

Computational Probabilistic Non-Interference*

Michael Backes¹, Birgit Pfitzmann²

¹ IBM Zurich Research Laboratory, Rüschlikon, Switzerland
e-mail: mbc@zurich.ibm.com

² IBM Zurich Research Laboratory, Rüschlikon, Switzerland
e-mail: bpf@zurich.ibm.com

The date of receipt and acceptance will be inserted by the editor

Abstract Information flow and non-interference are popular concepts for expressing confidentiality and integrity properties. We present the first general definition of probabilistic non-interference in reactive systems that includes a computational case. This case is essential to cope with real cryptography, since non-interference properties can usually only be guaranteed if the underlying cryptographic primitives have not been broken. This might happen, but only with negligible probability. We show that our non-interference definition is maintained under simulatability, the notion of secure implementation of modern cryptography. This allows secure composition of systems and yields a general strategy for including cryptographic primitives in information-flow proofs. As an example we study a cryptographic firewall guarding two honest users from their environment.

1 Introduction

Information flow and non-interference are powerful concepts for expressing the confidentiality and integrity requirements that a program or a cryptographic protocol should fulfill. The term “absence of information flow” illustrates the confidentiality view: One requires that no information flows from a secret system part or data item into a less secure system part or data item, or from users with high confidentiality needs to less trusted users. The term “non-interference” illustrates the integrity view: One requires that no untrusted information should interfere with data items with higher correctness needs, or that highly trusted users are not misinformed or bothered by such information.

The concept of information flow was first investigated for secure operating systems by Bell and LaPadula [8] and by Denning [17]. Subsequently, various definitions have been proposed that rigorously specify when information flow is

* A preliminary version of this paper appeared at ESORICS 2002.

considered to occur. The first, named *non-interference*, was introduced by Goguen and Meseguer [22,23] to analyze the security of computer systems. Their work was limited to deterministic systems; nevertheless, subsequent work is still based on their ideas. Afterwards, research focused on non-deterministic systems. The main distinction is between probabilistic and possibilistic behaviors. Beginning with Sutherland [64], the possibilistic case was treated in [45,69,49,71,40], while definitions handling probabilistic and information-theoretic behaviors were proposed by Gray [29,30] and McLean [47]. Clark et. al. showed in [14] that possibilistic information-flow analysis can be used to check for probabilistic interference in certain cases.

For reasoning about real cryptographic systems, we need probabilistic behaviors. On this probabilistic side, Gray's definition of "Probabilistic Non-Interference" of reactive systems stands out. It is closely related to the perfect case of our definition, but it does not cover computational aspects, which are essential for reasoning about systems using real cryptographic primitives. As an example, consider an arbitrary public-key encryption scheme. Obviously, an adversary with unlimited computing power can break the scheme by computing all possible encryptions of every plaintext and comparing the results with a given ciphertext. Moreover, even polynomially bounded adversaries may have a very small, so-called *negligible* probability of success, e.g., by trying just a few encryptions of a few plaintexts. Thus, cryptographic definitions usually state that every polynomially bounded adversary can only achieve its goal with a negligible probability. We present the first general definition of non-interference for this *computational* case. Thus, non-interference properties can be expressed for reactive systems containing arbitrary cryptographic primitives, which is of great importance for extensible systems like applets, kernel extensions, mobile agents, virtual private networks, etc.

In contrast to other definitions, we do not abstract from cryptographic details and probabilism a priori, e.g., by using the common Dolev-Yao abstraction [19] or special type systems, but include the computational variant in our definition. This enables sound reduction proofs with respect to the security definitions of the included cryptographic primitives, such as an underlying public-key encryption scheme. This means that every possibility to break the non-interference properties of the system can be used to break the underlying cryptography. Moreover, we show that our definition behaves well under simulatability, which is the common concept in modern cryptography for defining a notion of secure implementation. We show that non-interference properties proved for an abstract specification automatically carry over to a concrete implementation if that implementation is correct in the sense of simulatability. This theorem is essential since it enables modular proofs of large systems, i.e., proofs done for ideal systems not containing any probabilism simply carry over to their real cryptographic counterparts. Moreover, properties of these ideal systems could quite easily be proved with machine assistance, so our theorem provides a link between cryptography and formal proof tools for non-interference.

As an example, we present a cryptographic firewall which enables two honest users to communicate with each other, but guards them from their environment, and prove its desired non-interference property.

Further related literature. An important application of information flow is the static analysis of program code with respect to certain privacy requirements. This problem was first treated by Denning [18] using flow graphs on I/O variables. Recently, type-based systems have been proposed [66,63,67,62] for detecting and eliminating information flow in different kinds of languages. Some of these type systems were proven correct by Sabelfeld and Sands [59,60], who presented a semantic characterization of probabilistic bisimulation and used it to express non-interference for multi-threaded and sequential programs. Formal soundness proofs of security type systems in general have been pioneered by Volpano et al. [68]. Mantel and Sabelfeld [44] investigated the integration of security properties of programming languages and abstract-level properties of information flow, providing an interesting overview of how models of different security properties can be combined to increase the relative power of their analysis. Moreover, a tool for automatically checking the information flow in concurrent languages has been developed by Focardi and Gorrieri [20] for a variety of information-flow models.

Today, there is no general definition of the absence of information flow, but several of them coexist. Every definition has advantages and disadvantages, and which one to take depends on the application area. Many of these definitions have the shortcoming that they are not preserved under refinement, cf. [47,33,42]; moreover, special care must be taken concerning the composition of secure system when reasoning about non-interference, see [45,34,46,48,50,43,16]. Furthermore, many definitions of non-interference were overly restrictive, so that many useful systems did not fulfill them. This problem is often tackled by downgrading certain information, so that it may then leak from the system, see [53,72]. In some cases, the amount of leaked information can be rigorously defined using information-theoretic techniques [52,35].

Some recent research also investigated non-interference properties involving real cryptographic primitives. Laud [36] presented a sequential language for which he expressed real computational secrecy. Besides our work, this paper contains the only definition of non-interference including a computational case. However, the definition is non-reactive, i.e., it does not comprise continuous interaction between users, an adversary, and the system. This is a severe restriction on the set of security systems that can be handled. Further, encryption is the only primitive covered there so far, i.e., other important primitives like authentication, pseudo-number generators, etc. are not considered. Our definition is reactive and comprises arbitrary cryptographic primitives. Volpano [65] investigated conditions for safely using one-way functions in a programming language, but his underlying definition does not express non-interference, but the secrecy of a specific secret. Abadi and Blanchet [1] introduced type systems where asymmetric communication primitives, especially public-key encryption, can be handled, but these primitives are only relative to a Dolev-Yao abstraction [19], i.e., the primitives are idealized so

that no computational non-interference definition is needed. For a discussion why the Dolev-Yao abstraction is not justified by current cryptography, see [54].

Outline of the paper. In Section 2 we briefly review the underlying model of asynchronous reactive system. The original contributions are presented in Sections 3 to 5. In Section 3 we present our definition of non-interference. This includes extending the underlying model to multiple users. In Section 4 we show that our definition behaves well under simulatability, i.e., secure implementation does not change the non-interference properties. In Section 5 we present an abstract specification and an implementation of a cryptographic firewall guarding two honest users from their environment, and we prove that they fulfill our definition of non-interference. We conclude with a summary of our results and open issues for future research.

2 General System Model for Reactive Systems

In this section we recapitulate the model for probabilistic reactive systems introduced by Pfitzmann and Waidner in [55,6]. Several definitions will only be sketched, whereas those that are important for understanding our new definitions and proofs are given in full detail. All other details can be looked up in the original paper.

In particular, we repeat the scheduling model in full detail because scheduling is important for achieving non-interference: We cannot let the adversary schedule everything and hence need special schedulers below. Cryptographic asynchronous systems need two specific scheduling aspects compared with other asynchronous system models: Schedulers are “normal” system machines, so that they schedule with realistic knowledge, and different channels may be scheduled by different machines, e.g., so that local submachines can be represented.

2.1 General System Model

Systems mainly consist of several interacting machines. Usually we consider real systems containing a set \hat{M} of machines $\{M_1, \dots, M_n\}$, one for each user u from a set $\mathcal{M} = \{1, \dots, n\}$, and ideal systems containing only one machine $\{TH\}$.

Communication between different machines is done via ports. Inspired by the CSP-notation [32], we write input and output ports as $p?$ and $p!$, respectively. The input and output ports in a port set P are written $\text{in}(P)$ and $\text{out}(P)$, respectively. Connections are defined by naming convention: port $p!$ sends messages to $p?$. To achieve asynchronous timing, a message is not immediately delivered to its recipient, but first stored in a special machine \tilde{p} called a buffer, where it waits to be scheduled. This can be done by the machine with the unique clock-out port $p^{\leftarrow!}$. To schedule the i -th message of buffer \tilde{p} , it outputs i at $p^{\leftarrow!}$, see Figure 1. The buffer then delivers the i -th message and removes it from its internal list. Most buffers are scheduled either by a specific master scheduler or by the adversary, i.e., one of those has the clock-out port. Ports $p!$ and $p?$, in contrast to the other four port types

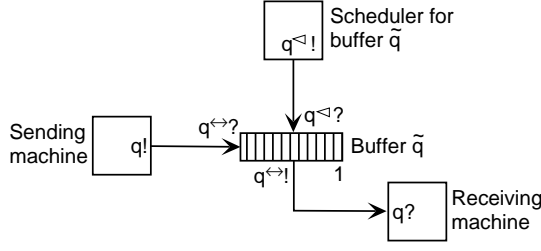


Fig. 1 Naming of ports around a buffer. Later one can often abstract from the buffer and simply regard $q!$ and $q?$ as asynchronously connected.

occurring at the buffers, are called *simple*, and a *simple machine* has only simple ports and clock-out ports.

The precise definition of machines is a variant of probabilistic state-transition machines, similar to probabilistic I/O automata as sketched by Lynch [39]. If a machine is switched, it reads an input tuple at its input ports and performs its transition function. This yields a new state and an output tuple. A probabilistic transition function actually describes a finite distribution over the pairs of a new state and an output tuple. Furthermore, each machine has bounds on the length of considered inputs. This allows time bounds independent of the environment.

Definition 1 (*Machines*) A machine (for an alphabet Σ) is a tuple

$$M = (\text{name}_M, \text{Ports}_M, \text{States}_M, \delta_M, l_M, \text{Ini}_M, \text{Fin}_M)$$

of a machine name $\text{name}_M \in \Sigma^+$, a finite sequence Ports_M of ports, a set $\text{States}_M \subseteq \Sigma^*$ of states, a probabilistic state-transition function δ_M , a length function $l_M : \text{States}_M \rightarrow (\mathbb{N} \cup \{\infty\})^{|\text{in}(\text{Ports}_M)|}$, and sets $\text{Ini}_M, \text{Fin}_M \subseteq \text{States}_M$ of initial and final states. Its input set is $\mathcal{I}_M := (\Sigma^*)^{|\text{in}(\text{Ports}_M)|}$; the i -th element of an input tuple denotes the input at the i -th in-port. Its output set is $\mathcal{O}_M := (\Sigma^*)^{|\text{out}(\text{Ports}_M)|}$. The empty word, ϵ , denotes no in- or output at a port. δ_M maps each pair $(s, I) \in \text{States}_M \times \mathcal{I}_M$ to a finite distribution over $\text{States}_M \times \mathcal{O}_M$.

If $s \in \text{Fin}_M$ or $I = (\epsilon, \dots, \epsilon)$, then $\delta_M(s, I) = (s, (\epsilon, \dots, \epsilon))$ deterministically. Inputs are ignored beyond the length bounds, i.e., $\delta_M(s, I) = \delta_M(s, I \upharpoonright_{l_M(s)})$ for all $I \in \mathcal{I}_M$, where $(I \upharpoonright_{l_M(s)})_i := I_i \upharpoonright_{l_M(s)_i}$ for all i .

In the text, we often write “ M ” also for name_M . In the following, the initial states of all machines are a security parameter $k \in \mathbb{N}$ in unary representation.

We only briefly state here that these machines have a natural realization as probabilistic Turing machines, which is used to define runtimes [6].

A *collection* \hat{C} of machines is a finite set of machines with pairwise different machine names and disjoint sets of ports. All machines start with the same security parameter k . Let further $\text{ports}(\hat{C})$ denote the set of all ports of all machines in \hat{C} . The *completion* $[\hat{C}]$ of a collection \hat{C} consists of all machines of \hat{C} and the buffers needed for all the ports in \hat{C} . The *free* ports $\text{free}(\hat{C})$ in a collection are those to which no other port in the collection connects. A collection \hat{C} is *closed* if its completion $[\hat{C}]$ has no free ports except a special master clock-in port $\text{clk}^{\triangleleft?}$.

The machine with this port is the *master scheduler*, to which control returns as a default.

For a closed collection, a probability space of *runs* (sometimes called traces or executions) is defined. The machines switch sequentially, i.e., there is exactly one active machine M at any time, called the *current scheduler*. If this machine has clock-out ports, it can select the next message to be scheduled as explained above. If that message exists, it is delivered by the buffer and the recipient is the next active machine. If M tries to schedule multiple messages, only one is taken. If it schedules none or the message does not exist, the master scheduler is activated.

Formally, runs are sequences of *steps* defined as follows (where the state-transition function of buffers is as explained above).

Definition 2 (Runs) Given a closed collection \hat{C} with master scheduler X and a security parameter k , the probability space of *runs* is defined inductively by the following algorithm. It has variables r for the run under construction and M_{CS} for the current scheduler, and treats each port as a variable over Σ^* . Here r is an initially empty list, M_{CS} a machine name initialized with X , and all port variables are initially ϵ except for $\text{clk}^{q?} := 1$. Probabilistic choices only occur in Phase 1.

