Proactive Secure Message Transmission in Asynchronous Networks

Michael Backes Christian Cachin Reto Strobl
IBM Research
Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{mbc,cca,rts}@zurich.ibm.com

ABSTRACT

We study the problem of secure message transmission among a group of parties in an insecure asynchronous network, where an adversary may repeatedly break into some parties for transient periods of time. A solution for this task is needed in order to use proactive cryptosystems in wide-area networks with loose synchronization. Parties have access to a secure hardware device that stores some cryptographic keys, but can carry out only a very limited set of operations. We provide a formal model of the system, using the framework for asynchronous reactive systems proposed by Pfitzmann and Waidner (Symposium on Security & Privacy, 2001), present a protocol for proactive message transmission, and prove it secure using the composability property of the framework.

Keywords

Proactive Security, Secure Communication

General Terms

Security, Reliability

Categories and Subject Descriptors

D.4.4 [**Security and Protection**]: Communications Management; D.4.6 [**Security and Protection**]: Cryptographic controls

1. INTRODUCTION

Secure message transmission in an open distributed system is one of the fundamental goals of cryptography and network security. Most secure applications rely in some form on the ability of two parties to exchange private and authentic messages.

In this paper, we address the communication among a *group* of parties, of which some may be corrupted by an adversary. The corruptions may be transient and caused by a *mobile adversary* [15] that may move from party to party and eventually corrupt every member of the system, but no more than a certain number at any given time.

PODC'03, July 13-16, 2003, Boston, MA, USA.

Extensive research efforts have resulted in so-called *proactive cryptosystems* [12, 5, 11, 9] in this setting. They typically share the secret keys and the cryptographic capability among the group (using the approach of *threshold cryptography* [8]) and operate in a sequence of *phases*; such systems are secure as long as the adversary does not corrupt more than a certain fraction of the servers during any phase. Technically, the idea is that the servers periodically *refresh* their shares of the secret key between two phases to generate a fresh set of shares that is independent of the previous shares and then erase the old shares.

Extensions of proactive cryptosystems to *asynchronous* networks have been proposed [7, 18, 3], and eliminate a further vulnerability by removing the synchrony assumptions made in previous works. Although some synchronization is necessary to define the notion of a common phase, it has been shown that a minimal synchronization is sufficient and may consist only of one *local* clock pulse to delimit the phases locally for every server [3]. Because phases are long compared to the time spent for transmitting messages (hours compared to milliseconds), such a loose synchronization is easy to implement, using for example the techniques proposed by Barak et al. [2].

The proactive cryptosystem proposed by Cachin et al. [3] relies on network links with the property that every pair of non-corrupted servers that are in the same local phase can exchange private and authenticated messages. Cachin et al. [3] left open the question of how *proactive secure message transmission* could be implemented.

Contribution. We present a protocol for proactive secure message transmission over an asynchronous network and analyze it in the framework for asynchronous reactive systems proposed by Pfitzmann and Waidner [17]. In particular, we specify proactive secure message transmission in terms of an idealized service that has simple deterministic semantics and hides cryptographic objects from its interfaces. Hence, it can easily be analyzed, even by using formal proof tools such as theorem proving. Furthermore, we propose a real implementation and prove it to be at least as secure as the ideal service. Using the framework of Pfitzmann and Waidner for this purpose has the benefit of provably guaranteeing composability, i.e., the security of any system relying on an ideal service for proactive secure message transmission remains the same when our real protocols are used to implement the ideal system.

Just like previous work (see below), our solution relies on a hardware assumption, which in this model is necessary to prevent an adversary from stealing the identity of a corrupted server and masquerading for it henceforth, even if the server is no longer corrupted. In particular, we assume that every server has access to a secure co-processor that cannot be corrupted by the adversary and always behaves as specified.

Recall that a server's local phase is defined by its clock pulse. We show that the way in which this clock pulse is connected to the co-processor determines to what extent proactive secure message transmission can be implemented. Ideally, there is a trusted path from clock to co-processor, but this is not always justified, e.g., when the co-processor is implemented by a smart card. In particular, we distinguish the following three types of connections from clock to co-processor:

1. authentic and reliable connection (clocked co-processor) — clock pulses and only clock pulses are received by the co-processor;

2. reliable but not authentic connection — clock pulses are received, but some may be from the adversary;

3. no connection (stand-alone co-processor) — the adversary controls the clock signal for the co-processor.

To formally capture the possibilities of these different models, we introduce a more general notion for proactive secure networks in terms of an allowed *inaccuracy* $\Delta \in \mathbb{N}$. In particular, such a network only provides secure communication to every pair of non-corrupted servers during their local phase τ if both servers are honest throughout their local phases $[\tau-\Delta, \tau+\Delta]$. In this terminology, we show that the first connection type allows the implementation of an asynchronous proactive secure network for any $\Delta \geq 0$, the second for any $\Delta \geq 0$ only if all clocks are also synchronized within Δ, and the third only for $\Delta \geq 1$ and also if clocks are synchronized within Δ units.

The bulk of the paper addresses the second case with a reliable but not authentic connection from clock to co-processor. The protocol is carried out with the help of the co-processors, which ensures that the adversary has direct access to the messages transmitted in the network. Additionally, the protocol must ensure that if the adversary corrupts a server in some phase and temporarily obtains access to the interface of the co-processor, this does not allow the adversary to gather information about a message of another phase, in which this party is honest. We achieve this by means of a protocol that exchanges a fresh nonce between every pair of parties every time they enter the same local phase.

Related Work. The maintenance of *authenticated* communication in the presence of repeated corruptions has so far only been investigated in synchronous networks [6]. In contrast to our solution, Canetti et al. [6] assume that the underlying network is authentic for short time intervals, but relies only on read-only memory instead of a co-processor. The synchronous solution maintains authenticity by a protocol in which the servers choose new cryptographic keys at the beginning of every phase and exchange these keys over the network. This key exchange is only successful if the network is authentic at this moment. However, the protocol has the attractive feature that a server that has been impersonated and for which the exchange is not successful becomes aware of this and may raise an alarm.

Informal descriptions of proactive secure communication in asynchronous networks have also been given by Castro and Liskov [7] and by Zhou et al. [18]. The former relies on secure co-processors while the latter trusts in system administrators to re-initialize communication keys in every phase.

Organization. In Section 2 we recall the framework to model asynchronous reactive systems that we use to investigate proactive secure message transmission. In Section 3, we provide the ideal specification for proactive secure message transmission. In Section 4 we propose an implementation for the second connection model mentioned above. The other two connection models are discussed in Section 5.

2. PRELIMINARIES

We briefly review the framework and notations for analyzing asynchronous reactive systems proposed by Pfitzmann and Waidner [17].

A *system* consists of a set $\hat{M}$ of probabilistic interactive Turing machines (short *machines*) and a subset S of free ports, called *specified ports*. The machines $\hat{M}$ interact with an *adversary* A and a set of *users* summarized as a single machine H.[1] H connects only to ports in S and A to the rest, and they may interact. In the general scheduling model of [17], a message on a connection c (from an output port c! to an input port c?) is scheduled by the machine that has the corresponding clock output port $\mathsf{c}^{\triangleleft}!$. In asynchronous cryptographic systems, network connections are typically scheduled by A. This yields a runnable system, i.e., one obtains a probability space of runs. Views of individual machines in these runs are the restriction to all in- and outputs this machine sees and its internal state. The computational model is parameterized by a security parameter k, and the running time of every machine — including the users and the adversary — is bounded by a fixed polynomial in k, as in Modern Cryptography [10].

The desired properties of a system are defined in terms of an *ideal* system that typically consists of a single machine TH called *trusted host*. Intuitively, a *real* system is *at least as secure as* an ideal system (denoted by the symbol $\geq$), if whatever can happen to the users in the real system can also happen to the same users in the ideal system; see Figure 1. Formally, the views of H in the real and ideal system, respectively, must be indistinguishable, where indistinguishability is is based on the notion of negligible probabilities [10].

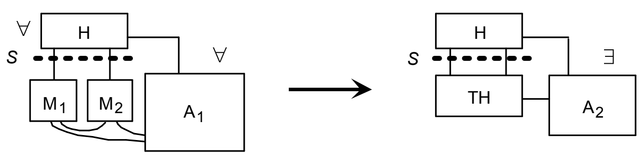

Figure 1: A real system consisting of M_1 and M_2 is shown on the left-hand side, and an ideal system consisting of TH on the right-hand side. For any adversary A_1, and any user H, there must exist an adversary A_2 such that the view of H in both systems is indistinguishable.

Indistinguishability is the strongest possible cryptographic relation between an ideal and a real system, also known as *simulatability*. It is well known that simulatability is transitive [17] and preserved under composition [17, 4], i.e., if a real system securely implements an ideal system in the above sense, then the ideal system can even be securely replaced by the real system if it is used as a subsystem within a larger system. Both properties are crucial for step-wise refinement.

Two-to-One-Cast. In the model described above, an output port p! has exactly one complementary input port p? to which it connects. To model our real system, we will need the concept of a "two-to-one-cast", which can be seen as a connection between *two* output

[1] In [17], a set of machines and specified ports is called a *structure*, and a system is a set of structures. In this work, we only need one structure per system, and call this structure the system. This happens w.l.o.g. as we consider users that are corrupted from time to time, and not necessarily for the entire lifetime of the system.

ports $\mathsf{p}^{\mathsf{a}}!$ and $\mathsf{p}^{\mathsf{b}}!$ of some machines and one input port $\mathsf{p}?$ of some other machine. If a message is output at $\mathsf{p}^{\mathsf{a}}!$ or $\mathsf{p}^{\mathsf{b}}!$, it is delivered to $\mathsf{p}?$; its origin cannot be distinguished.

