Comparing Two Notions of Simulatability

Dennis Hofheinz and Dominique Unruh

IAKS, Arbeitsgruppe Systemsicherheit, Prof. Dr. Th. Beth,
Fakultät für Informatik, Universität Karlsruhe, Am Fasanengarten 5,
76 131 Karlsruhe, Germany

Abstract. In this work, relations between the security notions *standard simulatability* and *universal simulatability* for cryptographic protocols are investigated.

A simulatability-based notion of security considers a protocol π as secure as an idealization τ of the protocol task, if and only if every attack on π can be simulated by an attack on τ.

Two formalizations, which both provide secure composition of protocols, are common: *standard simulatability* means that for every π-attack and protocol user H, there is a τ-attack, such that H cannot distinguish π from τ. *Universal simulatability* means that for every π-attack, there is a τ-attack, such that no protocol user H can distinguish π from τ.

Trivially, universal simulatability implies standard simulatability. We show: the converse is true with respect to perfect security, but not with respect to computational or statistical security.

Besides, we give a formal definition of a *time-lock puzzle*, which may be of independent interest. Although the described results do not depend on any computational assumption, we show that the existence of a time-lock puzzle gives an even stronger separation of standard and universal simulatability with respect to computational security.

Keywords: Reactive simulatability, universal simulatability, protocol composition.

1 Introduction

Recently, simulatability-based characterizations of security for cryptographic protocols received a lot of attention. In particular, several modelings of multiparty computation have been presented which allow for secure composition of protocols, cf. [PW00,Can00,PW01,Can01,BPW04b]. All these models share the idea of simulatability: a protocol is considered secure only relative to another protocol. That is, a protocol π is as secure as another protocol τ (usually an idealization of the respective protocol task), if every attack on π can be simulated by an attack on τ.

A little more formally, this means that for every adversary A_π attacking π, there is an adversary A_τ (sometimes referred to as the simulator) that attacks τ, such that from an outside view, both attacks and protocols “look the same.” There are different interpretations of what “looking the same” means concretely.

Roughly, the interpretation of [Can01] is the following: π is as secure as τ, iff for every A_π, there is an A_τ such that no protocol user H (this entity is called the environment $\mathcal{Z}$ in [Can01]) is able to distinguish running with π and A_π from running with τ and A_τ.

Although [PW00,PW01,BPW04b] provide this criterion as "universal simulatability," the default notion of security in these works is that of "standard simulatability." Roughly, standard simulatability demands that for every A_π and every protocol user H, there is an A_τ such that H cannot distinguish π and A_π from τ and A_τ. So basically, the difference between these notions is that with standard simulatability, the simulator A_τ may depend on the user H, whereas universal simulatability requests the existence of "user-universal" simulators A_τ.

All presently known proofs (e.g., in [Can01,BPW04a]) that one can securely compose a *polynomial* number of concurrent protocols depend on the fact that the honest user/environment is chosen in dependence of the simulator. Consequently, we do not know how to prove such a composition theorem in the case of standard security.

1.1 Our Results

In this contribution, we study the relation between standard and universal simulatability. Therefore, we focus on the modeling of [BPW04b], which provides both flavors of simulatability. For a relation to the framework [Can01,CLOS02] of universal composability, see Section 1.2.

By definition, universal simulatability implies standard simulatability. We show that even the converse is true when requiring perfect security (i.e., equality of user-views in the definitions). Apart from giving structural insights, this result may be of practical interest: especially when dealing with idealized protocols, often perfect simulatability can be achieved. Our result enables to conduct a (potentially easier) proof of standard simulatability, and then to conclude universal simulatability using Theorem 1.

On the other hand, we can show that standard simulatability does *not* imply universal simulatability with respect to statistical or computational security. For this, we construct a protocol which is secure only with respect to standard simulatability (in the statistical or computational case). This result shows that proofs of universal simulatability can be stronger than proofs of standard simulatability.

Unfortunately, the constructed protocol is not strictly polynomial-time. So in the computational case, one may wish to have a stronger separation by means of a strictly polynomial-time protocol. We provide such a protocol, and prove that it separates standard and universal simulatability in the computational case. As a technical tool, we need the computational assumption of *time-lock puzzles*, cf. [RSW96]. So additionally, we provide a formal definition of a time-lock puzzle, which may be of independent interest.

1.2 Connections to Universal Composability

Although the framework [Can01,CLOS02] of Universal Composability (UC) does not directly provide an equivalent to the notion of standard simulatability, a formulation of standard simulatability there would seem straightforward. As our proofs below do not depend on specific model characteristics, we believe that our proofs can then be adapted to that framework; this would show that standard and universal simulatability can be separated there, too.

However, recently we have been told [Can04] by Ran Canetti, that in a slightly modified UC setting with a different formulation of polynomial-time, the two notions coincide. At a closer look, this is no contradiction to our results. Namely, Canetti proposes a different notion of standard simulatability than used in, e.g., [BPW04b]: in Canetti's formulation, the environment[1] has a runtime bounded polynomially in the length of its auxiliary input, which again is chosen in dependence of the simulator. So effectively, the (polynomial) runtime bound of the environment is chosen after the simulator, whence our proofs do not apply in that case.

However, since we show that our separating examples also hold for the case of honest users H with non-uniform auxiliary input (that does *not* affect H's runtime), they should be applicable to the notion of "Specialized-simulator UC"[2] defined in [Lin03].

1.3 Organisation

Section 2 establishes the equality of standard and universal simulatability for the case of perfect security; in Section 3, a separation of these notions for statistical and computational security is presented. The discussed stronger separation by means of a strictly polynomial-time protocol using time-lock puzzles is investigated in Section 4. This work ends with a conclusion in Section 5. In Appendix A, we briefly review the modeling of [BPW04b].

2 The Perfect Case

We start by relating standard and universal simulatability for the case of perfect security. Perfect security demands that the respective user-views in the compared protocol situations are completely equal. We show that with respect to perfect security, standard and universal simulatability are equivalent notions. For this, we only need to show that standard simulatability already implies universal simulatability—the other direction is trivial from the definitions.

The idea of our proof is to construct a "universal" protocol user H_u, that simply chooses all of its outputs at random, such that any finite sequence of outputs occurs with nonzero probability. In a sense, H_u incorporates all possible protocol users H.

Now standard simulatability implies that there is a simulator which is "good" with respect to this user H_u. But informally, anything H could do will be done by H_u with nonzero probability. Since H_u's views are completely identical in both

[1] i.e., the UC counterpart of the honest user H

[2] This notion is identical to standard simulatability, except that a possible non-uniform auxiliary input for the environment is chosen after the simulator.

protocols, this allows to conclude that this simulator is not only "good" with respect to H_u, but with respect to all possible users H.

