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] INRIA, 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, 14, 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 just properties of single execution traces. Examples of such properties are balance [11] and abuse-freeness [14]. 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); while related to models in [4, 9, 13, 16], our exposition follows more closely the one in [23]. Based on the general computational model, we define a security-specific model 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. 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}. Similarly, if a resilient channel is scheduled by the scheduler to deliver a message, then the resilient channel sends the message and then deletes it from its buffer, and hence, changes its local configuration. 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 definitions for fairness and balance of (contract signing) protocols. One should not confuse the concept of fair scheduling with the notion of fairness of protocols. The concept of fair scheduling is needed in the definition of fairness (and balance) of protocols. The definition of balance requires 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 definitions to the ASW two-party optimistic contract-signing protocol [1] and show it to be fair and balanced when implemented with primitives that satisfy standard security assumptions. Our proof of balance of this protocol is the first computational proof of this (now rigorously defined) property for a contract signing protocol. Also, while Asokan et al. [1] argue informally about the fairness of their protocol, our proof of fairness is the first one for this protocol w.r.t. a rigorous definition of fairness.

RELATED WORK. Rigorous models and security definitions for branching time properties of contract signing protocols have already been proposed in [11, 21, 20]. However, these definitions are w.r.t. a symbolic (Dolev-Yao based) model and do not consider the more involved computational case. Within Dolev-Yao based models, different contract-signing protocols have been analyzed using finite-state model checkers or certain logics [24, 21, 5] (see also [12]), and decision procedures for automatic analysis have been proposed [18, 19].

Backes et al. [7] (see also [6]) proposed a definition of fair scheduling in a computational model. Their notion and setting differs from ours in several aspects. First, and most importantly, while their notion of fair scheduling is only w.r.t. the scheduling of buffers (fair message delivery), we need, as explained, a more general notion which captures fair scheduling also for honest principals and the adversary. We therefore base our notion on the general concept of change of local configuration, which is essential in the present work, but has not been considered by Backes et al. The notion by Backes et al. is in fact unsuitable for the properties of contract signing protocols, fairness and balance, we consider since it does not capture that honest principals and the adversary finish their execution. Second, they study fair scheduling in a simulation-based setting which we do not do. Third, the notion of fair scheduling of Backes et al. is parameterized by a polynomial which determines after what time a buffer has to be triggered. Our definition does not need such parameters. Other works that use some kind of fairness in specific settings are [3] and [15]. None of the mentioned works, [7, 6, 3, 15], 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 this 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, which they do not study. In their setting, fair scheduling is only w.r.t. *honest* parties and is guaranteed by imposing restrictions on the adversary; they do not have a separate scheduler. This is insufficient for branching time properties, such as balance: First, as explained, for branching time properties fair scheduling has to be guaranteed also for dishonest parties, i.e., the adversary, which is why we propose a general notion of fair scheduling that applies to arbitrary ITMs. Second, a scheduler independent of the adversary is needed in order to model situations in which the scheduler plays against the adversary.

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.

STRUCTURE OF THE PAPER. We start with an informal description of the ASW protocol which serves as our running example throughout the paper. Next, in Section 3, we present the general computational model, which forms the basis of our security-specific model, introduced in Section 4. We then define fair schedulers in Section 5. The model and the notion of fair schedulers are then used in Section 6 for defining fairness of (contract signing). We also show here that the ASW protocol is fair. The more complex notion of balance, and the proof that the ASW protocol is balanced are in Section 7. We conclude in Section 8. More details and proofs can be found in the appendix.

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 Appendix B.

CRYPTOGRAPHIC PRIMITIVES. The ASW protocol uses concatenation, signatures and (keyed) 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 $\mathsf{sigver}(m, s, v_P) =$ 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

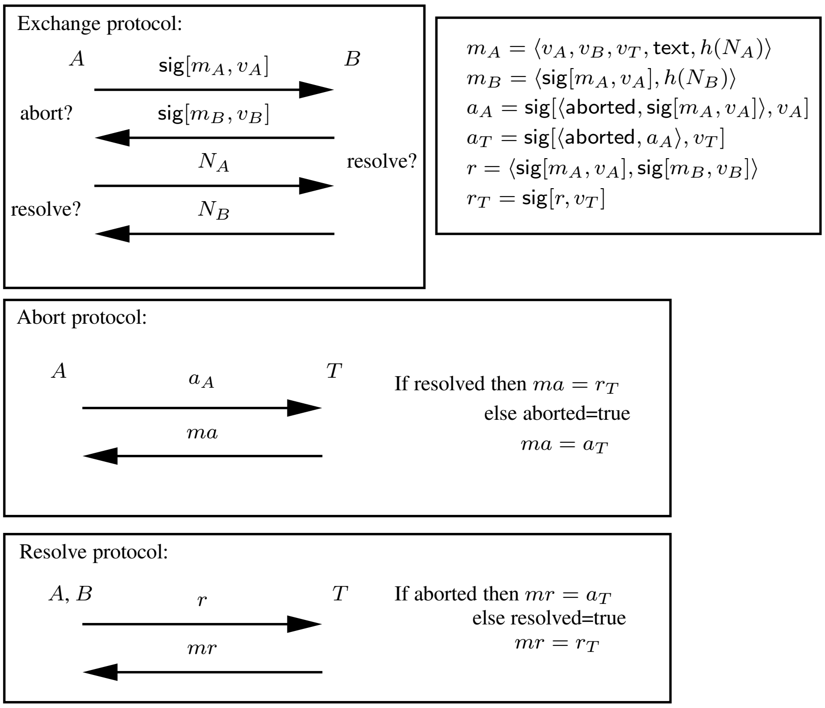

Fig. 1. The ASW Protocol. We indicate where during the execution of the Exchange protocol parties can choose to run the Abort of the Resolve subprotocols.

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 $m_A = \langle v_A, v_B, v_T, \mathsf{text}, h(N_A)\rangle$, $m_B = \langle \mathsf{sig}[m_A, v_A], h(N_B)\rangle$, and N_A and N_B are nonces.

The ASW protocol consists of three subprotocols: the exchange, abort, and resolve protocol. These subprotocols are explained next (see also Figure 1).

Exchange protocol. The basic idea of the *exchange protocol* is that A first indicates her interest to sign the contract. To this end, she sends to B the message $\mathsf{sig}[m_A, v_A]$ as defined above, where N_A is a nonce generated by A. By sending this message, A "commits" to signing the contract. Then, similarly, B indicates his interest to sign the contract by generating a nonce N_B and sending the message $\mathsf{sig}[m_B, v_B]$ to A. 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. (In our formal model, this is modeled as a non-deterministic action of A and we use schedulers to resolve this non-determinism.) In the abort protocol, A first sends the message $a_A = \mathsf{sig}[\langle \mathsf{aborted}, \mathsf{sig}[m_A, v_A]\rangle, v_A]$. 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 with $r = \langle \mathsf{sig}[m_A, v_A], \mathsf{sig}[m_B, v_B]\rangle$.

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. 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. Note that contacting T is again a non-deterministic action of B.

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.

In our formal model, a *fair* scheduler guarantees, for example, that messages on resilient channels will eventually be delivered and that if (honest) A in the exchange protocol is in state "abort?" and (dishonest) B does not respond, then A will start the abort protocol with the TTP.

3 The General Computational Model

Our general computational model talks about systems of interactive Turing machines (ITMs) and is related to the models in [4, 9, 13, 16]. However, our exposition follows more closely that of [23] since the model in [23] contains most features needed in the present work. Similar to [23], our general computational model uses inexhaustible interactive Turing machines; the attribute "inexhaustible" will become clear later. While in [23] systems with an unbounded number of ITMs have been studied, in the present work, we only need to deal with systems consisting of a fixed and finite number of ITMs, and therefore, we do not need to define how new ITMs are generated and how dynamically generated ITMs are addressed. Conversely, in the present work we consider ITMs which may have access to certain oracles. This is a convenient feature of our setting. We note that while the works mentioned above are concerned with simulation-based security, simulation-based security is not considered here; we only borrow the definition of systems of ITMs.

SYNTAX OF ITMS. An *(inexhaustible) interactive Turing machine (ITM, for short)* M is a probabilistic Turing machine with the following tapes: a read-only tape on which the security parameter is written (the *security parameter tape*), a read-only tape on which random coins are stored (the *random tape*), zero or more *input* and *output tapes*, and *work tapes*. The input and output tapes have names and, in addition, input tapes have an attribute with values $\mathsf{consuming}$ or $\mathsf{enriching}$ (see below for an explanation). We require that different tapes of M have different names. The names of input and output tapes 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. We require that each ITM comes with an associated polynomial q which is used to bound the time taken by the computations of M. 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.

An ITM may have a (consuming) input tape named start which serves a particular purpose: It will be used to trigger an ITM if no other ITM was triggered. An ITM is triggered by another ITM if the latter sends a message to the former. An ITM with an input tape named start is called *master ITM*.

COMPUTATION OF ITMS. To specify the computation of an ITM, let l denote the length of the security parameter plus the accumulated length of all inputs written on *enriching* input tapes of M so far (i.e., the sum of the lengths of inputs written on enriching input tapes in the current and all previous activations).

Each time when M is activated, it is the case that the security parameter η is written on the security parameter tape, and one message, say m, is written on one of the input tapes, say c (the other input tapes and the output tapes are empty—or otherwise will be emptied before M starts to run). We require that the computation in every activation of M satisfies the following conditions: (i) Similar to other models [23, 4, 9, 13, 16], at the end of the activation, M has written *at most one* message on one of its output tapes (i.e., only one message can be sent to another ITM at a time), (ii) the number of transitions taken in the activation is bounded by $q(n)$ where q is the polynomial associated with M and n is the security parameter plus the length of the content of the input and work tapes at the beginning of the activation, (iii) the sum of the lengths of all outputs written on output tapes so far by M (in all activations) is bounded by $q(l)$, (iv) at the end of the current activation, the length of the content of the work tapes is bounded by $q(l)$, and (v) if (non-empty) output was written on one of the output tapes, the local configuration of the machine before the activation is different from the local configuration of the machine after the activation, where a *local configuration* of an ITM consists of the current state of the ITM, the content and head positions of all work tapes as well as the head position of the security parameter tape and the random tape. This last condition guarantees that if a machine wants to "take an action" by sending a message (see the introduction), then this is indicated by the change of the local configuration. This is obviously not a real restriction but a useful and natural requirement in the context of fair scheduling. When activated, M may query oracles associated to it. To query one oracle, M writes a message on a designated work tape. The answer of the oracle will then immediately be returned (on some designated work tape). The evaluation of the queries is not part of the computation of M, and in particular, the steps taken by oracles are not added to M's runtime. Once M finishes its current activation, the input tapes are emptied. Hence, at the end of an activation at most one of the output tapes is non-empty and the other output tape as well as the input tapes are empty.

We emphasize that an ITM M as defined above can not be exhausted (therefore the name *inexhaustible* interactive Turing machine): Whenever M is activated it is able to "scan" its complete current configuration, including the incoming message. As can be seen from the above conditions, by writing messages on *enriching* input tapes of M the resources of M, in terms of number and length of messages M may output and the size of the local configuration M may have, increases. Conversely, messages written on *consuming* input tapes of M do not increase M's resources.

SYSTEMS OF ITMS. A *system* $\mathcal{S}$ of ITMs is a parallel composition $M_1 \,||\, \cdots \,||\, M_n$ of ITMs M_i, $i = 1, \ldots, n$, such that the set of names of input tapes of M_i is disjoint from the set of names of input tapes of M_j for $i \neq j$. In particular, $\mathcal{S}$ can only have at most one master ITM, i.e., at most one ITM in $\mathcal{S}$ may have start as input tape. Also, the output tape of an ITM in $\mathcal{S}$ is connected to at most one input tape of (another) ITM. The *set of tapes of a system* $\mathcal{S}$ is defined to be the set of all tapes of ITMs occurring in $\mathcal{S}$. We call a tape of $\mathcal{S}$ *internal* if it occurs both as an input tape of an ITM in $\mathcal{S}$ and an output tape of (another) ITM in $\mathcal{S}$. Otherwise, a tape is called *external*. An external tape is called *external input tape* if it occurs as an input tape of some ITM in $\mathcal{S}$. Otherwise, it is called *external output tape*.

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.

