Compositional Analysis of Contract Signing Protocols*

Michael Backes
IBM Zürich Research Lab
mbc@zurich.ibm.com

Anupam Datta
Stanford University
danupam@cs.stanford.edu

Ante Derek
Stanford University
aderek@cs.stanford.edu

John C. Mitchell
Stanford University
jcm@cs.stanford.edu

Mathieu Turuani †
LORIA-INRIA Nancy
turuani@loria.fr

Abstract

We develop a general method for reasoning about contract-signing protocols using a specialized protocol logic. The method is applied to prove properties of 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 which each protocol is comprised. Second, the formal proofs are carried out in a "template" form, which gives us a reusable proof that may be instantiated for the ASW and GJM 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 that the specific strategy that the protocol designer had in mind works, instead of showing that one exists. Finally, our results hold even 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 do. While there are no fixed-round fair two-party protocols [13, 21], it is possible for two parties to exchange signatures optimistically. In optimistic 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 [15], for example.

Several methods have been used to analyze contract-signing protocols and either find errors or suggest their absence. For example, [22] uses the finite-state enumeration tool Murφ, [16, 4] use game-theoretic concepts and the MOCHA branching-time temporal logic model checker, [3] uses inductive arguments, and [5] uses ad hoc game tree arguments to prove the non-existence of protocols satisfying certain conditions. However, 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. The 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 [14] which, although otherwise rigorous, overlooks the case leading to an error reported in [22].

In this paper, we develop a method for reasoning about contract-signing protocols using a specialized protocol 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,

* This work was partially supported by the DoD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795, by OSD/ONR CIP/SW URI through ONR Grant N00014-04-1-0725, by NSF CCR-0121403, Computational Logic Tools for Research and Education, and by NSF CyberTrust Grant 0430594, Collaborative research: High-fidelity methods for security protocols.

† Partially supported by AVISPA IST-2001-39252W

without limitation on the number of additional protocol sessions that may run in parallel and may provide alternate sources of signed messages by any 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 properties of the entire protocol can be proved compositionally from independent proofs for the three subprotocols (called the *exchange, abort,* and *resolve* subprotocols in the body of the paper).

The compositional protocol logic was proposed in [12, 6], which give example proofs for two-party authentication and key exchange protocols, and used by different authors in [20] to uncover a previously undetected bug and establish correctness of group key management protocols. A version of the logic that is particulary relevant to the present paper is given in [8], where the idea of proving correctness of protocol templates is explained. In this approach, 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 [14] 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.

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 modelling 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 briefly describes the ASW and GJM protocols (with further details in Appendix A). 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 proof system used in the axiomatic proofs in the paper is in Appendix B. Section 4 presents the analysis of the ASW protocol, emphasizing the compositional proof method. Complete formal proofs are in Appendix C. 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 briefly describe the two protocols of interest. More detailed descriptions including the arrows-and-messages diagrams are given in Appendix A.

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 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 from the network, each obtains a valid contract upon completion of the *exchange* subprotocol. The originator O also has the option of requesting 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. After receiving the initial message of the *exchange* protocol, they may do so by executing the *resolve* subprotocol with T.

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 4.

2.2. Garay-Jakobsson-MacKenzie Protocol

The protocol of Garay, Jakobsson, and MacKenzie (called the *GJM protocol* henceforth) [14] is closely related to the ASW protocol in that both protocols involve a 4-step *exchange* subprotocol and similar *abort* and *resolve* subprotocols. Even though the two protocols have similar structure, the actual 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 modelled and proved in Section 5. 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 $TSig_O(m)$, 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 [14].

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.

3. Methodology

3.1. Cord Calculus

One important part of security analysis involves understanding the way honest agents running a protocol will respond to messages from a malicious attacker. The common informal arrows-and-messages notation is therefore insufficient, since it only presents the intended executions (or traces) of the protocol. In addition, the protocol logic requires more information about a protocol than the set of protocol executions obtained from honest and malicious parties; we need a high-level description of the program executed by each agent performing each protocol role. We used a programming language called *cords* [11].

3.2. A Protocol Logic

The basic protocol logic and proof system are developed in [11, 6, 7, 10], with [9] providing a relatively succinct presentation of the most recent form. A summary of the proof system is included in Appendix B.