1. *Switch current scheduler*: Switch machine M_{CS} , i.e., set $(s', O) \leftarrow \delta_{M_{CS}}(s, I)$ for its current state s and in-port values I . Then assign ϵ to all in-ports of M_{CS} .
2. *Termination*: If X is in a final state, the run stops.
3. *Buffer new messages*: For each simple out-port $q!$ of M_{CS} , switch buffer \tilde{q} with input $q^{\leftrightarrow?} := q!$, cf. Figure 1. Then assign ϵ to all these ports $q!$ and $q^{\leftrightarrow?}$.
4. *Clean up scheduling*: If at least one clock out-port of M_{CS} has a value $\neq \epsilon$, let $q^{\leftarrow!}$ denote the first such port and assign ϵ to the others. Otherwise let $\text{clk}^{q?} := 1$ and $M_{CS} := X$ and go back to Phase 1.
5. *Deliver scheduled message*: Switch buffer \tilde{q} with input $q^{\leftarrow?} := q^{\leftarrow!}$ (see Figure 1), set $q^? := q^{\leftarrow!}$ and then assign ϵ to all ports of \tilde{q} and to $q^{\leftarrow!}$. Let $M_{CS} := M'$ for the unique machine M' with $q^? \in \text{ports}(M')$. Go back to Phase 1.

Whenever a machine (this may be a buffer) with name name_M is switched from (s, I) to (s', O) , we append a *step* $(\text{name}_M, s, I, s', O)$ to the run r for $I' := I \upharpoonright_{l_M(s)}$, except if s is final or $I' = (\epsilon, \dots, \epsilon)$. This gives a family of random variables

$$\text{run}_{\hat{C}} = (\text{run}_{\hat{C}, k})_{k \in \mathbb{N}}.$$

For a number $l \in \mathbb{N}$, the l -step *prefix* of a run r is the list of the first l steps.

Next we define what a machine (e.g., an untrusted user in a non-interference definition) sees in a run and what events happen at a set of ports, and the probabilities of such views and events.

Definition 3 (Views and Restrictions to Ports) The *view* of a set of machines \hat{M} in a run r is the subsequence of all steps $(\text{name}_M, s, I, s', O)$ where name_M is the name of a machine $M \in \hat{M}$. The *restriction* $r \upharpoonright_S$ of a run to a set S of ports is a sequence derived as follows: First only retain the inputs and outputs, (I, O) , from

each step, and further restrict I and O to the ports in S . Then delete pairs where both I and O have become empty.

The corresponding families of random variables (in the probability space over the runs) are denoted by

$$\begin{aligned} \text{view}_{\hat{C}}(\hat{M}) &= (\text{view}_{\hat{C},k}(\hat{M}))_{k \in \mathbb{N}} \text{ and} \\ \text{run}_{\hat{C}} \upharpoonright_S &= (\text{run}_{\hat{C},k} \upharpoonright_S)_{k \in \mathbb{N}}. \end{aligned}$$

With an additional index l , we denote the $l(k)$ -step prefixes of the views and restrictions.

For a one-element set $\hat{M} = \{H\}$ we write $\text{view}_{\hat{C}}(H)$ for $\text{view}_{\hat{C}}(\{H\})$.

2.2 Security-specific System Model

For security purposes, we have to define how adversaries and honest users connect to specified machines of a collection. First, an adversary may take over parts of the initially intended machines. These machines are then absorbed into the adversary, and the remaining machines form a *structure*. Formally, a structure is a collection of machines in which one additionally distinguishes at which free ports honest users can connect and expect some reasonable service (e.g., message transport in a cryptographic firewall), and which ports are only used by adversaries. The former are the *specified ports* in the following definition. A valid honest user should neither try to connect to the remaining free ports of a structure, nor, for unique naming, have ports that already occur inside the structure. This is expressed by *forbidden ports*.

Depending on which machines the adversary has taken control of, we obtain different structures, and we call the set of all structures a *system*. Typically, a system is defined by means of a so-called intended structure and a trust model. The intended structure represents a benign world where each machine behaves as specified, and the trust model is then used to designate the potential sets of machines which are considered to be under control of the adversary. An example of the typical derivation of these structures from an intended structure and a trust model occurs in Section 5.

The ports connecting to a given port set P are expressed by the complement notation P^c , e.g., $q!^c = q^{\leftrightarrow?}$, $q^{\leftarrow!^c} = q^{\leftarrow?}$, $q^{\leftrightarrow!^c} = q^?$ in Figure 1, and vice versa.

Definition 4 (Structures and Systems)

- a) A *structure* is a pair (\hat{M}, S) where \hat{M} is a collection of simple machines called *correct machines*, and $S \subseteq \text{free}([\hat{M}])$ is called *specified ports*.
- b) If \hat{M} is clear from the context, let $\bar{S} := \text{free}([\hat{M}]) \setminus S$. We call $\text{forb}(\hat{M}, S) := \text{ports}(\hat{M}) \cup \bar{S}^c$ the *forbidden ports*.
- c) A *system* Sys is a set of structures. It is polynomial-time iff all machines in all its collections \hat{M} are polynomial-time.

A structure is completed to a *configuration* by adding machines H and A , modeling the joint honest users and the adversary. As explained above, H does not have certain ports. A connects to the remaining free ports of the structure. Both machines can interact, e.g., in order to model chosen-message attacks.

Definition 5 (*Configurations*)

- a) A *configuration* of a system Sys is a tuple $conf = (\hat{M}, S, H, A)$ where $(\hat{M}, S) \in Sys$ is a structure, H is a simple machine without forbidden ports, i.e., $ports(H) \cap forb(\hat{M}, S) = \emptyset$, and $\hat{C} := \hat{M} \cup \{H, A\}$ is a closed collection. For simplicity, we often write run_{conf} and $view_{conf}(\hat{M})$ instead of $run_{\hat{C}}$ and $view_{\hat{C}}(\hat{M})$
- b) The set of configurations is written $Conf(Sys)$. The subset of configurations with polynomial-time user H and adversary A is called $Conf_{poly}(Sys)$. The index $_{poly}$ is omitted if it is clear from the context.

3 Expressing Non-Interference

In this section we define non-interference for reactive systems as introduced in Section 2. Information flow properties consist of two components: a *flow policy* and a *definition of information flow*. Flow policies specify restrictions on the information flow within a system. They presuppose the existence of security domains S , between which information flow is either permitted or forbidden. Roughly, our flow policies are graphs with the users as nodes. In the definition of information flow, we want to express that there is no information flow from a user H_H to a user H_L iff the view of H_L does not allow to distinguish (perfectly or computationally) any behaviors of H_H .

3.1 Multi-Party Configurations

In Definition 5, all honest users were modeled by a single machine H . For expressing non-interference between two users based on their individual views, we must model different users as different user machines. Hence we first define multi-party configurations. The only difference to normal configurations is that we have a set U of user machines instead of the joint machine H .

Definition 6 (*Multi-Party Configurations*) A *multi-party configuration* of a system Sys is a tuple (\hat{M}, S, U, A) where $(\hat{M}, S) \in Sys$ is a structure, U is a set of machines called *users* without forbidden ports, i.e., $ports(U) \cap forb(\hat{M}, S) = \emptyset$, and the completion $\hat{C} := [\hat{M} \cup U \cup \{A\}]$ is a closed collection. The set of these configurations is denoted by $Conf^{mp}(Sys)$, those with polynomial-time users and a polynomial-time adversary by $Conf_{poly}^{mp}(Sys)$. We omit the indices mp and $poly$ if they are clear from the context.

Runs and views are automatically defined for multi-party configurations because they are defined for all closed collections, here \hat{C} .



Fig. 2 A Typical Flow-Policy Graph Consisting of High and Low Users.

3.2 Flow Policies

We start by defining flow policy graphs independent of our model.

Definition 7 (General Flow Policy) A general flow policy is a graph $\mathcal{F} = (\mathcal{S}, \rightsquigarrow)$ with a non-empty set \mathcal{S} and $\rightsquigarrow \subseteq \mathcal{S} \times \mathcal{S}$. We use infix notation, i.e., we write $s_1 \rightsquigarrow s_2$ for $(s_1, s_2) \in \rightsquigarrow$, and $s_1 \not\rightsquigarrow s_2$ for the negation. We demand $s \rightsquigarrow s$ for all $s \in \mathcal{S}$.

A general flow policy is *transitive* if the relation \rightsquigarrow is transitive.

Intuitively, $s_1 \rightsquigarrow s_2$ means that information may flow from s_1 to s_2 , and $s_1 \not\rightsquigarrow s_2$ means that it must not. Transitivity means that whenever indirect flow is possible from s_1 to s_2 via some other elements, direct flow from s_1 to s_2 is also permitted. In this paper, we only consider transitive flow policies.

Example 1 The set \mathcal{S} often has only two elements $\mathcal{S} = \{L, H\}$, called low and high users. The typical flow policy for this case is that information flow from high to low users is forbidden, i.e., $H \not\rightsquigarrow L$, while $L \rightsquigarrow L$, $L \rightsquigarrow H$, and $H \rightsquigarrow H$; see Figure 2.

This definition is quite general since it uses an arbitrary set \mathcal{S} . We now refine it to our model of reactive systems. The intuition is to define a graph on the users and the adversary. However, a definition over the machine set $U \cup \{A\}$ would depend on internal details of the user and adversary machines. Instead, we designate each user by the set of specified ports it connects to, and the adversary, who connects to the remaining free ports of each structure, by a symbol *adv*.

Thus our flow policies only depend on the specified ports of the structures of a system Sys .

Definition 8 (Flow Policy) A flow policy for a port set S is a general flow policy $(\Delta, \rightsquigarrow)$ where $\Delta = \Gamma \cup \{\text{adv}\}$ and $\Gamma = \{S_i \mid i \in \mathcal{I}\}$ is a partition of S for a finite index set \mathcal{I} .

A flow policy for a system Sys is a mapping \mathcal{F} that assigns each structure $(\hat{M}, S) \in Sys$ a flow policy $\mathcal{F}_{(\hat{M}, S)}$ for the set S of specified ports. We require that $\mathcal{F}_{(\hat{M}, S)}$ only depends on S , and also write \mathcal{F}_S .

Note that the adversary is explicitly included in the flow policy. This will be essential to capture that, e.g., the adversary does not learn any information from an honest user.

Some abbreviations will be useful.

Definition 9 (Additional Flow-Policy Notation) Given a flow policy \mathcal{F} for a system Sys , we write $\Delta_{(\hat{M}, S)}$, $\rightsquigarrow_{(\hat{M}, S)}$, $\Gamma_{(\hat{M}, S)}$, and $\mathcal{I}_{(\hat{M}, S)}$ for the components of

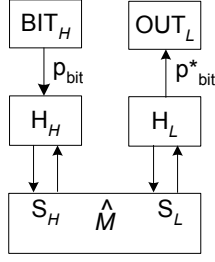


Fig. 3 Sketch of our Non-Interference Definition: H_L tries to guess a bit that H_H tries to transfer.

the flow policies $\mathcal{F}_{(\hat{M}, S)}$, and similarly Δ_S etc. If the structure or at least S is clear from the context, we even just write Δ etc.

We assume without loss of generality that $\text{adv} \notin \mathcal{I}$ for every occurring index set \mathcal{I} and always define $\mathcal{I}^{\text{adv}} := \mathcal{I} \cup \{\text{adv}\}$. Given a structure (\hat{M}, S) , we further write $S_{\text{adv}} := \bar{S}$ and sometimes identify adv with S_{adv} . Then the node set Δ of a flow policy $\mathcal{F}_{(\hat{M}, S)}$ is identified with a partition of $\text{free}([\hat{M}])$ with index set \mathcal{I}^{adv} .

Given a structure (\hat{M}, S) and a flow policy $(\Delta, \rightsquigarrow)$ for S , the relation $S_H \not\rightsquigarrow S_L$ for two port sets $S_H, S_L \in \Delta$ intuitively means that no information must flow from the user machines connected to S_H to the user machines connected to S_L in any configuration of this structure. With the identification we made, this also holds for the adversary, who is connected to the port set $S_{\text{adv}} = \bar{S}$.

3.3 Definition of Non-Interference

We now define the semantics of the non-interference relation $\not\rightsquigarrow$. Usually, expressing this semantics is the most difficult part of an information-flow definition. Given our underlying model, it is a bit easier because we already have definitions of runs, views, and indistinguishability. We first present the intuitive idea of the semantics and give the formal definition afterwards.

Figure 3 contains a sketch of our definition of non-interference between two users H_H and H_L ; one of them may be the adversary. Intuitively, H_H should not be able to transfer a bit, or even part of one, to the user H_L . This is modeled as follows. At the start of the run, a bit $b \in \{0, 1\}$ is chosen at random and input to H_H . Perfect non-interference means that H_H should not be able to change the view of H_L at all, so H_L should be unable to output the bit b with a probability better than $\frac{1}{2}$. For statistical non-interference, we allow H_L a small advantage in guessing the bit b , modeled by a class *SMALL* of small functions in the security parameter k . For computational non-interference, we only consider polynomial-time configurations, and the advantage should be negligible; this is a special class of small functions. The approach of guessing a bit is essential to extend the notion of non-interference to the computational case. It is a fundamental concept in cryptography, so our definition serves as a link between prior work in non-interference and security definitions of real cryptographic primitives.

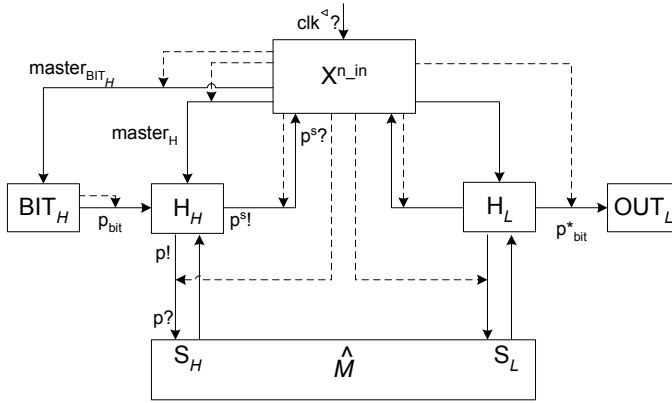


Fig. 4 Main Parts of a Non-Interference Configuration. The ports of the master scheduler X^{n-in} and the two emphasized users H_H and H_L are sketched.

