Analysis of Abuse-Free Contract Signing

Vitaly Shmatikov John C. Mitchell

Computer Science Department
Stanford University
Stanford, CA 94305-9045, U.S.A.

{shmat,jcm}@cs.stanford.edu

Abstract. Optimistic contract signing protocols may involve subprotocols that allow a contract to be signed normally or aborted or resolved by a third party. Since there are many ways these subprotocols might interact, protocol analysis involves consideration of a number of complicated cases. With the help of Murφ, a finite-state verification tool, we analyze the abuse-free optimistic contract signing protocol of Garay, Jakobsson, and MacKenzie. In addition to verifying a number of subtle properties, we discover an attack in which negligence or corruption of the trusted third party may allow abuse or unfairness. Contrary to the intent of the protocol, the cheated party is not able to hold the third party accountable. In addition to analyzing a modification to the protocol that avoids these problems, we discuss issues involved in the application of finite-state analysis to fair exchange protocols, in particular models of fairness guarantees, abuse, and corrupt protocol participants.

1 Introduction

Contracts are an important part of business. If two parties wish to sign a contract, but do not share other motives, then each may refuse to sign until the other has demonstrated their commitment to the contract. While simultaneous commitment can be achieved by sitting at a table and signing identical paper copies together, distributed contract signing over a network is inherently asymmetric: someone has to send the first message. In one contemporary style of contract-signing protocol, two rounds of communication are used. In the first round, each party declares their willingness to be bound by the contract. In the second, they each send some remaining data needed to complete the contract. If a trusted third party is able to enforce the contract based on partial completion of the protocol, then it is possible to conduct distributed contract signing so that various symmetric correctness conditions are satisfied. In optimistic contract signing, the third party is only needed in case of a dispute. Otherwise, the protocol can be completed without involving the third party.

The most basic correctness condition for contract signing is called fairness. A contract signing protocol is *fair* if, after completion of the protocol, either both parties have a signed contract or neither does. Another property is called *accountability*: if any party cheats by not following the steps required by the protocol, the resulting network messages will unambiguously show which party has cheated. Accountability is particularly important for the trusted third party, since the third party has the ability to resolve or abort a contract.

A more complex condition, introduced in [GJM99], has been called *abuse-freeness*. This condition is intended to guarantee that neither party has a specific kind of advantage over the other during the execution of the protocol. To illustrate by example, suppose Alice offers to sell her house to Bob and Bob signs a contract for a certain price. If Alice holds the contract without signing, she may be able to use the contract to convince another buyer to pay more than Bob. Meanwhile, Bob has committed his financial resources to the incomplete transaction and cannot enter into competing deals. In this scenario, Alice obtains evidence she can use to convince another buyer that she alone can decide whether to complete the contract or reject it. This kind of asymmetry can be prevented in physical simultaneous transactions, but it is difficult to prevent abuse in distributed protocols.

In this paper, we describe an automated analysis of the two-party contract signing protocol presented in [GJM99] (henceforth, the GJM protocol). This two-party protocol is designed to be fair, abuse-free, and to provide accountability. Using a finite-state enumeration tool called Murφ, we verify fairness and completeness, and uncover a weakness in the protocol. Specifically, the contract initiator, A, using a weak form of passive assistance (or information leak) from the third party, is able to choose whether to reveal a completed contract or accept an abort token provided by the third party. Furthermore, if A chooses to reveal her completed contract, and the discrepancy with B's abort token is observed, it is not possible to determine whether the third party participated in the inconsistency or not.

Although the sequence of actions demonstrating this weakness in the protocol is relatively short and easy to follow, the analysis is subtle in several respects. First, the sequence involves interaction between the optimistic two-party transaction normally used to sign a contract, the abort protocol used by one party to time out and stop the protocol, and the resolve protocol used to request enforcement by the third party. As a result of the complexity of interactions between these three subprotocols, we did not suspect any problems until our analysis tool uncovered a violation of one of our correctness conditions. Only then, after examining the trace provided, were we able to isolate a specific aspect of the GJM protocol that allows the attack. This led us to a simple repair, also proposed by the authors of the protocol after we described the attack [Mac99]. The repaired protocol appears to be correct; Murφ analysis does not suggest any errors.

There is some subtlety in the way that the basic protocol requirements, fairness, abuse-freeness, and accountability, are specified. In examining fairness, for example, we realized that an *abort* message from the third party does not mean that no participant will receive a contract. This is inherent in optimistic two-party protocols: after the protocol has finished without involving the trusted third party, one of the parties can ask the third party to abort the protocol. Another subtlety surrounds abuse-freeness, which is an assertion about choices at intermediate states in the execution of the protocol. Abuse-freeness is not a property that can be determined by examining individual traces of protocol execution independently. Since Murφ is a trace-based tool, we had to devise some extension of the protocol environment, involving an outside party who issues *sign* and *abort* challenges, in order to automatically verify the states in which one participant has the power to determine the eventual outcome of the protocol.

