Analysis of a Fair Exchange Protocol

Vitaly Shmatikov John C. Mitchell

Computer Science Department
Stanford University
Stanford, CA 94305-9045

{shmat,jcm}@cs.stanford.edu

Abstract

We analyze an optimistic contract signing protocol of Asokan, Shoup, and Waidner as a case study in the applicability of formal methods to verification of fair exchange protocols. After discussing the challenges involved in formalizing fairness, we use Murφ, a finite-state analysis tool, to discover a weakness in the protocol that enables a malicious participant to produce inconsistent versions of the contract or mount a replay attack. We show that the protocol can be repaired, and that the confidentiality assumption about the communication channels may be relaxed while preserving security against the conventional Dolev-Yao intruder.

1. Introduction

With continuing growth of electronic commerce on the Internet, the issues of trust and fairness are becoming increasingly important. The current proliferation of online auction venues, Internet shopping malls, and other sites that act as brokers between individual users and/or commercial entities makes it difficult, if not altogether infeasible, for a user to establish the credibility of a counterparty in a commercial transaction on the Internet. E-commerce protocols are increasingly required to provide some sort of mutual guarantees to protocol participants, ensure fairness, or furnish participants with a proof that a transaction has taken place.

Formal methods have been widely used to analyze the security properties of key exchange and authentication protocols [14, 22, 18, 5, 21]. In particular, finite-state analysis has been successfully applied to protocols such as Needham-Schroeder, Kerberos, SSL, and others [15, 17, 16, 19, 20]. However, less attention has been paid to fair exchange and related protocols.

In this paper, we analyze the optimistic contract signing protocol of Asokan, Shoup, and Waidner [1], which we shall refer to as the ASW protocol. In contrast to security protocols that involve secrecy and authentication, the correctness conditions for the contract signing protocol involve mutual fairness and verifiability of third party. These conditions present interesting challenges for a formal analysis tool. In particular, third party verifiability is difficult to formalize as a protocol invariant. Other challenges include the need to consider intentionally malicious protocol participants and the fact that some of the properties claimed for the protocol depend on resilience of the communication channels, *i.e.*, the properties only hold if all messages are eventually delivered to their intended recipients. The protocol is also interesting because it contains several interdependent subprotocols that must be analyzed together. While Heintze et al. [13] have used the FDR model checker to verify properties of the NetBill [8] and Digicash [7] protocols, the correctness conditions they establish are different in character from the ones we consider here.

We use Murφ, a finite-state analysis tool, to discover a potential weakness in the protocol. This weakness enables a malicious protocol participant to obtain a valid contract while the honest participant, even if it resorts to the help of the trusted third party, can only obtain a replacement contract which is inconsistent with the one possessed by the malicious participant. The same weakness also allows the intruder to stage a replay attack. While it is unclear if this violates the intentions of the protocol designers, it seems clear that the protocol would be more useful if these attacks were not possible. Fortunately, a small change to one of the messages prevents both attacks and therefore improves the protocol. We also show that some assumptions about the communication channels can be relaxed without violating fairness or other intended properties of the protocol.

We start by giving an overview of Murφ (section 2) and discussing the general notion of fair exchange and the challenges it presents to finite-state analysis (section 3). In section 4, we describe the contract signing protocol. Section 5 contains the results of Murφ analysis. In section 6, we suggest modifications to the protocol that prevent the attacks discovered by Murφ, and summarize our conclusions in section 7.

2. Overview of Murφ

Murφ [10] is a verification tool for analyzing finite-state systems. Originally developed for hardware verification, Murφ has been successfully used for analyzing security protocols [19, 20, 23]. The Murφ input language is a simple high-level language for describing nondeterministic finite-state machines. The input model consists of the description of variables that define the state of the system and a set of guarded rules that represent actions. While there is no explicit notion of process, a process can be implicitly modeled by a set of related rules. Communication between processes is modeled by shared variables. The Murφ system automatically checks, by explicit state enumeration, if every reachable state of the model satisfies a given set of invariants.

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 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). In the rest of the paper, we refer to this intruder model as the Dolev-Yao intruder, following [11].

If Murφ finds a reachable state in which an invariant is violated, the tool exhibits the attack by printing the sequence of steps from the start state to the error state. However, when Murφ fails to find an invariant violation, this is not a proof that the protocol is correct. Since Murφ can only consider finite models, only a bounded number of protocol instances can be analyzed. Therefore, an attack that requires a large number of protocol instances will not be discovered when a smaller number is considered. Certain kinds of attacks, such as attacks involving cryptanalysis, are also beyond the scope of the model.