Theorem 1. *With respect to perfect security, standard simulatability implies universal simulatability.*

Proof. As a prerequisite, let $\mathcal{D}$ be a probability distribution over Σ^* which satisfies $\mathsf{Pr}[s \leftarrow \mathcal{D}] > 0$ for all $s \in \Sigma^*$. (As in [PW01,BPW04b], Σ denotes the (finite) message alphabet over which messages sent by machines are formed.) Such a $\mathcal{D}$ necessarily exists, since Σ^* is countable.

Let $(\hat{M}_1, S)$ and $(\hat{M}_2, S)$ be structures with $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{perf}} (\hat{M}_2, S)$. Then, let further $(\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$. That is, let H, A_1 be a valid pair of user and adversary for protocol $\hat{M}_1$. Without loss of generality, we assume H to have exactly one self-clocked self-connection (i.e., a connection from H to itself) with name loop, and to have its ports ordered lexicographically.[3] Then the sequence of ports of H only depends on A_1.

Let H_u be a machine with the same port sequence as H, but with a state set Σ^* and initial states $\{\mathsf{1}\}^*$. H_u's transition function makes H_u switch as follows: independent of state and input, H_u's next state and all of its outputs, including outputs on clock ports, are drawn (independently) from $\mathcal{D}$.

Intuitively, H_u is universal in the following sense: for a fixed A_1, H_u is independent of H. H_u's construction guarantees that every finite prefix of H_u-outputs and -states has non-zero probability.

Clearly we have $(\hat{M}_1, S, \mathsf{H}_u, \mathsf{A}_1) \in \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$, which means that $\mathsf{H}_u, \mathsf{A}_1$ is a valid pair of user and adversary for protocol $\hat{M}_1$. Then the standard security $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{perf}} (\hat{M}_2, S)$ which we assumed ensures the existence of an A_2 with

$$view_{(\hat{M}_1, S, \mathsf{H}_u, \mathsf{A}_1)}(\mathsf{H}_u) = view_{(\hat{M}_2, S, \mathsf{H}_u, \mathsf{A}_2)}(\mathsf{H}_u). \tag{1}$$

We will show

$$view_{(\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1)}(\mathsf{H}) = view_{(\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2)}(\mathsf{H}), \tag{2}$$

which suffices to prove $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{uni,perf}} (\hat{M}_2, S)$, since A_2 does not depend on H.

So let $k \in$ be an arbitrary security parameter.[4] The following notation for views of H in protocol runs with A_1 and $\hat{M}_1$, resp. A_2 and $\hat{M}_2$ will simplify the presentation: let $tr_{\mathsf{H}}^{(1)} := view_{(\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1), k}(\mathsf{H})$, and $tr_{\mathsf{H}}^{(2)} := view_{(\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2), k}(\mathsf{H})$. Analogously, we define $tr_u^{(1)}$ and $tr_u^{(2)}$ for views of H_u. For $n \in$, let $(tr)_n$ denote the n-th step in a view tr; $(tr)_{1..n}$ is the n-step prefix of tr. When it is clear that $I \in \Sigma^*$ is a vector of inputs, we write $I \in st$ to denote the event that in a step st, the machine input vector is I.

We prove the following two statements simultaneously by induction over $n \in$:

$$A(n): \qquad \left(tr_{\mathsf{H}}^{(1)}\right)_{1..n} = \left(tr_{\mathsf{H}}^{(2)}\right)_{1..n},$$

[3] Every H can be turned into an H' of this form, so that there is a probability-respecting identification of H-views and H'-views.

[4] Note that this already determines the initial state $\mathsf{1}^k$ for both H and H_u

and

$$B(n): \qquad \forall s: \ \mathsf{Pr}\Big[\Big(tr_{\mathsf{H}}^{(1)}\Big)_{1..n} = s\Big] > 0 \ \Rightarrow \ \mathsf{Pr}\Big[\Big(tr_{u}^{(1)}\Big)_{1..n} = s\Big] > 0.$$

$A(0)$ and $B(0)$ hold trivially. So assume that $A(n)$ and $B(n)$ hold. Let an arbitrary $(n+1)$-step prefix $(st)_{1..n+1}$ with

$$\alpha := \mathsf{Pr}\left[\left(tr_{\mathsf{H}}^{(1)}\right)_{1..n+1} = (st)_{1..n+1}\right] > 0 \tag{3}$$

be given. To show $A(n+1)$, it suffices to prove

$$\mathsf{Pr}\left[\left(tr_{\mathsf{H}}^{(2)}\right)_{1..n+1} = (st)_{1..n+1}\right] = \alpha. \tag{4}$$

To see (4), we first remark that for the machine input vector I_{n+1} in $(st)_{n+1}$, (1) implies

$$\begin{aligned} &\mathsf{Pr}\left[I_{n+1} \in \left(tr_{u}^{(1)}\right)_{n+1} \mid \left(tr_{u}^{(1)}\right)_{1..n} = (st)_{1..n}\right] \\ &= \mathsf{Pr}\left[I_{n+1} \in \left(tr_{u}^{(2)}\right)_{n+1} \mid \left(tr_{u}^{(2)}\right)_{1..n} = (st)_{1..n}\right]. \end{aligned} \tag{5}$$

Here we also need $B(n)$ to be sure that the probabilities of the conditions are not only equal, but also positive. Furthermore, we have for $i \in \{1, 2\}$:

$$\begin{aligned} &\mathsf{Pr}\left[I_{n+1} \in \left(tr_{\mathsf{H}}^{(i)}\right)_{n+1} \mid \left(tr_{\mathsf{H}}^{(i)}\right)_{1..n} = (st)_{1..n}\right] \\ &= \mathsf{Pr}\left[I_{n+1} \in \left(tr_{u}^{(i)}\right)_{n+1} \mid \left(tr_{u}^{(i)}\right)_{1..n} = (st)_{1..n}\right] > 0, \end{aligned} \tag{6}$$

because the distribution on the next user-inputs is completely determined by the history over all preceding user-outputs. The probabilities for the conditions are positive by (3), $A(n)$, and the construction of H_u. From here, $B(n+1)$ follows from the construction of H_u.