RUNS OF SYSTEMS. In a run of $\mathcal{S}(1^\eta, r_1, \ldots, r_n)$ at every time only one ITM is active and all other ITMs wait for new input. The active machine may write at most one message on one of its output tapes, say c. This message is then delivered to an ITM with an input tape named c, if any (recall that there exists at most one such machine). The previously active machine goes into a wait state and the receiver of the message is activated, resulting, after some internal computation, into a new output which is sent to another ITM, and so on. The first ITM to be activated in a run is the master ITM. It gets ε as external input (on tape start). The master ITM is also activated if an ITM does not produce output (and hence, does not trigger another machine) or the output is written on an output tape for which there is no ITM with a matching input tape. A run stops if the master ITM, after being activated, does not produce output. More formally, 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 given a security parameter simulates runs of $\mathcal{S}$ and runs in polynomial-time with overwhelming probability (in the security parameter). 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. It suffices to choose a polynomial number of coins since the portion of runs exceeding the polynomial bound is negligible and can be ignored. For a global configuration q, 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) all input tapes of $\mathcal{E}$ are consuming 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 $\mathcal{S}$: formally, the set of input tapes of $\mathcal{E}$ is disjoint from the set of external input tapes and internal tapes of $\mathcal{S}$, and the set of output tapes of $\mathcal{E}$ is disjoint from the set of external output and internal tapes of $\mathcal{S}$. Adopting terminology from [17], 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

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

where $\mathbf{H}_i$, $i \in \mathcal{H}$, is a protocol machine of Π modeling an honest principal, $\mathbf{Net}^i_j$, $i \in \mathcal{H}$, $j \in \mathcal{P}$ is a network buffer (machine) on which i sends messages over the network intended for j, $\mathbf{RC}^i_j$, $i \in \mathcal{H}$, $j \in \mathcal{P}$ 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, $\mathbf{A}$, and $\mathbf{S}$* 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 Appendix A for details).

A network buffer machine $\mathbf{Net}^i_j$ works as follows: It internally stores a sequence of messages, which is initially empty. Whenever it receives a message from $\mathbf{H}_i$ (on some designated tape), it appends this message at the end of the internal sequence and acknowledges receipt of the message by sending `ack` on back to $\mathbf{H}_i$. The acknowledgment gives control back to $\mathbf{H}_i$ thereby allowing $\mathbf{H}_i$ to send further messages (to possibly other buffers). In other words, $\mathbf{H}_i$ can broadcast messages. We do not have tapes between the adversary machine and the network buffer as the adversary can read the network buffer via its view oracle (see below).

A resilient channel buffer machine $\mathbf{RC}^i_j$ works as follows: It internally stores a sequences of messages, which initially is empty. Whenever $\mathbf{RC}^i_j$ receives a message from $\mathbf{H}_i$, it appends it at the end of the internal sequence and acknowledges receipt of the message by sending `ack` back to $\mathbf{H}_i$. (Again, the purpose of the acknowledgment is to enable $\mathbf{H}_i$ to broadcast messages.) The resilient channel buffer is scheduled by the scheduler who can send a number k to $\mathbf{RC}^i_j$ to instruct $\mathbf{RC}^i_j$ to deliver the kth message of the sequence of messages stored in $\mathbf{RC}^i_j$ (if any). Again, the adversary does not have direct access to $\mathbf{RC}^i_j$. If $\mathbf{RC}^i_j$ is not required to be read-protected, then the view oracle of the adversary can be defined in such a way that it provides the adversary with the messages stored in $\mathbf{RC}^i_j$.

A protocol machine $\mathbf{H}_i$ may send messages to the network buffers $\mathbf{Net}^i_j$ and the resilient channel buffers $\mathbf{RC}^i_j$ for every $j \in \mathcal{P}$ as explained above. If $\mathbf{H}_i$ does not produce output, the scheduler $\mathbf{S}$ (which is declared to be the master ITM) is activated. A protocol machine $\mathbf{H}_i$ can be activated in three different ways: a) It receives a message from the network on $\texttt{netin}^j_i$ supposedly from j for some $j \in \mathcal{P}$ (these messages will always come from the adversary who controls the network); b) It receives a message from a resilient channel $\texttt{rcin}^j_i$ from j for $j \in \mathcal{P}$ (if $j \in \mathcal{H}$, then the message received was in fact sent by j and if $j \in \mathcal{D}$, then the message comes from some dishonest principal, and hence, the adversary); c) It receives a message (on some designated tape) from the scheduler, where we assume that $\mathbf{H}_i$ only accepts a fixed, finite set of messages on this channel and ignores all messages that do not belong to this set. The messages from the scheduler are meant to resolve non-deterministic choices made by $\mathbf{H}_i$. If, for example, in the ASW protocol, at some point of the protocol run $\mathbf{H}_i$ has the choice to wait for a message (sent over the network) from the communication partner or start the abort protocol with the TTP, then the scheduler could send the message `abort` to $\mathbf{H}_i$ in order to instruct $\mathbf{H}_i$ to start the abort protocol.

We allow all input tapes of network, resilient channel, and protocol machines to be enriching. We therefore explicitly require that the system $\mathcal{S}(\Pi)$ is reactively polynomial. (For a given protocol Π this is typically not hard to check, see, e.g., Section 6.2 and 7.2.) Note that if all input tapes of protocol machines were consuming (the buffers could have enriching input tapes), then reactive polynomiality would follow. However, if protocol machines may have enriching tapes, then, for example, TTPs (as those in the ASW protocol) can conveniently be modeled in such a way that they process an arbitrary number of requests, without any fixed polynomial bound.

The adversary machine **A** is associated with an oracle, called the *view oracle*. Recall that if this oracle is $\mathcal{O}$, we often write $\mathbf{A}(\mathcal{O})$ to say that **A** is an ITM with associated oracle $\mathcal{O}$. This oracle can be invoked by **A** to obtain a *view* on the history of the run of the overall system so far. The exact definition of the view oracle depends on what **A** should be allowed to see. Typically, the view contains not full information about the history but the content of all network buffers (so far) and the content of (some) resilient channel buffers. The view of the resilient channels depends on the type of the channel. For example, for an authenticated but not read-protected channel the view oracle returns the complete content of the channel. In addition to invoking the view oracle, **A** can send messages to honest principals either via network or resilient channel connections. A message sent by the adversary on one of these channels is delivered directly. In particular, the protocol machine connected to this channel will be activated immediately. More precisely, since network connections are not authenticated, **A** can send a message pretending to be j directly to honest principals $\mathbf{H}_i$, $i \in \mathcal{H}$, via the tape $\mathtt{netin}_i^j$ for every $j \in \mathcal{P}$. Resilient channels are meant to be authenticated and therefore the adversary can only send a message pretending to be j' directly to an honest principal $\mathbf{H}_i$, $i \in \mathcal{H}$ (via the tape $\mathtt{rcin}_i^{j'}$) if $j' \in \mathcal{D}$. A possible alternative to allowing the adversary to send messages directly to the honest principals is to add resilient (scheduler controlled) and/or network channel buffers between the adversary and honest principals. We note, however, that in this case the adversary would be less powerful, and therefore the resulting model would yield weaker security guarantees. The adversary machine **A** can be activated by the scheduler (and no other machine). For this purpose, the scheduler sends `schedule` on some designated tape to **A**. We require that **A** ignores all other messages on this tape. All input tapes of **A** may be enriching. However, we only allow those adversary machines for which the system $\mathcal{S}(\Pi, \mathbf{A})$ is reactively polynomial, which, for example, includes all adversary machines whose input tapes are consuming. (Recall that $\mathcal{S}(\Pi)$ is also required to be reactively polynomial.)

The scheduler **S** is also associated with a *view oracle* which provides **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 **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 **A** is triggered. More precisely, **S** can send messages to $\mathbf{H}_i$, $i \in \mathcal{H}$, in order to resolve non-deterministic choices, e.g., in the ASW protocol **S** could send `abort` or `resolve` to $\mathbf{H}_i$ in order to instruct $\mathbf{H}_i$ to start the abort or resolve protocol. As explained above, the scheduler can also send messages to the resilient channel buffers $\mathbf{RC}_j^i$ to determine which message is scheduled next. The message scheduled is then immediately sent to the intended recipient j. Finally, **S** can send the message `schedule` to **A** in order to trigger **A**. Note that there is no direct connection between **S** and the network buffers since these buffers are under the control of the adversary. However, the view oracle of **S** might (or might not, depending on the security property and the desired strength of the security guarantee) provide **S** with the messages stored in network buffers.

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]; these works were only concerned with fair message delivery, which, however, does not suffice for fairness and balance of contract signing protocols.

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

Alternatively to using a full-information oracle, the definition could be parameterized with an oracle that $\mathbf{S}'$ is allowed to use.

Applied to protocol systems (Section 4), a fair scheduler may only stop if i) the resilient channel buffers are empty, since otherwise a scheduler could schedule a message in a non-empty buffer, which would cause the buffer to deliver the message and delete it, and hence, the buffer would change its local configuration, 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, since otherwise a scheduler could send such a message causing the protocol machine to change its local configuration (e.g., go from state q_{wait} to q_{abort}), 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 otherwise a scheduler could send `schedule` to the adversary and the adversary would change its local configuration.

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 exist 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. First, we put a restriction on the adversary. As formalized in the following definition, we require that the number of configuration changes of the adversary in a run of a system (and hence, the number of actions, such as sending messages, the adversary can perform) can polynomially be bounded independently of the scheduler considered. This restriction follows the intuition that the adversary is the entity which "pushes" the run of a system, and hence, it is mainly the adversary who determines the runtime of the system. Conversely, the scheduler is not meant to "push" the run of the system. It is only an imaginary entity which is used to determine how non-deterministic choices are resolved in real protocol runs and who goes next if anybody wants to take an action. In particular, note that the role of the scheduler is different from the role of an environment in simulation-based settings: Such an environment tries to distinguish real from ideal systems, and therefore, "pushes" the run of a system following its own interests. In the following definition, the number of changes of local configurations of the adversary in a run $q_1 \cdots q_n$ of a system is defined as follows: If $q_i^{\mathbf{A}}$ denotes the local configuration of **A** in the global configuration q_i, then this number is $\#\{i \in \{0, \ldots, n-1\} \mid q_i^{\mathbf{A}} \neq q_{i+1}^{\mathbf{A}}\}$; in Definition 3 we use an analogous definition for protocol machines.

Definition 2. *Given a protocol Π, oracles $\mathcal{O}_{adv}$ and $\mathcal{O}_{sch}$, and a family of distributions $D = \{D_\eta\}_\eta$ on (initial) global configurations, we say that an adversary machine $\boldsymbol{A}(\mathcal{O}_{adv})$ for Π, $\mathcal{O}_{adv}$, $\mathcal{O}_{sch}$, and D is* fairness-enabling *if there exists a polynomial p such that for all schedulers $\boldsymbol{S}(\mathcal{O}_{sch})$ for Π the probability that in a run of $\mathcal{S}_D(\eta)$, with $\mathcal{S} = \mathcal{S}(\Pi, \boldsymbol{A}(\mathcal{O}_{adv}), \boldsymbol{S}(\mathcal{O}_{sch}))$, the number of changes of local configurations of $\boldsymbol{A}(\mathcal{O}_{adv})$ is bounded by $p(\eta)$ is overwhelming (in η), where the probability is over the random coins used by D_η and the machines in $\mathcal{S}$.*

Analogously to the above definition, we could put a restriction on protocol machines. However, this 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. *Given a protocol Π, oracles $\mathcal{O}_{adv}$ and $\mathcal{O}_{sch}$, and a family of distributions $D = \{D_\eta\}_\eta$ on (initial) global configurations, we say that Π is* fairness-enabling *if for all fairness-enabling adversary machines $\boldsymbol{A}(\mathcal{O}_{adv})$ for Π, $\mathcal{O}_{adv}$, $\mathcal{O}_{sch}$, and D there exists a polynomial p such that for all schedulers $\boldsymbol{S}(\mathcal{O}_{sch})$ for Π the probability that in a run of $\mathcal{S}_D(\eta)$, with $\mathcal{S} = \mathcal{S}(\Pi, \boldsymbol{A}(\mathcal{O}_{adv}), \boldsymbol{S}(\mathcal{O}_{sch}))$, the number of changes of local configurations of every protocol machine of $\mathcal{S}$ is bounded by $p(\eta)$ is overwhelming (in η), where the probability is over the random coins used by D_η and the machines in $\mathcal{S}$.*

The following theorem, proved in Appendix C, 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 Fair Protocols and Results for the ASW Protocol

In this section, we define the notion of fairness of protocols and, as a proof of concept, apply it to the ASW protocol. In the definition of fairness, we use the previously introduced concept of fair schedulers. We note that fairness is not a branching time property. However, it is a good warming-up for the more complex notion of balance studied in the next section.

6.1 Definition of Fairness

The definition of fairness of a protocol Π is w.r.t. a deterministic polynomial-time algorithm `checkfair` which given a global configuration of a run of a system for Π returns 1 (for fair) or 0 (for unfair). We do not put any restriction on `checkfair` at this point. This function will be defined depending on the protocol and the party under consideration. In the ASW protocol, for instance, `checkfair` may return 0 in a global configuration if dishonest B has a signed contract from honest A but A does not have a signed contract from B (see Section 6.2). Parameterizing the definition of fairness by `checkfair` 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 [14]).