3. Fair Exchange

Intuitively, a protocol is *fair* if no protocol participant can gain an advantage over other participants by misbehaving. For example, a protocol in which two parties exchange one item for another is fair if it ensures that at the end of the exchange, either each party receives the item it expects, or neither receives any information about the other's item [1]. Fair exchange protocols are used for applications such as online payment systems [8], in which a payment is exchanged for an item of value, contract signing [4, 1], in which parties exchange commitments to a contractual text, certified electronic mail [2, 24, 9], and other purposes.

There are several categories of fair exchange protocols. Gradual exchange protocols [4, 6] work by having the parties release their items in small installments, thus ensuring that at any given moment the amount of knowledge on both sides is approximately the same. The drawback of this approach is that a large number of communication steps 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* [8, 24, 9]. 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 [12]. 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 [1, 3]. While these protocols still require a trusted third party, the third party is only needed when messages are delayed or one of the parties misbehaves. This may ease the communication bottleneck, making fair exchange protocols more practical for realistic applications.

Fair exchange protocols present several challenges for formal analysis in general, and finite-state techniques in particular. To begin with, the correctness conditions are often subtle and it is difficult to extract precise statements from informal protocol specification. There are also inherent subtleties in modeling dishonest parties and partially faulty communication channels. Since fairness involves protection against the actions of a malicious participant, it is necessary to allow protocol participants to become malicious and try to cheat the honest parties. In our analysis, we model dishonest participants by having them reveal their private information to the intruder. This appears to be an accurate method, although it depends on giving the intruder sufficient power to use the revealed information effectively. The communication channels present another challenge. In other protocols, unreliable communication channels are typically modeled by giving the intruder full control over the channel. In the traditional approach, the intruder is able to intercept any message, break messages into components, and create new messages within the cryptographic constraints of the formal intruder model. Fair exchange protocols, however, may rely on communication assumptions that would be violated by the traditional intruder model. For example, it may be required that all messages deposited in the channel are eventually delivered, but the intruder may still be able to delay and re-schedule them.

4. Optimistic Contract Signing Protocol

This section describes the optimistic contract signing protocol designed by Asokan, Shoup, and Waidner [1]. We start by describing the objectives of the protocol and the assumptions under which the protocol operates. We then define the protocol itself and formalize the correctness conditions posed by the designers of the protocol. The notation has been changed from the original paper to facilitate explanation.

A common point of confusion about this protocol is the notion of "contract." In general, one might expect a contract to be a pair of digital signatures of an agreed upon text, one signature from each party negotiating the contract. In the ASW protocol, normal termination without use of the third party will produce a contract that contains two digital signatures and additional data produced in the run of the protocol. However, the contracts produced by the third party are not necessarily of this form. In order to understand the ASW protocol, it is important to keep in mind not only the steps of each subprotocol (discussed below), but also the forms of contract that the protocol designers have established for the protocol.

4.1. Objectives

The optimistic contract signing protocol is designed to enable two parties, called O (originator) and R (responder), to obtain each other's commitment on a previously agreed contractual text. The protocol is asynchronous, allowing either O, or R to contact the third party. The third party may decide, on the basis of communication it has received, whether to issue a replacement contract or an abort token. Abort tokens are *not* a proof that the exchange has been canceled (see sections 4.3 and 5.1.4 below) but a promise by T not to respond differently in the future.

4.2. Assumptions

The protocol uses digital signatures and a hash function. We write $\mathrm{sig}_i(\ldots)$ for a message signed by party i and assume that all protocol participants have the ability verify signatures produced by any party. We also assume that there exists a collision-resistant one-way hash function, $\mathrm{H}()$.

Prior to executing the protocol, we assume that the parties agree on each other's identity, the identity of the trusted third party T, and the contractual text. It is also assumed that every protocol participant knows everybody else's signature verification key. This implies that the protocol must be preceded by the "handshake" phase in which a key exchange and/or authentication protocol is executed to establish the shared initial knowledge. However, since it is not necessary for the handshake protocol to guarantee fairness, we do not consider it as part of this study.

