Compositional Analysis of Contract Signing Protocols

Michael Backes [a] , Anupam Datta [b] , Ante Derek [b] , John C. Mitchell [b] and Mathieu Turuani [c]

[a] *Saarland University*

[b] *Stanford University*

[c] *LORIA-INRIA Nancy*

Abstract

We develop a general method for proving properties of contract-signing protocols using a specialized protocol logic. The method is applied to the Asokan-Shoup-Waidner and the Garay-Jacobson-MacKenzie protocols. Our method offers certain advantages over previous analysis techniques. First, it is compositional: the security guarantees are proved by combining the independent proofs for the three subprotocols of each protocol. Second, the formal proofs are carried out in a "template" form, which gives us a reusable proof that may be instantiated for the two example protocols, as well as for other protocols with the same arrangement of messages. Third, the proofs follow the design intuition. In particular, in proving game-theoretic properties like *fairness*, we demonstrate success for the specific strategy that the protocol designer had in mind, instead of nonconstructively proving that a strategy exists. Finally, our logical proofs demonstrate security when an unbounded number of sessions are executed in parallel.

1 Introduction

Contract-signing protocols allow two or more parties to exchange signatures fairly, so that no party receives a signed contract unless all parties are able to do so. While there are no fixed-round fair two-party protocols [14,22], it is possible for two parties to exchange signatures optimistically. In optimistic

Email addresses: backes@cs.uni-sb.de (Michael Backes), danupam@cs.stanford.edu (Anupam Datta), aderek@cs.stanford.edu (Ante Derek), jcm@cs.stanford.edu (John C. Mitchell), turuani@loria.fr (Mathieu Turuani).

protocols, the two parties may exchange signatures if circumstances are favorable, but if either party chooses, they may ask a trusted third party to intervene and either complete the exchange or refuse to complete the exchange for either party. Some of the history of optimistic contract signing is summarized in [17].

Several methods have been used to analyze contract-signing protocols and either find errors or suggest their absence. In one previous study [23], which considers the same two protocols explored in this paper, the automated finite-state enumeration tool Murφ is used to find errors or anomalies in both protocols. Game-theoretic concepts and the MOCHA branching-time temporal logic model checker are used for analysis in [18,4], while another previous study [3] uses unformalized inductive arguments about a symbolic model of protocol execution. Some negative results about optimistic contract-signing protocols and the ability to achieve "abuse freeness" (a property discussed later in this paper) are proved in [5], which uses unformalized game tree arguments. These previous studies only consider a bounded number of participants (usually an initiator, a responder, and a trusted third party) or involve arguments about an often confusing number of cases. One reason that proving properties of optimistic contract signing protocols is difficult is that there are typically three sub-protocols, one allowing an optimistic exchange to proceed to completion, one allowing a dissatisfied participant to abort the exchange, and one allowing either signer to ask the third party to complete the exchange. Some appreciation for the inherent difficulty may be gained by reading the proof in [15] which, although otherwise rigorous, overlooks the case leading to an error discovered by automated analysis and reported in [23].

In this paper, we develop a method for reasoning about contract-signing protocols using a protocol security logic that was originally intended for authentication protocols. Surprisingly, the logic proves appropriate to the task, requiring only a minor modification to accommodate the if-then-else behavior of the trusted third party. In addition, we find that a direct argument establishes correctness of a family of related protocols, without limitation on the number of additional protocol sessions that may run in parallel. (The reason why parallel sessions are important is that additional sessions may provide alternate sources of messages signed by one of the principals.) The formal proof proceeds along direct, intuitive lines and is carried out in a "template" form that may be instantiated to provide correctness proofs for two standard protocols and protocol variants that use the same arrangement of messages. In addition, it is not necessary to consider interleaving of actions from different subprotocols, since the design of the protocol logic allows properties of the entire protocol to be proved compositionally from independent proofs for the *exchange, abort,* and *resolve* subprotocols of each contract-signing protocol.

Protocol Composition Logic (PCL) [7,6,13] uses a modal operator similar to

Floyd-Hoare logic. Intuitively, the formula $\psi\,[P]_X\,\varphi$ means that if ψ is true at some point in the execution of a protocol (in the presence of a malicious attacker), then φ will be true after agent X performs the sequence P of actions. The pre- and post-conditions may describe actions taken by various principals and characterize the information that is available to or hidden from them. PCL has been used for a variety of case studies, including proofs of correctness for the IEEE 802.11i wireless networking standard [16], which includes SSL/TLS. While PCL was originally formulated over a symbolic model of computation (used in this paper), we have also developed a computational interpretation [11] that considers arbitrary probabilistic polynomial-time attacks. An aspect of the logic that is particulary relevant to the present paper is explored in [9], where the idea of proving correctness of protocol templates is explained. In the template method, a protocol template is an abstract protocol containing function variables for some of the operations used to construct messages. Correctness of a protocol template may be established under certain assumptions about these function variables. Then, a proof for an actual protocol is obtained by replacing the function variables with combinations of operations that satisfy the proof assumptions. We follow this method for a family of contract-signing protocols, establishing correctness of the Asokan-Shoup-Waidner [1,2] and Garay-Jacobson-MacKenzie [15] protocols by instantiation. Although the formal proofs reflect the intricacies associated with any formal logic, the proofs seem to be direct encodings of natural lines of argument. In addition to compositional reasoning about the combined properties of three independent subprotocols, the protocol logic does not require any explicit reasoning about the possible behavior of any dishonest party, since the axioms and inference rules are sound for any hostile environment. Furthermore, although the formal proofs in this paper have been conducted manually, they should be accessible to automated proof techniques; we leave this as future work.

We prove fairness by explicitly showing that if participant A takes a certain number of steps, then if the opposing party B has a contract, party A has one as well. The actions involved in this certain number of steps depend on whether A is the protocol initiator or responder, and the state reached so far. In effect, this form of argument shows that A has a strategy to obtain a contract from any state by explicitly presenting the strategy. Further, these strategies are the natural ones inherent in the protocol design. However, the logic proves unsuitable for showing directly that it is possible to complete these steps - that is a modeling assumption that remains outside the formalism. Further, the safety-oriented logic seems less adept for non-trace-based properties such as abuse freeness than game-theoretic approaches. Nonetheless, these axiomatic, general proofs for unbounded runs offer additional validation of optimistic contract signing protocols not readily available through previous approaches.

The rest of this paper is organized as follows. Section 2 describes the Asokan-

Shoup-Waidner [1,2] and Garay-Jacobson-MacKenzie [15] protocols. Section 3 describes the protocol description language and logic and sketches the extensions of the logic required to reason about these protocols. A summary of the semantics of the logic and the proof system used in the axiomatic proofs in this paper appear in Appendix A. Section 4 presents the analysis of the ASW protocol, emphasizing the compositional proof method. Complete formal proofs are in Appendix B. Section 5 describes and proves properties of the template for optimistic contract signing protocols of which ASW and GJM are instances. Finally, Section 6 presents our conclusions.

2 Contract Signing Protocols

In this section, we describe two contract signing protocols. The description of each protocol is informal, using the "arrows and messages" notation that is common in the security protocol literature. Since this notation is not sufficient for rigorous protocol analysis, more formal characterizations of the protocol actions appear in section 4 and 5, using the protocol programming notation given in section 3. Further discussion of the design, intended properties, and limitations of both protocols may be found in the original presentations [1,2,15] and previous analyses of optimistic contract-signing protocols [3–5,18,23].

2.1 Asokan-Shoup-Waidner Protocol

The protocol of Asokan, Shoup, and Waidner (called the *ASW protocol* henceforth) [1,2] consists of three interdependent subprotocols: *exchange*, *abort*, and *resolve*. The two main parties, an *originator* O and a *responder* R, generally start the exchange by following the *exchange* subprotocol. If both O and R are honest and there is no interference or message loss on the network, each obtains a valid contract upon the completion of the exchange subprotocol. The originator O also has the option of asking the trusted third party T to abort an exchange that O has initiated. To do so, O executes the *abort* subprotocol with T. Finally, both O and R may each request that T resolve an exchange that has not been completed. Intuitively, this is important because there may be network delays or a lack of response from a devious party. After receiving the initial message of the exchange protocol, O or R may execute the *resolve* subprotocol with T. While the conditions for executing the resolve protocol are not specified as part of the protocol definition, the resolve protocol allows the originator or responder to complete the protocol if either times out waiting for the other.

The *exchange* protocol consists of the following four messages:

$$
\begin{aligned}
O \to R \qquad & me_1 = Sig_O\{V_O, V_R, T, text, Hash(N_O)\} \\
R \to O \qquad & me_2 = Sig_R\{me_1, Hash(N_R)\} \\
O \to R \qquad & me_3 = N_O \\
R \to O \qquad & me_4 = N_R
\end{aligned}
$$

In the first message, the originator O sends responder R a message consisting of data V_O, V_R, T identifying the originator, responder, and trusted third party (respectively), the text of the contract, and a hash $Hash(N_O)$ of a nonce (random number) N_O chosen by the originator, all signed using the originator's private signing key. The responder returns a signed message comprising the first message and a hash $Hash(N_R)$ of a nonce N_R chosen by the responder. Intuitively, each message indicates the willingness to be bound by the contract, but the exchange is not considered complete until both nonces are known. The third and fourth messages exchange these nonces, so that both parties have a complete contract.

In the *abort* protocol, the originator may ask the third party T not to issue a replacement contract, using the following message and response:

$$
\begin{aligned}
O \to T \qquad & ma_1 = Sig_O\{aborted, me_1\} \\
T \to O \qquad & ma_2 = \text{Has } me_1 \text{ been resolved already?} \\
& \qquad \text{Yes}: Sig_T\{me_1, me_2\} \\
& \qquad \text{No}: \ Sig_T\{aborted, ma_1\} \\
& \qquad\qquad aborted := \text{true}
\end{aligned}
$$

In the first message, the originator sends T a signed message indicating a request to abort a run of the contract-signing protocol, together with the contents of the first message of this run. If the third party has not already received a request to resolve this protocol (see next subprotocol), then the third party can confirm this by sending a signed message indicating that the exchange has been aborted. Otherwise, the trusted third party must have received the first and second messages (in order to resolve the contract) and in this case the third party sends the originator a replacement contract consisting of the third party's signature on the first two messages of the exchange protocol.

One confusing aspect of the abort protocol is the meaning of the abort message. Since a dishonest originator can send an abort message after completing the exchange protocol (in which both the originator and responder receive a contract), the third party may issue an abort message even if both parties have a contract. Therefore, an abort message from the third party does not mean that neither party has a contract. Instead, when the third party behaves

honestly, the abort message means that neither party can have a replacement contract issued by the trusted third party.

The *resolve* protocol may be used by either the originator or the responder to obtain an alternate form of contract. Intuitively, this is useful if the originator and/or responder have progressed far enough with the exchange protocol to be committed to the contract, but have not received the nonce from the other party. The two messages of the resolve protocol are written below, as if the protocol is initiated by the responder R:

$$
\begin{array}{ll}
R \rightarrow T & mr_1 = \{me_1, me_2\} \\
T \rightarrow R & mr_2 = \text{Has } me_1 \text{ been aborted already?} \\
 & \qquad \text{Yes}: \; Sig_T\{aborted, ma_1\} \\
 & \qquad \text{No}: \; Sig_T\{me_1, me_2\} \\
 & \qquad\qquad resolved := \text{true}
\end{array}
$$

The protocol definition in [2] provides two forms of contract:

$$
\begin{array}{ll}
\{me_1, N_O, me_2, N_R\} & \text{(standard contract)} \\
Sig_T\{me_1, me_2\} & \text{(replacement contract)}
\end{array}
$$

As suggested above, the first form is the contract that both parties will have after the exchange subprotocol is successfully completed. The second is the form of contract issued by the third party, on request.

This protocol was designed to provide *fairness* to both parties and *trusted third party accountability*. Fairness is the property that either both parties may each obtain a contract (of one form or the other), or neither may obtain a contract. Third party accountability is a property involving evidence to show misbehavior by any third party that acts dishonestly. In the case that the third party behaves dishonestly, such as by issuing both an abort message and a replacement message, the protocol design should guarantee that a collection of network messages will show definitively that the third party has misbehaved.

