A Formal Definition of Online Abuse-freeness*

Ralf Küsters[1], Henning Schnoor[2], and Tomasz Truderung[1]

[1] Universität Trier, Germany,
{kuesters,truderung}@uni-trier.de
[2] Christian-Albrechts-Universität zu Kiel, Germany,
schnoor@ti.informatik.uni-kiel.de

Abstract. Abuse-freeness is an important security requirement for contract-signing protocols. In previous work, Kähler, Küsters, and Wilke proposed a definition for *offline* abuse-freeness. In this work, we generalize this definition to *online* abuse-freeness and apply it to two prominent contract-signing protocols. We demonstrate that online abuse-freeness is strictly stronger than offline abuse-freeness.

Key words: contract signing, cryptographic protocols, formal verification

1.1 Introduction

In a (two-party) contract-signing protocol (see, e.g., [4, 3, 9]), two parties, A (Alice) and B (Bob), aim to exchange signatures on a contractual text that they previously agreed upon. In this paper, we consider *optimistic* contract-signing protocols. In such protocols, a trusted third party T (TTP), serving as an impartial judge, is not involved in every protocol run, but in case of a problem only.

A central security property for optimistic contract-signing, introduced in [9], is abuse-freeness: This property (formulated for the case of the honest signer Alice) requires that there is no state in a protocol run in which dishonest Bob (the prover) can convince an outside party, Charlie (the verifier), that the protocol is in an unbalanced state, i.e., a state in which Bob has both (i) a strategy to prevent Alice from obtaining a valid contract and (ii) a strategy to obtain a valid contract himself. In other words, if a contract-signing protocol is not abuse-free, then Alice can be misused by Bob to get leverage for another contract (with Charlie). Obviously, abuse-freeness is a highly desirable security property.

In [12], Kähler, Küsters, and Wilke presented the first rigorous and protocol-independent definition of abuse-free for (two-party) optimistic contract-signing. However, their definition focusses on an *offline* setting: Charlie is not actively involved in the protocol run and may receive a single message from Bob only, based on which he has to make his decision.

* This work was partially supported by *Deutsche Forschungsgemeinschaft* (DFG) under Grant KU 1434/5-1.

The goal of this work is to generalize the definition from [12] to the setting in which Charlie may be *online*, i.e., may be actively involved in the protocol run, and to apply the definition to prominent contract-signing protocols.

Contribution of this Work. We propose a definition for *online* abuse-freeness, generalizing the definition of Kähler et al., who considered *offline* abuse-freeness. As theirs, our definition is protocol-independent. More precisely, we define two variants of online abuse-freeness: *Weak* abuse-freeness requires that there is no way for dishonest Bob to convince the verifier that the protocol is *currently* in an unbalanced state. As will be explained in Section 1.3, in the setting of *offline* abuse-freeness, every contract-signing protocol is weakly abuse-free. *Strong* abuse-freeness requires that Bob cannot even prove to Charlie that the protocol was in an unbalanced state at some point of the run.

We apply our definitions to two prominent contract signing protocols: a protocol by Asokan, Shoup, and Waidner [3] (ASW protocol), and one by Garay, Jakobsson, and P. MacKenzie [9] (GJM protocol). The latter was explicitely designed with abuse-freeness in mind. Depending on whether the verifier is allowed to eavesdrop on the network connection between the signers or on the channel between the signers and the TTP, and whether the initiator or responder in the protocol is dishonest, the protocols behave differently: We show that if the verifier can read the messages between the signers and the TTP, and the initiator is dishonest, then the ASW protocol is vulnerable to a very strong attack, i.e. it is not even weakly abuse-free. In this attack, the online aspect of our definition plays a crucial role, as the verifier "dictates" parts of the messages sent by the dishonest signer. In all other situations, ASW is weakly, but not strongly abuse-free. The GJM protocol shows a stronger resistance against abuse: It is weakly abuse-free in all situations, and strongly abuse free if the verifier cannot eavesdrop on the network channel between the signers.

Related Work. As mentioned above, Kähler et al. [12] introduced the first rigorous and protocol-independent definition of offline abuse-freeness.

Kremer et al. [13] analyzed the ASW and GJM protocol w.r.t. abuse-freeness using a finite-state model checking tool. They explicitly needed to specify the behavior of dishonest principals and which states are the ones that are convincing to Charlie.

Chadha et al. [5] introduce a stronger notion than abuse-freeness, namely *balance*: A protocol is balanced, if unbalanced states (see above) do not occur at all. Obviously, a balanced protocol is abuse-free as well. However, balance is very difficult to achieve. In fact, as shown by Chadha et al. [6], if principals are optimistic, i.e., they are willing to wait for messages of other parties, balance is impossible to achieve.

Procedures for deciding properties of contract-signing protocols, including balance, were presented in [11] and [10].

Shmatikov and Mitchell [16] employ the finite-state model checker Murφ to automatically analyze contract-signing protocols. They approximate the notion of abuse-freeness by a notion similar to balance.

A cryptographic definition of the balance property was presented by Cortier, Küsters, and Warinschi in [7].

Aizatulin, Schnoor, and Wilke [2] introduced a contract signing protocol which satisfies a probabilistic notion of balance. Wang [17] introduces an abuse-free contract signing protocol based on the RSA signature scheme.

Structure of the Paper. In Section 1.2, we introduce our protocol model. The definition of abuse-freeness is then given in Section 1.3, and applied to the ASW and GJM protocols in Sections 1.4 and 1.5, respectively. Proofs of our results can be found in our technical report [14].

1.2 Protocol Model

In this section, following [15], we present a quite abstract symbolic protocol model. In this model, processes are represented as functions that from a sequence of input messages (the messages received so far) produce output messages. This model is the basis of our definition of abuse-freeness provided in the next section. We note, however, that the details of the model are not essential for the definition. The main motivation for using this model is brevity of presentation. We could as well have used another protocol model, such as the applied pi calculus [1].