We can model such a channel by a special machine TO, with ports $\{\mathsf{p}^{\mathsf{a}}?, \mathsf{p}^{\mathsf{b}}?, \mathsf{p}!, \mathsf{p}^{\triangleleft}!\}$, see Figure 2. On input m at either $\mathsf{p}^{\mathsf{a}}?$ or $\mathsf{p}^{\mathsf{b}}?$, TO forwards this input at $\mathsf{p}!$, and schedules it by outputting 1 at $\mathsf{p}^{\triangleleft}!$.

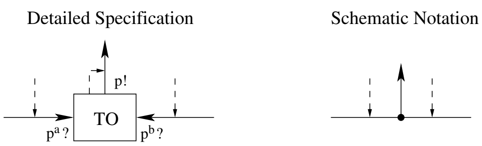

Figure 2: Two-To-One-Cast Machine. Dashed arrows symbolize clock-out ports.

Describing Transition Rules. We use the following conventions for describing transitions of our systems. Each transition starts with ***transition name:*** $\mathsf{p}?(m)$, where $\mathsf{p}?$ is an input port and m an abstract message, i.e., a message template with free variables. We write **enabled if:** *cond* (where *cond* is a boolean expression on machine-internal state variables) to specify the condition under which the transition is enabled. An expression **output:** (m) at $\mathsf{p}!$ specifies that the message m is output at port $\mathsf{p}!$, and that the corresponding clock port $\mathsf{p}^{\triangleleft}!$ is used to schedule this message.

3. PROACTIVE SECURE MESSAGE TRANSMISSION

In this section, we give a specification for proactive secure message transmission with inaccuracy Δ in terms of an ideal system $Sys^{\mathsf{psn,ideal}}_{\Delta,n}$. To motivate our ideal system, we first sketch the asynchronous proactive system model introduced in [3], and then show how our ideal system plugs into that model.

There are n machines, called *servers*, operating in a sequence of local phases defined by a local clock. In particular, every server has a dedicated local clock that emits *clock ticks*, and its *local phase* is defined as the number of clock ticks emitted by this clock. The servers can be corrupted by a *mobile* adversary, which may corrupt a server transiently and repeatedly. Finally, the servers are connected through a proactive secure network, which guarantees intuitively that any two servers that are honest during the same local phase can communicate in a secure way during that phase.

Our ideal system $Sys^{\mathsf{psn,ideal}}_{\Delta,n}$ models the proactive secure network as a single machine $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$, and the n servers as the users $u \in [1, n]$. To send and receive messages, every user u connects to $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ through ports $\mathsf{in}_u!$ and $\mathsf{out}_u?$. The local clock of a user u is modeled as messages of the form $(\mathsf{clock\_tick})$, which $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ receives at a port $\mathsf{clk}_u?$. Corrupting and leaving a user is modeled as messages of the form $(\mathsf{corrupt})$ and (leave), respectively, which $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ receives at a port $\mathsf{cor}_u?$. Finally, $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ connects to the adversary through ports $\{\mathsf{to\_Adv}_u!, \mathsf{from\_Adv}_u? \mid u \in [1, n]\}$; this allows us in the following to leave the scheduling up to the adversary.

Figure 3 summarizes this ideal system, which is of the form $Sys^{\mathsf{psn,ideal}}_{\Delta,n} = (\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}, S)$, where the set S of specified ports consists of all ports the users connect to.

Transition Rules of $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$. The ideal machine $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ models initialization, as well as sending, and receiving of messages. It therefore maintains several internal data-structures, that are denoted by the superscript spec. In particular, it maintains for every

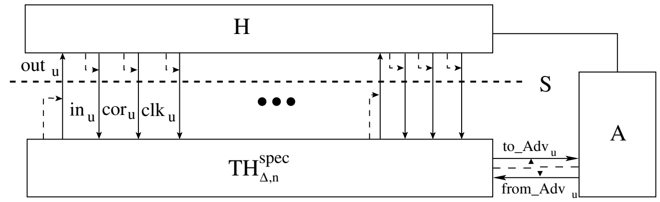

Figure 3: The ideal system for proactive secure message transmission.

$u, v \in [1, n]$ the following variables:

τ^{spec}_u : The current phase of user u. Initially set to 0.

$state^{\mathsf{spec}}_u$: The current state of user u (either $\mathsf{corrupt}$ or honest). Initially set to honest.

$crpt^{\mathsf{spec}}_\tau$: The set of users corrupted during phase τ. Initially set to $\emptyset$ for all phases $\tau \geq 0$

$init^{\mathsf{spec}}_{vu}$: The status of the initialization between user v and u (either 0 or 1). Initially set to 0.

$deliver^{\mathsf{spec}}_{vu}$: The list of messages to be delivered from user v to user u. Initially set to $\emptyset$.

$rec\_msgs^{\mathsf{spec}}_{vu}$: The set of messages that user u has received from user v. Initially set to $\emptyset$.

$\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ works as follows. For every user u, it updates its current phase τ^{spec}_u whenever an input $(\mathsf{clock\_tick})$ occurs at port $\mathsf{clk}_u?$, and subsequently outputs the message $(\mathsf{clock\_tick})$ at port $\mathsf{to\_Adv}_u!$, signaling to the adversary that the phase of u has changed. Similarly, it updates the status of the user whenever an input $(\mathsf{corrupt})$ or (leave) occurs at $\mathsf{cor}_u?$. In either case, it subsequently outputs the message $(\mathsf{corrupt})$ or (leave), respectively, at port $\mathsf{to\_Adv}_u!$, signaling to the adversary that the user u is now corrupted or honest, respectively. The detailed transition rules are as follows.

Corruption: $\mathsf{cor}_u?(\mathsf{corrupt})$

enabled if: $state^{\mathsf{spec}}_u = \mathsf{honest}$
$state^{\mathsf{spec}}_u \leftarrow \mathsf{corrupt}$
Add u to $crpt^{\mathsf{spec}}_{\tau^{\mathsf{spec}}_u}$
output: $(\mathsf{corrupt})$ at $\mathsf{to\_Adv}_u!$

Un-Corruption: $\mathsf{cor}_u?(\mathsf{leave})$

enabled if: $state^{\mathsf{spec}}_u = \mathsf{corrupt}$
$state^{\mathsf{spec}}_u \leftarrow \mathsf{honest}$
output: (leave) at $\mathsf{to\_Adv}_u!$

Clock ticks: $\mathsf{clk}_u?(\mathsf{clock\_tick})$

$\tau^{\mathsf{spec}}_u \leftarrow \tau^{\mathsf{spec}}_u + 1$
if $state^{\mathsf{spec}}_u = \mathsf{corrupt}$ **then**
Add u to $crpt^{\mathsf{spec}}_{\tau^{\mathsf{spec}}_u}$
output: $(\mathsf{clock\_tick})$ at $\mathsf{to\_Adv}_u!$

The transition rules for initialization, sending, and receiving messages are as follows.

Initialization: A user u can initialize communication with other users by inputting a command of the form $(\mathsf{snd\_init})$ to port $\mathsf{in}_u?$ of $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$. This corresponds to key generation and authenticated key exchange in a real system. Upon this input,

$\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ outputs a command $(\mathsf{snd\_init})$ to the adversary at port $\mathsf{to\_Adv}_u!$. This models that the adversary will notice in the real world that keys are being exchanged.

Send Init: $\mathsf{in}_u?(\mathsf{snd\_init})$
  enabled if: $init^{\mathsf{spec}}_{uu} = 0$
  $init^{\mathsf{spec}}_{uu} \leftarrow 1$
  output: $(\mathsf{snd\_init})$ at $\mathsf{to\_Adv}_u!$

The adversary signals to $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ that a user u has received initialization information from a user v by sending a message $(\mathsf{rec\_init}, v)$ at port $\mathsf{from\_Adv}_u?$. $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ then outputs a message $(\mathsf{rec\_init}, v)$ at port $\mathsf{out}_u!$. In a real system, this corresponds to the user u receiving the public key of user v.

Rec Init: $\mathsf{from\_Adv}_u?(\mathsf{rec\_init}, v)$
  enabled if: $(init^{\mathsf{spec}}_{vu} = 0) \wedge (init^{\mathsf{spec}}_{vv} = 1)$
  $init^{\mathsf{spec}}_{vu} \leftarrow 1$
  output: $(\mathsf{rec\_init}, v)$ at $\mathsf{out}_u!$

Sending Messages: A user u can send a message m to another user v by inputting a command of the form (send, m, v) to port $\mathsf{in}_u?$. $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ then stores the tuple $(m, \tau^{\mathsf{spec}}_u)$ in an internal list $deliver^{\mathsf{spec}}_{uv}$; recall that τ^{spec}_u denotes the current phase of user u. Next, a command $(\mathsf{send\_blindly}, i, l, v)$ is output to the adversary at port $\mathsf{to\_Adv}_u!$, where l and i denote the length of the message m and its position in the list, respectively. The length l models the tolerable imperfections, i.e., real cryptographic primitives such as asymmetric encryption do not hide the length of the plain-text. The index i can be used as a *handle* to schedule the message.

Send: $\mathsf{in}_u?(\mathsf{send}, m, v)$
  enabled if: $(init^{\mathsf{spec}}_{vu} = 1) \wedge (init^{\mathsf{spec}}_{uu} = 1)$
  $l \leftarrow \mathsf{len}(m), i \leftarrow \mathsf{size}(deliver^{\mathsf{spec}}_{uv}) + 1$
  $deliver^{\mathsf{spec}}_{uv}[i] \leftarrow (m, \tau^{\mathsf{spec}}_u)$
  output: $(\mathsf{send\_blindly}, v, i, l)$ at $\mathsf{to\_Adv}_u!$

Receiving messages: The adversary signals to $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ that a certain message should be delivered to user u by sending a message of the form $(\mathsf{receive\_blindly}, v, i)$ at $\mathsf{from\_Adv}_u?$. Here, i is the above-mentioned handle. $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ then sets $(m, \tau) \leftarrow deliver^{\mathsf{spec}}_{vu}[i]$, and checks whether it is legitimate (as described below) to deliver the corresponding message to the user u in terms of a message $(\mathsf{received}, v, m)$ at port $\mathsf{out}_u!$.

