: Correct Sender and Recipient. As in Case R, we only summarize the situations where we put runs into error sets here.

- Similar to Case R, we encounter cases where the adversary forges a signature sig_f on a message with identifier $\mathsf{m1}$ with sk_s, or on a message with identifier $\mathsf{m2}$ with sk_r. We put those runs in the error sets Forge_k with $u := s$ or r, respectively.
- If $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ send m_1, but do not obtain a correct m_2 via the adversary, and the adversary nevertheless inputs an acceptable m_3 to $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$, then this contains values that open the commitment c from m_1, i.e., values (m^a, r_S^a) with $c = \mathsf{com}_{pkc}(m^a, r_S^a)$. However, $\mathsf{cm\_s}$ did not use the value r_S from the assignment $(c, r_S) \leftarrow \mathsf{comr}_{pkc}(m)$ anywhere else. This situation therefore contradicts Lemma 1. We put the run in the error set $\mathit{ComOwfBreak}_k$.
- If the adversary forwards correct messages m_1 and m_2, but no acceptable m_3, and nevertheless inputs an acceptable m_4 to $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$, then this contains a one-way preimage r_R' with $\mathsf{f}(r_R') = p_R$ for the one-time public key p_R from m_2. However, $\mathsf{cm\_r}$ and $\mathsf{cm\_r}'$ never used the original r_R except in the assignment $p_R \leftarrow \mathsf{f}(r_R)$. Hence we put the run in the error set $\mathit{OwfBreak}_k$.

- If the adversary forwards correct messages m_1 and m_2, and an acceptable but incorrect third message $m_3^a = ((s, r, i), m^a, r_S^a)$ with $m^a \neq m$ for the payload m used by $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$, then $\mathsf{com}_{pkc}(m^a, r_S^a) = \mathsf{com}_{pkc}(m, r_S) = c$ for the commitment used by $\mathsf{cm\_s}$. We put the run in the error set $BindBreak_k$ and set $bindbreak := (m, r_S, m^a, r_S^a)$.

8 Multi-Transaction Systems

In a real certified-mail system it must be possible to execute many certified-mail transactions. We first present options for constructing the overall system. We then show how our security proof for one transaction carries over to a multi-transaction system in spite of the joint use of primitives by these transactions.

8.1 Multi-tasking Options

There are multiple options for executing multiple transactions of certified mail, reflecting programming styles known from real distributed systems with multiple transactions (also called sessions, tasks, or threads).

Network dispatching. In the real system, multiple transactions of multiple types of protocols are running in the same overall machine with a fixed number of physical connections. Hence network inputs must be dispatched to the correct protocol and transaction. Three standard solutions are:

- Explicit dispatcher. The messages arrive on one connection and a specific program, called dispatcher, distributes them to the transactions that expect them by some transaction identifier. The transaction identifier may come from a flat domain, e.g., be chosen randomly or assigned by the dispatcher, or be hierarchic, i.e., starting with a protocol identifier like “certified mail”.
- Dispatcher in the network layer. A dispatcher may already be inherent in the underlying network stack in the ability to set up multiple network sessions, one for each transaction.
- Token-based communication. The network outputs of a transaction may be given to the caller of the transaction for transport. This can minimize the communication overhead as only top-level transactions communicate.

Another distinction with dispatcher solutions is whether a dispatcher starts a new transaction if a message with an unregistered transaction identifier arrives, or whether a local caller must generate the transaction first.

User in- and outputs. The different transactions in one machine need inputs from their users and make outputs to these users. Typically the user is another protocol, but some transactions have an interface to people. This is essentially the same question as network dispatching, only seen from the other side.

- Explicit dispatcher. The users make the inputs into a fixed “port” (e.g., they call a fixed library) which internally handles multiple transactions.
- Visible multi-tasking. Each user once calls a fixed port to generate a new transaction locally and then interacts directly with this transaction. In practice, the transaction may come from a pool instead of being newly generated.

Multi-tasking or not. In real systems, multi-tasking or multi-threading is often too inefficient, and a high-level design with transactions is realized by a low-level design without, typically with global state and function tables. It may seem uninteresting at this point whether this happens with a real implementation of certified mail (although in principle the correctness of the transformation must be proved), but it may even concern ideal systems: While for a higher-level transaction that uses only one certified-mail transaction it may be best to consider just this transaction, an application using many lower-layer transactions may best represent them all as one machine also in some automated proof tools.

We believe that all these variants can be formalized without problems for cryptographic protocols. In the original report [46] we chose a scheme with an explicit dispatcher for network in- and outputs as well as for user in- and outputs. Further, multi-tasking is partially realized with global state, as there are global variables and the dispatchers are not explicitly clocked as separate machines. However, these decisions are not inherent in the design of the individual transaction machines. (The only exception is the treatment of signature keys as global variables, but these could be replaced by submachines, in particular the signer machines from Definition 2 with an additional dispatcher.) Hence the same transaction machines could be used in any other multi-tasking scheme. Similarly, the multi-tasking scheme from [46] could essentially be used for other transactions than certified mail, i.e., where the concrete in- and outputs of certified mail were dispatched according to the concrete transaction identifier (s, r, i), arbitrary inputs and outputs could be dispatched according to arbitrary transaction identifiers.

As the dispatching is tedious and has been defined and proved in full detail in [46], we omit it here. Essentially, we defined an overall trusted host TH as well as overall real machine types Ms, Mr, Mt, and Mv. Each real machine has at most one input and one output port for its user, while the trusted host has all these ports together. Per round, a machine accepts a list of inputs at each input port and dispatches them to the appropriate transaction machines. The list elements carry the full transaction identifiers (here (s, r, i)), while we usually omitted them in in- and outputs of the transaction machines, e.g., (failed) and (show).[7] Similarly, each pair of overall machines has one network connection (insecure for senders and recipients, reliable and authentic for third parties and verifiers) and dispatches the network messages of its submachines over those. An additional broadcast machine is present in the first two rounds to distribute public keys consistently. The trust model is that any number of senders and recipients may be statically corrupted, while the third party, verifier, and broadcast machine are correct. Let $Sys_{\mathsf{CM,id}}$ and $Sys_{\mathsf{CM,real}}$ denote the resulting multi-transaction real and ideal system for a certain parameter set, i.e., participant numbers, label and message set, and specific cryptographic primitives used in the real system. Then our multi-transaction security theorem is (where corresponding structures of the two systems are defined in the canonical way via equal service port sets):

Theorem 1 *The real multi-transaction certified-mail system $Sys_{\mathsf{CM,real}}$ is as secure as the ideal certified-mail system $Sys_{\mathsf{CM,id}}$ with blackbox simulatability,*

$$Sys_{\mathsf{CM,real}} \geq_{\mathsf{sec}}^{\mathsf{poly,bb}} Sys_{\mathsf{CM,id}}.$$

One reason why we chose this particular multi-tasking scheme was that it avoids the explicit generation of new machines. This was not defined for Turing machines in 2000 (and thus neither for our state-transition machines with Turing machine realizations), and to our knowledge it is still not defined in 2004 even though many theorems and constructions implicitly assume it. We do not doubt that it can be done, in particular as similar constructions exist for π-calculus with polynomial-time restrictions [41]. Nevertheless, there may be subtleties in the overall polynomiality of such systems, as well as tedious details for defining how newly generated machines are connected (e.g., we cannot simply allow someone to generate a machine that connects to someone else's signature machine, while it must connect to the local signature machine). Hence we did not want to assume this.

