RZ 3089 (#93135) 12/14/1998
Computer Science/Mathematics   11 pages

Research Report

Optimistic Synchronous Multi-Party Contract Signing

N. Asokan[1], Birgit Baum-Waidner[2], Matthias Schunter[3], Michael Waidner[1]

[1] IBM Zurich Research Laboratory
Säumerstrasse 4
CH-8803 Rüschlikon
Switzerland
{aso,wmi}@zurich.ibm.com

[2] Entrust Technologies Europe
CH-8301 Glattzentrum/Zürich
Switzerland
birgit.baum@entrust.com

[3] Universität des Saarlandes
Im Stadtwald 45
D-66123 Saarbrücken
Germany
schunter@cs.uni-sb.de

Optimistic Synchronous Multi-Party Contract Signing (Extended Abstract)*

N. Asokan[1], B. Baum-Waidner[2], M. Schunter[3], M. Waidner[1]

[1]IBM Zurich Research Laboratory, Rüschlikon, Switzerland
[2]Entrust Technologies Europe, Zürich, Switzerland
[3]Universität des Saarlandes, Fachbereich Informatik, Saarbrücken, Germany

Abstract

A contract is a non-repudiable agreement on a given contract text, i.e., a contract can be used to prove agreement between its signatories to any verifier. A contract signing protocol is used to *fairly* compute a contract so that, even if $n-1$ of the n signatories misbehave, either all or none of them obtain a contract.

Optimistic contract signing protocols use a third party to ensure fairness, but in such a way that the third party is not actively involved in case all parties are honest. Since no satisfactory protocols without any third party exist, this seems to be the best one can hope for.

We present an optimistic *multi-party* contract signing protocol for synchronous networks. The construction is significantly more efficient than the only known asynchronous multi-party contract signing protocol, and only 2-3 times more expensive than the trivial solution using an *inline* third party.

We show how to use multi-party contract signing to efficiently solve other atomicity problems securely, in particular *optimistic certified mail* and *optimistic fair exchange of signatures.* We also outline a generic construction for optimistic multi-party fair exchange.

*This report supersedes IBM Research Report RZ 2892, November 1996 [AsSW296]. Our work was supported by the ACTS Project AC026, *SEMPER.* However, it represents the view of the authors. *SEMPER* is part of the Advanced Communication Technologies and Services (ACTS) research program established by the European Commission, Directorate General XIII.

1 Introduction

1.1 Contract Signing and Verifiable Commit

In order to achieve *atomicity* or to prepare *dispute handling* business transactions often demand secure, verifiable agreement on how to proceed:

1. All parties involved must *agree* to either abort or to complete the transaction.
2. If the transaction is not aborted then the decision to continue must be *verifiable* by a third party (e.g., a court).

The parties in a business transactions are likely to not trust each other completely. Therefore the second requirement should be satisfied even if $n-1$ dishonest parties conspire against a single honest one.

As an example consider *electronic payments:* payer and payee have to agree to *transfer* the payment, or to *abort* it. If the transfer happens the payer requires a *proof* that will be accepted by any third party.

Probably the best example is *contract signing:* After having negotiated the contract text all signatories have to agree whether the contract shall become valid (signed) or not (failed). Any signatory must be able to show a signed contract to any third party (called *verifier* in the following). To a large extent, contract signing is actually the main tool to solve the general problem: Like an ordinary commit protocol [Lync96], contract signing ensures agreement, with a safe default decision failed ($\cong$ abort). But in addition it allows each party to *prove* to any verifier (such as a court) an agreement on signed ($\cong$ commit). Therefore one might call contract signing also *verifiable commit.*

The most trivial contract signing protocol uses a trusted third party, T, that first collects all inputs and then distributes the decision signed or failed [Rab183]. The protocol requires only $2n$ messages in two rounds of communication, but as the third party has to be involved in *all* protocol executions it might easily become a reliability and performance bottleneck.

Optimistic protocols avoid this bottleneck: Like the trivial protocol they depend on a third party to ensure fairness. But they do this in a way such that this third party is *not involved at all in the case where all parties are honest* – which hopefully is the most frequent case in real life. Already for the 2-party case no satisfactory contract signing protocols *without* any third party exist. Thus, the optimistic approach seems to be the best one can hope for.

Section 2 provides a concise definition of (optimistic) multi-party contract signing. In Section 3 we will present the first optimistic multi-party contract signing protocol.[1] Roughly speaking, the protocol (Protocol 2) is only 2-3 times as expensive as the trivial protocol: In the worst case Protocol 2 requires $6n-4$ messages in 6 rounds, in the all-honest case at most $4n-4$ messages in 4 rounds.