In particular, if $\tau = \tau^{\mathsf{spec}}_u$, $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ delivers the message. Otherwise, $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ only delivers the message if $\tau \in [\tau^{\mathsf{spec}}_u - \Delta, \tau^{\mathsf{spec}}_u + \Delta]$, and either user u is currently corrupted, or user v is corrupted during phase τ.

To prevent the adversary from scheduling the same message more than once, $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ in addition keeps track of all messages delivered from v to u by storing the indices of delivered messages in an internal set $rec\_msgs^{\mathsf{spec}}_{vu}$. Before actually delivering a message, $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ checks if $i \notin rec\_msgs^{\mathsf{spec}}_{vu}$, and will not deliver the message in case of failure.

Receive: $\mathsf{from\_Adv}_u?(\mathsf{receive\_blindly}, v, i)$
  enabled if: $\mathsf{init}^{\mathsf{spec}}_{uu} = \mathsf{init}^{\mathsf{spec}}_{vu} = 1$
  if $i \leq \mathsf{size}(deliver^{\mathsf{spec}}_{vu}) \wedge i \notin rec\_msgs^{\mathsf{spec}}_{vu}$ **then**
    $(m, \tau) \leftarrow deliver^{\mathsf{spec}}_{vu}[i]$
    if $\tau \in [\tau^{\mathsf{spec}}_u - \Delta, \tau^{\mathsf{spec}}_u + \Delta] \wedge$
      $(v \in crpt^{\mathsf{spec}}_{\tau} \vee u \in crpt^{\mathsf{spec}}_{\tau^{\mathsf{spec}}_u} \vee \tau = \tau^{\mathsf{spec}}_u)$ **then**
      $rec\_msgs^{\mathsf{spec}}_{vu} \leftarrow rec\_msgs^{\mathsf{spec}}_{vu} \cup \{i\}$
      output: $(\mathsf{received}, m, v)$ at $\mathsf{out}_u!$

In a real system, messages to be sent might have to be buffered at some point; for example at the beginning of every phase, until secure communication has been re-established. To model that such message buffers can be flushed, we allow the adversary to schedule many messages from a user v to a user u at once, using the command $(\mathsf{receive\_blindly}, v, idx)$ at $\mathsf{from\_Adv}_u?$, where idx is a list of handles denoting the messages to be delivered. $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ performs for every such message the checks above and delivers the messages satisfying these checks encoded as one long message according to some publicly known encoding algorithm. We write $m_1 \bullet m_2$ to denote the concatenation of two messages according to some fixed encoding.

Receive many: $\mathsf{from\_Adv}_u?(\mathsf{receive\_blindly}, v, idx)$
  enabled if: $\mathsf{init}^{\mathsf{spec}}_{uu} = \mathsf{init}^{\mathsf{spec}}_{vu} = 1$
  $\tilde{m} \leftarrow \perp$
  for all $i \in idx$ **do**
    if $i \leq \mathsf{size}(deliver^{\mathsf{spec}}_{vu}) \wedge i \notin rec\_msgs^{\mathsf{spec}}_{vu}$ **then**
      $(m, \tau) \leftarrow deliver^{\mathsf{spec}}_{vu}[i]$
      if $\tau \in [\tau^{\mathsf{spec}}_u - \Delta, \tau^{\mathsf{spec}}_u + \Delta] \wedge$
        $(v \in crpt^{\mathsf{spec}}_{\tau} \vee u \in crpt^{\mathsf{spec}}_{\tau^{\mathsf{spec}}_u} \vee \tau = \tau^{\mathsf{spec}}_u)$
      then
        $rec\_msgs^{\mathsf{spec}}_{vu} \leftarrow rec\_msgs^{\mathsf{spec}}_{vu} \cup \{i\}$
        $\tilde{m} \leftarrow \tilde{m} \bullet m$
  if $\tilde{m} \neq \perp$
    output: $(\mathsf{received}, m, v)$ at $\mathsf{out}_u!$

Finally, if an input occurs at port $\mathsf{in}_u?$ such that no transition rule applies, $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ forwards this input to the adversary at the port $\mathsf{to\_Adv}_u!$.

For our implementation of this ideal system, the local clocks need to be synchronized to some extent. We will express this synchronization as the *upper bound on the clock shift*, which is a value $\rho \in \mathbb{N}$ such that the following holds: At every point in time there exists an integer τ such that every user u is in a phase τ_u, where $\tau \leq \tau_u \leq \tau + \rho$. Note that in a real implementation, where a phase is typically on the order of days, assuming $\rho = 1$ is a very loose synchronization and easy to achieve.

Liveness. Protocols designed for an asynchronous proactive network model typically guarantee liveness properties, i.e., that something "good" will eventually happen, only to the extent that all messages sent between honest users in one particular phase are also delivered within this phase. We capture this property as a condition on the messages that occur at the specified ports of $Sys^{\mathsf{psn,ideal}}_{\Delta,n}$ as follows:

A run of $Sys^{\mathsf{psn,ideal}}_{\Delta,n}$ (or an implementation thereof) is called *live*, if for any phase τ and any users u, v that are honest throughout the phases $[\tau - \Delta, \tau + \Delta]$, every message sent by user u during phase τ to user v is delivered to user v.

We require that a real implementation of $Sys^{\mathsf{psn,ideal}}_{\Delta,n}$ produces live runs for certain schedulers. This rules out trivial protocols that do nothing at all.

Inaccuracy. Existing protocols designed in the proactive model typically rely on a proactive secure network with inaccuracy 0, and guarantee security to the extent the adversary A corrupts at most t servers during every $\nu (\geq 1)$ subsequent phases. The parameter ν can be seen as the speed with which the adversary may switch the set of corrupted servers. We also call such an adversary (t, ν)-*limited.*

If such a protocol Π is run on a proactive secure network with inaccuracy $\Delta > 0$, it guarantees the same security for a (t, ν')-limited adversary A', where $\nu' \geq \nu + 2\Delta$.

This can be verified as follows. The advantage that A' gains over A by having a network with inaccuracy $\Delta > 0$ is that A' can send and receive messages in the name of an honest server during a phase τ, if she corrupts this server during a phase in $[\tau - \Delta, \tau + \Delta]$. W.l.o.g. one can consider such a server as corrupted from Π's point of view. On the other hand, A' corrupts at most t servers during any $\nu + 2\Delta$ subsequent phases. Hence, during any ν subsequent phases, at most t servers will be considered corrupted in Π, which may also happen if Π is run against adversary A in a proactive network with inaccuracy 0.

4. DERIVING A SECURE IMPLEMENTATION

In this section we show how to build a real system $Sys_{\Delta,n}^{\mathsf{psn,real}}$ that is at least as secure as the ideal system for proactive secure message transmission. Our real system consists of n machines that communicate over an asynchronous insecure network. Furthermore, every machine has a secure co-processor or, alternatively, a secure module. This module is secure in the sense that it is not affected by a transient corruption of the machine.

Our construction consists of two steps. In the first step, we assume that the n machines communicate over an asynchronous *secure* network (modeled as a trusted host $\mathsf{TH}_n^{\mathsf{sm}}$), and filter the messages exchanged according to the rules of proactive secure message transmission. We call this the *hybrid* system $Sys_{\Delta,n}^{\mathsf{psn,hybrid}}$, and refer to the n machines as the real system $Sys_{\Delta,n}^{\mathsf{filter}}$, and to the trusted host as the ideal system $Sys_n^{\mathsf{sm,ideal}}$; see Figure 4. In a second step, we derive the real system $Sys_{\Delta,n}^{\mathsf{psn,real}}$ by replacing the ideal specification $Sys_n^{\mathsf{sm,ideal}}$ with a secure implementation $Sys_n^{\mathsf{sm,real}}$.

We prove the security of $Sys_{\Delta,n}^{\mathsf{psn,real}}$ in two steps. In the first step, we show that $Sys_n^{\mathsf{sm,real}}$ is at least as secure as $Sys_n^{\mathsf{sm,ideal}}$. Because our specifications for these systems follow very closely the ones given in [1], this proof is almost identical to the one given in [1], and comes essentially "for free". In the second step, we show that $Sys_{\Delta,n}^{\mathsf{psn,hybrid}}$ is at least as secure as $Sys_{\Delta,n}^{\mathsf{psn,ideal}}$. The security of the real system $Sys_{\Delta,n}^{\mathsf{psn,real}}$ then follows from the composition theorem (cf. Section 2).

For the remainder of this section we give the specifications of $Sys_n^{\mathsf{sm,ideal}}$ and $Sys_n^{\mathsf{sm,real}}$, as well as the specification for the filtering system $Sys_{\Delta,n}^{\mathsf{filter}}$. We then prove the security of the resulting hybrid system, and describe how to derive the real system $Sys_{\Delta,n}^{\mathsf{psn,real}}$.

4.1 Secure Message Transmission

The Ideal System. The specification of secure message transmission is of the form $Sys_n^{\mathsf{sm,ideal}} = (\mathsf{TH}_n^{\mathsf{sm}}, S)$, where the specified ports S are $\{\mathsf{in}'_u?, \mathsf{out}'_u! \mid u \in [1,n]\}$. The ideal machine $\mathsf{TH}_n^{\mathsf{sm}}$ models initialization, sending and receiving of messages similar to the machine $\mathsf{TH}_{\Delta,n}^{\mathsf{spec}}$. It therefore maintains for every $v, u \in [1,n]$ the internal data-structures $init_{vu}^{\mathsf{sm}}$, $deliver_{vu}^{\mathsf{sm}}$, and $rec\_msgs_{vu}^{\mathsf{sm}}$, which have the same meaning as the corresponding data-structures of $\mathsf{TH}_{\Delta,n}^{\mathsf{spec}}$, and are initialized in the same way. Note that we use the superscript sm to denote internal variables of $\mathsf{TH}_n^{\mathsf{sm}}$.