[7] Where we explicitly kept them, as in $\mathsf{out\_v}!.(\mathsf{received}, (s, r, i), m)$ they are needed for their real content, i.e., another transaction identifier tid would augment (s, r, i) here, not replace it.

8.2 Proof of Multi-Transaction Composition

Independent of how we connect them, we have multiple transactions running and have to show that the overall real system is as secure as the overall ideal system. Recall that even now in 2004 we cannot avoid proving this composition specifically for certified mail as we use joint state over primitives that do not have similar transaction-wise abstractions yet. The proof is nevertheless relatively generic, in particular in its treatment of signatures. As we have omitted the precise definition of a dispatching scheme here, we prove this for a scheme where all the transaction machines are simply run in parallel, i.e., they interact directly with the honest users and the adversary. Formally, for this we rename the ports of each machine by indexing them with the transaction identifier, and we assume that each machine is started with the correct initial inputs. Port renaming is well-defined in [47] and does not change the behavior. Assuming a definition of machines whose number of ports (or Turing tapes) depends on the security parameter (here for the machines A and H representing the overall adversary and honest users), the proof also holds for polynomially many transactions. More precisely, the numbers of participants and of overall rounds could be polynomial in the security parameter k, and so could, with more general transaction identifiers, the number of transactions per participant pair started per round.

8.2.1 Overview. Our walkthrough lemma (Lemma 2) together with our definitions of the error sets would immediately lead to a security proof for one transaction. To use the lemma in this multi-transaction setting, we have to show that indeed every transaction gets the same external inputs in the real and the simulated setting (even though now other transactions are running in parallel) and that the signature assumption (Definition 9) is always fulfilled. For this, we show that the walkthrough proofs of all transactions can be joined into one proof. Let an adversary A and honest users H for the multi-transaction systems be given.

- We define corresponding reachable states of the multi-transaction real and simulated structures as states where the global keys are equal, and all transaction substructures are in corresponding states. (We did not need a name for this correspondence above, but its restriction to the random values that are different in the real and simulated sender machines was called ϕ.)
- We define each overall error set as the set of overall runs where at least one run of a transaction (easily defined as a restriction of the overall run) belongs to the corresponding transaction error set.
- Simple induction over the round number shows that in all overall runs that do not belong to an error set, the multi-transaction systems are always in corresponding states, and all transactions produce identical external inputs and outputs and make equal use of the global signature keys.

Hence the views of A and H are identical in all runs of the two multi-transaction structures except for the error sets. It is therefore sufficient for the desired computational indistinguishability of the views of H to show that the overall error sets are negligible.

8.2.2 Proof that the Overall Error Sets are Negligible The proof of Lemma 2 defined four types of error sets: $Forge_k$ for runs where a signature of an honest participant u is forged, $OwfBreak_k$ for runs where an unknown one-way preimage is found, $BindBreak_k$ for runs where a commitment is opened in two ways, and $ComOwfBreak_k$ for runs where the adversary can open a commitment of an honest user. In slight abuse of notation we use the same error set names for runs of the multi-transaction system. The index k denotes the security parameter, i.e., we actually have four sequences of error sets. As the finite sum of negligible functions is again negligible, we only need to show that each of these sequences has negligible probability. The following reduction proofs essentially yield the concrete complexity of the reductions. We therefore assume that an adversary and honest users A

and H are given that start at most ω transactions of certified mail in at most ρ rounds, for a system with at most n honest participants, use overall runtime τ together with the correct machines, and achieve that an error set has probability ϵ for a specific k.

Signature Forgery. Assume that the probability of Forge_k is ϵ. We construct an adversary $\mathsf{A}_{\mathsf{sig}}$ against the signer machine Sig_{s^*} from Definition 2 with $s^* := \omega$. It randomly chooses a participant v to attack among the at most n participants. It simulates the real configuration using the public key pk obtained from Sig_{s^*} as pk_v. I.e., it executes all machines of this configuration with security parameter k for at most ρ rounds and with random values chosen in the execution as usual, except that it sends every message m_j to be signed with sk_v to Sig_{s^*} instead and uses the result as the signature. In addition, it keeps track of the conditions for putting the resulting run in the error set Forge_k with the attacked participant $u = v$. Such a u was defined for each run in Forge_k in the proof of Lemma 2, and the conditions can be verified efficiently. (Formally, we have put joint runs of both systems in the error sets, but it is easy to see that all error set conditions can be verified on the real system alone.)

If such a condition is fulfilled, $\mathsf{A}_{\mathsf{sig}}$ outputs the designated value sig_f (again from the proof of Lemma 2) as its forged signature. In each case, it was already shown in the walkthrough that sig_f is a valid signature for pk_u and that the contained message m_f was not signed by the given transaction machine or any other simulated machine, i.e., $\mathsf{A}_{\mathsf{sig}}$ did not ask Sig_{s^*} to sign m_f.

As v was chosen randomly, the success probability of $\mathsf{A}_{\mathsf{sig}}$ is at least ϵ/n. The runtime of $\mathsf{A}_{\mathsf{sig}}$ is essentially τ and it uses at most ω calls to the signature machine, as each machine signs at most once per transaction.

Finding One-way Preimages. Assume that the probability of $\mathit{OwfBreak}_k$ is ϵ. We construct an adversary $\mathsf{A}_{\mathsf{owf}}$ as in Definition 3. It randomly chooses one of the at most ω possible transactions, e.g., assuming they are sorted according to starting round and, within the round, lexicographically by transaction identifier. It simulates the real configuration using the given one-way image p as p_R in the machine $\mathsf{cm\_r}$ for this transaction instead of setting $p_R := \mathsf{f}(r_R)$ for random r_R. It checks whether the resulting run belongs to $\mathit{OwfBreak}_k$ for the chosen transaction. If yes, a value r'_R with $\mathsf{f}(r'_R) = p_R = p$ is obtained, and $\mathsf{A}_{\mathsf{owf}}$ outputs it. It was already shown in the walkthroughs that the (now unknown) value r_R was not used outside the replaced assignment $p_R := \mathsf{f}(r_R)$ up to this point; hence the simulation is possible.