The formulas of the logic are given by the grammar in Table 1, where ρ may be any role, written using the notation of cord calculus. Here, t and P denote a term and a thread respectively. We use the word *thread* to refer to a principal executing an instance of a role. As a notational convention, 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 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.

$\mathsf{Fresh}(X, t)$ means the term t generated in X is "fresh" in the sense that no one else has seen any term containing t as a subterm. Typically, a fresh term will be a nonce and freshness will be used to reason about the temporal ordering of actions in runs of a protocol.

$\mathsf{Honest}(\hat{X})$ 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.

$\mathsf{a}_1 \leq \mathsf{a}_2$ means that both actions a_1 and a_2 happened in the run and moreover, that the action a_2 happened after the action a_1.

3.3. Extensions of the Logic

We extend cord calculus and the protocol logic with "if" constructs in the spirit of Dijkstra's guarded commands. These extensions allow us to capture more complicated behavior of protocol participants. Formally, the grammar of actions is extended by the production $a ::= \ \mathtt{if}\ t \quad t_1 : P_1; t_2 : P_2; \dots t_n : P_n;\ \mathtt{fi}$, where t, t_1, $\dots$, t_n stand for terms, and $P_1, \dots, P_n$ for strands (sequences of actions).

Informally, this construct works as a generalization of pattern matching. If i is the lowest number such that t matches t_i, then the above statement is reduced to $\mathtt{match}\ t/t_i : P_i;$. The inference rule **IF** for reasoning about this construct is given in Table 2 in Appendix B.

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 in the cords formalism. 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 C. 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, written using the cords formalism are given in Figure 1, with one cord (program) **ExchangeInit** for the initiator of the protocol and one cord **ExchangeResp** for the responder. A role consists of static input parameters, and a list of actions to be executed. For example, the cord **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 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. In particular, the first and second message will be important as they grant the respective parties the ability to resolve a protocol execution. For reasons of readability, we introduce syntactic shorthand 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. Using the extensions of the logic presented in this paper, it is easy to write down the cord

Action formulas

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

Formulas

$\phi \quad ::= \quad \mathsf{a} \mid \mathsf{Has}(P, t) \mid \mathsf{Computes}(P, t) \mid \mathsf{Fresh}(P, t) \mid \mathsf{Honest}(N) \mid \phi \wedge \phi \mid \neg\phi \mid \exists x.\phi \mid \mathsf{a} \leq \mathsf{a}$

Modal form

$\Psi \quad ::= \quad \phi\, \rho\, \phi$

Table 1. Syntax of the logic

ExchangeInit $\equiv (\hat{X}, \hat{Y}, \hat{T}, text)[$
- `new` x;
- `send` $\hat{X}, \hat{Y}, SIG_{\hat{X}}\{\!|\hat{X}, \hat{Y}, \hat{T}, text, HASH\{\!|x|\!\}|\!\}$;
- `receive` $\hat{Y}, \hat{X}, z$;
- `match` $z/SIG_{\hat{Y}}\{\!|SIG_{\hat{X}}\{\!|\hat{X}, \hat{Y}, \hat{T}, text, HASH\{\!|x|\!\}|\!\}, w|\!\}$;
- `send` $\hat{X}, \hat{Y}, x$;
- `receive` $\hat{Y}, \hat{X}, y$;
- `match` $HASH\{\!|y|\!\}/w;]_X$

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

Abort $\equiv (\hat{X}, \hat{Y}, \hat{T}, msg1)[$
- `send` $\hat{X}, \hat{T}, SIG_{\hat{X}}\{\!|Abort, msg1|\!\}$
- `receive` $\hat{T}, \hat{X}, z$;
- `if` z
  - $SIG_{\hat{T}}\{\!|Aborted, msg1|\!\}$:;
  - $r(\hat{A}, \hat{B}, \hat{T}, text, HASH\{\!|x|\!\}, w)$:;
- `fi` $]_X$

Resolve $\equiv (\hat{X}, \hat{Y}, \hat{T}, msg1, msg2)[$
- `send` $\hat{X}, \hat{T}, msg1, msg2$;
- `receive` $\hat{T}, \hat{X}, z$;
- `if` z
  - $SIG_{\hat{T}}\{\!|Aborted, msg1|\!\}$:;
  - $r(\hat{A}, \hat{B}, \hat{T}, text, HASH\{\!|x|\!\}, w)$:;
- `fi` $]_X$

Figure 1. Roles of the ASW protocol

for the trusted third party and prove that these logical formulas represent properties of its protocol. We omit the technical details.