$\mathsf{TH}_n^{\mathsf{sm}}$ works as follows. A user u can initialize communication and send messages by inputting commands at port $\mathsf{in}'_u?$ of the form $(\mathsf{snd\_init})$ and (send, m, v), respectively. $\mathsf{TH}_n^{\mathsf{sm}}$ notifies the adversary of such events by outputting $(\mathsf{snd\_init})$ and $(\mathsf{send\_blindly}, i, l, v)$, respectively, at $\mathsf{to\_Adv}'_u!$, where i is a handle to the message m, and l is the length of m.

Send Init: $\mathsf{in}'_u?(\mathsf{snd\_init})$
    enabled if: $init_{uu}^{\mathsf{sm}} = 0$
    $init_{uu}^{\mathsf{sm}} \leftarrow 1$
    output: $(\mathsf{snd\_init})$ at $\mathsf{to\_Adv}'_u!$

Send: $\mathsf{in}'_u?(\mathsf{send}, m, v)$
    enabled if: $(init_{vu}^{\mathsf{spec}} = 1) \wedge (init_{uu}^{\mathsf{spec}} = 1)$
    $l \leftarrow \mathsf{len}(m)$, $i \leftarrow \mathsf{size}(deliver_{uv}^{\mathsf{sm}}) + 1$
    $deliver_{uv}^{\mathsf{sm}}[i] \leftarrow (m, \tau_u^{\mathsf{spec}})$
    output: $(\mathsf{send\_blindly}, v, i, l)$ at $\mathsf{to\_Adv}'_u!$

The adversary can then schedule the delivery of initialization information from v to u, or the delivery of a message by outputting $(\mathsf{rec\_init}, v)$ or $(\mathsf{deliver\_blindly}, i, v)$, respectively, at the port $\mathsf{from\_Adv}'_u!$. If this happens, $\mathsf{TH}_n^{\mathsf{sm}}$ outputs corresponding messages of the form $(\mathsf{rec\_init}, v)$ and $(\mathsf{received}, v, m)$ at port $\mathsf{out}'_u!$ to user u. $\mathsf{TH}_n^{\mathsf{sm}}$ guarantees that the adversary can only schedule a message if she has initialized the corresponding connection before. Moreover, it guarantees that the adversary cannot schedule the same message more than once.

Rec Init: $\mathsf{from\_Adv}'_u?(\mathsf{rec\_init}, v)$
    enabled if: $(init_{vu}^{\mathsf{sm}} = 0) \wedge (init_{vv}^{\mathsf{sm}} = 1)$
    $init_{vu}^{\mathsf{sm}} \leftarrow 1$
    output: $(\mathsf{rec\_init}, v)$ at $\mathsf{out}'_u!$

Receive: $\mathsf{from\_Adv}'_u?(\mathsf{receive\_blindly}, v, i)$
    enabled if: $\mathsf{init}_{uu}^{\mathsf{sm}} = \mathsf{init}_{vu}^{\mathsf{sm}} = 1$
    if $i \leq \mathsf{size}(deliver_{vu}^{\mathsf{sm}}) \wedge i \notin rec\_msgs_{vu}^{\mathsf{sm}}$ **then**
      $m \leftarrow deliver_{vu}^{\mathsf{sm}}[i]$
      $rec\_msgs_{vu}^{\mathsf{sm}} \leftarrow rec\_msgs_{vu}^{\mathsf{sm}} \cup \{i\}$
      output: $(\mathsf{received}, m, v)$ at $\mathsf{out}'_u!$

A user u can also send a message to all other users by inputting a single command $(\mathsf{send\_to\_all}, m)$ at $\mathsf{in}'_u?$, and $\mathsf{TH}_n^{\mathsf{sm}}$ subsequently outputs $(\mathsf{send\_blindly\_to\_all}, (i_1, \ldots, i_n), l)$ at $\mathsf{to\_Adv}'_u!$, where l is the length of the message, and i_v is a handle for the copy of the message to be delivered to v.

Send many: $\mathsf{in}'_u?(\mathsf{send\_to\_all}, m)$
    enabled if: $(init_{uu}^{\mathsf{spec}} = 1) \wedge_{i=1}^{n} (init_{iu}^{\mathsf{spec}} = 1)$
    for $v \in [1,n]$ **do**
      $i_v \leftarrow \mathsf{size}(deliver_{uv}^{\mathsf{sm}}) + 1$
      $deliver_{uv}^{\mathsf{sm}}[i_v] \leftarrow m$
    $l \leftarrow \mathsf{len}(m)$
    output: $(\mathsf{send\_blindly\_to\_all}, (i_1, \ldots, i_n), l)$ at $\mathsf{to\_Adv}'_u!$

This system $Sys_n^{\mathsf{sm,ideal}}$ differs from the one given in [1] only in the following points. The original specification guarantees that messages are delivered in order; our specification only guarantees that messages are not delivered twice. Furthermore, our specification has the additional functionality that a user can simultaneously send a message to all other users by the special $\mathsf{send\_to\_all}$ command.

The Real System. The real system is of the form $Sys_n^{\mathsf{sm,real}} = (\{\mathsf{M}_u^{\mathsf{sm}} \mid u \in [1,n]\}, S)$, i.e., it consists of n machines, where the specified ports S are the same as in the ideal system. Every pair of machines u and v communicate over an insecure asynchronous point-to-point link, modeled as net_{uv}, which is scheduled by the adversary. Moreover, the adversary may read all messages sent over such a link, and insert messages of her choice. During initialization, the machines may use asynchronous authentic channels aut_{uv}, i.e, the adversary may schedule and read messages on these channels, but not insert any message.

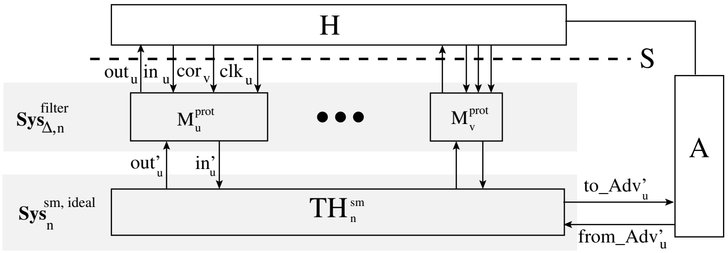

Figure 4: Overall structure of the hybrid system

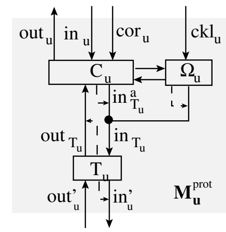

Figure 5: Internal structure of $\mathsf{M}_u^{\mathsf{prot}}$

The implementation uses asymmetric encryption and digital signatures as cryptographic primitives. If an input $(\mathsf{snd\_init})$ occurs at $\mathsf{in}'_u?$, $\mathsf{M}_u^{\mathsf{sm}}$ creates signature and encryption keys and sends the corresponding public keys to every other machine $\mathsf{M}_v^{\mathsf{sm}}$ over the corresponding authentic channel aut_{uv}. If $\mathsf{M}_u^{\mathsf{sm}}$ receives the public keys of $\mathsf{M}_v^{\mathsf{sm}}$, it outputs $(\mathsf{rec\_init}, v)$ at $\mathsf{out}'_u!$. Furthermore, if an input (send, m, v) occurs at $\mathsf{in}'_u?$, $\mathsf{M}_u^{\mathsf{sm}}$ sends the message m signed and encrypted over the insecure channel net_{uv} to $\mathsf{M}_v^{\mathsf{sm}}$, which decrypts the cipher-text, verifies the signature, and if the check succeeds, outputs $(\mathsf{received}, m, u)$ at $\mathsf{out}'_v!$. An input $(\mathsf{send\_to\_all}, m)$ at $\mathsf{in}'_u?$ is treated similarly, except that the signed and encrypted message is sent to every other machine $\mathsf{M}_v^{\mathsf{sm}}$ for $v \in [1, n]$ over the insecure channel. To prevent replay attacks, every machine keeps track of the number of messages sent to every other machine $\mathsf{M}_v^{\mathsf{sm}}$, and sends this index (signed and encrypted) along with the corresponding message; if it receives a message with the same index from the same machine $\mathsf{M}_v^{\mathsf{sm}}$ twice, it discards the message.

The detailed transition rules for $Sys_n^{\mathsf{sm,real}}$ can easily be derived from [1] and the description given above, and are omitted. Furthermore, we also omit the proof that $Sys_n^{\mathsf{sm,real}}$ is at least as secure as our ideal specification, as it can easily be derived from the proof given in [1].

4.2 The Filtering System

The filtering system consists of n machines. The machines connect to the users H through the same set of ports as in the ideal specification, i.e., $\mathsf{in}_u?$, $\mathsf{out}_u!$, $\mathsf{cor}_u?$, and $\mathsf{clk}_u?$ for each user u. The machines communicate with each other through the ideal service for asynchronous secure message transmission described in Section 4.1, i.e., every machine $\mathsf{M}_u^{\mathsf{prot}}$ can access $\mathsf{TH}_n^{\mathsf{sm}}$ through ports $\mathsf{out}'_u?$, and $\mathsf{in}'_u!$. This situation is illustrated in Figure 4.

Internally, every machine $\mathsf{M}_u^{\mathsf{prot}}$ consists of three machines C_u, T_u, and Ω_u. C_u models the part of $\mathsf{M}_u^{\mathsf{prot}}$, which can be corrupted by the adversary, as opposed to the other two machines, which always behave as specified. T_u represents the concept of a secure co-processor that executes the actual filtering protocol. Ω_u represents the concept of a secure module that keeps track of the current phase of M_u. Technically, it is a counter that increases an internal variable τ whenever it observes a $\mathsf{clock\_tick}$ at $\mathsf{clk}_u?$, and subsequently outputs $(\mathsf{clock\_tick}, \tau)$ at its output port. Figure 5 describes the internal structure of $\mathsf{M}_u^{\mathsf{prot}}$ in detail.

