Copyright 1998 IEEE. Published in the Proceedings of S&P'98, 3-6 May 1998 in Oakland, California.

Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from the IEEE. Contact: Manager, Copyrights and Permissions / IEEE Service Center / 445 Hoes Lane / P.O. Box 1331 / Piscataway, NJ 08855-1331, USA. Telephone: + Intl. 908-562-3966.

Errata

The following is a corrected version of the paper that appeared in the proceedings of the 1998 IEEE Security and Privacy Symposium. The corrections are as follows.

In Section 3.3, we indicated that T can be made invisible in the contract signing protocol by replacing instances of $h()$ with $Enc_T()$. This is incorrect. T can be made invisible for R by changing com_O from $h(o_O)$ to $Enc_T(o_O|V_O|V_R)$. A valid contract for R is always: $\{me_1, o_O, me_2, o_R\}$, where

$$me_1 = Sig_O(V_O, V_R, \mathsf{T}, text, com_O)$$
$$me_2 = Sig_R(me_1, com_R)$$
$$com_O = Enc_T(o_O|V_O|V_R),\ com_R = h(o_R)$$

The second instantiation for certified mail in Section 5.1 is changed similarly. Table 3 as follows:

Generic Protocol	Instantiation
i_O	mail
i_R	receipt text
d_O	not needed
d_R	receipt text
com_O	$Enc_T(\text{mail}\|key_O\|V_O\|V_R)$
com_R	$h(key_R)$
resolve for O	T issues replacement receipt
resolve for R	if com_O is correct, T recovers $mail, key_O$ in return for key_R

The protocol figure corresponding to this instantiation (Figure 4) has also been corrected accordingly.

Asynchronous Protocols for Optimistic Fair Exchange

N. Asokan Victor Shoup
Michael Waidner
IBM Research Division, Zurich Research Laboratory,
Säumerstrasse 4, CH 8803 Rüschlikon, Switzerland.
{aso,sho,wmi}@zurich.ibm.com

Abstract

The ***optimistic*** *approach of involving a third party only in the case of exceptions is a useful technique to build secure, yet practical fair exchange protocols. Previous solutions using this approach implicitly assumed that players had reliable communication channels to the third party. In this paper, we present a set of optimistic fair exchange protocols which tolerate temporary failures in the communication channels to the third party. A central feature of the protocols is that either player can asynchronously and unilaterally bring a protocol run to completion.*

1 Introduction

The crux of a commercial transaction is usually an exchange of one item for another. We can find various instances of the general exchange scenario in different types of commercial activity:

- In a **purchase**, a *payment* is exchanged for a *receipt* of some valuable item.
- in **contract signing**, each player exchanges a non-repudiable commitment to the contract text in return for the other player's non-repudiable commitment to the same contract text,
- In **certified mail**, a message is exchanged for an acknowledgment of receipt.
- In **barter**, an arbitrary item of value is exchanged for another item of value.

An important security requirement on exchanges is *fairness*. An exchange is fair if at the end of the exchange, either each player receives the item it expects or neither player receives any additional information about the other's item. In electronic commerce scenarios, exchanges have to be carried out over insecure networks. An attacker can gain control of the network or corrupt the systems used by the other player. Thus, carefully designed exchange protocols are necessary to guarantee fairness in electronic commerce transactions. Typically these protocols must also possess additional properties. For example, there may be subsequent disputes about what was exchanged during the transaction even if the exchange itself was completed fairly. In this case, sufficient evidence must be accumulated during the exchange to support the resolution of any future disputes.

Various solutions to the fair exchange problem have been reported in the literature. Early solutions fall into two categories: *third party* protocols [6, 14, 7, 8] which make use of a trusted, on-line third party, and *gradual exchange* protocols [4] where the probability of correctness is gradually increased over several rounds of communication.

Both types of solutions had shortcomings: the third party solutions ran the risk of the third party becoming a bottleneck, whereas the gradual exchange solutions were impractical due to the high communication overhead. A third category of solutions involved the use of third party only under exceptional circumstances. The basic idea is as follows. First, both players agree on what is to be exchanged, and which third party to use in case of an exception. This "agreement" is informal — it has no validity outside the context of the protocol. Then, one player (the originator) takes the risk of sending its item first, hoping that the other player will respond with its item. If the other player responds as expected, the protocol ends successfully. Otherwise, the originator contacts the third party to resolve the fair exchange. This approach is known as the *optimistic approach* [1]. It also relies on the use of a third party, but only in the case of an exception.

Optimistic protocols exchanging money for a receipt or goods was first outlined in [5]. Recently, detailed optimistic fair exchange protocols were presented for generic items [1], and certified mail [12]. All of these protocols rely on certain strong assumptions to guarantee termination. Only the risk-taking player (originator) is allowed to invoke the third party. The *other* player (responder) cannot know the final state of the exchange unless and until the originator invokes the third party.

The use of an overall time limit parameter was proposed in [1] as a solution to this problem. The third party will not resolve an exchange after the specified time limit. However, this implicitly forces the additional assumptions that the messages sent to or from the third party are guaranteed to be delivered within a finite, sufficiently small, time, and that the systems of all the players are guaranteed to be continuously available until the overall time limit. These assumptions are not completely satisfactory because in practice, it is difficult to choose the right value for the time limit which makes them reasonable. If the time limit is too short, the originator may not be quick enough to start the recovery on time. Failure to start the recovery within the time limit will destroy the fairness for the originator, even if it is honest. Similarly, if the time limit is too long, the responder may have to wait without knowing what happened. Worse still, if the responder happens to crash during this time, it may not be able to prove later that it behaved correctly otherwise.

In this paper, we present optimistic protocols which, in addition to being efficient and secure, guarantee a *timely* conclusion to a protocol run: at any time during a protocol run, either player can *unilaterally* choose to force an end to the protocol without losing fairness.

In Section 2, we first define fair exchange and the properties required of it. Then we outline our communication model and the cryptographic primitives used in our protocols. In Section 3, we outline our approach by describing a protocol contract signing. In Section 4, we address the issue of generalizing the approach for exchanging arbitrary items and discuss the relationship between the nature of the items and the degree of fairness provided by the optimistic protocol. In Section 5, we briefly describe some optimized instantiations of the generic protocol.