Formal methods have been used to analyze the security properties of key exchange and authentication protocols [KMM94,Ros95,Mea96b,Bol97,Pau98]. In particular, finite-state analysis has been successfully applied to protocols such as Needham-Schroeder [Low96,Mea96a,MCJ97], Kerberos [MMS97], SSL [MSS98], and others. However, less attention has been paid to other kinds of protocols, such as fair exchange. In [HTWW96], Heintze et al. used the FDR model checker to verify NetBill [CTS95] and Digicash [CFN88] protocols. We have previously used Murφ [SM00] to analyze the optimistic contract signing protocol of Asokan, Shoup, and Waidner [ASW98a] (henceforth, the ASW protocol). While the ASW and GJM protocols both involve a 4-step exchange protocol and similar abort and resolve subprotocols, the actual contents of the messages differ. In addition, the ASW protocol is not designed to be abuse-free. Therefore, our analysis of the GJM protocol involves several new concepts and modeling techniques not needed for analysis of the ASW protocol.

The remainder of this paper is structured as follows: section 2 provides background on formal tools and fair-exchange protocols, section 3 describes the GJM protocol, section 4 presents our modeling assumptions and analysis results, and section 5 describes the analysis of the repaired protocol. Brief concluding remarks appear in section 6.

2 Background

2.1 Overview of Murφ

Murφ [Dil96] is a finite-state machine verification tool. Originally developed for hardware verification, Murφ has been successfully used for analyzing security protocols [MMS97,MSS98,SS98,SM00].

To analyze a security protocol in Murφ, it is necessary to combine the finite-state model of the protocol expressed in the Murφ language with the *intruder model*, specify the start state of the protocol, and formally state protocol invariants as boolean conditions that must be true in every state reachable from the start state. The intruder model typically consists of a set of variables that contain the intruder's knowledge and a set of actions that the intruder may take. We use a very simple, mechanical intruder model. The intruder is assumed to have full control over the public network and allowed to take the following actions: (1) overhear every message, decrypt encrypted messages if it has the key, store parts of message in its internal database, (2) intercept messages and remove them from the network, (3) generate messages using any combination of its initial knowledge, parts of overheard messages, known keys, etc. If at any moment there are several possible actions that the intruder can take, one is chosen nondeterministically. The Murφ system will analyze all states that are reachable via any interleaving of enabled actions.

Our intruder model has no notion of partial information or probability. It cannot perform cryptanalysis or statistical tests of the network traffic, and it follows the "black box" cryptography model: an encrypted message can be read only if the decrypting key is known, otherwise its contents are assumed to be invisible to the intruder (who is still capable of storing the message and replaying it later in a different context).

A new-generation Murφ, currently under development at Stanford, uses the predicate abstraction method to model check infinite state spaces. It was used by Satyaki Das to analyze multiple instances of the GJM protocol (see section 5 below).

2.2 Fair Exchange

Fair exchange protocols are used for online payment systems, in which a payment is exchanged for an item of value [CTS95], contract signing, in which parties exchange commitments to a contractual text [BOGMR90,ASW98a,GJM99], certified electronic mail [BT94,ZG96,DGLW96], and other purposes. There are several varieties of fair exchange protocols.

Gradual exchange protocols [BOGMR90,BCDvdG87] work by having the parties release their items in small installments, thus ensuring that at any given moment the amount of information received by each side is approximately the same. The drawback of this approach is that a large number of communication steps between the parties is required. Gradual exchange is also problematic if the items to be exchanged have "threshold" value (either the item is valuable, or it is not).

Another category of fair exchange protocols is based on the notion of a *trusted third party* [CTS95,ZG96,DGLW96]. The trusted third party supervises communication between the protocol participants and ensures that no participant receives the item it wants before releasing its own item. Variations of this approach include fair exchange protocols with a semi-trusted third party [FR97]. The main drawback of the third party solution is that the third party may become the communication bottleneck if it has to be involved in all instances of the protocol in order to guarantee fairness. The protocol may also need to impose demands on the communication channels, *e.g.*, by requiring that all messages are eventually delivered to their intended recipients.

Recently, several protocols have been proposed for *optimistic* fair exchange [ASW98a,BDM98,GJM99]. While the third party T may need to be trusted by all parties to the exchange, T needs to act only if one of the parties misbehaves or there is a communication failure. This may ease the communication bottleneck associated with T, making fair exchange more practical for realistic applications.

3 Abuse-Free Optimistic Contract Signing Protocol

In this section, we describe the abuse-free optimistic contract signing protocol of Garay, Jakobsson, and MacKenzie [GJM99]. We start by giving a high-level description of the objectives of the GJM protocol and the standard cryptographic primitives used. We then explain the properties of *private contract signatures* (PCS), an innovation of Garay, Jakobsson, and MacKenzie used to make contract signing abuse-free. Finally, we describe the protocol steps in detail and formalize the correctness conditions posed by its designers, including fairness and abuse-freeness.

3.1 Objectives and assumptions

The GJM protocol is designed to enable two parties, A and B, to exchange signatures on a contractual text. It is assumed that prior to executing the protocol, the parties agree on each other's identity, the contractual text, and the identity of the trusted third party T. Every protocol participant is assumed to know the correct signature verification key of the other party and T. It is also assumed that every participant has a private communication channel with T.