The following definition says that Π is fair (relative to a particular `checkfair` algorithm) if for every (fairness-enabling) adversary and every *fair* scheduler the probability that a run ends in an unfair global configuration is negligible. (One should not confuse fair scheduling with fair global configurations, the latter is determined by `checkfair`.) While in the following definition, we use the notions of fairness-enabling protocol, fairness-enabling adversaries, and fair scheduler, which were defined w.r.t. a family of distributions $D = \{D_\eta\}_\eta$ on (initial) global configurations, we now omit D and simply assume that (standard) initial configurations, with empty work tapes, are used as starting points of runs.

Definition 4. *Let Π be a fairness-enabling protocol, $\mathcal{O}_{sch}$ be a view oracle for a scheduler, $\mathcal{O}_{adv}$ be an adversary view oracle, and* `checkfair` *be a deterministic polynomial-time algorithm as above. Then, Π is called* fair *w.r.t. $\mathcal{O}_{sch}$, $\mathcal{O}_{adv}$, and* `checkfair` *if for every fairness-enabling adversary machine $\boldsymbol{A} = \boldsymbol{A}(\mathcal{O}_{adv})$ for Π and every scheduler $\boldsymbol{S} = \boldsymbol{S}(\mathcal{O}_{sch})$ fair for $\mathcal{S}(\Pi, \boldsymbol{A})$, we have that the probability that in the following experiment 0 is returned is negligible in the security parameter η where the probability is taken over the random coins of the protocol machines of Π, the adversary $\boldsymbol{A}$, and the scheduler $\boldsymbol{S}$.*

Exp$(\eta, \Pi, \boldsymbol{A}, \boldsymbol{S}, \texttt{checkfair})$*:*

$\mathcal{S}(\eta) \rightsquigarrow q$ *where* $\mathcal{S} = \mathcal{S}(\Pi, \boldsymbol{A}, \boldsymbol{S})$.
Return $\texttt{checkfair}(q)$.

Note that the above definition would not make sense if our notion of fair scheduling would only talk about fair message delivery (as, e.g., in [7, 6]) as in this case a fair scheduler could stop the run of the system even though, for example, an honest principal could still contact the TTP or the adversary still wants to send a message to some honest principal. Hence, fair message delivery on its own would be insufficient for defining fairness of, e.g., contract signing protocols.

6.2 The ASW Protocol is Fair

We prove that the ASW protocol is fair 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 on the contractual text text, 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 on text.

More formally, 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 Appendix B.1 and B.1, respectively (also see the remarks and notation at the beginning of Appendix B.1). We note that A writes `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 text. The watch dog W is used to check whether the adversary (dishonest B) has a valid contract. More precisely, W waits for a message m on some network channel and writes `Contract` on some of its work tapes if m is a standard or replacement contract for A,B,T,text as described in Section 2; W ignores messages if they do not have the correct format. 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 Appendix B.1. It is not hard to check that $\Pi^{\mathsf{ASW\text{-}A}}$ ($\mathcal{S}(\Pi^{\mathsf{ASW\text{-}A}})$) and $\Pi^{\mathsf{ASW\text{-}B}}$ ($\mathcal{S}(\Pi^{\mathsf{ASW\text{-}A}})$) are fairness-enabling (reactively polynomial).

The algorithm `checkfair` that checks whether a global configuration is fair for an honest party is defined as follows: given a global configuration q, $\texttt{checkfair}(q) = 1$ if and only if `Contract` is not written on the work tape of W or `Contract` is written on the honest protocol machine A and B for $\Pi^{\mathsf{ASW\text{-}A}}$ and $\Pi^{\mathsf{ASW\text{-}B}}$, respectively, i.e., `checkfair` returns 1 if the adversary (dishonest party) does not have a signed contract from the honest party or the honest party has a signed contract from the dishonest party.

We define the view oracle $\mathcal{O}_{adv}^{\mathsf{ASW}}$ for the adversary to be an adversary view oracle (Section 5) which returns 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 consider a very weak view oracle $\mathcal{O}_{sch}^{\mathsf{ASW}}$ for the scheduler which provides the scheduler with no information whatsoever about the current status of the protocol run; this potentially makes the job of the adversary easier. (Note that according to Theorem 2, in this situation fair scheduling is still possible.)

We are now ready to state the theorem on fairness of the ASW protocol. The theorem holds for instances of the protocol implemented with primitives that satisfy standard cryptographic assumptions (see Appendix F and G 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 fair w.r.t.* `checkfair` *and view oracles $\mathcal{O}_{adv}^{\mathsf{ASW}}$ and $\mathcal{O}_{sch}^{\mathsf{ASW}}$.*

The proof, presented in Appendix D, is by reduction to the security of the underlying cryptographic primitives. It cannot be carried out using existing results on relating symbolic and cryptographic methods since these results do not (and in some cases provably cannot) take into account preimage-resistant hash functions. Our proof uses in an essential way that schedulers are fair. Without fair scheduling the proof would not go through and in fact the notion of fairness would not make sense since as soon as the dishonest party has a valid contract, the scheduler could stop the run of the protocol. We note that the ASW protocol could be proved to be unfair in our setting, if honest parties are optimistic in the sense that they only contact the TTP if the dishonest party tells them to do so (see [12] for more on optimistic parties in Dolev-Yao based models).

While in $\Pi^{\mathsf{ASW\text{-}A}}$ and $\Pi^{\mathsf{ASW\text{-}B}}$ the honest initiator and responder, respectively, are modeled in such a way that they only run one instance of the protocol, we can, as illustrated in Appendix B.2, also model principals that run an unbounded number of copies of the protocol. The proof of the theorem should extend to this case if for different instances of the protocol unique session identifiers are used (see remarks in Section B.2), but this is not the main focus of this paper.

7 Balanced Protocols and Results for the ASW Protocol

In this section, we define the notion of balance and show that the Asokan-Shoup-Waidner protocol is balanced. As in the definition of fairness, the definition uses the previously introduced concept of fair scheduling.

7.1 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 of *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 **A** and a scheduler **S** which may resolve non-deterministic choices of honest principals and schedule messages on resilient channels and the adversary in a way that helps **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 **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, as in case of fairness (Section 6.1), be insufficient.

In order to ensure that, in the second phase, fair scheduling is possible, we split the adversary in two parts, **A** and **A**′—one for the first phase and one for the second phase of the experiment—and require that **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 **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). Similarly to the function `checkfair` (Section 6.1), the precise definition of these functions depends on the protocol under consideration and cannot be avoided in the general definition (see Section 7.2 for an example of these functions). 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 5. *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* `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.*

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

$\mathcal{S}(\eta) \rightsquigarrow q$ *where* $\mathcal{S} = \mathcal{S}(\Pi, \boldsymbol{A}, \boldsymbol{S})$.
$i = \mathtt{challenge}(q)$.

$\mathcal{S}'_{q'}(\eta) \rightsquigarrow q''$ where $\mathcal{S}' = \mathcal{S}(\Pi, \mathbf{A}', \mathbf{S}')$, the initial configuration of $\mathbf{A}'$ is obtained by writing i and the current configuration of $\mathbf{A}$ on the work tape of $\mathbf{A}'$, and q' is obtained from q by replacing the configuration of $\mathbf{S}$ by the initial configuration of $\mathbf{S}'$ and the configuration of $\mathbf{A}$ by the initial configuration of $\mathbf{A}'$.
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 `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.

7.2 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 on the contractual text text, 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 on text. More formally, we need to specify the protocols, oracles, and functions used as parameters in the balance definition.

The protocols that we consider are the same as in Section 6.2, i.e., $\Pi^{\mathsf{ASW\text{-}A}}$ (honest initiator A) and $\Pi^{\mathsf{ASW\text{-}B}}$ (honest responder B); it is easy to check that these protocols are fairness-enabling w.r.t. the distribution used in Definition 5. We also define $\mathcal{O}^{\mathsf{ASW}}_{adv}$ and $\mathcal{O}^{\mathsf{ASW}}_{adv'}$ as in Section 6.2. 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); $\mathcal{O}^{\mathsf{ASW}}_{sch}$ is defined accordingly. Conversely, we make the scheduler in the second phase weak by defining $\mathcal{O}^{\mathsf{ASW}}_{sch'}$ 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. The following theorem is proved in Appendix E.

Theorem 4. *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.* $\mathtt{goal}_1$, $\mathtt{goal}_2$, $\mathcal{O}^{\mathsf{ASW}}_{adv}$, $\mathcal{O}^{\mathsf{ASW}}_{adv'}$, $\mathcal{O}^{\mathsf{ASW}}_{sch}$, *and* $\mathcal{O}^{\mathsf{ASW}}_{sch'}$.

The proof is again done by reduction to the security of the primitives: Assuming that the protocol is unbalanced, it is shown that one of the primitives would be insecure. As in case of fairness, the proof should extend to the case that a party runs multiple copies of the protocol (see also Section B.2).

8 Conclusion

In this paper, we introduced the first computational model for the specification and rigorous analysis of complex, branching time properties of protocols. Our model includes schedulers to deal with non-deterministic behavior of principals and resilient channels. We proposed a general definition of what it means for a scheduler to be fair. Our definition not only takes into account fair scheduling for honest parties and certain channels, but also *dishonest* parties, and hence, arbitrary ITMs. This definition is of interest independent of our application to branching time properties. Using our computational model and the notion of fair scheduling, we provided definitions of fairness and balance in (contract signing) protocols. The definition of balance required to talk about different strategies and goals of principals, and involved both schedulers that work with and schedulers that work against the adversary. As a proof of concept, we applied these definitions to the ASW two-party contract signing protocol. Our model and the notion of fair scheduling that we introduced form a good basis for also dealing with other branching time properties, such as abuse-freeness, which is a weak form of balance, or properties studied in [22, 21]. Our computational model uses an interleaving semantics; it might be interesting to study concurrent models as concurrency may have an impact on the security properties (see, e.g., [20] for the case of Dolev-Yao based models).

References

1. N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 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 *Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS)*, volume 2502 of *Lecture Notes in Computer Science*, 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 '05: Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW'05)*, 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 *3rd ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code*, 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 *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, 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, *Automata, Languages and Programming, 27th International Colloqium (ICALP 2000)*, volume 1853 of *Lecture Notes in Computer Science*, 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, *20th International Symposium on Distributed Computing (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, *8-th ACM Conference on Computer and Communications Security (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 - Concurrency Theory, 14th International Conference*, volume 2761 of *Lecture Notes in Computer Science*, pages 361–377. Springer, 2003.
13. A. Datta, R. Küsters, J.C. Mitchell, and A. Ramanathan. On the Relationships Between Notions of Simulation-Based Security. In J. Kilian, editor, *Proceedings of the 2nd Theory of Cryptography Conference (TCC 2005)*, volume 3378 of *Lecture Notes in Computer Science*, pages 476–494. Springer-Verlag, 2005. Full version available at http://www.ti.informatik.uni-kiel.de/~kuesters/publications_html/DattaKuestersMitchellRamanathan-TR-SPPC-2004.ps.gz.
14. J.A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology – CRYPTO'99, 19th Annual International Cryptology Conference*, volume 1666 of *Lecture Notes in Computer Science*, pages 449–466. Springer-Verlag, 1999.
15. D. Hofheinz and J. Müller-Quade. A synchronous model for multi-party computation and the incompleteness of oblivious transfer. In *Proceedings of the Foundations of Computer Security Workshop, Proceedings of FCS 2004, 2004.*, 2004.
16. D. Hofheinz, J. Müller-Quade, and D. Unruh. Polynomial Runtime in Simulatability Definitions. In *18th IEEE Computer Security Foundations Workshop (CSFW-18 2005)*, pages 156–169. IEEE Computer Society, 2005.
17. 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.

18. D. Kähler and R. Küsters. Constraint Solving for Contract-Signing Protocols. In M. Abadi and L. de Alfaro, editors, *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR 2005)*, volume 3653 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2005.
19. D. Kähler, R. Küsters, and Th. Wilke. Deciding Properties of Contract-Signing Protocols. In Volker Diekert and Bruno Durand, editors, *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS 2005)*, number 3404 in Lecture Notes in Computer Science, pages 158–169. Springer-Verlag, 2005.
20. 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, *Proceedings of the 33rd International Colloqium on Automata, Languages, and Programming (ICALP 2006)*, volume 4052 of *Lecture Notes in Computer Science*, pages 95–106. Springer, 2006.
21. S. Kremer and J.-F. Raskin. Game analysis of abuse-free contract signing. In *Computer Security Foundations Workshop 2002 (CSFW 2002)*, pages 206–220. IEEE Computer Society, 2002.
22. Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. In *12th International Conference on Concurrency Theory (CONCUR 2001)*, volume 2154 of *Lecture Notes in Computer Science*, pages 551–565. Springer-Verlag, 2001.
23. R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 309–320. IEEE Computer Society, 2006.
24. V. Shmatikov and J.C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science (TCS), special issue on Theoretical Foundations of Security Analysis and Design*, 283(2):419–450, 2002.