1.2.1 Terms and Messages

Let Σ be some signature for cryptographic primitives (including a possibly infinite set of constants for representing participant names, etc.), $X = \{x_1, x_2, \dots\}$ be a set of variables, and Nonce be an infinite set of *nonces*, where the sets Σ, X, and Nonce are pairwise disjoint. For $N \subseteq \mathsf{Nonce}$, the set T_N of *terms* over $\Sigma \cup N$ and X is defined as usual. Ground terms, i.e., terms without variables, represent messages. We assume some fixed equational theory associated with Σ and denote by $\equiv$ the congruence relation on terms induced by this theory. The exact definition of Σ and the equational theory will depend on the cryptographic primitives used in the protocol under consideration. A simple example of a signature Σ_{ex} and its associated equational theory is provided in Figure 1.1. A term of the form $\mathsf{sig}(\mathsf{sk}(k), m)$ represents a message m signed using the (private) key $\mathsf{sk}(k)$. Checking validity of such a signature is modeled by equation (1.1). The fact that signatures do not necessarily hide the signed message is expressed by equation (1.2). A term of the form $\{x\}^r_{\mathsf{pk}(k)}$ represents the ciphertext obtained by encrypting x under the public key $\mathsf{pk}(k)$ using randomness r. Decryption of such a term using the corresponding private key $\mathsf{sk}(k)$ is modeled by equation (1.3). A term of the form $\langle x, y\rangle$ models the pairing of terms x and y. The components x and y of $\langle x, y\rangle$ can be extracted by applying the operators $\mathsf{first}(\cdot)$ and $\mathsf{second}(\cdot)$, respectively, as modeled by the equations (1.4) and (1.5). A term of the form $\mathsf{hash}(m)$ represents the result of applying a hash function to a message m. Note that $\mathsf{hash}(\cdot)$ is a free symbol, i.e. there is no equation involving this

$$\begin{aligned} \mathsf{checksig}(\mathsf{sig}(\mathsf{sk}(k), m), \mathsf{pk}(k)) &= \mathsf{T} && (1.1)\\ \mathsf{extractmsg}(\mathsf{sig}(\mathsf{sk}(k), m)) &= m && (1.2)\\ \mathsf{dec}(\{x\}^r_{\mathsf{pk}(k)}, \mathsf{sk}(k)) &= x && (1.3)\\ \mathsf{first}(\langle x, y\rangle) &= x, && (1.4)\\ \mathsf{second}(\langle x, y\rangle) &= y && (1.5) \end{aligned}$$

Fig. 1.1. The equational theory associated with the signature $\Sigma_{ex} = \{\mathsf{sig}(\cdot,\cdot), \langle\cdot,\cdot\rangle, \{\cdot\}^{\cdot}_{\cdot}, \mathsf{T}, \mathsf{checksig}(\cdot,\cdot), \mathsf{extractmsg}(\cdot), \mathsf{first}(\cdot), \mathsf{second}(\cdot), \mathsf{hash}(\cdot), \mathsf{pk}(\cdot), \mathsf{sk}(\cdot)\}$.

symbol in the given equational theory. For example, let $\equiv_{ex}$ denote the congruence relation induced by the equational theory in Figure 1.1, then we have that $\mathsf{dec}(\{a\}^r_{\mathsf{pk}(k)}, \mathsf{first}(\langle\mathsf{sk}(k), b\rangle)) \equiv_{ex} a$.

1.2.2 Event Sequences and Views

Let Ch be a set of *channels* (*channel names*). An *input/output event* is of the form $(c : m)$ and $(\bar{c} : m)$, respectively, for $c \in \mathsf{Ch}$ and a message m (note that $\bar{c} \notin \mathsf{Ch}$). A finite or infinite sequence of events is called an *event sequence.* For a sequence $\rho = (c_1 : m_1)\,(c_2 : m_2), \ldots$ of input events, we denote by $\mathrm{chan}(\rho)$ the sequence $c_1, c_2, \ldots$ of channels in this sequence. For $C \subseteq \mathsf{Ch}$, we denote by $\rho_{|C}$ the subsequence of ρ containing only the events of the form $(c : m)$ with $c \in C$.

Let $\tau \in T_N$ be a term, which may contain variables x_1, x_2, ... Then, with ρ as above, we denote by $\tau[\rho]$ the message $\tau[m_1/x_1, m_2/x_2, \ldots]$, where x_i is replaced by m_i. For example, assume that $\tau_{ex} = \mathsf{dec}(x_1, \mathsf{first}(x_2))$ and $\rho_{ex} = (c_1 : \{a\}^r_{\mathsf{pk}(k)}), (c_2 : \langle\mathsf{sk}(k), b\rangle)$. Then $\tau_{ex}[\rho_{ex}] = \mathsf{dec}(\{a\}^r_{\mathsf{pk}(k)}, \mathsf{first}(\langle\mathsf{sk}(k), b\rangle)) \equiv_{ex} a$.

Borrowing the notion of static equivalence from [1], we call two event sequences ρ and ρ' *statically equivalent w.r.t. a set* $C \subseteq \mathsf{Ch}$ *of channels and a set* $N \subseteq \mathsf{Nonce}$ *of nonces*, written $\rho \equiv^C_N \rho'$, if (i) $\mathrm{chan}(\rho_{|C}) = \mathrm{chan}(\rho'_{|C})$ and (ii) for every $\tau_1, \tau_2 \in T_N$ we have that $\tau_1[\rho_{|C}] \equiv \tau_2[\rho_{|C}]$ iff $\tau_1[\rho'_{|C}] \equiv \tau_2[\rho'_{|C}]$. Intuitively, $\rho \equiv^C_N \rho'$ means that a party listening on channels C and a priori knowing the nonces in N cannot distinguish between the inputs received according to ρ and ρ'. We call the equivalence class of ρ w.r.t. $\equiv^C_N$, the (C, N)*-view* on ρ. For example, if a and b are different constants, k, k', r and r' are nonces, $C = \{c_1, c_2\}$, and $N = \emptyset$, then it is easy to see that $\rho^1_{ex} = (c_1 : \{a\}^r_{\mathsf{pk}(k)}), (c_2 : \langle\mathsf{sk}(k'), b\rangle), (c_3 : \mathsf{sk}(k))$ and $\rho^2_{ex} = (c_1 : \{b\}^{r'}_{\mathsf{pk}(k)}), (c_2 : \langle\mathsf{sk}(k'), b\rangle)$ yield the same (C, N)-view w.r.t. $\equiv_{ex}$.

