A Cryptographic Model for Branching Time Security Properties – the Case of Contract Signing Protocols*

Véronique Cortier[1], Ralf Küsters[2], and Bogdan Warinschi[3]

[1] CNRS, Loria, veronique.cortier@loria.fr
[2] ETH Zurich, ralf.kuesters@inf.ethz.ch
[3] University of Bristol, bogdan@cs.bris.ac.uk

Abstract. Some cryptographic tasks, such as contract signing and other related tasks, need to ensure complex, branching time security properties. When defining such properties one needs to deal with subtle problems regarding the scheduling of non-deterministic decisions, the delivery of messages sent on resilient (non-adversarially controlled) channels, fair executions (executions where no party, both honest and dishonest, is unreasonably precluded to perform its actions), and defining strategies of adversaries against all possible non-deterministic choices of parties and arbitrary delivery of messages via resilient channels. These problems are typically not addressed in cryptographic models and these models therefore do not suffice to formalize branching time properties, such as those required of contract signing protocols.
In this paper, we develop a cryptographic model that deals with all of the above problems. One central feature of our model is a general definition of fair scheduling which not only formalizes fair scheduling of resilient channels but also fair scheduling of actions of honest and dishonest principals. Based on this model and the notion of fair scheduling, we provide a definition of a prominent branching time property of contract signing protocols, namely balance, and give the first *cryptographic* proof that the Asokan-Shoup-Waidner two-party contract signing protocol is balanced.

1 Introduction

Cryptographic tasks, such as contract signing [1, 15, 8] and other related tasks, need to ensure complex, branching time properties, i.e., properties of the overall structure of the set of all possible executions of a protocol (as opposed to properties of single execution traces). Examples of such properties are balance [11]

* The first and third author were partly supported by ACI Jeunes Chercheurs JC9005 and ARA SSIA Formacrypt. The second author was supported by the SNF under Grant 200021-116596/1. The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

and abuse-freeness [15]. Defining such properties requires to cope with several challenges that are typically not addressed in cryptographic models. The main challenges include: modeling non-deterministic behavior of honest parties, resilient (non-adversarially controlled) channels, fair executions in which no party, honest or *dishonest*, can unreasonably be precluded to perform its actions, and strategies of adversaries to achieve certain goals against all possible behaviors of resilient channels and honest parties; the existence or absence of such strategies is a branching time property of a protocol, not a property of a single execution trace. Providing a computational model that deals with all such challenges and applying it to branching time properties of contract signing protocols is the main purpose of this paper.

We illustrate the above points via the balance property for (two-party) optimistic contract signing protocols as first defined by Chadha et al. [11] in a symbolic (Dolev-Yao based) model. These protocols can be used by two parties, A and B, to obtain each other's signature on a previously agreed contractual text with the help of a trusted third party (TTP), which, however, is only contacted in case of a problem. If and when the TTP is contacted depends on *non-deterministic decisions* of the parties. For example, A may decide to send an abort request to the TTP in case she doesn't want to wait any longer for a message from B, or suspects that B is dishonest. Contract signing protocols typically assume that A and B communicate with the TTP over *resilient (non-adversarially controlled) channels*: without such channels an adversary could block all messages from/to the TTP. Now, balance for an honest party A and a dishonest party B, as defined by Chadha et al., requires that in a protocol run it is not possible to reach a state where B has both i) a *strategy* to obtain a signed contract from A (no matter how A, the TTP, and the resilient channels behave) and ii) a (possibly different) *strategy* to prevent A from obtaining a signed contract from B (no matter how A, the TTP, and the resilient channels behave). Since, when following one of these strategies, the adversary, i.e., B, has to achieve his goal—obtaining a signed contract or preventing A from obtaining a signed contract—against the behavior of other entities that he cannot control or foresee (non-deterministic choices of A and delivery of messages on resilient channels), in a computational model it is necessary to determine the behavior of these entities by a *scheduler* which is *independent* of the adversary, and in fact, may work against the adversary. Moreover, for the balance property to make sense, the scheduler should not stop the run of a system if one of the entities in the system (A, the TTP, the resilient channels, the adversary) "can still take an action". In other words, the scheduling should be *fair* for all entities (both honest and dishonest). For example, if at some point A could still contact the TTP, then the scheduler should not stop the run of the system at this point but should eventually schedule A: contacting the TTP might enable A to get the contract. Stopping the system before scheduling A would be unfair and unrealistic since no one stops A from contacting the TTP in a real protocol run. Note that a scheduler is just an imaginary entity that is only needed to *model* how things are potentially scheduled in a real protocol run. Conversely, if B (the

adversary) wants to send a message to the TTP, the scheduler should not stop the run of the system but eventually schedule B: sending a message to the TTP might enable B to obtain a signed contract which he otherwise might not be able to get. Again, stopping the system before scheduling B would be unfair and unrealistic since no one stops B from contacting the TTP in a real protocol run. Note that B is an arbitrary adversary (machine), and hence, a general notion of fair scheduling is needed to capture whether "B can still take an action" (e.g., send a message).

Clearly, standard cryptographic models, in which only one adversary is considered controlling the complete communication network, and honest principals can not make non-deterministic choices are insufficient for dealing with the class of protocols and properties considered here. Some cryptographic models take some (not all) of the above aspects into account, but with a different focus and in a way not suitable for the classes of protocols and properties we consider (see the related work).

Contribution of this paper. In this paper, we propose a computational model that deals with the challenges mentioned above and allows to specify complex, branching time properties.

More precisely, our model is based on a general computational model for systems of interactive Turing machines (ITMs). The model is presented in Section 3. Based on this model, we define a security-specific model (see Section 4) where we use ITMs to capture the behavior of the honest principals, the adversary, the network and resilient channels, and the scheduler. The purpose of the scheduler is to resolve non-deterministic behavior of honest principals, to schedule the resilient channels, and to trigger the adversary. As explained above, modeling the scheduler as an entity independent of the adversary is important. The adversary and the scheduler are each equipped with what we call a *view oracle* which can be invoked by these entities to obtain a *view* on the history of the run of the protocol so far, and hence, to adapt their actions accordingly; typically, the adversary and the scheduler have different view oracles, and hence, different views on the history. The view of the adversary typically includes all messages on the network channels and only messages on those resilient channels which are not required to be read-protected. Conversely, the scheduler might have complete information about the resilient channels. The exact definition of the views (view oracles) depends on the security properties considered and can be adapted depending on the strength of the security guarantee desired. The ITMs that we use cannot be exhausted and can respond to an unbounded number of requests, as for example needed when modeling the TTP in contract signing protocols. Also, this, for example, ensures that the scheduler cannot exhaust the adversary or honest parties, which otherwise would lead to unrealistic runs (recall that the scheduler is only an imaginary entity that is used to model reality).