The protocol is asynchronous. As the exchange protocol progresses, either participant may contact the trusted third party T. The third party may decide, on the basis of the communication it received, to either resolve the protocol by issuing the other party's signature, or "abort" the protocol by issuing an abort token. Abort tokens are *not* a proof that the exchange has been canceled, as explained below. The intruder may schedule messages and insert its own messages in the network, but cannot delay messages sent between participants and T indefinitely.

It is assumed that all protocol participants have the ability to compute and verify conventional, universally-verifiable digital signatures. Below, we write $\text{S-Sig}_A(m)$ for the result of signing text m with the key of party A. (These signatures can be verified by anybody in possession of A's signature verification key, which is typically A's public key).

3.2 Private Contract Signatures

The GJM protocol relies on the cryptographic primitive called *private contract signature* (PCS). We write $\text{PCS}_A(m, B, T)$ for party A's private contract signature of text m for party B (known as the *designated verifier*) with respect to third party T. The main properties of PCS are summarized below:

- $\text{PCS}_A(m, B, T)$ can be verified like a conventional signature, *i.e.*, there exists a probabilistic polynomial-time algorithm PCS-Ver such that $\text{PCS-Ver}(m, A, B, T, s)$ is *true iff* $s = \text{PCS}_A(m, B, T)$.
- $\text{PCS}_A(m, B, T)$ can be feasibly computed by either A, or B, but nobody else. This is the key property of PCS that distinguishes it from a conventional, universally-verifiable signature, as the latter can only be computed by A.

When the designated verifier B receives $s = \mathrm{PCS}_A(m, B, T)$, he will be convinced that s was computed by A, but, unlike A's conventional signature, s cannot be used by B to prove this to an outside party.
- $\mathrm{PCS}_A(m, B, T)$ can be converted into a conventional signature by either A, or T, but nobody else, including B. For the purposes of this paper, we focus on the *third-party accountable* version of PCS, in which the converted signatures produced by A and T can be distinguished. We will call them $\text{S-Sig}_A(m)$ and $\text{TP-Sig}_A(m)$, 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 [GJM99].

3.3 Protocol

The GJM protocol consists of three interdependent subprotocols: *exchange*, *abort*, and *resolve*. The parties (A and B) generally start the exchange by following the *exchange* subprotocol. If both A and B are honest and there is no interference from the network, they obtain each other's signatures as the final steps of the *exchange* subprotocol. The originator A also has the option of requesting the trusted third party T to abort an exchange that A has initiated. To do so, A executes the *abort* subprotocol with T. Finally, both A and B 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.

At the end of the protocol, each party is guaranteed to end up with the other party's universally-verifiable signature of the contractual text, or an abort token signed by T and A, of the form $\text{S-Sig}_T(\text{S-Sig}_A(m, A, B, abort))$. An abort token should *not* be interpreted as a proof that the exchange has been aborted. The protocol does not prevent a dishonest A from obtaining an abort token after signing the contract with B. (In this case, A may have both an abort token and B's signature, while B has only A's signature). The protocol is designed, however, to prevent one party from *only* receiving the abort token while the other receives a valid signature.

Exchange subprotocol. When there is no delay or blockage of network messages and neither party tries to cheat the other, A and B may exchange signatures by the following steps:

$$\begin{array}{ll}
A \rightarrow B & me_1 = \mathrm{PCS}_A(m, B, T) \\
B \rightarrow A & me_2 = \mathrm{PCS}_B(m, A, T) \\
A \rightarrow B & me_3 = \text{S-Sig}_A(m) \\
B \rightarrow A & me_4 = \text{S-Sig}_B(m)
\end{array}$$

In the first step of this subprotocol, A commits to the contractual text m by producing a private contract signature of m with B as the designated verifier.

The purpose of PCS is to convince B that A signed m, while depriving B of the possibility to prove this to an outside party. In the second step, B replies with its own PCS of m with A as the designated verifier. Finally, A and B exchange their actual, universally-verifiable signatures of m. At end of the exchange, both A and B obtain a signed contract of the form $\{\text{S-Sig}_A(m), \text{S-Sig}_B(m)\}$.

Abort subprotocol. The initiator A may attempt to abort the exchange. An honest A may do this if a reply from B is not received within a reasonable amount of time. To abort, A sends an abort request to T by signing the contractual text m together with the identities of the protocol participants and *abort*. The exact format of *abort* is not specified in [GJM99]; we assume that it is some predefined bit string.

Here are the steps of the abort subprotocol, with further description of T's action below.

$$
\begin{array}{lll}
A \to T & ma_1 = \text{S-Sig}_A(m, A, B, \mathit{abort}) & \\
T \to A & ma_2 = \text{Has } A \text{ or } B \text{ resolved?} & \\
 & \qquad\quad \text{Yes}: \text{S-Sig}_B(m) & \text{if } B \text{ had resolved, or} \\
 & \qquad\qquad\quad\ \ \text{TP-Sig}_B(m) & \text{if } A \text{ had resolved} \\
 & \qquad\quad \text{No}: \text{S-Sig}_T(ma_1) & \\
 & \qquad\qquad\quad\ \ \mathit{aborted} := \text{true} &
\end{array}
$$

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 A. 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 A's and B's universally-verifiable signatures of m. Therefore, in response to A's abort request, T forwards A either $\text{S-Sig}_B(m)$ or $\text{TP-Sig}_B(m)$, either of which can serve as a proof that B indeed signed m.