The original paper [1] is self-contradictory in its description of the assumptions about the communication channels between protocol participants. It first states that the communication channels between any two protocol participants are assumed to be *confidential*, *i.e.*, eavesdroppers will not be able to determine the contents of messages traveling through these channels. This can be achieved by encrypting all messages with the intended recipient's public key. Later, however, the paper states that no assumptions are made about the communication channel between O and R. In [1], it is also assumed that the channels between each participant and the trusted third party T is *resilient*, *i.e.*, any message deposited into the channel will eventually be delivered to its intended recipient. However, there are no time guarantees: the intruder can succeed in delaying messages by an arbitrary, but finite amount of time. In section 5 below, we analyze the protocol under various assumptions about the quality of communication channels.

Implicit in the protocol description [1] is the assumption that the trusted third party T must maintain a permanent database with the status of every protocol run that it has ever been asked to abort or resolve. (Each run can be identified by the first message me_1 — see below.) Abort and resolve requests are processed by T on the first-come, first-served basis. Therefore, in order to ensure fairness, T must always be able to determine whether a particular instance of the protocol has been aborted or resolved already.

4.3. Protocol

The optimistic contract signing protocol consists of three interdependent subprotocols: *exchange*, *abort*, and *resolve*. The parties (O and R) start the exchange by following the *exchange* subprotocol. If both O and R are honest and messages are received in time to satisfy both parties, both obtain 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. Intuitively, an honest O might choose to do this if a response from R is not received after a reasonable waiting period. However, neither time, nor any conditions governing this decision are included in the protocol definition. To abort the transaction, O executes the *abort* subprotocol with T. Finally, either O, or R may individually request that T resolve the exchange by issuing a replacement contract. They request this action by executing the *resolve* subprotocol with T.

At the end of the protocol, each party is guaranteed to end up with a valid contract or an abort token. As described briefly above, the protocol definition in [1] provides two forms of contract:

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

where me_1, me_2, N_O, N_R are defined below. Note that the protocol definition does *not* consider a signed contractual text by itself a valid contract.

Abort tokens have the following form:

$$\mathrm{sig}_T\{\mathit{aborted}, ma_1\}$$

where ma_1 is defined below.

An abort token should *not* be interpreted as a proof that the exchange has been canceled. The protocol does not prevent a dishonest O from obtaining an abort token after signing the contract with R (in this case, O may have both the abort token and the contract, while R only has the contract). The protocol is designed, however, to prevent one party from receiving *only* the abort token, in any situation where the other can receive a valid contract.

Exchange subprotocol. As mentioned earlier, it is assumed that prior to initiating the exchange, the two parties agree on the contractual text (*text*) and the identity of the trusted third party (T). They are also assumed to know each other's public verification key. Specifically, O knows the key V_R that can be used to verify messages signed by R, and R knows V_O.

When communication is not blocked or delayed and neither party tries to cheat the other, O and R execute the following subprotocol:

$$\begin{array}{ll} O \to R & me_1 = \mathrm{sig}_O\{V_O, V_R, T, \mathit{text}, \mathrm{H}(N_O)\} \\ R \to O & me_2 = \mathrm{sig}_R\{me_1, \mathrm{H}(N_R)\} \\ O \to R & me_3 = N_O \\ R \to O & me_4 = N_R \end{array}$$

In the first step of the subprotocol, O commits to the contractual text by hashing a random number N_O, and signing a message that contains both $\mathrm{H}(N_O)$ and *text*. N_O is called the *contract authenticator*. While O does not actually reveal the value of the contract authenticator to the recipient of message me_1, O is committed to it. As in a standard commitment protocol, we assume that the hash function is 2nd-preimage resistant: it is not computationally feasible for O to find a different number N'_O such that $\mathrm{H}(N'_O) = \mathrm{H}(N_O)$.

In the second step, R replies with its own commitment. Finally, O and R exchange the actual contract authenticators. At the end of this subprotocol, both O and R obtain a standard contract of the form $\{me_1, N_O, me_2, N_R\}$.

Abort subprotocol. An honest or dishonest O may attempt to cancel the exchange. This allows an honest O to time out, if a response from R is not received. To cancel, O sends an abort request to the trusted third party T by signing the first message me_1 of the exchange together with *aborted*. It is not clear from the protocol description [1] what the exact format of *aborted* is, but for the purposes of our analysis we assume that it is a predefined bit string.