As mentioned, fair scheduling is an important ingredient in the definition of many security properties, and it is non-trivial to define in computational, resource-bounded settings. We provide a general definition of when a scheduler is fair for a system of ITMs (see Section 5). We emphasize that our definition

is independent of the specific structure of the system or the specific ITMs used in the system. This is important as we need to capture fair scheduling also for arbitrary dishonest parties, i.e., adversary machines. Intuitively, we call a scheduler fair for a system if it does not stop the run of the system at a point where at least one of the other machines in the system, e.g., honest parties, the adversary, resilient channels, "can still take an action", e.g., an honest principal could (non-deterministically) decide to start an abort protocol, a resilient channel could deliver a message, or the adversary is ready to send a message to an honest principal. We formalize that a machine "can still take an action" in a general way as follows: We say that a machine can take an action if the machine can be activated by the scheduler with some input so that at the end of the activation the machine has changed its local configuration, and hence, performed some action. (We note that according to our definition of ITMs, if an ITM outputs a message, then it changes its local configuration.) The above definition in particular applies to adversary machines and also to honest parties and resilient channels. For example, if at some point A in a contract signing protocol could either wait for a message from B or contact the TTP to run the abort protocol and the scheduler schedules A to run the abort protocol with TTP, then A changes its local configuration, e.g., goes from state q_{wait} to state q_{abort}. While there does not exist a fair scheduler for every system, we identify sufficient, reasonable conditions for a system to have a fair scheduler. The way fair scheduling is defined here appears to be new and is of interest independent of its application to branching time properties (see also the related work).

Based on our computational model and the notion of fair schedulers, we provide a definition for balance of (contract signing) protocols (see Section 6). In this definition, we need to quantify (universally and existentially) over two different schedulers. The first scheduler may be unfair and may collude with the adversary in order to reach a certain point in the protocol run. The second one has to be fair, but tries to prevent the adversary from achieving his goal. As a proof of concept, we apply our definition to the ASW two-party optimistic contract-signing protocol [1], which is presented in Section 2, and show it to be balanced when implemented with primitives that satisfy standard security assumptions (see Section 6.2). Our proof of balance of this protocol is the first computational proof of this (now rigorously defined) property for a contract signing protocol.

We point the reader to the long version of our paper [13] for further details.

Related work. Rigorous models, security definitions as well as analysis methods and tools for branching time properties of contract signing protocols have been proposed in [11, 22, 21, 25, 22, 5, 12, 19, 20]. However, all these works are based on the symbolic (Dolev-Yao) model and do not consider the more involved computational case.

Backes et al. [7] (see also [6]) proposed a definition of fair scheduling in a computational model. While they only consider fair scheduling of channels (which is insufficient for branching time properties), our definition is concerned with fair scheduling of arbitrary machines. Other works that use some kind of

fairness in specific settings are [3] and [16]. None of the mentioned works, [7, 6, 3, 16], studies branching time properties or properties of contract signing protocols.

Asokan, Shoup, and Waidner [2] propose a fair contract signing protocol and present a computational model to study fairness of their protocol. However, the model and the notion of fair scheduling that they use is tailored to their specific setting and does not apply to branching time properties.

Canetti et al. [10] study a computational model based on probabilistic I/O automata (PIOAs) in which non-deterministic behavior of principals can be modeled. However, they focus on simulation-based security and do not study fairness issues or branching time properties.

2 A Running Example: The ASW Protocol

In this section, we provide an informal description of the ASW protocol [1]. This protocol is our running example which we use throughout the paper to provide intuition for the models and the notions that we introduce. A more formal description in terms of the model that we propose in this paper can be found in [13].

Cryptographic primitives. The ASW protocol uses concatenation, signatures and hashing. We denote the concatenation of bit strings $m_1, \ldots, m_n$ by $\langle m_1, \ldots, m_n \rangle$, and sometimes by $m_1, \ldots, m_n$. We assume that every m_i can uniquely be recovered from the concatenation. Verification and signing keys of principal P are denoted by v_P and s_P, respectively. The signature of m generated using s_P is denoted by $\mathsf{sig}_{v_P}(m)$. We require for the associated signature verification algorithm $\mathsf{sigver}(\cdot, \cdot, \cdot)$ that $\mathsf{sigver}(m, s, v_P) = \mathsf{true}$ if s is a signature on m generated using s_P, and that $\mathsf{sigver}(m, s, v_P) = \mathsf{false}$ otherwise. We write $\mathsf{sig}[m, v_P]$ for $\langle m, \mathsf{sig}_{v_P}(m) \rangle$, and write $h(m)$ for the hash of message m.

Protocol description. The ASW protocol enables two principals A (the originator) and B (the responder) to obtain each other's signature on a previously agreed contractual text text (a fixed bit string) with the help of a trusted third party (TTP) T, which however is only invoked in case of problems. In other words, the ASW protocol is an optimistic two-party contract-signing protocol.

There are two kinds of valid contracts: the standard contract, which is of the form $\langle \mathsf{sig}[m_A, v_A], N_A, \mathsf{sig}[m_B, v_B], N_B \rangle$, and the replacement contract, which is of the form $\mathsf{sig}[\langle \mathsf{sig}[m_A, v_A], \mathsf{sig}[m_B, v_B] \rangle, v_T]$, where N_A and N_B are nonces, generated by A and B, respectively, $m_A = \langle v_A, v_B, v_T, \mathsf{text}, h(N_A) \rangle$ and $m_B = \langle \mathsf{sig}[m_A, v_A], h(N_B) \rangle$.

The ASW protocol consists of three subprotocols: the exchange, abort, and resolve protocol. These subprotocols are explained next.