Formally, to close the configurations, we add machines BIT_H and OUT_L that produce and consume the bit, respectively. They do this at ports $p_{bit}!$ and $p_{bit}^*?$ connected to special ports of H_H and H_L .

The configurations with these bit machines will be called *non-interference configurations*. Another characteristic of non-interference configurations is that the different user machines and the adversary have no direct connections, because otherwise they could trivially transmit the bit. Moreover, these configurations contain a master scheduler X^{n-in} . It mainly schedules the users. Where correct machines schedule each other is fixed in the structures, and which inputs to correct machines are scheduled by the adversary must be taken as given after the trust model. However, some way is needed to transfer control between different user (and adversary) groups even if they cannot interfere with each other. Here the master scheduler is needed. Our specific master scheduler essentially performs round-robin scheduling of users, when users get the control in the configuration at all, but lets the users choose certain subsequent system inputs.¹

The ports where the users get these master scheduling signals are called $master_i?$. For the master scheduler to get control once a user was scheduled (to prevent loops among a small user group and some correct machines), the users have no clock-out ports. Each port $p^{a!}$ that a user H_i would have according to its set S_i of specified ports is taken over by the master scheduler. Instead, H_i gets an output $p^s!$ to the master scheduler. When it is H_i 's turn, the master scheduler reads there whether H_i wants to schedule \tilde{p} next. This is shown in Figure 4 for a port $p^{a!} \in S_H^c$.

¹ Instead of considering round-robin scheduling only, it is probably also sufficient for most applications to consider schedulers with the following two properties: They are polynomially fair [4], i.e., they ensure that each user is always scheduled after a fixed polynomial number of steps, and their scheduling cannot be affected by previous inputs from the users and the adversary so that it is not possible to transmit information by interfering with the scheduler.

Definition 10 (Non-Interference Configuration) Let a system Sys be given and a structure $(\hat{M}, S) \in Sys$. Let Δ be a permitted node set for a flow policy for S with index set \mathcal{I} , and let $H, L \in \mathcal{I}^{adv}$ with $H \neq L$. Let $conf = (\hat{M}, S, U^{n.in}, A)$ be a multi-party configuration with $U^{n.in} = U \cup \{\text{BIT}_H, \text{OUT}_L, X^{n.in}\}$ and $U = \{H_i \mid i \in \mathcal{I}\}$, and let $H_{adv} := A$. We call it a *non-interference configuration* for Δ , H , and L if the following holds.

- a) **Ports of Special Machines:** The ports of BIT_H are $\{\text{master}_{\text{BIT}_H}?, \text{p}_{\text{bit}}!, \text{p}_{\text{bit}}^{\triangleleft}!\}$. The machine OUT_L has only one port $\text{p}_{\text{bit}}^*?$. The master scheduler $X^{n.in}$ has the following ports:
- $\{\text{clk}^{\triangleleft}?\}$: The master clock-in port.
 - $\{\text{master}_i!, \text{master}_i^{\triangleleft}! \mid i \in \mathcal{I}^{adv} \cup \{\text{BIT}_H\}\}$: The ports for fair scheduling of the users, the adversary and BIT_H .
 - $\{\text{p}^{\triangleleft}! \mid \text{p}^{\triangleleft}! \in S^c\} \cup \{\text{p}_{\text{bit}}^*{\triangleleft}!\}$: The clock-out ports that the honest users would have had.
 - $\{\text{p}^{s?}, \text{p}^{s{\triangleleft}!} \mid \text{p}^{\triangleleft}! \in S^c\} \cup \{\text{p}_{\text{bit}}^{s*}?, \text{p}_{\text{bit}}^{s*{\triangleleft}!}\}$: The ports for receiving scheduling proposals from the users for the previous set of ports.²
- b) **Ports of Normal Users:** The ports of H_i with $i \in \mathcal{I} \setminus \{H, L\}$ are $\{\text{master}_i?\} \cup (S_i^c \setminus \{\text{p}^{\triangleleft}! \mid \text{p}^{\triangleleft}! \in S_i^c\}) \cup \{\text{p}^{s!} \mid \text{p}^{\triangleleft}! \in S_i^c\}$, as explained above. Additionally, H_H has a port $\text{p}_{\text{bit}}?$, and H_L has ports $\text{p}_{\text{bit}}^*!$ and $\text{p}_{\text{bit}}^{s*}!$.³
- c) **Behavior of BIT_H and OUT_L :** If BIT_H receives an input at $\text{master}_{\text{BIT}_H}?$, it chooses a bit $b \in \{0, 1\}$ at random, outputs b at $\text{p}_{\text{bit}}!$, and schedules it by outputting 1 at $\text{p}_{\text{bit}}^{\triangleleft}!$. The machine OUT_L does nothing on inputs at $\text{p}_{\text{bit}}^*?$. (It just “catches” the inputs to close the collection.)
- d) **Behavior of $X^{n.in}$:** The machine $X^{n.in}$ maintains two flags *start* and *sched_demand* over $\{0, 1\}$, initialized with 0, and a counter *cnt* over the index set \mathcal{I}^{adv} , initialized with *adv*. Without loss of generality we assume $\mathcal{I} := \{1, \dots, n\}$ and *adv* := 0, so the counter is defined over $\{0, \dots, n\}$. Additionally, to allow for polynomial-time versions of $X^{n.in}$, it has a counter *steps* and a bound P . (Formally this gives a family of master schedulers $X_P^{n.in}$, one for each polynomial P and for $P = \infty$.) If $X^{n.in}$ is scheduled, it behaves as follows:
- Case 1: Start of the run.* If *start* = 0: Set *start* := 1 and output 1 at $\text{master}_{\text{BIT}_H}!$ and at $\text{master}_{\text{BIT}_H}^{\triangleleft}!$.
- Case 2: Schedule users.* If *start* = 1 and *sched_demand* = 0, increment *steps* if *cnt* = n and stop forever if *steps* $\geq P(k)$. Set *cnt* := $\text{cnt} + 1 \bmod (n + 1)$ and output 1 at $\text{master}_{\text{cnt}}!$ and $\text{master}_{\text{cnt}}^{\triangleleft}!$. If *cnt* $\neq 0$, i.e., the clocked machine is an honest user, additionally set *sched_demand* := 1 to handle the scheduling demands of this user next.
- Case 3: Handle scheduling demands.* If *start* = 1 and *sched_demand* = 1, output 1 at every port $\text{p}^{s{\triangleleft}!}$ with $\text{p}^{\triangleleft}! \in S_{\text{cnt}}^c$, and for *cnt* = L also at $\text{p}_{L.\text{bit}}^{s*{\triangleleft}!}$. Then test whether there was a non-empty input i at exactly one port

² We assume without loss of generality that there is a systematic naming scheme for such new ports that does not clash with prior names.

³ The adversary always closes the collection; hence if $H = \text{adv}$ or $L = \text{adv}$ this also holds for $H_{adv} = A$. The adversary always has the ports $\{\text{master}_i?\} \cup \bar{S}^c$.

p^s ?⁴ If yes, output i at $p^!$, i.e., follow the user's proposal. In both cases set $sched_demand := 0$.

We denote the set of non-interference configurations of a structure (\hat{M}, S) for given Δ, H , and L by $\text{Conf}_{\Delta, H, L}^{n_in}(\hat{M}, S)$ and the subset of polynomial-time ones by $\text{Conf}_{\Delta, H, L, \text{poly}}^{n_in}(\hat{M}, S)$. All non-interference configurations of a system Sys are written $\text{Conf}^{n_in}(Sys)$ and $\text{Conf}_{\text{poly}}^{n_in}(Sys)$, respectively.

Now we can define the probability that the low user correctly guesses the bit that the high user tries to transmit.

Definition 11 (Guessing Probability) For a non-interference configuration $conf \in \text{Conf}_{\Delta, H, L}^{n_in}(\hat{M}, S)$ of a structure (\hat{M}, S) , the *guessing probability* $P_{\text{guess}, conf}$ is defined as

$$P_{\text{guess}, conf} := P(b = b^* \mid r \leftarrow run_{conf, k}; b := r \upharpoonright_{p^!}; b^* := r \upharpoonright_{p^*}).$$

This is a function of the security parameter k .

Several times we need notions that functions are small, e.g., the advantage of the guessing probability over one half. Hence we define corresponding function classes.

Definition 12 (Small functions)

- a) The class *NEGL* of *negligible functions* contains all functions $s : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ that decrease faster than the inverse of every polynomial, i.e., for all positive polynomials $Q \exists n_0 \forall n > n_0 : s(n) < \frac{1}{Q(n)}$.
- b) A set *SMALL* of functions $\mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ is a *class of small functions* if it is closed under addition, and with a function g also contains every function $g' \leq g$. Typical classes of small functions are *EXPSMALL*, which contains all functions bounded by $Q(k) \cdot 2^{-k}$ for a polynomial Q , and the larger class *NEGL*.

Now we are ready to give the non-interference definition, i.e., the definition of the semantics of a flow policy for a reactive system.

Definition 13 (Non-Interference) Let a system Sys , a structure $(\hat{M}, S) \in Sys$, and a flow policy $\mathcal{F} = (\Delta, \rightsquigarrow)$ with index set \mathcal{I} for (\hat{M}, S) be given. For two elements $H, L \in \mathcal{T}^{\text{adv}}$ with $S_H \rightsquigarrow S_L$, we say that (\hat{M}, S) *fulfills* the non-interference requirement $NIReq_{\mathcal{F}, H, L}$

- a) **perfectly**, written $(\hat{M}, S) \models_{\text{perf}} NIReq_{\mathcal{F}, H, L}$, iff for every non-interference configuration $conf \in \text{Conf}_{\Delta, H, L}^{n_in}(\hat{M}, S)$ we have

$$P_{\text{guess}, conf} \leq \frac{1}{2}.$$

⁴ More formally, X^{n_in} first sends 1 at the first of these ports. The buffer either delivers a message to X^{n_in} or does nothing. In both cases X^{n_in} is scheduled again, so it can send 1 at the second clock-out port, etc. It stores all received messages in an internal array.

- b) **statistically** for a class $SMALL$ of small functions, written $(\hat{M}, S) \models_{SMALL} NReq_{\mathcal{F},H,L}$, iff for every non-interference configuration $conf \in \text{Conf}_{\Delta,H,L}^{n,in}(\hat{M}, S)$ there exists a function $s \in SMALL$ such that

$$P_{\text{guess},conf} \leq \frac{1}{2} + s(k).$$

- c) **computationally**, written $(\hat{M}, S) \models_{\text{poly}} NReq_{\mathcal{F},H,L}$, iff for every polynomial-time non-interference configuration $conf \in \text{Conf}_{\Delta,H,L,\text{poly}}^{n,in}(\hat{M}, S)$ there exists a function $s \in NEGL$ such that

$$P_{\text{guess},conf} \leq \frac{1}{2} + s(k).$$

We write “ \models ” if we want to treat all cases together. If a structure fulfills all non-interference requirements $NReq_{\mathcal{F},H,L}$ with $S_H \rightsquigarrow S_L$ for a flow policy \mathcal{F} , we say it fulfills the requirement $NReq_{\mathcal{F}}$, written $(\hat{M}, S) \models NReq_{\mathcal{F}}$. A system Sys fulfills a flow policy \mathcal{F} for this system if every structure $(\hat{M}, S) \in Sys$ fulfills the requirement $NReq_{\mathcal{F}_{(\hat{M},S)}}$. We then write $Sys \models \mathcal{F}$.

4 Preservation of Non-Interference under Simulatability

The cryptographic variety of the notion that one system securely implements another one is based on the concept of *reactive simulatability* [55]. Reactive simulatability essentially means that whatever might happen to an honest user of a concrete system Sys_{real} can also happen to this user with a given ideal system Sys_{id} . More precisely, for every configuration $conf_1 \in \text{Conf}(Sys_{\text{real}})$, there exists a configuration $conf_2 \in \text{Conf}(Sys_{\text{id}})$ that yields an indistinguishable view for the same user. We abbreviate this by $Sys_{\text{real}} \geq_{\text{sec}} Sys_{\text{id}}$ and say that Sys_{real} is at least as secure as the system Sys_{id} . A typical situation is shown in Figure 5.

The notion of reactive simulatability serves as the culmination of a long line of research done on (non-reactive) simulatability: Simulatability was first sketched for secure multi-party function evaluation, i.e., for the computation of one output tuple from one tuple of secret inputs from each participant in [70] and defined (with different degrees of generality and rigorosity) in [26,27,7,51,11]. The idea of simulatability was subsequently also used for specific reactive problems, e.g., [21,9,37,38,13], without a detailed or general definition. In a similar way it was used for the construction of generic solutions for large classes of reactive problems [26,24,31] (usually yielding inefficient solutions and assuming that all parties take part in all subprotocols). The first fully reactive definition of simulatability was presented in [54] for a synchronous version of a general reactive model, and has been extended to an asynchronous setting in [55] and later but independently in [12].

We do not want to compare a structure $(\hat{M}_1, S_1) \in Sys_{\text{real}}$ with arbitrary structures of Sys_{id} , but only with certain “suitable” ones. This is specified by a mapping f from Sys_{real} to the powerset of Sys_{id} . The mapping f is called *valid* if $f(\hat{M}_1, S_1)$

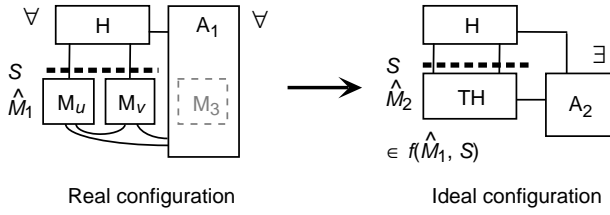


Fig. 5 Example of simulatability. For every user H of the real structure and every adversary A_1 , there must exist an adversary A_2 on a corresponding ideal structure such that the view of H is indistinguishable.

is non-empty and only contains structures (\hat{M}_2, S_2) with $S_2 = S_1$, i.e., with the same user interface, for all $(\hat{M}_1, S_1) \in Sys_{real}$.