2.2 Garay-Jakobsson-MacKenzie Protocol

The protocol of Garay, Jakobsson, and MacKenzie (called the *GJM protocol* henceforth) [15] is closely related to the ASW protocol. Both protocols provide a 4-step *exchange* subprotocol and two-step *abort* and *resolve* subprotocols. Although the ASW and GJM protocols have similar structure, the contents of the messages differ. Unlike the ASW protocol, the GJM protocol

is designed to guarantee *abuse-freeness* in addition to fairness and third party accountability. These properties are formally stated and proved in Section 5. Intuitively, *abuse-freeness* involves the inability of any party to prove control of the outcome to an outsider. More specifically, the originator O controls the outcome, at a certain point in execution of the contract-signing protocol, if O can either choose to give both parties a contract or choose to prevent both parties from obtaining a contract. If O controls the outcome, and can demonstrate this to an outsider, then O might be able to "abuse" R by using R's commitment to get a better contract from an outsider. For example, if O can show an alternative buyer that R will pay a certain amount for an item O has offered for sale, then the alternative buyer may offer more, and O can abort the contract to sell the item to R.

The GJM protocol relies on the cryptographic primitive called *private contract signature* (PCS). We write $PCS_O(m, R, T)$ for party O's private contract signature of text m for party R (known as the *designated verifier*) with respect to third party T. The main properties of PCS are as follows: (a) $PCS_O(m, R, T)$ can be verified by R like a conventional signature; (b) $PCS_O(m, R, T)$ can be feasibly computed by either O, or R, but nobody else; (c) $PCS_O(m, R, T)$ can be converted into a conventional signature by either O, or T, but nobody else, including R. For the purposes of this study, we focus on the *third-party accountable* version of PCS, in which the converted signatures produced by O and T can be distinguished. We will call them $Sig_O(m)$ and $T\_Sig_O(m)$, respectively. Unlike PCS, converted signatures are universally verifiable by anybody in possession of the correct signature verification key. An efficient discrete log-based PCS scheme is presented in [15].

This protocol was designed to provide *fairness* to both parties and *trusted third party accountability*. The formal presentation of the protocol and its properties are in Section 5.

The GJM *exchange* protocol is similar to the ASW exchange protocol, using private contract signatures instead of regular digital signatures in the first two messages, and with actual signatures on the contract m exchanged in the last two messages:

$$
\begin{aligned}
O \rightarrow R \qquad & me_1 = PCS_O(m, R, T) \\
R \rightarrow O \qquad & me_2 = PCS_R(m, O, T) \\
O \rightarrow R \qquad & me_3 = Sig_O(m) \\
R \rightarrow O \qquad & me_4 = Sig_R(m)
\end{aligned}
$$

The GJM *abort* protocol is similar in spirit to the ASW abort protocol de-

scribed above.

$O \rightarrow T$ $\quad ma_1 = Sig_O(m, O, R, abort)$

$T \rightarrow O$ $\quad ma_2 =$ Has O or R resolved already?

Yes : $Sig_R(m)$ if R has resolved, or

$T\_Sig_R(m)$ if O has resolved

No : $Sig_T(ma_1)$

$aborted :=$ true

When T receives an abort request, T checks its permanent database of past actions to decide how to proceed. If T has not previously been requested to resolve this instance of the protocol, T marks m as aborted in its permanent database and sends an abort token to O. If m is already marked as resolved, this means that T has previously resolved this exchange in response to an earlier request. As a result of the resolution procedure (described below), honest T must have obtained both O's and R's universally-verifiable signatures of m. Therefore, in response to O's abort request, T forwards O either $Sig_R(m)$ or $T\_Sig_R(m)$, either of which can serve as a proof that R indeed signed m.

The GJM *resolve* protocol is also similar in spirit to the ASW resolve protocol described above.

$R \rightarrow T$ $\quad mr_1 = PCS_O(m, R, T), Sig_R(m)$

$T \rightarrow R$ $\quad mr_2 =$ Has O aborted already?

Yes : Send $Sig_T(Sig_O(m, O, R, abort))$

No : Has O resolved already?

Yes : Send $Sig_O(m)$

No : Store $Sig_R(m)$

Convert $PCS_O(m, R, T)$ to $T\_Sig_O(m)$

Send $T\_Sig_O(m)$

$resolved :=$ true

Either party may request that T resolve the exchange. In order to do so, the party must possess the other party's PCS of the contract (with T as the designated third party), and submit it to T along with its own universally-verifiable signature of the contract. Therefore, R can send a resolve request at any time after receiving me_1, and O can do so at any time after receiving me_2. When T receives a resolve request, it checks whether the contract is already marked as aborted. If it is, T replies with the abort token. If the contract has been resolved by the other party, T replies with that party's signature. Finally, if the contract has been neither aborted, nor resolved by the other party, T

converts PCS into a universally-verifiable signature, sends it to the requestor, and stores the requestor's own signature in its private database.

3 Methodology

3.1 Protocol notation

We use a simple "protocol programming language" based on [13,7,6] to represent a protocol by a set of roles, such as "Originator", "Responder" or "Third-Party", each role specifying a sequence of actions to be executed by a honest participant. The syntax of terms and actions is given in Table 1.

We use $\hat{X}, \hat{Y}, \ldots$ as *names* for protocol participants. Since a particular participant might be involved in more than one session at a time, we will give unique names to sessions and use a pair $(\hat{X}, s)$ to designate a particular *thread* being executed by $\hat{X}$. Although this notational convention may seem strange initially, we generally use a Roman capital letter X to refer to an arbitrary thread of agent $\hat{X}$.

Terms are expressions for messages and their parts, including variables x, keys K, threads X, nonces n, and pairs $\langle t, t \rangle$. We write $SIG_K\{|t|\}$ for term t signed with key K, and $HASH\{|t|\}$ for the hash of t. Since the protocols we consider assume a public-key infrastructure, we may write a thread name for its associated key. The relation $m \subseteq m'$ indicates that m is a subterm of $m' \in t$.

Actions include nonce generation, pattern matching, and communication. The action `new` n chooses a new nonce n different from all other values associated with protocol execution to this point. The action `match` t_1/t_2 matches a term t_1 against a term t_2 representing a pattern. A variable occurring in the pattern t_2 will thereafter be replaced by the appropriate subterm of t_1. For example, `match` $x/SIG_K\{|y|\}$ will verify that x, which might have been a message received through communication, is a valid signature with key K. In this case, y will subsequently refer to the message that is signed in x. If x is not a valid signature with the correct key, then this action cannot occur, and subsequent actions in the remainder of the protocol role will not occur either. Send and receive actions, `send` t and `receive` x are largely straightforward.

An action not considered in previous papers on protocol composition logic is the "if" construct in the spirit of Dijkstra's guarded commands that we add here. This construct is used to express conditional behavior of protocol participants. This construct behaves like a generalization of pattern matching.

Terms

$$t ::= x \mid K \mid X \mid n \mid \langle t, t \rangle \mid SIG_K\{|t|\} \mid HASH\{|t|\}$$

Actions

$$\texttt{a} ::= \texttt{new}\ n \mid \texttt{match}\ t/t \mid \texttt{send}\ t \mid \texttt{receive}\ x \mid \texttt{if}\ t\quad t_1 : P_1; \ldots t_n : P_n;\ \texttt{fi}$$

Table 1
Syntax of protocol terms and actions

If i is the lowest number such that t matches t_i, then $\texttt{if}\ t\quad t_1 : P_1; \ldots t_n : P_n;\ \texttt{fi}$ simplifies to $\texttt{match}\ t/t_i : P_i$; In other words, this conditional action performs P_i iff t is equal to t_i.

The operational semantics of the protocol programming language is defined in the style of process calculus (see [6]). Execution begins from an *initial configuration*. The initial configuration of a protocol Q is determined by: (1) A set of principals, some of which are designated as honest; (2) A multiset of roles (programs determining a sequence of actions) constructed by assigning roles of Q to threads of honest principals; (3) A collection of intruder actions, which may use keys of dishonest principals; (4) A finite number of buffer actions, enough to accommodate every send action by honest threads and the intruder threads. A *run* R is a sequence of reaction steps from the initial configuration, subject to the constraint that every send/receive reaction step happens between some buffer action and some (nonbuffer) thread. A reaction step is either a local action executed by a principal or a communication action which involves a send action by some thread and a receive action by another.

3.2 Protocol Logic

The basic protocol logic and proof system are developed in [12,7,8,10], with [6] providing a relatively succinct presentation of a consistent form that unfortunately differs from more recent papers in some syntactic ways. A summary of the relevant portions of the semantics and proof system appears in Appendix A.

The formulas of the logic are given by the grammar in Table 2. Here, t and X denote a term and a thread respectively. As mentioned above, we use the word *thread* to refer to a principal executing an instance of a role, and we use X to refer to a thread belonging to principal $\hat{X}$. We use ϕ and ψ to indicate predicate formulas, and m to indicate a generic term we call a "message".

Most protocol proofs use formulas of the form $\theta[P]_X\phi$, which means that after actions P are executed in thread X, starting from a state where formula θ is true, formula ϕ is true about the resulting state of X. Here are the informal

Action formulas

$\mathsf{a} ::= \mathsf{Send}(X,t) \mid \mathsf{Receive}(X,t) \mid \mathsf{New}(X,t) \mid \mathsf{Decrypt}(X,t) \mid \mathsf{Verify}(X,t) \mid \mathsf{Start}(X)$

Formulas

$\phi ::= \mathsf{a} \mid \mathsf{Has}(X,t) \mid \mathsf{Computes}(X,t) \mid \mathsf{Honest}(\hat{X}) \mid \phi \wedge \phi \mid \neg\phi \mid \exists x.\phi \mid$

Modal form

$\Psi ::= \phi\ [P]_X\ \phi$

Table 2
Syntax of the logic

interpretations of the predicates:

$\mathsf{Has}(X,x)$ means principal $\hat{X}$ possesses information x in the thread X. This is "possess" in the limited sense of having either generated the data or received it in the clear or received it under encryption where the decryption key is known.

$\mathsf{Send}(X,m)$ means principal $\hat{X}$ sends message m in the thread X.

$\mathsf{Receive}(X,m)$, $\mathsf{New}(X,t)$, $\mathsf{Decrypt}(X,t)$, $\mathsf{Verify}(X,t)$ similarly mean that receive, new, decrypt and signature verification actions occur.

$\mathsf{Honest}(\hat{X})$ means the actions of principal $\hat{X}$ in the current run are precisely an interleaving of initial segments of traces of a set of roles of the protocol. In other words, $\hat{X}$ assumes some set of roles and does exactly the actions prescribed by them.

$\mathsf{Computes}(X,m)$ means that a principal $\hat{X}$ possesses enough information in thread X to build term m of some type. For example, a principal can possess an encrypted message if he received it in the past as a part of some message or if he possesses the plaintext and the encryption key. Computes is used to describe the latter case.

$\mathsf{Start}(X)$ means that the thread X did not execute any actions in the past.

4 Analysis of the ASW Protocol

The ASW protocol [1,2] consists of three interdependent subprotocols: *exchange*, *abort*, and *resolve*. We first show in Section 4.1 how to model these protocols as programs. Section 4.2 discusses the compositional proof method used in proving the security properties of the protocol. Section 4.3 contains the formal definitions and proof sketches of the fairness and accountability properties. The complete formal proofs are in Appendix B. We believe that the proof method illustrated by this application will be useful for analyzing similar properties of related protocols.

We emphasize two high-level aspects of this method which distinguishes it from existing analysis techniques. First, it is compositional: the security guarantees offered by the ASW protocol are proved by combining independent guarantees offered by the *exchange*, *abort*, and *resolve* subprotocols. Second, the proofs follow the design intuition. In particular, in the fairness proofs, we demonstrate that the appropriate strategy for a party to obtain a contract (via the abort/resolve protocols) depending on its local state, actually works. For example, if after sending the first message, the initiator executes the abort protocol, then he gets the contract if his peer has the contract. The fact that we prove a specific strategy works as opposed to proving one exists distinguishes us from prior game-theoretic analyses [4]. Also, it seems useful to have analysis techniques which can take advantage of and inform protocol design principles.