P		Q
Input : i_P, d_Q, Q	fair exchange $\longleftrightarrow$	**Input** : i_Q, d_P, P
Output :		**Output** :
i_Q (desc(i_Q) = d_Q)	**or**	i_P (desc(i_P) = d_P)
aborted		*aborted*

Figure 1: A Successful Fair Exchange

2 Preliminaries

2.1 Requirements for Fair Exchange

We assume that there exists a function desc() that maps any exchangeable item to a string describing it in "sufficient" detail. (For example, a signature may be described by the message and the public key of the signer; a payment may be described by the payee, value, and currency.) Figure 2.1 shows the inputs and outputs of each player during a successful fair exchange. P begins the exchange with an item i_P and a description d_Q of the expected item. A player P can consider a protocol run *completed*, when it is safe for P to terminate (and take no further part in the protocol). A protocol should indicate the points where it is completed.

We formulate the following requirements in terms of one player, P — i.e., assuming P behaves correctly. The requirements of the other player, Q, can be stated similarly. Any failures in the communication channel are subsumed in the notion of a misbehaving peer.

- **R1 - Effectiveness**: If Q also behaves correctly, and both P and Q do not want to abandon the exchange, then when the protocol has completed, P has i_Q such that desc(i_Q) = d_Q.
- **R2 - Fairness**: Two notions of fairness are possible,
  - **R2a - Strong Fairness**: When the protocol has completed, either P has i_Q such that desc(i_Q) = d_Q, or Q has gained no additional information about i_P, or
  - **R2b - Weak Fairness**: When the protocol has completed,
    * either P has i_Q such that desc(i_Q) = d_Q, or Q has gained no additional information about i_P,
    * or P can prove to an arbiter that Q has received (or *can* still receive) i_P such that desc(i_P) = d_P, without any further intervention from P.
- **R3 - Timeliness**: At the beginning of the exchange, P can be sure that the protocol will be completed at a finite point of time. At completion, the state of the exchange as of that point is either final or any changes to the state will not degrade the level of fairness achieved by P so far[1].

[1]For example, this implies that if P has already achieved

- **R4 - Non-repudiability**: After an effective exchange (i.e., P has received i_Q at the end of the exchange), P will be able to prove
  - **R4a - Non-repudiability of Origin**: the origin of i_Q (e.g., that it originated from Q), and
  - **R4b - Non-repudiability of Receipt**: that Q received i_P.

  Note that, in general, non-repudiability is not a primary requirement for fair exchange protocols. However, it is included here because it is useful in subsequent disputes after a fair exchange (even a successful one).

Some fair exchange protocols may also satisfy other desirable security properties such as anonymity. On the other hand, depending on the context, non-repudiability may not be required.

In optimistic fair exchange, one of the two players sends its item first. In order to make this distinction clear, we will identify the players as O (the "Originator") and R (the "Recipient"). Throughout this paper, whenever we use the names P and Q, we imply that it is not necessary to distinguish between the players. When it is necessary, we will use names O and R in order to clearly indicate the respective roles. A third player T, known to both O and R, will be used as a third party to be used in recovery protocols.

As usual, ordinary players are not required to trust each other. But all players are required to trust the third party to a certain extent. Naturally, it is desirable to minimize this trust as much as possible. Making the third party *verifiable* (that is, if it misbehaves, its victim(s) can prove the fact in a dispute) is one way of reducing this trust. We introduce a new requirement which captures this property:

- **R5 - Verifiability of Third Party**: Assuming the third party T can be forced to eventually send a valid reply to every request, the *verifiability of 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.

2.2 Communication Model

We assume that communication channels between any two players are *confidential*, meaning that eavesdroppers will not be able to determine the contents of messages traveling through these channels. Confidential channels can be implemented by encrypting messages for the recipient. We define two levels of quality for a communication channel.

Definition 2.1 *A communication channel between two correctly behaving players is operational if the messages inserted into it by the sender are received by the recipient within a known, constant (or constant factor) time interval.*

A communication channel between two correctly behaving players is ***reliable*** *if it is guaranteed to be always operational. An attacker will not be able to delay any message in a reliable channel.*

A communication channel between two correctly behaving players is ***resilient*** *if it is normally operational but an attacker can succeed in delaying messages by an arbitrary, but finite amount of time. In other words, a message inserted into a resilient channel will eventually be delivered.*

We will normally make the weaker assumption that the channel between any player and T is resilient. The resilient channel assumption leads to an *asynchronous* communication model: we can make no timing assumptions such as bounds on message delays, or deviations between local clocks.

Normally, we make no assumptions about the communication channel between O and R. The attacker may gain complete (and permanent) control it. If either O or R is dishonest or wants to abandon the fair exchange, it may choose not to respond to any message, regardless of the quality of the communication channel used to send the message. Any failures in the communication channel are subsumed in the notion of a misbehaving peer.

2.3 Cryptographic Tools

We assume that each player has the ability to compute and verify digital signatures using some arbitrary digital signature scheme

- in which each player P has a signing key S_P, and a corresponding verification key V_P,
- which has a signature algorithm sign such that, given a message m, $\mathsf{sign}(m, S_P) = Sig_P(m)$, where $Sig_P(m)$ is said to be a signature on message m with key S_P.
- which has a signature verification algorithm verifySig such that, given a message m and a claimed signature s on m, $\mathsf{verifySig}(m, s, V_P)$ evaluates to true if and only if there exists a $Sig_P(m)$ which is equal to s.

strong fairness, any subsequent change in state should not cause P to lose its fairness or even degrade it to weak fairness.

The verification keys can be potentially anonymous. We also assume that anyone who has $Sig_P(m)$ also has m. This is easily done by always appending m to the signature on m. Further, we assume that there exists a collision-resistant one-way hash function $h()$. The notations used for various functions and objects are summarized in Table 1 (some of the concepts are introduced later in the text). We refer readers interested in more details on cryptographic concepts like digital signatures and one-way hash functions to [11].

Notation	Explanation
V_P,S_P	Verification and signing keys of P
$Sig_P(m)$	Message m signed with S_P
verifySig()	Signature verification algorithm
$h()$	Collision-resistant one-way hash function
$Enc_P(m)$	Encryption of message m recoverable only by P

Table 1: Notation Summary

3 An Example: Contract Signing

First, we illustrate our approach by considering a two-party contract signing scenario. In two-party contract signing, both players have initially agreed on some contract text. A valid contract consists of non-repudiation tokens on the contract text by each player. A fair contract signing protocol must ensure that either both players end up with valid contracts or neither does.