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 [6, 9]. 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 (see [2] for discussion). We will use $\mathbf{ExchangeInit^3}(\hat{A}, \hat{B}, \hat{T}, text)$ to denote the prefix of the initiator role in the protocol upto 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 upto 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."

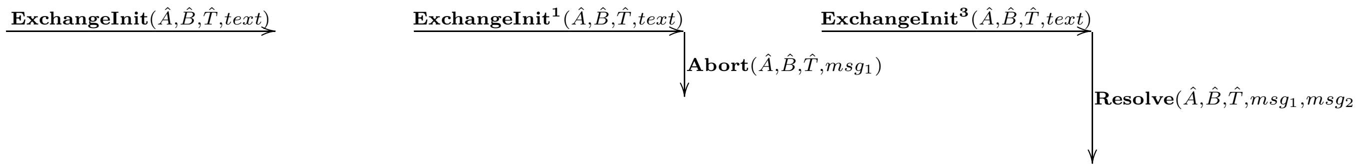

Figure 2. Possible runs for the initiator in the ASW protocol

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. We state below a variant of the theorem from [9] 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, cords $\mathbf{P}$ 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 cords $\mathbf{P}$ 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 C. Here, we sketch the proof structure and focus on the main 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 exchange protocol. This is the optimistic part of the protocol.

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

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

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

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 C.

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

$$\Gamma_T^1 \vdash \phi_1 \tag{3}$$

The fairness property in this case simply follows from the sequential composition theorem (Theorem 1).

Above, Γ_T^1 is a set of assumptions about the behavior of the trusted third party T, which we will now define. 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. Formally Γ_T^1 is defined as follows:

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

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 C.

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

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

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.

$$\begin{aligned}\Gamma_T^2 = \{&\mathsf{Honest}(\hat{T}) \wedge \mathsf{Send}(T, \hat{T}, \hat{A}, SIG_{\hat{T}}\{|Aborted, msg_1|\}) \supset \\ &\mathsf{Receive}(T, \hat{A}, \hat{T}, SIG_{\hat{A}}\{|Abort, msg_1|\})\}.\end{aligned}$$

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 [22] 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 Similar property can be shown for the responder in the ASW protocol. We omit the details.

4.3.2. Accountability Accountability means that if one of the parties gets cheated as a result of $\hat{T}$'s misbehavior, then 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:

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

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 proof for these two protocols and their properties. In this section, we answer this question in the affirmative. The technical machinery used toward this end is presented in [8].

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 [8]. 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 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 short-cuts for the first and second message of the exchange subprotocol, i.e.,

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

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 [22]. The only difference is that in the resolve protocol, the responder sends his PCS to the trusted third party instead of his signature.

ASW :

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

GJM :

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

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.

Other instances Although we focus on the ASW and GJM instances, we note that simple variants of the non-repudiation protocols of [18, 19, 24] 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.[1]

1 We are aware of only one timely, optimistic fair exchange protocol that substantially differs from our template [23]. The protocol in [23] 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

$$\begin{array}{l}
\textbf{ExchangeInit} \equiv (\hat{X}, \hat{Y}, \hat{T}, text)[\\
\quad \texttt{new}\ x; \\
\quad \texttt{send}\ \hat{X}, \hat{Y}, commit1(\hat{X}, \hat{Y}, \hat{T}, text, HASH\{\!|x|\!\}); \\
\quad \texttt{receive}\ \hat{Y}, \hat{X}, z; \\
\quad \texttt{match}\ z/commit2(\hat{Y}, \hat{X}, \hat{T}, text, commit1(\ldots), y); \\
\quad \texttt{send}\ \hat{X}, \hat{Y}, open(\hat{X}, text, x); \\
\quad \texttt{receive}\ \hat{Y}, \hat{X}, w; \\
\quad \texttt{match}\ chk(w)/open(\hat{Y}, text, y);]_X
\end{array}$$