4.1 Modelling Protocol Parties

The roles of the *exchange* subprotocol are given in Figure 1, with one program **ExchangeInit** for the initiator of the protocol and one program **ExchangeResp** for the responder. A role consists of static input parameters, and a list of actions to be executed. For example, **ExchangeInit** represents the program for initiator $\hat{X}$ starting a protocol with the responder $\hat{Y}$ and trusted third party $\hat{T}$ with contract *text*. The intuitive reading of the sequence of actions is: generate a new nonce, send a signature representing a commitment to the contract, receive a message, check that it is a valid commitment for $\hat{Y}$, release the nonce, receive a message and check that it is a valid decommitment corresponding to the responders commitment.

The ASW protocol provides for two kinds of contracts. The first one is called a *standard* contract. Standard contracts are obtained if the execution of the protocol successfully finishes without any party aborting or resolving the protocol. They are formally defined as follows.

$$s(\hat{X},\hat{Y},\hat{T},text,x,y) \equiv SIG_{\hat{X}}\{|\hat{X},\hat{Y},\hat{T},text,HASH\{|x|\}|\},x,$$
$$SIG_{\hat{Y}}\{|SIG_{\hat{X}}\{|\hat{X},\hat{Y},\hat{T},text,HASH\{|x|\}|\},HASH\{|y|\}|\},y$$

The second kind is called a *replacement contract*. It is always built by the trusted third party to resolve a protocol.

$$r(\hat{X},\hat{Y},\hat{T},text,w,z) \equiv SIG_T\{|SIG_{\hat{X}}\{|\hat{X},\hat{Y},\hat{T},text,w|\},$$
$$SIG_{\hat{Y}}\{|SIG_{\hat{X}}\{|\hat{X},\hat{Y},\hat{T},text,w|\},z|\}|\}$$

To improve readability, in the subsequent proofs we often write s and r instead of $s(\hat{A},\hat{B},\hat{T},text,x,y)$ and $r(\hat{A},\hat{B},\hat{T},text,HASH\{|w|\},z)$, respectively. In the following, we often have to reason about the messages that are exchanged during the protocol execution. Especially the first and second message will be

ExchangeInit $\equiv (\hat{X}, \hat{Y}, \hat{T}, text)[$

$\texttt{new}\ x;$

$\texttt{send}\ \hat{X}, \hat{Y}, SIG_{\hat{X}}\{|\hat{X}, \hat{Y}, \hat{T}, text, HASH\{|x|\}|\};$

$\texttt{receive}\ \hat{Y}, \hat{X}, z;$

$\texttt{match}\ z/SIG_{\hat{Y}}\{|SIG_{\hat{X}}\{|\hat{X}, \hat{Y}, \hat{T}, text, HASH\{|x|\}|\}, w|\};$

$\texttt{send}\ \hat{X}, \hat{Y}, x;$

$\texttt{receive}\ \hat{Y}, \hat{X}, y;$

$\texttt{match}\ HASH\{|y|\}/w;]_X$

ExchangeResp $\equiv (\hat{X}, \hat{Y}, \hat{T}, text)[$

$\texttt{receive}\ \hat{Y}, \hat{X}, z;$

$\texttt{match}\ z/SIG_{\hat{Y}}\{|\hat{Y}, \hat{X}, \hat{T}, text, y|\};$

$\texttt{new}\ x;$

$\texttt{send}\ \hat{X}, \hat{Y}, SIG_{\hat{X}}\{|z, HASH\{|x|\}|\};$

$\texttt{receive}\ \hat{Y}, \hat{X}, w;$

$\texttt{match}\ HASH\{|w|\}/y;$

$\texttt{send}\ \hat{X}, \hat{Y}, x;]_X$

Abort $\equiv (\hat{X}, \hat{Y}, \hat{T}, msg1)[$

$\texttt{send}\ \hat{X}, \hat{T}, SIG_{\hat{X}}\{|Abort, msg1|\}$

$\texttt{receive}\ \hat{T}, \hat{X}, z;$

$\texttt{if}\ z$

$SIG_{\hat{T}}\{|Aborted, msg1|\}$:;

$r(\hat{A}, \hat{B}, \hat{T}, text, HASH\{|x|\}, w)$:;

$\texttt{fi}\ ]_X$

Resolve $\equiv (\hat{X}, \hat{Y}, \hat{T}, msg1, msg2)[$

$\texttt{send}\ \hat{X}, \hat{T}, msg1, msg2;$

$\texttt{receive}\ \hat{T}, \hat{X}, z;$

$\texttt{if}\ z$

$SIG_{\hat{T}}\{|Aborted, msg1|\}$:;

$r(\hat{A}, \hat{B}, \hat{T}, text, HASH\{|x|\}, w)$:;

$\texttt{fi}\ ]_X$

Fig. 1. Roles of the ASW protocol

important as they grant the respective parties the ability to resolve a protocol execution. For reasons of readability, we introduce syntactic shortcuts msg_1 and msg_2 for these messages, i.e.,

$$msg_1 \equiv SIG_{\hat{A}}\{|\hat{A}, \hat{B}, \hat{T}, text, HASH\{|x|\}|\}$$
$$msg_2 \equiv SIG_{\hat{B}}\{|SIG_{\hat{A}}\{|\hat{A}, \hat{B}, \hat{T}, text, HASH\{|x|\}|\}, HASH\{|y|\}|\}.$$

The *abort* and *resolve* subprotocols for the initiator and the responder are given in Figure 1. We use the "if" construct to model the fact that agents do not know in advance which of the two possible responses they will receive from the trusted third party $\hat{T}$.

In analyzing the ASW protocol, we do not model the program of the trusted

third party explicitly. Instead we capture its desired behavior by a set of logical formulas—Γ_T^1 and Γ_T^2 below. Using the extensions of the logic presented in this paper, it is easy to write down the program for the trusted third party and prove that these logical formulas represent properties of its protocol. Since the proofs are similar to other proofs presented in this paper, we omit them here.

$$\begin{aligned}\Gamma_T^1 =&\{\mathsf{Honest}(\hat{T}) \wedge \mathsf{Send}(T, \hat{T}, \hat{B}, r(\hat{A}, \hat{B}, \hat{T}, text, x, z)) \supset \\ &\neg\mathsf{Send}(T, \hat{T}, \hat{A}, SIG_{\hat{T}}\{\!|Aborted, msg_1|\!\})\} \\ \Gamma_T^2 =&\{\mathsf{Honest}(\hat{T}) \wedge \mathsf{Send}(T, \hat{T}, \hat{A}, SIG_{\hat{T}}\{\!|Aborted, msg_1|\!\}) \supset \\ &\mathsf{Receive}(T, \hat{A}, \hat{T}, SIG_{\hat{A}}\{\!|Abort, msg_1|\!\})\}\end{aligned}$$

Γ_T^1 says that $\hat{T}$ will never issue both a replacement contract and the abort token. Γ_T^2 says that $\hat{T}$ will never send an abort token unless A has initiated the abort subprotocol for the corresponding commitment.

4.2 Compositional proof method

In this section, we sketch the method used to prove properties of the protocol. In the ASW protocol, there is more than one intended run. For example, after sending the first message, the initiator can decide not to wait for the response but to run the abort subprotocol instead. Using the protocol logic, we are able to analyze the components of the ASW protocol independently, and combine the proofs using the composition theorems presented in [7,6]. We focus primarily on the guarantees for the initiator in the protocol.

Runs of the ASW protocol There are three possible execution scenarios for the initiator in the ASW protocol. The initiator can complete the exchange subprotocol; complete the resolve subprotocol after sending the third message of the exchange subprotocol; or complete the abort protocol after sending the first message of the exchange subprotocol (Figure 2). The design intent was that each of these three combinations should result in the initiator obtaining a valid contract whenever the responder already has one. We will use $\mathbf{ExchangeInit^3}(\hat{A}, \hat{B}, \hat{T}, text)$ to denote the prefix of the initiator role in the protocol up to and including the second `send` action (send of the third message in the protocol) and $\mathbf{ExchangeInit^1}(\hat{A}, \hat{B}, \hat{T}, text)$ to denote the prefix up to and including the first `send` action.

Formulas of the logic and sequential composition Most logical statements that we use are of the form $\Gamma \vdash \phi[\mathbf{P}]_A\theta$. The intuitive reading of such statement is: "Given that the set of assumptions Γ holds in every state, and a thread A has finished executing the program $\mathbf{P}$ starting in a state where ϕ holds, then in the resulting state θ holds."

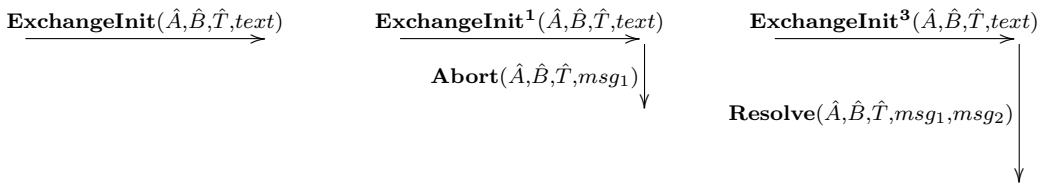

Fig. 2. Possible runs for the initiator in the ASW protocol

In this paper, Γ will typically contain a set of assumptions about the behavior of the trusted third party T which can be later discharged by analyzing T's program. For example, one of the assumptions we use is that T never sends a replacement contract if it has issued the abort token in the past.

We will combine statements about different subprotocols using *sequential composition* of the roles. Here we state a variant of the theorem from [6] that we use for this purpose. The precise form in which this theorem will be employed will become clearer in the next section.

Theorem 1 (Sequential composition) *For sets of assumptions* Γ_1 *and* Γ_2*, programs* $\mathbf{P}$ *and* $\mathbf{Q}$*, and formulas* ϕ*,* θ *and* ψ *if* $\Gamma_1 \vdash \phi[\mathbf{P}]_X\theta$ *and* $\Gamma_2 \vdash \theta[\mathbf{Q}]_X\psi$ *than* $\Gamma_1 \cup \Gamma_2 \vdash \phi[\mathbf{P};\mathbf{Q}]\psi$*, where* $\mathbf{P};\mathbf{Q}$ *is a sequential composition of programs* $\mathbf{P}$ *and* $\mathbf{Q}$*.*

4.3 Proving protocol properties

In this section, we show how to express and prove the desired properties in the underlying logic using the described proof method. Complete formal proofs are given in Appendix B. Here, we sketch the proof structure and emphasize the crucial steps.

4.3.1 Fairness

We start with the fairness property of the protocol. Informally, fairness means that after the protocol has been successfully completed, either both parties have a signed contract or neither does. In our model, fairness is expressed using a set of logical formulas. As described in the previous section, we look separately at the three possible scenarios for the initiator to complete the protocol.

Initiator completes the exchange subprotocol Formula ϕ_0 states that the initiator A has a valid contract after the successful execution of the ex-

change protocol. This is the optimistic part of the protocol.

$$\begin{aligned}\phi_0 \equiv & \mathsf{Start}(A) \\ & [\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)]_A \\ & \mathsf{Has}(A, s(\hat{A}, \hat{B}, \hat{T}, text, x, y))\end{aligned}$$

The formal proof of this property is given in Appendix B. We show that:

$$\vdash \phi_0 \tag{1}$$

Therefore, in this scenario fairness holds without any assumptions about the behavior of the trusted third party or the responder in the protocol.

Initiator runs the abort protocol If A started the protocol as the initiator but did not complete it, we want to show that whenever some other party has a valid contract, then A will get the replacement contract if it executes the abort subprotocol after sending the first message. This part of the analysis is done using the compositional proof method. First, we identify a sufficient precondition that needs to hold in order for the initiator to get the contract after executing the abort subprotocol; then we show that the precondition is satisfied if the initiator only executed his role upto the first `send` action.

