

Computational Probabilistic Non-Interference

Michael Backes¹ and Birgit Pfitzmann²

¹ Saarland University, Saarbrücken, Germany

² IBM Zurich Research Laboratory, Rüschlikon, Switzerland

¹mbackes@cs.uni-sb.de, ²bpf@zurich.ibm.com

Abstract. In recent times information flow and non-interference have become very popular concepts for expressing both integrity and privacy properties. We present the first general definition of probabilistic non-interference in reactive systems which 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. Furthermore, our definition links non-interference with the common approach of simulatability that modern cryptography often uses. We show that our definition is maintained under simulatability, which allows secure composition of systems, and we present a general strategy how cryptographic primitives can be included in information flow proofs. As an example we present an abstract specification and a possible implementation of a cryptographic firewall guarding two honest users from their environment.

1 Introduction

Nowadays, information flow and non-interference are known as powerful possibilities for expressing privacy and integrity requirements a program or a cryptographic protocol should fulfill. The first models for information flow have been considered for secure operating systems by Bell and LaPadula [3], and Denning [5]. After that, various models have been proposed that rigorously define when information flow is considered to occur. The first one, named *non-interference*, was introduced by Goguen and Meseguer [6, 7] in order to analyze the security of computer systems, but their work was limited to deterministic systems. Nevertheless, subsequent work was and still is based on their idea of defining information flow. After that, research focused on non-deterministic systems, mainly distinguishing between probabilistic and possibilistic behaviors. Beginning with Sutherland [19] the possibilistic case has been dealt with in [15, 20, 16, 22, 14], while the probabilistic and information-theoretic cases have been analyzed by Gray [9, 10] and McLean [17]. Clark et. al. have shown in [4] that possibilistic information flow analysis can be used to check for probabilistic interference.

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. Thus, if we want to consider real cryptography we cannot restrict ourselves to perfect non-interference as captured by the definition of Gray (nor to any other definition mentioned before, because they are non-probabilistic and hence not suited to cope with real cryptography) because it will not be sufficient for most cryptographic purposes. As an example, consider an arbitrary public key encryption scheme. Obviously, an adversary with unlimited computing power can always break the scheme by

computing all possible encryptions of every plaintext and comparing the results with the given ciphertext. Moreover, even polynomially bounded adversaries may have a very small, so-called *negligible* probability of success. Thus, cryptographic definitions usually state that every polynomially bounded adversary can only achieve its goal with a negligible probability. Adopting this notion we present the first general definition of non-interference for this so-called *computational* case. Besides our work, the paper of Laud [12] contains the only definition of non-interference including such a computational case. However, only encryption is covered so far, i.e., other important concepts like authentication, pseudo-number generators, etc. are not considered. Moreover, the definition is non-reactive, i.e., it does not comprise continuous interaction between the user, the adversary, and the system, which is a severe restriction to the set of considered cryptographic systems. Our definition is reactive and comprises arbitrary cryptographic primitives.

In contrast to other definitions, we will not abstract from cryptographic details and probabilism, e.g., by using the common Dolev-Yao abstraction or special type systems, but we immediately include the computational variant in our definition. This enables sound reduction proofs with respect to the security definitions of the included cryptographic primitives (e.g., reduction proofs against the security of an underlying public key encryption scheme), i.e., a 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 the concept of simulatability that modern cryptography often uses, i.e., non-interference properties proved for an abstract specification automatically carry over to the concrete implementation. This theorem is essential since it enables modular proofs in large systems, i.e., proofs done for ideal systems not containing any probabilism simply carry over to their corresponding real cryptographic counterparts. Moreover, properties of these ideal systems could quite easily be proved machine-aided, so our theorem additionally provides a link between cryptography and formal proof tools for non-interference. 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.

Outline of the paper. In Section 2 we briefly review the underlying model of asynchronous reactive system introduced in [18]. The original contributions are presented in Sections 3, 4 and 5. In Section 3 we extend the underlying model to multiple users and we refer to as multi-party configurations; built on this definition we construct our definition of non-interference. In Section 4 we show that our definition behaves well under simulatability, hence refinement does not change the non-interference properties. In Section 5 we present an abstract specification and a possible implementation of a cryptographic firewall guarding two honest users from their environment, and we prove that they fulfill our definition of non-interference.

2 General System Model for Reactive Systems

In this section we briefly recapitulate the model for probabilistic reactive systems as introduced in [18]. All details of the model which are not necessary for understanding are omitted; they can be looked up in the original paper.

Systems mainly are compositions of different machines. Usually we consider real systems consisting of a set \hat{M} of machines $\{M_1, \dots, M_n\}$ and ideal systems built by one machine $\{TH\}$, called *trusted host*. The machine model is probabilistic state-transition machines, similar to I/O automata as introduced in [13]. For complexity we consider every automata to be implemented as a probabilistic Turing machine; complexity is measured in the length of its initial state, i.e. the initial worktape content (often a security parameter k , given in unary representation).

Communication between different machines is done via ports. Inspired by the CSP-Notation [11], we write output and input ports as $p!$ and $p?$, respectively. The ports of a machine M will be denoted by $\text{ports}(M)$. Connections are defined implicitly by naming convention, i.e., port $p!$ sends messages to $p?$. To achieve asynchronous timing, a message is not directly sent to its recipient, but it is first stored in a special machine \tilde{p} called a buffer and waits to be scheduled. If a machine wants to schedule the i -th message of buffer \tilde{p} (this machine must have the unique clock-out port $p^{\dagger!}$) it simply sends i at $p^{\dagger!}$. The i -th message is then scheduled by the buffer and removed from its internal list. In our case, most buffers are either scheduled by a specific master scheduler or the adversary, i.e., one of those has the corresponding clock-out port.

A *collection* \mathcal{C} of machines is a finite set of machines with pairwise different machine names and disjoint sets of ports. The *completion* $[\mathcal{C}]$ of a collection \mathcal{C} is the union of all machines of \mathcal{C} and the buffers needed for every channel.

A *structure* is a pair (\hat{M}, S) , where \hat{M} is a collection of machines and $S \subseteq \text{free}([\hat{M}])$, the so called *specified ports*, are a subset of the free¹ ports of $[\hat{M}]$. Roughly speaking the ports of S guarantee special services to the users. We will always describe specified ports by their complements S^c , i.e., the ports honest users should have. A structure can be completed to a *configuration* by adding special machines H and A , modeling honest users and the adversary. The machine H is restricted to the specified ports S , A connects to the remaining free ports of the structure and both machines can interact. If we now consider sets of structures we obtain a *system* Sys .

Scheduling of machines is done sequentially, so we have exactly one active machine M at any time. If this machine has clock-out ports, it is allowed to select the next message to be scheduled as explained above. If that message exists, it is delivered by the buffer and the unique receiving machine is the next active machine. If M tries to schedule multiple messages, only one is taken, and if it schedules none or the message does not exist, the special master scheduler is scheduled.

Altogether we obtain a runnable system which we refer to as a *configuration* and a probability space over all possible executions (also denoted as *runs* or *traces*) of the system. If we restrict runs to certain sets \hat{M} of machines, we obtain the *view* of \hat{M} . Moreover we can restrict a run r to a set S of ports which is denoted by $r \upharpoonright_S$.

For a configuration $conf$, we furthermore obtain random variables over this probability space which are denoted by $run_{conf,k}$ and $view_{conf,k}$, respectively.

¹ A port is called *free* if its corresponding port is not in the system. These port will be connected to the users and to the adversary.

3 Expressing Non-Interference and Multi-Party Configurations

In this section we define non-interference for reactive systems as introduced in Section 2. At first we look at the more general topic of information flow. Information flow properties consist of two components: a *flow policy* and a *definition of information flow*. Flow policies are built by graphs with two different classes of edges. The first class symbolizes that information may flow between two users, the second class symbolizes that it may not. If we now want to define non-interference, we have to provide a semantics for the second class of edges.² Intuitively, 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 change for every possible behaviour of H_H , i.e., H_L should not be able to distinguish arbitrary two families of views induced by two behaviours of H_H . As we have seen in Section 2, we did not regard different honest users as different machines so far, we just combined them into one machine H . Obviously, this distinction is essential for expressing non-interference, so we first have to define multi-party configurations for the underlying model. Multi-party configurations are defined identically to usual configurations except that we have a set U of users instead of a one-user machine H .

3.1 Multi-Party Configurations

Definition 1. (*Multi-Party Configurations*) A multi-party configuration $conf^{mp}$ 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., $ports(U) \cap forb(\hat{M}, S) = \emptyset$ must hold and the completion $\hat{C} := [\hat{M} \cup U \cup \{A\}]$ is a closed collection. The set of these configurations will be denoted by $Conf^{mp}(Sys)$, those with polynomial-time users and a polynomial-time adversary by $Conf_{poly}^{mp}(Sys)$. We will omit the indices mp and $poly$ if they are clear from the context. \diamond

It is important to note that runs and views are also defined for multi-party configurations because we demanded the completion \hat{C} to be closed.