A Specification of the ITMs in Protocol Systems

In this section, we provide more formal definitions of protocol, network buffer, resilient channel buffer, adversary, and scheduler machines. In particular, we are more precise about how machines are connected via tapes.

A *protocol machine* $M = \mathbf{A}_i$, $i \in \mathcal{H}$ for principal i in protocol Π is an ITM which has the following tapes for every $j \in \mathcal{P}$:

- Input tapes $\mathtt{netin}_i^j$ and $\mathtt{rcin}_i^j$ for receiving input over the network or the resilient channel from j, respectively.
- Output tapes $\mathtt{netout}_j^i$ and $\mathtt{rcin}_j^i$ for sending messages on the network or the resilient channel to j, respectively. (Theses messages will be written into a network or resilient channel buffer, which will acknowledge receipt of the messages).
- Input tapes $\mathtt{netack}_j^i$ and $\mathtt{rcack}_j^i$ on which M receives an acknowledgment that the message sent before on $\mathtt{netout}_j^i$ or $\mathtt{rcin}_j^i$, respectively, has been received. (This allows to broadcast messages since after sending a message on one of the network or resilient channel output tapes, control is given back to M such that M can send other messages as well).
- An input tape $\mathtt{sch\_p}_i$ on which M expects a message, such as `abort` or `resolve`, after which M may or may not take an action (e.g., start the abort or resolve subprotocol).

All input tapes of M are defined to be enriching.

A *network buffer (machine)* $\mathbf{Net}_j^i$, $i \in \mathcal{H}$ and $j \in \mathcal{P}$, for a protocol Π is an ITM which has the following tapes:

- An input tape $\mathtt{netout}_j^i$ for receiving messages from $\mathbf{A}_i$. (Recall that $\mathbf{A}_i$ has an identically named output tape.)
- An output tape $\mathtt{netack}_j^i$ to acknowledge receipt of a message from $\mathbf{A}_i$. (Recall that $\mathbf{A}_i$ has an identically named input tape.)

All input tapes of $\mathbf{Net}_j^i$ are enriching. The machine $\mathbf{Net}_j^i$ works as follows: It internally stores a sequence of messages, which initially is empty. Whenever $\mathbf{Net}_j^i$ receives a message on $\mathtt{netout}_j^i$, it appends it at the end of the internal sequence and acknowledges receipt of the message by sending `ack` on $\mathtt{netack}_j^i$ (to $\mathbf{A}_i$).

A *resilient channel buffer (machine)* $\mathbf{RC}^i_j$, $i \in \mathcal{H}$ and $j \in \mathcal{P}$,[4] for a protocol Π is an ITM which has the following tapes:

- An input tape $\texttt{rcout}^i_j$ for receiving messages from $\mathbf{A}_i$. (Recall that $\mathbf{A}_i$ has an identically named output tape.)
- An output tape $\texttt{rcack}^i_j$ to acknowledge receipt of a message from $\mathbf{A}_i$. (Recall that $\mathbf{A}_i$ has an identically named input tape.)
- An input tape $\texttt{rcin\_sch}^i_j$ on which $\mathbf{RC}^i_j$ expects a number (the index of the message to be sent) from the scheduler.
- An output tape $\texttt{rcin}^i_j$ on which $\mathbf{RC}^i_j$ sends the messages requested on $\texttt{rcin\_sch}^i_j$.

All input tapes of $\mathbf{RC}^i_j$ are enriching. The machine $\mathbf{RC}^i_j$ works as follows: It internally stores a sequences of messages, which initially is empty. Whenever $\mathbf{RC}^i_j$ receives a message on $\texttt{rcout}^i_j$, it appends it at the end of the internal sequence and acknowledges receipt of the message by sending $\texttt{ack}$ on $\texttt{rcack}^i_j$ (to $\mathbf{A}_i$). Whenever $\mathbf{RC}^i_j$ receives a number k on $\texttt{rcin\_sch}^i_j$ (from the scheduler),[5] it writes the kth message of the sequences on the tape $\texttt{rcin}^i_j$ (if the kth message exists, otherwise is does nothing) and deletes this message from the sequence.

An *adversary (machine)* $\mathbf{A}$ for a protocol Π is an ITM which may have a view oracle as described in Section 4 and which has the following tapes:

- Output tapes $\texttt{netin}^j_i$ for all $j \in \mathcal{P}$ and $i \in \mathcal{H}$ to send a message (as j) to i over the network.
- Output tapes $\texttt{rcin}^j_i$ for all $j \in \mathcal{D}$ and $i \in \mathcal{H}$ to send a message (as j) to i over the resilient channel.
- An input tape $\texttt{schadv}$ on which $\mathbf{A}$ expects the message $\texttt{schedule}$ after which $\mathbf{A}$ may or may not take an action.

All input tapes of $\mathbf{A}$ are enriching. Note that $\mathbf{A}$ does not have any direct connection to the network buffers. This is because $\mathbf{A}$ can use the view oracle to obtain all messages sent to the network. Similarly for resilient channels.

A *scheduler (machine)* $\mathbf{S}$ for a protocol Π is an ITM which has the following tapes:

- Output tapes $\texttt{sch\_p}_i$, for every $i \in \mathcal{H}$, on which $\mathbf{S}$ may write a message to trigger $\mathbf{A}_i$. (Recall that $\mathbf{A}_i$ will only accept a message among a finite set of messages and will ignore all other messages.)
- Output tapes $\texttt{rcin\_sch}^i_j$, for every $i \in \mathcal{H}$ and $j \in \mathcal{P}$, on which $\mathbf{S}$ may write a number, the index of the message to be send over the resilient channel from i to j.[6]
- An output tape $\texttt{schadv}$ on which $\mathbf{S}$ may write $\texttt{schedule}$ to trigger $\mathbf{A}$.
- An input tape start, i.e., $\mathbf{S}$ is a master ITM.

All input tapes of $\mathbf{S}$ are declared to be consuming. As explained in Section 4, the scheduler $\mathbf{S}$ is equipped with an oracle, called *view oracle*. This oracle can be invoked by $\mathbf{S}$ to obtain a view on the history of the run of the overall system so far.

[4] Alternatively, one could consider resilient channel buffer machines also for $i \in \mathcal{D}$ (and in this case $j \in \mathcal{H}$), which would mean that the adversary would also have to write into a resilient channel buffer machine to send a message to a another principal.

[5] Alternatively, one could require that $\mathbf{RC}^i_j$ only expects a message $\texttt{next}$ which would trigger $\mathbf{RC}^i_j$ to send the next message in its internal sequence.

[6] If $j \in \mathcal{D}$ and one wants to model that the channel $\mathbf{RC}^i_j$ is controlled by the adversary, then $\mathbf{S}$ would not have the output tape $\texttt{rcin\_sch}^i_j$, i.e., $\mathbf{S}$ can not trigger $\mathbf{RC}^i_j$.

B Formal Specification of the ASW Protocol

In this section, we provide a formal specification of the ASW protocol. More precisely, we specify the actions of honest principals in the ASW protocol as ITMs. While in Section B.1, we consider honest principals running a single session on their machine, in Section B.2 we specify an honest principal running multiple sessions.

B.1 Single Session Specification

In this section, we specify A running one instance of the ASW protocol as an initiator with B (Section B.1), B running one instance of the ASW protocol as a responder with A (Section B.1), and the TTP (Section B.1).

For simplicity of presentation, in the following specifications we implicitly assume an initialization phase where the parties are provided with a random $k \in \{0,1\}^\eta$ (the index of the hash function to use) and the public-keys of the other parties. This initialization phase could be modeled in different ways. For example, every entity could generate its own public and private keys and then send the public keys over resilient channels to the other parties. One trusted entity, e.g., the TTP, could in addition randomly choose k and send it to the other entities. Alternatively, one could model the initialization phase by an additional entity (ITM) which generates the public and private keys (at least for the honest parties) and the index k and then distribute the keys over resilient, read-protected channels to the different entities.

In what follows, whenever we say that, for example, A receives message $\mathsf{sig}[m, v_B]$, we implicitly require that A verifies B's signature. Also, we often simply write $h(m)$ instead of $h_k(m)$, i.e., the key (index) for the hash function is omitted. We assume that $h(m) \in \{0,1\}^l$ for a fixed l.

Specification of the Honest Initiator A The honest initiator A of the ASW protocol when talking to B, using the TTP T, and running the protocol on the contractual text text performs the following steps: We use the naming convention for tapes introduced in Section 4 and Appendix A. We model A in such a way that she non-deterministically decides whether or not to start the protocol, i.e., the scheduler makes this decision. This property is crucial for proving the balance of the protocol.

E.1 If input `init` is received on $\mathtt{sch\_p}_A$ (from the scheduler), then continue with E.2. Otherwise, if input stop is received on $\mathtt{sch\_p}_A$, then stop, i.e., from now on ignore all incoming messages. Otherwise, ignore the incoming message and continue to wait in state E.1.

E.2 Choose $N_A \stackrel{R}{\leftarrow} \{0,1\}^\eta$ and output $m_1 = \mathsf{sig}[\langle v_A, v_B, v_T, \mathsf{text}, h(N_A)\rangle, v_A]$ on $\mathtt{netout}^A_B$; ignore the subsequent acknowledgment on $\mathtt{netack}^A_B$ from $\mathbf{Net}^A_B$ and continue with E.3.

E.3 If a message of the form $m_2 = \mathsf{sig}[\langle m_1, x\rangle, v_B]$ is received on $\mathtt{netin}^B_A$ where $x \in \{0,1\}^l$, then output N_A on $\mathtt{netout}^A_B$; ignore the subsequent acknowledgment on $\mathtt{netack}^A_B$ from $\mathbf{Net}^A_B$ and continue with E.4. Otherwise, if the message `abort` is received on $\mathtt{sch\_p}_A$, then continue with A.1 (abort protocol). Otherwise, ignore the incoming message and continue to wait in E.3

E.4 If a nonce $N \in \{0,1\}^\eta$ is received on $\mathtt{netin}^B_A$ such that $h(N) = x$, then write `Contract` on some work tape and stop. Otherwise, if `resolve` is received on $\mathtt{sch\_p}_A$, then continue with R.1 (resolve protocol). Otherwise, ignore the incoming message and continue to wait in state E.4.

A.1 Send $m_a = \mathsf{sig}[\langle \mathsf{aborted}, m_1\rangle, v_A]$ on $\mathtt{rcout}^A_T$; ignore the subsequent acknowledgment from $\mathbf{RC}^A_T$ and continue with A.2.

A.2 If the message $\mathsf{sig}[\langle \mathsf{aborted}, m_a\rangle, v_T]$ is received on $\mathtt{rcin}^T_A$, then stop. Otherwise, if a message of the form $\mathsf{sig}[\langle m_1, m_2\rangle, v_T]$ is received on $\mathtt{rcin}^T_A$, then write `Contract` on some work tape and stop. Otherwise, ignore the incoming message and continue to wait in state A.2.

R.1 Send $\langle m_1, m_2\rangle$ on $\mathtt{rcout}_T^A$; ignore the subsequent acknowledgment from $\mathbf{RC}_T^A$ and continue with R.2.
R.2 If $\mathsf{sig}[\langle m_1, m_2\rangle, v_T]$ is received on $\mathtt{rcin}_A^T$, then write `Contract` on some work tape and stop. Otherwise, ignore the incoming message and continue to wait in state R.2.

Specification of the Honest Responder B The honest responder B of the ASW protocol when talking to A, using the TTP T, and running the protocol on the contractual text text performs the following steps: Similar to the case of A, when B receives the first message, B makes a non-deterministic decision whether or not to continue the protocol run, i.e., the scheduler makes this decision.

E.1 If a message of the form $m_1 = \mathsf{sig}[\langle v_A, v_B, v_T, \mathsf{text}, x\rangle, v_A]$ for some $x \in \{0,1\}^l$ is received on $\mathtt{netin}_B^A$, then continue with E.2. Otherwise, ignore the incoming message and continue to wait in state E.1.
E.2 If input `start` is received on $\mathtt{sch\_p}_B$ (from the scheduler), then continue with E.3. Otherwise, if input stop is received on $\mathtt{sch\_p}_B$, then stop.[7] Otherwise, ignore the incoming message and continue to wait in state E.2.
E.3 Choose $N_B \xleftarrow{R} \{0,1\}^\eta$ and output $m_2 = \mathsf{sig}[\langle m_1, h(N_B)\rangle, B]$ on $\mathtt{netout}_A^B$; then, ignore the subsequent acknowledgment on $\mathtt{netack}_A^B$ from $\mathbf{Net}_A^B$ and continue with E.4.
E.4 If a nonce $N \in \{0,1\}^\eta$ is received on $\mathtt{netin}_B^A$ such that $h(N) = x$, then write `Contract` on some work tape, output N_B on $\mathtt{netout}_A^B$ (the subsequent acknowledgment can be ignored), and stop. Otherwise, if `resolve` is received on $\mathtt{sch\_p}_B$, then continue with R.1 (resolve protocol). Otherwise, ignore the incoming message and continue to wait in state E.4.
R.1 Send $\langle m_1, m_2\rangle$ on $\mathtt{rcout}_T^B$; ignore the subsequent acknowledgment from $\mathbf{RC}_T^B$ and continue with R.2.
R.2 If $\mathsf{sig}[\langle m_1, m_2\rangle, T]$ is received on $\mathtt{rcin}_B^T$, then write `Contract` on some work tape and stop. Otherwise, ignore the incoming message and continues to wait in state R.2.