3.1 Protocol Description

O and R want to sign a contract so that they both expect to end up with each other's non-repudiation token on a previously agreed contractual text. Our optimistic contract signing protocol has three subprotocols: exchange, abort, and resolve. In the normal case, only protocol exchange is executed. The other two are used only if one of the two players decides to forcibly complete (presumably because it has decided that something has gone wrong) the protocol. This is an indeterministic choice made locally by a player. Note that, since our model is asynchronous, a player deciding something has gone wrong does not necessarily imply that any other player has behaved incorrectly. The protocol is shown in Figure 2. Places where a player can decide to give up and forcibly complete the protocol are marked with the tag "**give_up**?". For example, "**give_up**? *abort*" means "if the player decides to give up, it should run protocol *abort*."

Step 1. We assume that O and R have already agreed on the contract text and run protocol *exchange* with the contract text as input. O generates a random o_O and computes $com_O = h(o_O)$. com_O will be used as a public commitment to the secret, o_O, meaning that once com_O is given to R, O cannot change the secret o_O. Note that $h()$ need not be a secure string commitment scheme (see Section 4.2) — $h(x)$ can leak information about x as long as it is computationally infeasible to determine x, given $h(x)$. O generates message m_1 as shown and signs it to produce message me_1;[2] me_1 is sent to R.

Step 2. If R decides to give up, it simply terminates the protocol run. In practice, R will decide this if it does not receive me_1 within a reasonable time. Note that we do not require globally synchronized clocks: it is entirely up to R to decide what is a "reasonable time" and when to time out.

If m_1 is not formed correctly, or if verifySig(m_1, me_1, V_O) evaluates to false, R ignores the message. In practice, R may send a negative acknowledgment to O and continue to wait for a valid message until timing out. In the rest of this description, we do not explicitly mention that malformed or incorrect messages are ignored.

Otherwise, R generates a random number r_O and computes $com_R = h(o_R)$. R generates message m_2 as shown and signs it to produce me_2 which is then sent to O.

Step 3. If O decides to give up, it invokes T by running protocol *abort*. In practice, O decides to give up if it does not receive me_2 within a reasonable time.

Otherwise it sends o_O to R.

Step 4. If R decides to give up, it invokes T by running protocol *resolve*. Typically, R gives up if it does not receive an x in time, such that $h(x) = com_O$.

Otherwise it sends o_R to O.

Step 5. If O decides to give up, it invokes T by running protocol *resolve*. Typically, R gives up if it does not receive a y in time, such that $h(y) = com_R$,

[2] The messages are labeled so that the second letter of the label identifies the protocol, and the subscript identifies the message number. For example, me_1 is the first message of protocol *exchange*, and mr_2 is the second message of protocol *resolve*.

Protocol *abort* is used by O to abort the protocol so that T will not resolve the protocol at a later time. Protocol *resolve* is used by either O or R to force a successful termination. Clearly *only one* of *abort* or *resolve* can succeed for a given instance of *exchange*. On T's system each of protocol *resolve* and protocol *abort* is guaranteed to be atomic.

In protocol *abort*, O sends a signed abort request to T. If the corresponding exchange has not already been resolved, T issues an *abort_token*. The *abort_token* is a guarantee by T that it has not and will not resolve this particular instance of *exchange*. If the protocol has already been resolved, T will simply forward the result of the protocol *resolve*.

In protocol *resolve*, either player sends me_1 and me_2 to T. If the exchange has already been aborted, T simply returns the *abort_token*. Otherwise, it issues a *replacement_contract* by counter signing me_1 and me_2 (i.e., $Sig_T(me_1, me_2)$).

Definition 3.1 *A valid contract is of the form*

1. $\{me_1, o_O, me_2, o_R\}$, *or*
2. $Sig_T(me_1, me_2)$

where,
$me_1 = Sig_O(V_O, V_R, \mathsf{T}, text, com_O)$
$me_2 = Sig_R(me_1, com_R)$
$com_O = h(o_O), com_R = h(o_R)$

Note that, even though this definition dictates the structure of contracts, it does not dictate any restriction on the signatures used to construct valid contracts. Any standard signature scheme, as long as it satisfies the requirement in Section 2.3, can be used.

3.2 Analysis

The "items" expected by both parties are the contract authenticators: o_O and o_R respectively. The descriptions of the items are com_O and com_R, respectively, along with the contract text itself.

Now we list the claims about the protocol with respect to the requirements with respect to the requirements identified in Section 2.1. Proofs can be found in the full version of the paper [3].

Claim 1 *Assuming that the communication channel between* O *and* R *is resilient, the protocol satisfies the effectiveness requirement (**R1**).*

Claim 2 *Assuming that the communication channel between* T *and any other player is resilient, the optimistic contract signing protocol satisfies requirements **R2a**, **R3**, **R4**, and **R5** for both* O *and* R.

3.3 Invisibility of Third Party

In [12], Micali has described the notion of an *invisible* third party. According to Definition 3.1, there are two forms of a valid contract. The second form makes it obvious that T was involved in protocol *resolve*. T is said to be *visible* in such a protocol. If the third party were invisible, the end result (i.e., the form of a valid contract) will be the same regardless of whether T was invoked or not. We can easily change our protocol so that T is invisible to R: in the first step of the protocol, we can replace the one-way hash function $h()$ with the one-way trapdoor function $Enc_T()$, where the public encryption key of T is used to encrypt the random number o_O along with V_O and V_R. A valid contract for R is: $\{me_1, o_O, me_2, o_R\}$, where

$me_1 = Sig_O(V_O, V_R, \mathsf{T}, text, com_O)$
$me_2 = Sig_R(me_1, com_R)$
$com_O = Enc_T(o_O|V_O|V_R), com_R = h(o_R)$

During protocol *resolve* for R, T decrypts com_O with its private decryption key to recover the random authenticator. This version of the protocol satisfies the same set of requirements as the original version except for **R5**. The proof for the verifiablity of T [3] depends on its being visible. However, the real intent behind invisibility should be *non-invasiveness* — our contract-signing protocol is invasive in the sense that we define the structure of a valid contract. A truly non-invasive fair exchange protocol should be able to exchange *arbitrary* items such as tokens from electronic payment protocols (coins, checks etc.), credentials, and contracts, without making any demands on its structure. In Section 4, we study the problem of fair exchange of arbitrary items. Typically, non-invasiveness implies invisibility of third party. In [2], we describe techniques to enable non-invasive fair exchange of digital signatures.

4 Generic Fair Exchange

Clearly, generic fair exchange mechanisms capable of exchanging arbitrary items is quite useful. In this section, we investigate how feasible it is to develop protocols for generic fair exchange and in particular what difficulties arise in using the optimistic approach for such protocols. We will attempt to generalize our protocol from Section 3 so that it is similar in functionality to the generic protocols of [1].