Exchange protocol. First A sends the message $\mathsf{sig}[m_A, v_A]$ indicating A's interest to sign the contract. By sending this message, A "commits" to signing the contract. Then, similarly, B indicates his interest to sign the contract, by replying with $\mathsf{sig}[m_B, v_B]$. Finally, first A and then B reveal N_A and N_B, respectively.

Abort protocol. If, after A has sent her first message, B does not respond, A may contact T to abort, i.e., A runs the abort protocol with T. Note that A may wait as long as she wants before contacting T (non-deterministic action). In the abort protocol, A first sends $a_A = \mathsf{sig}[\langle \mathsf{aborted}, \mathsf{sig}[m_A, v_A]\rangle, v_A]$. If T has not received a resolve request before (see below), then T sends back to A the *abort token* $a_T = \mathsf{sig}[\langle \mathsf{aborted}, a_A\rangle, v_T]$. Otherwise (if T received a resolve request, which in particular involves the messages $\mathsf{sig}[m_A, v_A]$ and $\mathsf{sig}[m_B, v_B]$ from above), it sends the *replacement contract* $r_T = \mathsf{sig}[r, v_T]$ to A, where $r = \langle \mathsf{sig}[m_A, v_A], \mathsf{sig}[m_B, v_B]\rangle$.

Resolve protocol. If, after A has sent the nonce N_A, B does not respond, A may contact T to resolve, i.e., A runs the resolve protocol with T. Again, A may wait for as long as she wants before contacting T (non-deterministic action). In the resolve protocol, A sends the message r to T. If T has not sent out an abort token before, then T returns the replacement contract r_T, and otherwise T returns the abort token a_T. Analogously, if, after B has sent the nonce N_B, A does not respond, B may contact T to resolve, i.e., B runs the resolve protocol with T similarly to the case for A.

We note that the communication with T (for both A and B) is carried out over resilient channels. More specifically, these channels are authenticated, so the adversary can read their content but he is not entitled to modify, delete, or delay messages sent over these channels.

3 The General Computational Model

Our general computational model is defined in terms of systems of interactive Turing machines (ITMs) and is related to the models in [4, 9, 14, 17]. However, our exposition follows more closely that of [24].

ITMs. An *(inexhaustible) interactive Turing machine (ITM, for short) M* is a probabilistic Turing machine with the following tapes: a *security parameter tape* for storing the security parameter, a *random tape* for storing random coins, zero or more *input* and *output tapes*, and *work tapes*. The input and output tapes have names; different tapes have different names. These determine how ITMs are connected in a system of ITMs. If an ITM sends a message on an output tape named c, then only an ITM with an input tape named c can receive this message. An ITM M may use oracles, called *the oracles associated with the ITM*. If the oracles $\mathcal{O}_1, \ldots, \mathcal{O}_n$ are associated with M we sometimes write $M(\mathcal{O}_1, \ldots, \mathcal{O}_n)$ instead of M to emphasize this fact. The runtime of an ITM is polynomially bounded per activation (in the security parameter, the current input, and the size of the current configuration). This allows the ITM to "scan" the complete incoming message and its complete current configuration. Hence, an ITM can not be exhausted (therefore the name *inexhaustible* interactive Turing machine).

Systems of ITMs. A *system* $\mathcal{S}$ of ITMs is a parallel composition $M_1 \,||\, \cdots \,||\, M_n$ of ITMs M_i, $i = 1, \ldots, n$. These machines communicate over input and output tapes; at every time only one ITM is active and all other ITMs wait for new

input. A machine is activated when it receives input on one of its input tapes. If a machine does not activate another machine (by outputting a message), the so-called *master ITM* is triggered. We require w.l.o.g. that if a machine outputs a message, then its local configuration changes. Given $\mathcal{S} = M_1 \,||\, \cdots \,||\, M_n$, we write $\mathcal{S}(1^\eta, r_1, \ldots, r_n)$ for the system obtained from $\mathcal{S}$ by writing a security parameter η on the security parameter tapes and random coins $r_i \in \{0,1\}^*$ on the random tapes of the M_i. A run of $\mathcal{S}(1^\eta, r_1, \ldots, r_n)$ is defined to be a sequence of global configurations q where a *global configuration* q is a tuple $(q_1, \ldots, q_n)$ of the configurations q_i of the single machines M_i, for every $i = 1, \ldots, n$.

In general, a run of a system does not necessarily terminate. For example, if in $\mathcal{S} = M_1 \,||\, M_2$ the ITMs M_1 and M_2 are connected via enriching input tapes, then they can send message back and forth between each other forever.

We say that a system $\mathcal{S}$ is a *polynomial-time system* if there exists a probabilistic Turing machine which simulates runs of $\mathcal{S}$ and whose runtime is polynomially bounded in the security parameter with overwhelming probability. For polynomial-time systems, we denote by $\mathcal{S}(\eta)$ the random variable that returns runs of $\mathcal{S}$ with security parameter η where the coins for the ITMs in $\mathcal{S}$ are chosen uniformly at random. We write $\mathcal{S}(\eta) \rightsquigarrow q$ to say that the final global configuration in a run returned by $\mathcal{S}(\eta)$ is q. If q' is a global configuration for $\mathcal{S}(\eta)$, we write $\mathcal{S}_{q'}(\eta)$ to denote the distribution of runs obtained when the initial configuration of the ITMs in $\mathcal{S}$ are defined according to q' (with possibly random coins added on random tapes if needed). In case q' is drawn from a family $D = \{D_\eta\}_\eta$ of distributions, we write $\mathcal{S}_D(\eta)$ for the random variable that returns a run according to the following experiment: $q' \xleftarrow{R} D_\eta$, output $\mathcal{S}_{q'}(\eta)$. We define $\mathcal{S}_{q'}(\eta) \rightsquigarrow q$ and $\mathcal{S}_D(\eta) \rightsquigarrow q$ analogously to $\mathcal{S}(\eta) \rightsquigarrow q$. Here, and in the rest of the paper we only consider families of distributions D that are polynomially samplable, i.e., that are the output of a probabilistic polynomial-time Turing machine.