3.2 Flow Policies

We start by defining the flow policy graph.

Definition 2. (*General Flow Policy*) A general flow policy is a pair $\mathcal{G} = (S, \mathcal{E})$ with $\mathcal{E} \subseteq S \times S \times \{\rightsquigarrow, \not\rightsquigarrow\}$. Thus, we can speak of a graph \mathcal{G} with two different kind of edges: $\rightsquigarrow, \not\rightsquigarrow \subseteq S \times S$. Furthermore we demand $(s_1, s_1) \in \rightsquigarrow$ for all $s_1 \in S$, and every pair (s_1, s_2) of nodes should be linked by exactly one edge, so \rightsquigarrow and $\not\rightsquigarrow$ form a partition of $S \times S$. \diamond

Remark 1. The set S often consists of only two elements $S = \{L, H\}$ which are referred to as low- and high-level users. A typical flow policy would then be given by $L \rightsquigarrow L, L \rightsquigarrow H, H \rightsquigarrow H$, and finally $H \not\rightsquigarrow L$, cf. Figure 1, so there should not be any information flow from high- to low-level users.

² We will not present a semantics for the first class of edges here, because we only focus on absence of information flow in this paper.



Fig. 1. A Typical Flow Policy Graph Consisting of High and Low Users Only.

This definition is quite general since it uses an arbitrary set S . If we want to use it for our purpose, we have to refine it so that it can be applied to a system Sys of our considered model. The intuition is to define a graph on the possible participants of the protocol, i.e., users and the adversary. However, this definition would depend on certain details of the users and the adversary, e.g., their port names, so we specify users by their corresponding specified ports of Sys , and the adversary by the remaining free ports of the system to achieve independency. After that, our flow policy only depends on the ports of Sys .

Definition 3. (*Flow Policy*) Let a structure (\hat{M}, S) be given, and let $\Gamma_{(\hat{M}, S)} = \{S_i \mid i \in \mathcal{I}\}$ denote a partition of S for a finite index set \mathcal{I} , so $\Delta_{(\hat{M}, S)} := \Gamma_{(\hat{M}, S)} \cup \{\bar{S}\}$ is a partition of $\text{free}([\hat{M}])$. A flow policy $\mathcal{G}_{(\hat{M}, S)}$ of the structure (\hat{M}, S) is now defined as a general flow policy $\mathcal{G}_{(\hat{M}, S)} = (\Delta_{(\hat{M}, S)}, \mathcal{E}_{(\hat{M}, S)})$.

The set of all flow policies for a structure (\hat{M}, S) and a partition $\Delta_{(\hat{M}, S)}$ of $\text{free}([\hat{M}])$ will be denoted by $POL_{\hat{M}, S, \Delta_{(\hat{M}, S)}}$. Finally, a flow policy for a system Sys is a mapping

$$\begin{aligned} \phi_{Sys} : \quad Sys &\rightarrow POL_{\hat{M}, S, \Delta_{(\hat{M}, S)}} \\ (\hat{M}, S) &\mapsto \mathcal{G}_{(\hat{M}, S)} \end{aligned}$$

that assigns each structure (\hat{M}, S) a flow policy $\mathcal{G}_{(\hat{M}, S)}$ which is defined on (\hat{M}, S) . \diamond

We will simply write \mathcal{G} , Δ , and \mathcal{E} instead of $\mathcal{G}_{(\hat{M}, S)}$, $\Delta_{(\hat{M}, S)}$, and $\mathcal{E}_{(\hat{M}, S)}$ if the underlying structure is clear from the context. Additionally, we usually consider graphs with the following property: for blocks of ports $S_H, S_L \in \Delta$ with $(S_H, S_L) \in \not\rightsquigarrow$ there should not be a path from S_H to S_L consisting of “ \rightsquigarrow ”-edges only. We will refer to this property as *transitivity property* and speak of a *transitive flow policy*.

The relation $\not\rightsquigarrow$ is the non-interference relation of \mathcal{G} , so for two arbitrary blocks $S_H, S_L \in \Delta$, $(S_H, S_L) \in \not\rightsquigarrow$ means that no information flow must occur directed from the machine connected to S_H to the machine connected to S_L . The notion of a transitive flow policy is motivated by our intuition that if a user H_H should not be allowed to directly send any information to user H_L , he should also not be able to send information to H_L by involving additional users, similar for the adversary.

3.3 Definition of Non-Interference

We still have to define the semantics of our non-interference relation $\not\rightsquigarrow$. Usually, expressing this semantics is the most difficult part of the whole definition. In our underlying model, it is a little bit easier because we already have definitions for runs, views, and indistinguishability that can be used to express the desired semantics.

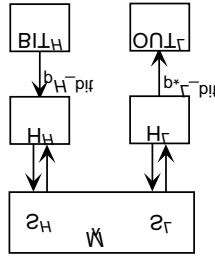


Fig. 2. Sketch of our Non-Interference Definition

Figure 2 contains a sketch of our definition of non-interference between two users H_H and H_L (one of them might also be the adversary). Mainly, we define a specific machine BIT_H , that simply chooses a bit $b \in \{0, 1\}$ at random and outputs it to H_H . Non-interference now means that H_H should not be able to change the view of H_L , so it should be impossible for H_L to output the bit b at $p_{L,bit}^*$ with a probability better than $\frac{1}{2}$ in case of perfect non-interference. Statistical and computational non-interference now means that the advantage of H_L for a correct guess of b should be a function of a class *SMALL* or negligible, respectively, measured in the given security parameter k . The approach of “guessing a bit”, i.e., including the machines BIT_H and OUT_L in our case, is essential to extend the notation of probabilistic non-interference to error probabilities and complexity-theoretic assumptions. Moreover, it is a fundamental concept in cryptography, so our definition additionally serves as a link between prior work in non-interference and real cryptographic primitives along with their security definitions.

These specific configurations including the special BIT_H - and OUT_L -machines will be called *non-interference configurations*. Before we turn our attention to the formal definition of these configurations we briefly describe which machines have to be included, how they behave, and which ports are essential for these sort of configurations, cf. Figure 3.

First of all, we have special machines BIT_H , OUT_L and $X^{n.in}$. As described above, BIT_H will uniformly choose a bit at the start of the run and output it to the user H_H , the second machine simply catches the messages received at port $p_{L,bit}^*$.

The machine $X^{n.in}$ is the master scheduler of the configuration. Its function is to provide liveness properties and to avoid denial of service attacks, so it ensures that every machine will be able to send messages if it wants to. Before we turn our attention to this machine, we briefly describe the ports of the users. In order to improve readability we encourage the reader to compare these descriptions with Figure 3.

At first, we demand that every user must not have any clock-out ports; instead they have additional output ports p^s connected to the master scheduler for every “usual” output port p . The master scheduler will use these ports to schedule outputs from ports p , so a user can tell the master scheduler which port it wants to be scheduled. This is essential for achieving liveness properties, because otherwise, there might be cycles inside of the system, so that neither the master scheduler nor some users will ever be scheduled. Therefore, we explicitly give the control to the master scheduler and define a suitable scheduling strategy in the formal definition.

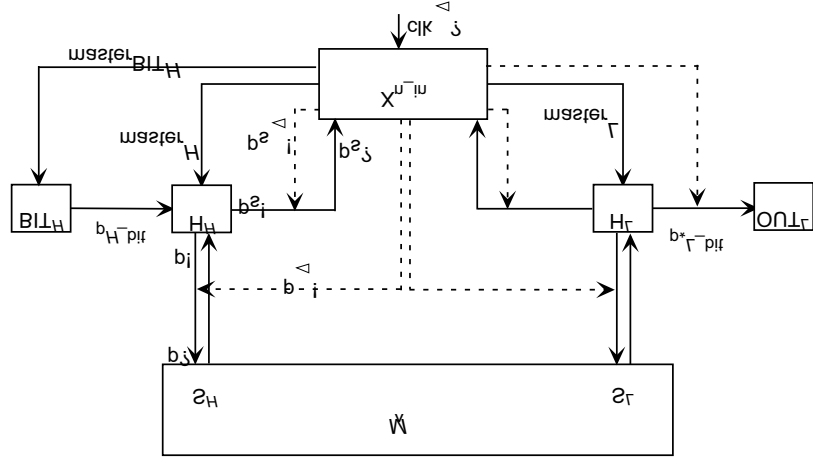


Fig. 3. Main Parts of a Non-Interference Configuration. Additionally, the ports of the master scheduler X^{n-in} and the two emphasized users H_H and H_L are sketched.

We now focus on the ports of the master scheduler. First of all, it has the corresponding input ports $p^{s?}$ for receiving scheduling demands from the users, and the corresponding clock-out ports $p^{s<?}$. Furthermore, it has clock-out ports $p^{<?}$ to schedule every buffer \tilde{p} that delivers messages from the users to the actual system. Finally, it has special ports $master_i^{<?}$ to schedule (or to give control to) the user H_i . The actual scheduling process, i.e., how and in what order users are scheduled will be described in the formal definition.