A sufficient precondition for this to hold is that A's nonce x has been kept secret, i.e.

$$\theta_1 = \mathsf{HasAlone}(A, x)$$

where $\mathsf{HasAlone}(X, t)$ is defined by $\mathsf{HasAlone}(X, t) \equiv \mathsf{Has}(X, t) \wedge (\mathsf{Has}(Y, t) \supset X = Y$. It is easy to verify that θ_1 holds in the state where A has only sent the first message of the protocol, since this message only contains the hash of x.

The property of the abort subprotocol needed for fairness is given below. Informally, it states that if at some state θ_1 holds then, after executing the abort subprotocol, if some thread X possesses any contract (standard or replacement) corresponding to A's nonce x and if T is honest, A will posses the replacement contract corresponding to the same nonce x.

$$\begin{aligned}\phi_1 \equiv & \theta_1 \\ & [\mathbf{Abort}(\hat{A}, \hat{B}, \hat{T}, msg_1)]_A \\ & ((\mathsf{Has}(X, s) \vee \mathsf{Has}(X, r)) \wedge \mathsf{Honest}(\hat{T})) \supset \mathsf{Has}(A, r)\end{aligned}$$

The formal proof of fairness in this scenario involves showing that θ_1 holds after A executed the first part of the exchange subprotocol, and that ϕ_1 holds as long as the trusted third party $\hat{T}$ behaves properly. The complete proof of

both statements are given in Appendix B.

$$\vdash \mathsf{Start}(A)[\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)^1]_A \theta_1 \quad (2)$$

$$\Gamma_T^1 \vdash \phi_1 \quad (3)$$

The fairness property in this case simply follows from the sequential composition theorem (Theorem 1). Recall that Γ_T^1 is a set of assumptions about the behavior of the trusted third party T defined in Section 4.1. Informally, Γ_T^1 says that $\hat{T}$ will never issue both a replacement contract and the abort token. Therefore Γ_T^1 is only going to hold if $\hat{T}$ is completely honest. A misbehaving $\hat{T}$ might otherwise cheat on A if A executed the abort protocol after receiving the first message.

Initiator runs the resolve protocol Finally, we want to show that if A has received the second message of the protocol, then it can obtain a valid contract by executing the resolve subprotocol, provided that it created the nonce x itself and did not send the abort message corresponding to that nonce in the past, i.e., for

$$\theta_2 = \mathsf{New}(A, x) \wedge \neg \mathsf{Send}(A, \hat{A}, \hat{T}, SIG_{\hat{A}}\{|Abort, msg_1|\})$$

we define ϕ_2 by

$$\begin{aligned} \phi_2 \equiv\ & \theta_2 \\ & [\mathbf{Resolve}(\hat{A}, \hat{B}, \hat{T}, msg_1, msg_2)]_A \\ & (\mathsf{Has}(X, s) \vee \mathsf{Has}(X, r) \wedge \\ & \mathsf{Honest}(\hat{T}) \wedge \mathsf{Honest}(\hat{A})) \supset \mathsf{Has}(A, r) \end{aligned}$$

It is easy to verify that θ_2 holds in the state where A has received the second message of the protocol and sent the third message of the protocol.

The formal proof of fairness in this scenario involves showing that θ_2 holds after A executed the first part of the exchange subprotocol, and that ϕ_2 holds as long as the trusted third party $\hat{T}$ behaves properly. The required fairness guarantee is obtained by the composition theorem as before. The complete proof of both statements are given in Appendix B.

$$\vdash \mathsf{Start}(A)[\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)^3]_A \theta_2 \quad (4)$$

$$\Gamma_T^2 \vdash \phi_2 \quad (5)$$

In this case, the assumption about the behavior of the trusted third party Γ_T^2 says that $\hat{T}$ will never send an abort token unless A has initiated the abort subprotocol for the corresponding commitment. This completes the proof for fairness for the initiator in the ASW protocol.

Discussion Notice that the property we prove in the optimistic part of the protocol is weaker than in the other two cases. Namely, we only show that the initiator has a contract corresponding to his nonce, it could be possible that the responder (or attacker) has obtained other contracts corresponding to the same initiator's nonce and different responder's nonce. As demonstrated in [23] that is indeed the case. We rediscovered the same attack when the proof of the stronger property failed. The following formula does *not* hold for the protocol:

$$\begin{aligned}&\mathsf{Start}(A)\\&[\mathbf{ExchangeInit}(\hat{A},\hat{B},\hat{T},text)]_A\\&\mathsf{Has}(X,s(\hat{A},\hat{B},\hat{T},text,x,w))\supset\mathsf{Has}(A,s(\hat{A},\hat{B},\hat{T},text,x,w))\end{aligned}$$

The interpretation of this formula is that no other thread X can have any other standard contract except the one obtained by A after executing the exchange subprotocol.

Fairness for the responder in the protocol A similar property can be shown for the responder in the ASW protocol. The proof structure is identical to the proof for the initiator. For brevity, we do not get into the details.

4.3.2 Accountability

Accountability means that if one of the parties gets cheated as a result of $\hat{T}$'s misbehavior, that it will be able to hold $\hat{T}$ accountable. More precisely, at the end of every run where an agent gets cheated, its trace together with a contract of the other party should provide non-repudiable evidence that $\hat{T}$ misbehaved.

The first step in the formalization of this property is to precisely define what it means for a set of terms to be a non-repudiable proof of $\hat{T}$'s misbehavior. One approach is to require that, assuming the correctness of $\hat{T}$ as specified by the set of formulas Γ, we can formally derive that if anyone possesses certain terms (typically involving $\hat{T}$'s signature), then $\mathsf{Honest}(\hat{T})$ does not hold. It is easy to prove that the replacement contract and the abort token corresponding to the same nonce constitute non-repudiable proofs that $\hat{T}$ misbehaved. Formally, for $\Gamma_T := \Gamma_T^1 \cup \Gamma_T^2$ we prove:

$$\begin{aligned}\Gamma_T \vdash &\mathsf{Has}(X, r(\hat{A},\hat{B},\hat{T},text,HASH\{\!|x|\!\},w))\wedge\\&\mathsf{Has}(A, SIG_{\hat{T}}\{\!|Aborted, msg_1|\!\})\supset\neg\mathsf{Honest}(\hat{T})\end{aligned}$$

Again we reason from the initiator's point of view and consider three scenarios.

Initiator completes the exchange subprotocol We already proved that A has a contract in that case, regardless of $\hat{T}$'s behavior and therefore A cannot get cheated in this case.

Initiator runs the abort protocol In this case, we prove two things. First of all, no one can have a standard contract. Secondly, after executing the abort subprotocol A will either get the abort token or the replacement contract. Therefore, if A gets cheated it has to be the case that some other party X has a replacement contract, while A has the abort token for the corresponding nonce. As explained above, these two terms are non-repudiable proofs that $\hat{T}$ misbehaved. We capture both properties with a single logical formula given below.

$$\begin{aligned}&\vdash \theta_1\\ &\quad [\mathbf{Abort}(\hat{A}, \hat{B}, \hat{T}, msg_1)]_A\\ &\quad \neg \mathsf{Has}(X, s) \wedge (\mathsf{Has}(A, r) \vee \mathsf{Has}(A, SIG_{\hat{T}}\{|Aborted, msg_1|\}))\end{aligned}$$

Initiator runs the resolve protocol This case is similar to the one above, and we omit the details.

4.3.3 Abuse-Freeness

Abuse-freeness means that no party can ever prove to the third party that it has the power to both enforce and cancel the contract. More precisely, a protocol is abuse-free for the initiator if in every state where the responder has publicly verifiable information that the initiator is bound to the contract it has to be that the responder is also bound to the contract.

Modelling the property that the responder has publicly verifiable information that the initiator is bound to the contract is beyond the scope of the logic. However, if we fix the set of terms $t_1, \ldots, t_n$ that we consider to constitute such information we can express abuse-freeness for the initiator in the following way: whenever a party X possess terms $t_1, \ldots, t_n$, the initiator has a strategy to obtain a contract. As pointed in the long version of [4], the definition of abuse-freeness that we are able to prove in the logic is strictly stronger than the standard definition of abuse-freeness that we first mentioned above.

For the ASW protocol, if we consider the signature in the first message as a proof that the initiator is bound to the contract, it is easy to see that the protocol does not provide this property for the initiator. In the logic, this is

reflected in the fact that the proof of the following formula fails:

$$\begin{aligned}\Gamma_T \vdash \theta_1 \wedge \mathsf{Has}(X, SIG_{\hat{A}}\{\!|\hat{A}, \hat{B}, \hat{T}, text, HASH\{\!|x|\!\}|\!\}\\ [\mathbf{Abort}(\hat{A}, \hat{B}, \hat{T}, msg_1)]_A\\ \mathsf{Honest}(\hat{T}) \supset \mathsf{Has}(A, r)\end{aligned}$$

5 Template for Optimistic Contract Signing Protocols

Both the ASW and GJM protocols consist of three interdependent subprotocols: *exchange*, *abort*, and *resolve*. The structure of these two protocols suggests a general pattern for two-party optimistic contract-signing protocols. Specifically, the *exchange* subprotocol proceeds in two stages. In the first stage, the two parties commit to the contract and in the second they open their commitment, in effect, ensuring that they are bound to the contract. Given this observation, it seems natural to ask if we could provide a unified representation and proofs for these two protocols and their properties. In this section, we answer this question in the affirmative.

5.1 Abstraction and Refinement Methodology

The concept of protocol templates and an abstraction-instantiation method using templates to develop unified proofs for related protocols is introduced in [9]. In order to make this paper self-contained, we reproduce the main ideas below.

Protocol Templates: A *protocol template* is a protocol that uses function variables. An example of an abstract challenge-response based authentication protocol using the informal trace notation is given below.

$$\begin{aligned}A &\rightarrow B : m\\ B &\rightarrow A : n, F\,(B, A, n, m)\\ A &\rightarrow B : G\,(A, B, m, n)\end{aligned}$$

Here, m and n are fresh nonces and F and G are function variables. Substituting cryptographic functions for F and G with the parameters appropriately filled in yields real protocols. For example, instantiating F and G to signatures yields the standard signature-based challenge-response protocol from the *ISO-9798-3* family, whereas instantiating F and G to a keyed hash yields the SKID3 protocol.

Characterizing protocol concepts: Protocol templates provide a useful method for formally characterizing design concepts. Our methodology for formal proofs involves the following two steps.

(1) Assuming properties of the function variables and some invariants, prove properties of the protocol templates. Formally,

$$\mathcal{Q}, \Gamma \vdash \phi_1[P]_A\phi_2$$

Here, $\mathcal{Q}$ is an abstract protocol and P is a program for one role of the protocol. Γ denotes the set of assumed properties and invariants.

(2) Instantiate the function variables to cryptographic functions and prove that the assumed properties and invariants are satisfied by the real protocol. Hence conclude that the real protocol possesses the security property characterized by the protocol templates.

$$\text{If } \mathcal{Q}' \vdash \Gamma', \text{ then } \mathcal{Q}' \vdash \phi_1'[P']_A\phi_2'$$

Here, the primed versions of the protocol, hypotheses, etc. are obtained by applying the substitution σ used in the instantiation.

The correctness of the method follows from the soundness of substitution and the transitivity of entailment in the logic.