If T has not previously been requested to resolve this instance of the protocol, T marks me_1 as aborted in its permanent database and sends an abort token to O. If me_1 is already marked as resolved, *i.e.*, somebody had resolved the exchange with T before by submitting valid me_1 and me_2, T sends O a replacement contract.

$$\begin{array}{lll} O \to T & ma_1 = \mathrm{sig}_O\{\mathit{aborted}, me_1\} & \\ T \to O & ma_2 = \mathit{resolved}? & \\ & \quad\quad \text{Yes:} & \mathrm{sig}_T\{me_1, me_2\} \\ & \quad\quad \text{No:} & \mathrm{sig}_T\{\mathit{aborted}, ma_1\} \\ & & \mathit{aborted} := \mathrm{true} \end{array}$$

Although an honest O may send an abort request to T if it does not receive me_2 within a reasonable time, there is no guarantee that O will be able to abort. If the exchange has been already resolved by someone who knows both me_1 and me_2, T will not grant the abort request and will send O a replacement contract instead — even if O has not received me_2.

Note also that R cannot send a valid abort request to T since ma_1 has to be signed by the same key as me_1. This does not put R at a disadvantage since it has the option of simply ignoring all messages from O.

Resolve subprotocol. Either party may request that T resolve the exchange. In order to do so, the party must possess both me_1 and me_2. Therefore, R can send a resolve request at any time after receiving me_1, and O can do so at any time after receiving me_2. When T receives a resolve request, it checks whether me_1 is already marked as aborted. If it is, T replies with the abort token, otherwise it marks me_1 as resolved and generates a replacement contract by counter-signing the resolve request.

$$\begin{array}{lll} O, R \rightarrow T & mr_1 = \{me_1, me_2\} & \\ T \rightarrow O, R & mr_2 = \mathit{aborted}? & \\ & & \text{Yes}: \ \mathrm{sig}_T\{\mathit{aborted}, ma_1\} \\ & & \text{No}: \ \ \mathrm{sig}_T\{me_1, me_2\} \\ & & \qquad \mathit{resolved} := \text{true} \end{array}$$

Although this contract has a different form than the contract produced by the exchange subprotocol, the protocol design assumes that in any transaction requiring a contract, either form would be accepted as binding. In other words, the protocol designers consider the definition of contract to be part of the protocol specification and choose to use two forms of valid contract in their protocol.

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 identical replies in response to all future requests. If the first request to reach T is an abort request from O, T's response to all requests will be the abort token. If the first request to reach T is a resolve request from O or R, T's response to all requests will be the replacement contract. This leads to an implicit race condition which is not, however, a violation of fairness requirements as defined in section 4.4.2.

4.4. Correctness conditions

The designers make the following claims for the optimistic contract signing protocol:

4.4.1. Claim 1: If the communication channel between O and R is resilient, the protocol satisfies the following requirement:

Effectiveness. If both parties behave correctly and do not want to abandon the exchange, then when the protocol has completed, each has the other's commitment and authenticator, *i.e.*, O has $\mathrm{H}(N_R)$ and N_R, while R has $\mathrm{H}(N_O)$ and N_O.

4.4.2. Claim 2: If the communication channels between O and T, and R and T are resilient, the optimistic contract signing protocol satisfies the following requirements:

Strong fairness. When the protocol has completed, either both O and R have valid contracts, or neither one does.

Timeliness. At the beginning of the exchange, every participant can be sure that the protocol will be completed within finite time. At completion, the state of the exchange will either be final, or, in the words of the protocol designers, any changes to it will not "degrade the level of fairness" achieved by the participant so far. For example, if a party has not been cheated at the end of the protocol, it cannot be cheated later on.

Non-repudiability. After an effective exchange (see above), each participant P will be able to prove the origin of the valid contract it has received, and prove that P's protocol counterparty has received P's authenticator or a valid replacement contract from T.

Verifiability of third party. If the trusted third party T can be forced to eventually send a valid reply to every request, then any participant who is cheated as a result of T's misbehavior will be able to prove that T misbehaved in an external dispute.

There are no other guarantees claimed for the protocol. In particular, consider the following:

- After O sends off me_1, there is no guarantee that it will be able to abort the exchange. This has been verified by Murφ analysis: if R computes me_2 and executes the *resolve* subprotocol with T while the intruder delays the abort request from O to T, then T will issue a replacement contract in response to O's abort request. Therefore, O must accept the risk that the exchange will be resolved as soon as it releases its commitment into the network, even if it never hears from R! In this sense, the protocol is disadvantageous to O, since R can always ensure that the exchange will eventually be aborted by simply ignoring me_1.
- There is no provision in the protocol for R to request that the exchange be aborted. In this sense,

the protocol is disadvantageous to R. After R sends off its commitment as part of me_2, it has no option to abort the protocol. In contrast, O does have such an option, but no guarantee (see above).