Definition 4. (*Non-Interference Configuration*) Let a finite index set \mathcal{I} with $A \notin \mathcal{I}$ and $H, L \in \mathcal{I} \cup \{A\}$, $H \neq L$ be given. Furthermore, let a multi-party configuration $conf_{H,L}^{mp} = (\hat{M}, S, U \cup \{BIT_H, OUT_L, X^{n-in}\}, A)$ of a system Sys with $U = \{H_i \mid i \in \mathcal{I}\}$ and a partition $\Delta = \{S_i \mid i \in \mathcal{I}\} \cup \{\bar{S}\}$ of $free([\hat{M}])$ be given. For naming convention we set $H_A := A$ and $S_A := \bar{S}$. We call this configuration a non-interference configuration of Sys if the following holds:

- a) The ports of BIT_H are given by $\{master_{BIT_H}^{<?>}, p_{H-bit}^{<?>}, p_{H-bit}^{<?>}\}$.
The specific machine OUT_L has only one input port $p_{L-bit}^{<?>}$ connected to H_L . The machine X^{n-in} is the master scheduler of the configuration. Its ports are given by
 - $\{clk^{<?>}\}$: The master clock-in port.
 - $\{p^{<?} \mid p^{<?} \in S_i^c, i \in \mathcal{I}\}$: The ports for scheduling buffers between users and the actual system.
 - $\{p^{s?}, p^{s<?} \mid p^{<?} \in S_i^c, i \in \mathcal{I}\}$: The ports connected to the users for receiving scheduling demands.
 - $\{p_{L-bit}^{s*}, p_{L-bit}^{s*}, p_{L-bit}^{<?>}\}$: The ports for scheduling demands and outputs to machine OUT_L .

- $\{\text{master}_i^!, \text{master}_i^{\triangleleft} \mid i \in \mathcal{I} \cup \{A\} \cup \{\text{BIT}_H\}\}$: The ports for scheduling (that is giving control to) the users, the adversary and BIT_H .
- b) For $i \in \mathcal{I}$ the ports of H_i must include $\{p^s! \mid p^{\triangleleft} \in S_i^c\} \cup \{\text{master}_i^?\}$, the ports of the adversary must include $\text{master}_A^?$. Additionally, H_H must have an input port $p_{H_bit}^?$, H_L must have output ports $p_{L_bit}^{*!}$ and $p_{L_bit}^{s*!}$. Furthermore, we demand that the remaining ports of every user and of the adversary connect exactly to their intended subset of specified ports. Formally,

$$\text{ports}(H_H) \setminus (\{p_{H_bit}^?\} \cup \{p^s! \mid p^{\triangleleft} \in S_H^c\} \cup \{\text{master}_H^?\}) = S_H^c \setminus \{p^{\triangleleft}! \mid p^{\triangleleft} \in S_H^c\},$$

$$\begin{aligned} & \text{ports}(H_L) \setminus (\{p_{L_bit}^{*!}, p_{L_bit}^{s*!}\} \cup \{p^s! \mid p^{\triangleleft} \in S_L^c\} \cup \{\text{master}_L^?\}) \\ &= S_L^c \setminus \{p^{\triangleleft}! \mid p^{\triangleleft} \in S_L^c\} \end{aligned}$$

must hold, respectively.

For the remaining users H_i with $i \in \mathcal{I} \cup \{A\}$, $i \notin \{H, L\}$ we simply have to leave out the special bit-ports, i.e., the equation

$$\text{ports}(H_i) \setminus (\{p^s! \mid p^{\triangleleft} \in S_i^c\} \cup \{\text{master}_i^?\}) = S_i^c \setminus \{p^{\triangleleft}! \mid p^{\triangleleft} \in S_i^c\}$$

must hold. Essentially this means that an arbitrary user H_i must not have any clock-out ports and its “usual” simple ports are connected exactly to the simple ports of S_i . The special ports $p_{H_bit}^?$, $p_{L_bit}^{*!}$, $p_{L_bit}^{s*!}$ and ports of the form $\text{master}_i^?$, $p^s!$ are excluded because they must be connected to the master scheduler and to the machines BIT_H and OUT_L .

If the adversary is one of the two emphasized users, i.e., $A \in \{H, L\}$, we have to leave out the term $\{p^s! \mid p^{\triangleleft} \in S_H^c\}$, or $\{p^s! \mid p^{\triangleleft} \in S_L^c\}$, respectively. Alternatively, we can wlog. restrict our attention to those adversaries that do not have such port names which can easily be achieved by consistent port renaming.

- c) The behaviour of the machine BIT_H is defined as follows. If BIT_H receives an arbitrary input at $\text{master}_{\text{BIT}_H}^?$, it chooses a bit $b \in \{0, 1\}$ at random, outputs it at $p_{H_bit}^!$, and schedules it. The machine OUT_L simply does nothing on inputs at $p_{L_bit}^{*!}$. It just “catches” the inputs to close the collection. This ensures that runs and views of the configuration are still defined without making any changes.
- d) The behaviour of the machine $X^{n \cdot n}$ is defined as follows. Internally, it maintains two flags start and fl over $\{0, 1\}$, both initialized with 0, and a counter cnt over the finite index set $\mathcal{I} \cup \{A\}$. Without loss of generality we assume $\mathcal{I} := \{1, \dots, n\}$, so the counter is defined over $\{0, \dots, n\}$, initialized with 0 (identifying the number 0 with A). Additionally, it has a counter *MaSc_poly* if the machine is demanded to be polynomial time, furthermore, a polynomial P must be given in this case that bounds the steps of $X^{n \cdot n}$. If $X^{n \cdot n}$ is scheduled, it behaves as follows:

Case 1: Start of the run. If $\text{start} = 0$: Set $\text{start} := 1$ and output 1 at $\text{master}_{\text{BIT}_H}^!$, 1 at $\text{master}_{\text{BIT}_H}^{\triangleleft}!$.

Case 2: Schedule users. If $\text{fl} = 0$ and $\text{start} = 1$: If $X^{n \cdot n}$ has to be polynomial time, it first checks $\text{cnt} = n$, increasing *MaSc_poly* in this case and checking whether $\text{MaSc_poly} < P(k)$ holds for the security parameter k , stopping at failure. Now, it

sets $cnt := cnt + 1 \bmod (n + 1)$, and outputs 1 at master $_{cnt}!$, 1 at master $_{cnt}^?$. If $cnt \neq 0$, i.e., the clocked machine is an honest user, it additionally sets $fl := 1$ to handle the scheduling demands of this user at its next clocking.

Case 3: Handling scheduling demands. If $fl = 1$ and $start = 1$: In this case it outputs 1 at every port $p^{s?}$ with $p^{?} \in S_{cnt}^c$ (for $cnt = L$ it also outputs 1 at $p_{L,bit}^{s*}$) and tests whether it gets a non-empty input at exactly one input port.³ If this does not hold, it sets $fl := 0$ and does nothing. Otherwise, let $p^{s?}$ denote this port with non-empty input i . $X^{n,in}$ then outputs i at $p^{?}$ and sets $fl := 0$. This case corresponds to a valid scheduling demand of the user, so the corresponding buffer is in fact scheduled.

We obtain a “rotating” clocking scheme on the set $\mathcal{I} \cup \{A\}$, so every user and the adversary will be clocked equally often with respect to the special master-ports.

Non-interference configurations are denoted by $conf_{H,L,\mathcal{I}}^{n,in} = (\hat{M}, S, U^{n,in}, A)_{\mathcal{I}}^{n,in}$ with $U^{n,in} := U \cup \{BIT_H, OUT_L, X^{n,in}\}$ but we will usually omit the index \mathcal{I} . $conf_{H,L}^{n,in}$ is called polynomial-time if its underlying multi-party configuration $conf_{H,L}^{mp}$ is polynomial-time. The set of all non-interference configurations of a system Sys for fixed H, L , and \mathcal{I} will be denoted by $Conf_{H,L,\mathcal{I}}^{n,in}(Sys)$, and the set of all polynomial-time non-interference configurations by $Conf_{H,L,\mathcal{I},poly}^{n,in}(Sys)$. \diamond

Definition 5. (*Non-Interference*) Let a flow policy $\mathcal{G} = (\Delta, \mathcal{E})$ for a structure (\hat{M}, S) be given. Given two arbitrary elements $H, L \in \mathcal{I} \cup \{A\}$, $H \neq L$ with $(S_H, S_L) \in \mathcal{A}$, we say that (\hat{M}, S) fulfills the non-interference requirement $NIREq_{H,L,\mathcal{G}}$

- a) **perfectly** (written $(\hat{M}, S) \models_{\text{perf}} NIREq_{H,L,\mathcal{G}}$) iff for any non-interference configuration $conf_{H,L}^{n,in} \in Conf_{H,L,\mathcal{I}}^{n,in}(Sys)$ of this structure the inequality

$$P(b = b^* \mid r \leftarrow \text{run}_{conf_{H,L}^{n,in},k}; b := r \upharpoonright_{p_{H,bit}!}; b^* := r \upharpoonright_{p_{L,bit}^?}) \leq \frac{1}{2}$$

holds.