$$\begin{array}{l}
\textbf{ExchangeResp} \equiv (\hat{X}, \hat{Y}, \hat{T}, text)[\\
\quad \texttt{receive}\ \hat{Y}, \hat{X}, z; \\
\quad \texttt{match}\ z/commit1(\hat{Y}, \hat{X}, \hat{T}, text, y); \\
\quad \texttt{new}\ x; \\
\quad \texttt{send}\ \hat{X}, \hat{Y}, commit2(\hat{X}, \hat{Y}, \hat{T}, text, z, HASH\{\!|x|\!\}); \\
\quad \texttt{receive}\ \hat{Y}, \hat{X}, w; \\
\quad \texttt{match}\ chk(w)/open(\hat{Y}, text, y); \\
\quad \texttt{send}\ \hat{X}, \hat{Y}, open(\hat{X}, text, x);]_X
\end{array}$$

$$\begin{array}{l}
\textbf{Abort} \equiv (\hat{X}, \hat{Y}, \hat{T}, msg1, text, x)[\\
\quad \texttt{send}\ \hat{X}, \hat{T}, abortreq(\hat{X}, \hat{Y}, text, msg1) \\
\quad \texttt{receive}\ \hat{T}, \hat{X}, z; \\
\quad \texttt{if}\ z \\
\quad\quad tpabort(\hat{X}, \hat{Y}, \hat{T}, text, msg1) :; \\
\quad\quad tpresp(\hat{X}, \hat{Y}, \hat{T}, text, x, y, \hat{Y}) :; \\
\quad \texttt{fi}\ ]_X
\end{array}$$

$$\begin{array}{l}
\textbf{ResolveInit} \equiv (\hat{X}, \hat{Y}, \hat{T}, msg1, msg2, text, x)[\\
\quad \texttt{send}\ \hat{X}, \hat{T}, msg1, msg2; \\
\quad \texttt{receive}\ \hat{T}, \hat{X}, z; \\
\quad \texttt{if}\ z \\
\quad\quad tpabort(\hat{X}, \hat{Y}, \hat{T}, text, msg1) :; \\
\quad\quad tpresp(\hat{X}, \hat{Y}, \hat{T}, text, x, y, \hat{Y}) :; \\
\quad \texttt{fi}\ ]_X
\end{array}$$

Figure 3. Roles of the protocol template

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.

$$\begin{aligned}
&\vdash \mathsf{Has}(Z, msg1(\hat{A}, \hat{B}, \hat{T}, text, x)) \wedge \\
&\quad \mathsf{Has}(Z, msg2(\hat{A}, \hat{B}, \hat{T}, text, x, y)) \wedge \\
&\quad \mathsf{Has}(Z, open(\hat{A}, text, x)) \wedge \qquad (6)\\
&\quad \mathsf{Has}(Z, open(\hat{B}, text, y)) \supset \mathsf{Has}(Z, s(\hat{A}, \hat{B}, \hat{T}, text, x, y))
\end{aligned}$$

$$\begin{aligned}
&\vdash \mathsf{Has}(A, text, x) \supset \\
&\quad \mathsf{Has}(A, msg1(\hat{A}, \hat{B}, \hat{T}, text, x), open(\hat{A}, text, x)) \qquad (7)
\end{aligned}$$

$$\begin{aligned}
&\vdash \mathsf{Has}(Z, w) \wedge \mathsf{Has}(Z, commit2(\hat{Y}, \hat{X}, \hat{T}, text, m, y)) \\
&\quad [\,\texttt{match}\ chk(w)/open(\hat{X}, text, y)]_Z \\
&\quad \mathsf{Has}(Z, commit2(\hat{Y}, \hat{X}, \hat{T}, text, m, HASH\{\!|w|\!\}), \\
&\qquad open(\hat{X}, text, w)) \qquad (8)
\end{aligned}$$

$$\begin{aligned}
&\vdash \mathsf{Has}(Z, s(\hat{A}, \hat{B}, \hat{T}, text, x, y)) \supset \\
&\quad \mathsf{Has}(Z, open(\hat{A}, text, x)) \qquad (9)
\end{aligned}$$

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.

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

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

$$\begin{aligned}
&\vdash \mathsf{Has}(Z, tpresp(\hat{A}, \hat{B}, \hat{T}, text, x, y, \hat{B}), open(A, text, x)) \supset \\
&\quad \mathsf{Has}(Z, r(\hat{A}, \hat{B}, \hat{T}, text, x, y)) \qquad (12)
\end{aligned}$$

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.

commitment-based message flows.