Given a system $\mathcal{S}$, we call an ITM $\mathcal{E}$ an *environment* for $\mathcal{S}$ if i) the runtime of $\mathcal{E}$ is polynomial in the security parameter alone (and independent of the length of the input that $\mathcal{E}$ receives) and ii) $\mathcal{E}$ is I/O-compatible with $\mathcal{S}$, i.e., $\mathcal{E}$ only writes to external input tapes of $\mathcal{S}$ and $\mathcal{E}$ only reads from external output tapes of $\mathcal{S}$. Adopting terminology from [18], we call $\mathcal{S}$ *reactively polynomial* if $\mathcal{S} \,||\, \mathcal{E}$ is a polynomial-time system for every environment $\mathcal{E}$ of $\mathcal{S}$ where $\mathcal{E}$ does not have an associated oracle.

4 The Security-specific Model

Based on the general computational model introduced above, we define below the security-specific model. In this model, we consider specific systems of ITMs, called protocol systems. These systems consist of protocol machines, which determine the actions of honest principals, an adversary machine, a scheduler, and buffers for network and resilient channels. The adversary does not have complete control over the communication. Specifically, while we let the adversary control the network, he does not control resilient channels, i.e., the adversary

can not modify, delete, or delay messages sent on this channel. (We often allow the adversary to read messages sent on resilient channels, though.) The purpose of the scheduler is to schedule messages sent over resilient channels, i.e., the scheduler decides when and which messages written on the resilient channel are delivered. Also, the scheduler resolves non-deterministic choices made by honest principals, e.g., whether to wait for a message of another party or to abort the protocol. Furthermore, the scheduler determines when the adversary is activated. In particular, the adversary is not necessarily scheduled as soon as an honest principal outputs a message. Instead some message sent on a resilient channel or an honest principal that needs to make a non-deterministic decision might be scheduled first (by the scheduler). However, if the adversary sends a message to an honest principal this principal is activated right away. Allowing the scheduler to first schedule other entities (honest principals or resilient channels) would significantly weaken the power of the adversary.

PROTOCOLS. A *protocol* Π is defined by a tuple $(\mathcal{H}, \mathcal{D}, \{\mathbf{H}_i\}_{i\in\mathcal{H}})$ where $\mathcal{H}$ and $\mathcal{D}$ are finite disjoint sets of names of *honest* and *dishonest principals*, respectively, and $\{\mathbf{H}_i\}_{i\in\mathcal{H}}$ is a family of ITMs, called protocol machines (see below), which specify honest principals; dishonest principals will be simulated by the adversary. We define $\mathcal{P} = \mathcal{H} \cup \mathcal{D}$ to be the set of all principals. We note that $\mathbf{H}_i$ may specify the actions of principal i in one session of a specific protocol, e.g., it specifies one session of the initiator of the ASW protocol, or multiple sessions of i in possibly different roles.

PROTOCOL SYSTEMS. A system induced by Π consists of the protocol machines of Π, an adversary machine $\mathbf{A}$, a scheduler machine $\mathbf{S}$, and buffer machines for the network and resilient channels. More precisely, a *(protocol) system $\mathcal{S}$ for Π* is of the form

$$\mathcal{S} = (\,||_{\,i\in\mathcal{H}}\mathbf{H}_i) \;||\; (\,||_{\,i\in\mathcal{H},j\in\mathcal{P}}\mathbf{Net}_j^i)\,||\; (\,||_{\,i\in\mathcal{H},j\in\mathcal{P}}\mathbf{RC}_j^i) \;||\; \mathbf{A} \;||\; \mathbf{S}$$

where $\mathbf{H}_i$, $i \in \mathcal{H}$, is a protocol machine of Π modeling an honest principal, $\mathbf{Net}_j^i$, $i \in \mathcal{H}$, $j \in \mathcal{P}$ is a network buffer (machine) on which i sends messages over the network intended for j, $\mathbf{RC}_j^i$, $i \in \mathcal{H}$, $j \in \mathcal{P}$ is a resilient channel buffer (machine) on which i sends messages intended for j, $\mathbf{A}$ is the adversary (machine), and $\mathbf{S}$ the scheduler (machine). We call $\mathcal{S}$ the *system induced by Π, $\boldsymbol{A}$, and $\boldsymbol{S}$* and denote it by $\mathcal{S}(\Pi, \mathbf{A}, \mathbf{S})$. We refer to the system $\mathcal{S}$ with $\mathbf{A}$ and $\mathbf{S}$ removed by $\mathcal{S}(\Pi)$. Analogously, we refer to the system $\mathcal{S}$ with $\mathbf{S}$ removed by $\mathcal{S}(\Pi, \mathbf{A})$. We now explain informally how the machines of $\mathcal{S}(\Pi, \mathbf{A}, \mathbf{S})$ work and how they are connected via tapes (see [13] for details).

A network buffer machine $\mathbf{Net}_j^i$ receives messages from $\mathbf{H}_i$ and stores them. The adversary has typically access to these messages.

A resilient channel buffer machine $\mathbf{RC}_j^i$ stores messages and interacts with $\mathbf{H}_i$ just as $\mathbf{Net}_j^i$. In addition, $\mathbf{RC}_j^i$ is scheduled by the scheduler who determines which messages are delivered. The adversary may or may not have access to $\mathbf{RC}_j^i$. This depends on whether or not $\mathbf{RC}_j^i$ should be read-protected.

A protocol machine $\mathbf{H}_i$ may send messages to the network buffers $\mathbf{Net}_j^i$ and the resilient channel buffers $\mathbf{RC}_j^i$ for every $j \in \mathcal{P}$ as explained above. If $\mathbf{H}_i$ does not produce output, the scheduler $\mathbf{S}$ is activated. A protocol machine $\mathbf{H}_i$ can be activated by messages from the network (the adversary), the resilient channels (these messages are guaranteed to be authentic), and the scheduler. Messages from the scheduler are meant to resolve non-deterministic choices made by $\mathbf{H}_i$ (these messages are assumed to come from a fixed, finite set of messages).

The adversary machine $\mathbf{A}$ is associated with an oracle, called the *view oracle*, which can be invoked by $\mathbf{A}$ to obtain a *view* on the history of the run of the overall system so far. The view usually does not contain full information about the history. It is typically restricted to the content of the network buffers so far and the content of (some) of the resilient channel buffers, depending on whether these channels are supposed to be read-protected. In addition to invoking the view oracle, $\mathbf{A}$ can send messages to honest principals either via network (unauthenticated) or resilient channel connections (authenticated). A message sent by the adversary on one of these channels is delivered directly. The adversary machine $\mathbf{A}$ can only be activated by the scheduler. We only allow adversary machines for which the system $\mathcal{S}(\Pi, \mathbf{A})$ is reactively polynomial.