We continue proving $A(n+1)$. Combining (5) and (6) yields

$$\begin{aligned} &\mathsf{Pr}\left[I_{n+1} \in \left(tr_{\mathsf{H}}^{(1)}\right)_{n+1} \mid \left(tr_{\mathsf{H}}^{(1)}\right)_{1..n} = (st)_{1..n}\right] \\ &= \mathsf{Pr}\left[I_{n+1} \in \left(tr_{\mathsf{H}}^{(2)}\right)_{n+1} \mid \left(tr_{\mathsf{H}}^{(2)}\right)_{1..n} = (st)_{1..n}\right]. \end{aligned}$$

But since input and current state already determine the distribution on outputs and next states, we have

$$\begin{aligned} &\mathsf{Pr}\left[\left(tr_{\mathsf{H}}^{(1)}\right)_{n+1} = (st)_{n+1} \mid \left(tr_{\mathsf{H}}^{(1)}\right)_{1..n} = (st)_{1..n}\right] \\ &= \mathsf{Pr}\left[\left(tr_{\mathsf{H}}^{(2)}\right)_{n+1} = (st)_{n+1} \mid \left(tr_{\mathsf{H}}^{(2)}\right)_{1..n} = (st)_{1..n}\right]. \end{aligned} \tag{7}$$

Because the probabilities for the respective conditions in (7) are positive and equal by $A(n)$ and 3, this shows (4), and thus $A(n+1)$.

Summarising, $A(n)$ holds for all n, which in particular implies (2), and thus shows the theorem. □

This proof idea does not work in the computational or statistical case. Very informally, H_u behaves like a given user H too seldom; the resulting success to distinguish protocols would be much smaller than that of H.

So one may ask whether Theorem 1 also holds for computational or statistical security. The next section shows that this is not the case.

3 The Statistical and the Computational Case

Recall that simulatability with respect to *statistical security* demands that polynomial prefixes of the user-views in the real, resp. ideal model must be of "small" statistical distance. Here, "small" may denote a negligible or even exponentially small function in k. The following proof deals with negligible functions as those "small" functions. However, the proof carries over to other classes of "small" functions.

On the other hand, for simulatability with respect to *computational security*, users and adversaries are restricted to (strict) polynomial-time. In this case, the user-views in the real, resp. ideal model only need to be computationally indistinguishable.

Here we give a real and an ideal protocol such that the real protocol is as secure as the ideal one with respect to standard simulatability, but not with respect to universal simulatability. Roughly, the ideal protocol asks adversary and user for bitstrings and then outputs who of them gave the longest bitstring. The real protocol does the same, but always outputs "adversary".

A successful simulator must hence be able to give a longer input than the user with overwhelming probability. We show that such a simulator exists for any given user; we also show that there can be no such simulator which gives longer inputs than every user.

Theorem 2. *With respect to computational and statistical security, standard simulatability does not imply universal simulatability. This holds also if we allow non-uniform polynomial-time honest users for the case of computational security.*

Proof. Let $(\hat{M}_1, S)$ be a structure with machines $\hat{M}_1 = \{\mathsf{M}_1\}$ and service ports S, where $S^c = \{\mathsf{user}!, \mathsf{user}^{\triangleleft}!, \mathsf{out}?\}$. The machine M_1 is depicted in Figure 1. M_1 waits for an input h on port $\mathsf{user}?$, and an input a on port $\mathsf{adv}?$; only the first respective input is considered. When having received both such inputs h and a, M_1 outputs and clocks the value $b = 0$ on $\mathsf{out}!$.

Let $(\hat{M}_2, S)$ be a structure with $\hat{M}_2 = \{\mathsf{M}_2\}$. The machine M_2 is identical to M_1, except that the value b that is eventually output on $\mathsf{out}!$ is determined as $b = 1$ if $|h| > |a|$, and $b = 0$ otherwise. So intuitively, $b = 1$ (resp. $b = 0$) indicates that the user (resp. the simulator) delivered the longest bitstring.

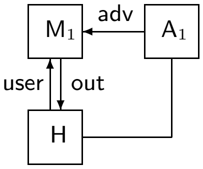

Fig. 1. Machines in the real case.

We claim $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{NEGL} (\hat{M}_2, S)$. So let a real configuration $(\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$ be given. Denote by h_k the random variable that describes M_1's first user?-input in runs with security parameter k, or $\bot$, if there is no user?-input. Since H and A_1 are fixed, there is a function $f : \quad \to \quad$ for which

$$\mathsf{Pr}\left[h_k < f(k) \vee h_k = \bot\right] > 1 - 2^{-k}. \tag{8}$$

Thus, let A_2 be the combination of A_1 and a special machine S, cf. Figure 2. In this combination, the adv! and $\mathsf{adv}^{\triangleleft}$! ports of A_1 are renamed to $\overline{\mathsf{adv}}$! and $\overline{\mathsf{adv}}^{\triangleleft}$!, respectively. The special machine S converts every $\overline{\mathsf{adv}}$?-input into an adv!-output $1^{f(k)}$, which is clocked by S immediately. If we restrict to runs in which either $h_k = \bot$ or $h_k < f(k)$, then we get exactly the same distribution on H-views as in the real configuration. Using (8), we get

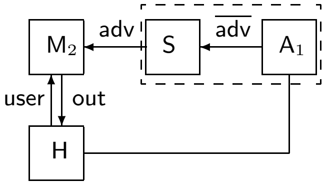

Fig. 2. The simulator for standard simulatability.

$$view_{(\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1)}(\mathsf{H}) \approx_{NEGL} view_{(\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2)}(\mathsf{H}).$$

This implies in particular $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{NEGL} (\hat{M}_2, S)$.

On the other hand, we claim $(\hat{M}_1, S) \not\geq_{\mathsf{sec}}^{\mathsf{uni}, NEGL} (\hat{M}_2, S)$. For this, consider the following real adversary A_1, which is master scheduler and has an additional token connection to the honest user. In its first activation, A_1 outputs and clocks the value $a^{(1)} = 1$ on adv!. In its second activation, it activates H by outputting and clocking 1 on port token!.

Furthermore, for a function $g : \quad \to \quad$, let H_g be the machine which writes and clocks $h = 1^{g(k)}$ onto out! in its first activation. The remaining ports of H_g are chosen to close the collection $\{\mathsf{M}_1, \mathsf{H}_g, \mathsf{A}_1\}$.

We show that for every simulator A_2, there is a function g for which H_g distinguishes M_1 and A_1 from M_2 and A_2. So let A_2 be a simulator for which $(\hat{M}_2, S, \mathsf{H}_1, \mathsf{A}_2) \in \mathsf{Conf}(\hat{M}_2, S)$. (Then $(\hat{M}_2, S, \mathsf{H}_g, \mathsf{A}_2) \in \mathsf{Conf}(\hat{M}_2, S)$ for all g.)

Denote by $a_k^{(2)}$ the random variable that describes the first adv?-input that M_2 gets in runs with security parameter k, or $\bot$, if there is no adv? input. Since A_2 is fixed, there is a function g for which

$$\mathsf{Pr}\left[a_k^{(2)} < g(k) \vee a_k^{(2)} = \bot\right] > 1 - 2^{-k}. \tag{9}$$

In the configuration $(\hat{M}_1, S, \mathsf{H}_g, \mathsf{A}_1)$, H's view in its second activation contains out?-input 0. But by (9), and since the distribution of $a_k^{(2)}$ is independent of g, H_g's view in $(\hat{M}_2, S, \mathsf{H}_g, \mathsf{A}_2)$ contains out?-input 0 with only negligible probability. So $(\hat{M}_1, S) \not\geq_{\mathsf{sec}}^{\mathsf{uni},NEGL} (\hat{M}_2, S)$, and the theorem follows for the statistical case.