- b) **statistically** for a class $SMALL$ ($(\hat{M}, S) \models_{SMALL} NIREq_{H,L,\mathcal{G}}$) iff for any non-interference configuration $conf_{H,L}^{n,in} \in Conf_{H,L,\mathcal{I}}^{n,in}(Sys)$ of this structure there is a function $s \in SMALL$ such that that the inequality

$$P(b = b^* \mid r \leftarrow \text{run}_{conf_{H,L}^{n,in},k}; b := r \upharpoonright_{p_{H,bit}!}; b^* := r \upharpoonright_{p_{L,bit}^?}) \leq \frac{1}{2} + s(k)$$

holds. $SMALL$ must be closed under addition and with a function g also contain every function $g' \leq g$.

- c) **computationally** ($(\hat{M}, S) \models_{\text{poly}} NIREq_{H,L,\mathcal{G}}$) iff for any polynomial-time non-interference configuration $conf_{H,L}^{n,in} \in Conf_{H,L,\mathcal{I},poly}^{n,in}(Sys)$ the inequality

$$P(b = b^* \mid r \leftarrow \text{run}_{conf_{H,L}^{n,in},k}; b := r \upharpoonright_{p_{H,bit}!}; b^* := r \upharpoonright_{p_{L,bit}^?}) \leq \frac{1}{2} + \frac{1}{\text{poly}(k)}$$

³ More formally, it enumerates the ports and sends 1 at the first one. The buffer either schedules a message to $X^{n,in}$ or it does nothing. In both cases $X^{n,in}$ is scheduled again, so it can send 1 at the second clock-out port and so on. Every received message is stored in an internal array so the test can easily be applied.

holds.

We write " \models " if we want to treat all cases together.

If a structure fulfills all non-interference requirements $NIReq_{H,L,G}$ with $(S_H, S_L) \in \not\sim$, we say it fulfills the (global) requirement $NIReq_G ((\hat{M}, S) \models NIReq_G)$. A system Sys fulfills a flow policy ϕ_{Sys} if every structure $(\hat{M}, S) \in Sys$ fulfills its requirement $NIReq_{\phi_{Sys}(\hat{M}, S)}$, and we consequently write $Sys \models NIReq_{\phi_{Sys}(\hat{M}, S)}$, or $Sys \models \phi_{Sys}$ for short.

◇

4 Preservation of Non-Interference under Simulatability

Simulatability essentially means that whatever might happen to an honest user H in a real system Sys_{real} can also happen to the same honest user in an ideal system Sys_{id} . Formally speaking, for every configuration $conf_1$ of Sys_{real} there is a configuration $conf_2$ of Sys_{id} with the same users yielding indistinguishable views of H in both systems [21]. We abbreviate this by $Sys_{real} \geq_{sec} Sys_{id}$ (Sys_{real} is “at least as secure as” Sys_{id}), indistinguishability of the views of H is denoted by $view_{conf_1}(H) \approx view_{conf_2}(H)$. Usually only certain “corresponding” structures (\hat{M}_1, S_1) of Sys_{real} and (\hat{M}_2, S_2) of Sys_{id} are compared. Structures are called corresponding or *validly mapped* if their set of specified ports are equal. In this section we show that our definition of non-interference behaves well under simulatability. More precisely, we will show that the relation “at least as secure as” will not change the non-interference relation between two arbitrary users (one of them might also be the adversary). Usually, defining a cryptographic system starts with an abstract specification of what the system actually should do, possible implementations have to be proven to be at least as secure as this specification. Such a specification usually consists of a monolithic idealized machine that neither contains any cryptographic details nor any probabilism. Thus, it can be validated quite easily by formal proof systems, e.g., formal theorem proving or even automatic model checking, at least if it is not too complex. Hence, it would be of great use to also verify integrity and privacy properties for this idealized machine that will automatically carry over to the real system.

Our theorem states that non-interference properties are in fact preserved under the relation “at least as secure as”. In the proof of the preservation theorem, the following lemma will be needed.

Lemma 1. *The statistical distance $\Delta(\phi(\text{var}_k), \phi(\text{var}'_k))$ for function ϕ of random variables is at most $\Delta(\text{var}_k, \text{var}'_k)$.* □

This is a well-known fact; a proof can be found in, e.g., [8].

Theorem 1. *(Preservation of Non-Interference Properties) Let a flow policy ϕ_{Sys_2} for a system Sys_2 be given, so that $Sys_2 \models \phi_{Sys_2}$ holds. Furthermore, let a system Sys_1 be given with $Sys_1 \geq^f Sys_2$ for a mapping f with $S_1 = S_2$ whenever $(\hat{M}_2, S_2) \in f(\hat{M}_1, S_1)$. Then $Sys_1 \models \phi_{Sys_1}$ for all ϕ_{Sys_1} with $\phi_{Sys_1}(\hat{M}_1, S_1) := \phi_{Sys_2}(\hat{M}_2, S_2)$ for an arbitrary structure $(\hat{M}_2, S_2) \in f(\hat{M}_1, S_1)$. This holds for the perfect, statistical, and the computational case.* □

Proof. We first show that ϕ_{Sys_1} is a well-defined flow policy for Sys_1 under our preconditions. Let an arbitrary structure $(\hat{M}_1, S_1) \in Sys_1$ be given. Simulatability implies that for every structure $(\hat{M}_1, S_1) \in Sys_1$, there exists $(\hat{M}_2, S_2) \in f(\hat{M}_1, S_1)$.

ϕ_{Sys_2} is a flow policy for Sys_2 , so we have a flow policy $\mathcal{G}_{(\hat{M}_2, S_2)} = (\Delta, \mathcal{E})$ for (\hat{M}_2, S_2) . Furthermore, we have $S_1 = S_2$ by precondition, so we can indeed build the same set Γ of blocks on the specified ports and therefore the same partition Δ of the free ports of $[\hat{M}_1]$.⁴ Hence, $\phi_{Sys_1}(\hat{M}_1, S_1)$ is defined on (\hat{M}_1, S_1) , so ϕ_{Sys_1} is a well-defined flow policy for Sys_1 .

We now have show that Sys_1 fulfills ϕ_{Sys_1} . Let a structure $(\hat{M}_1, S_1) \in Sys_1$ and two elements $H, L \in \mathcal{I} \cup \{A\}$, $H \neq L$ with $(S_H, S_L) \in \not\sim$ (with respect to the flow policy $\phi_{Sys_1}(\hat{M}_1, S_1)$) be given. We have to show that (\hat{M}_1, S_1) fulfills the non-interference requirement $NIR_{H,L,G}$.

Let now a non-interference configuration $conf_{H,L,1}^{n-in} = (\hat{M}_1, S_1, U^{n-in}, A)^{n-in} \in \text{Conf}_{H,L,\mathcal{I}}^{n-in}(Sys_1)$ be given. Because of $Sys_1 \geq^f Sys_2$ there is a configuration $conf_{H,L,2} = (\hat{M}_2, S_2, U^{n-in}, A') \in \text{Conf}_{H,L,\mathcal{I}}^{n-in}(Sys_2)$ for $(\hat{M}_2, S_2) \in f(\hat{M}_1, S_1)$ with $view_{conf_{H,L,1}^{n-in}}(U^{n-in}) \approx view_{conf_{H,L,2}}(U^{n-in})$. Moreover, the honest users U^{n-in} are unchanged by simulatability, so $conf_{H,L,2}$ is again a non-interference configuration; hence, we write $conf_{H,L,2}^{n-in}$ in the following instead of $conf_{H,L,2}$. As usual we distinguish between the perfect, statistical, and the computational case. In the computational case, both configurations must be polynomial-time.

In the perfect case, we have $view_{conf_{H,L,1}^{n-in}}(U^{n-in}) = view_{conf_{H,L,2}^{n-in}}(U^{n-in})$ because of $Sys_1 \geq_{\text{perf}}^f Sys_2$. Now, both $b := r \upharpoonright_{p_{H,\text{bit}}}$ and $b^* := r \upharpoonright_{p_{L,\text{bit}}}$ are part of the view of U^{n-in} because BIT_H and OUT_L are elements of U^{n-in} , so we obtain the same probabilities in both configurations. Our precondition $(\hat{M}_2, S_2) \models_{\text{perf}} NIR_{H,L,G}$ and our arbitrary choice of $conf_{H,L,1}^{n-in}$ implies that (\hat{M}_1, S_1) also fulfills $NIR_{H,L,G}$.