Note that the counter Ω_u could be omitted, assuming that the index τ of the local phase is passed along with the $\mathsf{clock\_tick}$ messages received at $\mathsf{clk}_u?$. However, as a consequence, the security of an implementation could only be proven for users H which pass the correct phase indices along with the $\mathsf{clock\_tick}$ messages. Introducing Ω_u as we propose allows to avoid such an undesired restriction on the users H in the security analysis.

Note also that T_u receives messages from Ω_u and C_u at the same port (see "two-to-one cast" in Section 2). Such a connection from Ω_u to T_u can be seen as *reliable but not authentic*. In Section 5 we consider two natural variations thereof, and analyze them in terms of practicality and the possible implementations.

In a prototype implementation done at IBM Research, the structure shown in Figure 5 was realized as a Java smart card [14] connected to the serial port of a server, in conjunction with a trusted system administrator that resets the smart card at the beginning of every phase on the current phase index. Alternatively, one could use a TCPA chip [16] in conjunction with a watchdog timer (provided by the motherboard or an extension card) that interrupts processing and resets the chip accordingly. Such hardware modules are available at essentially no additional cost.

The protocol. We now present a protocol that implements a proactive secure network with inaccuracy Δ whenever Δ is an upper bound on the clock shift.

The protocol works as follows. C_u acts as a transparent connection between the user and T_u, except that it does not forward inputs of the form $(\mathsf{clock\_tick}, \cdot)$ from an honest user to T_u. This models the fact that honest users do not send fake $\mathsf{clock\_tick}$'s to the secure co-processor. The corresponding transition rules of C_u can be easily derived, and are omitted.

T_u executes the main part of the protocol. It therefore maintains several data-structures, that are denoted by the superscript prot. In particular, it maintains the following variables for $v \in [1, n]$:

τ^{prot} : The (supposed) current phase. Initially set to 0.

r_{uv}^{prot} : Nonces that hold values in $\{0,1\}^k$, where k is the security parameter. For $v \neq u$, r_{uv}^{prot} is initially set to the distinguished symbol $\perp$, and r_{uu}^{prot} is initially set to a value uniformly chosen at random from $\{0,1\}^k$.

sb_{uv}^{prot} : A buffer for messages to be sent to T_v. Initially set to $\perp$.

T_u works as follows. Whenever an input $(\mathsf{clock\_tick}, \tau')$ occurs, it sets $\tau^{\mathsf{prot}} \leftarrow \tau'$, assigns to the nonce r_{uu}^{prot} a value uniformly chosen at random from $\{0,1\}^k$, and sets $r_{uv}^{\mathsf{prot}} \leftarrow \perp$ for all other users v. It then exchanges the nonce r_{uu}^{prot} tagged with the current phase τ^{prot} with every other server by means of the following two-flow handshake protocol. It first sends a message $(\mathsf{open}, r_{uu}^{\mathsf{prot}}, \tau^{\mathsf{prot}})$ to all servers. Upon receiving $(\mathsf{open}, r', \tau')$ from T_v where $\tau^{\mathsf{prot}} = \tau'$, it sends $(\mathsf{ack}, r', r_{uu}^{\mathsf{prot}})$ to T_v. Finally, upon receiving $(\mathsf{ack}, r'_1, r'_2)$ from T_v, it checks whether $r_{uu}^{\mathsf{prot}} = r'_1$. If this holds, and r_{uv}^{prot} is still equal to $\perp$, it sets $r_{uv}^{\mathsf{prot}} \leftarrow r'_2$.

Sending and receiving messages works as follows. If an input (send, m, v) occurs at $\mathsf{in}_{\mathsf{T}_u}?$, it checks that $r_{uv}^{\mathsf{prot}} \neq \perp$. If this holds, it sends a message $m' = (m, r_{uv}^{\mathsf{prot}})$ to v; otherwise, it appends m to the buffer sb_{uv}^{prot}, which it sends to v as soon as r_{uv}^{prot} is set to a value different from $\perp$. However, if a $(\mathsf{clock\_tick}, \tau')$ occurs before, it resets the buffer to $\perp$ and sends nothing to v. If an input $(\mathsf{received}, (m, r'), v)$ occurs at $\mathsf{out}'_u?$, it verifies that

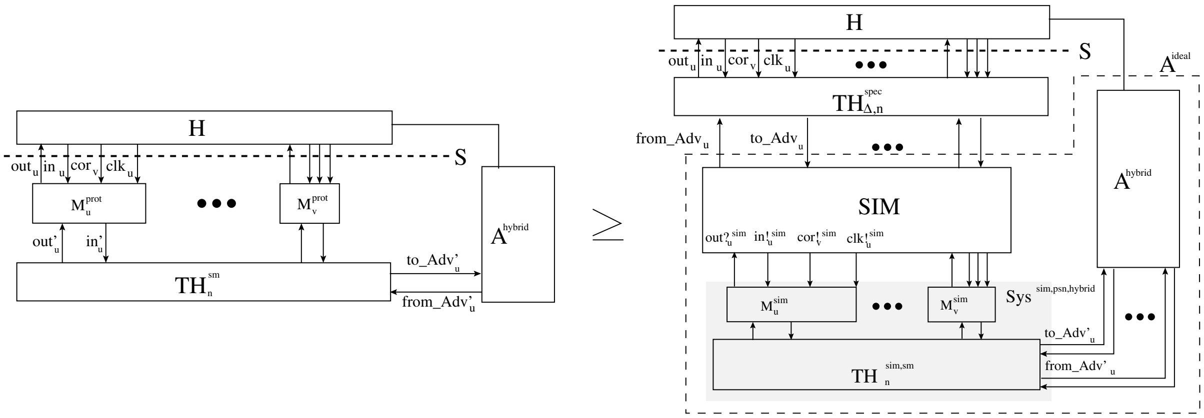

Figure 6: Proof overview of $Sys^{\mathsf{psn,hybrid}}_{\Delta,n} \geq Sys^{\mathsf{psn,ideal}}_{\Delta,n}$

$r^{\mathsf{prot}}_{uu} = r'$ holds, and outputs $(\mathsf{received}, m, v)$ if the check succeeds. Technically, this yields the following transition rules. We write $a \leftarrow_R \{0,1\}^k$ to denote that a value is chosen uniformly at random over $\{0,1\}^k$ and assigned to the variable a.

Clock tick: $\mathsf{in}'_{\mathsf{T}_u}?(\mathsf{clock\_tick}, \tau')$
$\tau^{\mathsf{prot}} \leftarrow \tau'$
$sb^{\mathsf{prot}}_{uv} \leftarrow \bot$ for all $v \in [1,n]$
$r^{\mathsf{prot}}_{uv} \leftarrow \bot$ for all $v \in [1,n]$
$r^{\mathsf{prot}}_{uu} \leftarrow_R \{0,1\}^k$
output: $(\mathsf{send\_to\_all}, (\mathsf{open}, r^{\mathsf{prot}}_{uu}, \tau^{\mathsf{prot}}))$ at $\mathsf{in}'_u!$

Handshake (stage one): $\mathsf{out}'_u?(\mathsf{receive}, (\mathsf{open}, r', \tau'), v)$
if $\tau^{\mathsf{prot}} = \tau'$ **then**
output: $(\mathsf{send}, (\mathsf{ack}, r', r^{\mathsf{prot}}_{uu}), v)$ at $\mathsf{in}'_u!$

Handshake (stage two): $\mathsf{out}'_u?(\mathsf{receive}, (\mathsf{ack}, r'_1, r'_2), v)$
if $(r^{\mathsf{prot}}_{uu} = r'_1) \wedge (r^{\mathsf{prot}}_{uv} = \bot)$ **then**
$r^{\mathsf{prot}}_{uv} \leftarrow r'_2$
if $sb^{\mathsf{prot}}_{uv} \neq \bot$ **then**
$m \leftarrow sb^{\mathsf{prot}}_{uv},\ sb^{\mathsf{prot}}_{uv} \leftarrow \bot$
output: $(\mathsf{send}, (\mathsf{tx}, m, r^{\mathsf{prot}}_{uv}), v)$ at $\mathsf{in}'_u!$

Send: $\mathsf{in}'_{\mathsf{T}_u}?(\mathsf{send}, m, v)$
if $r^{\mathsf{prot}}_{uv} \neq \bot$ **then**
output: $(\mathsf{send}, (\mathsf{tx}, m, r^{\mathsf{prot}}_{uv}), v)$ at $\mathsf{in}'_u!$
else
$sb^{\mathsf{prot}}_{uv} \leftarrow sb^{\mathsf{prot}}_{uv} \bullet m$

Receive: $\mathsf{out}'_u?(\mathsf{receive}, (\mathsf{tx}, m, r'), v)$
if $r^{\mathsf{prot}}_{uu} = r'$ **then**
output: $(\mathsf{received}, m, v)$ at $\mathsf{out}'_{\mathsf{T}_u}!$

Finally, T_u transparently forwards initialization messages $(\mathsf{snd\_init})$ and $(\mathsf{rec\_init}, v)$ between $\mathsf{TH}^{\mathsf{sm}}_n$ and C_u. These transition rules can easily be derived and are omitted.

4.3 Security of the Hybrid System

In this section, we show that $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$ is at least as secure as $Sys^{\mathsf{psn,ideal}}_{\Delta,n}$ whenever Δ is an upper bound on the clock shift. Moreover, we show that a run of $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$ is live, if the adversary schedules the messages sent over $Sys^{\mathsf{sm,ideal}}_{n}$ as follows: For every phase τ, and every two users u and v honest throughout phases $[\tau-\Delta, \tau+\Delta]$, it holds that if (send, m, v) occurs at $\mathsf{in}'_u?$ when user u is in phase τ, then $(\mathsf{receive}, m, u)$ occurs at $\mathsf{out}'_v!$ when user v is in phase τ. We also say that $Sys^{\mathsf{sm,ideal}}_{n}$ is *live within* $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$.