The proof above carries over literally to the computational case (with respect to uniform as well as non-uniform honest users), because for polynomial-time H and A, there are polynomials f and g fulfilling (8) and (9). □

4 A Stronger Separation

The proof from the preceding section does not work, if we restrict to protocol machines (i.e., structures) that are strictly polynomial-time. Although the machines M_1 and M_2 used in the proof above are weakly polynomial-time (i.e., they are polynomial-time in the overall length of their inputs and the security parameter, cf. [BPW04b]), at least M_2 needs to accept arbitrarily long inputs.[5] For a separation of the computational simulatability notions by means of strictly polynomial-time structures, we have to work a little harder. As a technical tool, we use time-lock puzzles (see [RSW96]).

Definition 1. *A PPT-algorithm*[6] $\mathcal{G}$ *(called the problem generator) together with a PPT-algorithm* $\mathcal{V}$ *(the solution verifier) is called a* time-lock puzzle *iff the following holds:*

- sufficiently hard puzzles*: for every PPT-algorithm* B *and every* $e \in$ *, there is some* $c \in$ *with*

$$\sup_{t \geq k^c, |h| \leq k^e} \mathsf{Pr}\left[(q, a) \leftarrow \mathcal{G}(1^k, t) : \ \mathcal{V}(1^k, a, B(1^k, q, h)) = 1\right] \tag{10}$$

  negligible in k*.*
- sufficiently good solvers*: there is some* $b \in$ *such that for every* $d \in$ *there is a PPT-algorithm* C *such that*

$$\min_{t \leq k^d} \mathsf{Pr}\left[(q, a) \leftarrow \mathcal{G}(1^k, t); c \leftarrow C(1^k, q) : \ \mathcal{V}(1^k, a, c) = 1 \wedge |c| \leq k^b\right] \tag{11}$$

  is overwhelming in k*.*

[5] However, intuitively, nothing “superpolynomial” happens: M_2 determines the length of its inputs.

[6] Probabilistic polynomial time algorithm

Intuitively, $\mathcal{G}(\mathtt{1}^k, t)$ generates a puzzle q of hardness t, and a description a of valid solutions for q. $\mathcal{V}(\mathtt{1}^k, a, b)$ verifies if b is a valid solution as specified by a.

First, we require that any given PPT-algorithm B can't solve sufficiently hard puzzles. Formally, we want B to be unable to solve puzzles which are of hardness t, $t \geq k^c$ for some c depending on B.

We add an auxiliary input h (of polynomial length) to prevent the following scenario: Bob (B) wants to show to Alice, that he is able to perform calculations of some hardness t, therefore *he* chooses t, and then Alice ($\mathcal{G}$) chooses the puzzle. It is now imaginable, that Bob may choose some auxiliary information h and the hardness t simultaneously, s.t. using h one can solve time-lock puzzles of hardness t.[7] This is prevented by our definition, since (10) is negligible even in presence of polynomially length-bounded auxiliary inputs h.[8]

Second, we demand that for every polynomial hardness value, there is an algorithm C solving puzzles of this hardness. It is sensible here to ask for short solutions (i.e., $|c| \leq k^b$): otherwise, the definition allows time-lock puzzles in which the solution of every t-hard puzzle is deterministically $\mathtt{1}^t$.[9]

[RSW96] promotes the following family of puzzles as candidates for time-lock puzzles. A puzzle of hardness t consists of the task to compute $2^{2^{t'}} \bmod n$ where $t' := \min\{t, 2^k\}$ and $n = pq$ is a Blum integer.[10] In our notation, this is denoted by $\mathcal{G}(\mathtt{1}^k, t) = ((n, \min\{t, 2^k\}), (p, q, \min\{t, 2^k\}))$, where n is a random k-bit Blum integer with factorisation $n = pq$, and $\mathcal{V}(\mathtt{1}^k, (p, q, t'), c) = 1$ if and only if $c \equiv 2^{2^{t'}} \bmod pq$. (Note that $2^{t'} \bmod \varphi(pq)$ can be efficiently computed with knowledge of p and q.)

We return to the problem of separating standard and universal simulatability by means of strictly polynomial-time structures. The idea is very similar to the one used in the proof of Theorem 2. In the ideal setting, we let a machine M_2 check and output whether H has more "computational power" than the adversary A_2. The corresponding real machine M_1 always outputs "no". Here we have standard simulatability, since A_2 can be chosen in dependence of H (and thus more powerful). On the other hand, the simulatability is not universal, since an A_2-dependent user H can be chosen so powerful that M_2 outputs "yes".

Now time-lock puzzles are exactly what M_2 needs to check the computational power of A_2 and H. Concretely, M_2 simply picks a puzzle for both A_2 and H and outputs "yes" if H is the *only* one to solve that puzzle. Our definition of time-lock

[7] Imagine that, e.g., being able to find the pre-image of t under some function would already solve the puzzle.

[8] Note that for the proof of Theorem 3, this additional constraint is not necessary.

[9] In fact, using such a "degenerate" puzzle would yield a proof for Theorem 2, very similar to that given in Section 3.

[10] One might wonder why our formulation uses $t' = \min\{t, 2^k\}$ instead of t (in contrast to the original formulation in [RSW96]). It can be shown that for $t := (2^k)! + 1$ it is $2^{2^t} \equiv 4 \mod n$ for all k-bit Blum integers. Therefore (10) would be violated (consider $B(\mathtt{1}^k, q, h) = 4$). For practical purposes, this does not pose a threat, since the length of $t = (2^k)! + 1$ is not polynomial in k.

puzzles guarantees that puzzles can be generated and solutions can be checked by a strictly polynomially bounded M_2.

An exact theorem statement and a proof follow:

Theorem 3. *Assume that time-lock puzzles exist. Then for computational security, standard simulatability and universal simulatability can be separated by two strictly polynomial-time structures. This holds also if we allow non-uniform polynomial-time honest users.*

Proof. First, let $\mathcal{D}$ denote a PPT-algorithm which, upon input 1^k, returns a uniformly chosen t from $\{2^1, 2^2, \ldots, 2^k\}$.