We will treat the statistical and the computational case together. In the statistical (computational) case we have $view_{conf_{H,L,1}^{n-in}}(U^{n-in}) \approx_{SMALL} view_{conf_{H,L,2}^{n-in}}(U^{n-in})$ ($view_{conf_{H,L,1}^{n-in}}(U^{n-in}) \approx_{\text{poly}} view_{conf_{H,L,2}^{n-in}}(U^{n-in})$). We assume for contradiction that (\hat{M}_1, S_1) does not fulfill the non-interference requirement $NIR_{H,L,G}$, so the probability $p(k)$ of a correct guess for $b = b^*$ is not smaller than $\frac{1}{2} + s(k)$ for any $s \in SMALL$ in the statistical case (or $p(k) - \frac{1}{2}$ is not negligible in the computational case). Thus, the advantage $\epsilon(k) := p(k) - \frac{1}{2}$ of the adversary is not contained in $SMALL$ (or $\epsilon(k)$ is not negligible). (\hat{M}_2, S_2) fulfills the non-interference requirement, so in this configuration, the advantage $\epsilon'(k)$ for a correct guess is a function of $SMALL$ in the statistical or negligible in the computational case.

We can then define a distinguisher D as follows. Given the views of U^{n-in} in both configurations it explicitly knows the views of BIT_H and OUT_L . Now D outputs 1 if

⁴ More, precisely the block \bar{S}_1 is identified with \bar{S}_2 . The ports of both sets may be different, but this does not matter because our definition of flow policies only uses whole blocks, so the different ports do not cause any trouble.

$b = b^*$ and 0 otherwise. Its advantage in distinguishing is

$$\begin{aligned} & |P(D(1^k, \text{view}_{\text{conf}_{H,L,1}^{n,in},k}(U^{n,in})) = 1) - P(D(1^k, \text{view}_{\text{conf}_{H,L,2}^{n,in},k}(U^{n,in})) = 1)| \\ &= \left| \frac{1}{2} + \epsilon(k) - \left(\frac{1}{2} + \epsilon'(k) \right) \right| = \epsilon(k) - \epsilon'(k). \end{aligned}$$

For the polynomial case, this immediately contradicts our assumption $Sys_1 \geq_{\text{poly}}^f Sys_2$ because $\epsilon(k) - \epsilon'(k)$ is not negligible. For the statistical case, the distinguisher D can be seen as a function on the random variables, so Lemma 1 implies

$$\begin{aligned} & \Delta(\text{view}_{\text{conf}_{H,L,1}^{n,in},k}(U^{n,in}), \text{view}_{\text{conf}_{H,L,2}^{n,in},k}(U^{n,in})) \\ & \geq |P(D(1^k, \text{view}_{\text{conf}_{H,L,1}^{n,in},k}(U^{n,in})) = 1) - P(D(1^k, \text{view}_{\text{conf}_{H,L,2}^{n,in},k}(U^{n,in})) = 1)| \\ &= \epsilon(k) - \epsilon'(k). \end{aligned}$$

But $\epsilon(k) - \epsilon'(k) \notin \text{SMALL}$ must hold, because $\epsilon'(k) \in \text{SMALL}$ and SMALL is closed under addition. Thus, $\Delta(\text{view}_{\text{conf}_{H,L,1}^{n,in},k}(U^{n,in}), \text{view}_{\text{conf}_{H,L,2}^{n,in},k}(U^{n,in})) \notin \text{SMALL}$ because SMALL is closed under making functions smaller which yields the desired contradiction. $(S_H, S_L) \in \mathcal{A}$ and (\hat{M}_1, S_1) have been chosen arbitrary so $(\hat{M}_1, S_1) \models \text{NReq}_{\mathcal{G}}$ and finally $Sys_1 \models \phi_{Sys_1}$ which finishes the proof. \blacksquare

5 A Cryptographic Firewall

In the following we present an example of a system that allows authenticated communications between two users and furthermore ensures that these two users cannot be affected by their environment. This yields a flow policy our system has to (and indeed will) fulfill.

5.1 Some Preliminaries

We start with a brief review on standard cryptographic systems and composition, cf. [18] for more details. In cryptographic protocols every user u usually has exactly one machine M_u and its machine is correct if and only if the user is honest.

The machine M_u of user u has special ports $\text{in}_u?$ and $\text{out}_u!$ for connecting to user u . A standard cryptographic system Sys can now be derived by a *trust model*. The trust model consists of an access structure \mathcal{ACC} and a channel model χ . \mathcal{ACC} is a set of subsets \mathcal{H} of $\{1, \dots, n\}$ and denotes the possible sets of correct machines. For each set \mathcal{H} there will be exactly one structure built by the machines belonging to the set \mathcal{H} . The channel model classifies every connection as either secure (private and authentic), authenticated or insecure. In the considered model these changes can easily be achieved via port renaming (see [18]).

For a fixed set \mathcal{H} and a fixed channel model we obtain modified machines for every machine M_u which we refer to as $M_{u,\mathcal{H}}$. We denote their combination by $\hat{M}_{\mathcal{H}}$ (i.e., $\hat{M}_{\mathcal{H}} := \{M_{u,\mathcal{H}} \mid u \in \mathcal{H}\}$), so real systems are given by $Sys_{\text{real}} = \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}$. Ideal systems typically are of the form $Sys_{\text{id}} = \{(\{\text{TH}_{\mathcal{H}}\}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}$ with the same sets $S_{\mathcal{H}}$ as in the corresponding real system Sys_{real} .

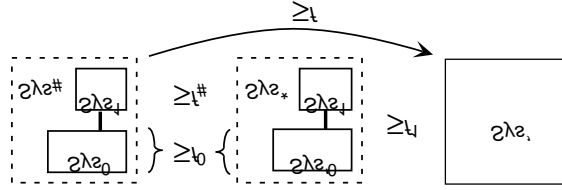


Fig. 4. Composition of Systems

We now briefly review what has already been proven about composition of reactive systems. What we actually want is the relation “at least as secure as” to be consistent with the composition of systems. Assume that we have already proven that a system Sys_0 is at least as secure as another system Sys'_0 . Typically, Sys_0 is a real system whereas Sys'_0 is an ideal specification of the real system. If we now consider larger protocols that use Sys'_0 as an ideal primitive we would like to be able to securely replace it with Sys_0 . In practice this means that we replace the specification of a system with its implementation.

Usually, replacing means we have another system Sys_1 using Sys'_0 ; we call this combination Sys^* . We now want to replace Sys'_0 with Sys_0 inside of Sys^* which gives a combination $Sys^\#$. Typically, $Sys^\#$ is a completely real system whereas Sys^* is at least partly ideal. This fact is illustrated in the left and middle part of Figure 4. The composition theorem now states that this replacement maintains security, i.e., $Sys^\#$ is at least as secure as Sys^* (see [18] for further details).

After this brief review we can turn our attention to the cryptographic firewall. The construction of both our ideal and our real system can be explained using Figure 4. Our ideal specification is based on an ideal specification for secure message transmission with ordered channels introduced in [2] which we will slightly modify to fit our requests. Mainly, we have to model reliable channels to avoid denial of service attacks. We will denote this modified ideal system by Sys'_0 following the notation of Figure 4. Furthermore, a possible implementation has also been presented in [2] which we have to modify similar to the ideal specification to maintain the *at least as secure as* relation.

Our cryptographic firewall will then be derived by defining a new system Sys_1 so that combination with Sys'_0 yields the ideal system, replacing Sys'_0 with Sys_0 finally yields a possible implementation. Sys_1 will be designed to filter messages sent by “wrong” senders that should not be allowed to influence the special users according to the flow policy shown in Figure 5. According to Figure 4, we denote our ideal system as Sys^* and our real implementation as $Sys^\#$.

We start with a brief review of the ideal system for secure message transmission with ordered channels and present our modifications of the system afterwards which will be essential for achieving non-interference. After that, we introduce our system Sys_1 , and briefly sketch the concrete implementation. We then prove that our ideal system Sys^* fulfills its non-interference requirements. Finally, we apply our preservation theorem 1 and conclude that these non-interference requirements carry over to the concrete implementation, which successfully finishes our attempt to design a real example that fits our non-interference definition.

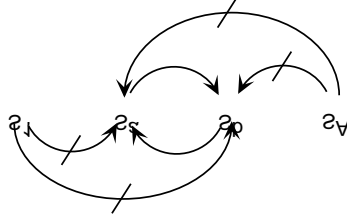


Fig. 5. Sketch of the flow policy of our system. Only one non-emphasized user S_1 is considered and some edges of the graph are omitted. Missing edges are of the form “ \sim ”.

5.2 The Ideal System

Let n denote the number of participants, $\mathcal{I} := \{1, \dots, n\}$ the set of indices of the considered participants, and $\mathcal{I}_A := \mathcal{I} \cup \{A\}$ the set of participants including the adversary. In the following we will identify these indices with their corresponding user. Intuitively, we want a system that fulfills the flow policy shown in Figure 5. We consider two distinguished users a and b with $\{a, b\} \in \mathcal{I}$. We now have two blocks of specified ports S_a and S_b , so that information must not flow to one of these ports from the outside. More precisely, we have non-interference requirements $NIReq_{i_1, i_2, \mathcal{G}}$ for every pair (i_1, i_2) with $i_1 \in \mathcal{I}_A \setminus \{a, b\}$, $i_2 \in \{a, b\}$.