Since T stores the result of aborting (indicated by $\mathit{aborted} := \text{true}$) in its permanent database, an abort token is effectively a promise by T that it will not resolve this instance of the protocol in the future. As mentioned above, an abort token is *not* a proof that the exchange has been aborted, as the parties can complete contract signing without involving T if they follow the *exchange* subprotocol.

It is useful to bear in mind that while an honest A may send an abort request to T if she does not receive me_2 within a reasonable time, that there is no guarantee that A will be able to abort. Note also that even though B is not allowed to send abort requests to T, this does not put B at a disadvantage since it has the option of simply ignoring all messages from A.

Resolve subprotocol. 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, B can send a resolve request at any time after receiving me_1, and A 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.

Below, we show the resolve protocol between B and T. The protocol between A and T is symmetric.

$$
\begin{array}{ll}
B \to T & mr_1 = \mathrm{PCS}_A(m, B, T), \text{S-Sig}_B(m) \\
T \to B & mr_2 = \text{Has } A \text{ aborted?} \\
& \qquad \text{Yes : Send S-Sig}_T(\text{S-Sig}_A(m, A, B, \textit{abort})) \\
& \qquad \text{No : Has } A \text{ resolved?} \\
& \qquad\qquad \text{Yes : Send S-Sig}_A(m) \\
& \qquad\qquad \text{No : Store S-Sig}_B(m) \\
& \qquad\qquad\qquad \text{Convert } \mathrm{PCS}_A(m, B, T) \text{ into TP-Sig}_A(m) \\
& \qquad\qquad\qquad \text{Send TP-Sig}_A(m) \\
& \qquad\qquad \textit{resolved} := \text{true}
\end{array}
$$

The first request received by T determines the permanent status of the protocol. After T resolves or aborts the protocol for the first time, it should send consistent replies in response to all future requests. If the first request to reach T is an abort request from A, T's response to all requests will be the abort token. If the first request to reach T is a resolve request from A or B, T's response to all requests will be a signed contract. This leads to an implicit race condition which is not, however, a violation of fairness as defined in section 3.4.

3.4 Correctness conditions

The designers claim that the GJM protocol has the following properties:

Completeness. A restricted adversary cannot prevent a set of correct participants from obtaining a valid signature of a contract. The restricted adversary has signing oracles that can be queried on any message except the contractual text m and can arbitrarily schedule messages from participants to T, but cannot delay messages between the correct participants enough to cause any timeouts.

Fairness. The GJM protocol satisfies the following fairness conditions:

- It is impossible for a corrupt participant to obtain a valid contract without allowing the remaining participant to also obtain a valid contract.
- Once an honest participant obtains a cancellation message (*i.e.*, an abort token) from the trusted third party T, it is impossible for any other participant to obtain a valid contract.
- Every honest participant is guaranteed to complete the protocol.

Abuse-freeness. It is impossible for a protocol participant, at any point in the protocol, to be able to prove to an outside party that he has the power to choose between aborting and successfully completing the contract. One of the main contributions of [GJM99] is to introduce the notion of abuse-freeness to electronic contract signing.

Trusted third party accountability. If one of the parties is cheated because of T's misbehavior, the cheated party should be able to prove to an outside arbiter that T misbehaved. It is not specified precisely in [GJM99] what can serve as a proof of misbehavior, but typically such proof consists of two contradictory messages signed by T, *e.g.*, an abort token and a converted PCS signature of the same text m [ASW98b]. Since the steps of the protocol do not allow T to both abort and resolve the protocol, any PCS conversion performed by T after it aborted the protocol (and vice versa) serves as a proof of T's misbehavior.

There are actually two versions of the GJM protocol, one providing third party accountability and the other not. The difference between the two protocols lies in two versions of PCS. In our analysis, we focus on the case when the PCS scheme provides third-party accountability, *i.e.*, the distributions of $\text{S-Sig}_A(m)$ and $\text{TP-Sig}_A(m)$ are disjoint, and thus it is possible for the verifier to distinguish whether the signature is a "real" signature of A, or a PCS of A converted by T.

4 Analysis

In order to search for protocol errors, we implemented the exchange, abort and resolve subprotocols in the Murφ language. The protocol was combined with the standard intruder model described in section 2.1, modified in certain ways to account for the reliability of communication with the trusted third party. Most of the correctness conditions of section 3.4 were stated as Murφ invariants. During state exploration, Murφ checks that each invariant holds in every reachable state. The one exception is that abuse-freeness cannot be trivially represented as a state invariant.

We discuss the modeling of corrupt protocol participants and a partial method for verifying abuse-freeness in section 4.1. The subsequent subsections discuss the analysis of each protocol correctness condition in turn.

4.1 Modeling issues

Fair exchange protocols must protect an honest participant from being cheated by a malicious counterpart. Therefore, analysis of a fair exchange protocol must consider the possibility of one or more participants becoming corrupt and cooperating with the intruder.