The simulatability definition is based on the indistinguishability of views. Indistinguishability is a notion defined for arbitrary random variables.

Definition 14 (Indistinguishability) Two families $(var_k)_{k \in \mathbb{N}}$ and $(var'_k)_{k \in \mathbb{N}}$ of random variables (or probability distributions) on common domains D_k are

- perfectly indistinguishable* (“=”) if for each k , the two distributions var_k and var'_k are identical.
- statistically indistinguishable* (“ \approx_{SMALL} ”) for a class $SMALL$ of small functions if the distributions are discrete and their statistical distances

$$\Delta_{stat}(var_k, var'_k) := \frac{1}{2} \sum_{d \in D_k} |P(var_k = d) - P(var'_k = d)| \in SMALL$$

(as a function of k).

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

$$|P(Dis(1^k, var_k) = 1) - P(Dis(1^k, var'_k) = 1)| \in NEGL.$$

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

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

We are now ready to present the reactive simulatability definition, i.e., a notion of cryptographically secure implementation.

Definition 15 (Reactive Simulatability) Let systems Sys_1 and Sys_2 with a valid mapping f be given.

- We say $Sys_1 \geq_{sec}^{f, perf} Sys_2$ (*perfectly at least as secure as*) if for every configuration $conf_1 = (\hat{M}_1, S, H, A_1) \in Conf(Sys_1)$, there exists a configuration $conf_2 = (\hat{M}_2, S, H, A_2) \in Conf(Sys_2)$ with $(\hat{M}_2, S) \in f(\hat{M}_1, S)$ (and the same H) such that

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

- b) We say $Sys_1 \geq_{\text{sec}}^{f, \text{SMALL}} Sys_2$ (*statistically at least as secure as*) for a class *SMALL* of small functions if the same as in a) holds with $\text{view}_{\text{conf}_1, l}(\text{H}) \approx_{\text{SMALL}} \text{view}_{\text{conf}_2, l}(\text{H})$ for all polynomials l , i.e., statistical indistinguishability of all families of l -step prefixes of the views.
- c) We say $Sys_1 \geq_{\text{sec}}^{f, \text{poly}} Sys_2$ (*computationally at least as secure as*) if the same as in a) holds with configurations from $\text{Conf}_{\text{poly}}(Sys_1)$ and $\text{Conf}_{\text{poly}}(Sys_2)$ and computational indistinguishability of the families of views.

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

Below we want to prove that non-interference properties are preserved under reactive simulatability. As flow policies are defined per system, we first have to define how a flow policy from one system, typically a specification, is applied to another system, typically an implementation. We do this based on a valid mapping between the system, i.e., based on the information which real structures implement which ideal structures. As valid mappings retain the specified ports (the user interfaces) and our flow policies for systems depend only on these specified ports, this is a canonical transformation.

Definition 16 (*Corresponding Flow Policies*) Let systems Sys_1 and Sys_2 with a valid mapping f be given, and a flow policy $\mathcal{F}^{(2)}$ for Sys_2 . The *corresponding flow policy* $\mathcal{F}^{(1)}$ for Sys_1 is defined as follows: For every $(\hat{M}_1, S) \in Sys_1$, let $\mathcal{F}_{(\hat{M}_1, S)}^{(1)} := \mathcal{F}_{(\hat{M}_2, S)}^{(2)}$ for an arbitrary structure $(\hat{M}_2, S) \in f(\hat{M}_1, S)$.

This is well-defined because $f(\hat{M}_1, S)$ is non-empty and only contains structures with the same set S , and $\mathcal{F}_{(\hat{M}_2, S)}^{(2)}$ is equal for all these structures. Further, every $\mathcal{F}_{(\hat{M}_1, S)}^{(1)}$ is a valid flow policy for S by definition. We often call such corresponding flow policies \mathcal{F} on both systems.

Our following preservation theorem states that non-interference properties are preserved under the reactive simulatability relation \geq_{sec} , as the name “at least as secure as” suggests.

Theorem 1 (*Preservation of Non-Interference*) Let a system Sys_1 be as secure as Sys_2 , i.e., $Sys_1 \geq^f Sys_2$ for a valid mapping f . Let Sys_2 fulfill a flow policy \mathcal{F} , i.e., $Sys_2 \models \mathcal{F}$, and let \mathcal{F} also denote the corresponding flow policy for Sys_1 according to Definition 16. Then also $Sys_1 \models \mathcal{F}$. This holds for the perfect, statistical, and computational case.

Proof We have to show that every structure $(\hat{M}_1, S) \in Sys_1$ fulfills its flow policy $\mathcal{F}_{(\hat{M}_1, S)}$. We fix such a structure $(\hat{M}_1, S) \in Sys_1$ and its flow policy $\mathcal{F}_S = (\Delta, \not\sim)$ with index set \mathcal{I} . For all $H, L \in \mathcal{I}^{\text{adv}}$ with $S_H \not\sim S_L$ we have to show that (\hat{M}_1, S) fulfills the non-interference requirement $\text{NIREq}_{\mathcal{F}_S, H, L}$.

Let a non-interference configuration $\text{conf}_1 = (\hat{M}_1, S, U^{n\text{-in}}, A_1) \in \text{Conf}_{\Delta, H, L}^{n\text{-in}}(\hat{M}_1, S)$ be given. Because of $Sys_1 \geq^f Sys_2$ there exists a configuration $\text{conf}_2 = (\hat{M}_2, S, U^{n\text{-in}}, A_2) \in \text{Conf}(Sys_2)$ with $(\hat{M}_2, S) \in f(\hat{M}_1, S)$

and $view_{conf_1}(U^{n.in}) \approx view_{conf_{H,L,2}}(U^{n.in})$. As the user set $U^{n.in}$ is equal in both configurations, $conf_2$ is also a non-interference configuration in $\text{Conf}_{\Delta,H,L}^{n.in}(\hat{M}_2, S)$. By precondition, (\hat{M}_2, S) fulfills $NIRReq_{\mathcal{F}_S,H,L}$.

Now we distinguish the perfect, statistical, and the computational case (for the reactive simulatability type and the non-interference type together). In the computational case, both configurations are polynomial-time.

In the perfect case, we have $view_{conf_1}(U^{n.in}) = view_{conf_2}(U^{n.in})$. Both $b := r \upharpoonright_{\text{pbit}}$ and $b^* := r \upharpoonright_{\text{pbit}^*}$ are part of the view of $U^{n.in}$, so we obtain $P_{\text{guess}, conf_1} = P_{\text{guess}, conf_2} \leq \frac{1}{2}$. With our arbitrary choice of $conf_1$ this implies that (\hat{M}_1, S) also fulfills $NIRReq_{\mathcal{F}_S,H,L}$.

In the statistical and the computational case we have a given class $SMALL$ of small functions, where $SMALL = NEGL$ in the computational case. We assume for contradiction that $P_{\text{guess}, conf_1} = \frac{1}{2} + ns(k)$ for a function $ns \notin SMALL$, whereas we know that $s(k) := P_{\text{guess}, conf_2} - \frac{1}{2} \in SMALL$.

We then define a distinguisher D as follows. Given the view of $U^{n.in}$ in one of the configurations, D knows both the bit b and the guess b^* . It outputs 1 if $b = b^*$ and 0 otherwise. Its advantage in distinguishing the configurations by this is

$$\begin{aligned} \delta_D &:= |P(D(1^k, view_{conf_1,k}(U^{n.in})) = 1) \\ &\quad - P(D(1^k, view_{conf_2,k}(U^{n.in})) = 1)| \\ &= \left| \frac{1}{2} + ns(k) - \left(\frac{1}{2} + s(k) \right) \right| = ns(k) - s(k) \\ &\notin SMALL. \end{aligned}$$

The last line holds because $SMALL$ is closed under addition. For the polynomial case, this immediately contradicts the indistinguishability of the views, $view_{conf_1}(U^{n.in}) \approx_{\text{poly}} view_{conf_2}(U^{n.in})$.

For the statistical case, the results of the distinguisher D are a function on the random variables of the views. As the results are Boolean, their statistical distance is easily computed as

$$\begin{aligned} \delta_{\text{stat}, D} &:= \Delta_{\text{stat}}(D(1^k, view_{conf_1,k}(U^{n.in})), D(1^k, view_{conf_2,k}(U^{n.in}))) \\ &= \frac{1}{2}(2\delta_D) = \delta_D. \end{aligned}$$

By a well-known lemma, the statistical distance between a function of two random variables is at most the statistical distance of the original random variables; a proof can be found in [25]. This implies

$$\begin{aligned} \delta_{\text{view}} &:= \Delta_{\text{stat}}(view_{conf_1,k}(U^{n.in}), view_{conf_2,k}(U^{n.in})) \\ &\geq \delta_{\text{stat}, D} = \delta_D \\ &\notin SMALL. \end{aligned}$$

The last line holds because $SMALL$ is closed under making functions smaller. This contradicts the statistical indistinguishability of the views, $view_{conf_1}(U^{n.in}) \approx_{SMALL} view_{conf_2}(U^{n.in})$.

With our arbitrary choice of $conf_1$ this implies that (\hat{M}_1, S) also fulfills $NIRReq_{\mathcal{F}_S,H,L}$ in the computational and statistical case. This finishes the proof.

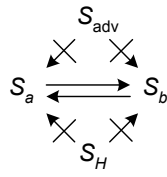


Fig. 6 Sketch of the flow policy for the firewall. The port sets of the protected users a and b , an outsider H , and the adversary are shown. Missing edges in the graph are of the form “ \sim ”.

5 A Cryptographic Firewall

We now present a cryptographic firewall as an example of a system that must fulfill a non-interference property and uses cryptographic primitives. The goal is to allow communication between certain participants, while ensuring that these participants cannot be affected by their environment. This goal corresponds to a flow policy. Most firewalls distinguish users mainly by IP addresses. For high security, however, different external users should rather be distinguished by cryptographic authentication.

5.1 Introduction to the Cryptographic Firewall

We build the firewall systems in two layers. The lower layer provides secure message transmission, the higher layer the filtering function. In particular, the secure message-transmission layer provides cryptographic authentication, so that the filtering layer can filter by name.

For the lower layer, we would like to reuse the secure message-transmission system with ordered channels from [2]. Ordered channels are one step beyond a related system with unordered channels from [55] towards non-interference. However, we have to go slightly further, because an adversary could also interfere with the protected users by scheduling all their messages either immediately or never, which gives different views. Recall that the explicit master scheduler only ensures that each user can send messages from time to time, while the adversary still schedules the network. Hence we need a new type of reliable, non-authenticated channels.

For simplicity, we only define and prove one particular information flow policy and corresponding filtering rule set. The policy is sketched in Figure 6: Two users a and b should be able to communicate, but should not be disturbed by the other users and the adversary. As this is an integrity view of information flow, it makes a and b the low users, and all others the high users. As usual, the users are represented in the flow policies by the sets S_i of specified ports they connect to.

We start the detailed description with a review of how systems are commonly derived from an intended structure with a trust model, and of the composition theorem needed to prove a two-layer system in a modular way (Section 5.2). In Section 5.3 we present the specification of the lower layer, i.e., an ideal system for secure and reliable message transmission. In Section 5.4, we define the higher

layer, the filtering system. As it is built on the ideal lower layer and uses no additional cryptography, we need not distinguish a real and ideal version here.

In Section 5.5, we sketch the real lower-layer system and define the new reliable channel type. We also sketch why the security proof from [2] still applies to the modified secure message-transmission system. Finally, in Section 5.6 we prove that the two-layer firewall fulfills its non-interference requirement. We mainly do this for the combination – a well-defined notion introduced in [55] – of the ideal lower layer and the filtering system, and use our theorems to show that the result also applies to the fully real firewall system.

5.2 Preliminaries

The lower layer of the firewall system is of a class called *standard cryptographic systems* in [55]. In the intended structure of such a system, users are numbered $1, \dots, n$. Every user u has one machine M_u , which is correct if and only if the user is honest. The machine M_u has ports $in_u?$ and $out_u!$ for connecting to its user. A real system Sys_{real} is derived from such an intended structure by a *trust model* consisting of an access structure \mathcal{ACC} and a channel model χ . An *access structure* \mathcal{ACC} is a set of subsets \mathcal{H} of $\{1, \dots, n\}$. Intuitively, it denotes the possible sets of honest users, and thus of correct machines. The *channel model* classifies every connection as secure (private and authentic), authenticated or insecure. For achieving non-interference with the firewall, we need a fourth channel type called *reliable non-authenticated*.

The resulting system contains one structure for every set $\mathcal{H} \in \mathcal{ACC}$, consisting of the remaining correct machines from the intended structure with modified channels according to the channel model. For the three predefined channel types, this modification is a well-defined port renaming scheme [55]. The corresponding scheme for the new channel type is given in Definition 17. We denote such a set of remaining modified machines by $\hat{M}_{\mathcal{H}} := \{M_{u,\mathcal{H}} \mid u \in \mathcal{H}\}$, and the remaining set of specified ports by $S_{\mathcal{H}}$. Thus the resulting system is of the form $Sys_{\text{real}} = \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}$. A corresponding *standard ideal system* is of the form $Sys_{\text{id}} = \{(\{TH_{\mathcal{H}}\}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}$ with the same sets $S_{\mathcal{H}}$. The machines $TH_{\mathcal{H}}$ are called *trusted hosts*.