We start with a brief description of the ideal specification for secure message transmission with ordered channels. The specification is of the typical form $Sys'_0 = \{(\{TH_{\mathcal{H}}\}, S_{\mathcal{H}}) | \mathcal{H} \subseteq \mathcal{M}\}$, i.e., there is one structure for every subset of the machines, denoting the honest users.

The ideal machine $TH_{\mathcal{H}}$ models initialization, sending and receiving of messages. A user u can initialize communications with other users by inputting a command of the form (snd_init) to the port $in_u?$ of $TH_{\mathcal{H}}$. In real systems, initialization corresponds to key generation and authenticated key exchange. Sending of messages to a user v is triggered by a command $(send, m, v)$. If v is honest, the message is stored in an internal array $deliver_{u,v}^{spec}$ of $TH_{\mathcal{H}}$ together with a counter indicating the number of the message. After that, a command $(send_blindly, i, l, v)$ is output to the adversary, l and i denote the length of the message m and its position in the array, respectively. This models that the adversary will notice in the real world that a message has been sent and he might also be able to know the length of that message. We speak of tolerable imperfections that are explicitly given to the adversary. Because of the underlying asynchronous timing model, $TH_{\mathcal{H}}$ has to wait for a special term $(receive_blindly, v, i)$ or (rec_init, u) sent by the adversary, signaling, that the message stored at the i th position of $deliver_{u,v}^{spec}$ should be delivered to v , or that a connection between u and v should be initialized. In the first case, $TH_{\mathcal{H}}$ reads $(m, j) := deliver_{u,v}^{spec}[i]$ and checks whether $msg_out_{u,v}^{spec} \leq j$ holds for a message counter $msg_out_{u,v}^{spec}$. If the test is successful the message is delivered and the counter is set to $j + 1$, otherwise $TH_{\mathcal{H}}$ outputs nothing. The condition $msg_out_{u,v}^{spec} \leq j$ ensures that messages can only be delivered in the order they have been received by $TH_{\mathcal{H}}$, i.e., neither replay attacks nor reordering messages is possible for the adversary; cf. [2] for details. The user will receive inputs of the form $(receive, u, m)$ and (rec_init, u) , respectively. If v is dishonest, $TH_{\mathcal{H}}$ will simply output $(send, m, v)$ to the adversary. Finally, the adversary can send a message

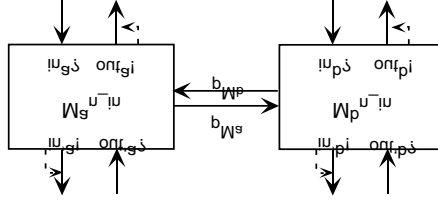


Fig. 6. Sketch of system Sys_1 .

m to a user u by sending a command $(receive, v, m)$ to the port $from_adv_u$? of $TH_{\mathcal{H}}$ for a corrupted user v , and he can also stop the machine of any user by sending a command $(stop)$ to a corresponding port of $TH_{\mathcal{H}}$, which corresponds to exceeding the machine's runtime bound in the real world.

Necessary modifications of the abstract scheme for secure ordered channels. We want our system to fulfill our flow policy shown in Figure 5, so especially the non-interference requirement $NIReq_{A, a, \mathcal{G}}$ must hold. If we explicitly allow the adversary to schedule the communication between H_a and H_b he can obviously achieve two distinguishable behaviours by denial of service attacks as follows. On the one hand, he directly schedules every message sent from H_b to H_a in one behaviour, on the other hand he does not schedule any message sent from H_b to H_a . This problem cannot be solved by the filtering system Sys_1 if scheduling of the communication channel is done by the adversary.⁵ In practice, this means that two persons will not be able to communicate without being interfered from outside if the channel they use can be cut off by the adversary. A possible solution for the problem is to define reliable, non-authenticated channels between a and b , so messages sent between two participants are not only output to the adversary but also output to the recipient and directly scheduled. Obviously, channels which are reliable *and* authenticated could be used as well for sending of messages, but in this case, we would no longer need the underlying cryptography (e.g., authentication). Therefore, we only consider these authenticated channels for key exchange as usual, but sending of messages is still performed over non-authenticated channels. The modifications carry over to the trusted host $TH_{\mathcal{H}}$ as follows:

- If $TH_{\mathcal{H}}$ receives an input (snd_init) from H_a , it implicitly initializes a communication with H_b and outputs (snd_init) to the adversary, (rec_init, a) to H_b and schedules the second output.
- If $TH_{\mathcal{H}}$ receives an input $(send, m, b)$ from H_a , it outputs $(send_blindly, i, l, b)$ to the adversary, $(receive, m, a)$ to H_b scheduling the second output.

These modifications are also done for switched variables a and b . We will omit a more detailed description of the machine $TH_{\mathcal{H}}$ due to lack of space and refer the reader to the long version of this paper. After this brief review and modification of Sys'_0 we can turn our attention to the system Sys_1 . The system Sys_1 is built by additional machines

⁵ Sys_1 can only sort out messages from “wrong” senders, the messages mentioned are sent by the “valid” user H_b , so they have to be delivered because Sys_1 has no internal clock to check for denial of service attacks.

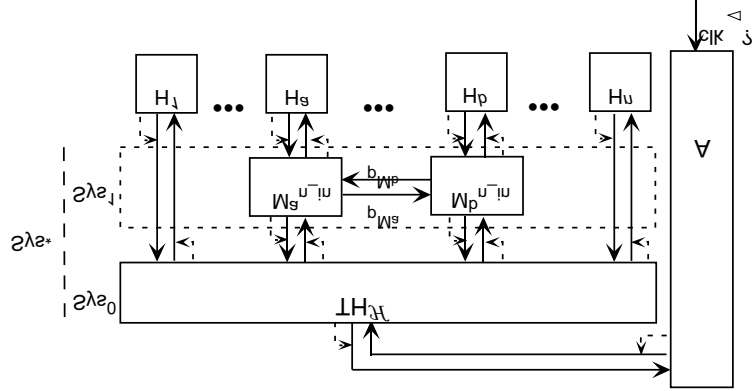


Fig. 7. Ideal System for Non-Interfered Communication.

$M_u^{n,jn}$ for $u \in \{a, b\}$. These machines will be inserted between the users and the trusted host $\text{TH}_{\mathcal{H}}$, see Figure 7. Formally, we obtain the following scheme:

Scheme 1 (Sys_1) Let $n \in \mathbb{N}$ and polynomials $L, s, s' \in \mathbb{N}[x]$ be given. Here n denotes the number of intended participants, $L(k)$ 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 . Let $\mathcal{I} := \{1, \dots, n\}$ denote the set of possible users again and $a, b \in \mathcal{I}$ the special users that should not be influenced from outside. Then

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

with $\hat{M} = \{M_a^{n,jn}, M_b^{n,jn}\}$ and $S^c := \{\text{out}_u?, \text{in}_u!, \text{in}_u^{\triangleleft!} \mid u \in \{a, b\}\} \cup \{\text{in}'_u?, \text{out}'_u!, \text{out}'_u^{\triangleleft!} \mid u \in \{a, b\}\}$. Without loss of generality we just describe the ports and the behaviour of machine $M_a^{n,jn}$. The machine $M_b^{n,jn}$ is defined analogously by exchanging the variables a and b . The ports of machine $M_a^{n,jn}$ are $\{\text{in}_a?, \text{out}_a!, \text{out}_a^{\triangleleft!}\} \cup \{\text{out}'_a?, \text{in}'_a!, \text{in}'_a^{\triangleleft!}\} \cup \{\text{p}_{M_a}?, \text{p}_{M_a}!, \text{p}_{M_a}^{\triangleleft!}\}$, cf. Figure 6.

Internally, $M_a^{n,jn}$ maintains a counter $s_a \in \{0, \dots, s(k)\}$ and an array $(s'_{a,u})_{u \in \mathcal{I}}$ over $\{0, \dots, s'(k)\}$ bounding the number of messages H_a can send and receive, respectively, and a variable $\text{stopped}_a \in \{0, 1\}$ all initialized with 0 everywhere. The state-transition function of $M_a^{n,jn}$ is defined by the following rules, written in a simple pseudo-code language.

Initialization.

- **Send initialization:** On input (snd_init) at $\text{in}_a?$: If $s_a < s(k)$ it sets $s_a := s_a + 1$, otherwise it stops. If $\text{stopped}_a = 0$ it outputs (snd_init) at $\text{in}'_a!$, 1 at $\text{in}'_a^{\triangleleft!}$, otherwise it outputs (snd_init) at $\text{p}_{M_a}!$, 1 at $\text{p}_{M_a}^{\triangleleft!}$.
- **Receive initialization:** On input ($\text{rec_init}, u$) at $\text{out}'_a?$: It first checks whether $s'_{a,u} < s'(k)$ hold. In this case it sets $s'_{a,u} = s'_{a,u} + 1$, otherwise it stops. If $\text{stopped}_a = 0$ it checks $u = b$. If this also holds it outputs ($\text{rec_init}, b$) at $\text{out}_a!$ and 1 at $\text{out}_a^{\triangleleft!}$. On input ($\text{rec_init}, b$) at $\text{p}_{M_b}?$, it outputs ($\text{rec_init}, b$) at $\text{out}_a!$ and 1 at $\text{out}_a^{\triangleleft!}$.