1.2.3 Processes

A process is, basically, a function that given a sequence of input events (representing the history so far) produces a sequence of output events. We require that a process behaves the same on inputs on which it has the same view. More precisely, a *process* is a tuple $\pi = (I, O, N, f)$ where

(i) $I, O \subseteq \mathsf{Ch}$ are finite sets of *input* and *output* channels, respectively,
(ii) $N \subseteq \mathsf{Nonce}$ is a set of *nonces used by* π,
(iii) f is a mapping which assigns a sequence $f(U) = (c_1 : \tau_1) \cdots (c_n : \tau_n)$ with $c_i \in O$ and $\tau_i \in T_N$ to each (I, N)-view U.

We note that (iii) guarantees that π performs the same computation on event sequences that are equivalent according to $\equiv_N^I$, and hence, on which π has the same view.

For an event sequence ρ, we write $\pi(\rho)$ for the output produced by π on input ρ. This output is $(c_1 : \tau_1[\rho']) \cdots (c_n : \tau_n[\rho'])$, where $\rho' = \rho_{|I}$ and $(c_1 : \tau_1) \cdots (c_n : \tau_n) = f(U)$ for the equivalence class U of ρ w.r.t. $\equiv_N^I$. For example, let $I = \{c_1, c_2\}$, $N = \emptyset$, U be the equivalence class of ρ_{ex}^1, and assume that $f(U) = (c_4 : \langle x_1, \mathsf{first}(x_2)\rangle)$. Then, $\pi(\rho_{ex}^1) = (c_4 : \langle \{a\}_{\mathsf{pk}(k)}^r, \mathsf{first}(\langle \mathsf{sk}(k'), b\rangle)\rangle)$, which modulo $\equiv_{ex}$ can be written equivalently as $(c_4 : \langle \{a\}_{\mathsf{pk}(k)}^r, \mathsf{sk}(k')\rangle)$ and $\pi(\rho_{ex}^2) = (c_4 : \langle \{b\}_{\mathsf{pk}(k)}^{r'}, \mathsf{first}(\langle \mathsf{sk}(k'), b\rangle)\rangle)$, which modulo $\equiv_{ex}$ can be equivalently written as $(c_4 : \langle \{b\}_{\mathsf{pk}(k)}^{r'}, \mathsf{sk}(k')\rangle)$. Note that since ρ_{ex}^1 and ρ_{ex}^2 yield the same (I, N)-view w.r.t. $\equiv_{ex}$, π performs the same transformation on ρ_{ex}^1 and ρ_{ex}^2. We refer to I, O and N by I_π, O_π, and N_π, respectively. We note that the sets I_π and O_π do not have to be disjoint, i.e., π can send messages to itself. By $\mathsf{Proc}(I, O, N)$ we denote the set of all processes π with $I_\pi \subseteq I$, $O_\pi \subseteq O$, and $N_\pi \subseteq N$.

1.2.4 Systems and Runs

A *system* S is a finite set of processes with disjoint sets of input channels and sets of nonces, i.e., $I_\pi \cap I_{\pi'} = \emptyset$ and $N_\pi \cap N_{\pi'} = \emptyset$, for distinct $\pi, \pi' \in S$. We will write $\pi_1 \parallel \cdots \parallel \pi_n$ for the system $\{\pi_1, \cdots, \pi_n\}$.

Given a system S and a finite sequence s_0 of output events, a *run* ρ *of* S *initiated by* s_0 is a finite or infinite sequence of input and output events which evolves from s_0 in a natural way: An output event is chosen non-deterministically (initial from s_0). Once an output event has been chosen, it will not be chosen anymore later on. By definition of systems, there exists at most one process, say π, in S with an input channel corresponding to the output event. Now, π (if any) is given the input event corresponding to the chosen output event, along with all previous input events on channels of π. Then, π produces a sequence of output events as described above. Now, from these or older output events an output event is chosen non-deterministically, and the computation continues as before.

We emphasize that s_0 may induce many runs, due to the non-deterministic delivery of messages. In what follows, we assume *fair* runs, i.e., every output event in a run will eventually be chosen. A run is *complete* if it is either infinite or else all output events have been chosen at some point. For runs ρ, ρ', we write $\rho \leq \rho'$, if ρ' is an extension of ρ, i.e., is obtained by continuing the run ρ.

1.2.5 Protocols

A *protocol* is a tuple $P = (A, in, out, nonce, s_0, \Pi)$, where

(i) A is a finite set of *agent names*. An agent $a \in A$ has access to his/her nonces $nonce(a)$, input and output channels $in(a), out(a) \subseteq \mathsf{Ch}$, respectively, such that $nonce(a) \cap nonce(a') = \emptyset$ and $in(a) \cap in(a') = \emptyset$, for $a \neq a'$,
(ii) s_0 is a finite sequence of output events, the *initial output sequence*, for initializing parties,
(iii) for every $a \in A$, $\Pi(a) \subseteq \mathsf{Proc}(in(a), out(a), nonce(a))$ is the set of *programs* or *processes of* a. We will write $P(a)$ for $\Pi(a)$.