5. Analysis

After implementing a Murφ model of the three subprotocols described in section 4.3, we combined it with an intruder model of the form described in section 2. Correctness conditions of section 4.4 were represented by invariants that must hold in every state reachable by the protocol. We then used Murφ to perform the finite-state analysis of the protocol.

Our first attempt to analyze the protocol failed because according to the protocol specification, the trusted third party T is always ready to accept abort and resolve requests. Therefore, if one of the parties is corrupt (*i.e.*, the intruder has access to its signing key — see section 5.1.4 below), then in every state of the protocol the intruder can generate a new resolve or, if O is the corrupt party, abort request and send it to T. The trusted third party will then add the request to its database, resulting in a new, larger state. This makes the state space of the protocol infinite. The only solution is to arbitrarily limit the number of times the intruder can generate a request to T in the course of one instance of the protocol. This restriction is not necessary if there are no corrupt parties, since there is only a finite number of frivolous requests that can be computed by the intruder. However, Murφ analysis is slowed down considerably if in every state there is an enabled rule allowing the intruder to send a request to T.

This section describes the results of our analysis with the intruder limited to no more than 2 requests to T per protocol instance. We are currently investigating the effects of increasing the bound on the number of intruder-generated requests.

5.1. Fairness

We started the analysis by verifying the strong fairness property of the protocol (see section 4.4). As a reminder, strong fairness guarantees that when the protocol has completed, either both protocol participants have valid contracts, or neither one does.

5.1.1. Confidential channels, one instance of the protocol: First, we analyzed one run, or instance of the protocol under the assumption that all communication channels are confidential. This prevents the intruder from learning anything from the messages as they pass through the network. The only operation the intruder can perform in this setting is to store a message and replay it later.

In fact, the protocol specification [1] says that the protocol provides fairness if the communication channel between O and R, or those between O, R and T, is *resilient*, *i.e.*, any message deposited in the channel will eventually be delivered to the recipient. Resilience is not a safety property and requires special effort to model with Murφ. Adding liveness properties to Murφ (*e.g.*, by means of rules that are enabled only in "final" states where no other rules apply) is a topic of current research.

For the purposes of this study, we made all protocol invariants conditional on the protocol's successful completion. Therefore, in order for an attack on the protocol to succeed, the intruder must deliver messages to their intended recipients so that the latter are convinced that they have successfully completed the protocol. As long as one of the parties is in a state where it's waiting for a message, the protocol is not complete, and there are no fairness guarantees. This approximation of resilience actually *strengthens* the intruder by not requiring it to eventually forward *all* intercepted messages to their intended recipients. Therefore, if Murφ did not find any attack on the protocol in our model, it would not have found any attacks in the model where the channels are both confidential and resilient.

Murφ did not discover any violations of strong fairness by analyzing a single instance of the protocol under the channel confidentiality assumption. It did discover that the intruder can achieve the following:

- Prevent O from aborting the protocol by delaying its abort request to T until R computes me_2, and then submitting me_1 and me_2 (ostensibly from R) to T, thus resolving the protocol. Then O will receive a replacement contract in response to its abort request.

- Force O to submit an abort request to T by intercepting me_2.

- Force R (respectively, O) to submit a resolve request to T by intercepting me_3 (me_4).

- Resolve the protocol directly 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.

None of the above, however, is a violation of strong fairness as defined in section 4.4.2.