4.1 Properties of Exchangeable Items

In the simplest case, each item in an exchange can be represented as one or more strings. For example, parts of a valid contract in Section 3 was represented as strings. To transfer such an item, it is enough to simply send the string to the recipient. Consequently, exchange protocol

O		R
Input : $V_R, \mathsf{T}, text$		**Input** : $V_O, \mathsf{T}, text$
	$\xrightarrow{me_1 = Sig_O(\overbrace{V_O, V_R, \mathsf{T}, text, com_O}^{m1})}$	**give_up**? : **quit**
give_up? : *abort*	$\xleftarrow{me_2 = Sig_R(\overbrace{me_1, com_R}^{m_2})}$	
	$\xrightarrow{me_3 = o_O}$	**give_up**? : *resolve*
give_up? : *resolve*	$\xleftarrow{me_4 = o_R}$	

abort protocol

O		T
	$\xrightarrow{ma_1 = Sig_O(aborted, me_1)}$	**if** resolved **then** $ma_2 = Sig_T(me_1, me_2)$ **else** aborted = true $ma_2 = Sig_T(aborted, ma_1)$
	$\xleftarrow{ma_2}$	

resolve protocol

O, R		T
	$\xrightarrow{mr_1 = (me_1, me_2)}$	**if** aborted **then** $mr_2 = Sig_T(aborted, ma_1)$ **else** resolved = true $mr_2 = Sig_T(me_1, me_2)$
	$\xleftarrow{mr_2}$	
Output : *contract* (Def. 3.1)		**Output** : *contract* (Def. 3.1)
	or	
abort_token		*abort_token*

Figure 2: Optimistic Protocol for Contract Signing

an exchange of items is essentially an exchange of secret strings. However, in general, items are transferred using specific transfer protocols. For example, a transfer of value may involve a payment protocol. A transfer of a digital good from a seller to a buyer may involve an interactive fingerprinting protocol [13]. In theory, one may still be able to identify the communication steps which transfer critical secret strings in an arbitrary transfer protocol, and separate them. Alternatively, it may be possible to reduce the transfer of the item to the transfer of a specific secret string [2]. In these cases, we can still model any exchange as an exchange of secret strings. In practice, this is difficult because it will involve changing existing applications. Even if the transfer protocols support a generic token-based interface [9], they have to be modified so that they indicate which tokens contain critical strings.

We will assume that the transfer of items require specific transfer protocols over which we have no control. In Section 4.2, we describe an asynchronous protocol for the optimistic fair exchange of a class of items which we call *forwardable* items. In the general case, this protocol can only guarantee weak fairness. However, for certain classes of items, it is possible to guarantee strong fairness. We return to this issue in Section 4.4. Whether it is possible to generalize further to design optimistic fair exchange protocols for arbitrary items is an open question.

4.2 Exchange of Forwardable Items

We assume the same requirements, communication model, and cryptographic tools as in Section 2. In addition, we introduce the following.

First, we assume that P can send the item directly to Q, or it can send it to T; T will be able to verify the correctness of the item (with respect to the stated description) and either store it or send it to Q. We also assume that the transfer protocols for these items are *idempotent*: meaning that re-sending the item by repeating the transfer protocol is harmless. If an item satisfies both properties, we call it a *forwardable item.* Simple strings are forwardable items.

Secondly, we assume the existence of a string commitment scheme with some special properties. A suitable string commitment scheme should consist of a function commit() which, given a string m, and a random key key, generates a commitment com, $\mathsf{commit}(m, key) = com$, and a function verifyCom() such that, given a string m, a commitment com, and a key key,

$$\begin{aligned} & \mathsf{verifyCom}(com, key, m) = \mathsf{true} \\ \Longleftrightarrow \quad & \mathsf{commit}(m, key) = com \end{aligned}$$

The idea is that once a commitment com of a string m is made, nobody can change the content m without invalidating com. Further, given com, nobody can get any information of m. We will use a commitment of an item and its key as a non-repudiation of origin token for that item. In the protocol description below, when we have to compute commitments on items that are not plain strings, we will use an empty string instead — we will still be able to retain the non-repudiability property, as explained in the proof of claim 6[3].

The optimistic protocol for the fair exchange of forwardable items is shown in Figure 3. Initially, O and R agree on the descriptions of the items they expect from each other: d_R and d_O describe the items expected by O and R respectively. Each player then runs protocol *exchange* with its own item and the description of the item it expects to receive.

Step 1. O generates a random number key_O from the domain of commit() and a random number r_O. Then it computes $com_O = \mathsf{commit}(i_O, key_O)$, $h_O = h(r_O)$, and $d_O = \mathsf{desc}(i_O)$; collects these pieces of data into a message m_1 as shown in Figure 3, and signs it to produce message me_1; me_1 is sent to R. As we noted earlier, if i_O is not a simple string, we compute com_O as a commitment of key_O only. In this case, key_O serves as a non-repudiation of origin token for any item described by d_O. If d_O is capable of uniquely identifying i_O, the two ways of computing com_O are equivalent. The same applies to the computation of com_R below.

Step 2. If R decides to give up, it simply terminates the protocol run. Typically, R gives up if it does not receive a valid me_1 within a reasonable time limit. All invalid messages are ignored. A valid message is such that last but one component of m_1 is the same as $\mathsf{desc}(i_R)$, the last component of m_1 is d_O, and $\mathsf{verifySig}(m_1, me_1, V_O)$ evaluates to true.

Otherwise, R generates a random number key_R from the domain of commit() and a random number r_R. It then computes $com_R = \mathsf{commit}(i_R, key_R)$, $h_R = h(r_R)$ and then generates message m_2 as shown and signs it to produce message me_2; me_2 is sent to O.

Step 3. If O decides to give up (e.g., if it does not receive a valid me_2 within a reasonable time) it runs protocol *abort*. A valid me_2 is such that $\mathsf{verifySig}(m_2, me_2, V_R)$ evaluates to true.