Specification of the Honest TTP T The TTP T maintains a database DB of requests received so far. It can interact with the parties in the set $\{P_1, \dots, P_n\}$. Entries of the database are of the form $\langle\langle v_O, v_R, v_T, text\rangle, token\rangle$ where v_O and v_R are public-keys of principals in the mentioned set of principals (the public-key v_T is the public key of T), $text$ is a contractual text, and $token$ is either an abort or a resolve token.

T.0 $\mathsf{DB} := \varepsilon$.
T.1 If input of the form m=$\mathsf{sig}[\langle \mathsf{aborted}, \mathsf{sig}[\langle v_O, v_R, v_T, text, h\rangle, v_O]\rangle, v_O]$ is received on $\mathtt{rcout}_T^O$, then check whether DB contains an entry $\langle\langle v_O, v_R, v_T, text\rangle, token\rangle$ for some $token$. If so, then return $token$ on $\mathtt{rcin}_O^T$; ignore the subsequent acknowledgment from $\mathbf{RC}_O^T$ and continue with T.1. Otherwise, if DB does not contain such an entry, then add the entry $\langle\langle v_O, v_R, v_T, text\rangle, \mathsf{sig}[\langle \mathsf{aborted}, m\rangle, v_T]\rangle$ to DB and output (the abort token) $\mathsf{sig}[\langle \mathsf{aborted}, m\rangle, v_T]$; ignore the subsequent acknowledgment from $\mathbf{RC}_O^T$ and continue with T.1.
Otherwise, if input of the form $m' = \langle m'', \mathsf{sig}[\langle m'', h'\rangle, v_R]\rangle$ for some $m'' = \mathsf{sig}[\langle v_O, v_R, v_T, text, h\rangle, v_O]$ is received on $\mathtt{rcout}_T^O$, then check whether DB contains an entry of the form $\langle\langle v_O, v_R, v_T, text\rangle, token\rangle$ for some $token$. If so, then return $token$ on $\mathtt{rcin}_O^T$; ignore the subsequent acknowledgment from $\mathbf{RC}_O^T$ and continue with T.1. Otherwise, if DB does not contain such an entry, then add the entry $\langle\langle v_O, v_R, v_T, text\rangle, \mathsf{sig}[m', v_T]\rangle$ to the DB and output (the resolve token) $\mathsf{sig}[m', v_T]$; ignore the subsequent acknowledgment from $\mathbf{RC}_O^T$ and continue with T.1.

The model of the TTP as just described corresponds to the TTP as specified by Asokan et al. [1], although Asokan et al. only specify the TTP for handling one session of the protocol, and therefore, they do not specify how the database the TTP has to maintain looks like.

[7] Alternatively, B could go back to E.1.

As pointed out by Chadha et al. [11] for the GJM protocol [14] (and the same it true for the ASW protocol), in case multiple runs of the protocol (with the same contractual partners and on the same contractual text) are carried out, a session identifiers is needed that uniquely identifies a session. Such identifiers need to be part of the entries that the TTP stores in the database. Without such identifiers the ASW protocol would neither be fair nor balanced: Consider the situation of an honest initiator O and a dishonest responder R. If, in a first session of the protocol, R does not respond to the first message O sends, then O sends an abort request to the TTP. If later, in a second session of the protocol, O agrees to run the protocol again with R (on the same contractual text), and the session gets to a point where O sent the nonce, then R has a valid contract from O. If at this point R doesn't return his nonce, then O cannot get a valid contract since when contacting the TTP, O would get back the abort token from the previous session. Hence, this state of the protocol is unfair and unbalanced for O. We note that including in the database the digest $h(N_O)$ computed by O would still not solve the problem: It is not hard to see that if O is dishonest, the protocol would be unbalanced for R.

B.2 Multi-Session Specification

In this Section B.2, we specify a principal willing to run multiple sessions of the ASW protocol. As pointed out at the end of Section B.1, in this case every session should have a unique session identifier. Such an identifier can easily be established by the initiator and responder: At the beginning of the session, both parties contribute to one part of the identifier. Even if one party is dishonest, if the honest party ensures that the part of the identifier that he/she contributes is different from the parts contributed in other sessions, then the combined identifier will be unique. We note that the TTP as defined in Section B.1 can already deal with an unbounded number of requests, which possibly come from different sessions. However, now the messages the TTP receives should include the session identifiers and these session identifiers will be part of the database entries.

Now, let us turn to the principal, say A, willing to run multiple sessions of the protocol. We will model principal A in such a way that she can run an unbounded number of sessions with B, using the TTP T, on the contractual text text. However, A will only start another session with B if the previous session has been aborted. This models the realistic situation that even though A has aborted the protocol, she might be convinced by B to start another session because, for example, technical problems prevented B from sending his willingness to sign the contract in time. More precisely, A first receives a request from B asking whether A wants to take part in a protocol run. If she is currently running a session with B, she ignores such requests. If she is not running a session with B she can non-deterministically decide whether or not to start a (new) session with B, i.e., the scheduler makes this decision. Before starting a new session, A and B establish an identifier for the session which is required to be unique for all sessions and which will be part of the messages signed. As explained above, the uniqueness of the identifier is easily guaranteed if both parties contribute to the identifier. The formal specification follows:

E.0 $counter := 0$.[8]

E.1 If input of the form $\langle \mathsf{request}, id_B \rangle$ is received on $\mathtt{netin}_A^B$, then continue with E.2. Otherwise, ignore the incoming message and continue to wait in state E.1.

E.2 If input $\mathtt{start}$ is received on $\mathtt{sch\_p}_A$ (from the scheduler), then continue with E.3. Otherwise, if input $\mathtt{stop}$ is received on $\mathtt{sch\_p}_A$, then stop, i.e., from now on ignore all incoming messages.[9] Otherwise, ignore the incoming message and continue to wait in state E.2.

[8] Instead of a counter, one could define A to choose a random number bit string in $\{0,1\}^\eta$.

[9] Instead of stopping for ever, one could alternatively go back to E.1.

E.3 Set $counter := counter + 1$. Next, chose a random nonce $N_A \xleftarrow{R} \{0,1\}^\eta$ and output the message $m = \mathsf{sig}[\langle A, B, T, \mathsf{text}, h_k(N_A), \langle\langle A, counter\rangle, id_B\rangle\rangle, A]$ on $\texttt{netout}_B^A$ (see Section 2); ignore the subsequent acknowledgment on $\texttt{netack}_B^A$ from $\mathbf{Net}_B^A$ and continue with E.4.

E.4 If a message of the form $\mathsf{sig}[\langle m, x\rangle, B]$ is received on $\texttt{netin}_A^B$ where $x \in \{0,1\}^l$, then output N_A on $\texttt{netout}_B^A$; ignore the subsequent acknowledgment on $\texttt{netack}_B^A$ from $\mathbf{Net}_B^A$ and continue with E.5. Otherwise, if the message $\mathsf{aborted}$ is received on $\texttt{sch\_p}_A$, then continue with A.1 (abort protocol). Otherwise, ignore the incoming message and continue to wait in E.4

E.5 If a nonce $N \in \{0,1\}^\eta$ is received on $\texttt{netin}_A^B$ such that $h_k(N) = x$, then write the message $\texttt{Contract}$ on some work tape and stop. Otherwise, if $\texttt{resolve}$ is received on $\texttt{sch\_p}_A$, then continue with R.1 (resolve protocol). Otherwise, ignore the incoming message and continue to wait in state E.5.

A.1 Send $m' = \mathsf{sig}[\langle \mathsf{aborted}, m\rangle, A]$ on $\texttt{rcout}_T^A$; ignore the subsequent acknowledgment from $\mathbf{RC}_T^A$ and continue with A.2.

A.2 If the message $\mathsf{sig}[\langle \mathsf{aborted}, m'\rangle, T]$ is received on $\texttt{rcin}_A^T$, then continue with E.1. Otherwise, if a message of the form $\mathsf{sig}[\langle m, \mathsf{sig}[\langle m, x\rangle, B]\rangle, T]$ is received on $\texttt{rcin}_A^T$, then write the message $\texttt{Contract}$ on some work tape and stop. Otherwise, ignore the incoming message and continue to wait in state A.2.

R.1 Send $\langle m, \mathsf{sig}[\langle m, x\rangle, B]\rangle$ on $\texttt{rcout}_T^A$; ignore the subsequent acknowledgment from $\mathbf{RC}_T^A$ and continue with R.2.

R.2 If $\mathsf{sig}[\langle m, \mathsf{sig}[\langle m, x\rangle, B]\rangle, T]$ is received on $\texttt{rcin}_A^T$, then write $\texttt{Contract}$ on some work tape and stop. Otherwise, ignore the incoming message and continue to wait in state R.2.

In a similar way, a responder running multiple sessions could be specified. It is also straightforward (but tedious) to model principals that run multiple sessions with different principals, on different contractual texts, and in different roles at the same time (and these parameter could be determined by the adversary). In particular, one could model principals in such a way that they run several instances of the ASW protocol with the same contractual partner and the same TTP on the same contractual text at the same time. This may or may not be realistic. Also, one could add to the protocol specification a signing oracle (formally modeled as an honest party in the protocol specification, similar to the watch dog) which allows the adversary to generate signatures of the honest party, e.g., A, on messages of his choice, subject to certain restrictions, as otherwise the adversary could simply simulate A.

C Proof of Theorem 2

PROOF. We denote the protocol machines of Π by $\mathbf{H}_i$ for $i \in \mathcal{H}$.

Since $\mathbf{A}$ is fairness-enabling, we know that there exists a polynomial $p_{\mathbf{A}}(\eta)$ such that the number of configuration changes of $\mathbf{A}$ in a run of $\mathcal{S}_D(\eta)$ with $\mathcal{S}(\Pi, \mathbf{A}, \mathbf{S}')$ is bounded by $p_{\mathbf{A}}(\eta)$ (with overwhelming probability) for any scheduler $\mathbf{S}'$. Also, since Π is fairness-enabling we know that there exists a polynomial $p_\Pi(\eta)$ such that the number of configuration changes of every protocol machine of Π in a run of $\mathcal{S}_D(\eta)$ is bounded by $p_\Pi(\eta)$. Hence, given Π and $\mathbf{A}$ there exists a polynomial $p(\eta)$ such that the overall number of configuration changes of $\mathbf{A}$ and the protocol machines of Π in a run of $\mathcal{S}_D(\eta)$ is bounded by $p(\eta)$ (with overwhelming probability).

Since, by definition, ITMs can only output messages if they change their local configuration, we know that the number of messages written on output tapes by $\mathbf{A}$ and the ITMs in Π is bounded by $p(\eta)$ (with overwhelming probability). In particular, the overall number of messages sent to resilient channels is bounded by $p(\eta)$ (with overwhelming probability). Since D is polynomially samplable, the number of messages initially stored in resilient channel buffers is also bounded by some polynomial $p'(\eta)$. Hence, with overwhelming probability, not more than $p''(\eta) = p(\eta) + p'(\eta)$ messages are stored in a resilient channel at any point in a run.

By definition, protocol and adversary machines only accept a fixed and finite set of messages from a scheduler. For $\mathbf{A}$ the set is the singleton $\mathcal{M}_{\mathbf{A}} = \{\texttt{schedule}\}$. Let $\mathcal{M}_i$ denote the (finite) set of messages that $\mathbf{H}_i$ accepts from a scheduler.

We are now ready to define a scheduler **S** and then show that it is fair for $\mathcal{S}(\Pi, \mathbf{A})$ and D. The scheduler **S** works in rounds. Every round consists of two phases. In the first phase of a round, **S** does the following: First, **S** sends `schedule` to **A**. When activated again, **S** sends for every $i \in \mathcal{H}$ and every $m \in \mathcal{M}_i$, the message m to $\mathbf{H}_i$. Note that after sending one message m, **S** has to wait to be scheduled again before another message can be sent. Once the first phase of a round is completed, **S** sends, in the second phase of the round, to every resilient channel the index 1 $p''(\eta)$ times. (Again, after every activation of a resilient channel, **S** has to wait to be activated again.) Note that since, as explained above, the number of messages in a resilient channel is bounded by $p''(\eta)$, after the second phase of a round, the resilient channels are guaranteed to be empty. We define **S** to perform $p(\eta)$ rounds. We now argue that **S** is fair.