If $A = \{a_1, \ldots, a_n\}$ and $\pi_i \in \Pi(a_i)$, then the system $(\pi_1 \parallel \cdots \parallel \pi_n)$ is an *instance of* P. A *run of* P is a fair run of some instance of P initiated by s_0. A *property* γ of P is a subset of runs of P.

We note that our model allows to express nondeterminism: To make a nondeterministic choice, a program can simply send two (or more) messages to itself, and change its behaviour depending on which message arrives first.

1.3 Online Abuse-Freeness

We define (online) abuse-freeness of a protocol $P = (A, in, out, nonce,\ s_0, \Pi)$ with respect to two distinct agents of P: the prover $\mathsf{p} \in A$ and the verifier $\mathsf{v} \in A$. Both agents are considered to be dishonest, and hence, the sets of programs of these agents will typically contain all possible processes, i.e., $P(\mathsf{p}) = \mathsf{Proc}(in(\mathsf{p}), out(\mathsf{p}), nonce(\mathsf{p}))$ and $P(\mathsf{v}) = \mathsf{Proc}(in(\mathsf{v}), out(\mathsf{v}), nonce(\mathsf{v}))$; these processes are only limited by their network interfaces, i.e., the set of input/output channels available to them.

Moreover, we define abuse-freeness of P with respect to two properties of P: γ^+ and γ^-. The property γ^+ is supposed to contain all the runs of P in which p obtains a valid contract from an honest signer a and γ^- is supposed to contain all runs where the honest signer a is prevented from obtaining a valid contract from p.

To define abuse-freeness, we first need to formalize the notion of an unbalanced run. Intuitively, a run of an instance of a protocol P is unbalanced with respect to the properties γ^+ and γ^- if p has both a strategy to achieve γ^+ (i.e., enforce a continuation of the run so that the overall run belongs to γ^+) and a strategy to achieve γ^-. In other words, in an unbalanced state, the prover can unilaterally determine the outcome of the protocol: i) obtain a signed contract from the honest signer a or ii) prevent a from obtaining a signed contract from p.

To model the choice made by the prover to either achieve γ^+ or γ^-, we introduce the following notation. We assume that the prover p has a distinct input channel $\mathsf{ch}_{\mathsf{choice}}$ which is not an output channel of any agent in the protocol P. Moreover, we assume that the events $(\overline{\mathsf{ch}}_{\mathsf{choice}} : 0)$ and $(\overline{\mathsf{ch}}_{\mathsf{choice}} : 1)$ belong to the initial event sequence s_0 of P. Intuitively, if in a run p receives 1 on $\mathsf{ch}_{\mathsf{choice}}$, then p will try to achieve γ^+. If p receives 0 on $\mathsf{ch}_{\mathsf{choice}}$, then p will try to achieve γ^-. More precisely, a run ρ in which neither $(\mathsf{ch}_{\mathsf{choice}} : 0)$ nor $(\mathsf{ch}_{\mathsf{choice}} : 1)$ has been delivered is called *open*; intuitively, in such a run the prover has not yet made a decision. Otherwise the run is called *closed*. In such a run, p tries to achieve γ^+ or γ^- depending on the message received; note that only the first message received on $\mathsf{ch}_{\mathsf{choice}}$ will set p's goal.

In the following definition, given a finite open run ρ, we denote by $\rho^{(\mathsf{ch}_{\mathsf{choice}}:0)}$ the run obtained from ρ by delivering 0 on channel $\mathsf{ch}_{\mathsf{choice}}$ to p, i.e., in $\rho^{(\mathsf{ch}_{\mathsf{choice}}:0)}$ the prover p is now determined to achieve γ^-. The run $\rho^{(\mathsf{ch}_{\mathsf{choice}}:1)}$ is defined analogously. We say that a run ρ' is a *complete extension* of ρ if ρ' is an extension of ρ and is complete.

We are now ready to formally define unbalanced runs.

Definition 1. *Given an instance S of a protocol P as above with a prover* p *and two properties γ^+ and γ^-, we say that a finite open run ρ of S is* unbalanced*, if the following two conditions hold true:*

(i) γ^- holds in every complete extension of $\rho^{(\mathsf{ch}_{\mathsf{choice}}:0)}$.
(ii) γ^+ holds in every complete extension of $\rho^{(\mathsf{ch}_{\mathsf{choice}}:1)}$.

Now, intuitively, a protocol is abusive if a prover p can convince the verifier v that the current run is unbalanced. In other words, p can convince v that in the current run he, the prover, has a strategy to obtain a valid contract from the honest signer (and hence, close the deal) and a strategy to prevent the honest signer from obtaining a valid contract (and hence, cancel the deal). This may convince v to agree into a deal with p that for p is more profitable than the one with the honest signer. Thus, in an abusive protocol, p can take advantage of the honest signer.

Since we consider *online* abuse-freeness in this paper, we allow v to be actively involved in the protocol run. In particular, p and v can freely exchange messages during a run. For example, v could dictate (parts of the) messages p is supposed to send to the honest signer, and v could request to receive the private keys of p. The verifier v may even control some of the network traffic. However, this is not hard-wired in our definition. The power of p and v can be modeled in a flexible way in terms of the programs p and v may run and the network interface they have.

We will consider two forms of abuse-freeness, namely strong and weak abuse-freeness. In the strong form, p merely needs to convince v that the run was unbalanced at some point. In contrast, for the weak form, p needs to convince v that the run is unbalanced in the *current state* of the run. Since in the latter case, the task of p is harder, the latter form of abuse-freeness is weaker. It is desirable that a protocol is abuse-free in the strong sense since the fact that a run was and potentially still is unbalanced might already be sufficient incentive for v to agree into a deal with p.