The scheduler $\mathbf{S}$ is also associated with a *view oracle* which provides $\mathbf{S}$ with a *view* on the history of the run of the overall system so far. Typically this view will be different from the view of the adversary and depending on the security property may contain full information about the history, no information at all, or something in between. As explained above, the purpose of $\mathbf{S}$ is to resolve non-deterministic choices of honest principals ($\mathbf{H}_i$), to schedule messages on resilient channels, and to determine when the adversary $\mathbf{A}$ is triggered. For this purpose, $\mathbf{S}$ sends appropriate messages to these machines.

5 Fair Schedulers

Intuitively, we define a scheduler to be fair if it does not stop the run of a system when at least one of the (other) machines in the system can still take an action, e.g., an honest principal could start an abort protocol, a resilient channel could deliver a message, or the adversary is ready to output a message to an honest principal. As already explained in the introduction, fair scheduling is important in the definition of many security properties, such as fairness and balance for contract signing protocols.

The problem of defining fair schedulers is to make precise what it means that a machine "can still take an action". Notice that we need a general definition that works for arbitrary machines (honest principal machines, resilient channel machines, *and adversary machines*) not only for specific machines, such as specific buffers as in [7, 6].

Roughly speaking, we say that a machine "can still take an action" if the machine can be activated by the scheduler with some input so that at the end of the activation the machine has changed its local configuration, i.e., scheduling the machine causes it to make some progress or to perform some action. (Recall

from Section 3 that if an ITM sends out a message, then it changes its local configuration.) For example, if an adversary machine wants to send a message to an honest principal, then when it is triggered by the scheduler it would send the message and change its local configuration. Hence, a fair scheduler has to eventually trigger the adversary as the adversary "can still take an action" in the above sense. Similarly, a fair scheduler has to eventually trigger a protocol machine that does not receive a message from the network but has the option of contacting the TTP, as contacting the TTP causes the protocol machine to change its local configuration.

We note that a scheduler does not necessarily know when a machine, including the adversary, "can still take an action" in the sense just explained. Hence, it might schedule such a machine even though this machine does not want to take an action. However, a machine can always read the message received from the scheduler (possibly even query the view oracle in case of the adversary) and, in case it does not want to take an action, it can return to its old local configuration. Note that here we use that ITMs cannot be exhausted. In case of exhaustible ITMs unrealistic runs would occur.

The above discussion motivates the following definition of fair schedulers. Roughly speaking, the definition below says that if the run of a system stops, then even if in the system the old scheduler is replaced by a new one (even one with full information on the history of the run), the new scheduler cannot continue the run of the system (at least not with non-negligible probability) such that one of the ITMs in the system changes its local configuration. In other words, a fair scheduler may only stop the run of a system if no ITM in the system (other than the scheduler itself) can or wants to take a further action, i.e., no other scheduler can cause an ITM to change its local configuration. We state this definition for general systems rather than only for protocol systems (Section 4). In this definition, we use what we call a full-information oracle. Called at some point in a run of a system, a *full-information oracle* returns the whole history of the run so far for all machines involved including the random coins used so far by the ITMs. We state the definition for the case that the initial global configuration comes from a family $D = \{D_\eta\}_\eta$ of distributions. This is useful for modeling, for example, an initialization phase.

Definition 1. *Let Q be a reactively polynomial system which does not contain a master ITM. An ITM $\boldsymbol{S}$ is a* fair scheduler *for Q and a family $D = \{D_\eta\}_\eta$ of distributions on (initial) global configurations if it is an environment for Q and if for every environment $\boldsymbol{S}'$ for Q which has access to a full-information oracle the probability that the following experiment returns 1 is negligible in the security parameter η:*

Exp$(\eta, \boldsymbol{S}, \boldsymbol{S}')$*:*

Run Q with $\boldsymbol{S}$, i.e.: $\mathcal{S}_D(\eta) \rightsquigarrow q'$ with $\mathcal{S} = Q \,||\, \boldsymbol{S}$
Continue the run with $\boldsymbol{S}'$ instead of $\boldsymbol{S}$, i.e.: $\mathcal{S}'_{q''}(\eta) \rightsquigarrow q'''$ with $\mathcal{S}' = Q \,||\, \boldsymbol{S}'$ and q'' is obtained from q' by replacing the configuration of $\boldsymbol{S}$ by the initial configuration of $\boldsymbol{S}'$ and writing the history of the run so far on one of the work tapes of $\boldsymbol{S}'$.

If there exists an ITM M in Q such that the local configuration of M in q' is different from the corresponding local configuration in q''', then output 1*, and otherwise, output* 0*.*

Applied to protocol systems (Section 4), a fair scheduler may only stop if i) the resilient channel buffers are empty, ii) triggering a protocol machine with any message (among the finite set of possible once, e.g., `abort`) does not change the local configuration of this machine, and iii) triggering the adversary machine with the message `schedule` does not change the local configuration of this machine (which means that the adversary does not want to take a step anymore).

Since ITMs cannot be exhausted they might change their local configuration whenever they are invoked. Hence, a fair scheduler would never be allowed to stop. Thus, we observe:

Observation 1 *There are systems for which no fair scheduler exists.*

Systems with fair schedulers. We now identify some reasonable restrictions on protocols and adversaries as to ensure the existence of a fair scheduler. Due to space limitations, we only provide informal definitions. First, we put a restriction on the adversary.