5.1.2. Confidential channels, two instances of the protocol: After increasing the bound on the number of protocol instances, Murφ discovered the following replay attack:

I observes an instance of the protocol
$O \to R \;\; me_1 = \text{sig}_O\{V_O, V_R, T, text, \text{H}(N_O)\}$
$R \to O \;\; me_2 = \text{sig}_R\{me_1, \text{H}(N_R)\}$
$O \to R \;\; me_3 = N_O$
$R \to O \;\; me_4 = N_R$

Later . . .
$I \;\to R \;\; me_1 = \text{sig}_O\{V_O, V_R, T, text, \text{H}(N_O)\}$
$R \to O \;\; me_2' = \text{sig}_R\{me_1, \text{H}(N_R')\}$ $\langle I$ gets it$\rangle$
$I \;\to R \;\; me_3 = N_O$
$R \to O \;\; me_4' = N_R'$ $\langle I$ gets it$\rangle$

To stage this attack, the intruder must observe an instance of the protocol, recording all messages sent by O. After the protocol completes, the intruder can initiate another instance of the protocol by replaying the recorded me_1. R will respond with a new me_2', to which the intruder responds with the old me_3. The result of this attack is that the intruder can get R to commit to the text of an old contract with O without O's or T's knowledge.

The protocol as described in [1] contains no protection against this kind of attack. Perhaps this was a conscious decision on the part of the protocol designers who did not intend the protocol to be secure against replay attacks. If the contractual text contains a timestamp, expiration date, or some other information that might help in determining its freshness, R may be able to detect the attack. It can be argued that any well-written contract must contain such information. However, this should be stated explicitly as part of the protocol specification and not left for the protocol user to infer.

The attack we discovered is different from the simpler one in which a malicious R keeps the old contract to which O had previously committed and tries to reuse it. In case of our replay attack, the new contract is *different* from the old one. Recall that a standard contract is the combination of me_1, me_2, and contract authenticators: $\{me_1, me_2, N_O, N_R\}$. Since me_2 is different in the second instance of the protocol, the contract is different. This implies that O cannot even obtain a valid replacement contract by requesting it from the trusted third party since in order to do so, it needs me_2' which it never receives. In fact, O is not even aware that an exchange between R and the intruder has taken place.

The replay attack succeeds even if both O and R are honest. Suppose that O is a retailer who periodically purchases supplies from R online using the contract signing protocol. All purchase contracts are exactly the same, as is often the case in real life, and it is agreed (offline) that all contracts expire immediately upon fulfillment (*i.e.*, R receives the order, fills it, and forgets about it). Then the intruder can use the replay attack to impersonate O and submit a false purchase contract on its behalf, convincing R that O has committed to a new purchase and providing R with a false proof of O's commitment.

Note that there is no need for the intruder to involve the trusted third party in the protocol in order to stage the replay attack. This means that there will be no evidence of the attack such as could have been provided by a resolve request kept by T.

The main weakness of the protocol is the fact that O's message me_3 that contains the contract authenticator is sent in response to R's commitment message me_2 but is not related to it in any way, making it possible for the intruder to replay an old me_3. An easy fix that prevents the replay attack is described in section 6.

5.1.3. Standard channels: After repairing the protocol to prevent the replay attack, we performed Murφ analysis without the confidentiality assumption on the channels but still within the constraints of the standard Dolev-Yao intruder model (see section 2). Murφ did not discover any new attacks. This can be attributed to the fact that messages $me_{1,2}$, $ma_{1,2}$, and mr_2 are all signed, and mr_1 contains signed messages as its components. Assuming that every protocol participant knows everybody else's correct public key (this is a necessary requirement for the protocol to succeed even in the absence of the intruder), signatures prevents the intruder from modifying messages in transit. Since no signing keys are transmitted as part of the protocol, the intruder cannot gain the ability to sign messages unless one of the parties leaks its key. Therefore, the intruder is just as powerful as in the case of confidential channels.

This result suggests that the channel confidentiality assumption can be relaxed. The protocol ensures fairness even if the channels are controlled by a Dolev-Yao intruder.

5.1.4. Corrupt protocol participant: Finally, we analyzed the protocol under the assumption that one of the participants is corrupt. We modeled this by giving the intruder access to the corrupt party's private information such as its private key and its contract authenticator even before the latter is divulged as part of the *exchange* subprotocol.

Clearly, fairness for the corrupt party cannot be guaranteed in this case. For instance, if the intruder is able to sign messages with O's private key, it is then able to impersonate O in any exchange and convince R that O has committed to a contract. Protocol invariants must be formulated carefully so as to avoid spuriously flagging this situation as an error. The real question is whether sharing its private key with the intruder allows the corrupt participant to gain an unfair advantage over the *other* party.

Murφ discovered that a corrupt O can obtain *both* an abort token signed by T and a valid contract with R. In addition, O does not require assistance from the intruder to do this. It can simply execute the *exchange* subprotocol with R and then the *abort* subprotocol with T. As a result, R obtains a valid contract with O, while O obtains a contract with R and T's abort token. Since both parties have the contract, this is not a violation of fairness as defined in section 4.4.2, but it also implies that abort tokens may not be accepted as a proof that contract negotiation failed.