Modeling corrupt participants There are several ways to model a corrupt protocol participant in Murφ. In our analysis [SM00] of the ASW optimistic contract signing protocol [ASW98a], we assumed that corrupt participants share their private key with the intruder, enabling the intruder to sign and decrypt messages on their behalf. This is equivalent to the intruder using the corrupt party as an oracle for signing and decrypting messages with its private key. We will call such collaboration with the intruder *strong corruption*.

A weaker form of corruption occurs when a protocol participant does not share its key with the intruder, and does not sign any messages it is not supposed to sign in the normal course of the protocol. However, it may be willing to engage the intruder's help in obtaining an unfair advantage in the exchange or contract signing process. This may involve accepting messages from the intruder and lying to an outside party about their source, *e.g.*, by claiming that they arrived from the protocol counterpart or T through the standard communication channels. We will call this *weak corruption*.

A weakly corrupt protocol participant is akin to a fence who is willing to accept hot goods without asking too many questions but will not do anything overtly illegal himself. A contract signing protocol that does not protect an honest participant from being cheated by a weakly corrupt counterpart defeats its own purpose and is largely useless. In the real world, it is impossible to be sure that an untrusted agent is not weakly corrupt, *i.e.*, that it is not acting in collusion with the intruder who has control over the public network on which the contract is negotiated.

The weakest form of corruption is the case when a participant, perhaps unintentionally, gives the intruder an ability to monitor (but not to modify or re-schedule) all incoming network traffic. This kind of corruption does not require that the corrupt party has a malicious intent. All the intruder needs is an oversight in network protection. For example, careless disposal of incoming messages may enable the intruder to root through the garbage and read all discarded messages. We will call this form of corruption *accidental corruption*.

Modeling power and abuse-freeness Our approach to verifying whether the protocol is abuse-free consists of two parts. First, we use Murφ to determine whether any protocol participant possesses the power to determine the outcome of the protocol regardless of the actions of the other party, assuming the other party is honest and genuinely interested in signing the contract. This is done by augmenting the system with an additional outside party we call the *Challenger*.

In order to verify whether a participant P has the power at some point in the protocol, we have it send a message to the challenger asserting its control over the outcome. The Challenger then nondeterministically chooses a desired outcome: abort or successful contract completion. (It is a consequence of fairness that there are only two possible outcomes: either T aborts and no one receives a signed contract or both parties receive a signed contract.)

After receiving the Challenger's request, P has to interact with the honest participant in such a way so as to drive the protocol to the requested outcome. If there exists a trace in which the outcome of the protocol is not consistent with that requested by the Challenger we conclude that P does not possess the power to determine the outcome. The key idea here is that determining whether

P satisfies the Challenger's request is a state invariant and can be verified by Murφ.

The second part of abuse is that a participant P with the power to determine the outcome must be able to prove this to an outside arbiter. However, we have not formulated a straightforward way of verifying properties such as "P can prove something" in Murφ. Therefore, we have only analyzed this part of the protocol by informal means.

Our analysis of abuse-freeness of the original and repaired GJM protocols can be found in sections 4.5 and 5, respectively.

4.2 Completeness

To verify the completeness guarantee (section 3.4), we used Murφ to analyze the protocol under the assumption that neither protocol participants, nor the trusted third party T are corrupt. We also restricted the intruder by requiring it to forward all messages originating from protocol participants to their intended recipients, and assumed that the channels between the participants and T are completely private, *i.e.*, the intruder cannot eavesdrop on the traffic or introduce new messages into the channels.

Under these restrictions, Murφ failed to find an attack that would prevent the participants from obtaining valid signatures of the contract. Therefore, our analysis confirms that the GJM protocol is indeed complete modulo limitations of the Murφ model (see section 2.1).

4.3 Fairness

First, we analyzed the protocol under the assumption that both participants are honest, *i.e.*, neither tries to cheat the other. This also implies that neither participant knowingly cooperates with the intruder. Murφ discovered that the intruder can achieve the following:

- Force A to submit an abort request to T by intercepting me_2.
- Prevent A from aborting the protocol by delaying A's abort request to T until B times out waiting for me_3 and submits a resolve request to T. Then A will receive B's signature in response to its abort request.
- Force B (respectively, A) to submit a resolve request to T by intercepting me_3 (me_4).

Murφ also discovered that A can use the protocol to obtain *both* an abort token signed by T and a valid contract signed by B. To do so, A executes the *exchange* subprotocol with B and then the *abort* subprotocol with T. As a result, B obtains A's signature, while A obtains B's signature and T's abort token.

There is also an important difference between the GJM protocol and the ASW protocol [ASW98a]. In the latter, the intruder can directly resolve the protocol by submitting a resolve request to T once both me_1 and me_2 have been sent into the network as part of the *exchange* subprotocol. This is impossible in the GJM protocol since resolve requests must include the originating party's signature on the contract which the intruder cannot compute without cooperating with that party.

None of the above is a violation of fairness as defined in section 3.4.