Theorem 4.1 *For all $n, \Delta \in \mathbb{N}$, $Sys^{\mathsf{psn,hybrid}}_{\Delta,n} \geq Sys^{\mathsf{psn,ideal}}_{\Delta,n}$, if Δ is an upper bound on the clock shift. Moreover, a run of $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$ is live, provided that $Sys^{\mathsf{sm,ideal}}_{n}$ is live within $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$.* □

The proof of the theorem consists of security and liveness.

Security. To prove the first part of Theorem 4.1, we provide for every user H, and every adversary A^{hybrid} against the hybrid system, the construction of an adversary A^{ideal} against the ideal system with the same users H, such that the view at the specified ports of both systems is indistinguishable. We construct the adversary A^{ideal} as the combination of several machines: A simulated hybrid system $Sys^{\mathsf{sim,psn,hybrid}}_{\Delta,n}$, the given adversary A^{hybrid}, and a simulator SIM. The ports of the machines are renamed and connected to each other as sketched in Figure 6. Clock ports are omitted for readability.

The goal of the simulator is to ensure that for the given users H and the given adversary $\mathsf{A}^{\mathsf{hybrid}}$, the view at the ports of H in $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$ is indistinguishable from the view at the same ports in $Sys^{\mathsf{psn,ideal}}_{\Delta,n}$. Our simulator achieves this by simulating the machines H for the hybrid sub-system $Sys^{\mathsf{sim,psn,hybrid}}_{\Delta,n}$ as follows. Whenever $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ indicates that a user has become corrupted or honest, or that its clock has ticked, it sends the corresponding message to the hybrid sub-system. If another input occurs at port $\mathsf{to\_Adv}_u?$, it forwards this input directly to the port $\mathsf{in}'_u!$, except if is of the form $(\mathsf{send\_blindly}, v, i, l)$. In this case, SIM tells the sub-system to send a message $m' \triangleq (i, \mathsf{pad})$ of length l from user u to user v. On the other hand, whenever a user u from the sub-system indicates to SIM that it has received a message $m' \triangleq (i, \mathsf{pad})$ from user v, SIM instructs to $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ to deliver the i-th message from user u to v. In case the message m' is of the form $(i_1, \mathsf{pad}_1) \bullet (i_2, \mathsf{pad}_2), \cdots$, i.e., encodes multiple messages, then SIM summarizes all indices i_j in a list *idx* and instructs $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$ to deliver the messages from user u to v denoted by idx at the same time. The detailed transition rules are given below. We use the notation $|i|$ to denote the bit-length of an integer i.

Ideal-Hybrid: $\mathsf{to\_Adv}_u?(m)$
if $m = \mathsf{clock\_tick}$ **then output:**$(\mathsf{clock\_tick})$ at $\mathsf{clk}^{\mathsf{sim}}_u!$
else if $m = \mathsf{corrupt}$ **then output:** $(\mathsf{corrupt})$ at $\mathsf{cor}^{\mathsf{sim}}_u!$
else if $m = \mathsf{leave}$ **then output:** (leave) at $\mathsf{cor}^{\mathsf{sim}}_u!$
else if $m = (\mathsf{send\_blindly}, v, i, l)$ **then**
$\mathsf{pad} \leftarrow 0^{l-|i|}$
output: $(\mathsf{send}, (i, \mathsf{pad}), v)$ at $\mathsf{in}^{\mathsf{sim}}_u!$
else output: (m) at $\mathsf{in}^{\mathsf{sim}}_u!$

Hybrid-Ideal: $\mathsf{out}_u^{\mathsf{sim}}?(m)$
 if $m = (\mathsf{rec\_init}, v)$ **then**
  output: $(\mathsf{rec\_init}, v)$ at $\mathsf{from\_Adv}_u!$
 else if $m = (\mathsf{received}, v, \tilde{m})$ **then**
  $(i, \mathsf{pad}) \leftarrow \tilde{m}$
  output: $(\mathsf{receive\_blindly}, v, i)$ at $\mathsf{from\_Adv}_u!$
 else if $m = (\mathsf{received}, v, \tilde{m}_1 \bullet \cdots \bullet \tilde{m}_L)$ **then**
  forall $j \in [1, L]$ **do**
   $(i_j, \mathsf{pad}_j) \leftarrow \tilde{m}_j$
  output:
   $(\mathsf{deliver\_blindly}, v, (i_1, \ldots, i_L))$ at $\mathsf{from\_Adv}_u!$

To prove that our simulator ensures indistinguishability of the view at the ports of H, we have to show that the collections $M^{\mathsf{hybrid}} \triangleq \{\mathsf{M}_u^{\mathsf{prot}} \mid u \in [1,n]\} \cup \{\mathsf{TH}_n^{\mathsf{sm}}\}$ and $M^{\mathsf{ideal}} \triangleq \{\mathsf{TH}_{\Delta,n}^{\mathsf{spec}}\} \cup \{\mathsf{SIM}\} \cup \{\mathsf{M}_u^{\mathsf{sim}} \mid u \in [1,n]\} \cup \{\mathsf{TH}_n^{\mathsf{sim,sm}}\}$ have the same input-output behavior. More precisely, we have to show that they produce identically distributed outputs at ports $\{\mathsf{out}_u!, \mathsf{to\_Adv}'_u! \mid u \in [1,n]\}$, if they receive identically distributed inputs at ports $\{\mathsf{in}_v?, \mathsf{clk}_v?, \mathsf{cor}_v?, \mathsf{from\_Adv}'_v? \mid v \in [1,n]\}$, except with negligible probability.

This can be done by a probabilistic bisimulation, i.e., one first defines a relation ϕ between the states of the two collections M^{hybrid} and M^{ideal}, and shows that the relation holds for the initial states. In a second step, one shows that if the two collections are in states such that the relation holds, then (1) they produce for every possible input identical outputs, and moreover, (2) the relation also holds for the corresponding next states.

In the following, we sketch only a simplified bisimulation for lack of space. In particular, we only consider runs of the hybrid system $Sys_{\Delta,n}^{\mathsf{psn,hybrid}}$ where the machines $\mathsf{M}_u^{\mathsf{prot}}$ never have to buffer messages, i.e., for every machine $\mathsf{M}_u^{\mathsf{prot}}$, it holds that $sb_{uv}^{\mathsf{prot}} = \bot$ for all $v \in [1,n]$ throughout the entire run. We also call such a run *simple*. Considering only simple runs allows to simplify the relation ϕ, and to focus on the main ideas of the proof. It is straightforward but tedious to extend our mapping ϕ, and to conduct a rigorous bisimulation for *any* run of the hybrid system.

The relation ϕ of our simplified bisimulation is defined as the logical AND-conjunction of the relations ϕ_i given below. The superscript sim denotes the internal data-structures of the simulated hybrid system $Sys_{\Delta,n}^{\mathsf{sim,psn,hybrid}}$.

$$\begin{aligned}
&\forall\, u, v \in [1,n],\ i \in [1, \mathsf{size}(deliver_{vu}^{\mathsf{sm}})]\ \text{ such that}\\
&\quad deliver_{vu}^{\mathsf{sm}}[i] = (\mathsf{tx}, m', r^{\mathsf{prot}}) \wedge\\
&\quad deliver_{vu}^{\mathsf{sim}}[i] = (\mathsf{tx}, (j, \mathsf{pad}), r^{\mathsf{sim}}) \wedge\\
&\quad deliver_{vu}^{\mathsf{spec}}[j] = (m, \tau):
\end{aligned}$$

$$\begin{array}{lrcl}
(\phi_1) & \mathsf{size}(deliver_{vu}^{\mathsf{sm}}) & = & \mathsf{size}(deliver_{vu}^{\mathsf{sim}})\\
(\phi_2) & init_{vu}^{\mathsf{sm}} & = & init_{vu}^{\mathsf{sim}}\\
(\phi_3) & (rec_msgs_{vu}^{\mathsf{sm}}) & = & (rec_msgs_{vu}^{\mathsf{sim}})\\
(\phi_4) & r^{\mathsf{prot}} = r_{uu}^{\mathsf{prot}} & \Leftrightarrow & r^{\mathsf{sim}} = r_{uu}^{\mathsf{sim}}\\
(\phi_5) & r^{\mathsf{sim}} = r_{uu}^{\mathsf{sim}} & \Rightarrow & \tau \in [\tau_u^{\mathsf{spec}} - \Delta, \tau_u^{\mathsf{spec}} + \Delta]\\
(\phi_6) & r^{\mathsf{sim}} = r_{uu}^{\mathsf{sim}} & \Rightarrow & \begin{cases} \tau = \tau_u^{\mathsf{spec}} \vee \\ v \in crpt_{\tau}^{\mathsf{spec}} \vee \\ u \in crpt_{\tau_u^{\mathsf{spec}}}^{\mathsf{spec}} \end{cases}\\
(\phi_7) & i \notin rec_msgs_{vu}^{\mathsf{sim}} & \Rightarrow & j \notin rec_msgs_{vu}^{\mathsf{spec}}\\
(\phi_8) & m & = & m'
\end{array}$$

It is obvious that these relations hold for the initial states of the two collections. It remains to show that for every input of the form $(\mathsf{snd\_init})$, $(\mathsf{send}, \cdot, \cdot)$, $(\mathsf{rec\_init}, \cdot)$, $(\mathsf{deliver\_blindly}, \cdot, \cdot)$, $(\mathsf{clock\_tick})$, $(\mathsf{corrupt})$, and (leave), the relations guarantee identical outputs and remain valid for the corresponding next state. We first investigate the outputs of the two collections.

It follows directly by the specification of SIM and the relations ϕ_1 and ϕ_2, that M^{hybrid} and M^{ideal} produce identical outputs, whenever an input of the form $(\mathsf{snd\_init})$, $(\mathsf{send}, \cdot, \cdot)$, or $(\mathsf{rec\_init}, \cdot)$ occurs.