Murφ also discovered the following attack, in which a malicious R obtains a contract which is inconsistent with that obtained by O.

$O \rightarrow R \;\; me_1 = \mathrm{sig}_O\{V_O, V_R, T, \mathit{text}, \mathrm{H}(N_O)\}$

$R \rightarrow O \;\; me_2 = \mathrm{sig}_R\{me_1, \mathrm{H}(N_R)\}$

R computes new random N'_R and $me'_2 = \mathrm{sig}_R\{me_1, \mathrm{H}(N'_R)\}$, but keeps them secret

$O \rightarrow R \;\; me_3 = N_O$

R sends nothing

$O \rightarrow T \;\; ma_1 = \{me_1, me_2\}$

$T \rightarrow O \;\; ma_2 = \mathrm{sig}_T\{me_1, me_2\}$

In this attack, R computes two different responses me_2 and me'_2 to O's initial message me_1 using different random numbers N_R and N'_R. It sends out me_2 and keeps the other secret. After it receives O's contract authenticator N_O, R does not respond at all. It has already obtained a valid standard contract $\{me_1, N_O, me'_2, N'_R\}$. Since O does not receive me_4 from R, it requests trusted third party T to resolve the protocol. T issues a replacement contract by countersigning me_1 and me_2. However, O's contract is *different* from that possessed by R because it contains the hash of a different random number: N_R rather than N'_R.

Clearly, this is a problem, since each party possesses a valid contract, but the two contracts are inconsistent. Recall that the protocol employs a non-standard definition of contracts (section 4.3), according to which a valid contract is *more* than a signed contractual text. Even though the contractual texts in the two contracts are the same, the random numbers and commitments are different, and it is unclear how the contracts should be enforced or interpreted, given that both are valid according to the protocol specification. The original paper [1] does not say anything about how this situation should be handled.

The problem is caused by the same weakness of the protocol that makes the replay attack described in section 5.1.2 possible. O's contract authenticator N_O is sent in response to me_2 but is not explicitly linked to it. This enables R to use N_O with a different message me'_2 to form a valid contract without revealing its own commitment to O. The modification of the protocol described in section 6 prevents this attack.

5.2. Timeliness

Eventual completion of the protocol is guaranteed by channel resilience. Since we did not fully model resilience, it is possible for the intruder in our model to prevent the protocol from completing, but this attack is trivial.

The concept of "fairness degradation" is not defined in the paper [1] and thus difficult to formalize so that it can be checked by a finite-state analysis tool. Based on our informal understanding of the protocol, if fairness is achieved at the end of the protocol, then it cannot be lost since the protocol provides no means of invalidating a contract. If a party has the valid contract once the protocol completes, then it cannot be cheated regardless of whether the other party has the contract or not. If a party does not have a valid contract, then the other party must not have a contract either (otherwise, there is no fairness). The only remaining question is whether it is possible to reuse information from an instance of the protocol that did not result in a valid contract in another instance that does produce a valid contract (then even if the first instance of the protocol was fair, fairness will be lost in the second instance). Murφ did not find any attacks of this nature.

5.3. Non-repudiability

Non-repudiability condition (see section 4.4) requires that after an honest party completes the protocol, it must be able to prove the origin of the valid contract it receives. Since the ability to prove something is impossible to formalize, we did not attempt to verify non-repudiability with Murφ. One can use informal reasoning to conclude that since commitment messages are signed and it is assumed that the signature scheme is secure, O's signature on me_1, R's signature on me_2, and T's signature on mr_2 prove the origin of any valid contract, whether it is a standard contract computed by O and R, or a replacement contract issued by T. Non-repudiability of receipt is conditional on fairness: if O has a valid contract at the completion of the protocol, then R must have a valid contract, too (otherwise, there is no fairness). Therefore, R must have received O's contract authenticator or a replacement contract from T. R's non-repudiability of receipt is symmetric. Unfortunately, this sort of reasoning is difficult to verify with a finite-state tool.

5.4. Verifiability of trusted third party

The optimistic contract signing protocol is not intended to guarantee fairness if the trusted third party T is corrupt. However, *verifiability of third party* implies that if one of the participants loses fairness as a result of T's misbehavior, it should be able to prove this misbehavior to an independent arbiter.