Sending and receiving messages.

- **Send:** On input (send, m, v) at $\text{in}_a?$ with $m \in \Sigma^+$ and $\text{len}(m) \leq L(k)$ it checks whether $s_a < s(k)$. If this holds it sets $s_a := s_a + 1$, otherwise it stops. If $\text{stopped}_a = 0$ holds, it outputs (send, m, v) at $\text{in}'_a!$, 1 at $\text{in}'_a \triangleleft!$. Otherwise it first checks $v = b$. After a successful test it outputs (receive, a, m) at $\rho_{M_a}!$ and 1 at $\rho_{M_a} \triangleleft!$.
- **Receive:** On input (receive, u, m) at $\text{out}'_a?$ it first checks whether $s'_{a,u} < s'(k)$. If this holds it sets $s'_{a,u} := s'_{a,u} + 1$, otherwise it stops. If $u = b$ holds it outputs (receive, b, m) at $\text{out}_a!$ and 1 at $\text{out}_a \triangleleft!$. On input (receive, b, m) at $\rho_{M_b}?$ it outputs (receive, b, m) at $\text{out}_a!$ and 1 at $\text{out}_a \triangleleft!$.
- **Stop:** On input (stop) at $\text{out}'_a?$ or $\rho_{M_b}?$: If $\text{stopped}_a = 0$, it sets $\text{stopped}_a = 1$ and outputs (stop) at $\rho_{M_a}!$ and 1 at $\rho_{M_a} \triangleleft!$.

The special communication ports ρ_{M_a} and ρ_{M_b} are just included to prevent denial of service attacks. We already briefly stated in our review of Sys'_0 that a mighty attacker could simply overpower the machine of an honest user by sending too many messages, i.e., to exceed its runtime bound in the real world. In the ideal system this is modeled by letting the adversary stop arbitrary machines any time he likes. If we now consider an adversary that stops the machine of user a at the very start of the run and another one that never stops this machine, we would certainly obtain different views for this user. This problem cannot really be avoided if we do not provide additional channels for communication that guarantee availability. In practice this would correspond to a connection that contains trash all the time sent by the adversary, so the users (their machines in our case) would certainly look for a new way to communicate. Furthermore, this problem is much weaker in practice than in theory because it ought to be impossible (or at least very difficult) for an adversary to overpower a machine (the machine would surely be able to take countermeasures). If we did not consider these sorts of attacks the ports ρ_{M_a} and ρ_{M_b} could as well be omitted. Finally, a stopped machine $M_a^{n \rightarrow n}$ would want the machine $M_b^{n \rightarrow n}$ also to use the special communication ports, so it will stop the machine as soon it has been stopped itself. Before we now build the combination of both systems to obtain our complete system Sys^* , we rename the ports $\text{in}_u?$, $\text{out}_u!$ and $\text{out}_u \triangleleft!$ of Sys'_0 into $\text{in}'_u?$, $\text{out}'_u!$ and $\text{out}'_u \triangleleft!$, respectively, for $u \in \{a, b\}$ such that Sys'_0 and Sys_1 are connected in the desired way. Furthermore, we restrict the structures of Sys'_0 to all sets \mathcal{H} with $\{a, b\} \subseteq \mathcal{H}$. Combination now means that we combine every structure of Sys'_0 with the (only) structure of Sys_1 . The resulting system $Sys^* = \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \{a, b\} \subseteq \mathcal{H} \subseteq \mathcal{I}\}$ is shown in Figure 7.

Remark 2. It is quite obvious how to modify the system Sys_1 to an arbitrary set of users (instead of $\{a, b\}$) that have to be guarded by the firewall. Moreover we can easily consider multiple disjoint sets of users so that a user can communicate with other users of its own set without being interfered from outside. This corresponds to multiple firewalls and can easily be achieved by modifying the filtering system Sys_1 , so our specification carries over to arbitrary transitive flow policies.

5.3 The Real System

The real system $Sys^\#$ is derived by replacing the ideal system Sys'_0 with its concrete implementation Sys_0 . For our purpose, it is sufficient to give an informal review of the system Sys_0 . The system is a standard cryptographic system of the form $Sys_0 = \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \subseteq \mathcal{M}\}$, i.e., any subset of participants may be dishonest. It uses asymmetric encryption and digital signatures as cryptographic primitives. A user u can let his machine create signature and encryption keys that are sent to other users over authenticated channels $aut_{u,v}$. Furthermore, messages sent from user u to user v will be signed and encrypted by M_u and sent to M_v over an insecure channel $net_{u,v}$, representing the net in the real world. Similar to $TH_{\mathcal{H}}$ each machine maintains internal counters which are used for discarded messages that are out of order. The adversary is able to schedule the communication between the users, and he can furthermore send arbitrary messages m to arbitrary users u for a dishonest sender v .

We now have to implement the modification of the ideal system in our concrete implementation. This can simply be achieved by changing the channel type of the (formerly insecure) channels between a and b to reliable, non-authenticated. This channel type is not comprised by the original model of [18], but it can be defined quite easily, cf. [1]. Moreover, by inspection of the proof of [2] and [18], it is easy to see that the “at least as secure as” relation still holds for these modified systems Sys_0 and Sys_1 with only slight changes in the proof. Therefore, and due to lack of space, we omit the proof here and refer the reader to the long version again.

5.4 Non-Interference Proof

In the following we will show that our abstract system Sys^* (cf. Figure 7) fulfills its non-interference requirements given by the following flow policy. For two given elements $i_1, i_2 \in \mathcal{I} \cup \{A\}$, we define $(S_{i_1}, S_{i_2}) \in \mathcal{R}$ iff $i_1 \in (\mathcal{H} \setminus \{a, b\}) \cup \{A\}$ and $i_2 \in \{a, b\}$. The flow policy is sketched in Figure 5.

Theorem 2. (*Non-Interference Properties of Sys^**) *Let an arbitrary structure $(\{TH_{\mathcal{H}}, M_a^{n_{in}}, M_b^{n_{in}}\}, S_{\mathcal{H}}) \in Sys^*$ be given. For the sake of readability, we set $\hat{M}_{\mathcal{H}} := \{TH_{\mathcal{H}}, M_a^{n_{in}}, M_b^{n_{in}}\}$ in the following. Let a function ϕ_{Sys^*} be given that maps the structures $(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})$ of Sys^* to the flow policy $\mathcal{G}_{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})} = (\Delta_{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})}, \mathcal{E}_{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})})$ as defined above. The partition $\Delta_{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})}$ of $S_{\mathcal{H}}$ is defined by $\Delta_{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})} := \{S_i \mid i \in \mathcal{H}\} \cup \{\bar{S}\}$ with $S_i^c := \{\text{out}_i?, \text{in}_i!, \text{in}_i^{\leftarrow}!\}$ for $i \in \mathcal{H}$ and $S_A := \bar{S} = \text{free}([\hat{M}_{\mathcal{H}}]) \setminus (\bigcup_{i \in \mathcal{H}} S_i)$. Then Sys^* fulfills ϕ_{Sys^*} perfectly. \square*

Before we turn our attention to the proof of Theorem 2, we present the following lemma.

Lemma 2. *By definition of the system, the following invariants hold for all possible runs of the configuration.*

1. *The collection $\{M_a^{n_{in}}, M_b^{n_{in}}\}$, i.e., the system Sys_1 , is polynomial-time.*
2. *If H_a receives an input at $\text{out}_a?$, it is of the form $(\text{rec_init}, b)$ or $(\text{receive}, b, m)$ for an arbitrary $m \in \Sigma^+$. If H_a receives an input at $\text{master}_a?$, it is sent by the master scheduler and is of the form 1.*

3. No output of $X^{n\text{-in}}$ at master a ! depends on inputs from other machines. Each machine is clocked equally often using a rotating clocking scheme. Furthermore, each output at a port p^q ! for $p^q \in S_a^c$ and the scheduled message does only depend on prior outputs of H_a at port p^s ! and p !
4. If H_a receives a term of the form $(\text{rec_init}, b)$ at $\text{out}_a?$, it is a direct consequence of the input (snd_init) sent by H_b (i.e., the scheduling sequence must have been $H_b, X^{n\text{-in}}, M_b^{n\text{-in}}, \text{TH}_{\mathcal{H}}, M_a^{n\text{-in}}, H_a$ or $H_b, X^{n\text{-in}}, M_b^{n\text{-in}}, M_a^{n\text{-in}}, H_a$). This also implies that initializing a communication between H_a and H_b is not possible for the adversary, so there cannot be any replay attacks with initialization commands because they will be sorted out by $\text{TH}_{\mathcal{H}}$.
5. If H_a receives a term of the form $(\text{receive}, b, m)$ at $\text{out}_a?$, it is a direct consequence (in the sense of Point 4) of the message (send, a, m) sent by H_b , so the scheduling sequence has been $H_b, X^{n\text{-in}}, M_b^{n\text{-in}}, \text{TH}_{\mathcal{H}}, M_a^{n\text{-in}}, H_a$ or $H_b, X^{n\text{-in}}, M_b^{n\text{-in}}, M_a^{n\text{-in}}, H_a$. Thus, it is not possible for the adversary to pretend to be user H_b and furthermore the number of received messages of this form equals the number of messages sent by H_b to H_a . Therefore, the adversary can neither replay these messages nor throw them away.