Definition 2. *(informal) An adversary machine for a protocol Π, view oracles $\mathcal{O}_{adv}$, $\mathcal{O}_{sch}$, and a family of distributions $D = \{D_\eta\}_\eta$ on (initial) global configurations D is* fairness-enabling *if the number of configuration changes of the adversary in every run of the system $\mathcal{S} = \mathcal{S}(\Pi, \boldsymbol{A}(\mathcal{O}_{adv}), \boldsymbol{S}(\mathcal{O}_{sch}))$ (and hence, the number of actions, such as sending messages, the adversary can perform) can be bounded by a polynomial which is independent of the scheduler $\boldsymbol{S}(\mathcal{O}_{sch})$.*

The immediate analog to the definition above for protocol machines would be too restrictive since the number of configuration changes of a protocol machine might depend on the number of interactions with the adversary, and hence, depends on the adversary. For example, if a TTP is modeled in such a way that it reacts to all requests (which could come from the adversary), then the number of configuration changes of the TTP depends on the adversary. This motivates the following definition.

Definition 3. *(informal) Given oracles $\mathcal{O}_{adv}$ and $\mathcal{O}_{sch}$, and a family of distributions $D = \{D_\eta\}_\eta$ on (initial) global configurations, a protocol Π is* fairness-enabling *if the number of configuration changes of protocol machines in Π in every run of the system $\mathcal{S} = \mathcal{S}(\Pi, \boldsymbol{A}(\mathcal{O}_{adv}), \boldsymbol{S}(\mathcal{O}_{sch}))$ can be bounded by a polynomial which may depend on $\boldsymbol{A}(\mathcal{O}_{adv})$ but not on $\boldsymbol{S}(\mathcal{O}_{sch})$.*

The following theorem states that for every fairness-enabling protocol and every fairness-enabling adversary, there exists a fair scheduler (even without access to a view oracle). Hence, for systems built from fairness-enabling protocols and adversaries, fair scheduling is possible. In the rest of the paper, we concentrate on such systems, which seem to capture all realistic cases. In order to state and

prove the theorem, we first need to be more precise about the view oracle of adversaries.

A view oracle is called an *adversary view oracle* if it is a deterministic polynomial-time algorithm which when invoked in a run of a protocol system gets as input the history of the run so far, except for the history of the scheduler, i.e., the history of the configurations (including the random coins used so far) of all machines in the system, except for the history of the configurations of the scheduler. We require that if the configurations of the ITMs, other than the scheduler, in a run of the protocol system have not changed from one point in the run to the next step in the run, then the adversary view oracle returns the same view as before. Note that even if the adversary view oracle obtains as input the full history of the system (excluding the scheduler) it typically will only return a restricted view on that history to the adversary.

Theorem 2. *For every fairness-enabling protocol Π, view oracle $\mathcal{O}_{sch}$, adversary view oracle $\mathcal{O}_{adv}$, polynomially samplable family of distributions $D = \{D_\eta\}_\eta$ on (initial) global configurations, and fairness-enabling adversaries $\boldsymbol{A}=\boldsymbol{A}(\mathcal{O}_{adv})$, there exists a scheduler $\boldsymbol{S}$ (even one without access to a view oracle) that is fair for $\mathcal{S}(\Pi, \boldsymbol{A})$ and D.*

6 Balanced Protocols and Results for the ASW Protocol

In this section, we define the notion of balance and show that the ASW protocol is balanced. The definition makes use of the previously introduced concept of fair scheduling.

6.1 Definition of Balance

The notion of balance for (two-party) contract-signing protocols was first introduced by Chadha et al. [11] in the symbolic (Dolev-Yao) setting. In a nutshell, their definition says that a protocol is balanced for an honest signer, say A, if no "unbalanced" state can be reached in a run of the contract-signing protocol where a run involves A, the Dolev-Yao intruder playing the role of the dishonest signer B, the TTP, the network and resilient channels. A state is *unbalanced* (for A) if in this state B has both i) a strategy to obtain a signature on the contract from A and ii) a (possibly different) strategy to prevent A from obtaining a signature on the contract from B. In other words, B can unilaterally determine the outcome of the protocol, which puts him in an advantageous position, for example, when making a deal with another party. In the first phase—*reaching* an (unbalanced) state— the non-deterministic choices made by honest principals and the way messages on resilient channels are scheduled might help B to reach the (unbalanced) state. However, in the second phase, B needs to have the mentioned strategies to achieve the two goals—obtaining a valid contract and preventing A from obtaining a valid contract—, and these strategies have to work no matter what non-deterministic choices the honest principals make and no matter how messages on resilient channels are scheduled.

Now, we introduce a computational analogue of the notion that we sketched above. We measure the success probability of an adversary that tries to undermine the balancedness of the protocol via an experiment which works in two phases (see below for a formal definition): In the first phase, the protocol runs along with the adversary $\mathbf{A}$ and a scheduler $\mathbf{S}$ which may resolve non-deterministic choices of honest principals and schedule messages on resilient channels and the adversary in a way that helps $\mathbf{A}$. At the end of this phase, a state (global configuration), say q, is reached. Now, one of the two goals (having the contract or preventing the other party from getting one) is picked (by some function `challenge`) and the adversary is asked to reach the chosen goal, starting from q but now running with a different scheduler which will try to resolve non-deterministic choices of honest principals and schedule resilient channels and the adversary in a way that is disadvantageous for $\mathbf{A}$. Intuitively, for balanced protocols, from any state q that is reached, at least for one of the two goals the probability that the adversary can reach this goal should be low.

In the following definition, we require that the scheduler used in the second phase of the experiment is fair in order to ensure that protocol runs are in fact completed both by honest parties *and* the adversary. This is crucial for two reasons: On the one hand, the adversary might otherwise be prevented from taking further actions, but these actions may be necessary for the adversary to achieve the required goal. Hence, the scheduling would be unfair for the adversary. And in fact, it would be unrealistic since in real protocol runs no one stops the adversary from taking further actions. On the other hand, honest principals might otherwise be prevented from taking counter-measures to the misbehavior of the adversary. Hence, the scheduling would be unfair (and again unrealistic) for the honest parties. Note that achieving fair scheduling for both honest parties and the adversary is guaranteed by our definition of fair scheduling (Section 5). However, a notion only based on fair message delivery [7, 6] would be insufficient.

In order to ensure that, in the second phase, fair scheduling is possible, we split the adversary in two parts, $\mathbf{A}$ (for the first phase) and $\mathbf{A}'$ (for the second phase) and require that $\mathbf{A}'$ is fairness-enabling. The scheduler used in the first phase is not required to be fair (in particular it can stop at arbitrary points), and adversary $\mathbf{A}$ is not assumed to be fairness-enabling.