Hence the success probability of $\mathsf{A}_{\mathsf{owf}}$ is at least ϵ/ω, and its runtime is essentially τ.

Binding Property of the Commitments. Assume that the probability of $\mathit{BindBreak}_k$ is ϵ. We construct an adversary $\mathsf{A}_{\mathsf{bind}}$ as in Definition 4a. Given a public commitment key pkc, it simulates the real configuration using this commitment key. Note that the corresponding secret key is never used in the real system (only in the simulator); hence this simulation is possible. If yes, it outputs the designated tuple $\mathit{bindbreak}$, for which we have already shown that it fulfils the condition from Definition 4a.

Hence the success probability of $\mathsf{A}_{\mathsf{bind}}$ is at least ϵ, and its runtime is essentially τ.

One-way Property of the Commitments. Assume that the probability of $\mathit{ComOwfBreak}_k$ is ϵ. We construct adversary algorithms A_1, A_2 as in Lemma 1b. The first algorithm A_1 randomly chooses one of the at most ω possible transactions as in the second case. It then starts simulating the real configuration using the given public commitment key pkc. This is possible as in the previous case. When the simulated H starts the selected transaction with a payload message m, 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 message m_1 for this transaction, instead of choosing it as $(c, r_S) \leftarrow \mathsf{comr}_{pkc}(m)$. If the condition for putting the run in $\mathit{ComOwfBreak}_k$ for this transaction is fulfilled, then A_2 outputs the designated values (m^a, r_S^a) that open the commitment c. We have already shown that they fulfill the condition from Lemma 1b and that r_S (now unknown) is not used elsewhere, so that the simulation is possible.

Hence the overall success probability of A_1 and A_2 is at least ϵ/ω, and their runtime is essentially τ.

Summary of the Reductions. The four concrete-complexity reductions that we made for individual values of k immediately imply that if there were an overall polynomial-time adversary and users successful against an overall polynomial-time variant of the real certified-mail system, then one could break one of the underlying cryptographic primitives. With a bit more effort, one sees the same for the system with weakly polynomial-time machines: It is sufficient to show that the combination of all real machines is weakly polynomial-time. This holds because every external input only results in a polynomial amount of internal activity, as can be seen by inspection of Definition 5. This finishes the security proof of the generic multi-transaction versions of our certified-mail system.

A security proof for a system with a specific dispatching scheme can be derived by additionally comparing the real and simulated dispatching schemes. A detailed proof for the specific dispatching scheme sketched at the end of Section 8.1 can be found in [46]. Altogether this finishes the proof of Theorem 1.

9 Symbolically Proving Properties of Certified Mail

One reason to define the security of real systems by abstract ideal systems without probabilism and cryptographic objects is as a link to formal methods, in particular to automated proof tools. Such tools can be used for at least three purposes in this context:

- To prove individual properties of the system in question, e.g., the unforgeability of receipts in certified mail.
- To prove that a higher protocol using the system in question is as secure as some ideal system for the higher protocol. For instance, one might prove that a multi-party contract signing protocol based on certified mail is as secure as an ideal multi-party contract signing system.
- To directly prove individual properties of a higher protocol using the system in question. For instance, one might directly prove that a multi-party contract signing protocol based on certified mail provides unforgeability and fairness.

For the first and third possibility one needs preservation theorems for the desired properties with respect to reactive simulatability, for the second and third possibility a composition theorem. Both are available in the underlying framework from [47]. The last possibility can usually be replaced by the first two, but may be quicker if one is only interested in certain properties.

This section sketches the earliest example of the first possibility by proving integrity properties of certified mail. This corresponds to abstracted versions of properties that the classic cryptographic literature used to define certified mail.

An *integrity requirement* for a set S of service ports is a set Req of finite traces of events at these ports. Intuitively it describes the set of allowed event sequences. Recall that traces of events at any set of ports are well-defined for runs in the underlying system model. Hence we can say that a run fulfils an integrity requirement or not (the resulting trace at S lies in Req or not). Now [47] defines that a structure fulfils an integrity requirement Req perfectly if all its runs with all users and adversaries fulfil Req, and computationally if for all polynomial-time users and adversaries, the probability that Req is not fulfilled is negligible in the security parameter k. The definition is lifted from structures to systems (sets of structures) in the canonical way. The preservation theorem states that fulfilling an integrity requirement is preserved under reactive simulatability. Hence if one proves an integrity requirement for the ideal certified-mail system, it also holds computationally for the real system.

We now present integrity requirements on one transaction of certified mail. For finite traces, "after at most Δ rounds" is defined to be automatically fulfilled if less than Δ further rounds exist.

Definition 10 (Integrity Requirements on Labeled Certified Mail) *The integrity requirements on one transaction of certified mail, with transaction identifier* (s, r, i)*, for the three cases of the trust model for one transaction, are the sets* Req_i *defined by the following formulas, where* $i \in \{1, \ldots, 7\}$ *corresponds to the item labels.*

For correct senders (with correct or incorrect recipient).

1. Verifiability of Valid Receipts. *If an output* (sent) *occurs at* $\mathsf{out\_s}!$ *after an input* (send, r, l, m) *at* $\mathsf{in\_s}?$ *in Round* i*, then a later input* (show) *at* $\mathsf{in\_s}?$ *leads to the output* $(\mathsf{received}, l, m)$ *at* $\mathsf{out\_v}!$ *within one round.*
2. Termination for Sender. *An input* (send, r, l, m) *at* $\mathsf{in\_s}?$ *in Round* i *leads to an output* (sent) *or* (failed) *at* $\mathsf{out\_s}!$ *after at most* Δ *rounds, and no second such output occurs at* $\mathsf{out\_s}!$.

For correct recipients (with correct or incorrect sender).