As we construct the firewall system as a composition of two layers, we briefly review the composition theorem of [55]. It states that the relation “at least as secure as” is consistent with such compositions. Assume we have already proven that a system Sys_0 is at least as secure as a specification Sys'_0 , and we build a system Sys_1 on top of the specification Sys'_0 . Then we want to replace Sys'_0 by the real system Sys_0 . We call the former, partially abstract composition Sys^* and the latter, real composition $Sys^\#$. The composition theorem states that this replacement is secure, i.e., $Sys^\#$ is at least as secure as Sys^* ; see [55] for the precise theorem and its proof. This is illustrated in Figure 7.

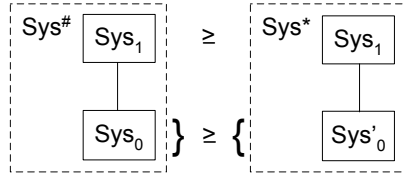


Fig. 7 Composition of Systems

5.3 Lower Layer: Ideal System for Secure and Reliable Message Transmission

In this section, we present the specification of the lower layer of the firewall, secure and reliable message transmission. We first sketch the original ideal system from [2], more precisely its perfectly ordered variant where no gaps in the message sequence are accepted. Then we describe the modifications needed to include reliable communication between the protected users. The resulting ideal system is defined in detail in Appendix A.

5.3.1 Sketch of the original system. The secure message-transmission system with ordered channels from [2] is a standard cryptographic system as described in Section 5.2, and its access structure ACC contains all subsets of the user indices $\{1, \dots, n\}$. Thus the ideal system is of the form $Sys'_{0,orig} = \{(\{TH_{\mathcal{H}}\}, S_{\mathcal{H}}) | \mathcal{H} \subseteq \{1, \dots, n\}\}$.

The ideal machine $TH_{\mathcal{H}}$ models initialization and sending and receiving of messages. Initialization corresponds to key generation and authenticated key exchange in real system. Besides the specified ports, $TH_{\mathcal{H}}$ has ports $to_adv_u!$ and $from_adv_u?$ where it informs the adversary, and accepts adversary inputs, regarding the service to user u . This is necessary because in efficient implementations, the adversary gets certain information and has certain influence, in particular by network scheduling.

A user u initializes communication with other users by inputting a command (snd_init) at the port $in_u?$ of $TH_{\mathcal{H}}$. To reflect the asynchronous timing model, $TH_{\mathcal{H}}$ waits for a command (rec_init, u) from the adversary at a port $from_adv_v?$ before regarding communication between u and v as initialized.

A user u sends a message to user v by inputting ($send, m, v$). If v is dishonest, $TH_{\mathcal{H}}$ immediately outputs ($send, m, v$) to the adversary. If v is honest, $TH_{\mathcal{H}}$ stores the message in an array $deliver_{u,v}^{spec}$ together with a counter that indicates the number of messages sent from u to v , and outputs ($send_blindly, i, l, v$) to the adversary, where l and i denote the length of m and its position in the array, respectively. This models that the adversary learns that a message of a certain length is being sent.

For delivering this message to v , the machine $TH_{\mathcal{H}}$ waits for a command ($receive_blindly, u, i$) from the adversary at a port $from_adv_v?$. Then $TH_{\mathcal{H}}$ reads $(m, j) := deliver_{u,v}^{spec}[i]$ and checks whether $msg_out_{u,v}^{spec} = j$ holds for the number $msg_out_{u,v}^{spec}$ of the next expected message.

If yes, it outputs $(\text{receive}, u, m)$ to v and sets the expected number to $j + 1$. This last condition ensures that messages can only be delivered in the order as they were input to $\text{TH}_{\mathcal{H}}$.

The adversary can send a message m to a user u by inputting $(\text{receive}, v, m)$ at the port $\text{from_adv}_u?$ of $\text{TH}_{\mathcal{H}}$ for a corrupted user v ; this message is output to u immediately.

The adversary can also stop the service for user u by inputting (stop) at port $\text{from_adv}_u?$. This models that an adversary may achieve that the runtime bound of u 's machine is exceeded in the real system.

Note that this ideal system is completely deterministic and without cryptography.

5.3.2 Adding reliable message transmission. We want to build a firewall by adding a filtering policy on top of this ideal message-transmission system, and the firewall should fulfill the flow policy shown in Figure 6. However, as long as the adversary can schedule the messages between the protected users a and b (recall that $\text{TH}_{\mathcal{H}}$ waits for a command $(\text{receive_blindly}, b, i)$ for that, where without loss of generality we always consider a as the sender), it can achieve two distinguishable behaviors by either immediately scheduling each such message, or never scheduling them. This problem cannot be solved by the filtering system on top.⁵ This corresponds to a small covert channel. To close it, we need to specify reliable communication for the two protected users a and b in this section, and later define suitable implementations of reliable channels in the real system.

The modified trusted host for the modified system Sys'_0 , still called $\text{TH}_{\mathcal{H}}$, behaves identically for inputs from and outputs to users $u \notin \{a, b\}$. For communication from a to b , it is modified as follows (and vice versa):

- If a inputs an initialization command (snd_init) at $\text{in}_a?$, $\text{TH}_{\mathcal{H}}$ immediately initializes communication with b . In particular, it outputs $(\text{rec_init}, a)$ to H_b and schedules this output.
- If a sends a message to b , i.e., inputs (send, m, b) at $\text{in}_a?$ then $\text{TH}_{\mathcal{H}}$ immediately outputs $(\text{receive}, m, a)$ at port $\text{out}_b!$ and schedules it.

In both cases, $\text{TH}_{\mathcal{H}}$ informs the adversary of the event as above. The complete resulting definition of the ideal system Sys'_0 is given in Appendix A.

5.4 The Filtering System

We now present the filtering system, the upper layer of the firewall. It is called Sys_1 as in Figure 7. We only need filters for the protected users a and b , and filtering only works if the machines of these users are correct, i.e., $a, b \in \mathcal{H}$. Hence the system has only one structure, which contains only two machines $\text{M}_a^{\text{filter}}$ and $\text{M}_b^{\text{filter}}$. The composition of these filtering machines with the secure message-transmission system is shown in Figure 8 with individual user machines H_u , but

⁵ The filter at user b can only delete messages from other senders, but has to let messages from user a pass, while it cannot let them pass if they do not arrive.

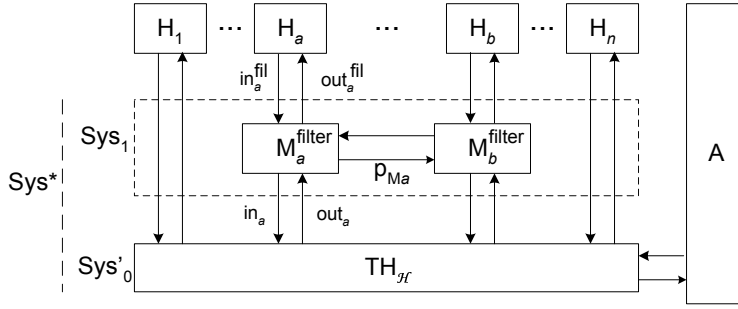


Fig. 8 Firewall system consisting of filtering machines and the specification of secure reliable message transmission. Clock-out ports are omitted; every machine has the corresponding clock-out port for each of its output ports.

without the special machines of the non-interference configurations and that $X^{n\text{-in}}$ takes over the clock-in ports.

The inputs and outputs to the filters from above and below are as in the message transmission system, because the goal is to filter such messages. Recall that even the ideal secure message-transmission system with reliable channels allows the adversary to stop the service to a user at any time, modeling that an adversary overpowers a correct machine in the real system. If an adversary either stops the service to user a at the start of the run, or never stops it, we obtain different views for the user a . To avoid this problem, we provide additional reliable channels p_{M_a} and p_{M_b} to close a covert channel based on stopping a service. The reliable channels p_{M_a} and p_{M_b} are only used if the underlying service for secure message transmission has been stopped, and they should hence only be regarded as a remedy against denial-of-service attacks. In the real world, this means that if the commonly used communication method falls prey to a denial-of-service attack, the users will look for an alternative way to communicate.

Scheme 1 (Filtering System) Let $n \in \mathbb{N}$, $a, b \in \mathcal{M} := \{1, \dots, n\}$ with $a \neq b$, and polynomials $L, s, s' \in \mathbb{N}[x]$ be given. Here n denotes the number of intended users and a and b the users to be protected from the others. $L(k)$ bounds the message length and $s(k), s'(k)$ bound the number of messages each user can send and receive respectively for a security parameter k .⁶ Then

$$Sys_1 := \{(\hat{M}^{\text{filter}}, S^{\text{filter}})\}$$

with $\hat{M}^{\text{filter}} = \{M_a^{\text{filter}}, M_b^{\text{filter}}\}$ and $S^{\text{filter}^c} := \{\text{out}_u^{\text{fil}^?}, \text{in}_u^{\text{fil}^!}, \text{in}_u^{\text{fil}^{\triangleleft!}}, \text{in}_u^{\text{?}}, \text{out}_u^{\text{!}}, \text{out}_u^{\triangleleft!} \mid u \in \{a, b\}\}$. We only define the machine M_a^{filter} ; we obtain M_b^{filter} by exchanging the variables a and b .

Ports of a filter. The ports of machine M_a^{filter} are $\{\text{in}_a^{\text{fil}^?}, \text{out}_a^{\text{fil}^!}, \text{out}_a^{\text{fil}^{\triangleleft!}}\} \cup \{\text{out}_a^{\text{?}}, \text{in}_a^{\text{!}}, \text{in}_a^{\triangleleft!}\} \cup \{p_{M_b}^{\text{?}}, p_{M_a}^{\text{!}}, p_{M_a}^{\triangleleft!}\}$.

⁶ These bounds ensure that Sys_1 is polynomial-time. This is essential for applying the composition theorem to replace Sys'_0 with Sys_0 in the overall system.

States of a filter. M_a^{filter} maintains a counter $s_a \in \{0, \dots, s(k)\}$ and an array $(s'_{a,u})_{u \in \mathcal{M}}$ over $\{0, \dots, s'(k)\}$ counting the messages that a sends and receives from u , respectively. Further, it contains a variable $stopped_a \in \{0, 1\}$ denoting whether the lower-layer service to a has stopped. All variables are initialized with 0 everywhere.

Filter transitions. The state-transition function of M_a^{filter} is defined by the following rules:

- **Send initialization:** On input (snd_init) at $\text{in}_a^{\text{fil}}?$: If $s_a < s(k)$, set $s_a := s_a + 1$, else stop.⁷
If $stopped_a = 0$, output (snd_init) at $\text{in}_a!$ and 1 at $\text{in}_a^{\triangleleft}!$. Else output (rec_init, a) at $\text{p}_{M_a}!$ and 1 at $\text{p}_{M_a}^{\triangleleft}!$, i.e., use the special reliable channel.
- **Receive initialization:** On input (rec_init, u) at $\text{out}_a?$ with $u \in \mathcal{M}$: If $s'_{a,u} < s'(k)$, set $s'_{a,u} := s'_{a,u} + 1$, else stop. If $stopped_a = 0$ and $u = b$ (this is the filtering), output (rec_init, b) at $\text{out}_a^{\text{fil}}!$ and 1 at $\text{out}_a^{\text{fil}\triangleleft}!$.
- **Receive initialization, extra channel:** On input (rec_init, b) at $\text{p}_{M_b}?$, output (rec_init, b) at $\text{out}_a^{\text{fil}}!$ and 1 at $\text{out}_a^{\text{fil}\triangleleft}!$.
- **Send:** On input (send, m, v) at $\text{in}_a^{\text{fil}}?$ with $m \in \Sigma^+$ and $\text{len}(m) \leq L(k)$, where $\text{len}(m)$ denotes the message length: If $s_a < s(k)$, set $s_a := s_a + 1$, else stop. If $stopped_a = 0$, output (send, m, v) at $\text{in}_a!$ and 1 at $\text{in}_a^{\triangleleft}!$. If $stopped_a = 0$ and $v = b$, output (receive, a, m) at $\text{p}_{M_a}!$ and 1 at $\text{p}_{M_a}^{\triangleleft}!$.
- **Receive:** On input (receive, u, m) at $\text{out}_a?$ with $u \in \mathcal{M}$: If $s'_{a,u} < s'(k)$, set $s'_{a,u} := s'_{a,u} + 1$, else stop. If $u = b$, output (receive, b, m) at $\text{out}_a^{\text{fil}}!$ and 1 at $\text{out}_a^{\text{fil}\triangleleft}!$.
- **Receive, extra channel:** On input (receive, b, m) at $\text{p}_{M_b}?$, output (receive, b, m) at $\text{out}_a^{\text{fil}}!$ and 1 at $\text{out}_a^{\text{fil}\triangleleft}!$.
- **Stop:** On input (stop) at $\text{out}_a?$: If $s_a < s(k)$, set $s_a := s_a + 1$, else stop. If $stopped_a = 0$, set $stopped_a := 1$ and output (stop) at $\text{p}_{M_a}!$ and 1 at $\text{p}_{M_a}^{\triangleleft}!$. (This ensures that machine M_b^{filter} also uses the special channel from now on.)
- **Stop, extra channel:** On input (stop) at $\text{p}_{M_b}?$: Set $stopped_a := 1$. \square

Remark 1 The filtering system Sys_1 can easily be modified to an arbitrary set of protected users instead of $L = \{a, b\}$. Moreover we can treat multiple disjoint sets of users where each user can communicate with other users of its own set without outside interference.

Lemma 1 The system Sys_1 is polynomial-time.