Verifiability only holds if the trusted third party is guaranteed to send a valid response to all requests. Our Murφ model approximates this guarantee by making all protocol invariants, including verifiability of third party, conditional on the protocol's successful completion. Therefore, in order to succeed, any attack staged by the intruder, possibly acting in collusion with corrupt T, must involve generating a valid response to every request sent by an honest participant. Otherwise, the honest participant will not complete the protocol, and the attack will fail. Also, verifiability is only possible if O is notified when R tries to enforce a contract and vice versa. If a protocol participant does not know that it is being cheated, it cannot go after T to prove its misbehavior.

Before formulating a formal protocol invariant that could be verified with the help with Murφ, we had to determine what it means to be able to prove T's misbehavior. Based on our interpretation of the protocol description in [1], we believe that the cheated protocol participant can prove that T misbehaved if and only if it can produce two documents, both signed by T, that contradict each other. More specifically, the cheated participant must be able to demonstrate an abort token signed by T *and* a replacement contract for the same instance of the protocol, also signed by T. Since T is supposed to process all abort and resolve requests on the first-come, first-served basis and the initial request determines the status of the protocol in perpetuity, it should never be the case that T issues both an abort token and a replacement contract for the same instance of the protocol.

Based on the above interpretation, we believe that verifiability of third party is violated *if and only if* the following conditions hold (the conditions are formulated assuming that O is the cheated party; the conditions for R are symmetric):

- T is corrupt (modeled by giving the intruder access to T's signing key).
- R has O's contract authenticator.
- O has neither R's contract authenticator, nor a replacement contract signed by T.

If R has a replacement contract signed by T instead of a standard contract with O's contract authenticator, then T is always verifiable! Suppose that R tries to enforce its replacement contract. When O goes to T and requests to either abort, or resolve the protocol, T must send O a valid response. If T sends a replacement contract, then there is no fairness violation and O is not cheated since both parties possess the same contract. If T sends an abort token, then O is indeed cheated (since R has a contract and O does not), but O can then prove T's misbehavior by demonstrating its abort token and R's replacement contract, both signed by T.

However, if R has a standard contract with O's contract authenticator, then R's contract is *not* signed by T, and O cannot prove T's misbehavior since it cannot produce two inconsistent documents signed by T. This case satisfies the conditions listed above.

Murφ did not discover any states reachable by the protocol that satisfy the conditions. We conclude that the third party is indeed verifiable.

6. Repairing the Protocol

The optimistic contract signing protocol can be repaired so as to prevent the attacks described in sections 5.1.2 and 5.1.4 by explicitly linking message me_3 with message me_2. This is a standard technique to ensure that an old me_3 cannot be replayed by the intruder in response to a fresh me_2 and that R can obtain a standard contract only with the same contract authenticator that it has sent to O as part of me_2. A similar change must be made to me_4 to prevent a symmetric replay attack.

$$\begin{array}{lll} O \to R & me_1 = \mathrm{sig}_O\{V_O, V_R, T, \mathit{text}, \mathrm{H}(N_O)\} \\ R \to O & me_2 = \mathrm{sig}_R\{me_1, \mathrm{H}(N_R)\} \\ O \to R & me_3 = \mathbf{sig_O}\{N_O, \mathrm{H}(\mathbf{N_R})\} \\ R \to O & me_4 = \mathbf{sig_R}\{N_R, \mathrm{H}(\mathbf{N_O})\} \end{array}$$

7. Conclusions

Application of formal techniques to fair exchange protocols involves challenges that are not usually encountered in the analysis of secrecy and authentication protocols. Correctness conditions such as fairness and verifiability of third party are difficult to understand and formalize, communication channels must satisfy non-safety properties such as resilience, protocol participants may intentionally misbehave. This paper presents a case study in the formal analysis of fair exchange, using a finite-state analysis tool to verify an optimistic contract signing protocol and to discover gray areas and weaknesses. In addition to finding weaknesses in the protocol that can be exploited by a Dolev-Yao intruder, we suggest and analyze a repair that prevents these attacks. We believe that our work extends the area of applicability of finite-state analysis and provides an example of how formal methods can be profitably used to study a variety of security protocols.

References

[1] 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.

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

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

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

[11] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[12] 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.

[13] 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.

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

[15] 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.

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

[17] 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.

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

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.