Otherwise it sends i_O and key_O to R in message me_3. Note that this operation consists of two parallel sub-protocols: a run of the transfer protocol of i_O and

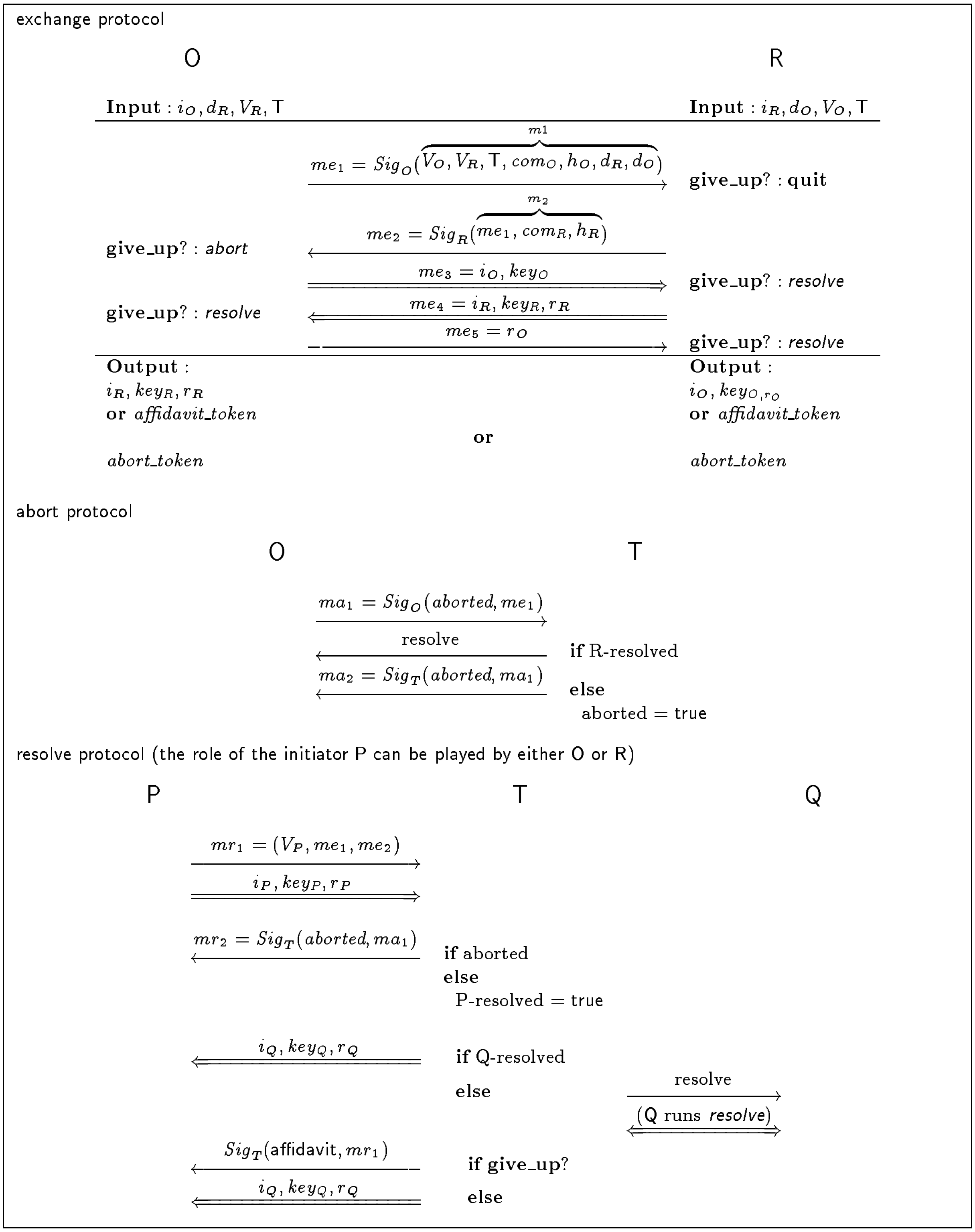

Figure 3: Optimistic Fair Exchange of Forwardable Items (double arrows indicate a transfer protocol)

a simple message transfer for sending key_O, r_O. For simplicity, we indicate these as a single "transfer protocol" in Figure 3. The same applies to message me_4 below.

Step 4. If R decides to give up (e.g., if it does not receive a valid me_3 in time), it runs protocol *resolve*. A valid me_3 is such that $d_O = \mathsf{desc}(i_O)$, $\mathsf{verifyCom}(com_O, key_O, i_O)$ evaluates to true, and $h(r_R) = h_R$. Again, note that *all* the sub-protocols mentioned above must succeed for R to conclude that it has received a valid me_3. The same applies to message me_4 below.

Otherwise it sends i_R, key_R, r_R to O in message me_4.

Step 5. If O decides to give up (e.g., if it does not receive a valid me_4 in time), it runs protocol *resolve*. A valid me_4 is such that $d_R = \mathsf{desc}(i_R)$, and $\mathsf{verifyCom}(com_R, key_R, i_R)$ evaluates to true.

Otherwise it sends r_O to R in message me_5.

Step 6. If R decides to give up (e.g., if it does not receive a valid me_5 (such that $h(r_O) = h_O$) in time), it runs protocol *resolve*.

If the protocol completes without any exception, each player P will end up with:

- the item i_Q from the other player, Q,
- its non-repudiation of origin (NRO) token consisting of me_1, me_2, com_Q, key_Q, and
- a non-repudiation of receipt (NRR) token for i_P consisting of me_1, me_2, h_Q, r_Q.

Only O is allowed to initiate protocol *abort*. On receiving a request to abort a particular instance of an exchange, T will first check to see if R has already resolved. If it has, T will tell O to run protocol *resolve* instead. Otherwise, T will mark the exchange as aborted and issue an *abort_token* to O.

Protocol *resolve* proceeds as follows. Either O or R can run *resolve*. Let us denote the invoker by P and the other player by Q.

Step 1. First, the signed message me_1 and me_2 are sent to T along with the invoker's verification key V_P. Then P sends its item to T using the item using the transfer protocol, as well as all the non-repudiation tokens.

Step 2. If the protocol instance has been aborted before, T replies with the *abort_token*.

Step 3. Otherwise, T will mark the protocol as resolved by P and adds i_P, key_P, and r_P to its stable storage.

Step 4. If the exchange is already marked as resolved by Q, T will extract i_Q, key_Q, and r_Q from its stable storage and forward them to P using the item transfer protocol.

Step 5. Otherwise, T will send a message to Q asking it to run protocol *resolve*.

Step 6. If Q runs *resolve* successfully, i_Q, key_Q, and r_Q will be added to T's storage.

Step 7. If T finds that Q has resolved the exchange, it can forward i_Q, key_Q, and r_Q to P. If T decides to give up and force a termination (e.g., when Q does not complete protocol *resolve* within a reasonable time), it will issue an *affidavit_token* to P. P may choose to terminate at this point, in which case it has achieved weak fairness, or to re-run protocol *resolve* later.