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 C that:

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

Initiator runs the abort protocol If A started the protocol as the initiator, but did not complete, we 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 C 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. Naturally, a misbehaving trusted third party T could easily cheat on any of the participants. Formally, Γ_T^1 is defined as follows:

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

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 received an abort request from the initiator.

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

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 [14] protocol, and other protocols such as variants using the primitives explored in [18, 24], for example. 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 for security protocols and its logical formalization. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 109–125. IEEE, 2003.

[7] 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.

[8] 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.

[9] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security (to appear)*, 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] 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.

[12] 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.

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

[14] 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.

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

[16] 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.

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

[18] 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.

[19] 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.

[20] C. Meadows and D. Pavlovic. Deriving, attacking and defending the GDOI protocol. In *Proceedings of 9th European Symposium On Research in Computer Security*, pages 53–72, 2004.

[21] 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.
[22] V. Shmatikov and J. C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, 2002.
[23] 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.
[24] 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. Contract Signing Protocols

A.1. Asokan-Shoup-Waidner Protocol

The ASW protocol consists of three interdependent subprotocols: *exchange*, *abort*, and *resolve*. The parties (O and R) generally start the exchange by following the *exchange* subprotocol. If both O and R are honest and there is no interference from the network, each obtains a valid contract upon the completion of the *exchange* subprotocol. The originator O also has the option of requesting 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. After receiving the initial message of the *exchange* protocol, they may do so by executing the *resolve* subprotocol with T.

The *exchange* protocol is given below.

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

The *abort* protocol is given below.

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

The *resolve* protocol is given below.

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

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

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

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 4.

A.2. Garay-Jakobsson-MacKenzie Protocol

The GJM protocol is closely related to the ASW protocol described above. Both involve a 4-step *exchange* subprotocol, and similar *abort* and *resolve* subprotocols. Even though the two protocols have similar structure, the actual 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 modelled and proved in Section 5. 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 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 $TSig_O(m)$, 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 [14].

The *exchange* protocol is given below.

$$\begin{array}{lcll} O \to R & me_1 & = & PCS_O(m, R, T) \\ R \to O & me_2 & = & PCS_R(m, O, T) \\ O \to R & me_3 & = & Sig_O(m) \\ R \to O & me_4 & = & Sig_R(m) \end{array}$$

The *abort* protocol is given below.

$$\begin{array}{lll} O \rightarrow T & ma_1 & = & Sig_O(m, O, R, abort) \\ T \rightarrow O & ma_2 & = & \text{Has } O \text{ or } R \text{ resolved already?} \\ \quad \text{Yes :} & Sig_R(m) & & \text{if } R \text{ has resolved, or} \\ & T - Sig_R(m) & & \text{if } O \text{ has resolved} \\ \quad \text{No :} & Sig_T(ma_1) & & \\ & aborted := \text{true} & & \end{array}$$

The *resolve* protocol is given below.

$$\begin{array}{llll} R \rightarrow T & mr_1 & = & PCS_O(m, R, T), Sig_R(m) \\ T \rightarrow R & mr_2 & = & \text{Has } O \text{ aborted already?} \\ \quad \text{Yes :} & \multicolumn{3}{l}{\text{Send } Sig_T(Sig_O(m, O, R, abort))} \\ \quad \text{No :} & \multicolumn{3}{l}{\text{Has } O \text{ resolved already?}} \\ & \quad \text{Yes :} & \multicolumn{2}{l}{\text{Send } Sig_O(m)} \\ & \quad \text{No :} & \multicolumn{2}{l}{\text{Store } Sig_R(m)} \\ & & \multicolumn{2}{l}{\text{Convert } PCS_O(m, R, T) \text{ to } T - Sig_O(m)} \\ & & \multicolumn{2}{l}{\text{Send } T - Sig_O(m)} \\ & \multicolumn{3}{l}{resolved := \text{true}} \end{array}$$

B. Protocol Logic

B.1. Proof System

B.1.1. Axioms and Inference Rules A representative fragment of the axioms and inference rules in the proof system are collected in Table 2. For expositional convenience, we divide the axioms into five groups.