5.2 Template for ASW and GJM Protocols

The roles of the template for the ASW and GJM protocols are given in Figure 3. Let us examine the program of the initiator. Notice that the initiator first sends his commitment (*commit*1), receives the responder's commitment (*commit*2), checks its validity, opens his commitment (*open*), and finally expects a message opening the responder's commitment. The responder's program is symmetric. *commit*1, *commit*2 and *open* are function variables representing the messages sent by $\hat{X}$ and $\hat{Y}$ during the protocol. The function variable *chk* models the verification performed by each participant after receiving its last message. These functions are instantiated for ASW and GJM

ExchangeInit $\equiv (\hat{X}, \hat{Y}, \hat{T}, text)[$
  `new` x;
  `send` $\hat{X}, \hat{Y}, commit1(\hat{X}, \hat{Y}, \hat{T}, text, HASH\{|x|\})$;
  `receive` $\hat{Y}, \hat{X}, z$;
  `match` $z/commit2(\hat{Y}, \hat{X}, \hat{T}, text, commit1(\ldots), y)$;
  `send` $\hat{X}, \hat{Y}, open(\hat{X}, text, x)$;
  `receive` $\hat{Y}, \hat{X}, w$;
  `match` $chk(w)/open(\hat{Y}, text, y);]_X$

ExchangeResp $\equiv (\hat{X}, \hat{Y}, \hat{T}, text)[$
  `receive` $\hat{Y}, \hat{X}, z$;
  `match` $z/commit1(\hat{Y}, \hat{X}, \hat{T}, text, y)$;
  `new` x;
  `send` $\hat{X}, \hat{Y}, commit2(\hat{X}, \hat{Y}, \hat{T}, text, z, HASH\{|x|\})$;
  `receive` $\hat{Y}, \hat{X}, w$;
  `match` $chk(w)/open(\hat{Y}, text, y)$;
  `send` $\hat{X}, \hat{Y}, open(\hat{X}, text, x);]_X$

Abort $\equiv (\hat{X}, \hat{Y}, \hat{T}, msg1, text, x)[$
  `send` $\hat{X}, \hat{T}, abortreq(\hat{X}, \hat{Y}, text, msg1)$
  `receive` $\hat{T}, \hat{X}, z$;
  `if` z
    $tpabort(\hat{X}, \hat{Y}, \hat{T}, text, msg1)$:;
    $tpresp(\hat{X}, \hat{Y}, \hat{T}, text, x, y, \hat{Y})$:;
  `fi` $]_X$

ResolveInit $\equiv (\hat{X}, \hat{Y}, \hat{T}, msg1, msg2, text, x)[$
  `send` $\hat{X}, \hat{T}, msg1, msg2$;
  `receive` $\hat{T}, \hat{X}, z$;
  `if` z
    $tpabort(\hat{X}, \hat{Y}, \hat{T}, text, msg1)$:;
    $tpresp(\hat{X}, \hat{Y}, \hat{T}, text, x, y, \hat{Y})$:;
  `fi` $]_X$

Fig. 3. Roles of the protocol template

as follows.

ASW :

$$\begin{aligned} commit1(\hat{X}, \hat{Y}, \hat{T}, text, h) &= SIG_{\hat{X}}\{|\hat{X}, \hat{Y}, \hat{T}, text, h|\} \\ commit2(\hat{X}, \hat{Y}, \hat{T}, text, z, h) &= SIG_{\hat{X}}\{|z, h|\} \\ open(\hat{X}, text, x) &= x \\ chk(w) &= HASH\{|w|\} \end{aligned}$$

GJM :

$$\begin{aligned} commit1(\hat{X}, \hat{Y}, \hat{T}, text, h) &= PCS_{\hat{X}}\{|text, \hat{Y}, \hat{T}|\} \\ commit2(\hat{X}, \hat{Y}, \hat{T}, text, z, h) &= PCS_{\hat{X}}\{|text, \hat{Y}, \hat{T}|\} \\ open(\hat{X}, text, x) &= SIG_{\hat{X}}\{|text|\} \\ chk(w) &= w \end{aligned}$$

In the ASW instantiation, the commitment messages are signatures over hashed nonces and the opening messages reveal the corresponding nonces. The verification action involves checking that the hash of the revealed nonce matches the hash in the commitment message. In the GJM instantiation, a commitment message is a PCS over the contract and the opening message is the corresponding universally verifiable signature. The verification action involves verifying the signature.

To improve readability, we use $msg1(\hat{A}, \hat{B}, \hat{T}, text, x)$ and $msg2(\hat{A}, \hat{B}, \hat{T}, text, x, y)$ as shorthand for the first and second message of the exchange subprotocol, i.e.,

$$\begin{aligned} msg1(\hat{A}, \hat{B}, \hat{T}, text, x) \equiv & commit1(\hat{A}, \hat{B}, \hat{T}, text, HASH\{|x|\}) \\ msg2(\hat{A}, \hat{B}, \hat{T}, text, x, y) \equiv & commit2(\hat{B}, \hat{A}, \hat{T}, text, \\ & msg1(\hat{A}, \hat{B}, \hat{T}, text, x), HASH\{|y|\}) \end{aligned}$$

The protocol definition provides two forms of a contract. A standard contract, s, is obtained on successful completion of the exchange subprotocol. This is the 'optimistic' aspect of these protocols. A replacement contract, r, is issued by the trusted third party in case of dispute. The syntactic forms of these contracts for the ASW and GJM protocols as well as some other message components used in the abort and resolve protocols are given below. Note that we work with the modified version of the GJM protocol as presented in [23]. The only difference is that in the resolve protocol, the responder sends

his PCS to the trusted third party instead of his signature.

ASW :

$$\begin{aligned}
abortreq(\hat{A},\hat{B},text,msg1) =& SIG_{\hat{A}}\{\!|Abort,msg1|\!\}\\
tpabort(\hat{A},\hat{B},\hat{T},text,msg1) =& SIG_{\hat{T}}\{\!|Aborted,\\
& \quad abortreq(\hat{A},\hat{B},text,msg1)|\!\}\\
tpresp(\hat{A},\hat{B},\hat{T},text,x,y,\hat{Z}) =& SIG_{\hat{T}}\{\!|msg1(\hat{A},\hat{B},\hat{T},text,x),\\
& \quad msg2(\hat{A},\hat{B},\hat{T},text,x,y)|\!\}\\
s(\hat{A},\hat{B},\hat{T},text,x,y) =& msg1(\hat{A},\hat{B},\hat{T},text,x),\\
& \quad msg2(\hat{A},\hat{B},\hat{T},text,x,y),\\
& \quad open(\hat{A},text,x),open(\hat{B},text,y)\\
r(\hat{A},\hat{B},\hat{T},text,x,y) =& tpresp(\hat{A},\hat{B},\hat{T},text,x,y,\hat{A})
\end{aligned}$$

GJM :

$$\begin{aligned}
abortreq(\hat{A},\hat{B},text,msg1) =& SIG_{\hat{A}}\{\!|text,\hat{A},\hat{B},Abort|\!\}\\
tpabort(\hat{A},\hat{B},\hat{T},text,msg1) =& SIG_{\hat{T}}\{\!|abortreq(\hat{A},\hat{B},text,msg1)|\!\}\\
tpresp(\hat{A},\hat{B},\hat{T},text,x,y,\hat{Z}) =& open(\hat{Z},text,w)\\
& \quad \text{With } w=x \text{ if } Z=A\\
& \quad \text{ and } w=y \text{ if } Z=B.\\
s(\hat{A},\hat{B},\hat{T},text,x,y) =& open(\hat{A},text,x),open(\hat{B},text,y)\\
r(\hat{A},\hat{B},\hat{T},text,x,y) =& open(\hat{A},text,x),open(\hat{B},text,y)
\end{aligned}$$

We now examine the templates for the *abort* and *resolve* sub-protocols, also given in Figure 3. The initiator A has the option of requesting the trusted third party T to abort an exchange that A has initiated by executing the *abort* subprotocol with T. Finally, both the initiator and the responder may request that T resolve an exchange that has not been completed by executing the *resolve* subprotocol with T.

We model the trusted third party's behavior using a set Γ_T of logical formulas. It is straightforward to write down the program for the trusted third party and establish these properties using the honesty rule for PCL (see Appendix A). In the same way, we will capture the desired behavior of the variable functions *commit*1, *commit*2, *open* and *chk* using a set of logical formulas Λ, and show that both ASW and GJM satisfy Λ. Γ_T^1 is the same assumption about the behavior of the trusted third party T as in Section 4. It says that if T is honest, then it will never issue both a replacement contract and the abort token. Naturally, a misbehaving trusted third party T could easily cheat on any of the participants. Formally, Γ_T^1 is defined as follows:

$$\begin{aligned}
\Gamma_T^1 =& \{\mathsf{Honest}(T) \wedge \mathsf{Send}(T,\hat{T},\hat{Z},tpresp(\hat{A},\hat{B},\hat{T},text,x,z,\hat{Z}'))\\
& \supset \neg\mathsf{Send}(T,\hat{T},\hat{A},tpabort(\hat{A},\hat{B},\hat{T},text,\\
& \quad msg1(\hat{A},\hat{B},\hat{T},text,x)))\}
\end{aligned}$$

Γ_T^2 is the second assumption about the behavior of the trusted third party. It says that T will abort the protocol (by sending the abort token) only if she received an abort request from the initiator.

$$\begin{aligned}\Gamma_T^2 = \{&\mathsf{Send}(T, \hat{T}, \hat{A}, tpabort(\hat{A}, \hat{B}, \hat{T}, text, \\ &msg1(\hat{A}, \hat{B}, \hat{T}, text, x))) \\ \supset\ &\mathsf{Receive}(T, \hat{A}, \hat{T}, abortreq(\hat{A}, \hat{B}, text, \\ &msg1(\hat{A}, \hat{B}, \hat{T}, text, x)))\}\end{aligned}$$

Other instances Although we focus on the ASW and GJM instances, we note that simple variants of the non-repudiation protocols of [20,21,25] are also instances of this template. Specifically, these variants are similar to the ASW protocol, the only difference being that the hash of the nonce is replaced by the encryption of nonce with a secret key which is later revealed in the decommit message. This indicates that the template provides a characterization of a broad class of optimistic contract signing protocols. We are aware of only one timely, optimistic fair exchange protocol that substantially differs from our template [24]. The protocol in [24] is started by the responder sending a specific generation message that serves as a characterization of the considered secret, and that can be used to circumvent the prevalent commitment-based message flows.

5.3 Hypotheses associated with the Template

We prove the security properties of the protocol template under the following hypotheses. It is easy to check that these assumptions are satisfied when the template is instantiated to the ASW and GJM protocols. We omit the rather straightforward proofs. One way to think about these hypotheses is that they represent general high-level specifications that we might expect any optimistic contract signing protocols to satisfy. They can be naturally divided into two

classes.

$$\begin{aligned}
&\vdash \mathsf{Has}(Z, msg1(\hat{A}, \hat{B}, \hat{T}, text, x)) \wedge \\
&\quad \mathsf{Has}(Z, msg2(\hat{A}, \hat{B}, \hat{T}, text, x, y)) \wedge \\
&\quad \mathsf{Has}(Z, open(\hat{A}, text, x)) \wedge \qquad (6) \\
&\quad \mathsf{Has}(Z, open(\hat{B}, text, y)) \supset \mathsf{Has}(Z, s(\hat{A}, \hat{B}, \hat{T}, text, x, y)) \\
&\vdash \mathsf{Has}(A, text, x) \supset \\
&\quad \mathsf{Has}(A, msg1(\hat{A}, \hat{B}, \hat{T}, text, x), open(\hat{A}, text, x)) \qquad (7) \\
&\vdash \mathsf{Has}(Z, w) \wedge \mathsf{Has}(Z, commit2(\hat{B}, \hat{A}, \hat{T}, text, m, y)) \\
&\quad [\ \texttt{match}\ chk(w)/open(\hat{A}, text, y)]_Z \\
&\quad \mathsf{Has}(Z, commit2(\hat{B}, \hat{A}, \hat{T}, text, m, HASH\{|w|\}), \\
&\qquad open(\hat{A}, text, w)) \qquad (8) \\
&\vdash \mathsf{Has}(Z, s(\hat{A}, \hat{B}, \hat{T}, text, x, y)) \supset \\
&\quad \mathsf{Has}(Z, open(\hat{A}, text, x)) \qquad (9)
\end{aligned}$$