In the formal definition of (online) abuse-freeness, we assume that the verifier v can *accept* a run by sending the message accept on the designated channel $\mathsf{ch}_{\mathsf{accept}}$, indicating that v is convinced that the run is/was unbalanced. We say that a finite run is *freshly accepted*, if the message accept is sent by v in the last step of this run.

We also use the following notation in the definition of abuse-freeness: Let $P = (A, in, out, nonce, s_0, \Pi)$ be a protocol. For a program $v \in \Pi(\mathsf{v})$ of the verifier, we write $P_{|v}$ for the protocol that coincides with P except that the set $\Pi(\mathsf{v})$ of programs of v is restricted to $\{v\}$. In particular, in every instance of $P_{|v}$ the verifier runs the program v.

We are now ready to define (online) abuse-freeness. We start with the strong form of abuse-freeness.

Definition 2. Let $P = (A, in, out, nonce, s_0, \Pi)$ with $\mathsf{p}, \mathsf{v} \in A$. Let γ^+ and γ^- be properties of P. Then, the protocol P is called (γ^+, γ^-)-***abusive*** w.r.t. the prover p and the verifier v, if there is a program $v \in \Pi(\mathsf{v})$ of v such that the following conditions are satisfied:

(i) If an open run ρ of $P_{|v}$ is accepted by v, then there is an unbalanced run ρ' with $\rho' \leq \rho$.
(ii) There exists an open, freshly accepted, unbalanced run ρ of $P_{|v}$.

The protocol P is *(strongly)* (γ^+, γ^-)-***abuse-free*** w.r.t. p and v, if P is not (γ^+, γ^-)-abusive w.r.t. p and v.

Condition (i) in the above definition says that if v accepts a run, i.e., is convinced that the run was unbalanced at some point, then this is in fact the case. Note that according to the definition of unbalanced runs, v may help p to achieve his goals (γ^+ or γ^-). One could as well consider a variant where p has to achieve these goals against v (and in fact, our negative results, presented in Sections 1.4 and 1.5, use a prover that works without the help of the verifier). However, this would make the definition only weaker. We note that it would not make sense to consider closed runs in Condition (i): The definition of unbalanced runs only applies to open runs. Moreover, the restriction to open runs does not limit the power of any agent.

While Condition (i) is the core of the above definition, it would not make sense without Condition (ii): A verifier who never accepts a run would satisfy Condition (i) trivially. Moreover, a verifier who only accepts runs which are not unbalanced anymore would potentially also suffice to meet Condition (i). By Condition (ii) we require that the strategy of the verifier for accepting a run is reasonable in the sense that there is at least one run which is accepted and which is still unbalanced.

Altogether the above definition says that a protocol is abuse-free if there is no program a verifier could run which i) reliably tells, for any dishonest prover (which the verifier does not trust), whether a run was unbalanced and ii) accepts an actual unbalanced run.

We note that the above definition is possibilistic. It does, for example, not take into account the probability with which unbalanced runs occur or a verifier accepts an (unbalanced) run. As mentioned in the introduction, a cryptographic, in particular probabilistic definition of the balance property, which is stronger than abuse-freeness in that it does not require a dishonest signer to convince an outside party of the fact that he is in an unbalanced state, was presented in [7].

Given Definition 2, it is now straightforward to define weak abuse-freeness:

Definition 3. Let $P, \mathsf{p}, \mathsf{v}, \gamma^+$, and γ^- be given as in Definition 2. Then protocol P is *strongly* (γ^+, γ^-)*-abusive* w.r.t. p and v, if there is a program v such that the following conditions are satisfied:

(i) If an open run ρ of $P_{|v}$ is freshly accepted, then ρ is unbalanced,
(ii) There exists an open, freshly accepted, unbalanced run ρ of $P_{|v}$.

The protocol P is *weakly* (γ^+, γ^-)*-abuse-free* w.r.t. p and v, if P is not strongly (γ^+, γ^-)-abusive w.r.t. p and v.

This notion differs from the (strong) abuse-freenes only in Condition (i): Now we require that the accepted run *is* unbalanced, not only that it was unbalanced at some previous point. Clearly, (strong) abuse-freeness implies weak abuse-freeness.

Note that a notion like weak abuse-freeness does not make sense in the offline setting considered in [12]: If the verifier receives only a single message from the prover, this message can only prove that the protocol was in an unbalanced state at some point during the protocol run; since the prover may withhold that evidence for as long as he wishes, it does not prove that the current state is unbalanced.

1.4 The ASW Protocol

In this section, we study abuse-freeness of the contract-signing protocol proposed by Asokan, Shoup, and Waidner (ASW protocol) in [3].

In [12], it has been shown (in a *synchronous* communication model without optimistic honest parties, see below) that the ASW protocol is *offline* abusive. Not surprisingly, the protocol is also abusive in the online setting. More precisely, we show that the protocol is weakly abusive. Interestingly, we can show that the protocol is, in some cases, even *strongly abusive.* For this attack, it is crucial that the verifier is online, i.e., can interact with the prover. In fact, the verifier will dictate part of the message the prover sends to the honest signer.

1.4.1 Description of the Protocol

The ASW protocol assumes the following scenario: Alice and Bob want to sign a contract and a TTP is present. The following two types of messages, the *standard contract (SC)* and the *replacement contract (RC)*, will be recognized as valid contracts between Alice and Bob with contractual text text: $SC = \langle me_1, N_A, me_2, N_B \rangle$ and $RC = \mathsf{sig}(\mathsf{sk}(k_t), \langle me_1, me_2 \rangle)$ where N_A and N_B stand for nonces, $me_1 = \mathsf{sig}(\mathsf{sk}(k_a), \langle A, B, \mathsf{text}, \mathsf{hash}(N_A) \rangle)$, and $me_2 =$

$\mathsf{sig}(\mathsf{sk}(k_b), \langle me_1, \mathsf{hash}(N_B)\rangle)$, with $\mathsf{sk}(k_t)$, $\mathsf{sk}(k_a)$, and $\mathsf{sk}(k_b)$ denoting the private keys of the TTP, Alice, and Bob, respectively. In addition to SC and RC, the variants of SC and RC which one obtains by exchanging the roles of A and B are regarded as valid contracts.