The axioms about protocol actions state properties that hold in the state reached by executing one of the actions in a state in which formula ϕ holds. Note that the a in axiom **AA1** is any one of the 5 actions and a is the corresponding predicate in the logic.

VER and **SRC** respectively refer to the unforgeability of signatures and the need to possess the symmetric key in order to decrypt a message encrypted with that key. The additional condition requiring principal $\hat{X}$ to be honest guarantees that the intruder is not in possession of the private keys. These axioms (together with a few more axioms not described in this summary) provide an abstraction of the standard Dolev-Yao intruder model.

Axioms **P1**, **P2**, 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.

B.1.2. 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 [17]. 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 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.

C. Formal Proofs

Formal proofs are collected in Tables 3 through 8. Formal proofs of some formulas have been left out due to space constrains.

Axioms for protocol actions

AA1	$\phi[a]_X\ \mathsf{a}$
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
AA4	$\phi[a_1; a_2; \ldots; a_k]_X \mathsf{a}_1 < \mathsf{a}_2 \wedge \cdots \wedge \mathsf{a}_{k-1} < \mathsf{a}_k$
AN2	$\phi[\mathtt{new} x]_X\ \mathsf{Has}(Y,x) \supset (Y = X)$
AN3	$\phi[\mathtt{new} x]_X\ \mathsf{Fresh}(X,x)$
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	$\mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y) \supset \mathsf{Has}(X,(x,y))$
ENC	$\mathsf{Has}(X,x) \wedge \mathsf{Has}(X,K) \supset \mathsf{Has}(X, ENC_K\{\|x\|\})$
PROJ	$\mathsf{Has}(X,(x,y)) \supset \mathsf{Has}(X,x) \wedge \mathsf{Has}(X,y)$
DEC	$\mathsf{Has}(X, ENC_K\{\|x\|\}) \wedge \mathsf{Has}(X,K) \supset \mathsf{Has}(X,x)$

Encryption and Signature

SEC	$\mathsf{Honest}(\hat{X}) \wedge \mathsf{Decrypt}(Y, ENC_{\hat{X}}\{\|x\|\}) \supset (\hat{Y} = \hat{X})$
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

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

P1	$\mathsf{Persist}(X,t)[a]_X\mathsf{Persist}(X,t)$
P2	$\mathsf{Fresh}(X,t)[a]_X\mathsf{Fresh}(X,t)$, where $t \not\subseteq a$ or $a \neq \langle m \rangle$
P3	$\mathsf{HasAlone}(X,n)[a]_X\mathsf{HasAlone}(X,n)$, where $n \not\subseteq_v a$ 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 2. Fragment of the Proof System

$\mathbf{AA1}, \mathbf{ORIG}, \mathbf{G2}$	$\mathsf{Start}(A)[\ \mathtt{new}\ x]_A \mathsf{Has}(A, x)$	(18)
$\mathbf{AA1}, \mathbf{ARP}$	$\phi[\ \mathtt{receive}\ A, \hat{B}, \hat{A}, z;\ \ \mathtt{match}\ z/SIG_{\hat{B}}\{\!|msg_1, w|\!\}]_A$	
	$\mathsf{Receive}(A, SIG_{\hat{B}}\{\!|msg_1, w|\!\})$	(19)
$(19), \mathbf{PROJ}$	$\phi[\ \mathtt{receive}\ A, \hat{B}, \hat{A}, z;\ \ \mathtt{match}\ z/SIG_{\hat{B}}\{\!|msg_1, w|\!\}]_A$	
	$\mathsf{Has}(A, msg_1)$	(20)
$\mathbf{AA1}, \mathbf{ARP}, \mathbf{REC}$	$\mathsf{Receive}(A, SIG_{\hat{B}}\{\!|msg_1, w|\!\}$	
	$[\ \mathtt{receive}\ A, \hat{B}, \hat{A}, w;\ \ \mathtt{match}\ HASH\{\!|w|\!\}/y]_A$	
	$\mathsf{Has}(A, y) \wedge \mathsf{Has}(A, SIG_{\hat{B}}\{\!|msg_1, HASH\{\!|y|\!\}|\!\})$	(21)
$(18), (19), (20), (21), \mathbf{P1}, \mathbf{TUP}$	ϕ	
	$[\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)]_A$	
	$\mathsf{Has}(A, s(\hat{A}, \hat{B}, \hat{T}, text, x, y))$	(22)