Let $(\mathcal{G}, \mathcal{V})$ be a time-lock puzzle. Let $(\hat{M}_1, S)$ be a structure with machines $\hat{M}_1 = \{\mathsf{M}_1\}$ and service ports S, where $S^c = \{\mathsf{user}!, \mathsf{user}^{\triangleleft}!, \mathsf{puz\_user}?, \mathsf{out}?\}$.

The machine M_1 is depicted in Figure 3. All indicated connections (not counting the potential connections between A_1 and H) are clocked by the sending machine.

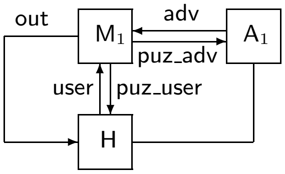

Fig. 3. Machines in the real case.

Upon its first activation, M_1 chooses $t \leftarrow \mathcal{D}(1^k)$. Then it computes $(q, a) \leftarrow \mathcal{G}(1^k, t)$.

Upon the first input via $\mathsf{user}?$, M_1 sends and clocks q via $\mathsf{puz\_user}$ to H. Upon the first input via $\mathsf{adv}?$, M_1 sends and clocks q via $\mathsf{puz\_adv}$ to A_1.

The second input via $\mathsf{user}?$ (called c_{H}) is verified via $v_{\mathsf{H}} \leftarrow \mathcal{V}(1^k, a, c_{\mathsf{H}})$, and analogously we set $v_{\mathsf{A}} \leftarrow \mathcal{V}(1^k, a, c_{\mathsf{A}})$ upon the second $\mathsf{adv}?$-input c_{A}.

At the time both v_{H} and v_{A} are determined, M_1 outputs and clocks $b = 0$ on $\mathsf{out}!$.

Let $(\hat{M}_2, S)$ be a structure with machines $\hat{M}_2 = \{\mathsf{M}_2\}$. The machine M_2 is identical to M_1, except that the value $b \in \{0, 1\}$ that is eventually output on $\mathsf{out}!$ is determined as an evaluation of the predicate $v_{\mathsf{H}} \wedge \neg v_{\mathsf{A}}$. Intuitively, $b = 1$ happens (i.e., H can distinguish) if and only if H is able to successfully solve harder puzzles than the adversary.

Note that by setting suitable length functions (i.e., $l(\mathsf{user}?) = l(\mathsf{adv}?) = k^b$ for the $b \in$ from (11), M_1 and M_2 can be made polynomial-time.

The rest of this proof follows an idea similar to that of the proof of Theorem 2. For showing that $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{poly}} (\hat{M}_2, S)$, we construct a simulator A_2 that can solve any puzzle H can solve, and for showing $(\hat{M}_1, S) \not\geq_{\mathsf{sec}}^{\mathsf{uni,poly}} (\hat{M}_2, S)$ we construct a H which can solve some puzzles A_2 cannot solve.

For showing $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{poly}} (\hat{M}_2, S)$, let a real configuration $(\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$ be given.

We sketch how to view H as a PPT-algorithm B: $B(1^k, q', h)$ simulates a run of the collection $[\{\mathsf{H}, \mathsf{A}_1, \mathsf{M}_1\}]$ with security parameter k (auxiliary input h is ignored); only a possible message q from M_1 to H or A_1 on $\mathsf{user\_puz}$ or $\mathsf{adv\_puz}$ is substituted by q'. B outputs H's answer a_{H} (i.e., M_1's second $\mathsf{user?}$-input), or $\perp$, if M_2 never receives such an a_{H}.

Let $f(k) = k^c$ for the $c \in$ that arises from (10) for this B. Let C be the PPT-algorithm from (11) when setting $d = c$. So intuitively, C is able to solve (except with negligible error) any puzzle that H can solve. Similar to the construction of the simulator A_2 in the proof of Theorem 2 (cf. also Figure 2), let S be a machine which places itself between A_1 and M_1. A_1's ports $\mathsf{puz\_adv?}$, $\mathsf{adv!}$, and $\mathsf{adv}^{\triangleleft}\mathsf{!}$ are renamed to $\overline{\mathsf{puz\_adv}}\mathsf{?}$, $\overline{\mathsf{adv}}\mathsf{!}$, and $\overline{\mathsf{adv}}^{\triangleleft}\mathsf{!}$, respectively. Finally, A_2 is the combination of A_1 and this machine S.

The idea of S is simple: Whenever A_1 sends a (possibly wrong) solution of a puzzle of hardness $f(k)$ to M_2, S solves the puzzle and sends a correct solution to M_2. This will allow to show indistinguishability, since H will only notice that it runs with the ideal protocol if A_2 sends a wrong solution to M_2 for a puzzle H was able to solve.

More formally, S immediately forwards all messages from M_2 to A_2. However, the first message q on $\mathsf{adv\_puz}$ is stored. When A_1 sends the second message to M_2 via adv, that message is replaced by a solution $c \leftarrow C(1^k, q)$ and then forwarded to M_2.

Using a suitable length function, S can be made polynomial-time. Similar to Figure 2, let A_2 be the combination of S and A_1 (with renamed ports). S only substitutes messages between M_1 and A_1, and does not change the scheduling.

We can now observe the following: First, if we can show that for the output $b = v_{\mathsf{H}} \wedge \neg v_{\mathsf{A}}$ by M_2, it is $b = 1$ with only negligible probability, then H's views when running with the real and the ideal protocol are indistinguishable, and thus $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{poly}} (\hat{M}_2, S)$.

Second, consider the definition of B. We can define B' completely analogous, using A_2 and M_2 instead of A_1 and M_1. By noticing that H's answer to the puzzle is chosen before M_1 or M_2 outputs b, we see that B and B' have the same output distributions.

Let v_{A} and v_{H} denote the corresponding predicates calculated by M_2 in a run of the ideal protocol, where we set $v_{\mathsf{A}} = \perp$ and $v_{\mathsf{H}} = \perp$ if the respective variable predicate is never determined (this can happen only when no answer to the respective puzzle is made).

By (10), the following is negligible:

$$\begin{aligned}
&\sup_{t \geq k^c, |h| \leq k^e} \mathsf{Pr}\left[(q,a) \leftarrow \mathcal{G}(1^k, t) :\ \mathcal{V}(1^k, a, B(1^k, q, h)) = 1\right] \\
&\quad \geq \mathsf{Pr}\left[t \leftarrow \mathcal{D}(1^k), (q,a) \leftarrow \mathcal{G}(1^k, t) :\ \mathcal{V}(1^k, a, B(1^k, q, 0)) = 1 \ \wedge\ t \geq k^c\right] \\
&\quad \overset{(*)}{=} \mathsf{Pr}\left[v_{\mathsf{H}} = 1 \ \wedge\ t \geq k^c\right] \\
&\quad \geq \mathsf{Pr}\left[v_{\mathsf{H}} = 1 \ \wedge\ v_A = 0 \ \wedge\ t \geq k^c\right]. \qquad (12)
\end{aligned}$$

(Note that in the last two terms of (12), t denotes the t chosen by M_2.) These inequalities hold for all sufficiently large k. To see $(*)$, note that M_2 chooses t, q, a via $t \leftarrow \mathcal{D}(1^k)$, $(q, a) \leftarrow \mathcal{G}(1^k, t)$, and then calculates v_{H} via $\mathcal{V}(1^k, a, a_{\mathsf{H}})$. Further $B(1^k, q, 0)$ generates a_{H} by simulating the ideal configuration for given q, so $(*)$ follows.