3. Unforgeable Receipts. *If an output* $(\mathsf{received}, (s, r, i), l, m)$ *occurs at* $\mathsf{out\_v}!$ *in a round* j*, then* $i \leq j$*, and an input* $(\mathsf{receive}, s, l)$ *occurred at* $\mathsf{in\_r}?$ *in Round* i*.*
4. No Surprises for the Recipient. *If an output* (failed) *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.*
5. Fixed Receipts. *If an output* $(\mathsf{received}, m)$ *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.*
6. Termination for Recipient. *An input* $(\mathsf{receive}, s, l)$ *at* $\mathsf{in\_r}?$ *in Round* i *leads to an output* $(\mathsf{received}, m)$ *or* (failed) *at* $\mathsf{out\_r}!$ *after at most* Δ *rounds, and no second output of these types occurs at* $\mathsf{out\_r}!$.

For correct sender and recipient.

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

For the parameterized version of certified mail, we assume input typing in these formulas, i.e., inputs called m and l must be in the message and label space, respectively, and outputs with these names will be in the same sets.

By Theorem 3.2 of [47], logical derivations from integrity requirements are valid also for computational fulfillment. Hence one can draw conclusions on this abstract level, or, e.g., join some of the requirements for correct recipients into one. The current formulation is standard predicate logic using round numbers. If one is interested in a simpler fragment of logic, most of the requirements can be expressed in temporal logic; only requirements mentioning "after at most Δ rounds" are a bit awkward.

The following theorem validates the ideal system for one transaction with respect to these requirements.

Theorem 2 (Integrity of the Ideal Certified-Mail System for One Transaction) *The ideal system from Definition 1 fulfills Definition 10. More precisely, the trusted host* thsr *fulfils all requirements, while* ths *fulfils Requirements 1, 2, and 7, and* thr *fulfils Requirements 3 to 5.*

Consequently, also the real system from Definition 5 fulfils these requirements.

Proof All requirements can easily be verified by inspection of the state-transition diagrams.

The original report [46] formulates the requirements for the multi-transaction system and includes the dispatching in the proof. Further, it contains a secrecy property, but that is not proved via a preservation theorem, and an availability-style property stating that correct senders and recipients can successfully exchange a mail if the channels are authentic during this transaction. In the ideal system this corresponds to the absence of the input suppress for this transaction.

10 Outlook

10.1 Automating the Walkthrough Proof

Without cryptographic and probabilistic aspects, a proof of equal input-output behavior for one transaction of the given complexity would seem automatable with current techniques. However, already with the probabilism, and even more with the cryptography, there was no chance of automation in 2000 when we initially wrote this paper, and it would still be a significant challenge now. (In contrast, we believe that automating proofs of distributed protocols *using* the certified-mail system is very feasible, because the ideal system is free from probabilism and cryptographic objects, and proofs automatically carry over to the real system via composition and property preservation theorems.) In 2000, there was no automated proof over real cryptography at all. At present, the largest such proofs are the security of Diffie-Hellman encryption against passive adversaries, given the Diffie-Hellman decision assumption and special-purpose formalizations of the underlying algebra, and of construction for stretching the output of a pseudorandom generator [50, 36]. None of these proofs contains real distributed aspects yet.

We already mentioned in the outlook of [46] that we expected follow-up work that would abstract from individual primitives in order to allow automation of parts of this proof using standard tools, i.e., without probabilistic and cryptographic aspects. However, there is still no suitable abstraction from one-way functions and commitments for our case where special usage restrictions allow security in the standard model of cryptography. Early ideal signature systems as in [23] are not abstract in the sense expected by standard proof tools. Further, we use nested signatures (the message signed in m_2 contains m_1 and thus already a signature). Hence neither the temporal-logic abstraction from [43] nor a simple ideal signature system by a database of signed statements is applicable, and thus the Dolev-Yao style library from [13] would be needed. It would seem possible, however, to define a similar certified-mail protocol without nested signatures. All this seems feasible now, but far beyond the scope of the present paper.

10.2 Joint Use of Signatures

In the assumption on the joint use of a secret signature key, Definition 9, the use of different transaction identifiers in a fixed position of the signed messages corresponds to the same general principle as what we used to separate the certified-mail protocols. Among the known ways to assign such transaction identifiers, we used a hierarchical scheme in this place (the transaction identifier of the certified-mail transaction plus a message identifier). If one idealizes from the signature scheme first, in particular as a step towards automating the walkthrough proof (a hand-made proof based on an idealization would not be significantly simpler than the current proof) this assumption is still needed, while Assumption b) becomes part of the idealization. This holds both for idealizations by temporal logic and by ideal systems.

10.3 General Joint State Composition

The type of dispatching sketched in Section 8.1 and fully defined in [46] was also later used by Canetti and Rabin for their joint-state composition theorem [25]. Their protocol ρ corresponds to our real transaction machines, their protocol $\hat{\rho}$ to our multi-transaction real machines with dispatching, their functionality $\mathcal{F}$ to our ideal transaction machines, and their functionality $\hat{\mathcal{F}}$ to our multi-transaction ideal system with dispatching. The precondition of their theorem is that simulatability holds for the multi-transaction systems, i.e., $\hat{\rho}$ is as secure as $\hat{\mathcal{F}}$. I.e., they simply assume that a hand-proof of the joint-state aspects of the real system is made somehow. The theorem then states that a protocol π using the ideal system $\hat{\mathcal{F}}$ can be rewritten by pulling the user input-output dispatching up into π.

This would be a similarly simple consequence of our dispatching results but we did not state it in [46]. Thus the addition made by [25] is to formally state that higher protocols that use a system defined by transaction machines and dispatching like here can assume that they interact with individual transaction machines, as the overall approach suggests.

10.4 Symbolic Proofs over Ideal Systems

Section 9 on integrity properties of certified mail was only the first step into an overall program for linking formal methods and cryptography. The first tool-supported proof over an abstraction from cryptography that was justified by a simulatability result was made in [10]. It proves integrity properties for a given ideal system with the theorem prover PVS and thus corresponds to the first possibility from Section 9. The second possibility, a simulatability proof for a protocol that uses an ideal subsystem, was first explored in [9]. Important examples of the third possibility, symbolic proofs of properties of a protocol using an ideal subsystem, but by hand, are the proofs of the Needham-Schroeder-Lowe and Otway-Rees protocols over an ideal Dolev-Yao-style cryptographic library in [13, 6].

11 Conclusion

We have proven the security of an efficient certified-mail system with respect to an ideal system in the sense of reactive simulatability. The ideal system is abstract enough to be suitable for use in automated proof tools for larger systems; in particular it is deterministic and does not contain cryptographic objects.

The certified-mail system is optimistic and labeled. This means that a trusted third party is needed only in case of a dispute, and the recipient agrees to a mail subject in advance. The communication model is synchronous; we have discussed pros and cons of synchronous protocols. The concrete protocol needs four rounds in the optimistic case, which is optimal. Security holds in the standard model of cryptography, under standard assumptions, and with blackbox simulatability.