Proof The machine M_a^{filter} has counters s_a and $s'_{a,u}$ for each $u \in \{1, \dots, n\}$. At least one counter is increased whenever M_a^{filter} receives an input at port $\text{in}_a^{\text{fil}}?$ or $\text{out}_a?$, and once a counter reaches a polynomial bound $s(k)$ or $s'(k)$, the machine stops. Every output at $\text{p}_{M_a}!$ is a direct consequence of an input at $\text{in}_a^{\text{fil}}?$ or $\text{out}_a?$ by construction of M_a^{filter} . Hence the number of messages sent over $\text{p}_{M_a}!$ is at most

⁷ This means that this machine stops forever. It has nothing to do with the variable $stopped_a$.

$s(k) + n \cdot s'(k)$. This holds analogously for the machine M_b^{filter} . Thus the steps of the whole system Sys_1 (not counting buffers) are bounded by $4n \max\{s(k), s'(k)\}$. Since each transition can clearly be realized in polynomial time, we conclude that the collection $\{M_a^{\text{filter}}, M_b^{\text{filter}}\}$ is polynomial-time.

5.5 The Real Lower Layer: Secure and Reliable Message Transmission

In the real firewall system, the ideal system for secure reliable message transmission is replaced with its concrete implementation Sys_0 . We only need a brief description of the original concrete implementation $Sys_{0,\text{orig}}$ here to show how we modify it to include reliable channels, and to sketch a proof why this modified real system Sys_0 is as secure as the modified ideal system Sys'_0 from Section 5.3. Otherwise, the benefit of the modular approach enabled by the composition theorem is exactly that the higher-layer systems work with all correct implementations of lower-layer systems.

The original system $Sys_{0,\text{orig}}$, described in detail in [2], is a standard cryptographic system $Sys_0 = \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \subseteq \{1, \dots, n\}\}$ with the same access structure, specified ports, and in- and output types at the specified ports as in the ideal system Sys'_0 .

It uses asymmetric encryption and digital signatures, which must fulfill the accepted cryptographic definition, i.e., security against adaptive chosen-ciphertext attack for encryption and security against existential forgery under adaptive chosen-message attacks for digital signatures [10,28]. Efficient cryptographic primitives exist for both cases under reasonable assumptions, e.g., [15,28].

On input of the command (snd_init), a machine M_u creates signature and encryption keys and sends it to the other machines M_v over authenticated channels. On input (send, m, v), machine M_u signs and encrypts the message m with certain additional parameters and sends it to M_v over an insecure channel, representing a real network. If M_v obtains such a message with correct syntax and the next expected message number, it outputs (receive, u, m) to user v . The adversary schedules the communication between correct machines; otherwise its capabilities arise from what machines and channels it has replaced in the actual structures according to the trust model.

We now describe the modification to this system by reliable channels. Formally, reliable channels are a new type in the channel model of standard cryptographic system, see Section 5.2.

Definition 17 (Reliable, Non-authenticated Channels) Let an intended structure (\hat{M}, S) with $\hat{M} := \{M_u \mid u \in \mathcal{H}\}$ be given, and a channel defined by ports $p!, p^{\dagger!} \in \text{ports}(M_u)$ and $p? \in \text{ports}(M_v)$. If the channel model χ classifies this channel as *reliable non-authenticated*, then in the derivation of the actual structure $(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})$ for an index set \mathcal{H} , the channel is modified as follows: If only one of u, v lies in \mathcal{H} , it is treated like an authenticated channel. If $u, v \in \mathcal{H}$, then a specific, buffer-style machine is inserted as shown in Figure 9.

- $M_{u,\mathcal{H}}$ gets a new port $p^{\dagger!}$, where it duplicates outputs made at $p!$. (As $p^{\dagger!}$ is free, the adversary can connect to it.)

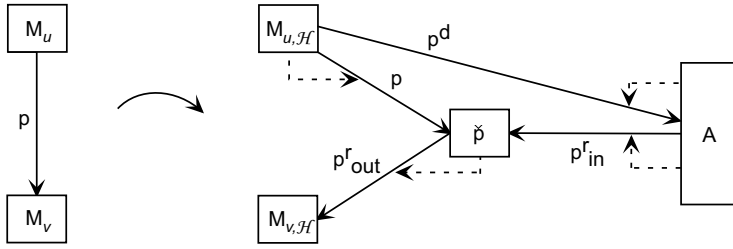


Fig. 9 Modeling Reliable, Non-Authenticated Channels

- The input port $p^?$ of $M_{v,\mathcal{H}}$ is renamed into $p_{out}^r?$.
- A machine \check{p} is defined as follows: Its ports are $\{p^?, p_{in}^r?, p_{out}^r!, p_{out}^r\langle!\rangle\}$. Given an input at $p^?$ or $p_{in}^r?$, it forwards this input to $p_{out}^r!$ and schedules it by outputting 1 at $p_{out}^r\langle!\rangle$.

We assume without loss of generality that there is a systematic naming scheme for such new ports (e.g., appending $^d, ^r_{out}, ^r_{in}$) that does not clash with prior names. Reliable, *authenticated* channels are defined similarly, simply by omitting the port $p_{in}^r?$ of \check{p} .

Now the modified system Sys_0 is derived from the original one $Sys_{0,orig}$ by classifying the initialization channels between the machines M_a and M_b of $Sys_{0,orig}$ as reliable and authenticated, whereas the network channels between these machines are classified as reliable, but non-authenticated.

In the following, we only briefly sketch that the relation “at least as secure as” still holds for the modified system Sys_0 and Sys'_0 , because we would have to redo the whole original proof of [55] with only small changes.

Theorem 2 Let Sys_0 and Sys'_0 be the modified real and ideal systems for reliable secure message transmission as introduced in the previous sections. Then $Sys_0 \geq_{poly}^f Sys'_0$ with the mapping f defined as in [55].

Proof (sketch) Both the ideal and the real system Sys_0 and Sys'_0 have been changed for initialization and sending of messages. We now briefly sketch that these changes are consistent for both systems.

In case of initialization, the only difference between the original and the modified ideal system is that initialization between a and b is done immediately by $TH_{\mathcal{H}}$, so the adversary is not able to initialize a connection between these two honest users by himself because initialization commands are filtered out by construction of $TH_{\mathcal{H}}$. In the real system Sys_0 , this implicit initialization of $TH_{\mathcal{H}}$ exactly corresponds to a reliable authenticated channel between a and b . In both systems, an initialization message is output to the user and immediately scheduled, along with the usual output to the adversary. This gives consistent changes from $Sys'_{0,orig}$ and $Sys_{0,orig}$ to Sys'_0 and Sys_0 , respectively, which yields indistinguishable behaviors of Sys'_0 and Sys_0 since $Sys'_{0,orig} \geq Sys_{0,orig}$ has already been shown in [55, 2].

In case of sending of messages, the difference to the original ideal scheme only affects sending messages between the two distinguished low users. In this case

$\text{TH}_{\mathcal{H}}$ immediately schedules the message to the corresponding user and outputs a message of the form $(\text{send_blindly}, i, l, v)$ to the adversary. In the real system, reliable non-authenticated channels do exactly the same: they schedule the message to the corresponding user and send a blinded copy to the adversary. Hence we have consistent changes and, using $Sys'_{0,\text{orig}} \geq Sys_{0,\text{orig}}$, we obtain indistinguishable behaviors.

5.6 Non-Interference Proof

We now show that the overall firewall system fulfills the flow policy sketched in Figure 6. First we formally define the ideal and real composed firewall systems. Next we formalize the flow policy. We then prove that the ideal firewall (consisting of the filtering system and the ideal secure message-transmission system) fulfills the flow policy. With the composition theorem and the non-interference preservation theorem, we finally obtain the result for the real firewall.

The ports of the lower- and higher-layer systems were named so that a composition of Sys'_0 and Sys_1 connects the structures in the desired way according to Figure 8. Thus we get a natural composition by composing every structure of Sys'_0 with the only structure of Sys_1 . We only restrict Sys'_0 to the structures where the two protected users a and b are honest.

Definition 18 (Ideal Firewall System) Let $Sys'_{0,a,b} := \{(\{\text{TH}_{\mathcal{H}}\}, S_{\mathcal{H}}) \in Sys'_0 \mid \{a, b\} \subseteq \mathcal{H}\}$. Then the ideal firewall system is defined as

$$Sys^* = \{(\hat{M}_{\mathcal{H}}^*, S_{\mathcal{H}}^*) \mid \{a, b\} \subseteq \mathcal{H} \subseteq \{1, \dots, n\}\}$$

with $\hat{M}_{\mathcal{H}}^* := \{\text{TH}_{\mathcal{H}}, M_a^{\text{filter}}, M_b^{\text{filter}}\}$ and $S_{\mathcal{H}}^{*c} := \{\text{out}_u^{\text{fil?}}, \text{in}_u^{\text{fil!}}, \text{in}_u^{\text{fil}^{\triangleleft!}} \mid u \in \{a, b\}\} \cup \{\text{out}_u^?, \text{in}_u^!, \text{in}_u^{\triangleleft!} \mid u \in \{1, \dots, n\} \setminus \{a, b\}\}$.

The definition for the real firewall system is analogous:

Definition 19 (Real Firewall System) Let $Sys_{0,a,b} := \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \in Sys_0 \mid \{a, b\} \subseteq \mathcal{H}\}$. Then the real firewall system is defined as

$$Sys^{\#} = \{(\hat{M}_{\mathcal{H}}^{\#}, S_{\mathcal{H}}^*) \mid \{a, b\} \subseteq \mathcal{H} \subseteq \{1, \dots, n\}\}$$

with $S_{\mathcal{H}}^*$ as in the ideal firewall system and $\hat{M}_{\mathcal{H}}^{\#} := \hat{M}_{\mathcal{H}} \cup \{M_a^{\text{filter}}, M_b^{\text{filter}}\}$.

These compositions fulfill the preconditions of the composition theorem from [55].

Given the composition, we can formally present the flow policy for the ideal firewall system Sys^* ; recall Figure 6.

Definition 20 (Flow Policy of the Firewall) We define a flow policy \mathcal{F}^* for Sys^* as follows: For all $(\hat{M}_{\mathcal{H}}^*, S_{\mathcal{H}}^*) \in Sys^*$, let $\mathcal{F}_{S_{\mathcal{H}}^*}^* := (\Delta_{S_{\mathcal{H}}^*}, \rightsquigarrow_{S_{\mathcal{H}}^*})$ with the index set $\mathcal{I}_{S_{\mathcal{H}}^*} := \mathcal{H}$ be defined as follows: For $u \in \{a, b\}$, let $S_u^c := \{\text{out}_u^{\text{fil?}}, \text{in}_u^{\text{fil!}}, \text{in}_u^{\text{fil}^{\triangleleft!}}\}$, and for $u \notin \{a, b\}$, let $S_u^c := \{\text{out}_u^?, \text{in}_u^!, \text{in}_u^{\triangleleft!}\}$. We define $S_u \rightsquigarrow_{S_{\mathcal{H}}^*}^* S_v$ iff $v \in \{a, b\}$ and $u \notin \{a, b\}$.

We can now prove that the ideal firewall system fulfills its flow policy.

Theorem 3 (*Non-Interference Property of the Ideal Firewall*) Let \mathcal{F}^* be the flow policy of Definition 20. Then the system Sys^* fulfills \mathcal{F}^* perfectly.

In the proof of Theorem 3, we use the following lemma.

Lemma 2 Let $(\hat{M}_{\mathcal{H}}^*, S_{\mathcal{H}}^*) \in Sys^*$ arbitrary, and let $\mathcal{F}_{S_{\mathcal{H}}^*}^* := (\Delta_{S_{\mathcal{H}}^*}, \rightsquigarrow_{S_{\mathcal{H}}^*})$ be the flow policy of Definition 20 for this structure. Then the following invariants hold for all possible runs of all non-interference configurations $conf \in \text{Conf}_{\Delta_{S_{\mathcal{H}}^*}, H, L}^{n, in}(\hat{M}_{\mathcal{H}}^*, S_{\mathcal{H}}^*)$ for $H \in \mathcal{H} \setminus \{a, b\}$ and $L \in \{a, b\}$.

1. If H_a receives an input at $\text{out}_a^{\text{fil}!}$, it is of the form $(\text{rec_init}, b)$ or $(\text{receive}, b, m)$ with $m \in \Sigma^+$. If H_a receives an input at $\text{master}_a^?$, it is sent by the master scheduler and equals 1.
2. No output of $X^{\text{n-in}}$ at $\text{master}_a^!$ depends on inputs from other machines. Each machine is clocked equally often by $X^{\text{n-in}}$ using a rotating clocking scheme. Furthermore, each output at a port $p^{\text{cl}!}$ for $p^{\text{cl}} \in S_a^c$ and the scheduled message only depends on prior outputs of H_a at port p^{cl} and $p^!$.
3. If H_a receives a term of the form $(\text{rec_init}, b)$ at $\text{out}_a^{\text{fil}!}$, it is a direct consequence of the input (snd_init) sent by H_b (i.e., the scheduling sequence must have been $H_b, X^{\text{n-in}}, M_b^{\text{filter}}, \text{TH}_{\mathcal{H}}, M_a^{\text{filter}}, H_a$ or $H_b, X^{\text{n-in}}, M_b^{\text{filter}}, M_a^{\text{filter}}, H_a$).
4. If H_a receives a term of the form $(\text{receive}, b, m)$ at $\text{out}_a^{\text{fil}!}$, it is a direct consequence (in the sense of Part 3) of the message (send, a, m) sent by H_b , so the scheduling sequence has been $H_b, X^{\text{n-in}}, M_b^{\text{filter}}, \text{TH}_{\mathcal{H}}, M_a^{\text{filter}}, H_a$ or $H_b, X^{\text{n-in}}, M_b^{\text{filter}}, M_a^{\text{filter}}, H_a$.

The invariants also hold if we exchange the variables a and b .

Part 3 means that the adversary cannot initialize the communication between H_a and H_b . Part 4 ensures that the adversary cannot pretend to be user H_b , and that the number of received messages from H_b equals the number of messages sent by H_b .