The main purpose of fair exchange protocols, including contract signing protocols such as the GJM protocol, is to protect an honest protocol participant from being cheated by a misbehaving counterparty. Therefore, we focused on the case when at least one of the protocol participants is malicious or corrupt, and used Murφ to analyze the GJM protocol under various assumptions about the corruptness of protocol participants and the security of the communication channels between the participants and the trusted third party T. For brevity, we omit the discussion of all combinations, and concentrate on the most interesting insights about the protocol revealed by our analysis.

Weakly corrupt A, intruder monitors $B \to T$ channel We analyzed the protocol under the assumption that party A is malicious, *i.e.*, its intention is to cheat B by obtaining B's signature of the contractual text m without releasing its own signature. A is weakly corrupt: it is willing to engage the intruder's help in obtaining B's signature, but will not sign or decrypt messages for the intruder.

The intruder I is assumed to have the ability to eavesdrop on and delay messages sent from B to T, but not to modify or remove them. Below we analyze the protocol under the assumption that the communication channel between B and T is inaccessible to the intruder.

Under these assumptions, Murφ uncovered the following attack:

$A \to B \qquad me_1 = \mathrm{PCS}_A(m, B, T)$
$B \to A \qquad me_2 = \mathrm{PCS}_B(m, A, T)$
I intercepts me_2, or A receives and discards it

$A \to T \qquad ma_1 = \text{S-Sig}_A(m, A, B, \textit{abort})$
$B \to T \qquad mr_1 = \mathrm{PCS}_A(m, B, T), \text{S-Sig}_B(m)$
I eavesdrops on mr_1, learns $\text{S-Sig}_B(m)$, delays mr_1 until T receives ma_1

$T \to A \qquad ma_2 = \text{S-Sig}_T(\text{S-Sig}_A(m, A, B, \textit{abort}))$
I intercepts ma_2, or A receives and hides it

$T \to B \qquad mr_2 = \text{S-Sig}_T(\text{S-Sig}_A(m, A, B, \textit{abort}))$
$I \to A \qquad \text{S-Sig}_B(m, A, T)$

As a result, A obtains B's signature of the contract $\text{S-Sig}_B(m, A, T)$, while B obtains the abort token from T.

Recall the second fairness condition from section 3.4: once a correct participant (B) obtains an abort token from the trusted third party T, it should be impossible for any other participant to obtain a valid contract. Even though Murφ cannot currently be used to verify non-safety properties such as "it is impossible for a participant to obtain a valid contract," this condition can be approximated by the following safety invariant: "it is never the case that the correct participant possesses the abort token, while some other participant possesses a valid contract, if the abort token was received first." Clearly, the above attack violates this invariant.

The first fairness condition is violated as well: the corrupt participant (A) obtained a valid contract without allowing the remaining participant (B) to also obtain a valid contract. The reason for this is that the only information from A that B has in its possession is $\mathrm{PCS}_A(m, B, T)$ sent in message me_1. This PCS can be converted into a universally-verifiable signature either by A (who won't do this because it's corrupt), or by T (who won't do this because it has already aborted the protocol, and must send abort tokens in response to all requests). Therefore, B has no means to obtain A's universally-verifiable signature of the contractual text m. This condition, however, is not trivially reduced to a safety invariant and is thus difficult to verify with Murφ.

It is unclear whether this attack is a bona fide violation of fairness. The original paper [GJM99, p. 462] states that if one party shows the abort token, and the other a valid set of signatures $\text{S-Sig}_A(m), \text{S-Sig}_B(m)$, then the contract must be valid. Indeed, it can be argued that B implicitly agreed to sign the contract by sending its signature to T in message mr_1, even though it received an abort token in response. We do believe that this attack violates abuse-freeness (see section 4.5 below).

Weakly corrupt *A*, accidentally corrupt *T* In order to stage the attack described in the previous section, the intruder must be able to access the communication channel between B and T. The original paper [GJM99] specifies that communication between any participant and T is conducted over a private channel. In this case, the intruder will not be able to eavesdrop on message mr_1 sent by B to T in order to resolve the protocol, and will not be able to learn $\text{S-Sig}_B(m)$. In fact, even if B and T communicate over a public network, encrypting mr_1 with T's public key will prevent the intruder from splitting it into parts and reusing one of the parts to help A gain an unfair advantage. It is worth noting, however, that the protocol specification in [GJM99] does not require that mr_1 be encrypted.

Now consider the case when the $B \rightarrow T$ channel is secure, but T is *accidentally corrupt*, and I has passive access to all of its incoming communication (see section 4.1 for our definition of accidental corruptness). This does not require active cooperation with the intruder on the part of T, just negligence in handling messages it receives from protocol participants. I does not need the ability to split messages into parts, remove them from the network, or even insert its own messages into the network. Having passive access to T's communication with B is sufficient for I to learn $\text{S-Sig}_B(m)$ and divulge it to A. Therefore, the attack described succeeds in this case.