Reactive simulatability implies general composability. Apart from its value for certified mail, this paper (when first written) provided the first convincing evidence that a general reactive simulatability definition with a composition theorem is a useful basis for specifying and proving the security of practical reactive systems with multi-round transactions and multiple related transactions, here the sending of mail and showing of receipts. It further showed how simulatability is possible in the standard model of cryptography in spite of the (restricted) use of committing primitives.

We also introduced the main techniques that later became known as joint-state composition, i.e., the composition of a system for multiple transactions from systems for individual transactions that share keys of cryptographic primitives.

Finally, we have shown how properties of the ideal certified-mail system can be derived symbolically and automatically hold for the real system. This was one step in a program to link cryptographic systems and formal methods that was significantly extended since.

Acknowledgments

We thank *Ran Canetti*, *Victor Shoup* and *Michael Steiner* for interesting discussions, and an anonymous reviewer for detailed comments. Details of weakly versus strictly polynomial-time were first clarified with *Michael Backes* and Michael Steiner in the context of their thesises, although this was finally not mentioned there.

Major parts of this work were written while the first two authors were with Universität des Saarlandes, Saarbrücken. This work was supported by the European IST Project MAFTIA. However, it represents the view of the authors. The MAFTIA project was funded by the European Commission and the Swiss Department for Education and Science.

References

1. Ross Anderson, Roger Needham: Robustness Principles for Public Key Protocols; Crypto ’95, LNCS 963, Springer-Verlag, Berlin 1995, 236–247.
2. N. Asokan, Birgit Baum-Waidner, Matthias Schunter, Michael Waidner: Optimistic Synchronous Multi-Party Contract Signing; IBM Research Report RZ 3089 (#93135) 12/14/1998, IBM Research Division, Zurich, Dec. 1998.
3. N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Fair Exchange; 4th Conference on Computer and Communications Security (CCS), ACM, 1997, 6–17.
4. N. Asokan, Victor Shoup, Michael Waidner: Optimistic Fair Exchange of Digital Signatures; IEEE Journal on Selected Areas in Communications 18/4 (2000) 593–610.
5. Michael Backes: Unifying Simulatability Definitions in Cryptographic Systems under Different Timing Assumptions; 14th International Conference on Concurrency Theory (CONCUR), LNCS 2761, Springer-Verlag, Berlin 2003, 350–365.
6. Michael Backes: A Cryptographically Sound Dolev-Yao Style Security Proof of the Otway-Rees Protocol; to appear in 9th European Symposium on Research in Computer Security (ESORICS 2004), LNCS, Springer-Verlag.
7. Feng Bao, Robert Deng, Wenbo Mao: Efficient and Practical Fair Exchange Protocols with Off-Line TTP; 1998 IEEE Symposium on Research in Security and Privacy, 77–85.
8. Michael Backes, Dennis Hofheinz: How to Break and Repair a Universally Composable Signature Functionality; to appear in 7th Information Security Conference (ISC), LNCS, Springer-Verlag, 2004.
9. Michael Backes, Christian Jacobi: Cryptographically Sound and Machine-Assisted Verification of Security Protocols; 20th International Symposium on Theoretical Aspects of Computer Science (STACS), LNCS 2607, Springer-Verlag, Berlin 2003, 675–686.
10. Michael Backes, Christian Jacobi, Birgit Pfitzmann: Deriving Cryptographically Sound Implementations Using Composition and Formally Verified Bisimulation; Formal Methods Europe, LNCS, Springer-Verlag, Berlin 2002, 310–329.
11. Michael Backes, Birgit Pfitzmann: Symmetric Encryption in a Simulatable Dolev-Yao Style Cryptographic Library; 17th IEEE Computer Security Foundations Workshop (CSFW), 2004, 204–218.
12. Michael Backes, Birgit Pfitzmann, Michael Steiner, Michael Waidner: Polynomial Fairness and Liveness; 15th IEEE Computer Security Foundations Workshop (CSFW), 2002, 160–174.
13. Michael Backes, Birgit Pfitzmann, Michael Waidner: A Composable Cryptographic Library with Nested Operations; 10th Conference on Computer and Communications Security (CCS), ACM, 2003, 220–230.
14. Michael Backes, Birgit Pfitzmann, Michael Waidner: Secure Asynchronous Reactive Systems; Cryptology ePrint Archive, Report 2004/082, http://eprint.iacr.org/, March 2004
15. Donald Beaver: Secure Multiparty Protocols and Zero Knowledge Proof Systems Tolerating a Faulty Minority; Journal of Cryptology 4/2 (1991) 75–122.
16. Mihir Bellare, Phillip Rogaway: Entity Authentication and Key Distribution; Crypto ’93, LNCS 773, Springer-Verlag, Berlin 1994, 232–249.
17. 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.
18. Manuel Blum: How to Exchange (Secret) Keys; ACM Transactions on Computer Systems 1/2 (1983) 175–193.