The first class of assumptions identify the message components that a principal must possess in order to obtain a standard contract. (6) states that any participant possessing the four messages exchanged in the 'optimistic' protocol execution also possess the standard contract. It would be strange if an optimistic contract signing protocol did not satisfy this property! (7) states that an initiator A has enough information to produce the first and third messages of an optimistic protocol exchange. (8) ensures that given two messages from the responder, the *chk* function allows the initiator to verify that these messages constitute a valid commit–open pair. Finally, (9) shows that given a valid contract, one can extract the initiator's open message from it. These assumptions are discharged for the instances based purely on the properties of the functions used to construct the open and commit messages (e.g., in order to compute the hash of a message, it is essential to possess the message).

$$\begin{aligned}
&\vdash \mathsf{Has}(Z, r(\hat{A}, \hat{B}, \hat{T}, text, x, y)) \wedge \mathsf{Honest}(\hat{T}) \supset \exists Z'.\exists Z'' \\
&\quad \mathsf{Send}(T, \hat{Z}', tpresp(\hat{A}, \hat{B}, \hat{T}, text, x, y, \hat{Z}'')) \\
&\quad \vee \mathsf{Has}(Z, open(\hat{A}, text, x)) \qquad (10) \\
&\vdash \mathsf{Has}(Z, open(\hat{A}, text, x)) \wedge \neg \mathsf{Send}(A, open(\hat{A}, text, x)) \wedge \\
&\quad \mathsf{Honest}(\hat{T}, \hat{A}) \wedge \mathsf{New}(A, x) \supset (Z = A) \vee \exists Z'.\exists Z'' \\
&\quad \mathsf{Send}(T, \hat{Z}', tpresp(\hat{A}, \hat{B}, \hat{T}, text, x, y, \hat{Z}'')) \qquad (11) \\
&\vdash \mathsf{Has}(Z, tpresp(\hat{A}, \hat{B}, \hat{T}, text, x, y, \hat{B}), open(A, text, x)) \supset \\
&\quad \mathsf{Has}(Z, r(\hat{A}, \hat{B}, \hat{T}, text, x, y)) \qquad (12)
\end{aligned}$$

The second class of assumptions capture the hardness of constructing the replacement contract. More precisely, (10) states that the only ways to acquire a replacement contract are to get it from the trusted third party or to construct it from the open messages, while (11) states that the open message for the initiator can only be computed by initiator herself or extracted from the replacement contract issued by the trusted third party. Also, (12) states that

given the initiator's opening message and the response of the trusted third party to an accepted resolve request, one can build the replacement contract. These assumptions are discharged in the template instances based on the relations between the various open and commit messages as well as the protocol steps of the various parties.

5.4 Proving Template Properties

In this section, we prove the security properties of the protocol template. We focus on fairness; the proof of accountability is analogous. Abuse-freeness for the GJM protocol reduces to fairness, because of the properties of private contract signatures.

Fairness

The proof structure parallels that of the ASW fairness proof, the only difference being that we work with the templates instead of the concrete protocols. Some steps in the proof use the hypotheses listed in the previous section.

Initiator completes the exchange protocol Formula ϕ_0 states that the initiator A has a valid contract after the successful execution of the exchange protocol. This is the optimistic part of the protocol.

$$\begin{aligned}\phi_0 \equiv & \mathsf{Start}(A) \\ & [\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)]_A \\ & \mathsf{Has}(A, s)\end{aligned}$$

We formally show in Appendix B that:

$$\vdash \phi_0 \tag{13}$$

Initiator runs the abort protocol Consider the case that A started the protocol as the initiator but did not complete it. We then want to show that whenever some other party has a valid contract then it must be the case that A will get the replacement contract if it executes abort subprotocol after sending the first message. A necessary prerequisite for this to hold is that A's *open* message was not sent yet, i.e., for

$$\theta_1 = \neg\mathsf{Send}(A, open(\hat{A}, text, x)) \wedge \mathsf{New}(A, x)$$

we define the formula ϕ_1 by

$$\begin{aligned}\phi_1 \equiv & \theta_1 \\ & [\mathbf{Abort}(\hat{A}, \hat{B}, \hat{T}, msg1)]_A \\ & (\mathsf{Has}(X, s) \vee \mathsf{Has}(X, r)) \wedge \mathsf{Honest}(\hat{T}) \wedge \mathsf{Honest}(\hat{A}) \\ & \supset (\mathsf{Has}(A, s) \vee \mathsf{Has}(A, r))\end{aligned}$$

It is easy to verify that θ_1 holds in the state where A has only sent the first message of the protocol. We formally show in Appendix B that:

$$\vdash \mathsf{Start}(A)[\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)^1]_A \theta_1 \tag{14}$$

$$\Gamma_T^1 \vdash \phi_1 \tag{15}$$

The fairness property in this case follows from the sequential composition theorem. Here Γ_T^1 is the same assumption about the behavior of the trusted third party T as in Section 4. It says that if T is honest, then it will never issue both a replacement contract and the abort token.

Initiator runs the resolve protocol : Finally, we show that if A has received the second message of the protocol then it can obtain a valid contract by executing the resolve subprotocol, provided that he did not abort the protocol in the part, i.e., for

$$\begin{aligned}\theta_2 = & \neg \mathsf{Send}(A, \hat{A}, \hat{T}, abortreq(\hat{A}, \hat{B}, text, \\ & msg1(\hat{A}, \hat{B}, \hat{T}, text, x))) \wedge \mathsf{New}(A, x)\end{aligned}$$

we define ϕ_2 by

$$\begin{aligned}\phi_2 \equiv & \theta_2 \\ & [\mathbf{Resolve}(\hat{A}, \hat{B}, \hat{T}, msg1, msg2)]_A \\ & (\mathsf{Has}(X, s) \vee \mathsf{Has}(X, r)) \wedge \\ & \mathsf{Honest}(\hat{T}) \wedge \mathsf{Honest}(\hat{A}) \supset \mathsf{Has}(A, r)\end{aligned}$$

It is easy to verify that θ_2 holds in the state where A has received the second message and sent the third message of the protocol. To formally prove fairness from the point of view of the initiator (assuming the desired properties of the trusted third party), we need to show the following :

$$\vdash \mathsf{Start}(A)[\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)^3]_A \theta_2 \tag{16}$$

$$\Gamma_T^2 \vdash \phi_2 \tag{17}$$

Here Γ_T^2 is the second assumption about the behavior of the trusted third party. It says that T will abort the protocol (by sending the abort token) only if she received an abort request from the initiator.

6 Conclusion

We show how to reason compositionally about contract-signing protocols, using a specialized protocol logic to prove properties about general forms of *exchange, abort,* and *resolve* subprotocols and combine these properties using logical composition rules. The method is surprisingly direct for contract signing, given that the logic we used was originally aimed at two-party authentication protocols. The formal proof proceeds along direct, intuitive lines and is carried out in a "template" form that may be instantiated to provide correctness proofs for two standard protocols and protocol variants that use the same arrangement of messages. In addition, the compositional approach makes it unnecessary to consider interleaving of actions from different subprotocols. This is fortunate since interaction between separate subprotocols appears to have been a significant source of difficulty in previous studies. Further, the use of protocol templates gives us a single "reusable" proof that may be instantiated for the Asokan-Shoup-Waidner protocol [1,2], the Garay-Jacobson-McKenzie [15] protocol, and other protocols such as variants using the primitives explored in [20,25]. In this sense, we prove the relatively intuitive but otherwise difficult to state theorem that any protocol of a certain form has precise correctness properties.

Contract signing fairness for party A is proved by explicit reasoning about specific actions taken by A. In effect, this form of argument shows that A has a strategy to obtain a contract by explicitly presenting the strategy. However, the logic is not suited to showing directly that it is possible to complete these steps - that is a modelling assumption that remains outside the formalism. Further, the safety-oriented logic seems less adept at non-trace-based properties such as abuse freeness than game-theoretic approaches. Nonetheless, these axiomatic, general proofs for unbounded runs offer additional validation of optimistic contract signing protocols not readily available through previous approaches.

References

[1] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. Technical Report RZ 2976, IBM Research, 1997.

[2] 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, 1998.

[3] R. Chadha, M. Kanovich, and A. Scedrov. Inductive methods and contract-signing protocols. In *8-th ACM Conference on Computer and Communications Security*, pages 176–185. ACM Press, 2001.

[4] R. Chadha, S. Kremer, and A. Scedrov. Formal analysis of multi-party contract signing. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, pages 266–279. IEEE, 2004.

[5] R. Chadha, J. C. Mitchell, A. Scedrov, and V. Shmatikov. Contract signing, optimism, and advantage. In *14th International Conference on Concurrency Theory (CONCUR '03)*, volume 2761 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[6] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 14:423.

[7] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 109–125. IEEE, 2003.

[8] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition (Extended abstract). In *Proceedings of ACM Workshop on Formal Methods in Security Engineering*, pages 11–23, 2003.

[9] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Abstraction and refinement in protocol derivation. In *Proceedings of 17th IEEE Computer Security Foundations Workshop*, pages 30–45. IEEE, 2004.

[10] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition. In *Proceedings of 19th Annual Conference on Mathematical Foundations of Programming Semantics*, volume 83 of *Electronic Notes in Theoretical Computer Science*, 2004.

[11] A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP '05)*, Lecture Notes in Computer Science. Springer-Verlag, 2005.

[12] N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. In *Proceedings of 14th IEEE Computer Security Foundations Workshop*, pages 241–255. IEEE, 2001.

[13] N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11:677–721, 2003.

[14] S. Even and Y. Yacobi. Relations among public key signature schemes. Technical Report 175, Computer Science Deptartment, Technion, Israel, 1980.

[15] J. A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 449–466. Springer-Verlag, 1999.

[16] C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, 2005.

[17] S. Kremer, O. Markowitch, and J. Zhou. An intensive survey of non-repudiation protocols. *Computer Communications*, 25(17):1606–1621, 2002.

[18] S. Kremer and J.-F. Raskin. Game analysis of abuse-free contract signing. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 206–220. IEEE, 2002.

[19] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, 1995.

[20] O. Markowitch and S. Kremer. A multi-party optimistic non-repudiation protocol. In *Proceedings of the Third International Conference on Information Security and Cryptology*, pages 109–122. Springer-Verlag, 2001.

[21] O. Markowitch and S. Saeednia. Optimistic fair exchange with transparent signature recovery. In *Proceedings of the 5th International Conference on Financial Cryptography*, pages 339–350. Springer-Verlag, 2001.

[22] H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, Department of Computer Science, Darmstadt University of Technology, Darmstadt, Germany, 1999.

[23] V. Shmatikov and J. C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, 2002.

[24] H. Vogt. Asynchronous optimistic fair exchange based on revocable items. In *Proceedings of the 7th International Conference on Financial Cryptography*, pages 208–222. Springer-Verlag, 2003.

[25] J. Zhou, R. Deng, and F. Bao. Evolution of fair non-repudiation with TTP. In *Proceedings of the 4th Australasian Conference on Information Security and Privacy*, volume 1587 of *Lecture Notes in Computer Science*, pages 258–269. Springer-Verlag, 1999.

A Protocol Composition Logic

A.1 Semantics

The formulas of the logic are interpreted over runs, which are finite sequences of reaction steps from an initial configuration. An equivalent view is to think of a run as a linear sequence of states. Transition from one state to the next is effected by an action carried out by some principal in some role. A formula is true in a run if it is true in the last state of that run.

The main semantic relation, $\mathcal{Q}, R \models \phi$, may be read, "formula ϕ holds for run R of protocol $\mathcal{Q}$." If $\mathcal{Q}$ is a protocol, then let $\bar{\mathcal{Q}}$ be the set of all initial

configurations of protocol $\mathcal{Q}$, each including a possible intruder program. Let $\mathsf{Runs}(\bar{\mathcal{Q}})$ be the set of all runs of protocol $\mathcal{Q}$ with intruder, each a sequence of reaction steps within a cord space. If ϕ has free variables, then $Q, R \models \phi$ if we have $Q, R \models \sigma\phi$ for all substitutions σ that eliminate all the free variables in ϕ. For a set of formulas Γ, we say that $\Gamma \models \phi$ if $\mathcal{Q}, R \models \Gamma$ implies $\mathcal{Q}, R \models \phi$. We write $\mathcal{Q} \models \phi$ if $\mathcal{Q}, R \models \phi$ for all $R \in \mathsf{Runs}(\bar{\mathcal{Q}})$.