Table 3. Proof of Equation 1

$\mathbf{P3}$	$\mathsf{HasAlone}(A, x)$	
	$[\mathbf{Abort}(\hat{A}, \hat{B}, \hat{T}, msg_1)]_A$	
	$\mathsf{HasAlone}(A, x)$	(23)
$\mathbf{PROJ}$	$\mathsf{Has}(X, s) \supset \mathsf{Has}(X, x)$	(24)
$(24), \mathbf{P3}$	$\mathsf{HasAlone}(A, x) \wedge A \neq X \supset \neg\mathsf{Has}(X, s)$	(25)
(25)	$\neg\mathsf{Send}(B, s) \wedge A \neq B \supset \neg\mathsf{Has}(A, s)$	(26)
$\mathbf{VER}$	$\mathsf{Honest}(\hat{T}) \wedge \mathsf{Verify}(X, r) \wedge \hat{X} \neq \hat{T} \supset \mathsf{Send}(T, \hat{T}, \hat{X}, r)$	(27)
$\Gamma_T^1, (25), (26), (27)$	$\mathsf{HasAlone}(A, x) \wedge (\mathsf{Has}(X, s) \vee \mathsf{Has}(X, r))$	
	$\supset (\mathsf{Honest}(\hat{T}) \supset \neg\mathsf{Send}(T, \hat{T}, \hat{A}, SIG_{\hat{T}}\{\!|Aborted, msg_1|\!\}))$	(28)
$(28), \mathbf{IF}$	θ_1	
	$[\mathbf{Abort}(\hat{A}, \hat{B}, \hat{T}, msg_1)]_A$	
	$(\mathsf{Has}(X, s) \vee \mathsf{Has}(X, r)) \wedge \mathsf{Honest}(\hat{T}) \supset \mathsf{Has}(A, r)$	(29)

Table 4. Proof of Equation 3

$\mathbf{VER}, \mathbf{HON}$	$\mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T} \wedge \mathsf{Receive}(T, \hat{A}, \hat{T}, SIG_{\hat{A}}\{\!|Abort, msg_1|\!\})$	
	$\supset \mathsf{Send}(A, \hat{A}, \hat{T}, SIG_{\hat{A}}\{\!|Abort, msg_1|\!\})$	(30)
$(30),$	$\theta_2 \wedge \mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T} \supset \neg\mathsf{Receive}(T, \hat{A}, \hat{T}, SIG_{\hat{T}}\{\!|Aborted, msg_1|\!\})$	(31)
$(31), \Gamma_T^2$	$\theta_2 \wedge \mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T} \supset \neg\mathsf{Send}(T, \hat{T}, \hat{A}, SIG_{\hat{T}}\{\!|Aborted, msg_1|\!\})$	(32)
$(32), \mathbf{IF}$	θ_2	
	$[\mathbf{Resolve}(\hat{A}, \hat{B}, \hat{T}, msg_1, msg_2)]_A$	
	$(\mathsf{Has}(X, s) \vee \mathsf{Has}(X, r)) \wedge \mathsf{Honest}(\hat{T}) \supset \mathsf{Has}(A, r)$	(33)

Table 5. Proof of Equation 5

$\mathbf{AA1}, \mathbf{ORIG}, \mathbf{G2}$	$\mathsf{Start}(A)[\ \mathtt{new}\ x]_A \mathsf{Has}(A, x)$	(34)
$(34), (7), \mathbf{G2}$	$\mathsf{Start}(A)[\ \mathtt{new}\ x]_A \mathsf{Has}(A, msg1(\hat{A}, \hat{B}, \hat{T}, text, x), open(\hat{A}, text, x))$	(35)
$\mathbf{AA1}, \mathbf{ARP}, \mathbf{REC}$	ϕ	
	$[\ \mathtt{receive}\ A, \hat{B}, \hat{A}, z;\ \ \mathtt{match}\ z/commit2(\hat{B}, \hat{A}, \hat{T}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x), y)]_A$	
	$\mathsf{Has}(A, commit2(\hat{B}, \hat{A}, \hat{T}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x), y))$	(36)
$\mathbf{AA1}, \mathbf{REC}, (8), \mathbf{S1}$	$\mathsf{Has}(A, commit2(\hat{B}, \hat{A}, \hat{T}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x), y))$	
	$[\ \mathtt{receive}\ A, \hat{B}, \hat{A}, w;\ \ \mathtt{match}\ chk(w)/open(\hat{B}, text, y)]_A$	
	$\mathsf{Has}(A, open(\hat{B}, text, w), msg2(\hat{A}, \hat{B}, \hat{T}, text, x, w))$	(37)
$(35), (37), \mathbf{P1}, (6), \mathbf{G2}$	ϕ	
	$[\mathbf{ExchangeInit}(\hat{A}, \hat{B}, \hat{T}, text)]_A$	
	$\mathsf{Has}(A, s(\hat{A}, \hat{B}, \hat{T}, text, x, y))$	(38)