4.4 TTP accountability

Suppose that T is accidentally corrupt and I successfully stages the attack described in section 4.3, causing B to lose fairness as a result. Since we are analyzing a TTP-accountable version of the GJM protocol (see section 3.4), we would like to verify whether the trusted third party T can be held accountable. The original paper [GJM99] defines a TP-accountable PCS scheme, but does not give a precise definition of TTP accountability. Since the GJM protocol is closely related to the Asokan-Shoup-Waidner (ASW) optimistic contract signing protocol [ASW98b,ASW98a], for the purposes of our formal analysis we used the ASW definition of TTP accountability (called "verifiability of trusted third party" in [ASW98b,ASW98a]):

"Assuming the third party T can be forced to eventually send a valid reply to every request, the verifiability of trusted third party property requires that if T misbehaves, resulting in the loss of fairness for P, then P can prove the misbehavior of T to an arbiter (or verifier) in an external dispute."

Following the designers of the ASW protocol, we assume that the proof must consist of two inconsistent messages signed by T, *e.g.*, an abort token and a converted PCS (recall that in the TTP-accountable version of PCS, TP-Sig$_B(m)$ obtained as a result of T's conversion of PCS$_B(m)$ is distinct from S-Sig$_B(m)$). According to the protocol specification, T must process all requests on the first-come, first-served basis. Therefore, the first request received by T determines the status of the contract in perpetuity, and it should never be the case that T issues an abort token and a converted PCS signature for the same contract.

However, if B loses fairness as a result of T's accidental corruption, it has no means of proving to an outside party that T is corrupt. A is in possession of genuine S-Sig$_B(m)$, *not* a converted PCS. If A is willing to lie about the source of this signature, then B cannot pin the blame on T. The only message signed by T is the abort token, and in the absence of two inconsistent messages signed by T, it is unclear what B can use as a proof to hold T accountable.

Since abort requests are signed, B can prove that the abort token it received from T was originally generated by A. But protocol specification allows for the case when A obtains a valid signature of B after sending off its abort request. This may happen if, for example, T received B's resolve request before A's abort request, resolved the protocol, and forwarded B's signature in response to A's abort request. A can also claim that it received B's signature directly from B.

At best, B can argue that *either A, or T is lying*: either A is lying that it received B's signature from T in response to its abort request, or T is lying that it received A's abort request before B's resolve request (in the latter case, T would not have sent the abort token in response to B's request). This is a very weak form of TTP accountability - in effect, the cheated party in a 3-party protocol is arguing that one of the other two is lying.

We believe that the difference between the possibilities (A is corrupt, or T is corrupt) is too significant to allow any confusion between the two. The protocol is designed to withstand corrupt participants, so the fact that A is corrupt is fairly trivial. T, on the other hand, plays a crucial role due to its ability to resolve or abort contract signings, and any negligence or dishonesty on the part of T should be immediately detected and, if proved, should lead to revocation of T's authority to function as the trusted third party.

4.5 Abuse-freeness

As we mentioned above, it is unclear whether the attack described in section 4.3 violates fairness, since B actually signs the contractual text m, implicitly agreeing to the contract. Abuse-freeness, on the other hand, is clearly violated. After receiving S-Sig$_B(m)$ from the intruder and S-Sig$_T$(S-Sig$_A(m, A, B, \mathit{abort})$) from T, A is free to decide whether to enforce the contract using the former, or consider it aborted using the latter. A can present both messages to an outside party, thus proving that it has the power to abort or successfully complete the protocol. Therefore, the GJM protocol is not abuse-free in this case.

The argument about TTP accountability given in section 4.4 applies to abuse-freeness as well as to fairness. If B is abused by A as a consequence of T's accidental corruption, B cannot prove to an outside arbiter that T misbehaved.

5 Repairing the Protocol

The basic error in the GJM protocol can be attributed to the fact that data sent in the resolve protocol are exactly the same as data sent in the exchange protocol. The GJM protocol can therefore be repaired by replacing the conventional signature in each resolve request with PCS. This was independently suggested by the authors of the protocol after we brought the attack described in section 4.3 to their attention [Mac99].

In the repaired protocol, resolve requests from B to T will have the following form (requests from A to T are symmetric):

$$mr_1 = \mathrm{PCS}_A(m, B, T),\ \mathbf{PCS_B(m, A, T)}$$

Our analysis of the repaired protocol did not uncover any attacks. Murφ confirmed that B still has the power to determine the outcome of the protocol after receiving the first message from A (see section 4.1). However, the only information in B's possession at this point is $\mathrm{PCS}_A(m, B, T)$, and B cannot use it to prove anything to an outside arbiter due to the designated verifier property of PCS (see section 3.2). We conclude that the repaired protocol is abuse-free. By contrast, the ASW protocol is not abuse-free. In the ASW protocol, B, too, has the power to determine the outcome after the first message received from A, but since universally-verifiable signatures are used, this power can be proved to an outside arbiter.

Unlike the original protocol, the repaired protocol is TTP-accountable. In the repaired protocol, T never receives universally-verifiable signatures of the contract from either A, or B. Any universally-verifiable signature leaked by corrupt T must be the result of PCS conversion, and its origin can be traced to T if the TTP-accountable version of PCS is used.

Murφ analysis indicates that the private channel assumption for communication between protocol participants and T can be relaxed. Even if the intruder can eavesdrop on messages exchanged with T, the protocol is still fair and abuse-free as long as the channels are *resilient*, *i.e.*, every message is guaranteed to eventually reach its intended recipient. This is significant because this implies that the repaired protocol does not need to operate on top of a secrecy protocol, or use any form of encryption in order to guarantee fairness. The protocol can still be subject to cryptographic attacks on PCS and signature schemes and/or other attacks that could not have been discovered in the Murφ model.