19. Dan Boneh, Moni Noar: Timed Commitments and Timed Signatures; Crypto 2000, LNCS 1880, Springer-Verlag, Berlin 2000, 236–254.
20. Jurjen Bos, David Chaum, George Purdy: A Voting Scheme; unpublished manuscript, presented at the rump session of Crypto ’88.
21. Gilles Brassard, David Chaum, Claude Crépeau: Minimum Disclosure Proofs of Knowledge; Journal of Computer and System Sciences 37 (1988) 156–189.
22. Ran Canetti: Security and Composition of Multiparty Cryptographic Protocols; Journal of Cryptology 13/1 (2000) 143–202.
23. Ran Canetti: Universally Composable Security: A New Paradigm for Cryptographic Protocols; 42nd Symposium on Foundations of Computer Science (FOCS), IEEE, 2001, 136–145.
24. Ran Canetti: Universally Composable Signature, Certification, and Authentication; 17th IEEE Computer Security Foundations Workshop (CSFW), 2004, 219–234.
25. Ran Canetti, Tal Rabin: Universal Composition with Joint State; Crypto 2003, LNCS 2729, Springer-Verlag, Berlin 2003, 265–281.
26. 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.
27. Ivan Bjerre Damgård: Collision Free Hash Functions and Public Key Signature Schemes; Eurocrypt ’87, LNCS 304, Springer-Verlag, Berlin 1988, 203–216.
28. Robert H. Deng, Li Gong, Aurel A. Lazar, Weiguo Wang: Practical Protocols for Certified Electronic Mail; Journal of Network and Systems Management 4/3 (1996) 279–297.
29. Rosario Gennaro, Silvio Micali: Verifiable Secret Sharing as Secure Computation; Eurocrypt ’95, LNCS 921, Springer-Verlag, Berlin 1995, 168–182.
30. Oded Goldreich, Hugo Krawczyk: On the Composition of Zero-Knowledge Proof Systems; SIAM Journal on Computing 25/1 (1996) 169–192.
31. Oded Goldreich: Sending Certified Mail using Oblivious Transfer and a Threshold Scheme; Technion - Israel Institute of Technology, Computer Science Department, Technical Report, 1984.
32. Shafi Goldwasser, Leonid Levin: Fair Computation of General Functions in Presence of Immoral Majority; Crypto ’90, LNCS 537, Springer-Verlag, Berlin 1991, 77–93.
33. 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.
34. Oded Goldreich, Silvio Micali, Avi Wigderson: How to Play any Mental Game – or – a Completeness Theorem for Protocols with Honest Majority; 19th Symposium on Theory of Computing (STOC), ACM, New York 1987, 218–229.
35. Martin Hirt, Ueli Maurer: Player Simulation and General Adversary Structures in Perfect Multiparty Computation; Journal of Cryptology 13/1 (2000) 31–60.
36. Russell Impagliazzo, Bruce M. Kapron: Logics for Reasoning about Cryptographic Constructions; 44th Symposium on Foundations of Computer Science (FOCS), IEEE, 2003, 372–381.
37. ISO/IEC: Information technology – Security techniques – Non-reputation Part 3: Mechanisms using asymmetric techniques; ISO/IEC International Standard 13888-1, 1st Edition, 12.01.1997.
38. Patrick Lincoln, John C. Mitchell, Mark Mitchell, Andre Scedrov: A Probabilistic Poly-Time Framework for Protocol Analysis; 5th Conference on Computer and Communications Security (CCS), ACM, 1998, 112–121.
39. 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 Conference 1997.
40. Silvio Micali, Phillip Rogaway: Secure Computation; Crypto ’91, LNCS 576, Springer-Verlag, Berlin 1992, 392–404.
41. John Mitchell, Ajith Ramanathan, Andre Scedrov, Vanessa Teague: A Probabilistic Polynomial-Time Calculus for the Analysis of Cryptographic Protocols; March 2004, `http://theory.stanford.edu/people/jcm/papers/ProbProcessCalc.pdf`.
42. Torben Pryds Pedersen: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing; Crypto ’91, LNCS 576, Springer-Verlag, Berlin 1992, 129–140.
43. Birgit Pfitzmann: Sorting Out Signature Schemes; 1st Conference on Computer and Communications Security (CCS), ACM, 1993, 74–85.
44. 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`.

45. Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Secure Reactive Systems; IBM Research Report RZ 3206 (#93252) 02/14/2000, IBM Research Division, Zurich, May 2000, `http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz`.
46. Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Provably Secure Certified Mail; IBM Research Report RZ 3207 (#93253), IBM Research Division, Zurich, August 2000, `http://www.semper.org/sirene/publ/PfSW2CertMail.ps.gz`.
47. Birgit Pfitzmann, Michael Waidner: Composition and Integrity Preservation of Secure Reactive Systems; 7th Conference on Computer and Communications Security (CCS), ACM, 2000, 245–254.
48. Birgit Pfitzmann, Michael Waidner: A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission; 2001 IEEE Symposium on Security and Privacy, 184–200.
49. Michael O. Rabin: Transaction Protection by Beacons; Journal of Computer and System Sciences 27 (1983) 256–267.
50. Ajith Ramanathan, John Mitchell, Andre Scedrov, Vanessa Teague: Probabilistic Bisimulation and Equivalence for Security Analysis of Network Protocols; Foundations of Software Science and Computation Structures (FOSSACS) 2004, LNCS 2987, Springer-Verlag, Berlin 2004, 468–483.
51. Roberto Segala, Nancy Lynch: Probabilistic Simulations for Probabilistic Processes; 5th International Conference on Concurrency Theory (CONCUR), LNCS 836, Springer-Verlag, Berlin 1994, 481–497.
52. Matthias Schunter: Optimistic Fair Exchange; PhD thesis, Universität des Saarlandes, Saarbrücken, October 2000, `http://www.semper.org/sirene/publ/Schu7_00.thesis-final.ps.gz`.
53. 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.
54. Andrew C. Yao: Protocols for Secure Computations; 23rd Symposium on Foundations of Computer Science (FOCS), IEEE, 1982, 160–164.
55. Andrew C. Yao: Theory and Applications of Trapdoor Functions; 23rd Symposium on Foundations of Computer Science (FOCS), IEEE, 1982, 80–91.
56. Jianying Zhou, Dieter Gollmann: A Fair Non-repudiation Protocol; 1996 IEEE Symposium on Security and Privacy, 55–61.
57. Jianying Zhou, Dieter Gollmann: An Efficient Non-repudiation Protocol; 10th Computer Security Foundations Workshop (CSFW), IEEE, 1997, 126–132.

12 Appendix

12.1 Remaining Walkthrough Proofs

In the proof of Lemma 2 in the main text, we only presented the walkthrough proof for the case of an honest sender and summaries of the other two cases. We now present the full proofs of these other two cases.

Case R: Correct Recipient; Incorrect Sender. In this case, we have Figure 10 without $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$, and with thr. 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 receive*:* The only round where $\mathsf{cm\_r}$ and thr accept 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., $\mathsf{test}_{pk_r}(m_2) = ((s, r, i), \mathsf{m2}, \ldots)$. Then we put the run in the set Forge_k and set the forged signature to $\mathit{sig}_f := m_2$ and the attacked participant to $u := r$. Note that no other transaction machine signs a message starting $((s, r, i), \mathsf{m2}, \ldots)$ with sk_r, and recall that tuple decomposition is unambiguous. Otherwise (i.e., without acceptable inputs) cm_t and cm_t′ also do not make any outputs.

2. States reached on input receive*:* If $(\mathsf{receive}, s, l)$ is input at in_r?, then thr changes to State r_1 and outputs $(\mathsf{receive}, s, l)$, which dis dispatches to cm_r′. Thus cm_r and cm_r′ wait for an input m_1 in Round $i + 1$.