There are three interdependent parts to the protocol: an exchange protocol, an abort protocol, and a resolve protocol. The *exchange protocol* consists of four steps, which, in Alice-Bob notation, are displayed in Fig. 2. The first two messages, me_1 and me_2, serve as respective *promises* of Alice and Bob to sign the contract, and N_A and N_B serve as *contract authenticators*: After they have been revealed, Alice and Bob can compose the standard contract, SC.

The *abort protocol* is run between Alice and the TTP and is used by Alice to abort the contract signing process when she does not receive Bob's promise. Alice will obtain (from the TTP) an abort receipt or, if the protocol instance has already been resolved (see below), a replacement contract. The first step is $A \to T\colon ma_1$, where $ma_1 = \mathsf{sig}(\mathsf{sk}(k_a), \langle \mathsf{aborted}, me_1\rangle)$ is Alice's *abort request*; the second step is the TTP's reply, which is either $\mathsf{sig}(\mathsf{sk}(k_t), \langle \mathsf{aborted}, ma_1\rangle)$, the *abort receipt*, if the protocol has not been resolved, or the replacement contract, RC.

$$\begin{aligned} A &\to B : me_1 \\ B &\to A : me_2 \\ A &\to B : N_A \\ B &\to A : N_B \end{aligned}$$

Fig. 1.2: ASW exchange protocol

The *resolve protocol* can be used by Alice and Bob to resolve the protocol, which either results in a replacement contract or, if the protocol has already been aborted, in an abort receipt. When Bob runs the protocol (because Alice has not sent her contract authenticator yet), the first step is $B \to T\colon \langle me_1, me_2\rangle$; the second step is the TTP's reply, which is either the abort receipt $\mathsf{sig}(\mathsf{sk}(k_t), \langle \mathsf{aborted}, ma_1\rangle)$, if the protocol has already been aborted, or the replacement contract, RC. The same protocol (with roles of A and B exchanged) is also used by Alice.

1.4.2 Modeling

Our modeling of the ASW protocol uses the equational theory presented in Section 1.2 (however, without encryption, which is not used in the protocol). We consider, besides the regular protocol participants of the protocol—Alice, Bob, and the trusted third party—two additional parties, the verifier and a key distribution center. We will consider four cases depending on (a) which signer (Alice or Bob) is dishonest and plays the role of the prover and (b) which part of the network is controlled by the verifier.

In each case we assume that the honest signer is *optimistic* in the sense that he/she only contacts the TTP if the dishonest signer allows the honest signer to do so. In other words, the dishonest signer can buy himself as much time as he needs, before the honest signer contacts the TTP. This assumption, also made in [6], seems realistic. In any case, it only makes the dishonest party more powerful, and hence, strengthens our positive results.

Let $P^{A}_{\mathit{ASW\text{-}Net}}$ denote the specification of the ASW protocol, as a protocol in the sense of our definition (see Section 1.2.5), with dishonest Alice and honest

Bob, where the verifier can eavesdrop on (but not block) the network traffic between Alice and Bob. Analogously, $P^B_{\mathsf{ASW\text{-}Net}}$ denotes the protocol with dishonest Bob and honest Alice, where again the verifier can eavesdrop on the network traffic between Alice and Bob. Let $P^A_{\mathsf{ASW\text{-}TTP}}$ ($P^B_{\mathsf{ASW\text{-}TTP}}$) be the protocols with dishonest Alice (Bob) and honest Bob (Alice), where the verifier can eavesdrop on (but not block) the communication between the signers and the TTP.

In the modeling of these protocols (see below), we allow an honest signer to *not* be engaged in the protocol run. This is of course realistic; also, otherwise the initial state of the protocol would already be unbalanced before a signer has committed to the contract. To model this, we assume that an honest signer decides nondeterministically (see end of Section 1.2.5) as to whether he/she will participate in the protocol run.

More formally, the set of programs of the protocol participants are defined as follows:

Key Distribution Center. The set of programs for this party consists of exactly one program, which generates key pairs (using its set of nonces) for all other parties. Private keys, modeled as terms of the form $\mathsf{sk}(k)$, where k is a nonce, are sent, via dedicated channels, only to the respective parties. Public keys, modeled as terms of the form $\mathsf{pk}(k)$, are distributed to all parties, including the verifier. Honest parties will first wait to receive their public/private key pair and the public keys of the other protocol participants. In the specification of honest parties below this is assumed implicitly.

Dishonest parties (prover and verifier). The sets of programs of dishonest parties contain all the possible processes, only constrained by the network configuration and, possibly, some additional constrains, as described below. We allow the prover and verifier to communicate directly with each other via a direct (asynchronous) channel.

Network configuration. In the protocols $P^A_{\mathsf{ASW\text{-}TTP}}$ and $P^B_{\mathsf{ASW\text{-}TTP}}$ (in which the verifier can eavesdrop on messages between the signers and the TTP) we assume that the messages that Alice and Bob want to send to the TTP are routed through the verifier. We require the verifier to forward these messages to the recipient, i.e. we restrict the set of program of the verifier to those programs which comply with this constraint. However, we assume direct (asynchronous) channels between Alice and Bob.

Similarly, in $P^A_{\mathsf{ASW\text{-}TTP}}$ and $P^B_{\mathsf{ASW\text{-}TTP}}$ messages between Alice and Bob are routed through the verifier, who, as above, can only eavesdrop on these messages. Message between the signers and the TTP can be sent via direct (asynchronous) channels.

TTP. The set of programs of TTP consists of only one program (process), namely the one that performs exactly the steps defined by the protocol as described in Section 1.4.1.