The definition of balance is parameterized by two deterministic polynomial-time algorithms, $\mathtt{goal}_1$ and $\mathtt{goal}_2$, the *goal functions*, which given a global configuration return 1 (goal reached) or 0 (failed to reach the goal), e.g., $\mathtt{goal}_1$ might formalize "A does not have a signed contract from B" and $\mathtt{goal}_2$ might formalize "B has a signed contract from A" (see Section 6.2). Parameterizing the definition of balance by the goal functions seems unavoidable since, for example, what a signed contract is and what it means for a party to have a signed contract are details that may differ from one protocol to another (see, e.g., [1] and [15]). We call a deterministic polynomial-time algorithm which given a global configuration returns 1 (requiring the adversary to achieve $\mathtt{goal}_1$) or 2 (requiring the adversary to achieve $\mathtt{goal}_2$) a *challenge function*.

Definition 4. *Let Π be a protocol and* $\mathtt{goal}_1$ *and* $\mathtt{goal}_2$ *be deterministic polynomial-time algorithms as above. Let $\mathcal{O}_{sch}$ and $\mathcal{O}'_{sch}$ be view oracles, and $\mathcal{O}_{adv}$ and $\mathcal{O}'_{adv}$ be adversary view oracles. Then, Π is called* balanced *w.r.t.* $\mathtt{goal}_1$, $\mathtt{goal}_2$, $\mathcal{O}_{adv}$, $\mathcal{O}'_{adv}$, $\mathcal{O}_{sch}$, *and $\mathcal{O}'_{sch}$ if for all adversary machines $\boldsymbol{A} = \boldsymbol{A}(\mathcal{O}_{adv})$ and $\boldsymbol{A}' = \boldsymbol{A}'(\mathcal{O}'_{adv})$ for Π, and all (not necessarily fair) schedulers $\boldsymbol{S} = \boldsymbol{S}(\mathcal{O}_{sch})$ for Π, there exists a challenge function* $\mathtt{challenge}$ *such that if $\boldsymbol{A}'$ is fairness-enabling for Π, $\mathcal{O}'_{sch}$, $\mathcal{O}_{adv}$, and a family $D = \{D_\eta\}_\eta$ of distributions on (initial) global configurations defined below, then there exists a scheduler $\boldsymbol{S}' = \boldsymbol{S}'(\mathcal{O}'_{sch})$ fair for $\mathcal{S}(\Pi, \boldsymbol{A}')$ and D such that the probability that the following experiment returns* 1 *is negligible in the security parameter η.*

Exp$(\eta, \Pi, \boldsymbol{A}, \boldsymbol{A}', \boldsymbol{S}, \boldsymbol{S}', \mathtt{goal}_1, \mathtt{goal}_2, \mathtt{challenge})$*:*

$\mathcal{S}(\eta) \rightsquigarrow q$ *where* $\mathcal{S} = \mathcal{S}(\Pi, \boldsymbol{A}, \boldsymbol{S})$.
$i = \mathtt{challenge}(q)$.
$\mathcal{S}'_{q'}(\eta) \rightsquigarrow q''$ *where* $\mathcal{S}' = \mathcal{S}(\Pi, \boldsymbol{A}', \boldsymbol{S}')$*, the initial configuration of $\boldsymbol{A}'$ is obtained by writing i and the current configuration of $\boldsymbol{A}$ on the work tape of $\boldsymbol{A}'$, and q' is obtained from q by replacing the configuration of $\boldsymbol{S}$ by the initial configuration of $\boldsymbol{S}'$ and the configuration of $\boldsymbol{A}$ by the initial configuration of $\boldsymbol{A}'$.*
Return $\mathtt{goal}_i(q'')$.

The distribution D_η is defined to be the distribution of q' in the above experiment. (Note that $D = \{D_\eta\}$ is polynomially samplable.)

We emphasize that the above experiment can be simulated in polynomial time. This is a crucial fact when trying to show that a protocol is balanced via a proof by reduction. Note that while one could provide $\mathtt{challenge}$ and $\mathbf{S}'$ with more information, giving them less information only makes the balance property stronger. We also point out that in typical applications of the above definition the protocol Π will be fairness-enabling w.r.t. $\mathcal{O}'_{sch}$, $\mathcal{O}'_{adv}$, and D, and hence, fair scheduling is possible in the second phase of the experiment.

6.2 The ASW Protocol is Balanced

We prove that the ASW protocol is balanced for i) the case that an honest initiator A runs an instance of the protocol with a dishonest responder B (modeled as the adversary) and an honest TTP T, and ii) the case that an honest responder B runs an instance of the protocol with a dishonest initiator A (modeled as the adversary) and an honest TTP T. More formally, we need to specify the protocols, oracles, and functions used as parameters in the balance definition.

Let $\Pi^{\mathsf{ASW\text{-}A}}$ denote the protocol with honest parties A, T, and W, and dishonest party B where A acts as an initiator, T as a TTP, and W as a "watch dog". Formal specifications of A and T in terms of ITMs can be found in [13]. We note that A writes $\mathtt{Contract}$ on some of her work tapes if according to the specification of the protocol she has a valid contract (standard or replacement) with B and T on the contractual text. The watch dog W is used to check whether the adversary (dishonest B) has a valid contract. The protocol $\Pi^{\mathsf{ASW\text{-}B}}$ is defined similarly, except that now A is dishonest and B is honest. The formal

specification of the responder B as ITM can be found in [13]. It is not hard to check that $\Pi^{\mathsf{ASW\text{-}A}}$ and $\Pi^{\mathsf{ASW\text{-}B}}$ are fairness-enabling w.r.t. the distribution used in Definition 4 and that $\mathcal{S}(\Pi^{\mathsf{ASW\text{-}A}})$ and $\mathcal{S}(\Pi^{\mathsf{ASW\text{-}A}})$ are reactively polynomial.

We define the view oracles $\mathcal{O}_{adv}^{\mathsf{ASW}}$ and $\mathcal{O}_{adv'}^{\mathsf{ASW}}$ for the adversary to be adversary view oracles (Section 5) which return the history of all network and resilient channel buffers in the system (but no other machines). In particular, resilient channel buffers are not required to be read protected.