No m_1. If the adversary does not send a correct m_1, then cm_r outputs (failed) at out_r! in Round $i + 6$, while cm_r′ immediately outputs $(\mathsf{suppress})$ to thr via dis, and thr changes from State r_1 to r_2. Thus in Round $i + 6$, thr outputs (failed) at out_r! as well and changes into State failed. No network outputs are made in this process, and cm_r, cm_r′ and thr never accept any other inputs.
Again, if the adversary nevertheless inputs a correct m_5 to cm_t and cm_t′ in Round $i+5$, we let the run be in Forge_k with $\mathit{sig}_f := m_2$ for the message m_2 contained in m_5 and with $u := r$.

m_1 and m_3. If cm_r and 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 cm_r′ outputs (send, m) at out_r′!. This is dispatched to thr, which changes into State r_3. Thus both cm_r and thr output $(\mathsf{received}, m)$ at 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 cm_t and cm_t′ in Round $i+5$, they both output m_6 at t2s! and t2r!. Additionally, cm_t′ outputs $(\mathsf{send}, \ldots)$ at out_t′!, but thr ignores it, being already in State r_3.

m_1, no m_3, but m_5. If cm_r and cm_r′ do not receive a correct m_3 (after sending m_2), both wait until Round $i + 6$.
If cm_t and cm_t′ obtain a correct m_5 in Round $i + 5$, they both send m_6, and cm_t′ outputs (send, m) at out_t′! for the m contained in m_5. Thus thr changes to State r_3 and outputs $(\mathsf{received}, m)$ at out_r! in Round $i + 6$. As the channel from cm_t to cm_r is authentic, cm_r obtains m_6 as sent by cm_t and also outputs $(\mathsf{received}, m)$ at out_r!.

m_1, no m_3, no m_5. In this case, if cm_r and cm_r′ obtain a correct m_6 in Round $i + 6$, we let the run be in Forge_k with $\mathit{sig}_f := m_6$ and $u := t$. Note that the message signed in m_6 starts $((s, r, i), \mathsf{m6}, \ldots)$ and no other transaction machine signs such a message with sk_t.
Otherwise, cm_r outputs (failed) in Round $i + 6$, and so does thr (changing to State failed) because we saw that it is still in State r_1.

3. Inputs to cm_v *and* cm_v′*:* The remaining accepted inputs are at the ports s2v? of cm_v and cm_v′; they must be made in a round $j \geq i + 7$ and must be correct receipts. (Thus this part of the proof mainly proves that receipts are unforgeable and fixed.) We now denote receipts and their parts with primes (m'_i, l' etc.), and messages handled by the other machines as before.

If a correct message m'_7 arrives, cm_v outputs $(\mathsf{received}, (s, r, i), l', m')$ at out_v!, while cm_v′ outputs (show) at out_v′!, which is dispatched to in_a?. Then thr also outputs $(\mathsf{received}, (s, r, i), l', m')$ at out_v! if it is in State received with the parameters $l = l'$ and $m = m'$. We show that this is true except in certain cases that we put into error sets.

Proof of $l' = l$ (unforgeability of labels in receipts). Both acceptable forms of m'_7 must contain correct messages m'_1 and m'_2. If cm_r and cm_r′ did not send a message m_2 with the same content $((s, r, i), \mathsf{m2}, m_1, p_R)$, let the run be in Forge_k and $\mathit{sig}_f := m'_2$ and $u := r$. Note that no other transaction machine signs a message starting $((s, r, i), \mathsf{m2}, \ldots)$ with sk_r.
From now on, we consider that they sent m_2 and thus $m'_1 = m_1$ and $p'_R = p_R$. The verifications in cm_r and cm_r′ (implied by our notational conventions for state-transition diagrams) imply that the value l' in m_1 equals l as it was input to cm_r and cm_r′, and

thus to thr. Furthermore, cm_r′ makes no output $(\mathsf{suppress})$ and thus thr never changes to State r_2.

Proof that thr is in State received (unforgeability of a receipt).

i. If the correct m'_7 contains the identifier m7, it contains a correct m'_4, in particular a value r'_R with $\mathsf{f}(r'_R) = p_R$. If cm_r and cm_r′ did not send m_4, let the run be in the set $\mathit{OwfBreak}_k$. Note that the original r_R is internal to cm_r and cm_r′ and only used in the assignment $p_R \leftarrow \mathsf{f}(r_R)$.

   If cm_r′ sent m_4, it must have received a correct m_3, i.e., $((s, r, i), m, r_S)$ with $\mathsf{com}_{pkc}(m, r_S) = c$ for the component c of m_1. Then it output (send, m) at out_r′!, which caused thr to change to State r_3, and thus in Round $i + 6$ to State received, storing this variable m.

ii. If the correct m'_7 contains the identifier m6, but cm_t and cm_t′ did not send m_6, let the run be in Forge_k and $\mathit{sig}_f := m'_6$ and $u := t$. Note that no other transaction machine signs a message starting $((s, r, i), \mathsf{m6}, \ldots)$ with sk_t.

   If cm_t and 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 cm_r and cm_r′, we let the run be in Forge_k with $\mathit{sig}_f := m''_2$ and $u := r$. Otherwise, $m''_1 = m_1$ and $m''_3 = ((s, r, i), m, r_S)$ with $\mathsf{com}_{pkc}(m, r_S) = c$ for the component c of m_1. Then cm_t′ output (send, m). This caused thr to change to State r_3, and thus in Round $i + 6$ to received, with this parameter m, except if it was already in State r_3 with a parameter $m'' \neq m$. This would imply that cm_r′ output (send, m''), which it does only if the adversary sent a correct m_3 containing m'', r''_S with $\mathsf{com}_{pkc}(m'', r''_S) = c = \mathsf{com}_{pkc}(m, r_S)$. We then let the run be in $\mathit{BindBreak}_k$ and $\mathit{bindbreak} := (m, r_S, m'', r''_S)$.

Proof of $m' = m$ (unforgeability of the message in the receipt). Both correct forms of m'_7 contain a correct m'_3, i.e., $((s, r, i), m', r'_S)$ where $\mathsf{com}_{pkc}(m', r'_S) = c$ for the component c of m_1. This m' is indeed the one that cm_v outputs.

If $m' \neq m$, let the run be in $\mathit{BindBreak}_k$ and $\mathit{bindbreak} := (m, r_S, m', r'_S)$ with m, r_S from m_3 or m''_3 as derived in the proof that thr is in State received.

Case SR: Correct Sender and Recipient. In this case, we have Figure 10 with all machines, and with thsr. As in Case S, inputs to cm_t, cm_v and cm_t′, cm_v′ can only come from cm_s and cm_s′, respectively, i.e., those will not do anything unless cm_s and cm_s′ do.