Honest Alice or Bob. The set of programs of Alice in $P^A_{\mathsf{ASW\text{-}TTP}}$ and $P^A_{\mathsf{ASW\text{-}Net}}$ consists of only one program, namely the one described in Section 1.4.1. As mentioned above, at the beginning Alice first nondeterministically chooses whether to participate in the contract signing. Also, she contacts the TTP only if she receives a message from Bob that she is allowed to contact the TTP. The case of honest Bob is analogous.

Remark 1. One could also study the case where the verifier can eavesdrop on all channels. However in this case, both the ASW and GJM protocols clearly are strongly abusive, since the verifier always knows the exact stage of the protocol run.

1.4.3 Security Analysis

We define γ^+ as the set of all runs where the prover is able to construct the standard contract or has received the replacement contract. Analogously, γ^- consists of those runs in which the honest signer is not able to construct the standard contract and has not received the replacement contract. For the ASW protocol, we prove the following results (see our technical report [14] for the proof).

Theorem 1. *The protocol $P^A_{\mathit{ASW\text{-}TTP}}$ is not weakly (γ^+, γ^-)-abuse-free (and hence also not abuse-free). $P^B_{\mathit{ASW\text{-}TTP}}$, $P^A_{\mathit{ASW\text{-}Net}}$, and $P^B_{\mathit{ASW\text{-}Net}}$ are weakly (γ^+, γ^-)-abuse-free but not abuse-free.*

We note that the first result exhibits a particularly devastating attack, which makes heavy use of the fact that the verifier is an online agent. This result shows that under certain conditions the ASW protocol is not even weakly abuse-free. The above results also show that weak abuse-freeness is a much weaker security property than strong abuse-freeness.

Remark 2. The proofs for $P^A_{\mathit{ASW\text{-}Net}}$ and $P^B_{\mathit{ASW\text{-}Net}}$ easily carry over to the case when the verifier not only eavesdrops on the channels between Alice and Bob, but also controls these channels.

1.5 The GJM Protocol

In [12], it has been shown that, in a *synchronous* communication model, the GJM protocol is *offline* abuse-free. In this section, we show that whether it is *online* abuse-free depends on assumptions about what part of the network the verifier can eavesdrop on. In particular, we show that in some cases the GJM protocol is not online abuse-free, which, again, illustrates the fact that online abuse-freeness is stronger than offline abuse-freeness.

1.5.1 Informal Description and Model of the Protocol

The structure of the GJM protocol is the same as the one of the ASW protocol. However, the actual messages exchanged are different. In particular, the exchange

$$\mathsf{pcsver}(w, \mathsf{pk}(x), \mathsf{pk}(y), \mathsf{pk}(z), \mathsf{pcs}(u, \mathsf{sk}(x), w, \mathsf{pk}(y), \mathsf{pk}(z))) = \mathsf{pcsok}, \tag{1.6}$$
$$\mathsf{pcsver}(w, \mathsf{pk}(x), \mathsf{pk}(y), \mathsf{pk}(z), \mathsf{fake}(u, \mathsf{sk}(y), w, \mathsf{pk}(x), \mathsf{pk}(z))) = \mathsf{pcsok}, \tag{1.7}$$
$$\mathsf{sver}(w, \mathsf{pk}(x), \mathsf{pk}(z), \mathsf{sconv}(u, \mathsf{sk}(x), \mathsf{pcs}(v, \mathsf{sk}(x), w, \mathsf{pk}(y), \mathsf{pk}(z)))) = \mathsf{sok}, \tag{1.8}$$
$$\mathsf{tpver}(w, \mathsf{pk}(x), \mathsf{pk}(z), \mathsf{tpconv}(u, \mathsf{sk}(z), \mathsf{pcs}(v, \mathsf{sk}(x), w, \mathsf{pk}(y), \mathsf{pk}(z)))) = \mathsf{tpok}. \tag{1.9}$$

Fig. 1.3. Equations for private contract signatures.

protocol of the GJM protocol the first two messages are so-called private contract signatures (PCS) [9] and the last two messages are actual signatures (obtained by converting the private contract signatures into universally verifiable signatures).

For the GJM protocol we consider the signature $\Sigma_{GJM} = \{$ $\mathsf{sig}(\cdot,\cdot,\cdot)$, $\mathsf{sigcheck}(\cdot,\cdot,\cdot)$, $\mathsf{pk}(\cdot)$, $\mathsf{sk}(\cdot)$, $\mathsf{fake}(\cdot,\cdot,\cdot,\cdot,\cdot)$, $\mathsf{pcs}(\cdot,\cdot,\cdot,\cdot,\cdot)$, $\mathsf{pcsver}(\cdot,\cdot,\cdot,\cdot,\cdot)$, $\mathsf{sconv}(\cdot,\cdot,\cdot)$, $\mathsf{tpconv}(\cdot,\cdot,\cdot)$, $\mathsf{sver}(\cdot,\cdot,\cdot,\cdot)$, $\mathsf{tpver}(\cdot,\cdot,\cdot,\cdot)$, $\langle\cdot,\cdot\rangle$, $\mathsf{first}(\cdot)$, $\mathsf{second}(\cdot)$, A, B, T, text, $\mathsf{initiator}$, $\mathsf{responder}$, ok, pcsok, sok, tpok, $\mathsf{aborted}\}$.

The equational theory for GJM contains, in addition to the equations for pairing and signatures, equations for modeling private contract signatures, as depicted in Figure 1.3. A term of the form $\mathsf{pcs}(u, \mathsf{sk}(x), w, \mathsf{pk}(y), \mathsf{pk}(z))$ stands for a PCS computed by x (with $\mathsf{sk}(x)$) involving the text w, the party y, and the TTP z, while u models the random coins used to compute the PCS. Everybody can verify the PCS with the public keys involved (equation (1.6)), but cannot determine whether the PCS was computed by x or y (equation (1.7)): instead of x computing the "real" PCS, y could have computed a "fake" PCS which would also pass the verification with pcsver. Using sconv and tpconv, see (1.8) and (1.9), a "real" PCS can be converted by x and the TTP z, respectively, into a universally verifiable signature (verifiable by everyone who possesses $\mathsf{pk}(x)$ and $\mathsf{pk}(z)$).