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

The proof is omitted due to lack of space. It will be contained in the long version.

Proof (Theorem 2). We have to show that Sys^* fulfills the non-interference requirement ϕ_{Sys^*} . Let an arbitrary structure $(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \in Sys^*$ be given so we have a flow policy $\mathcal{G}_{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})} = (\Delta_{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})}, \mathcal{E}_{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})})$ for this structure. Let now two arbitrary blocks $S_{i_1}, S_{i_2} \in \Delta_{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})}$ with $i_1 \in (\mathcal{H} \setminus \{a, b\}) \cup \{A\}$, $i_2 \in \{a, b\}$ be given, so $(S_{i_1}, S_{i_2}) \in \mathcal{G}$ must hold. By definition of non-interference showing $(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \models_{\text{perf}} \text{NIREq}_{i_1, i_2, \mathcal{G}}$ is sufficient for proving $Sys^* \models_{\text{perf}} \phi_{Sys^*}$.

Let a non-interference configuration $\text{conf}_{i_1, i_2}^{n\text{-in}} = (\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}, U^{n\text{-in}}, A)^{n\text{-in}}$ for this structure be given. Without loss of generality we can assume $i_2 = a$ because of the symmetry of the flow policy. Depending on the choice of the bit b we denote the two families of views of H_a by $\text{view}_{\text{conf}_{i_1, a, 0}^{n\text{-in}}}(H_a)$ and $\text{view}_{\text{conf}_{i_1, a, 1}^{n\text{-in}}}(H_a)$. Assume for contradiction that the probability of a correct guess $b = b^*$ is greater than $\frac{1}{2}$, which implies $\text{view}_{\text{conf}_{i_1, a, 0}^{n\text{-in}}}(\{H_a, H_b\}) \neq \text{view}_{\text{conf}_{i_1, a, 1}^{n\text{-in}}}(\{H_a, H_b\})$. First of all, we can exclude denial of service attacks applying Part 3 of the above lemma, so there has to be a first input at $\{H_a, H_b\}$ with different probability in both cases because Part 3 ensures that scheduling of messages sent by a user only depends on its own prior behaviour. We will now use the previous lemma to show that this cannot happen.

By Part 2 of Lemma 2 this input can only be of the form $(\text{rec_init}, u)$, $(\text{receive}, u, m)$ at $\text{out}_u?$, or 1 at $\text{master}_u?$ for $u \in \{a, b\}$. We will in the following write \bar{u} for the other emphasized user (i.e., $\bar{u} \in \{a, b\} \setminus \{u\}$). Assume this input to be of the first form. Now Part 4 implies that this input is a direct consequence of an input (snd_init) sent by the other emphasized user $H_{\bar{u}}$. Hence, there had to be an input of $H_{\bar{u}}$ with different probability in both cases which contradicts our assumption of the *first* different input, so there cannot be any influence from outside Sys_1 . Thus, we obtain identical probability distributions for possible inputs of H_a in both cases yielding the desired contradiction.

Now assume this input to be of the form $(\text{receive}, u, m)$. By Part 5 the corresponding input $(\text{send}, \bar{u}, m)$ must have been sent directly by H_u with exactly the same message m . Furthermore, the underlying system for secure ordered channels ensures that the message has been sent exactly that often as H_a receives this input, so there cannot be any influence from outside because of the same reason as in the first case.

Finally, assume this input to be at port master u ?. This input does not depend on arbitrary behaviours of other machines by Part 3 so we obtain identical probability distributions again. Therefore, the views must in fact be identical in both cases so the probability of a correct guess $b = b^*$ is exactly $\frac{1}{2}$. Thus, we have $Sys^* \models_{\text{perf}} \phi_{Sys^*}$. ■

After proving the non-interference property for the ideal specification, we now concentrate on the concrete implementation.

Theorem 3. (*Non-Interference Properties of $Sys^\#$*) *The real system $Sys^\#$ fulfills the non-interference property $\phi_{Sys^\#}$ computationally, with $\varphi_{Sys^\#}$ given as in theorem 1. In formulas, $Sys^\# \models_{\text{poly}} \phi_{Sys^\#}$. □*

Proof. Putting it all together, we know that the original and also the modified real implementation of secure message transmission with ordered channels is computationally at least as secure as its (modified) specification. Using Part 1 of Lemma 2 we know that the system Sys_1 is polynomial-time, which is an essential precondition for applying the composition theorem in the computational case, so we have $Sys^\# \geq_{\text{poly}} Sys^*$. Since perfect fulfillment of non-interference requirements implies computational fulfillment, we have $Sys^\# \models_{\text{poly}} \phi_{Sys^\#}$ using theorem 1. ■

6 Conclusion

We have presented the first general definition of probabilistic non-interference in reactive systems which includes a computational case (Section 3). Our approach is mainly motivated by the concept of simulatability which is fundamental for modern cryptography, and it might help to build a bridge between prior research in the field of information flow and designing systems involving real cryptographic primitives. We have shown that our definition behaves well under simulatability (Section 4), which enables modular proofs and step-wise refinement without destroying the non-interference properties. This is not only important for the development process of cryptographic protocols but also because non-interference properties of ideal systems not containing any cryptographic details can often easily be validated by formal proof tools whereas real systems are usually much more difficult to validate. As an example, we have presented an abstract specification of a cryptographic firewall guarding two honest users from their environment (Section 5). Moreover, we have presented a concrete implementation that also fits our definition, which we have shown using our preservation theorem.

Acknowledgments

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

References

1. M. Backes. Cryptographically sound analysis of security protocols. Ph.D thesis, Computer Science Department, Saarland University, 2002. Available at <http://www-krypt.cs.uni-sb.de/~mbackes/diss.ps>.
2. M. Backes, C. Jacobi, and B. Pfitzmann. Deriving cryptographically sound implementations using composition and formally verified bisimulation. In Proceedings of Formal Methods Europe 2002 (FME'02), Copenhagen, 2002.
3. D. Bell and L. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306, The Mitre Corporation, Bedford MA, USA, March 1976.
4. D. Clark, C. Hankin, S. Hunt, and R. Nagarajan. Possibilistic information flow is safe for probabilistic non-interference. Workshop on Issues in the Theory of Security (WITS'00), available at www.doc.ic.ac.uk/~clh/Papers/witscnh.ps.gz.
5. D. E. Denning. A lattice model of secure information flow. *Communications of the ACM* 19/5 (1976) 236-243.
6. J. A. Goguen and J. Meseguer. Security policies and security models. *IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Washington 1982, 11-20.
7. J. A. Goguen and J. Meseguer. Unwinding and inference control. *IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Oakland 1984, 75-86.
8. O. Goldreich. *Foundations of cryptography: Basic tools*. Cambridge University Press, 2001.
9. J. W. Gray III. Probabilistic interference. *IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos 1990, 170-179.
10. J. W. Gray III. Toward a mathematical foundation for information flow security. *Journal of Computer Security*, Vol.1, no.3,4, 1992, 255-295.
11. C. A. R. Hoare. *Communicating sequential processes*. International Series in Computer Science, Prentice Hall, Hemel Hempstead 1985.
12. P. Laud. Semantics and program analysis of computationally secure information flow. 10th European Symposium On Programming (ESOP 2001), LNCS 2028, Springer-Verlag, Berlin 2001, 77-91.
13. N. Lynch. *Distributed algorithms*. Morgan Kaufmann Publishers, San Francisco 1996.
14. H. Mantel. Unwinding possibilistic security properties. 6th European Symposium on Research in Computer Security (ESORICS'00), Toulouse 2000, 238-254.
15. D. McCullough. Specifications for multi-level security and a hook-up property. *IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Oakland 1987, 161-166.
16. J. McLean. Security models. in: John Marciniak (ed.): *Encyclopedia of Software Engineering*; Wiley Press, 1994.
17. J. McLean. Security models and information flow. *IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos 1990, 180-187.
18. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. *IEEE Symposium on Security and Privacy*, Oakland, May 2001, 184-201.
19. D. Sutherland. A model of information. 9th National Computer Security Conference; National Bureau of Standards, National Computer Security Center, September 1986, 175-183.
20. J. T. Wittbold and D. M. Johnson. Information flow in nondeterministic systems. *IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos 1990, 144-161.
21. A. C. Yao. Protocols for secure computations. 23rd Symposium on Foundations of Computer Science (FOCS) 1982, IEEE Computer Society, 1982, 160-164.
22. A. Zakinthinos and E. S. Lee. A general theory of security properties. *IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Washington 1997, 94-102.