The *affidavit_token* given to P is a statement by T that if Q wants, it can achieve strong fairness at any time after the issue of *affidavit_token*. As before, we assume that T will execute each of protocol *resolve* and protocol *abort* in a mutually exclusive manner. The idea behind weak fairness is that the player who ends up with only an affidavit can initiate a dispute outside the fair exchange system and use the affidavit as evidence.

Note that the interaction with T and Q (steps 5 and 6) is not necessary as far as meeting the weak fairness requirement is concerned. It is included as a practical measure only.

4.3 Analysis

4.3.1 Security of the Protocol

The items expected by P from Q are the actual item itself (i_Q), its non-repudiation of origin token (key_Q), and the non-repudiation of receipt token for i_P (r_Q).

Below, we list our claims about the security of the protocol with respect to the requirements identified in Section 2.1. Proofs can be found in the full version of the paper [3].

Claim 3 *Assuming that the communication channel between* O *and* R *is resilient, the protocol satisfies the effectiveness requirement (**R1**).*

Claim 4 *Assuming that the channel between* T *and any other player is resilient, and that* T *behaves correctly, the protocol satisfies the weak fairness requirement (**R2b**).*

Claim 5 *Assuming that the channel between* T *and any other player is resilient, and that the underlying transfer protocols are guaranteed to complete within a finite time once they are begun, if the fair exchange protocol meets the requirement for weak fairness (**R2b**) it also meets the timeliness requirement (**R3**).*

Claim 6 *The protocol meets non-repudiation requirements (**R4a** and **R4b**).*

4.3.2 Remarks on Channel Quality

Throughout, we have assumed that the channel between T and each other player is resilient. It is reasonable to assume that semi-open networks such as corporate intra-nets or the telephone network are reliable. The resilience assumption is reasonable even on open networks such as the Internet. If an attacker did manage to disrupt a communication channel, it will be eventually detected and repaired. Also, protection against all benign network failures are covered under the resilient network assumption. We also assumed that an attacker can merely delay messages inserted into a resilient channel, but cannot remove them. In practice, this is achieved by using a datalink-layer protocol that takes care of message retransmissions.

Of course the resilience assumption does not place a bound on the delay before the channel becomes operational again. Therefore, the optimistic approach is inappropriate to exchange time-sensitive items (e.g., current stock quotes). Fair exchange of time-sensitive items over open networks is a difficult problem, regardless of the technique. Even if we used an online third party (instead of the optimistic approach), the protocol will not be secure against an attacker who can disrupt communication channels long enough. In any case, the optimistic approach is intended to be used in conjunction with a suite of other approaches such as those that use an on-line third party.

We also assumed that all communication channels are confidential. This is stronger than what we actually need in order to guarantee fairness. The fairness of the contract signing protocol does not require confidentiality for any of its message flows. In the exchange protocol for forwardable items, R must send message mr_1 confidentially, for otherwise, O can intercept i_R and then abort the exchange. Of course, if confidentiality of message contents against eavesdroppers is a requirement, then all messages need to be sent via a confidential channel.

4.4 Weak vs. Strong Fairness

Weak fairness is useful only if *both* O and R can be made to participate in the dispute resolution. While weak fairness is clearly not appropriate in all circumstances, there are situations where it is sufficient. With an anonymous individual peer, it may be necessary to use an exchange protocol that guarantees strong fairness. With a large, reputable, organization which is unlikely to "take your money and run," weak fairness may be sufficient.

If the items to be exchanged are such that T can either *generate* a replacement item or *revoke* an already released item, then optimistic fair exchange protocols can in fact guarantee strong fairness [1]. Contracts, receipts for certified mail etc. are examples of generatable items. When we use generatable or revocable items in an exchange, protocol *resolve* is changed so that instead of sending an affidavit, T will provide a replacement (in case of the generatable items) or have the item revoked (in case of revocable items).

Notice that if R is willing to forego the timeliness requirement (i.e., if it is willing to wait until it receives flow me_3), it can actually achieve strong fairness. Thus, by suitable assignment of roles, the generic protocol can be made to guarantee strong fairness. If the item of one player is generatable, then it should play the role of the respondent(R). Similarly, if the item of one player is revocable, then it should play the role of the originator(O). In passing, we also note that if one of the two players wishes to be anonymous, then it should play the role of the originator. Interestingly, it is also possible to transform a large class of items into generatable items. Techniques for such transformations and efficient protocols for the exchange of generatable items are discussed in [2].

5 Example Instantiations

We can instantiate the generic protocol of Figure 3 to yield concrete protocols for specific types of fair exchanges. In this section, we briefly outline two such instantiations: for certified mail, and payment for receipt. Note that the generic protocol has provision for non-repudiation tokens built into it. In both instantiations described below, we will treat the non-repudiation tokens as the item expected by the originator. In the generic protocol, an item i_P is always sent along with its NRO token, key_P. Therefore, by treating key_P as i_P, the fairness properties are not affected.

5.1 Certified Mail

In certified mail, a sender wants to send a mail message m to a receiver. The sender requires that the receiver not be able to deny receiving the message.

To achieve this, the sender needs a non-repudiation of receipt token from the receiver in exchange for the message. Thus certified mail is a fair exchange of the message and its non-repudiation of receipt token.

The mail message is neither generatable nor revocable. But it is forwardable. As in the contract signing example, we define a valid receipt for the message as either the tuple, $\{me_1,\ me_2,\ key_R\}$ or an *affidavit_token* consisting of T's signature on me_1, me_2. Thus, the mail sender takes the role of the originator and the receiver takes the role of the responder. The generic protocol can be instantiated as shown in Table 2. The receipt is, by definition, a generatable item.

Generic Protocol	Instantiation
i_O	mail
i_R	receipt text
d_O	not needed
d_R	receipt text
com_O	commit($mail$, key_O)
com_R	h(key_R)
resolve for O	if unsuccessful, T issues replacement receipt
resolve for R	not needed[a]

Table 2: Instantiating the Generic Protocol for Certified Mail

[a]i.e., use the same protocol as in Figure 3.

Generic Protocol	Instantiation
i_O	mail
i_R	receipt text
d_O	not needed
d_R	receipt text
com_O	$Enc_T(\mathrm{mail}\vert key_O\vert V_O\vert V_R)$
com_R	$h(key_R)$
resolve for O	T issues replacement receipt
resolve for R	if com_O is correct, T recovers $mail$, key_O in return for key_R

Table 3: Instantiating the Generic Protocol for Certified Mail

With resilient channels, this instantiation makes T verifiable, and guarantees strong fairness and timeliness for O. R is guaranteed either strong fairness without timeliness or weak fairness with timeliness.