In the following, $EVENT(R, X, P, \vec{n}, \vec{x})$ means that in run R, thread X executes actions P, receiving data $\vec{n}$ into variables $\vec{x}$, where $\vec{n}$ and $\vec{x}$ are the same length.

Action Formulas:

- $\mathcal{Q}, R \models \mathsf{Send}(A, m)$ if $EVENT(R, A, \texttt{send}\ m, \emptyset, \emptyset)$.
- $\mathcal{Q}, R \models \mathsf{Receive}(A, m)$ if $EVENT(R, A, \texttt{receive}\ x, m, x)$.
- $\mathcal{Q}, R \models \mathsf{New}(A, m)$ if $EVENT(R, A, \texttt{new}\ x, m, x)$.
- $\mathcal{Q}, R \models \mathsf{Decrypt}(A, ENC_K\{\!|m|\!\})$ if $EVENT(R, A, (\texttt{match}\ ENC_K\{\!|m|\!\}/SIG_K\{\!|x|\!\}), m, x)$
  Note: $\mathsf{Decrypt}(A, n)$ is *false* if $n \neq ENC_K\{\!|m|\!\}$ for some m and K.
- $\mathcal{Q}, R \models \mathsf{Verify}(A, SIG_K\{\!|m|\!\})$ if $EVENT(R, A, (\texttt{match}\ SIG_K\{\!|m|\!\}/ENC_K\{\!|m|\!\}), \emptyset, \emptyset)$
  Note: $\mathsf{Verify}(A, n)$ is *false* if $n \neq SIG_K\{\!|m|\!\}$ for some m and K.

Other Formulas:

- $\mathcal{Q}, R \models \mathsf{Has}(A, m)$ if there exists an i such that $\mathsf{Has}_i(A, m)$ where Has_i is inductively as follows:
  $(\mathsf{Has}_0(A, m)$ if $((m \in FV(R|_A))$
  $\vee\ EVENT(R, A, \texttt{new}\ x, m, x)$
  $\vee\ EVENT(R, A, \texttt{receive}\ x, m, x)$
  and $\mathsf{Has}_{i+1}(A, m)$ if $\mathsf{Has}_i(A, m) \vee\ (\mathsf{Has}_i(A, m')$
  $\vee\ (\mathsf{Has}_i(A, m') \wedge \mathsf{Has}_i(A, m'')$
  $\wedge\ ((m = m', m'') \vee (m = m'', m')))$
  $\vee(\mathsf{Has}_i(A, m') \wedge \mathsf{Has}_i(A, K)$
  $\wedge\ m = ENC_K\{\!|m'|\!\})$
  $\vee(\mathsf{Has}_i(A, a) \wedge \mathsf{Has}_i(A, g^b)$
  $\wedge\ m = g^{ab})$
  $\vee(\mathsf{Has}_i(A, g^{ab}) \wedge m = g^{ba})$
  Intuitively, Has_0 holds for terms that are known directly, either as a free variable of the role, or as the direct result of receiving or generating the term. Has_{i+1} holds for terms that are known by applying i operations (decomposing via pattern matching, composing via encryption or tupling, or by computing a Diffie-Hellman secret) to terms known directly.
- $\mathcal{Q}, R \models \mathsf{Honest}(\hat{A})$ if $\hat{A} \in HONEST(\mathbf{C})$ in initial configuration $\mathbf{C}$ for R and all threads of $\hat{A}$ are in a "pausing" state in R. More precisely, $R|_{\hat{A}}$ is an

interleaving of basic sequences of roles in Q.

- $\mathcal{Q}, R \models \mathsf{Contains}(t_1, t_2)$ if $t_2 \subseteq t_1$.
- $\mathcal{Q}, R \models (\phi_1 \wedge \phi_2)$ if $\mathcal{Q}, R \models \phi_1$ and $\mathcal{Q}, R \models \phi_2$
- $\mathcal{Q}, R \models \neg\phi$ if $\mathcal{Q}, R \not\models \phi$
- $\mathcal{Q}, R \models \exists x.\phi$ if $\mathcal{Q}, R \models (d/x)\phi$, for some d, where $(d/x)\phi$ denotes the formula obtained by substituting d for x in ϕ.
- $\mathcal{Q}, R \models \mathsf{Start}(X)$ if $R|_X$ is empty. Intuitively this formula means that X didn't execute any actions in the past.

Modal Formulas:

- $\mathcal{Q}, R \models \phi_1 \, [P]_A \, \phi_2$ if $R = R_0 R_1 R_2$, for some R_0, R_1 and R_2, and either P does not match $R_1|_A$ or P matches $R_1|_A$ and $\mathcal{Q}, R_0 \models \sigma\phi_1$ implies $\mathcal{Q}, R_0 R_1 \models \sigma\phi_2$, where σ is the substitution matching P to $R_1|_A$.

A.2 Proof System

A.2.1 Axioms and Inference Rules

The axioms and inference rules of the proof system that are used in this paper are collected in Table A.1. **AA1** states that if a principal has executed an action in some role, then the corresponding predicate asserting that the action had occurred in the past is true. **VER** captures the unforgeability of signatures of honest principals by the attacker. This axiom (together with a few more axioms not described in this summary) provide an abstraction of the standard Dolev-Yao intruder model. Axioms **P1** and **P3** capture the fact that most predicates are preserved by additional actions. For example, if in some state $\mathsf{Has}(X, n)$ holds, then it continues to hold, when X executes additional actions. The generic rules are used to reason about modal formulas in the style of Floyd-Hoare logic. The conditional rule is used for reasoning about the "if" construct introduced in this paper.

A.2.2 The Honesty Rule

The honesty rule is essentially an invariance rule for proving properties of all roles of a protocol. It is similar to the basic invariance rule of LTL [19]. The honesty rule is used to combine facts about one role with inferred actions of other roles.

For example, suppose Alice receives a response from a message sent to Bob. Alice may wish to use properties of Bob's role to reason about how Bob generated his reply. In order to do so, Alice may assume that Bob is honest and

Axioms for protocol actions

AA1 $true[a]_X\ \mathsf{a}$
AA2 $\mathsf{Start}(X)[\,]_X\ \neg\mathsf{a}(\mathsf{X})$
AA3 $\neg\mathsf{Send}(X,t)[b]_X\neg\mathsf{Send}(X,t)$ if $\sigma\mathsf{Send}(X,t) \neq \sigma\mathsf{b}$ for all substitutions σ
AN2 $true[\mathtt{new}x]_X\ \mathsf{Has}(Y,x) \supset (Y = X)$
ARP $\mathsf{Receive}(X,p(x))[\mathtt{match}q(x)/q(t)]_X\ \mathsf{Receive}(X,p(t))$

Possession Axioms

ORIG $\mathsf{New}(X,x) \supset \mathsf{Has}(X,x)$
REC $\mathsf{Receive}(X,x) \supset \mathsf{Has}(X,x)$
TUP $\mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y) \supset \mathsf{Has}(X,(x,y))$
PROJ $\mathsf{Has}(X,(x,y)) \supset \mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y)$
SIG $\mathsf{Has}(X, SIG_{\hat{Y}}\{|x|\}) \supset \mathsf{Has}(X,x)$

Signature

VER $\mathsf{Honest}(\hat{X}) \wedge \mathsf{Verify}(Y, SIG_{\hat{X}}\{|x|\}) \wedge \hat{X} \neq \hat{Y} \supset \exists X.\mathsf{Send}(X, [SIG_{\hat{X}}\{|x|\}])$

Preservation Axioms

$\mathsf{Persist} \in \{\mathsf{Has}, \mathsf{a}\}$:

P1 $\mathsf{Persist}(X,t)[a]_X\mathsf{Persist}(X,t)$

P3 $\mathsf{HasAlone}(X,n)[a]_X\mathsf{HasAlone}(X,n)$, where $n \not\sqsubseteq_v a$ or $a \neq \langle m \rangle$

$$\mathsf{HasAlone}(X,t) \equiv \mathsf{Has}(X,t) \wedge (\mathsf{Has}(Y,t) \supset X = Y)$$

Generic Rules

$$\frac{\theta[P]_X\phi \quad \theta[P]_X\psi}{\theta[P]_X\phi \wedge \psi}\ \mathbf{G1} \qquad \frac{\theta[P]_X\phi \quad \theta' \supset \theta \quad \phi \supset \phi'}{\theta'[P]_X\phi'}\ \mathbf{G2} \qquad \frac{\phi}{\theta[P]_X\phi}\ \mathbf{G3}$$

Conditional

$$\frac{\phi[\ \mathtt{match}\ t/t_1; P_1]\psi_1 \quad \phi[\ \mathtt{match}\ t/t_2; P_2]\psi_2 \quad \ldots \quad \phi[\ \mathtt{match}\ t/t_n; P_n]\psi_n}{\phi[\ \mathtt{if}\ t \quad t_1 : P_1; \quad t_2 : P_2; \ldots \quad t_n : P_n;\ \mathtt{fi}\ ]\psi_1 \vee \psi_2 \vee \cdots \vee \psi_n}\ \mathbf{IF}$$

Table A.1
Relevant Fragment of the PCL Proof System

derive consequences from this assumption. Since honesty, by definition in this framework, means "following one or more roles of the protocol," honest principals must satisfy every property that is a provable invariant of the protocol roles.

Since the honesty rule depends on the protocol, we write $\mathcal{Q} \vdash \theta[P]\phi$ if $\theta[P]\phi$ is provable using the honesty rule for $\mathcal{Q}$ and the other axioms and proof rules.

Using the notation just introduced, the honesty rule may be written as follows.

$$\frac{[\,]_X\ \phi \quad \forall \rho \in \mathcal{Q}. \forall P \epsilon BS(\rho).\ \phi\ [P]_X\ \phi}{\mathcal{Q} \vdash \mathsf{Honest}(\hat{X}) \supset \phi}\ HON \qquad \begin{array}{l}\text{no free variable}\\ \text{in } \phi \text{ except } X\\ \text{bound in } [P]_X\end{array}$$

In words, if ϕ holds at the beginning of every role of $\mathcal{Q}$ and is preserved by all its basic sequences, then every honest principal executing protocol $\mathcal{Q}$ must satisfy ϕ. The side condition prevents free variables in the conclusion $\mathsf{Honest}(\hat{X}) \supset \phi$ from becoming bound in any hypothesis. Intuitively, since ϕ holds in the initial state and is preserved by all basic sequences, it holds at all pausing states of any run.

B Formal Proofs

Formal proofs are collected in Tables B.1 through B.10.