We now show that M^{hybrid} and M^{ideal} produce identical outputs whenever A^{hybrid} decides to schedule the delivery of a message, i.e., whenever an input $(\mathsf{deliver\_blindly}, v, i)$ occurs at $\mathsf{from\_Adv}'_u?$. If $deliver_{vu}^{\mathsf{sm}}[i] = (\mathsf{open}, r)$ or $deliver_{vu}^{\mathsf{sm}}[i] = (\mathsf{ack}, r'_1, r'_2)$, then the claim follows directly by the specification of the filtering protocol, and the assumption that the run is simple. If $deliver_{vu}^{\mathsf{sm}}[i] = (\mathsf{tx}, m', r^{\mathsf{prot}})$, we argue as follows. The relation ϕ_3 implies that the input of the adversary causes $\mathsf{TH}_n^{\mathsf{sm}}$ to output a value m at port $\mathsf{out}'_u!$ if and only if the same input causes $\mathsf{TH}_n^{\mathsf{sim,sm}}$ to output some value at port $\mathsf{out}_u'^{\mathsf{sim}}!$. Relation ϕ_4 implies that $\mathsf{M}_u^{\mathsf{prot}}$ subsequently outputs the value m' at port $\mathsf{out}_u!$ if and only if $\mathsf{M}_u^{\mathsf{sim}}$ subsequently outputs some value at port $\mathsf{out}_u^{\mathsf{sim}}!$. Relations ϕ_5, ϕ_6, and ϕ_7 imply that if $\mathsf{M}_u^{\mathsf{sim}}$ outputs a value at $\mathsf{out}_u^{\mathsf{sim}}!$, then $\mathsf{TH}_{\Delta,n}^{\mathsf{spec}}$ subsequently outputs a value m at $\mathsf{out}_u!$. And finally, ϕ_8 guarantees that $m' = m$.

For the remaining inputs $(\mathsf{clock\_tick})$, $(\mathsf{corrupt})$, and (leave) it follows directly by the specification of SIM that the collections M^{hybrid} and M^{ideal} produce identical outputs. It remains to show that the relations are invariants.

For the relations ϕ_1, ϕ_2, ϕ_3 and ϕ_7, ϕ_8, this follows immediately by the specification of SIM and the construction of A^{ideal}. For ϕ_4, ϕ_5, and ϕ_6, we argue as follows.

Considering only simple runs, relation ϕ_4 can only be invalidated by an input $(\mathsf{clock\_tick})$ at $\mathsf{clk}_u?$, or by an input (send, m, u) at $\mathsf{in}_v?$. In the first case, r_{uu}^{prot} is set to a random value uniformly chosen from $\{0,1\}^k$. If k is very large, then this implies that for every $(\mathsf{tx}, m, r^{\mathsf{prot}}) \in deliver_{vu}^{\mathsf{sm}}$, it holds that $r^{\mathsf{prot}} \neq r_{uu}^{\mathsf{prot}}$.[2] The same argument implies that for every $(\mathsf{tx}, (j, \mathsf{pad}), r^{\mathsf{sim}}) \in deliver_{vu}^{\mathsf{sim}}$ it holds that $r^{\mathsf{sim}} \neq r_{uu}^{\mathsf{sim}}$; thus, ϕ_4 remains valid. In the second case, we argue as follows. By the assumption that ϕ_4 is valid before the input occurs, we only have to show that ϕ_4 holds for the entries $(\mathsf{tx}, m, r^{\mathsf{prot}})$ and $(\mathsf{tx}, (j, \mathsf{pad}), r^{\mathsf{sim}})$ that are added to $deliver_{vu}^{\mathsf{sm}}$ and $deliver_{vu}^{\mathsf{sim}}$, respectively, when the input (send, m, u) occurs. By the specification of the filtering protocol, these entries satisfy $r^{\mathsf{prot}} = r_{vu}^{\mathsf{prot}} \wedge r^{\mathsf{sim}} = r_{vu}^{\mathsf{sim}}$. Hence, it suffices to show that $r_{vu}^{\mathsf{prot}} = r_{uu}^{\mathsf{prot}} \Leftrightarrow r_{vu}^{\mathsf{sim}} = r_{uu}^{\mathsf{sim}}$ is an invariant. But this follows directly from the fact that $Sys_{\Delta,n}^{\mathsf{psn,hybrid}}$ and $Sys_{\Delta,n}^{\mathsf{sim,psn,hybrid}}$ interact with the same machines A^{hybrid} and H, and by the "Ideal-Hybrid" transition rule of SIM.

Considering only simple runs, relation ϕ_5 can only be invalidated by inputs $(\mathsf{clock\_tick})$ at $\mathsf{clk}_u?$, and (send, m, u) at $\mathsf{in}_v?$. In the first case, it follows by an analogous argument as given for ϕ_4 that the relation ϕ_5 is preserved. In the second case, we argue as follows. By the assumption that ϕ_5 holds before the input occurs, we only have to show that ϕ_5 holds for the entry (m, τ) that is added to $deliver_{vu}^{\mathsf{spec}}$ when the input (send, m, u) occurs. By the specification of $\mathsf{TH}_{\Delta,n}^{\mathsf{spec}}$, this entry satisfies $\tau = \tau_v^{\mathsf{spec}}$. By the assumption on the clock shifts, $\tau_v^{\mathsf{spec}} \in [\tau_u^{\mathsf{spec}} - \Delta, \tau_u^{\mathsf{spec}} + \Delta]$, and hence, $\tau \in [\tau_u^{\mathsf{spec}} - \Delta, \tau_u^{\mathsf{spec}} + \Delta]$.

Considering only simple runs, relation ϕ_6 can only be invalidated by inputs $(\mathsf{clock\_tick})$ at $\mathsf{clk}_u?$, and (send, m, u) at $\mathsf{in}_v?$. In the

[2] More precisely, the probability that $r^{\mathsf{prot}} \neq r_{uu}^{\mathsf{prot}}$ holds can be made arbitrarily close to 1 by choosing k large enough. This follows from the assumption that the running time of every machine is bounded by a fixed polynomial in k.

first case, it follows by an analogous argument as given for ϕ_4 that the relation ϕ_6 remains valid. In the second case, we argue as follows. By the assumption that ϕ_6 is valid before the input occurs, we only have to show that ϕ_6 holds for the entries $(\mathsf{tx}, (j, \mathsf{pad}), r^{\mathsf{sim}})$ and (m, τ) that are added to $deliver^{\mathsf{sim}}_{vu}$, and $deliver^{\mathsf{spec}}_{vu}$, respectively, when the input (send, m, u) occurs. By the specification of $\mathsf{TH}^{\mathsf{spec}}_{\Delta,n}$, these entries satisfy $r^{\mathsf{sim}} = r^{\mathsf{sim}}_{vu} \; \wedge \tau = \tau^{\mathsf{spec}}_{v}$. Hence, it suffices to show that

$$(r^{\mathsf{sim}}_{vu} = r^{\mathsf{sim}}_{uu}) \Rightarrow (\tau^{\mathsf{spec}}_{v} = \tau^{\mathsf{spec}}_{u} \vee v \in crpt^{\mathsf{spec}}_{\tau^{\mathsf{spec}}_{v}} \vee u \in crpt^{\mathsf{spec}}_{\tau^{\mathsf{spec}}_{u}}).$$

This is equivalent to showing that

$$(r^{\mathsf{sim}}_{vu} = r^{\mathsf{sim}}_{uu} \wedge u \notin crpt^{\mathsf{spec}}_{\tau^{\mathsf{spec}}_{u}} \; \wedge \; v \notin crpt^{\mathsf{spec}}_{\tau^{\mathsf{spec}}_{v}}) \Rightarrow (\tau^{\mathsf{spec}}_{v} = \tau^{\mathsf{spec}}_{u}).$$

We argue as follows. By the specification of the filter protocol, $r^{\mathsf{sim}}_{vu} = r^{\mathsf{sim}}_{uu}$ implies that $\mathsf{M}^{\mathsf{sim}}_{v}$ has received during phase τ^{spec}_{v} a message $(\mathsf{ack}, r'_1, r'_2)$ from $\mathsf{M}^{\mathsf{sim}}_{u}$, where $r'_1 = r^{\mathsf{sim}}_{vv}$ and $r'_2 = r^{\mathsf{sim}}_{uu}$. By the fact that the nonces are reset at the beginning of every phase, $r'_2 = r^{\mathsf{sim}}_{uu}$ implies that $\mathsf{M}^{\mathsf{sim}}_{u}$ has sent this acknowledge message during its phase τ^{spec}_{u}. This only happens if it receives a message $(\mathsf{open}, r', \tau')$ during phase τ^{spec}_{u}, such that $r' = r^{\mathsf{sim}}_{vv}$ (by $r'_1 = r^{\mathsf{sim}}_{vv}$), and $\tau' = \tau^{\mathsf{spec}}_{u}$ (by $u \notin crpt^{\mathsf{spec}}_{\tau^{\mathsf{spec}}_{u}}$). By the fact that nonces are reset at the beginning of every phase, $r' = r^{\mathsf{sim}}_{vv}$ implies that $\mathsf{M}^{\mathsf{sim}}_{v}$ has sent this open message during phase τ^{spec}_{v}. By $v \notin crpt^{\mathsf{spec}}_{\tau^{\mathsf{spec}}_{v}}$ it follows that $\tau' = \tau^{\mathsf{spec}}_{v}$, and hence, $\tau^{\mathsf{spec}}_{v} = \tau^{\mathsf{spec}}_{u}$.

Liveness. To prove the second part of Theorem 4.1, we have to show that for any τ, and any two users u, v honest throughout the phases $[\tau - \Delta, \tau + \Delta]$, the following holds: If u sends a message m to v during phase τ, then v receives the message m during phase τ, provided that $Sys^{\mathsf{sm,ideal}}_{n}$ is live within $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$.