A slightly different instantiation, shown in Table 3 guarantees strong fairness and timeliness to both players. A valid receipt for the mail message is either $\{me_1,\ me_2,\ key_R\}$ or an affidavit from T . This instantiation is illustrated in Figure 4. The figure depicts the result of a direct instantiation according to Table 3, followed by the removal of unnecessary flows and fields. For example, since we do not make use of the non-repudiation of receipt tokens (the r_P values), the exchange protocol has only four flows. Protocol *abort* is the same as in Figure 3.

5.2 Payment for Receipt

In payment for receipt, a payer wants to make a payment and get a receipt in return. If the payment messages are forwardable (e.g., an electronic check), a simple instantiation is to allow T to issue a replacement receipt when payer O runs protocol *resolve*. This would guarantee timeliness for both players, strong fairness for payee R but only weak fairness for O.

Suppose that the payment mechanism used supports revocation. Table 4 shows an instantiation that guarantees strong fairness and timeliness for the payer. The payee has a choice of either timeliness combined with weak fairness or strong fairness without timeliness. The protocol is illustrated in Figure 5. If O does not receive a payment-receipt after sending the payment, it can invoke T to revoke the payment. If R does not receive an NRR token after sending a payment-receipt, it can invoke T to receive a replacement NRR token. With a resilient channel between R and T, a misbehaving O may succeed in revoking a payment after a successful exchange. But R can prove that the O misbehaved (in choosing to run protocol *resolve*) by producing the original NRR token for the payment-receipt itself (i.e., r_O). If O presents the payment receipt issued by R to a verifier, the verifier must always check with R or T to ensure that the payment has not been subsequently revoked.

6 Conclusion

The asynchronous protocol for contract signing described here is the first protocol to provide timeliness guarantee while assuming only resilient communication channels. We have generalized the basic idea to design a generic protocol for the fair exchange of forwardable items. We used the notions of strong and weak fairness [1] and described how they relate to the the notions of generatable and revocable items. In a companion paper [2], we show how we can add the generatability property to a large class of items, using a technique known as *verifiable encryption*.