We first observe that if in one round no ITM in $\mathcal{S}(\Pi, \mathbf{A})$ has changed its local configuration, then these ITMs will also not change their configuration in subsequent rounds.

For the ITMs in $\mathcal{S}(\Pi)$ this is obvious: If such a machine is activated by the scheduler again, the machine will perform exactly the same computation as before and will as before return to the local configuration at the beginning of the activation. Note that the head position on the random tape is part of the local configuration, and hence, the random coins used in these activations do not change.

For **A**, we use that if the ITMs in $\mathcal{S}(\Pi, \mathbf{A})$ do not change their local configuration, then the adversary view oracle of **A** will return the same view when invoked by **A**. As a result, **A**'s computation will be unchanged. In particular, the local configuration of **A** at the beginning and at the end of the activation will be the same.

As explained above, after every round the resilient channels are empty, and hence, they do not change their configuration in the next round unless they receive a new message in the next round. Hence, if the adversary and the protocol machines do not change their local configuration in one round (and hence, do not produce output), then no machine in $\mathcal{S}(\Pi, \mathbf{A})$ will change its local configuration again. Since the adversary and the protocol machines can only change their local configurations at most $p(\eta)$ times (only with negligible probability they can change their local configurations more often) it follows that after $p(\eta)$ rounds, no ITM in $\mathcal{S}(\Pi, \mathbf{A})$ will change its local configuration again (only with negligible probability). Thus, since **S** performs $p(\eta)$ rounds, it follows that **S** is fair. $\square$

D The ASW Protocol is Fair

We provide a proof sketch of Theorem 3. The case of an honest initiator is restated in Proposition 1 and the case of an honest responder in Proposition 2.

Proposition 1. *$\Pi^{\mathsf{ASW\text{-}A}}$ is strongly fair w.r.t.* `checkfair` *and view oracles $\mathcal{O}^{\mathsf{ASW}}_{adv}$ and $\mathcal{O}^{\mathsf{ASW}}_{sch}$.*

We prove Proposition 1 by contradiction. Assume that there exist an adversary A and a fair scheduler **S** such that $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{S}, \texttt{checkfair}) = 0$ with non negligible probability. There are three cases.

1. Either the agent A has not sent her first message $\mathsf{sig}[m_A, A]$,
2. Or A has sent $\mathsf{sig}[m_A, A]$ but has not received any valid answer from the adversary,
3. Or A has sent $\mathsf{sig}[m_A, A]$ and has received a valid answer from the adversary.

At least one of the three cases must happen with non negligible probability. For each case in turn, we show how to turn an adversary **A** that wins against $\Pi^{\mathsf{ASW\text{-}A}}$ into adversaries against that break the primitives used in $\Pi^{\mathsf{ASW\text{-}A}}$.

Case 1 The agent A has not sent her first message $\mathsf{sig}[m_A, A]$ thus A cannot have the contract. Since `checkfair`=0 it follows that the adversary succeeded in getting a valid contract of the form $\langle \mathsf{sig}[m_A, A], N_A, \mathsf{sig}[m_B, B], N_B\rangle$ or of the form $\mathsf{sig}[\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B]\rangle, T]$. Either the adversary did not make any valid query to the trusted party (for the instance of the protocol under consideration), in which case, it means that he forged a valid signature of A or T. Or the adversary made a valid query to the trusted party, which means that he sent a message of the form $\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B]\rangle$ to T. Thus the adversary has forged a valid signature of A. In both cases, the adversary must have forged a valid signature of an honest agent.

The above intuition can be easily transformed into a reduction from the security of the protocol to that of the underlying signature scheme. Given an adversary **A** (that plays the role of party B) we construct an adversary $\mathbf{A}_{\mathsf{DS}}$ against the signature scheme DS. Recall that $\mathbf{A}_{\mathsf{DS}}$ has access to a signing oracle $\mathcal{O}_{\mathsf{DS}}(sk, \cdot)$ and takes as input the verification key pk that corresponds to sk. Adversary $\mathbf{A}_{\mathsf{DS}}$ simulates the experiment $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{S}, \texttt{checkfair})$. It uses **A** as a subroutine and it simulates the environment of **A**, i.e. it simulates parties A and T, as well as the execution of **S**. Moreover, it also simulates oracles $\mathcal{O}^{\mathsf{ASW}}_{sch}$ and $\mathcal{O}^{\mathsf{ASW}}_{adv}$. In particular, $\mathbf{A}_{\mathsf{DS}}$ sets the public key of A to pk. The adversary $\mathbf{A}_{\mathsf{DS}}$ starts the execution of **A** and answers all its queries (essentially only queries to $\mathcal{O}^{\mathsf{ASW}}_{adv}$.) Adversary $\mathbf{A}_{\mathsf{DS}}$ tracks the messages sent to the "watch dog" and verifies if any of the messages is a valid contract which (by definition) contains a valid signature on some message, with respect to pk. Since $\mathbf{A}_{\mathsf{DS}}$ did not make any queries to its signing oracle, such a signature is a valid forgery, and thus $\mathbf{A}_{\mathsf{DS}}$ breaks the security of DS. We note that $\mathbf{A}_{\mathsf{DS}}$ can simulate any adversary oracle, as long as it does not need the secret key of A to do so (i.e. the simulation works virtually for all reasonable adversaries).

Case 2 The agent A has sent $\mathsf{sig}[m_A, A]$ but has not received any valid answer from the adversary. Since **S** is fair, A must have contacted the trusted party, asking for aborting. Either T sent a valid contract in return, which means the adversary has sent a valid resolve request to **S** and thus has received a valid contract, in which case $\texttt{checkfair} = 1$. Or T replied with an abort message to A so, A does not have the contract. Since $\texttt{checkfair} = 0$ it must be the case that the adversary succeeded in obtaining a valid contract. We distinguish two cases, depending on the form of the contract.

First, assume that the contract is of the form $\mathsf{sig}[\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B]\rangle, T]$. Since **A** should not be able to obtain a contract, it must be the case that **A** did not send a resolve request to T, and therefore the contract must have been obtained by forging a signature of T.

Under these circumstances, we show how construct an adversary $\mathbf{A}_{\mathsf{DS}}$ against DS. Adversary $\mathbf{A}_{\mathsf{DS}}$ simulates the experiment $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{S}, \texttt{checkfair})$. In the execution, **A** plays the roles of parties A and T. It generates a pair of signing verification keys for A, via $(sk_A, pk_A) \xleftarrow{R} \mathcal{K}(\eta)$, it generates a key k for the hash function via $k \xleftarrow{R} hkg(\eta)$, and then sets pk as the public key of T. It then simulates the execution of the experiment answering the adversary's queries to oracle $\mathcal{O}^{\mathsf{ASW}}_{sch}$ (using the parameters set as above). When **S** schedules A to output its first message, $\mathbf{A}_{\mathsf{DS}}$ generates a nonce N_A and computes a signature $\sigma = \mathcal{S}(sk_A, m_A)$ on message $m_A = (A, B, T, \mathsf{text}, h_k(N_A))$ and sends (m_A, σ) to the adversary. Since we are in the case when **A** obtains a replacement contract, it must be the case that the replacement contract was created by **A** itself (without involving T, since otherwise when A contacts T (this event happens because **S** is fair)), then A would also obtain a replacement contract). Since $\mathbf{A}_{\mathsf{DS}}$ never makes a query to its signing oracle, the replacement contract output by **A** is in fact a successful forgery against DS.

In the case that **A** obtains a standard contract $\langle \mathsf{sig}[m'_A, A], N'_A, \mathsf{sig}[m_B, B], N_B \rangle$, we further distinguish two cases, depending on whether $h_k(N'_A) = h_k(N_A)$ or not.

First we construct an adversary $\mathbf{A}_{\mathcal{H}}$ against $\mathcal{H}$ which is successful if $h_k(N'_A) = h_k(N_A)$ (here N_A is the nonce that A sends in its first message and N'_A is the nonce in the contract that $\mathbf{A}'$ obtains.) As before, the adversary $\mathbf{A}_{\mathcal{H}}$ simulates the execution of $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{S}, \mathtt{checkfair})$: it plays the role of both A and T (in particular generates signing/verification keys for both) and simulates the execution of **S**. Recall that $\mathbf{A}_{\mathcal{H}}$ takes as input a key k and a hash value $y \leftarrow h_k(x)$ for some $x \stackrel{R}{\leftarrow}\in \{0,1\}^\eta$. The key of the hash function is set to k. When A has to output its first message, $\mathbf{A}_{\mathcal{H}}$ composes message $m_A = (A, B, T, \mathsf{text}, y)$, computes a signature σ on m_A, and sends (m_A, σ) to the adversary. When the adversary outputs a contract $\langle \mathsf{sig}[m'_A, A], N'_A, \mathsf{sig}[m_B, B], N_B \rangle$ such that $h_k(N'_A) = h_k(N_A)$, adversary $\mathbf{A}_{\mathcal{H}}$ outputs N'_A as a forgery. Notice that if **A** is successful in $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{S}, \mathtt{checkfair})$ (and the contract that **A** outputs is as above) then $\mathbf{A}_{\mathcal{H}}$ outputs a preimage of y with non-negligible probability.

Finally, if $h_k(N'_A) \neq h_k(N_A)$ we show how to construct an adversary $\mathbf{A}_{\mathsf{DS}}$ against signature scheme DS. Adversary $\mathbf{A}_{\mathsf{DS}}$ simulates the execution of the experiment $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{S}, \mathtt{checkfair})$ where it simulates the parties A and T. In particular, it generates signing/verification keys for T, and the key k for the hash function. The public key of A is set to pk (the key that $\mathbf{A}_{\mathsf{DS}}$ has as input). When A has to output its first message, $\mathbf{A}_{\mathsf{DS}}$ selects $N_A \stackrel{R}{\leftarrow} \{0,1\}^\eta$, computes the message $m_A = (A, B, T, \mathsf{text}, h_k(N_A))$, sends m_A to its signing oracle $\mathcal{O}_{\mathsf{DS}}(sk, \cdot)$ and receives a signature σ on m_A. It sends (m_A, σ) to **A**. It the continues execution, answering all queries that **A** may make to oracle $\mathcal{O}^{\mathsf{ASW}}_{adv}$, until **A** outputs the forgery contract $\langle \mathsf{sig}[m'_A, A], N'_A, \mathsf{sig}[m_B, B], N_B \rangle$. At this point, $\mathbf{A}_{\mathsf{DS}}$ outputs $(m'_A, \mathsf{sig}[m'_A, A])$ as its attempted forgery. Since the contract is valid, $\mathsf{sig}[m'_A, A]$ is a valid signature on m'_A. Moreover, since $h_k(N_A) \neq h_k(N'_A)$, it follows that the message $m'_A \neq m_A$, and therefore m'_A was not queried by $\mathbf{A}_{\mathsf{DS}}$ to its signing oracle. We conclude that the forgery that $\mathbf{A}_{\mathsf{DS}}$ outputs is valid.

Case 3 The agent A has sent $\mathsf{sig}[m_A, A]$ and has received a valid answer from the adversary. Since A is honest, she must have sent her nonce N_a to the adversary thus the adversary has the contract. Either the adversary sent his nonce N_b to A, which means that A has also a valid contract or A did not get any valid answer from the adversary. Since **S** is fair, A must have contacted the trusted party, asking for resolving. If the trusted party did not receive previously a valid abort request, he must have returned a valid contract to A. Otherwise (if T did receive a valid abort request), the adversary must have sent of message of the form $\mathsf{sig}[\mathsf{aborted}, \mathsf{sig}[m_A, A], A]$ to T on the secure channel between A and T (which is impossible in our model).

Conversely, the ASW protocol is fair to B, the second participant.

Proposition 2. *$\Pi^{\mathsf{ASW\text{-}B}}$ is strongly fair w.r.t.* `checkfair` *and view oracles $\mathcal{O}^{\mathsf{ASW\text{-}B}}_{adv}$ and $\mathcal{O}^{\mathsf{ASW\text{-}B}}_{sch}$.*

The proof is quite similar to the previous one. Assume that there exist an adversary **A** and a fair scheduler **S** such that $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}B}}, \mathbf{A}, \mathbf{S}, \mathtt{checkfair}) = 0$ with non negligible probability.

There are four cases.

1. Either the agent B has not received any valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary,
2. The agent B has or has not received a valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary but he has chosen not to answer
3. The agent B has received a valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary and he has sent his promise to sign $\mathsf{sig}[m_B, B]$ but he did not get a valid answer from the adversary,
4. The agent B has received a valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary, he has sent his promise to sign $\mathsf{sig}[m_B, B]$ and he got a valid answer from the adversary.