To get strong security guarantees, we allow the scheduler in the first phase of the definition of the balance property to see what the adversary sees plus the history of the configurations of the adversary (including the random coins used by the adversary); $\mathcal{O}_{sch}^{\mathsf{ASW}}$ is defined accordingly. Conversely, we make the scheduler in the second phase weak by defining $\mathcal{O}_{sch'}^{\mathsf{ASW}}$ in such a way that it does not provide any information about the history. For a global configuration q let $\mathtt{goal}_1(q) = 1$ iff the honest party (A in $\Pi^{\mathsf{ASW\text{-}A}}$ and B in $\Pi^{\mathsf{ASW\text{-}B}}$) does not have a contract, i.e., `Contract` is not written on one of its work tapes. Let $\mathtt{goal}_2(q) = 1$ iff the adversary has a valid contract, i.e., `Contract` is written on a work tape of the watch dog.

We are now ready to state the theorem on balance of the ASW protocol. The theorem holds for instances of the protocol implemented with primitives that satisfy standard cryptographic assumptions (see [13] for precise definitions).

Theorem 3. *If the signature scheme is existentially unforgeable under chosen message attacks and the hash function is preimage resistant, then* $\Pi^{\mathsf{ASW\text{-}A}}$ *and* $\Pi^{\mathsf{ASW\text{-}B}}$ *are balanced w.r.t.* $\mathtt{goal}_1$, $\mathtt{goal}_2$, $\mathcal{O}_{adv}^{\mathsf{ASW}}$, $\mathcal{O}_{adv'}^{\mathsf{ASW}}$, $\mathcal{O}_{sch}^{\mathsf{ASW}}$, *and* $\mathcal{O}_{sch'}^{\mathsf{ASW}}$.

The theorem should extend to the case that a party runs multiple copies of the protocol provided that different instances of the protocol use unique session identifiers (see [13]).

References

1. N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *S&P 1998*, pages 86–99. IEEE Computer Society, 1998.
2. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000.
3. M. Backes and B. Pfitzmann. Computational probabilistic non-interference. In *ESORICS 2002*, volume 2502 of *LNCS*, pages 1–23, 2002.
4. M. Backes, B. Pfitzmann, and M. Waidner. Secure Asynchronous Reactive Systems. Technical Report 082, Cryptology ePrint Archive, 2004.
5. Michael Backes, Anupam Datta, Ante Derek, John C. Mitchell, and Mathieu Turuani. Compositional analysis of contract signing protocols. In *CSFW 2005*, pages 94–110, Washington, DC, USA, 2005. IEEE Computer Society.
6. Michael Backes, Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. On fairness in simulatability-based cryptographic systems. In *FMSE 2005*, pages 13–22, September 2005. Preprint on IACR ePrint 2005/294.
7. Michael Backes, Birgit Pfitzmann, Michael Steiner, and Michael Waidner. Polynomial fairness and liveness. In *CSFW 2002*, pages 160–169. IEEE Computer Society, 2002.

8. B. Baum-Waidner and M. Waidner. Round-optimal and abuse free optimistic multi-party contract signing. In U. Montanari, J.D.P. Rolim, and E. Welzl, editors, *ICALP 2000*, volume 1853 of *LNCS*, pages 524–535. Springer, 2000.
9. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Technical report, Cryptology ePrint Archive, December 2005. Online available at `http://eprint.iacr.org/2000/067.ps`.
10. Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Time-bounded Task-PIOAs: A Framework for Analyzing Security Protocols. In S. Dolev, editor, *DISC 2006*, pages 238–253. Springer, 2006.
11. R. Chadha, M.I. Kanovich, and A.Scedrov. Inductive methods and contract-signing protocols. In P. Samarati, editor, *CCS 2001*, pages 176–185. ACM Press, 2001.
12. R. Chadha, J.C. Mitchell, A. Scedrov, and V. Shmatikov:. Contract Signing, Optimism, and Advantage. In R.M. Amadio and D. Lugiez, editors, *CONCUR 2003*, volume 2761 of *LNCS*, pages 361–377. Springer, 2003.
13. V. Cortier, R. Küsters, and B. Warinschi A Cryptographic Model For Branching Time Security Properties — the Case of Contract Signing Protocols. Technical Report 251, Cryptology ePrint Archive, 2007.
14. A. Datta, R. Küsters, J.C. Mitchell, and A. Ramanathan. On the Relationships Between Notions of Simulation-Based Security. In J. Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 476–494. Springer-Verlag, 2005.
15. J.A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 449–466. Springer-Verlag, 1999.
16. D. Hofheinz and J. Müller-Quade. A synchronous model for multi-party computation and the incompleteness of oblivious transfer. In *FCS 2004*.
17. D. Hofheinz, J. Müller-Quade, and D. Unruh. Polynomial Runtime in Simulatability Definitions. In *CSFW 2005*, pages 156–169. IEEE Computer Society, 2005.
18. Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. A simple model of polynomial time UC. Presented at the ECRYPT Workshop on Models for Cryptographic Protocols – MCP'06.
19. D. Kähler and R. Küsters. Constraint Solving for Contract-Signing Protocols. In M. Abadi and L. de Alfaro, editors, *CONCUR 2005*, volume 3653 of *LNCS*, pages 233–247. Springer, 2005.
20. D. Kähler, R. Küsters, and Th. Wilke. Deciding Properties of Contract-Signing Protocols. In Volker Diekert and Bruno Durand, editors, *STACS 2005*, number 3404 in *LNCS*, pages 158–169. Springer-Verlag, 2005.
21. D. Kähler, R. Küsters, and Th. Wilke. A Dolev-Yao-based Definition of Abuse-free Protocols. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *ICALP 2006*, volume 4052 of *LNCS*, pages 95–106. Springer, 2006.
22. S. Kremer and J.-F. Raskin. Game analysis of abuse-free contract signing. In *CSFW 2002*, pages 206–220. IEEE Computer Society, 2002.
23. Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. In *CONCUR 2001*, volume 2154 of *LNCS*, pages 551–565. Springer-Verlag, 2001.
24. R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *CSFW 2006*, pages 309–320. IEEE Computer Society, 2006.
25. V. Shmatikov and J.C. Mitchell. Finite-state analysis of two contract signing protocols. *TCS*, 283(2):419–450, 2002.