1. States reached without inputs send *and* receive*:* As in the first two cases, cm_s, cm_r and thsr only accept inputs (send, r, l, m) and $(\mathsf{receive}, s, l')$ at ports in_s? and 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 cm_s′ and cm_r′.

2a. Input send *alone:* If (send, r, l, m) is input, but $(\mathsf{receive}, s, l')$ is not, then thsr changes to State sr_1 and outputs (send, r, l), which dis dispatches to cm_s′, while cm_r′ obtains no input. cm_s and cm_s′ then send m_1. As in Case S, we define ϕ such that that these messages are equal. cm_r and cm_r′ never leave their starting state and send nothing. If the adversary now inputs a correct m_2 to cm_s and cm_s′, let the run be in Forge_k and $\mathit{sig}_f := m_2$ and $u := r$. (No other transaction machine signs a message $((s, r, i), \mathsf{m2}, \ldots)$ with sk_r.) Otherwise, cm_s outputs (failed) at out_s! in Round $i + 6$, and so does thsr, being in State sr_1 (ignoring an output $(\mathsf{suppress})$ from cm_s′). No further outputs are made, or inputs accepted, in this process, and cm_s, cm_s′, and thsr are in State failed.

2b. Input receive *alone:* If $(\mathsf{receive}, s, l)$ is input, but $(\mathsf{send}, \ldots)$ is not, thsr changes to State sr_2 and outputs $(\mathsf{receive}, s, l)$, which dis dispatches to cm_r′. Then cm_r and cm_r′ wait for m_1, while cm_s and cm_s′ remain in their starting states without making any outputs. If the adversary inputs a correct m_1 to cm_r and cm_r′, let the run be in Forge_k and $\mathit{sig}_f := m_1$ and $u := s$. Note that no other transaction machine signs a message $((s, r, i), \mathsf{m1}, \ldots)$ with

sk_s. Otherwise, $\mathsf{cm\_r}$ outputs (failed) at $\mathsf{out\_r!}$ in Round $i+6$ and so does thsr, being in State sr_2 (ignoring an output $(\mathsf{suppress})$ from $\mathsf{cm\_r}'$). 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, thsr changes to State sr_3 and makes outputs (send, r, l) and $(\mathsf{receive}, s, l')$, 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. As in Case S, we define a mapping ϕ on the random values used such that these messages are equal. 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_k and $\mathit{sig}_f := m_1'$ and $u := s$. 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 the run be in Forge_k and $\mathit{sig}_f := m_2$ and $u := r$. Otherwise, $\mathsf{cm\_s}$ and $\mathsf{cm\_s}'$ do not send further messages either, and $\mathsf{cm\_s}$ and $\mathsf{cm\_r}$ output (failed) in Round $i+6$. So does thsr, being in State sr_3 (ignoring outputs $(\mathsf{suppress})$ from $\mathsf{cm\_s}'$ and $\mathsf{cm\_r}'$). The machines $\mathsf{cm\_s}$, $\mathsf{cm\_s}'$, and thsr are in State failed.

2d. Two matching inputs: Finally, let inputs (send, r, l, m) and $(\mathsf{receive}, s, l)$ be made. Then thsr goes to State sr_4 and outputs (send, r, l) and $(\mathsf{receive}, s, l)$. When these are dispatched, $\mathsf{cm\_s}'$ sends m_1 like $\mathsf{cm\_s}$. From now on we distinguish the messages that arrive on an insecure channel from those that were sent by a superscript "a".

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 (failed) in Round $i+6$, while $\mathsf{cm\_r}'$ outputs $(\mathsf{suppress})$, which causes thsr to change to State sr_3 and thus to output (failed) 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 the run be in Forge_k and $\mathit{sig}_f := m_2$ and $u := r$. Otherwise, $\mathsf{cm\_s}$ also outputs (failed) in Round $i+6$, and no machine makes any further output (except $(\mathsf{suppress})$) or considers inputs. The machines $\mathsf{cm\_s}$, $\mathsf{cm\_s}'$, and thsr 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 the run be in Forge_k and $\mathit{sig}_f := m_1^a$ and $u := s$. 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 (failed) in Round $i+6$, while $\mathsf{cm\_s}'$ inputs $(\mathsf{suppress})$ to thsr, which changes to State sr_3 and thus to failed in Round $i+6$, outputting (failed) 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 the run be in $\mathit{ComOwfBreak}_k$. Note that m_3^a must contain values (m^a, r_S^a) with $c = \mathsf{com}_{pkc}(m^a, r_S^a)$, and that the value r_S from the assignment $(c, r_S) \leftarrow \mathsf{comr}_{pkc}(m)$ in $\mathsf{cm\_s}$, and similarly r_{sim} 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 (failed) 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 thsr. Thus thsr changes from State sr_4 to sr_5 and outputs (msg, m), which dis dispatches to $\mathsf{cm\_s}'$. Then $\mathsf{cm\_s}'$ sends $m_3 := ((s, r, i), m, r_S)$ as well (as in Case S). If m_2^a contains a value $p_R^a \neq p_R$, let the run be in Forge_k and $\mathit{sig}_f := m_2^a$ and $u := r$. Otherwise, we can now speak of one fixed p_R.

Now thsr will output (sent) at $\mathsf{out\_s!}$ and $(\mathsf{received}, m)$ 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 the real structure remain to be derived.

i. If the adversary does not forward a correct m_3^a, then $\mathsf{cm\_r}$ does not send m_4 and waits for m_6. If the adversary nevertheless inputs a correct m_4^a to $\mathsf{cm\_s}$, then this must contain an r_R^a with $\mathsf{f}(r_R^a) = p_R^a = p_R$. Then let the run be in $\mathit{OwfBreak}_k$ and note that the original r_R is internal to $\mathsf{cm\_r}$ and only used in the assignment $p_R \leftarrow \mathsf{f}(r_R)$. 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 (sent) at $\mathsf{out\_s}!$ and $(\mathsf{received}, m)$ 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 $\mathsf{com}_{pkc}(m^a, r_S^a) = \mathsf{com}_{pkc}(m, r_S) = c^a = c$. Then let the run be in $\mathit{BindBreak}_k$ and $\mathit{bindbreak} := (m, r_S, m^a, r_S^a)$. Otherwise, $\mathsf{cm\_r}$ now stores m and outputs $(\mathsf{received}, m)$ 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 (sent) in Round $i + 6$ and changes to State $\mathsf{received}$ or $\mathsf{received}'$.

3. Reactions on input show. The proof for an input (show) is identical to Case S, except that State sr_5 plays the role of State s_3.

This finishes the walkthrough proofs of the two cases omitted in the main text of the paper.