Table 6. Proof of Equation 13

$\gamma\ =$	$\mathsf{Honest}(T) \wedge \mathsf{Honest}(A) \wedge (\mathsf{Has}(Z, s(\hat{A}, \hat{B}, \hat{T}, x, y)) \vee \mathsf{Has}(Z, r(\hat{A}, \hat{B}, \hat{T}, x, y)))$	
$(10), (11), (9)$	$\theta_1 \wedge \gamma \supset (Z = A) \vee \exists Z'.\exists Z''.\mathsf{Send}(T, \hat{Z}', tpresp(\hat{A}, \hat{B}, \hat{T}, text, x, y, \hat{Z}''))$	(39)
$(39), (15)$	$\theta_1 \wedge \gamma \supset (Z = A) \vee \neg\mathsf{Send}(T, \hat{T}, \hat{A}, tpabort(\hat{A}, \hat{B}, \hat{T}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x)))$	(40)
(40)	$\theta_1 \wedge \gamma \supset (Z = A) \vee \neg\mathsf{Receive}(A, \hat{T}, \hat{A}, tpabort(\hat{A}, \hat{B}, \hat{T}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x)))$	(41)
$(41), \mathbf{IF}, \mathbf{P1}$	$\theta_1[\mathbf{Abort}]_A \gamma \supset (Z = A) \vee \mathsf{Receive}(A, \hat{T}, \hat{A}, tpresp(\hat{A}, \hat{B}, \hat{T}, text, x, y, \hat{B}))$	(42)
$(42), \mathbf{REC}$	$\theta_1[\mathbf{Abort}]_A \gamma \supset (Z = A) \vee \mathsf{Has}(A, tpresp(\hat{A}, \hat{B}, \hat{T}, text, x, y, \hat{B}))$	(43)
$(43), (12)$	$\theta_1[\mathbf{Abort}]_A \gamma \supset \mathsf{Has}(A, s(\hat{A}, \hat{B}, \hat{T}, x, y)) \vee \mathsf{Has}(A, r(\hat{A}, \hat{B}, \hat{T}, x, y))$	

Table 7. Proof of Equation 15

$\mathbf{VER}$	$\mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T} \wedge \mathsf{Receive}(T, \hat{A}, \hat{T}, abortreq(\hat{A}, \hat{B}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x)))$	(44)
	$\supset \mathsf{Send}(A, \hat{A}, \hat{T}, abortreq(\hat{A}, \hat{B}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x)))$	(45)
(45)	$\theta_2 \wedge \mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T} \supset \neg\mathsf{Receive}(T, \hat{A}, \hat{T}, abortreq(\hat{A}, \hat{B}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x)))$	(46)
$(46), \Gamma_T^2$	$\theta_2 \wedge \mathsf{Honest}(\hat{A}) \wedge \hat{A} \neq \hat{T} \supset \neg\mathsf{Send}(T, \hat{T}, \hat{A}, abortreq(\hat{A}, \hat{B}, text, msg1(\hat{A}, \hat{B}, \hat{T}, text, x)))$	(47)
$(47), \mathbf{IF}$	$\theta_2[\mathbf{Resolve}]_A \mathsf{Honest}(\hat{T}) \wedge \mathsf{Honest}(\hat{A}) \wedge (\mathsf{Has}(X, s) \vee \mathsf{Has}(X, r)) \supset \mathsf{Has}(A, r)$	(48)

Table 8. Proof of Equation 17