At least one of the following four cases must happen with non-negligible probability. Here, we describe how a successful forgery would relate to the security of the underlying primitives of the protocol. Security reductions similar to those for Proposition 1 can be easily constructed.

Cases 1 and 2 are similar. In both cases, B has not sent his promise to sign $\mathsf{sig}[m_B, B]$. Since $\texttt{checkfair} = 0$ it follows that the adversary obtained a valid contract of the form $\langle \mathsf{sig}[m_A, A], N_A, \mathsf{sig}[m_B, B], N_B\rangle$ or $\mathsf{sig}[\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B]\rangle, T]$. Either the adversary did not make any valid query to the trusted party, in which case, it means that he forged a valid signature of B or T. Or the adversary made a valid query to the trusted party, which means that he sent a message of the form $\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B]\rangle$ to T. Thus the adversary has forged a valid signature of B. In both cases, the adversary must have forged a valid signature of an honest agent.

Case 3 The agent B has received a valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary and he has sent his promise to sign $\mathsf{sig}[m_B, B]$ but he did not get a valid answer from the adversary. Since $\mathbf{S}$ is fair, B must have contacted the TTP, asking for resolving. Either T sent a valid contract in return, in which case $\texttt{checkfair} = 1$. Or T sent an abort message to B thus B does not have the contract. Since $\texttt{checkfair} = 0$, it must be the case that the adversary obtained a contract of the form $\langle \mathsf{sig}[m_A, A], N_A, \mathsf{sig}[m_B, B], N_B\rangle$ or $\mathsf{sig}[\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B]\rangle, T]$. Since B get an abort message, this means that T did not sent a valid contract to the adversary even if he sent valid resolve requests afterward. Thus the adversary must have forged a valid signature of an honest agent or computed N_B out of $\mathsf{sig}[m_B, B]$.

Case 4 The agent B obtained a valid contract. However, this can not be the case since $\texttt{checkfair} = 1$.

E Proving the ASW Protocol to be Balanced

We provide a proof sketch of Theorem 4. The case of an honest initiator is restated in Proposition 3 and the case of an honest responder in Proposition 4.

Proposition 3. *$\Pi^{\mathsf{ASW\text{-}A}}$ is balanced w.r.t.* $\texttt{goal}_1$, $\texttt{goal}_2$, $\mathcal{O}^{\mathsf{ASW}}_{adv}$, $\mathcal{O}^{\mathsf{ASW}}_{adv'}$, $\mathcal{O}^{\mathsf{ASW}}_{sch}$, *and* $\mathcal{O}^{\mathsf{ASW}}_{sch'}$.

Proof (sketch): The proof is done by contradiction. Assume there exist restricted adversary machines $\mathbf{A} = \mathbf{A}(\mathcal{O}^{\mathsf{ASW}}_{adv})$ and $\mathbf{A}' = \mathbf{A}'(\mathcal{O}^{\mathsf{ASW}}_{adv'})$ for $\Pi^{\mathsf{ASW\text{-}A}}$ and a scheduler $\mathbf{S} = \mathbf{S}(\mathcal{O}^{\mathsf{ASW}}_{sch})$ for $\Pi^{\mathsf{ASW\text{-}A}}$ and $\mathbf{A}$ such that, for any fair scheduler $\mathbf{S}' = \mathbf{S}'(\mathcal{O}^{\mathsf{ASW}}_{sch'})$ for $\Pi^{\mathsf{ASW\text{-}A}}$ and $\mathbf{A}'$ and for any challenge function $\texttt{challenge}$, it is the case that $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{A}', \mathbf{S}, \mathbf{S}', \texttt{goal}_1, \texttt{goal}_2, \texttt{challenge}) = 1$ with non negligible probability.

Let q be such that $\mathcal{S}(\eta) \rightsquigarrow q$ where $\mathcal{S} = \mathcal{S}(\Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{S})$. We distinguish several cases for q:

1. Either the agent A has not sent her first message $\mathsf{sig}[m_A, A]$,
2. Or A has sent $\mathsf{sig}[m_A, A]$ but has not received any valid answer from the adversary $\mathbf{A}$ and the TTP has answered a valid resolve query from the adversary for this contract,
3. Or A has sent $\mathsf{sig}[m_A, A]$ but has not received any valid answer from the adversary $\mathbf{A}$ and the TTP has not answered a valid resolve query from the adversary for this contract,
4. Or A has sent $\mathsf{sig}[m_A, A]$ and has received a valid answer from the adversary $\mathbf{A}$ (thus she has sent her nonce N_A) but did not get the last message from the adversary,
5. Or A has finished her protocol, that is, she got the last message from $\mathbf{A}$.

At least one of the five cases must happen with non negligible probability.

Case 1 The agent A has not sent her first message $\mathsf{sig}[m_A, A]$. For all states q as above this case, we set $\texttt{challenge}(q) = 2$, and show that there exists a fair scheduler $\mathbf{S}'$ for $\Pi^{\mathsf{ASW\text{-}A}}$ such that adversary $\mathbf{A}'$ cannot achieve $\texttt{goal}_2$.

Consider the scheduler $\mathbf{S}'$ that causes A to stop the execution of the protocol before sending its first message to $\mathbf{A}'$ (recall that stopping the execution is one of the valid actions that a protocol participant can take at any point). Since $\mathbf{A}'$ achieves $\mathtt{goal}_2$, it follows that there are three possibilities regarding how $\mathbf{A}'$ to obtains a valid contract. The first two are to compute by itself (with no interaction with T) a contract $\langle \mathsf{sig}[m_A, A], N_A, \mathsf{sig}[m_B, B], N_B \rangle$, or a contract $\mathsf{sig}[\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B]\rangle, T]$ (which would imply forging a signature of A), or to contact the TTP with a message of the form $\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B]\rangle$ (which would also imply that A' has forged a signature of A).

As we did in the case of fairness, we turn this intuition into a reduction. Assume that there exists adversaries $\mathbf{A}$ and $\mathbf{A}'$, scheduler $\mathbf{S}$ such that for all schedulers $\mathbf{S}'$ adversary $\mathbf{A}'$ achieves goal $\mathtt{goal}_2$ from state q (where $\mathcal{S}(\eta) \rightsquigarrow q$).

We show how to use the above adversaries and schedulers, together with the scheduler $\mathbf{S}'$ described also above in order to build an adversary $\mathbf{A}_{\mathsf{DS}}$ against DS.

Adversary $\mathbf{A}_{\mathsf{DS}}$ simulates $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{A}', \mathbf{S}, \mathbf{S}', \mathtt{goal}_1, \mathtt{goal}_2, \mathtt{challenge})$. In the execution $\mathbf{A}$ plays the role of parties A and T. In particular it generates keys for signing/verifying for party T, and sets the public verification key of A to pk (the key that $\mathbf{A}_{\mathsf{DS}}$ takes as input). It keeps tracks of the global state of the system, until simulator $\mathbf{S}$ finishes its execution. If in the resulting state q party A had already sent its first message the adversary $\mathbf{A}_{\mathsf{DS}}$ aborts. Otherwise, it continues its simulation with scheduler $\mathbf{S}'$. It answers all queries that $\mathbf{A}'$ makes to its oracle (this is possible since $\mathbf{A}_{\mathsf{DS}}$ knows the local states of all parties, except the signing key that corresponds to pk). It answers the queries that $\mathbf{A}'$ makes to T. If $\mathbf{A}'$ sends a valid contract to the "watch dog", $\mathbf{A}_{\mathsf{DS}}$ extracts an appropriate forgery (the part of the contract that consists of a valid signature of A) and outputs it as its own attempted forgery. Since $\mathbf{A}_{\mathsf{DS}}$ does not make any oracle requests to its own oracle, any valid signature with respect to pk is a valid forgery, and therefore $\mathbf{A}_{\mathsf{DS}}$ wins.

Case 2 The agent A has sent $\mathsf{sig}[m_A, A]$ but has not received any valid answer from $\mathbf{A}'$ and the TTP has answered a valid resolve query from $\mathbf{A}'$ for this contract. Let us show that the adversary $\mathbf{A}'$ cannot achieve $\mathtt{goal}_1$ (A does not have a contract). In particular, consider the scheduler $\mathbf{S}'$ that schedules A such that she immediately contacts the TTP, with the abort request $\mathsf{sig}[\mathsf{aborted}, \mathsf{sig}[m_A, A], A]$. Since T has already answered a resolve request from $\mathbf{A}'$ for this contract, A would receive a valid contract from the TTP thus $\mathtt{goal}_1$ is not achieved.

Case 3 The agent A has sent $\mathsf{sig}[m_A, A]$ but has not received any valid answer from the adversary $\mathbf{A}'$ and the TTP has not received a valid resolve query from the adversary that corresponds to this message of A. For such a state q, we set $\mathtt{challenge}(q) = 2$ and show that there exists a scheduler $\mathbf{S}'$ such that adversary $\mathbf{A}'$ cannot achieve $\mathtt{goal}_2$ ("having a valid contract"). Consider the schedule $\mathbf{S}'$ that as soon as $\mathbf{A}$ outputs its first message (i.e. $\mathsf{sig}[m_A, A]$) schedules A to send an abort request $\mathsf{sig}[\mathsf{aborted}, \mathsf{sig}[m_A, A], A]$ to TTP T.

By assumption T has not received a resolve request from the adversary so T would send a valid abort message to A. Since we only consider fair schedulers, this abort message would eventually be delivered to A so no party would receive a valid contact from T. We now distinguish two different cases depending on the contract that $\mathbf{A}'$ obtains.

If the contract is a replacement contract, $\mathsf{sig}[\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B]\rangle, T]$, then $\mathbf{A}'$ managed to forge a signature of T.

We turn this intuition into a proof by reduction. We construct the following adversary $\mathbf{A}_{\mathcal{H}}$ against the hash function family. As before, adversary $\mathbf{A}_{\mathsf{DS}}$ simulates $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{A}', \mathbf{S}, \mathbf{S}', \mathtt{goal}_1, \mathtt{goal}_2,$ $\mathtt{challenge})$, and in the execution $\mathbf{A}_{\mathsf{DS}}$ uses $\mathbf{A}, \mathbf{A}', \mathbf{S}$ as subroutines. Here, $\mathbf{A}_{\mathsf{DS}}$ plays the roles of both parties A and T. It generates a pair of signing verification keys for A, via $(sk_A, pk_A) \stackrel{R}{\leftarrow} \mathcal{K}(\eta)$, it generates a key k for the hash function via $k \xleftarrow{R} hkg(\eta)$, and then sets pk as the public key of T (recall that pk is the key that $\mathbf{A}_{\mathsf{DS}}$ receives as input.)

It then simulates the execution of the experiment answering the adversary's queries to oracle $\mathcal{O}_{sch}^{\mathsf{ASW}}$ (using the parameters set as above). When $\mathbf{S}$ schedules A to output its first message, $\mathbf{A}_{\mathsf{DS}}$ generates a nonce N_A and computes a signature $\sigma = \mathcal{S}(sk_A, m_A)$ on message $m_A = (A, B, T, \mathsf{text}, h_k(N_A))$ and sends (m_A, σ) to the adversary. At this point the scheduler $\mathbf{S}$ is changed with scheduler $\mathbf{S}'$ (which schedules A to contact T with an abort message, and carries out the abort protocol). Since we are in the case when $\mathbf{A}$ obtains a replacement contract, it must be the case that the replacement contract was created by $\mathbf{A}$ itself (since T would not provide one due to the abort message described above). The replacement contract is a valid signature with respect to pk. Furthermore, $\mathbf{A}_{\mathsf{DS}}$ never makes a query to its signing oracle, the replacement contract output by $\mathbf{A}$ is therefore a successful forgery against DS.

If $\mathbf{A}$ obtains a standard contract $\langle \mathsf{sig}[m'_A, A], N'_A, \mathsf{sig}[m_B, B], N_B\rangle$, we further distinguish two cases, depending on whether $h_k(N'_A) = h_k(N_A)$ or not (here N_A is the nonce that A sends in its first message and N'_A is the nonce in the contract that $\mathbf{A}'$ obtains).

Assume for now that $h_k(N'_A) = h_k(N_A)$ Therefore, the only way for the adversary $\mathbf{A}'$ to obtain a valid contract of the form $\langle \mathsf{sig}[m'_A, A], N'_A, \mathsf{sig}[m_B, B], N_B\rangle$ or $\mathsf{sig}[\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B]\rangle, T]$ is to compute N_A in the first case or compute a valid signature of T in the second case.