Additional analysis of the repaired protocol has been performed by Satyaki Das using the new-generation Murφ tool that relies on predicate abstractions to analyze infinite state spaces. It did not discover any attacks on an arbitrary number of protocol instances executed by different principals.

6 Conclusions

This paper shows how a finite-state analysis tool can be used to study a proposed abuse-free contract signing protocol and discover potential attacks and weaknesses. Our main results are the discovery of an error and a relatively simple change to the resolve subprotocol that produces a correct, abuse-free contract signing protocol. In addition, our Murφ-based analysis indicates that private channel assumptions can be relaxed.

In order to carry out our automated analysis, we needed to augment the system with an outside observer called the Challenger. The role of the Challenger is to nondeterministically challenge one party to demonstrate that this party has control over the outcome of the protocol. This method may be useful for verifying control-related properties of other protocols.

Fair exchange protocols are a new area of application for formal methods, and specification of protocol guarantees in the form suitable for automatic verification is still a challenge, especially in the case of such non-trivial properties as trusted third party accountability and abuse-freeness. We do believe that as online fair exchange and contract signing protocols gain increasing acceptance and a correspondingly high level of assurance is expected from them, formal techniques such as finite-state analysis will prove a useful tool for uncovering interesting insights and non-obvious attacks.

References

[ASW98a] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 86–99, 1998.

[ASW98b] N. Asokan, V. Shoup, and M. Waidner. Fair exchange of digital signatures. Technical Report RZ2973, IBM Research Report. Extended abstract in Eurocrypt '98, 1998.

[BCDvdG87] E. F. Brickell, D. Chaum, I. B. Damgard, and J. van de Graaf. Gradual and verifiable release of a secret. In *Proc. Advances in Cryptology – Crypto '87*, pages 156–166, 1987.

[BDM98] Feng Bao, R. H. Deng, and Wenbo Mao. Efficient and practical fair exchange protocols with off-line TTP. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 77–85, 1998.

[BOGMR90] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[Bol97] D. Bolignano. Towards a mechanization of cryptographic protocol verification. In *Proc. 9th International Conference on Computer Aided Verification*, pages 131–142, 1997.

[BT94] A. Bahreman and J. D. Tygar. Certified electronic mail. In *Proc. Internet Society Symposium on Network and Distributed Systems Security*, pages 3–19, 1994.

[CFN88] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Proc. Advances in Cryptology – Crypto '88*, pages 319–327, 1988.

[CTS95] B. Cox, J. D. Tygar, and M. Sirbu. NetBill security and transaction protocol. In *Proc. 1st USENIX Workshop on Electronic Commerce*, pages 77–88, 1995.

[DGLW96] R. H. Deng, Li Gong, A. A. Lazar, and Weiguo Wang. Practical protocols for certified electronic mail. *J. Network and Systems Management*, 4(3):279–297, 1996.

[Dil96] D. Dill. The Murφ verification system. In *Proc. 8th International Conference on Computer Aided Verification*, pages 390–393, 1996.

[FR97] M. Franklin and M. Reiter. Fair exchange with a semi-trusted third party. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 1–6. ACM Press, 1997.

[GJM99] J. A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Proc. Advances in Cryptology – Crypto '99*, pages 449–466, 1999.

[HTWW96] N. Heintze, J. D. Tygar, J. M. Wing, and H.-C. Wong. Model checking electronic commerce protocols. In *Proc. USENIX 1996 Workshop on Electronic Commerce*, pages 147–164, 1996.

[KMM94] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *J. Cryptology*, 7(2):79–130, 1994.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *Proc. 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166. Springer-Verlag, 1996.

[Mac99] P. MacKenzie. Email communication, September 23, 1999.

[MCJ97] W. Marrero, E. M. Clarke, and S. Jha. Model checking for security protocols. Technical Report CMU-SCS-97-139, Carnegie Mellon University, May 1997.

[Mea96a] C. Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. In *Proc. European Symposium On Research In Computer Security*, pages 365–384. Springer-Verlag, 1996.

[Mea96b] C. Meadows. The NRL Protocol Analyzer: An overview. *J. Logic Programming*, 26(2):113–131, 1996.

[MMS97] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murφ. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 141–151. IEEE Computer Society Press, 1997.

[MSS98] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Proc. 7th USENIX Security Symposium*, pages 201–215, 1998.

[Pau98] L. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.

[Ros95] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *Proc. 8th IEEE Computer Security Foundations Workshop*, pages 98–107. IEEE Computer Society Press, 1995.

[SM00] V. Shmatikov and J. C. Mitchell. Analysis of a fair exchange protocol. In *Proc. Internet Society Symposium on Network and Distributed Systems Security*, 2000. to appear.

[SS98] V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *Proc. 11th IEEE Computer Security Foundations Workshop*, pages 106–115, 1998.

[ZG96] J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 55–61. IEEE Computer Society Press, 1996.