AA1, **ORIG**, **G2**	$\mathsf{Start}(A)$	
	$[\ \texttt{new}\ x]_A$	
	$\mathsf{Has}(A, x)$	(B.1)
AA1, **ARP**	ϕ	
	$[\ \texttt{receive}\ A, \hat{B}, \hat{A}, z;\ \ \texttt{match}\ z/SIG_{\hat{B}}\{\mid msg_1, w \mid\}]_A$	
	$\mathsf{Receive}(A, SIG_{\hat{B}}\{\mid msg_1, w \mid\})$	(B.2)
$(B.2)$, **PROJ**	ϕ	
	$[\ \texttt{receive}\ A, \hat{B}, \hat{A}, z;\ \ \texttt{match}\ z/SIG_{\hat{B}}\{\mid msg_1, w \mid\}]_A$	
	$\mathsf{Has}(A, msg_1)$	(B.3)
AA1, **ARP**, **REC**	$\mathsf{Receive}(A, SIG_{\hat{B}}\{\mid msg_1, w \mid\}$	
	$[\ \texttt{receive}\ A, \hat{B}, \hat{A}, w;\ \ \texttt{match}\ HASH\{\mid w \mid\}/y]_A$	
	$\mathsf{Has}(A, y) \wedge \mathsf{Has}(A, SIG_{\hat{B}}\{\mid msg_1, HASH\{\mid y \mid\}\mid\})$	(B.4)
$(B.1), (B.2), (B.3), (B.4)$, **P1**, **TUP**	ϕ	
	$[\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)]_A$	
	$\mathsf{Has}(A, s(\hat{A}, \hat{B}, \hat{T}, text, x, y))$	(B.5)

Table B.1
Proof of Equation 1

$$\begin{aligned}
\mathbf{AA1}, \mathbf{ORIG}, \mathbf{AN2}\ & \mathsf{Start}(A) \\
& [\ \texttt{new}\ x]_A \\
& \mathsf{Has}(A,x) \wedge (\mathsf{Has}(B,x) \supset (B = A)) && \text{(B.6)} \\
(B.6), \mathbf{P3}\ & \mathsf{Start}(A) \\
& [\ \texttt{new}\ x;\ \texttt{send}\ \hat{A}, \hat{B}, msg_1]_A \\
& \mathsf{HasAlone}(A,x) && \text{(B.7)} \\
(B.7)\ & \mathsf{Start}(A) \\
& [\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)^1]_A \\
& \theta_1 && \text{(B.8)}
\end{aligned}$$

Table B.2
Proof of Equation 2

$$\begin{aligned}
\mathbf{P3}\ & \mathsf{HasAlone}(A,x) \\
& [\mathbf{Abort}(\hat{A}, \hat{B}, \hat{T}, msg_1)]_A \\
& \mathsf{HasAlone}(A,x) && \text{(B.9)} \\
\mathbf{PROJ}\ & \mathsf{Has}(X,s) \supset \mathsf{Has}(X,x) && \text{(B.10)} \\
(B.10), \mathbf{P3}\ & \mathsf{HasAlone}(A,x) \wedge A \neq X \supset \neg\mathsf{Has}(X,s) && \text{(B.11)} \\
(B.11)\ & \neg\mathsf{Send}(B,s) \wedge A \neq B \supset \neg\mathsf{Has}(A,s) && \text{(B.12)} \\
\mathbf{VER}\ & \mathsf{Honest}(\hat{T}) \wedge \mathsf{Verify}(X,r) \wedge \hat{X} \neq \hat{T} \\
& \supset \mathsf{Send}(T, \hat{T}, \hat{X}, r) && \text{(B.13)} \\
\Gamma_T^1, (B.11), (B.12), (B.13)\ & \mathsf{HasAlone}(A,x) \wedge (\mathsf{Has}(X,s) \vee \mathsf{Has}(X,r)) \\
& \supset (\mathsf{Honest}(\hat{T}) \supset \\
& \quad \neg\mathsf{Send}(T, \hat{T}, \hat{A}, SIG_{\hat{T}}\{|Aborted, msg_1|\})) && \text{(B.14)} \\
(B.14), \mathbf{IF}\ & \theta_1 \\
& [\mathbf{Abort}(\hat{A}, \hat{B}, \hat{T}, msg_1)]_A \\
& (\mathsf{Has}(X,s) \vee \mathsf{Has}(X,r)) \wedge \mathsf{Honest}(\hat{T}) \\
& \supset \mathsf{Has}(A,r) && \text{(B.15)}
\end{aligned}$$

Table B.3
Proof of Equation 3

	AA2 $\mathsf{Start}(A)$	
	$[]_A$	
	$\neg\mathsf{Send}(A, \hat{A}, \hat{T}, SIG_{\hat{A}}\{\!\lfloor Abort, msg_1 \rfloor\!\})$	(B.16)
	AA1 $[\mathsf{New}(x)]_A\mathsf{New}(A, x)$	(B.17)
$(B.16), (B.17),$	**AA3** $\mathsf{Start}(A)$	
	$[\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)^3]_A$	
	θ_2	(B.18)

Table B.4
Proof of Equation 4

VER, **HON**	$\mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T} \wedge \mathsf{Receive}(T, \hat{A}, \hat{T}, SIG_{\hat{A}}\{\!\lfloor Abort, msg_1 \rfloor\!\})$	
	$\supset \mathsf{Send}(A, \hat{A}, \hat{T}, SIG_{\hat{A}}\{\!\lfloor Abort, msg_1 \rfloor\!\})$	(B.19)
$(B.19),$	$\theta_2 \wedge \mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T}$	
	$\supset \neg\mathsf{Receive}(T, \hat{A}, \hat{T}, SIG_{\hat{T}}\{\!\lfloor Aborted, msg_1 \rfloor\!\})$	(B.20)
$(B.20), \Gamma_T^2$	$\theta_2 \wedge \mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T}$	
	$\supset \neg\mathsf{Send}(T, \hat{T}, \hat{A}, SIG_{\hat{T}}\{\!\lfloor Aborted, msg_1 \rfloor\!\})$	(B.21)
$(B.21),$ **IF**	θ_2	
	$[\mathbf{Resolve}(\hat{A}, \hat{B}, \hat{T}, msg_1, msg_2)]_A$	
	$(\mathsf{Has}(X, s) \vee \mathsf{Has}(X, r)) \wedge \mathsf{Honest}(\hat{T}) \supset \mathsf{Has}(A, r)$	(B.22)

Table B.5
Proof of Equation 5

$$\mathbf{AA1}, \mathbf{ORIG}, \mathbf{G2}\ \mathsf{Start}(A)[\ \texttt{new}\ x]_A\mathsf{Has}(A,x) \tag{B.23}$$

$$(B.23), (7), \mathbf{G2}\ \mathsf{Start}(A)\ [\ \texttt{new}\ x]_A\ \mathsf{Has}(A, msg1(\hat{A},\hat{B},\hat{T},text,x), open(\hat{A},text,x)) \tag{B.24}$$

$$\mathbf{AA1}, \mathbf{ARP}, \mathbf{REC}\ \phi\ [\ \texttt{receive}\ A,\hat{B},\hat{A},z;\ \texttt{match}\ z/commit2(\hat{B},\hat{A},\hat{T},text, msg1(\hat{A},\hat{B},\hat{T},text,x),y)]_A\ \mathsf{Has}(A, commit2(\hat{B},\hat{A},\hat{T},text, msg1(\hat{A},\hat{B},\hat{T},text,x),y)) \tag{B.25}$$

$$\mathbf{AA1}, \mathbf{REC}, (8), \mathbf{S1}\ \mathsf{Has}(A, commit2(\hat{B},\hat{A},\hat{T},text, msg1(\hat{A},\hat{B},\hat{T},text,x),y))\ [\ \texttt{receive}\ A,\hat{B},\hat{A},w;\ \texttt{match}\ chk(w)/open(\hat{B},text,y)]_A\ \mathsf{Has}(A, open(\hat{B},text,w), msg2(\hat{A},\hat{B},\hat{T},text,x,w)) \tag{B.26}$$

$$(B.24), (B.26), \mathbf{P1}, (6), \mathbf{G2}\ \phi\ [\mathbf{ExchangeInit}(\hat{A},\hat{B},\hat{T},text)]_A\ \mathsf{Has}(A, s(\hat{A},\hat{B},\hat{T},text,x,y)) \tag{B.27}$$

Table B.6
Proof of Equation 13

$$\mathbf{AA1}, \mathbf{P1}, \mathbf{AN2}\ \mathsf{Start}(A)\ [\ \texttt{new}\ x;\ \texttt{send}\ \hat{A},\hat{B},msg1(\hat{A},\hat{T},text,x)]_A\ \mathsf{New}(A,x) \tag{B.28}$$

$$\mathbf{AA3}\ \mathsf{Start}(A)\ [\ \texttt{new}\ x;\ \texttt{send}\ \hat{A},\hat{B},msg1(\hat{A},\hat{T},text,x)]_A\ \neg\mathsf{Send}(A, open(\hat{A},text,x)) \tag{B.29}$$

$$(B.28), (B.29), \mathbf{G1}\ \mathsf{Start}(A)\ [\mathbf{ExchangeInit}(\hat{A},\hat{B},\hat{T},text)^1]_A\ \theta_1 \tag{B.30}$$

Table B.7
Proof of Equation 14

	AA2 $\mathsf{Start}(A)$	
	$[]_A$	
	$\neg\mathsf{Send}(A, abortreq(\hat{A}, \hat{B}, text, msg1))$	(B.31)
	AA1 $[\mathsf{New}(x)]_A\mathsf{New}(A, x)$	(B.32)
$(B.31), (B.32),$	**AA3** $\mathsf{Start}(A)$	
	$[\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)^3]_A$	
	θ_2	(B.33)

Table B.8
Proof of Equation 16

	$\gamma = \mathsf{Honest}(T) \wedge \mathsf{Honest}(A) \wedge (\mathsf{Has}(Z, s(\hat{A}, \hat{B}, \hat{T}, x, y)) \vee \mathsf{Has}(Z, r(\hat{A}, \hat{B}, \hat{T}, x, y)))$	
$(10), (11), (9)$	$\theta_1 \wedge \gamma$	
	$\supset (Z = A) \vee \exists Z'.\exists Z''.\mathsf{Send}(T, \hat{Z'}, tpresp(\hat{A}, \hat{B}, \hat{T}, text, x, y, \hat{Z''}))$	(B.34)
$(B.34), (15)$	$\theta_1 \wedge \gamma$	
	$\supset (Z = A) \vee \neg\mathsf{Send}(T, \hat{T}, \hat{A}, tpabort(\hat{A}, \hat{B}, \hat{T}, text,$	
	$msg1(\hat{A}, \hat{B}, \hat{T}, text, x)))$	(B.35)
$(B.35)$	$\theta_1 \wedge \gamma$	
	$\supset (Z = A) \vee \neg\mathsf{Receive}(A, \hat{T}, \hat{A}, tpabort(\hat{A}, \hat{B}, \hat{T}, text,$	
	$msg1(\hat{A}, \hat{B}, \hat{T}, text, x)))$	(B.36)
$(B.36), \mathbf{IF}, \mathbf{P1}$	$\theta_1[\mathbf{Abort}]_A\gamma$	
	$\supset (Z = A) \vee \mathsf{Receive}(A, \hat{T}, \hat{A}, tpresp(\hat{A}, \hat{B}, \hat{T}, text, x, y, \hat{B}))$	(B.37)
$(B.37), \mathbf{REC}$	$\theta_1[\mathbf{Abort}]_A\gamma$	
	$\supset (Z = A) \vee \mathsf{Has}(A, tpresp(\hat{A}, \hat{B}, \hat{T}, text, x, y, \hat{B}))$	(B.38)
$(B.38), (12)$	$\theta_1[\mathbf{Abort}]_A\gamma$	
	$\supset \mathsf{Has}(A, s(\hat{A}, \hat{B}, \hat{T}, x, y)) \vee \mathsf{Has}(A, r(\hat{A}, \hat{B}, \hat{T}, x, y))$	

Table B.9
Proof of Equation 15

$$\textbf{VER}\ \mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T} \wedge \mathsf{Receive}(T, \hat{A}, \hat{T}, abortreq(\hat{A}, \hat{B}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x))) \quad \text{(B.39)}$$

$$\supset \mathsf{Send}(A, \hat{A}, \hat{T}, abortreq(\hat{A}, \hat{B}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x))) \quad \text{(B.40)}$$

$$(B.40)\ \theta_2 \wedge \mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T}$$

$$\supset \neg\mathsf{Receive}(T, \hat{A}, \hat{T}, abortreq(\hat{A}, \hat{B}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x))) \quad \text{(B.41)}$$

$$(B.41), \Gamma_T^2\ \theta_2 \wedge \mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T}$$

$$\supset \neg\mathsf{Send}(T, \hat{T}, \hat{A}, abortreq(\hat{A}, \hat{B}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x))) \quad \text{(B.42)}$$

$$(B.42), \textbf{IF}\ \theta_2[\textbf{Resolve}]_A\mathsf{Honest}(\hat{T}) \wedge \mathsf{Honest}(\hat{A}) \wedge (\mathsf{Has}(X, s) \vee \mathsf{Has}(X, r))$$

$$\supset \mathsf{Has}(A, r) \quad \text{(B.43)}$$

Table B.10
Proof of Equation 17