Proof Part 1 follows by construction of M_a^{filter} and $X^{\text{n-in}}$. In the initialization transition the only possible output at $\text{out}_a^{\text{fil}!}$ is of the form $(\text{rec_init}, v)$. The test $v = b$ of M_a^{filter} ensures that it is $(\text{rec_init}, b)$. The only remaining step in which M_a^{filter} may output something to H_a is the receive-message step. Outputs are of the form $(\text{receive}, u, m)$. Again, M_a^{filter} checks $u = b$ first so we can only have outputs of the form $(\text{receive}, b, m)$. Thus, every output at port $\text{out}_a^{\text{fil}!}$ has the desired form. The port $\text{master}_a^?$ is connected to the master scheduler, and outputs there are of the form 1 by definition of the master scheduler $X^{\text{n-in}}$.

Part 2 is proved by inspection of the definition of $X^{\text{n-in}}$. At the start of the run, $X^{\text{n-in}}$ schedules BIT_H and switches between “Case 2: Schedule users” and “Case 3: Handle scheduling demands”, cf. Definition 10, afterwards. In case 2, only the internal counter is checked and maybe the counter *steps* of Definition 10 is increased, so no outputs from outside are taken into account. Now assume that $X^{\text{n-in}}$

outputs anything at p^{\downarrow} for $p^{\downarrow} \in S_a^c$. This can only happen in case 3 if $X^{n,\text{in}}$ receives exactly one input at one of the ports p^{\uparrow} for $p^{\uparrow} \in \text{ports}(H_a)$. In this case it schedules the unique output port p^{\downarrow} . Because of $p^{\downarrow} \in \text{ports}(H_a)$ only messages sent by H_a are scheduled. This finishes this sub-part of the proof.

Furthermore, note that no honest user can perform a clocked self-loop by definition. Moreover, the control will automatically come back to $X^{n,\text{in}}$ after an arbitrary user is clocked by the system because users are forbidden to have any clock-out ports by definition. If the adversary is scheduled it either has to do nothing or it has to schedule a machine of the system. In the first case the control immediately goes to the master scheduler, in the second one the machine of the system will either output nothing if one of its internal tests fails, or it finally schedules one of the honest users. In both cases $X^{n,\text{in}}$ is clocked again. This ensures that the master scheduler will always be scheduled after a constant number of steps. Hence it can indeed perform its round-robin clocking scheme, which clocks every machine equally often.

Part 3 holds by construction of M_a^{filter} and $\text{TH}_{\mathcal{H}}$ and the previous part. First note that H_a can only receive a term $(\text{rec_init}, b)$ if it has been output by $\text{TH}_{\mathcal{H}}$ at $\text{out}_a^{\downarrow}$ or by M_b^{filter} at $\text{p}_{M_b}^{\downarrow}$ in the previous step. The second case fulfills our requirements by construction of M_b^{filter} , because a message (snd_init) must have been output by H_b and scheduled by $X^{n,\text{in}}$, so we can turn our attention to the first case. There are only two cases in which $\text{TH}_{\mathcal{H}}$ may have output this term. The first case is initialization of user H_b , the second case is ‘‘Receive initialization’’. In the first case H_b outputs (snd_init) , the master scheduler $X^{n,\text{in}}$ schedules it (if H_b tells him what port to schedule, otherwise nothing is scheduled), and $\text{TH}_{\mathcal{H}}$ directly outputs this term to M_a^{filter} and schedules it immediately. This fulfills our requirements. We will now show that $\text{TH}_{\mathcal{H}}$ does not output anything in the other case.

On input $(\text{rec_init}, b)$ at port $\text{from_adv}_a^?$, $\text{TH}_{\mathcal{H}}$ first checks $\text{stopped}_a^{\text{spec}} = 0$, doing nothing at failure. After a successful test it checks $\text{init}_{b,a}^{\text{spec}} = 0$. By our modification of $\text{TH}_{\mathcal{H}}$ this can only hold if H_b has not initialized itself, so $\text{init}_{b,b}^{\text{spec}} = 0$ must hold. Because of $b \in \mathcal{H}$, $\text{TH}_{\mathcal{H}}$ will not output anything. This finishes the proof of this part.

Part 4 is proved similar to the previous part. First note that H_a can only receive a term $(\text{receive}, b, m)$ if it has been output by $\text{TH}_{\mathcal{H}}$ at $\text{out}_a^{\downarrow}$ or by M_b^{filter} at $\text{p}_{M_b}^{\downarrow}$ in the previous step. An input by M_b^{filter} fulfills our requirements by construction of M_b^{filter} . Note that there are only two cases in which $\text{TH}_{\mathcal{H}}$ may have output this term. The first case is sending messages of user H_b to user H_a , the second case is ‘‘Receive messages from user b ’’. In the first case, H_b sends (send, m, a) , which is again scheduled by $X^{n,\text{in}}$. $\text{TH}_{\mathcal{H}}$ directly outputs this term to M_a^{filter} and schedules it immediately. This fulfills our requirements. Furthermore, it increases the internal counter $\text{msg_out}_{b,a}^{\text{spec}}$. We finally show that $\text{TH}_{\mathcal{H}}$ does not output anything in the second case. On input $(\text{receive_blindly}, b, i)$ at port $\text{from_adv}_a^?$, $\text{TH}_{\mathcal{H}}$ first does its usual initialization checks. We assume them to be successful, otherwise it outputs nothing anyway. It then checks $\text{msg_out}_{b,a}^{\text{spec}} = j$, if $(m, j) := \text{deliver}_{b,a}^{\text{spec}}[i] \neq \downarrow$. However, the message counter $\text{msg_out}_{b,a}^{\text{spec}}$ is set to $\text{msg_in}_{b,a}^{\text{spec}} + 1$ after every sent message from b to a and $j \leq \text{msg_in}_{b,a}^{\text{spec}}$ always holds by construction of

$\text{TH}_{\mathcal{H}}$ for every $(m, j) := \text{deliver}_{b,a}^{\text{spec}}[i] \neq \downarrow$. Hence, we have $j < \text{msg_out}_{b,a}^{\text{spec}}$ and $\text{TH}_{\mathcal{H}}$ will not output anything. This finishes the proof of this part.

Proof (Theorem 3) We have to show that Sys^* fulfills the flow policy \mathcal{F}^* . Let a structure $(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}^*) \in \text{Sys}^*$ be given, and $\mathcal{F}_{S_{\mathcal{H}}^*}^* := (\Delta_{S_{\mathcal{H}}^*}, \rightsquigarrow_{S_{\mathcal{H}}^*})$ the flow policy for this structure. Let two port sets $S_u \not\rightsquigarrow S_v$ be given. Because of the symmetry of the flow policy, we can assume $v = a$, while $u \notin \{a, b\}$. We have to show $(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \models_{\text{perf}} \text{NIRReq}_{\mathcal{F}_{S_{\mathcal{H}}^*}^*, u, v}$.

Let a non-interference configuration $\text{conf} = (\hat{M}_{\mathcal{H}}, S, U^{n\text{-in}}, A_{\mathcal{H}})$ for this structure and high and low user u and v be given. We denote the two families of views of H_a for the initial bit b by $\text{view}_{\text{conf}, b}(H_a)$, and assume a fixed security parameter k . Assume for contradiction that $P_{\text{guess}, \text{conf}}$ is greater than $\frac{1}{2}$. This implies $\text{view}_{\text{conf}, 0}(\{H_a, H_b\}) \neq \text{view}_{\text{conf}, 1}(\{H_a, H_b\})$. This means that there has to be a first input to $\{H_a, H_b\}$ with different probability in both cases. We use Lemma 2 to show that this cannot happen.

By Part 1 of Lemma 2, this input can only be of the form $(\text{rec_init}, c)$ or $(\text{receive}, c, m)$ at $\text{out}_c^{\text{fil}}?$, or 1 at $\text{master}_c?$ for $c \in \{a, b\}$. We write \bar{c} for the other protected user, i.e., $\{c, \bar{c}\} = \{a, b\}$. If the input is $(\text{rec_init}, c)$ Part 3 implies that this input is a direct consequence of an input (snd_init) by user $H_{\bar{c}}$. Hence, there had to be an input to $H_{\bar{c}}$ with different probability in both cases. This contradicts our assumption of the *first* different input.

Now assume this first different input is of the form $(\text{receive}, c, m)$. By Part 4 the corresponding input $(\text{send}, \bar{c}, m)$ must have been sent directly by H_c with the same message m . Furthermore, the underlying trusted host $\text{TH}_{\mathcal{H}}$ for secure reliable message transmission ensures that the message has been sent exactly as often as $H_{\bar{c}}$ receives this input, so there cannot be any influence from outside for the same reason as in the first case. This implies that there already had to be an input to H_c with different probability in both cases, contradicting our assumption of the first different input again.

Finally, assume this input is at port $\text{master}_c?$. By Part 2 this input does not depend on any behaviors of other machines. Analogous to the previous cases, we obtain a contradiction again.

Therefore, we obtain $P_{\text{guess}, \text{conf}} = \frac{1}{2}$. This finishes the proof for conf , and thus the overall proof.

Given that the ideal firewall fulfills its flow policy it follows easily that the real firewall fulfills the corresponding flow policy. However, this only holds computationally because the real firewall is only computationally as secure as the ideal one.

Theorem 4 (Non-Interference Property of the Real Firewall) The real system $\text{Sys}^{\#}$ for the cryptographic firewall fulfills the flow policy $\mathcal{F}^{\#}$ computationally, where $\mathcal{F}^{\#}$ is the corresponding flow policy for \mathcal{F}^* according to Definition 16. In formulas, $\text{Sys}^{\#} \models_{\text{poly}} \mathcal{F}^{\#}$.

Proof According to Theorem 2, the real secure reliable message-transmission system Sys_0 is computationally at least as secure as its specification Sys'_0 . By Part 1

of Lemma 2 the system Sys_1 is polynomial-time. The other, more technical pre-conditions for the composition theorem can easily be seen to be fulfilled. Hence we have $Sys^\# \geq_{\text{poly}} Sys^*$. Since perfect fulfillment of non-interference requirements implies computational fulfillment, we obtain $Sys^\# \models_{\text{poly}} \mathcal{F}^\#$ using Theorem 1.

6 Conclusion

We have presented the first general definition of probabilistic non-interference in reactive systems that includes a computational case (Section 3). We have established a preservation theorem stating that our definition behaves well under simulatability (Section 4); this enables modular proofs of cryptographic systems and step-wise refinement without destroying the non-interference properties. This is particularly important because non-interference properties of abstract specifications of cryptographic systems can often be validated by formal proof tools, whereas real cryptographic systems are much more difficult to validate. As an example, we have presented a cryptographic firewall system and proved a non-interference property via the preservation theorem (Section 5).

Actually using formal proof tools for similar proofs is one possibility for future research. Furthermore, we only considered transitive flow policies so far. However, there are several interesting examples of intransitive flow policies, and several definitions were made for deterministic and non-deterministic system models [58, 56, 57, 61, 41]. We extended these notions to probabilistic and computational models in [3]. Another possible direction is to consider different master schedulers in non-interference configurations and to study to what extent the resulting semantics depends on the scheduling strategy. Finally, the preservation theorem paves the way towards treating cryptographic primitives similar to the Dolev-Yao abstraction again, as in some articles mentioned in the introduction. A first suitable, composable library of cryptographic primitives with a simulatability proof between the ideal and the real version was presented in [5]. This may eventually lead again to a type-based analysis of larger systems that only use well-defined cryptographic primitives as black boxes. Note, however, that the computational non-interference definition and the preservation theorem were necessary to put such an approach on a sound basis.

Acknowledgments

We thank *Heiko Mantel*, *Matthias Schunter*, and *Michael Waidner* for interesting discussions.

References

1. M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. In *Proc. 4th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 2030 of *Lecture Notes in Computer Science*, pages 25–41. Springer, 2001.

2. M. Backes, C. Jacobi, and B. Pfizmann. Deriving cryptographically sound implementations using composition and formally verified bisimulation. In *Proc. 11th Symposium on Formal Methods Europe (FME 2002)*, volume 2391 of *Lecture Notes in Computer Science*, pages 310–329. Springer, 2002.
3. M. Backes and B. Pfizmann. Intransitive non-interference for cryptographic purposes. In *Proc. 24th IEEE Symposium on Security & Privacy*, pages 140–152, 2003.
4. M. Backes, B. Pfizmann, M. Steiner, and M. Waidner. Polynomial fairness and liveness. In *Proc. 15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 160–174, 2002.
5. M. Backes, B. Pfizmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, <http://eprint.iacr.org/>.
6. M. Backes, B. Pfizmann, and M. Waidner. Secure asynchronous reactive systems. IACR Cryptology ePrint Archive 2004/082, Mar. 2004.
7. D. Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.
8. D. Bell and L. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Computer Science Technical Report ESD-TR-75-306, The Mitre Corporation, 1976.
9. M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 419–428, 1998.
10. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998.
11. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.
12. 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. Extended version in Cryptology ePrint Archive, Report 2000/67, <http://eprint.iacr.org/>.
13. R. Canetti and S. Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *Advances in Cryptology: EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 90–106. Springer, 1999.
14. D. Clark, C. Hankin, S. Hunt, and R. Nagarajan. Possibilistic information flow is safe for probabilistic non-interference. In *Proc. WITS*, 2000. www.doc.ic.ac.uk/~clh/papers/witscnh.ps.gz.
15. R. Cramer and V. Shoup. Practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998.
16. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition (extended abstract). In *Proc. 1st ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 11–23, 2003.
17. D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.
18. D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.
19. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

20. R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, 1997.
21. R. Gennaro and S. Micali. Verifiable secret sharing as secure computation. In *Advances in Cryptology: EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 1995.
22. J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. 3rd IEEE Symposium on Security & Privacy*, pages 11–20, 1982.
23. J. A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 75–86, 1984.
24. O. Goldreich. Secure multi-party computation. Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, June 1998, revised Version 1.4 October 2002, 1998. <http://www.wisdom.weizmann.ac.il/users/oded/pp.htm>.
25. O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.
26. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.
27. S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology: CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 1990.
28. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
29. J. W. Gray III. Probabilistic interference. In *Proc. 11th IEEE Symposium on Security & Privacy*, pages 170–179, 1990.
30. J. W. Gray III. Toward a mathematical foundation for information flow security. *Journal of Computer Security*, 1(3):255–295, 1992.
31. M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.
32. C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science, Prentice Hall, Hemel Hempstead, 1985.
33. J. Jacob. Basic theorems about security. *Journal of Computer Security*, 1(4):385–411, 1992.
34. D. M. Johnson and F. Javier Thayer. Security and the composition of machines. In *Proc. 1st IEEE Computer Security Foundations Workshop (CSFW)*, pages 72–89, 1988.
35. M. H. Kang, I. S. Moskowitz, and D. C. Lee. A network version of the pump. In *Proc. 16th IEEE Symposium on Security & Privacy*, pages 144–154, 1995.
36. P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.
37. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.
38. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security analysis. In *Proc. 8th Symposium on Formal Methods Europe (FME 1999)*, volume 1708 of *Lecture Notes in Computer Science*, pages 776–793. Springer, 1999.
39. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Francisco, 1996.
40. H. Mantel. Unwinding possibilistic security properties. In *Proc. 6th European Symposium on Research in Computer Security (ESORICS)*, volume 1895 of *Lecture Notes in Computer Science*, pages 238–254. Springer, 2000.