Adversary $\mathbf{A}_{\mathcal{H}}$ simulates $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{A}', \mathbf{S}, \mathbf{S}', \mathtt{goal}_1, \mathtt{goal}_2, \mathtt{challenge})$ and it uses adversaries $\mathbf{A}, \mathbf{A}'$ soubroutines. In the simulation, $\mathbf{A}_{\mathcal{H}}$ plays the roles of both A and T (in particular generates signing/verification keys for both). Recall that $\mathbf{A}_{\mathcal{H}}$ takes as input a key k and a hash value $y \leftarrow h_k(x)$ for some $x \xleftarrow{R}\in \{0,1\}^\eta$. The key of the hash function is set to k. Adversary $\mathbf{A}_{\mathcal{H}}$ uses in its simulation the scheduler $\mathbf{S}$ up to the point when A sends its first message to the adversary $\mathbf{A}$. It then starts using the scheduler $\mathbf{S}'$, which as before, directs A to request T to abort the protocol. Next, it continues the execution up to the point when the adversary outputs a contract $\langle \mathsf{sig}[m'_A, A], N'_A, \mathsf{sig}[m_B, B], N_B\rangle$ such that $h_k(N'_A) = h_k(N_A)$, adversary $\mathbf{A}_{\mathcal{H}}$ outputs N'_A as a forgery. Notice that if $\mathbf{A}$ is successful in $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{S}, \mathtt{checkfair})$ (and the contract that $\mathbf{A}$ outputs is as above) then $\mathbf{A}_{\mathcal{H}}$ outputs a preimage of y with non-negligible probability.

Finally, if the adversary outputs a contract such that $h_k(N'_A) \neq h_k(N_A)$, then we construct an adversary against DS.

Adversary $\mathbf{A}_{\mathsf{DS}}$ uses adversaries $\mathbf{A}, \mathbf{A}'$ and the scheduler $\mathbf{S}$ as subroutines by simulating the experiment $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}A}}, \mathbf{A}, \mathbf{A}', \mathbf{S}, \mathbf{S}', \mathtt{goal}_1, \mathtt{goal}_2, \mathtt{challenge})$ as above, and simulates parties A and T. The public key of A is set to pk (the key that $\mathbf{A}$ receives as input). When A has to output its first message, $\mathbf{A}_{\mathsf{DS}}$ selects $N_A \xleftarrow{R} \{0,1\}^\eta$, computes the message $m_A = (A, B, T, \mathsf{text}, h_k(N_A))$, sends m_A to its signing oracle $\mathcal{O}_{\mathsf{DS}}(sk, \cdot)$ and receives a signature σ on m_A. It sends (m_A, σ) to $\mathbf{A}$. At this point, it switches the scheduler to $\mathbf{S}'$ (which directs A to carry out the abort protocol with T). In the remainder of the execution, $\mathbf{A}_{\mathsf{DS}}$ answers all queries that $\mathbf{A}$ may make to oracle $\mathcal{O}_{adv}^{\mathsf{ASW}}$, until $\mathbf{A}$ outputs the forgery contract $\langle \mathsf{sig}[m'_A, A], N'_A, \mathsf{sig}[m_B, B], N_B\rangle$. At this point, $\mathbf{A}_{\mathsf{DS}}$ outputs $(m'_A, \mathsf{sig}[m'_A, A])$ as its attempted forgery. Since the contract is valid, $\mathsf{sig}[m'_A, A]$ is a valid signature on m'_A. Moreover, since $h_k(N_A) \neq h_k(N'_A)$, it follows that the message $m'_A \neq m_A$, and therefore m'_A was not queried by $\mathbf{A}_{\mathsf{DS}}$ to its signing oracle. We conclude that the forgery that $\mathbf{A}_{\mathsf{DS}}$ outputs is valid.

Case 4 The agent A has sent $\mathsf{sig}[m_A, A]$ and has received a valid answer from the adversary $\mathbf{A}'$ (thus she has sent her nonce N_A) but did not get the last message from $\mathbf{A}'$. We show that the adversary $\mathbf{A}'$ cannot achieve $\mathtt{goal}_1$ (A does not have a contract). In particular, consider the scheduler $\mathbf{S}'$ that schedules A such that she immediately contacts the TTP, sending a resolve request with the message $\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B]\rangle$. Since A did not send any abort request, the TTP will return a valid contract to A that $\mathbf{S}'$ will immediately deliver to A thus $\mathtt{goal}_1$ is not achieved. Note that the adversary cannot have sent an abort request to the TTP, masquerading A, since the communications between A and the TTP are made over a secure channel.

Case 5 The agent A has finished her protocol, that is, she got the last message from the adversary. She has the contract thus $\mathtt{goal}_1$ cannot be achieved.

Conversely, the ASW protocol is balanced w.r.t. the second participant B. Let $\mathtt{goal}_1$ be "B does not have a contract" and $\mathtt{goal}_2$ be "the adversary has a valid contract".

Proposition 4. *$\Pi^{\mathsf{ASW\text{-}B}}$ is balanced w.r.t.* $\mathtt{goal}_1$, $\mathtt{goal}_2$, $\mathcal{O}_{adv}^{\mathsf{ASW\text{-}B}}$, $\mathcal{O}_{adv'}^{\mathsf{ASW\text{-}B}}$, $\mathcal{O}_{sch}^{\mathsf{ASW\text{-}B}}$, *and* $\mathcal{O}_{sch'}^{\mathsf{ASW\text{-}B}}$.

Proof (sketch): The proof is again done by contradiction. Assume there exist restricted adversary machines $\mathbf{A} = \mathbf{A}(\mathcal{O}_{adv}^{\mathsf{ASW}})$ and $\mathbf{A}' = \mathbf{A}'(\mathcal{O}_{adv'}^{\mathsf{ASW}})$ for $\Pi^{\mathsf{ASW\text{-}B}}$ and a scheduler $\mathbf{S} = \mathbf{S}(\mathcal{O}_{sch}^{\mathsf{ASW}})$ for $\Pi^{\mathsf{ASW\text{-}B}}$ and $\mathbf{A}$ such that, for any fair scheduler $\mathbf{S}' = \mathbf{S}'(\mathcal{O}_{sch'}^{\mathsf{ASW}})$ for $\Pi^{\mathsf{ASW\text{-}B}}$ and $\mathbf{A}'$ and for any challenge function $\mathtt{challenge}$, with non negligible probability it holds that $\mathbf{Exp}(\eta, \Pi^{\mathsf{ASW\text{-}B}}, \mathbf{A}, \mathbf{A}', \mathbf{S}, \mathbf{S}', \mathtt{goal}_1, \mathtt{goal}_2, \mathtt{challenge}) = 1$.

We have $\mathcal{S}(\eta) \rightsquigarrow q$ where $\mathcal{S} = \mathcal{S}(\Pi^{\mathsf{ASW\text{-}B}}, \mathbf{A}, \mathbf{S})$. We consider several cases for q.

1. Either the agent B has not received any valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary,
2. The agent B has or has not received a valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary but he has chosen not to answer,
3. The agent B has received a valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary and he has sent his promise to sign $\mathsf{sig}[m_B, B]$ but he did not get a valid answer from $\mathbf{A}$, and the TTP has answered a valid abort query from $\mathbf{A}$ for this contract,
4. The agent B has received a valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary and he has sent his promise to sign $\mathsf{sig}[m_B, B]$ but he did not get a valid answer from $\mathbf{A}$, and the TTP has not answered a valid abort query from $\mathbf{A}$ for this contract,
5. The agent B has received a valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary, he has sent his promise to sign $\mathsf{sig}[m_B, B]$ and he got a valid answer from the adversary.

At least one of these cases must happen with non negligible probability. Bellow we give the intuition that shows how attacks against the protocols can be translated into attacks against the primitives used in the construction. The intuition can be transformed into reduction proofs similar to the ones in the proof of Proposition 3.

Cases 1 and 2 are similar. In both cases, B has not sent his promise to sign $\mathsf{sig}[m_B, B]$. Let us show that $\mathbf{A}'$ cannot achieve $\mathtt{goal}_2$ ("having a valid contract") for every scheduler $\mathbf{S}'$ fair for $\Pi^{\mathsf{ASW\text{-}B}}$ and $\mathbf{A}'$. In particular, consider the scheduler that schedules B such that he refuses to initiate any contract-signing protocol with the adversary $\mathbf{A}'$. The only way for $\mathbf{A}'$ to obtain a valid contract either of the form $\langle \mathsf{sig}[m_A, A], N_A, \mathsf{sig}[m_B, B], N_B \rangle$ or of the form $\mathsf{sig}[\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B] \rangle, T]$ is to forge a valid signature of B or T or to contact the TTP sending a message of the form $\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B] \rangle$ to T. In that case $\mathbf{A}'$ has forged a valid signature of B. In both cases, $\mathbf{A}'$ must have forged a valid signature of an honest agent.

Case 3 The agent B has received a valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary and he has sent his promise to sign $\mathsf{sig}[m_B, B]$ but he did not get a valid answer from the adversary, and the TTP has answered a valid abort query from $\mathbf{A}$ for this contract. Let us show that the adversary $\mathbf{A}'$ cannot achieve $\mathtt{goal}_2$ ("having a valid contract"). The TTP has not provided and would not provide any valid contract to any party. The only way for the adversary $\mathbf{A}'$ to obtain a valid contract of the form $\langle \mathsf{sig}[m_A, A], N_A, \mathsf{sig}[m_B, B], N_B \rangle$ or $\mathsf{sig}[\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B] \rangle, T]$ is to forge compute N_B in the first case or compute a valid signature of T in the second case.

Case 4 The agent B has received a valid message of the form $\mathsf{sig}[m_A, A]$ from the adversary and he has sent his promise to sign $\mathsf{sig}[m_B, B]$ but he did not get a valid answer from **A**, and the TTP has not answered a valid abort query from **A** for this contract. Let us show that the adversary $\mathbf{A}'$ cannot achieve $\mathtt{goal}_1$ (B does not have a contract). In particular, consider the scheduler $\mathbf{S}'$ that schedules B such that she immediately contacts the TTP, sending a resolve request with the message $\langle \mathsf{sig}[m_A, A], \mathsf{sig}[m_B, B] \rangle$. Since T has not answered a valid abort request from $\mathbf{A}'$ for this contract, B would receive a valid contract from the TTP thus $\mathtt{goal}_1$ is not achieved.

Case 5 The agent B has has the contract thus $\mathtt{goal}_1$ cannot be achieved.

F Security for Digital Signature Schemes

Definition 6. [Security of a digital signature scheme] *Let* $\mathsf{DS} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ *be a digital signature scheme. Consider an adversary* **A** *that is given input a public key* pk *and access to a signing oracle* $\mathcal{O}_{\mathcal{S}}(\mathrm{sk}, \cdot)$*, where* pk *and* sk *are matching keys generated via* $(\mathrm{pk}, \mathrm{sk}) \stackrel{R}{\leftarrow} \mathcal{K}(1^\eta)$*. The oracle takes input a message* M *and returns a signature* $\sigma \stackrel{R}{\leftarrow} \mathcal{S}(\mathrm{sk}, M)$*.* **A** *queries this oracle on messages of its choice, and eventually outputs a forgery* (M, σ)*. The adversary's advantage in attacking the scheme is the probability that it outputs a pair* (M, σ) *such that* σ *is a valid signature for message* M *and this message was not queried to the signing oracle.* DS *is said to be secure against existential forgery under adaptive chosen-message attacks (or, simply, secure) if the advantage of any efficient* **A** *is negligible in* k*. Here and for other definitions in the paper we adopt the convention that the time complexity of adversary* **A** *is the execution time of the entire experiment, including the time taken for parameter and key generation, and computation of answers to oracle queries.*

G Preimage Resistant Hash Functions

Definition 7 (Hash Functions.). *A hash function family* $\{h_k(\cdot)\}_{k \in \{0,1\}^\eta}$ *consists of algorithms for key generation and function evaluation. We assume that for security parameter* η*, key generation consists in choosing* $k \stackrel{R}{\leftarrow} \{0,1\}^\eta$*. Hash function evaluation for key* k *takes an arbitrary input in* $\{0,1\}^*$ *and returns a bit string* $y \in \{0,1\}^l$*, for some constant* l*. We write* $y \leftarrow h_k(x)$ *for the process of evaluating the hash function on* x *for key* k*.*

In this paper we use hash functions that are preimage resistant.

Definition 8 (Preimage resistance). *We say that the hash function family* $\{h_k(\cdot)\}_{k \in \{0,1\}^\eta}$ *is preimage resistant if for probabilistic polynomial time algorithms* **A**

$$\Pr\left[\, y \leftarrow h_k(x)\,;\; x' \stackrel{R}{\leftarrow} \mathbf{A}(k, y) \;:\; h_k(x') = y \,\right]$$

is a negligible function in η*. The probability is taken over the random choices* $k \stackrel{R}{\leftarrow} \{0,1\}^\eta$; $x \stackrel{R}{\leftarrow} \{0,1\}^\eta$*, as well as the coins used by the adversary.*