Since A_2 solves the puzzle with overwhelming probability for $t < k^c$ (it uses C, which again does so by (11)), the following is negligible:

$$\begin{aligned} &\mathsf{Pr}\left[v_{\mathsf{A}} = 0 \ \wedge \ t < k^c\right] \\ \geq\ &\mathsf{Pr}\left[v_{\mathsf{H}} = 1 \ \wedge \ v_{\mathsf{A}} = 0 \ \wedge \ t < k^c\right]. \end{aligned} \tag{13}$$

By (12,13), the probability $\mathsf{Pr}\left[v_{\mathsf{H}} = 1 \ \wedge \ v_{\mathsf{A}} = 0\right]$ is negligible, too, therefore we can conclude $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{poly}} (\hat{M}_2, S)$.

If we allow non-uniform honest users, we have to modify the definition of B as follows: $B(1^k, q', h)$ runs a simulation as above, but now h is given to the non-uniform honest user as auxiliary input. Then, for some function $\tilde{h}$ (mapping the security parameter to the auxiliary input), $B(1^k, q', \tilde{h}(k))$ simulates the network containing the non-uniform honest user H. Replacing $B(1^k, q, 0)$ with $B(1^k, q, \tilde{h}(k))$ in (12) yields a valid proof for the non-uniform case.

Note that this construction even applies when the auxiliary input $\tilde{h}$ of the honest user H is chosen in dependence of the simulator (as with the "Specialized-simulator UC" formulation in [Lin03]).

The remaining statement $(\hat{M}_1, S) \not\geq_{\mathsf{sec}}^{\mathsf{uni,poly}} (\hat{M}_2, S)$ can be shown in a similar way: We define a family of honest users H_d, such that no simulator will be able to solve all puzzles these H_d can solve. Formally:

For any d, let C_d be the puzzle-solver C whose existence is guaranteed by (11) for that d. Then H_d is the user having ports $\{\mathsf{user!}, \mathsf{user}^{\triangleleft}\mathsf{!}, \mathsf{out?}, \mathsf{puz\_user?}, \mathsf{init?}\}$ and running the following program: Upon the first activation via $\mathsf{init?}$, send (and schedule) a non-empty message to M_1. When receiving q via $\mathsf{puz\_user?}$ from M_1, let $c \leftarrow C_d(1^k, q)$ and send (and schedule) c to M_2.

The real adversary A_1 has ports $\{\mathsf{adv!}, \mathsf{adv}^{\triangleleft}\mathsf{!}, \mathsf{init!}, \mathsf{init}^{\triangleleft}\mathsf{!}, \mathsf{puz\_adv?}\}$ and runs the following program: Upon the first activation, activate H_d via init, and upon any further activation send and schedule 1 to M_2 via adv.

The resulting network is depicted in Figure 4.

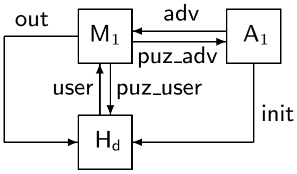

Fig. 4. The configuration conf_1^d.

We now assume for contradiction that $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{uni,poly}} (\hat{M}_2, S)$. Because then $\mathsf{conf}_1^d := (\hat{M}_1, S, \mathsf{H}_d, \mathsf{A}_1) \in \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$ for all d, there is a polynomial simulator A_2, s.t. for all d, $\mathsf{conf}_2^d := (\hat{M}_2, S, \mathsf{H}_d, \mathsf{A}_2) \in \mathsf{Conf}(\hat{M}_2, S)$ and

$$\mathit{view}_{\mathsf{conf}_1^d}(\mathsf{H}_d) \approx_{\mathsf{poly}} \mathit{view}_{\mathsf{conf}_2^d}(\mathsf{H}_d). \tag{14}$$

Similar to the construction of B in the first part of this proof, let $B_d(1^k, q', h)$ simulate a run of the collection $[\{\mathsf{H_d}, \mathsf{A}_2, \mathsf{M}_2\}]$ with security parameter k (auxiliary input h is ignored); only a possible message q from M_2 to H or A_1 on user_puz or adv_puz is substituted by q'. B_d outputs A_2's answer a_{A} (i.e., M_2's second adv?-input), or $\perp$, if M_2 never receives a_{A}.

It is easy to see that A_2's answers do not depend on H_d's answers, therefore B_d is independent of d.

Now, by (10), there is some c, s.t.

$$\sup_{t \geq k^c, |h| \leq k^e} \mathsf{Pr}\left[(q, a) \leftarrow \mathcal{G}(1^k, t) :\ \mathcal{V}(1^k, a, B_0(1^k, q, h)) = 1\right]$$

is negligible.

We will now examine the situation, where H_{c+1} runs with $\mathsf{A}_2, \mathsf{M}_2$; that is, we consider $\mathit{run}_{\mathsf{conf}_2^{c+1}}$. Let as above v_{H} and v_{A} denote the corresponding variables of M_2, with $v_{\mathsf{H}} = \perp$ or $v_{\mathsf{A}} = \perp$ if the corresponding variable is not set.

First note, that by definition of $\mathcal{D}$, it is

$$\mathsf{Pr}\left[t \leftarrow \mathcal{D}(1^k) :\ k^c \leq t \leq k^{c+1}\right] \geq \tfrac{1}{k} \tag{15}$$

for sufficiently large k.

By definition of c and (10), the following is overwhelming:

$$\begin{aligned}
&\min_{k^c \leq t \leq k^{c+1}} \mathsf{Pr}\left[(q, a) \leftarrow \mathcal{G}(1^k, t) :\ \mathcal{V}(1^k, a, B_0(1^k, q, 0)) \neq 1\right] \\
&\leq \mathsf{Pr}\left[t \leftarrow \mathcal{D}(1^k),\ (q, a) \leftarrow \mathcal{G}(1^k, t) :\ \mathcal{V}(1^k, a, B_0(1^k, q, 0)) \neq 1 \mid k^c \leq t \leq k^{c+1}\right] \\
&= \mathsf{Pr}\left[t \leftarrow \mathcal{D}(1^k),\ (q, a) \leftarrow \mathcal{G}(1^k, t) :\ \mathcal{V}(1^k, a, B_{c+1}(1^k, q, 0)) \neq 1 \mid k^c \leq t \leq k^{c+1}\right] \\
&\overset{(**)}{=} \mathsf{Pr}\left[v_{\mathsf{A}} \neq 1 \mid k^c \leq t \leq k^{c+1}\right].
\end{aligned} \tag{16}$$

Here $(**)$ is shown like $(*)$ in the first part of the proof.