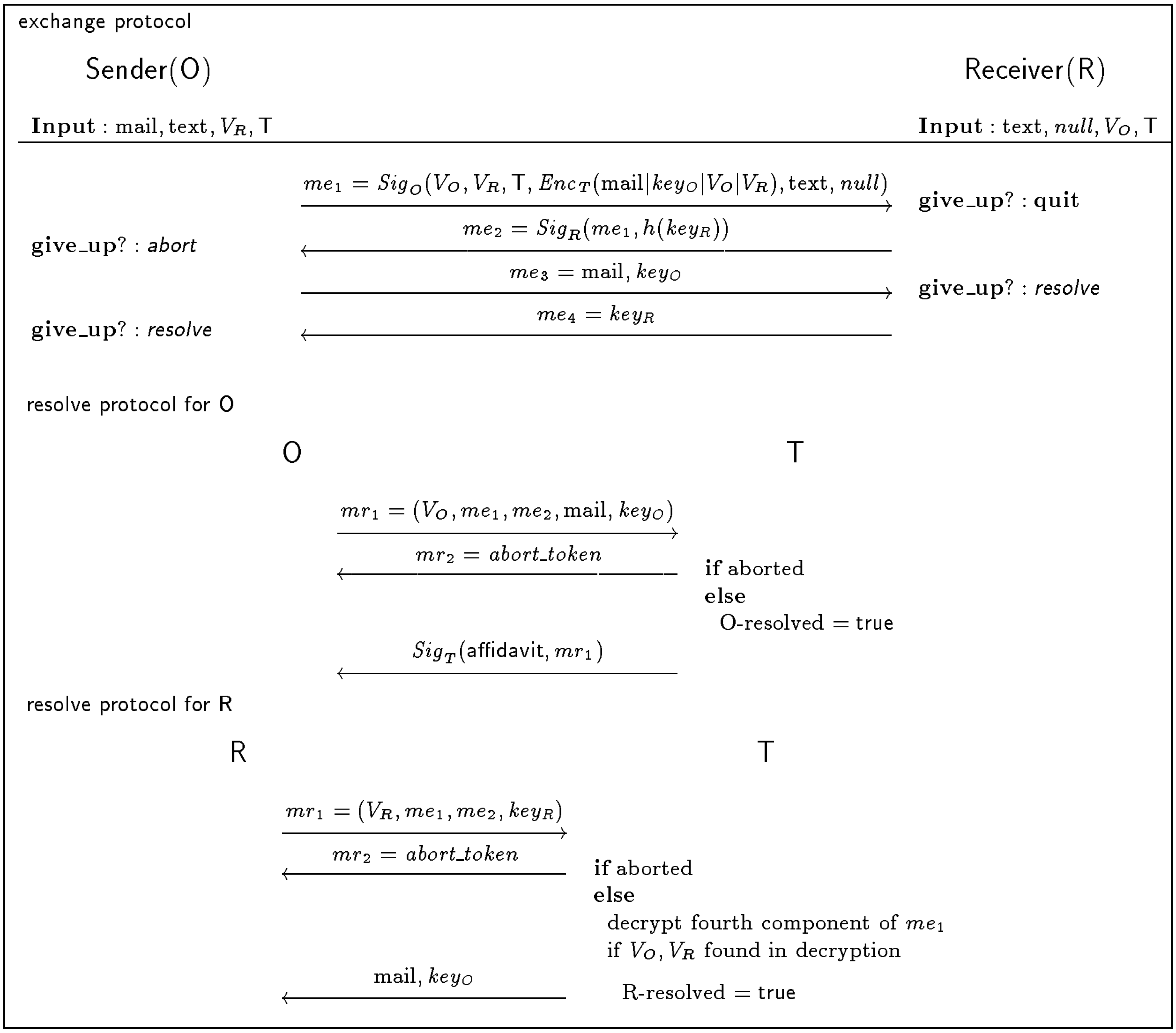

Figure 4: Certified Mail Protocol

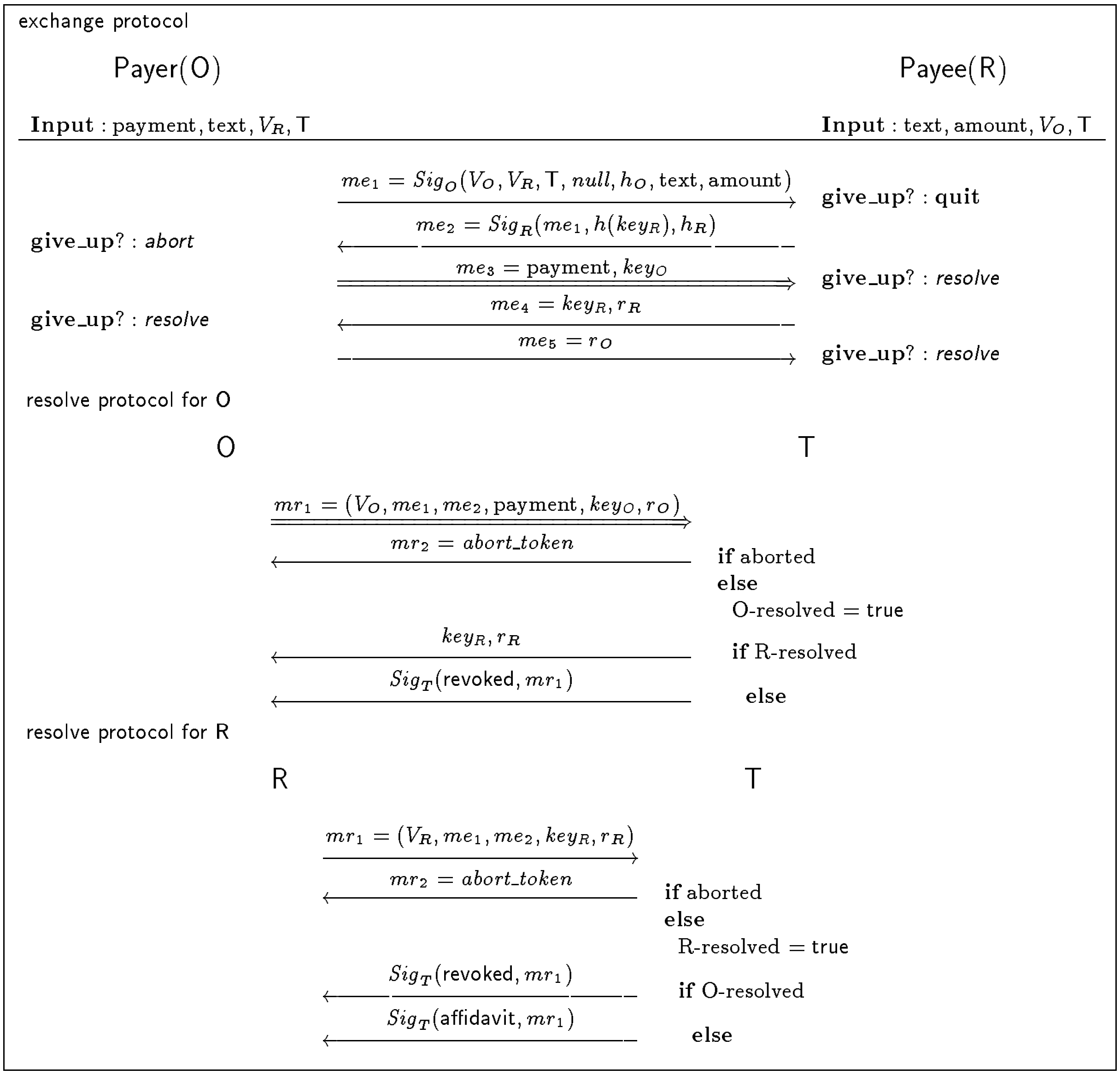

Figure 5: Payment for Receipt

Generic Protocol	Instantiation
i_O	payment
i_R	receipt text
d_O	amount
d_R	receipt text
com_O	not needed
com_R	$h(key_R)$
resolve for O	if unsuccessful, T revokes payment
resolve for R	if unsuccessful, T issues replacement NRR token for payment-receipt

Table 4: Instantiating the Generic Protocol for Payment-for-receipt

7 Acknowledgments

Jay Black, Ceki Gülcü, Matthias Schunter, Michael Steiner, and Gene Tsudik reviewed previous versions of this paper. We thank them for their valuable comments which have greatly improved the quality of this work. We also thank Dave Presotto for his guidance in preparing the final version.

References

[1] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In Matsumoto [10], pages 6, 8–17.

[2] N. Asokan and Victor Shoup. Optimistic fair exchange of digital signatures (to appear). In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT '98*, number to be assigned in Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 1998.

[3] N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. Research Report RZ 2976 (#93022), IBM Research, November 1997.

[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, January 1990.

[5] Holger Bürk and Andreas Pfitzmann. Value exchange systems enabling security and unobservability. *Computers & Security*, 9(8):715–721, 1990.

[6] Benjamin Cox, J. D. Tygar, and Marvin Sirbu. NetBill security and transaction protocol. In *First USENIX Workshop on Electronic Commerce*, New York, July 1995. USENIX.

[7] Robert H. Deng, Li Gong, Aurel A. Lazar, and Weiguo Wang. Practical protocols for certified electronic mail. *Journal of Network and System Management*, 4(3), 1996.

[8] Matthew K. Franklin and Michael K. Reiter. Fair exchange with a semi-trusted third party. In Matsumoto [10], pages 1–5,7.

[9] J. Linn. Generic security service application program interface, version 2. Internet Network Working Group, Standards Track, Request for Comments: RFC 2078, January 1997. Obsoletes RFC 1508.

[10] Tsutomu Matsumoto, editor. *4th ACM Conference on Computer and Communications Security*, Zurich, Switzerland, April 1997. ACM Press.

[11] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography.* CRC Press series on discrete mathematics and its applications. CRC Press, 1996. ISBN 0-8493-8523-7.

[12] Silvio Micali. Certified e-mail with invisible post offices. Available from author; an invited presentation at the RSA '97 conference, 1997.

[13] Birgit Pfitzmann and Matthias Schunter. Asymmetric fingerprinting (extended abstract). In Ueli Maurer, editor, *Advances in Cryptology – EUROCRYPT '96*, number 1070 in Lecture Notes in Computer Science, pages 84–95. Springer-Verlag, Berlin Germany, 1996.

[14] Jianying Zhou and Dieter Gollmann. A fair non-repudiation protocol. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Research in Security and Privacy, pages 55–61, Oakland, CA, May 1996. IEEE Computer Society,Technical Committee on Security and Privacy, The Institute of Electrical and Electronics Engineers IEEE Inc.: Computer Society Press.