We study the version of the GJM protocol with the modification proposed in [16] to obtain fairness. In the protocol, the following messages are exchanged: The initial messages containing the private contract signatures are $me_1 = \mathsf{pcs}(u, \mathsf{sk}(A), \mathsf{contract}, \mathsf{pk}(B), \mathsf{pk}(TTP))$ and $me_2 = \mathsf{pcs}(u', \mathsf{sk}(B), \mathsf{contract}, \mathsf{pk}(A), \mathsf{pk}(TTP))$, where $\mathsf{sk}(A)$, $\mathsf{pk}(A)$, $\mathsf{sk}(B)$, $\mathsf{pk}(B)$, and $\mathsf{pk}(TTP)$ are the private and public keys of Alice, Bob, and the TTP. The abort request sent by Alice is of the form $ma_1 = \mathsf{sig}(w, \mathsf{sk}(A), \langle\mathsf{contract}, A, B, \mathsf{aborted}\rangle)$, where w are random coins (for the GJM protocol, we consider randomized signatures). The resolve request sent by Alice is $\langle me_1, me_2\rangle$, the resolve request from Bob is $\langle me_2, me_1\rangle$. As mentioned earlier, the structure of the protocol is the same as for the ASW protocol (see Section 1.4).

1.5.2 Security Analysis

We study the cases $P^A_{\mathsf{GJM\text{-}TTP}}$, $P^B_{\mathsf{GJM\text{-}TTP}}$, $P^A_{\mathsf{GJM\text{-}Net}}$ and $P^B_{\mathsf{GJM\text{-}Net}}$, which are defined analogously to the case of ASW (see Section 1.4.2). The properties γ^+ and γ^- are also defined analogously to the case of the ASW protocol (see Section 1.4.3). The proof of this result can be found in our technical report [14].

Theorem 2. *1. $P^A_{\mathsf{GJM\text{-}TTP}}$ and $P^B_{\mathsf{GJM\text{-}TTP}}$ are (γ^+, γ^-)-abuse-free.*

2. $P^A_{\mathsf{GJM\text{-}Net}}$ and $P^B_{\mathsf{GJM\text{-}Net}}$ are weakly (γ^+, γ^-)-abuse-free but not (γ^+, γ^-)-abuse-free.

As made precise by this theorem, abuse-freeness of the GJM protocol in the online setting depends on the assumptions about what part of the network the verifier can eavesdrop on. In the offline case, the verifier was not allowed to easvesdrop on any part of the network (and of course, was also not allowed to be actively involved in the protocol run). Therefore, and just as in the case of the ASW protocol, our positive results are stronger than those shown for offline abuse-freeness. Conversely, our negative results exhibit the extra power of online verifiers.

References

1. M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *POPL 2001*, pages 104–115. ACM Press, 2001.
2. M. Aizatulin, and H. Schnoor, and Th. Wilke Computationally Sound Analysis of a Probabilistic Contract Signing Protocol. In *ESORICS 2009*, volume 5789 of *LNCS*, pages 571–586, Springer, 2009.
3. N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *IEEE Symposium on Research in Security and Privacy*, pages 86–99, 1998.
4. M. Ben-Or, O. Goldreich, S. Micali, and R.L. Rivest. Fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.
5. R. Chadha, M.I. Kanovich, and A.Scedrov. Inductive methods and contract-signing protocols. In *CCS 2001*, pages 176–185. ACM Press, 2001.
6. R. Chadha, J.C. Mitchell, A. Scedrov, and V. Shmatikov:. Contract Signing, Optimism, and Advantage. In *CONCUR 2003*, volume 2761 of *LNCS*, pages 361–377. Springer, 2003.
7. V. Cortier, R.Küsters, and B. Warinschi. A Cryptographic Model for Branching Time Security Properties - The Case of Contract Signing Protocols. In *ESORICS 2007*, volume 4734 of *LNCS*, pages 422–437, Springer, 2007.
8. D. Dolev and A.C. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
9. J.A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *CRYPTO'99*, volume 1666 of *LNCS*, pages 449–466. Springer-Verlag, 1999.
10. D. Kähler and R. Küsters. Constraint Solving for Contract-Signing Protocols. In *CONCUR 2005*, volume 3653 of *LNCS*, pages 233–247. Springer-Verlag, 2005.
11. D. Kähler, R. Küsters, and Th. Wilke. Deciding Properties of Contract-Signing Protocols. In *STACS 2005*, volume 3404 of *LNCS*, pages 158–169. Springer-Verlag, 2005.
12. D. Kähler, R. Küsters, and Th. Wilke. A Dolev-Yao-based Definition of Abuse-free Protocols. In *ICALP 2006*, volume 4052 of *LNCS*, pages 95–106. Springer-Verlag, 2006.
13. S. Kremer and J.-F. Raskin. Game analysis of abuse-free contract signing. In *CSFW 2002*,

14. R. Küsters, H. Schnoor, and T. Truderung. A Formal Definition of Online Abusefreeness. *Technical Report*, University of Trier, 2010. Available at `http://infsec.uni-trier.de/publications.html`.
15. R. Küsters and T. Truderung. An Epistemic Approach to Coercion-Resistance for Electronic Voting Protocols In *Security and Privacy 2009*, pages 251–266. IEEE Computer Society, 2009.
16. 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.
17. G. Wang. An Abuse-Free Fair Contract-Signing Protocol Based on the RSA Signature *IEEE Transactions on Information Forensics and Security*, 5(1):158–168, 2010.