Note further that by definition of H_d (in the particular case $d = c + 1$) and (11), and considering (15), we have that

$$\mathsf{Pr}\left[v_{\mathsf{H}} \neq 0 \mid k^c \leq t \leq k^{c+1}\right] \tag{17}$$

is overwhelming.

Combining (17) and (16), we conclude that

$$\mathsf{Pr}\left[v_{\mathsf{H}} \neq 0\ \wedge\ v_{\mathsf{A}} \neq 1 \mid k^c \leq t \leq k^{c+1}\right] \leq \mathsf{Pr}\left[b = 1\ \vee\ b = \perp \mid k^c \leq t \leq k^{c+1}\right]$$

are both overwhelming (consider that M_2's output is $b = 0$ only if v_{H} and v_{A} are defined and $\neg v_{\mathsf{H}} \vee v_{\mathsf{A}}$). Using (15), we finally have that

$$\mathsf{Pr}\left[b = 1\ \vee\ b = \perp\right]$$

is non-negligible. Since in the run of the real configuration conf_1^{c+1}, it is $b = 0$ with overwhelming probability, and b shows up in H_{c+1}'s view, this is a contradiction to (14) and shows $(\hat{M}_1, S) \not\geq_{\mathsf{sec}}^{\mathsf{uni,poly}} (\hat{M}_2, S)$. □

5 Conclusion

We have separated standard and universal simulatability in the case of computational and statistical security. This shows that these security notions are indeed different. However, it would be nice to know whether there is a less “artificial” separating example than ours. In particular, it is not clear whether there is a more “cryptographic” example.

We have also shown that for perfect security, standard and universal simulatability coincide. This result may ease security proofs—showing standard simulatability automatically shows universal simulatability.

Acknowledgements

We thank Ran Canetti, Jörn Müller-Quade, and Rainer Steinwandt for interesting and valuable discussions. Furthermore, we thank the anonymous referees for helpful comments.

References

[Bac04] Michael Backes. E-mail communication with the authors, June 2004.

[BPW04a] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In Moni Naor, editor, *Theory of Cryptography, Proceedings of TCC 2004*, number 2951 in Lecture Notes in Computer Science, pages 336–354. Springer-Verlag, 2004. Online available at `http://www.zurich.ibm.com/security/publications/2004/BaPfWa2004MoreGeneralComposition.pdf`.

[BPW04b] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Secure asynchronous reactive systems. IACR ePrint Archive, March 2004. Online available at `http://eprint.iacr.org/2004/082.ps`.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000. Full version online available at `http://eprint.iacr.org/1998/018.ps`.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Full version online available at `http://eprint.iacr.org/2000/067.ps`.

[Can04] Ran Canetti. Personal communication with one of the authors at TCC, February 2004.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, pages 494–503. ACM Press, 2002. Extended abstract, full version online available at `http://eprint.iacr.org/2002/140.ps`.

[Lin03] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *44th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2003*, pages 394–403. IEEE Computer Society, 2003. Online available at `http://www.research.ibm.com/people/l/lindell/PAPERS/gc-uc.ps.gz`.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *7th ACM Conference on Computer and Communications Security, Proceedings of CCS 2000*, pages 245–254. ACM Press, 2000. Extended version online available at `http://www.semper.org/sirene/publ/PfWa_00CompInt.ps.gz`.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy, Proceedings of SSP 2001*, pages 184–200. IEEE Computer Society, 2001. Full version online available at `http://eprint.iacr.org/2000/066.ps`.

[RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, Massachusetts Institute of Technology, February 1996. Online available at `http://theory.lcs.mit.edu/~rivest/RivestShamirWagner-timelock.ps`.

A Review of Reactive Simulatability

In this section, we present the notion of reactive simulatability. This introduction only very roughly sketches the definitions, and the reader is encouraged to read [BPW04b] for more detailed information and formal definitions.

Reactive Simulatability is a definition of security which defines a protocol $\hat{M}_1$ (the *real protocol*) to be *as secure as* another protocol $\hat{M}_2$ (the *ideal protocol*, the *trusted host*), if for any adversary A_1 (also called the *real adversary*), and any *honest user* H, there is a *simulator* A_2 (also called the *ideal adversary*), s.t. the view of H is indistinguishable in the following two scenarios:

- The honest user H runs together with the real adversary A_1 and the real protocol $\hat{M}_1$
- The honest user H runs together with the simulator A_2 and the ideal protocol $\hat{M}_2$.

Note that there is a security parameter k common to all machines, so that the notion of indistinguishability makes sense.

This definition allows to specify some trusted host—which is defined to be a secure implementation of some cryptographic task—as the ideal protocol, and then to consider the question, whether a real protocol is as secure as the trusted host (and thus also a secure implementation of that task). In order to understand the above definitions in more detail, we have to specify what is meant by machines "running together". Consider a set of machines (called a *collection*). Each machine has so-called *simple in-ports* (written $\mathsf{p}?$), *simple out-ports* (written $\mathsf{p}!$), and *clock out-ports* (written $\mathsf{p}^{\triangleleft}!$). Ports with the same name (p in our example) are considered to belong together and are associated with a *buffer* $\tilde{\mathsf{p}}$. These are then interconnected as in Figure 5 (note that some or all ports may originate from the same machine). Now, when a collection runs, the following happens: At every time, exactly one machine is activated. It may now read its simple in-ports (representing incoming network connections), do some work, and

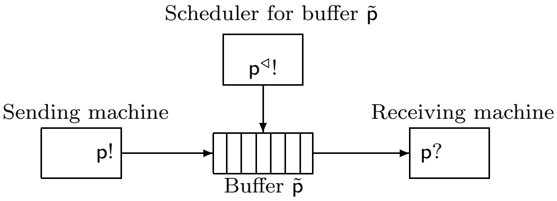
Scheduler for buffer $\tilde{\mathsf{p}}$
$\mathsf{p}^{\triangleleft}!$
Sending machine
$\mathsf{p}!$
Receiving machine
$\mathsf{p}?$
Buffer $\tilde{\mathsf{p}}$

Fig. 5. A connection

then write output to its simple out-ports. After such an activation the contents of the simple out-ports $\mathsf{p}!$ are appended to the queue of messages stored in the associated buffer $\tilde{\mathsf{p}}$. However, since now all messages are stored in buffers and will not be delivered by themselves, machines additionally have after each activation the possibility to write a number $n \geq 1$ to at most one clock out-port $\mathsf{p}^{\triangleleft}!$. Then the n-th undelivered message of buffer $\tilde{\mathsf{p}}$ will be written to the simple in-port $\mathsf{p}?$ and deleted from the buffer's queue. The machine that has the simple in-port $\mathsf{p}?$ will be activated next. So the clock out-ports control the scheduling. Usually, a connection is clocked by (i.e., the corresponding clock out-port is part of) the sender, or by the adversary. Since the most important use of a clock out-port is to write a 1 onto it (deliver the oldest message in the buffer), we say a machine clocks a connection or a message when a machine writes a 1 onto the clock port of that connection.