41. H. Mantel. Information flow control and applications – bridging a gap. In *Proc. 10th Symposium on Formal Methods Europe (FME 2001)*, volume 2021 of *Lecture Notes in Computer Science*, pages 153–172. Springer, 2001.
42. H. Mantel. Preserving information flow properties under refinement. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 78–91, 2001.
43. H. Mantel. On the composition of secure systems. In *Proc. 23rd IEEE Symposium on Security & Privacy*, pages 88–101, 2002.
44. H. Mantel and A. Sabelfeld. A generic approach to the security of multi-threaded programs. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 200–214, 2001.
45. D. McCullough. Specifications for multi-level security and a hook-up property. In *Proc. 8th IEEE Symposium on Security & Privacy*, pages 161–166, 1987.
46. D. McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, 1990.
47. J. McLean. Security models and information flow. In *Proc. 11th IEEE Symposium on Security & Privacy*, pages 180–187, 1990.
48. J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. 15th IEEE Symposium on Security & Privacy*, pages 79–93, 1994.
49. J. McLean. Security models. Chapter in *Encyclopedia of Software Engineering*, 1994.
50. J. McLean. A general theory of composition for a class of "possibilistic" security properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996.
51. S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1991.
52. J. K. Millen. Covert channel capacity. In *Proc. 8th IEEE Symposium on Security & Privacy*, pages 60–66, 1987.
53. A. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, pages 410–442, 2000.
54. B. Pfiztmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000. Extended version (with Matthias Schunter) IBM Research Report RZ 3206, May 2000, http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz.
55. B. Pfiztmann 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.
56. S. Pinsky. Absorbing covers and intransitive non-interference. In *Proc. 16th IEEE Symposium on Security & Privacy*, pages 102–113, 1995.
57. A. Roscoe and M. Goldsmith. What is intransitive noninterference? In *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW)*, pages 226–238, 1999.
58. J. Rushby. Noninterference, transitivity, and channel-control security. Technical report, Computer Science Laboratory, SRI International, 1992.
59. A. Sabelfeld and D. Sands. A per model of secure information flow in sequential programs. In *Proc. European Symposium on Programming (ESOP)*, pages 40–58. Springer, 1999.
60. A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *Proc. 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 200–214, 2000.
61. G. Schellhorn, W. Reif, A. Schairer, P. Karger, V. Austel, and D. Toll. Verification of a formal security model for multiapplicative smart cards. In *Proc. 6th European*

- Symposium on Research in Computer Security (ESORICS)*, volume 1895 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2000.
62. G. Smith. A new type system for secure information flow. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 115–125, 2001.
 63. G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Proc. 25th ACM Symposium on Principles of Programming Languages (POPL)*, pages 355–364, 1998.
 64. D. Sutherland. A model of information. In *Proc. 9th National Computer Security Conference*, pages 175–183, 1986.
 65. D. Volpano. Secure introduction of one-way functions. In *Proc. 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 246–254, 2000.
 66. D. Volpano and G. Smith. Eliminating covert flows with minimum typings. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 156–168, 1997.
 67. D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. In *Proc. 11th IEEE Computer Security Foundations Workshop (CSFW)*, pages 34–43, 1998.
 68. D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.
 69. J. T. Wittbold and D. M. Johnson. Information flow in nondeterministic systems. In *Proc. 11th IEEE Symposium on Security & Privacy*, pages 144–161, 1990.
 70. A. C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.
 71. A. Zakinthinos and E. S. Lee. A general theory of security properties. In *Proc. 18th IEEE Symposium on Security & Privacy*, pages 94–102, 1997.
 72. S. Zdancewic and A. C. Myers. Robust declassification. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 15–23, 2001.

A Ideal Secure Message Transmission with Reliable Channels

This appendix contains the full definition of the ideal lower layer of the firewall system, the secure message-transmission system with reliable channels, as sketched in Section 5.3.

Scheme 2 (Secure, Reliable Message Transmission with Ordered Channels)

Let $n \in \mathbb{N}$ and polynomials $L, s_1, s_2 \in \mathbb{N}[x]$ be given that bound the length of each message and the number of messages a user can send and receive, respectively, from another user. Let $\mathcal{M} := \{1, \dots, n\}$, and fix two elements $a, b \in \mathcal{M}$.

The system is a standard ideal system (see Section 5.3) with the access structure $\mathcal{ACC} := \{\mathcal{H} \subseteq \mathcal{M} \mid a, b \in \mathcal{H}\}$. Thus it is of the form

$$Sys'_0 = \{(\{TH_{\mathcal{H}}\}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}.$$

Specified ports. We define the specified ports by their complements, i.e., the ports the honest users should have: $S_{\mathcal{H}}^c := \{in_u^!, out_u^?, in_u^{<} \mid u \in \mathcal{H}\}$.

Ports of the trusted hosts. The ports of $TH_{\mathcal{H}}$ are $\{in_u^?, out_u^!, out_u^{<} \mid u \in \mathcal{H}\} \cup \{from_adv_u^?, to_adv_u^!, to_adv_u^{<} \mid u \in \mathcal{H}\}$.

State of the trusted hosts. Internally, $\text{TH}_{\mathcal{H}}$ maintains seven arrays:

- $(\text{init}_{u,v}^{\text{spec}})_{u,v \in \mathcal{M}}$ over $\{0, 1\}$ models the initialization state of the users,
- $(\text{stopped}_u^{\text{spec}})_{u \in \mathcal{H}}$ over $\{0, 1\}$ denotes whether the service to user u has stopped,
- $(sc_{u,v}^{\text{in,spec}})_{u \in \mathcal{H}, v \in \mathcal{M}}$ over $\{0, \dots, s_1(k)\}$ counts the inputs of user u intended for user v ,
- $(sc_{u,v}^{\text{out,spec}})_{u \in \mathcal{M}, v \in \mathcal{H}}$ over $\{0, \dots, s_2(k)\}$ counts the outputs for user v originating from user u ,
- $(\text{msg_in}_{u,v}^{\text{spec}})_{u \in \mathcal{H}, v \in \mathcal{M}}$ over $\{0, \dots, s_1(k)\}$ counts the valid sent messages from user u to user v ,
- $(\text{msg_out}_{u,v}^{\text{spec}})_{u,v \in \mathcal{H}}$ over $\{0, \dots, s_2(k)\}$ denotes the next message number expected from u at v , and
- $(\text{deliver}_{u,v}^{\text{spec}})_{u,v \in \mathcal{H}}$ over lists holds the messages in transit from u to v .

The first five arrays are initialized with 0 everywhere, the sixth one with 1 everywhere, and the lists in the seventh are initially empty.

Transition function. The state-transition function of $\text{TH}_{\mathcal{H}}$ is defined by the following rules for the individual inputs. Σ denotes the message alphabet, $\text{len}(m)$ the length of a message and $\text{size}(l)$ the length of a list. The value \downarrow denotes an error. ‘‘Abort’’ means finishing a state transition.

We give explanations in the first transition; the other transitions should then be understood similarly.

- **Send initialization:** On input (snd_init) at $\text{in}_u?$: If $sc_{u,v}^{\text{in,spec}} < s_1(k)$ for all $v \in \mathcal{M}$, set $sc_{u,v}^{\text{in,spec}} := sc_{u,v}^{\text{in,spec}} + 1$ for all $v \in \mathcal{M}$, otherwise abort. This ensures that the number of inputs remains polynomial. Verify $\text{stopped}_u^{\text{spec}} = 0$ and $\text{init}_{u,u}^{\text{spec}} = 0$, i.e., the service for user u is still alive and u has not initialized before. If not, abort. Set $\text{init}_{u,u}^{\text{spec}} := 1$ and output (snd_init) at $\text{to_adv}_u!$, i.e., the adversary learns of the initialization.
If $u = a$, immediately set $\text{init}_{a,b}^{\text{spec}} = 1$ because for this user pair we model reliable communication, and output $(\text{rec_init}, a)$ at $\text{out}_b!$ and 1 at $\text{out}_b^{\triangleleft!}$, and similarly with a and b exchanged. Otherwise output 1 at $\text{to_adv}_u^{\triangleleft!}$.
- **Receive initialization:** On input $(\text{rec_init}, u)$ at $\text{from_adv}_v?$ with $u \in \mathcal{M}, v \in \mathcal{H}$: If $\text{stopped}_v^{\text{spec}} = 0$, $\text{init}_{u,v}^{\text{spec}} = 0$, and $[u \in \mathcal{H} \Rightarrow \text{init}_{u,u}^{\text{spec}} = 1]$, set $\text{init}_{u,v}^{\text{spec}} := 1$, else abort. If $sc_{u,v}^{\text{out,spec}} < s_2(k)$ set $sc_{u,v}^{\text{out,spec}} := sc_{u,v}^{\text{out,spec}} + 1$ and output $(\text{rec_init}, u)$ at $\text{out}_v!$ and 1 at $\text{out}_v^{\triangleleft!}$.
- **Send:** On input (send, m, v) at $\text{in}_u?$: If $sc_{u,v}^{\text{in,spec}} < s_1(k)$, set $sc_{u,v}^{\text{in,spec}} := sc_{u,v}^{\text{in,spec}} + 1$, otherwise abort. Verify that $\text{stopped}_u^{\text{spec}} = 0$, $m \in \Sigma^+$, $l := \text{len}(m) \leq L(k)$, $v \in \mathcal{M} \setminus \{u\}$, $\text{init}_{u,u}^{\text{spec}} = 1$ and $\text{init}_{v,u}^{\text{spec}} = 1$, else abort. Set $\text{msg_in}_{u,v}^{\text{spec}} := \text{msg_in}_{u,v}^{\text{spec}} + 1$.
– If $v \notin \mathcal{H}$, output $(\text{send}, (m, \text{msg_in}_{u,v}^{\text{spec}}), v)$ at $\text{to_adv}_u!$ and 1 at $\text{to_adv}_u^{\triangleleft!}$.
– If $v \in \mathcal{H}$, set $i := \text{size}(\text{deliver}_{u,v}^{\text{spec}}) + 1$ and $\text{deliver}_{u,v}^{\text{spec}}[i] := (m, \text{msg_in}_{u,v}^{\text{spec}})$. Further, output $(\text{send_blindly}, i, l, v)$ at $\text{to_adv}_u!$.
• If $\{u, v\} \neq \{a, b\}$, output 1 at $\text{to_adv}_u^{\triangleleft!}$.

- If $\{u, v\} = \{a, b\}$ set $msg_out_{u,v}^{spec} := msg_in_{u,v}^{spec} + 1$ and output $(receive, u, m)$ at $out_v!$ and 1 at $out_v^{\triangleleft!}$.⁸
- **Receive from honest party u :** On input $(receive_blindly, u, i)$ at $from_adv_v?$ with $u, v \in \mathcal{H}$: Verify that $stopped_v^{spec} = 0$, $init_{v,v}^{spec} = 1$, $init_{u,v}^{spec} = 1$, $sc_{u,v}^{out,spec} < s_2(k)$ and $(m, j) := deliver_{u,v}^{spec}[i] \neq \downarrow$, else abort. Further verify $msg_out_{u,v}^{spec} = j$. If this holds set $sc_{u,v}^{out,spec} := sc_{u,v}^{out,spec} + 1$ and $msg_out_{u,v}^{spec} := j + 1$ and output $(receive, u, m)$ at $out_v!$ and 1 at $out_v^{\triangleleft!}$.
- **Receive from dishonest party u :** On input $(receive, u, m)$ at $from_adv_v?$ with $u \in \mathcal{M} \setminus \mathcal{H}$, $m \in \Sigma^+$, $len(m) \leq L(k)$ and $v \in \mathcal{H}$: If $stopped_v^{spec} = 0$, $init_{v,v}^{spec} = 1$, $init_{u,v}^{spec} = 1$ and $sc_{u,v}^{out,spec} < s_2(k)$, set $sc_{u,v}^{out,spec} := sc_{u,v}^{out,spec} + 1$ and output $(receive, u, m)$ at $out_v!$ and 1 at $out_v^{\triangleleft!}$.
- **Stop:** On input $(stop)$ at $from_adv_u?$ with $u \in \mathcal{H}$: If $stopped_u^{spec} = 0$, set $stopped_u^{spec} := 1$ and output $(stop)$ at $out_u!$ and 1 at $out_u^{\triangleleft!}$.

⁸ Increasing the out-message counter is essential for avoiding replay attacks because the message m is directly delivered to v using a reliable channel.