We first show that T_u computes during phase τ a value $r^{\mathsf{prot}}_{uv} \neq \perp$, if $Sys^{\mathsf{sm,ideal}}_{n}$ is live within $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$. We then show that this value corresponds to r^{prot}_{vv} of T_v during phase τ. Liveness of $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$ follows then directly from the specification of the send and receive transition for T_u, provided that $Sys^{\mathsf{sm,ideal}}_{n}$ is live within $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$.

1. By the liveness of $Sys^{\mathsf{sm,ideal}}_{n}$, T_v receives during phase τ the message $(\mathsf{open}, r^{\mathsf{prot}}_{uu}, \tau)$ sent by T_u during phase τ. Hence, by the specification of the handshake protocol, T_v sends a message $(\mathsf{ack}, r^{\mathsf{prot}}_{uu}, r^{\mathsf{prot}}_{vv})$ to T_u. By the liveness of $Sys^{\mathsf{sm,ideal}}_{n}$, this message is delivered to T_v during its phase τ. Hence, if r^{prot}_{uv} is still equal to $\perp$, $\mathsf{TH}^{\mathsf{sm}}_{n}$ will set it to r^{prot}_{vv}.
2. Towards a contradiction, suppose that the value r^{prot}_{uv} computed by T_u during u's phase τ does not correspond to the value r^{prot}_{vv} of T_v during v's phase τ. Together with the specification of the filtering protocol, this implies that r^{prot}_{uv} corresponds to the value r^{prot}_{vv} of T_v during some phase $\tilde{\tau} \neq \tau$. By the specification of the filtering protocol, T_u would therefore accept during phase τ messages sent by T_v during phase $\tilde{\tau}$. But if u and v are honest throughout $[\tau - \Delta, \tau + \Delta]$, such a behavior contradicts the security of $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$ which was proven in the previous section.

4.4 Deriving the Real System

The real system $Sys^{\mathsf{psn,real}}_{\Delta,n}$ for proactive secure message transmission can now be derived from the hybrid system $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$ by replacing the ideal specification for secure message transmission with the concrete implementation $Sys^{\mathsf{sm,real}}_{n}$ sketched in Section 4.1. Note that in the resulting real system $Sys^{\mathsf{psn,real}}_{\Delta,n}$, the machines $\mathsf{M}^{\mathsf{sm}}_{u}$ of $Sys^{\mathsf{sm,real}}_{n}$ always behave as specified, and cannot be affected by the adversary. Practically, one would implement the machines T_u and $\mathsf{M}^{\mathsf{sm}}_{u}$ of a user u on the same secure co-processor.

The security of $Sys^{\mathsf{psn,real}}_{\Delta,n}$ follows from the security of $Sys^{\mathsf{psn,hybrid}}_{\Delta,n}$ and $Sys^{\mathsf{sm,real}}_{n}$, and the composition theorem. Furthermore, by Theorem 4.1, a run of $Sys^{\mathsf{psn,real}}_{\Delta,n}$ is live, provided it is live at $Sys^{\mathsf{sm,real}}_{n}$. This holds if the asynchronous insecure network underlying $Sys^{\mathsf{sm,real}}_{n}$ delivers messages as follows: For every phase τ, and every two users u and v honest throughout phases $[\tau - \Delta, \tau + \Delta]$, it holds that if (m) occurs at $\mathsf{net}_{uv}!$ during u's phase τ, then (m) occurs at $\mathsf{net}_{vu}?$ during v's phase τ. Note that if phases are long compared to message delays, and if Δ is an upper bound on the clock shift, then such a loose synchronization of the underlying network is easy to guarantee.

5. ALTERNATIVE MACHINE MODELS

In this section, we discuss two natural variations of the structure of the real machines $\mathsf{M}^{\mathsf{prot}}_{u}$. They differ in the way the local clock Ω_u is connected to the secure module T_u. In particular, we first consider a machine structure in which this connection is authentic *and* reliable, and then investigate a structure in which there is no such connection at all. The first case is denoted as *clocked co-processor* and illustrated in Figure 7, whereas the second case is denoted as *stand-alone co-processor* and illustrated in Figure 8.

Clocked Co-Processors. This variation can be seen as the "strongest" hardware assumption in the sense that the secure co-processor must essentially have an internal clock. Such co-processors exist, for example the IBM 4758 [13], but at higher costs than a simple smart card or TCPA chip.

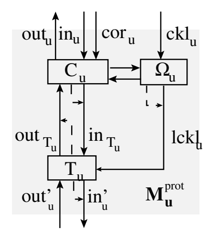

Figure 7: Clocked co-processor T_u

From a theoretical point of view, this model has the advantage that we can define a simple protocol for the filtering system $Sys^{\mathsf{filterA}}_{\Delta,n}$ consisting of such machines, such that the corresponding hybrid system $Sys^{\mathsf{psn,hybridA}}_{\Delta,n}$ implements a proactive secure network with $\Delta \geq 0$ for *any* adversary, independent of the local clock shift. This is captured by the following theorem (without proof).

Theorem 5.1 *For all $n, \Delta \in \mathbb{N}$, $Sys^{\mathsf{psn,hybridA}}_{\Delta,n} \geq Sys^{\mathsf{psn,ideal}}_{\Delta,n}$. Moreover, a run of $Sys^{\mathsf{psn,hybridA}}_{\Delta,n}$ is live, provided that it is live at $Sys^{\mathsf{sm,ideal}}_{n}$.* □

The idea of the filtering system is as follows. C_u acts transparently, while T_u keeps track of its local phase by observing its clock input port $\mathsf{clk}_u?$. It then tags all messages sent to the other servers with the current phase, and accepts only received messages that are tagged with the current phase.

Stand-Alone Co-Processors. This variation can be seen as the "weakest" hardware assumption. It only depends on a secure co-processor such as a TCPA chip [16] or a smart card [14], and on a local clock that cannot be reset by software. No specific "hardware" connection between clock and co-processor is required. On some motherboards, jumpers can be set to prevent software from resetting the local clock; otherwise, one could also rely on a system administrator to adjust the local clock at the beginning of every phase.

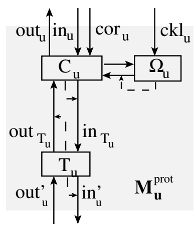

Figure 8: Stand-alone co-processor T_u

The difficulty that one faces in this model is that the secure module T_u can only receive $(\mathsf{clock\_tick}, \tau)$ messages from Ω_u through C_u. Hence, if C_u is corrupted, T_u may not detect a phase change at all. As a consequence, one cannot build a secure implementation of the specification $Sys_{\Delta,n}^{\mathsf{psn,ideal}}$ in this model due to the following argument. Suppose the adversary corrupts a user u during an arbitrary number of subsequent phases $\tau_1, \ldots, \tau_l$, and suppose T_u accepts some message m during phase τ_1. By the fact that the user u is corrupted during all phases $\tau_1, \cdots, \tau_l$, the module T_u may never change its state throughout these phases, and therefore would accept the message m during *any* such phase. This contradicts the definition of $\mathsf{TH}_{\Delta,n}^{\mathsf{spec}}$, which specifies that any message may only be accepted during an interval of at most $2\Delta + 1$ phases.

One can bypass the above difficulties by relaxing the specification of $\mathsf{TH}_{\Delta,n}^{\mathsf{spec}}$ in the following sense. If at some point, a message can be delivered to a currently corrupted user according to the original specification, then the relaxed specification allows the message to be delivered to this user during any phase when it is corrupted. In fact, such a relaxed specification would still be sufficient for most applications of proactive secure message transmission. Furthermore, it can be shown that there exists a real system that is at least as secure as this relaxed system for any $\Delta \geq 0$, provided that Δ is an upper bound on the clock shift. However, in this system, a user needs at least one phase to re-establish communication after a corruption. Hence, liveness is only guaranteed for $\Delta > 0$.

6. REFERENCES

[1] M. Backes, C. Jacobi, and B. Pfitzmann. Deriving cryptographically sound implementations using composition and formally verified bisimulation. In *Proc. Formal Methods Europe '02 (FME)*, pages 310–329, 2002.

[2] B. Barak, S. Halevi, A. Herzberg, and D. Naor. Clock synchronization with faults and recoveries. In *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 133–142, 2000.

[3] C. Cachin, A. Lysyanskaya, K. Kursawe, and R. Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proc. 9th ACM Conference on Computer and Communication Secuirty (CCS)*, pages 88–97, 2002.

[4] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[5] R. Canetti, R. Gennaro, A. Herzberg, and D. Naor. Proactive security: Long-term protection against break-ins. *RSA Laboratories' CryptoBytes*, 3(1):1–8, 1997.

[6] R. Canetti, S. Halevi, and A. Herzberg. Maintaining authenticated communication in the presence of break-ins. *Journal of Cryptology*, 13(1):61–106, 2000.

[7] M. Castro and B. Liskov. Proactive recovery in a Byzantine-fault-tolerant system. In *Proc. Fourth Symp. Operating Systems Design and Implementation (OSDI)*, pages 273–287, 2000.

[8] Y. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, 1994.

[9] Y. Frankel, P. Gemmel, P. Mackenzie, and M. Yung. Proactive RSA. In *Advances in Cryptology: CRYPTO '97*, pages 440–452, 1997.

[10] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer, 1999.

[11] A. Herzberg, M. Jakobson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *Proc. 4th ACM Conference on Computer and Communication Secuirty (CCS)*, pages 100–110, 1997.

[12] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or how to cope with perpetual leakage. In *Advances in Cryptology: CRYPTO '95*, pages 339–352, 1995.

[13] IBM PCI 4758 cryptographic coprocessor. `http://www-3.ibm.com/security/cryptocards/`.

[14] Java card technology. `http://java.sun.com/products/javacard/`.

[15] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 51–59, 1991.

[16] S. Pearson, B. Preneel, and G. Proudler. *Trusted Computing Platforms: TCPA Technology in Context*. Prendice Hall PTR, 2003.

[17] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001.

[18] L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A secure distributed on-line certification authority. Technical Report 2000-1828, Dept. of Computer Science, Cornell University, 2000.