At the start of a run, or when no machine is activated at some point, a designated machine called the *master scheduler* is activated For this, the master scheduler has a special port, called the *master clock port* $\mathsf{clk}^{\triangleleft}?$.

Note that not all collections can be executed, only so-called *closed* collections, where all connections have their simple in-, simple out-, and clock out-port. If a collection is not closed, we call the ports having no counterpart *free ports*.

In order to understand how this idea of networks relates to the above sketch of reactive simulatability, one has to get an idea of what is meant by a protocol. A protocol is represented by a so-called *structure* $(\hat{M}, S)$, consisting of a collection $\hat{M}$ of the protocol participants (parties, trusted hosts, etc.), and a subset of the free ports of $\hat{M}$, the so-called *service ports* S. The service ports represent the protocol's interface (the connections to the protocol's users). The honest user can then only connect to the service ports (and to the adversary), all other free ports of the protocol are intended for the communication with the adversary (they may e.g. represent side channels, possibilities of attack, etc.). Since usually a protocol does not explicitly communicate with an adversary, such free non-service ports are more commonly found with trusted hosts, explicitly modelling their imperfections.

With this information we can review the above "definition" of security. Namely, the honest user H, the adversary, and the simulator are nothing else but machines, and the protocols are structures. The view of H is then the restriction of the run (the transcripts of all states and in-/output of all machines during the protocols execution, also called trace) to the ports and state of H.

The definition, as presented so far, still has one drawback. We have not introduced the concept of a corruption. This can be accommodated by defining so-called systems. A *system* is a set of structures, where to each "corruption situation" (set of machines, which are corrupted) one structure corresponds. That is, when a machine is corrupted, it is not present anymore in the corresponding structure, and the adversary takes its place. For a trusted host, the corresponding system usually consists of structures for each corruption situation, too, where those connections of the trusted host, that are associated with a corrupted party, are under the control of the adversary.

We can now refine the definition of security as follows: A *real system* Sys_1 is as secure as an *ideal system* Sys_2, if every structure in Sys_1 is as secure as the corresponding structure in Sys_2.

A major advantage of a security definition by simulatability is the possibility of *composition*. The notion of composition can be sketched as follows: If we have on structure or system A (usually a protocol) implementing some other structure or system B (usually some primitive), and we have some protocol X^B (having B as a sub-protocol, i.e. using the primitive), then by replacing B by A in X^B, we get a protocol X^A which is as secure as X^B. This allows to modularly design protocols: first we design a protocol X^B, and then we find an implementation for B.

A.1 Glossary

In this section we explain the technical terms used in this paper. Longer and formal definitions can be found in [BPW04b].

$[\hat{C}]$: Completion of the collection $\hat{C}$. Results from adding all missing buffers to $\hat{C}$. **$\mathsf{Conf}_x(\hat{M}_2, S)$**: Set of ideal configurations that are possible for structure $(\hat{M}_2, S)$. **$\mathsf{Conf}_x^{\hat{M}_2}(\hat{M}_1, S)$**: Set of real configurations possible for structure $(\hat{M}_1, S)$. **$\mathsf{ports}(M)$**: The set of all ports, a machine or collection M has. **to clock**: To write 1 onto a clock out-port. ***EXPSMALL***: The set of exponentially small functions. ***NEGL***: The set of negligible functions (asymptotically smaller than the inverse of any polynomial). **buffer**: Stores message sent from a simple out- to a simple in-port. Needs an input from a clock port to deliver. **clock out-port $\mathsf{p}^{\triangleleft}!$**: A port used to schedule connection. **closed collection**: A collection is closed, if all ports have all their necessary counterparts. **collection**: A set of machines. **combination**: The combination of a set of machines is a new machine simulating the other machines. A set of machines can be replaced by its combination without changing the view of any machine. **composition**: Replacing sub-protocols by other sub-protocols. **computational security**: When in the security definition, honest user and adversary are restricted to machines running in polynomial time, and the views are computationally indistinguishable. **configuration**: A structure together with an honest user and an adversary. **free ports**: The free ports of a collection are those missing their counterpart. **honest user**: Represents the setting in which the protocol runs. Also called environment. **intended**

structure: A structure from which a system is derived making a structure for every corruption situation. **master clock port $\mathsf{clk}^{\triangleleft}?$**: A special port by which the master scheduler is activated. **master scheduler**: The machine that gets activated when no machine would get activated. **perfect security**: When in the security definition, the real and ideal run have to be identical, not only indistinguishable. Further the machines are completely unrestricted.[11] **run**: The transcript of everything that happens while a collection is run. Formally a random variable over sequences. $run_{\mathsf{conf},k,l}$ is the random variable of the run when running the configuration conf upon security parameter k, restricted to its first l elements. If k is omitted, a family of random variables is meant. If l is omitted, we mean the full run. **service ports**: The ports of a structure to which the honest user may connect. They represent the interface of the protocol. As service ports are most often ports of a buffer, they are sometimes specified through the set S^c of their complementary ports; S^c consists of all ports which directly connect to a service port. **simple in-port $\mathsf{p}?$**: A port of a machine, where it can receive messages from other machines. **simple out-port $\mathsf{p}!$**: As simple in-port, but for sending. **statistical security**: When in the security definition the statistical distance of polynomial prefixes of the views have a statistical distance which lies in a set of small functions $SMALL$ (in the security parameter k). Usually $SMALL = NEGL$. Further the machines are completely unrestricted.[11] **structure**: A collection together with a set of service ports, represents a protocol. **view**: A subsequence of the run. The $view(M)$ of some collection or machine M consists of the run restricted to the ports and states of M. Possible indices are as with runs.

[11] In [BPW04b] a machine can in every activation for a given input and current state only reach one of a finite number of states (this convention has been chosen for simplicity [Bac04]). However, this cannot even model the simple Turing machine that tosses (within one activation) coins until a 1 appears, and then stores the number of coin tosses. Therefore we will here adopt the convention that each state can have a countable number of potential successor states, from which one is chosen following some distribution depending on the input and the current state.