The protocol runs on synchronous networks only. The only known multi-party contract signing protocol for asynchronous networks is considerably more expensive: it requires $O(n^3)$ messages in $O(n)$ rounds, or $O(n^2)$ messages in $O(n^2)$ rounds [BaWa98].

In Section 4 we show how to use multi-party contract signing to efficiently solve other atomicity problems securely, in particular *optimistic certified mail* and *optimistic fair exchange of signatures.* We outline a generic construction for optimistic multi-party fair exchange. Optimistic *two-party* protocols for these problems were proposed in [AsSW97, AsSW198, BaDM98, Mica97].

1.2 Related Work

So far most work on contract signing (and fair exchanges in general) has been focused on the 2-party case:

There are 2-party contract signing protocols that do not involve any third party, but necessarily they introduce a non-negligible error. Basically the error probability is at least linear in the number of rounds [BGMR90], and obviously this result holds for the multi-party case as well.

The first *optimistic* 2-party contract signing protocol was proposed in [BGMR90], the first communication *optimized* ones in [AsSW97, PfSW98]. The first *asynchronous* 2-party protocol was described in [AsSW98, AsSW198].

[1] A preliminary version of Protocol 1 has been described in the technical report [AsSW296].

So far contract signing has not been used as verifiable commit. *Generic* optimistic 2-party fair exchange protocols (in the sense of Section 4.4) were proposed in [AsSW97, AsSW98]. Optimistic 2-party protocols for fair exchange of signatures or for certified mail have been described in [AsSW198, Mica97, BaDM98].

None of the optimistic 2-party protocols extends easily to the multi-party case. Some *non-optimistic* multi-party fair exchange protocols were described in [KeGa96, FrTs98]. The first *optimistic* multi-party fair-exchange protocols were described in a preliminary version of this paper [AsSW296], the first *asynchronous* optimistic multi-party contract signing protocol is described in [BaWa98]. As already mentioned, it is significantly less efficient than our construction in Section 3.

1.3 Model and Notation

Let $P_1, ..., P_n$ denote the parties directly involved in the contract signing, T the third party, and V any verifier. For simplicity we do not distinguish between a party and his or her protocol machine. We simply say a "party makes an input" ("receives an output") if the corresponding human user makes the input to (receives an output from) the corresponding machine. Each party is able to digitally sign messages, and to verify signatures of any other party [DiHe76, GoMR88]. The signature on message m associated with P_X is denoted by $\text{sign}_X(m)$.

Our contract signing protocol is structured in *synchronized* rounds of communication: Each party knows when a certain round starts and ends. In each round each party can send a message to each other party, and can process all messages received from all other parties. We further assume that messages sent between honest parties are reliably delivered within the same round.[2] This is the standard model of a *synchronized network* [Lync96]. An implementation has to justify these assumptions by means of reliable and secure message transport protocols, and reliable and secure clock synchronization.

We consider a static adversary who can a priori choose to corrupt a certain subset of all parties, i.e., gets full control over the behavior of those parties. The adversary can read all communication channels. We assume that the adversary cannot forge signatures [DiHe76, GoMR88].

For most security requirements we will assume that up to $n-1$ of all signatories might be corrupted. Fairness will always require T be non-corrupted, while unforgeability of a contract can be guaranteed even if T is corrupted.

The case where *all* parties are honest is called the *all-honest case.*

Disclaimer The following protocols are idealized, in several ways. In particular we omit all protocol and message identifiers in messages, all public key certificates, all time stamps and time outs, and even parameters such as the identity of T.

2 Definition of Multi-Party Contract Signing

Definition 1. (Multi-party Contract Signing)
A protocol for at least $n+1$ parties $P_1, ..., P_n, V$ that satisfies the following conditions is called a *Multi-Party Contract Signing* protocol, or just an *MPCS.*

An MPCS consists of two protocols, sign$[P_1, ..., P_n]$ and verify$[P_i, V]$. sign[] might involve an additional party, T, in which case we call it an *MPCS with third party.*[3] The machine V does *not* keep state between different executions of verify[] but always starts with the state it had immediately after initialization of the signature scheme. This models the fact that P_i shall be able to convince *any* verifier, i.e., V does not need to have any a priori knowledge about the contract.

A party P_i who wishes to start sign[] enters $(\mathsf{sign}, \mathit{tid}, \mathit{contr}, \mathit{decs})$. *tid* is a transaction identifier unique for all executions of sign[]. *contr* is the contract to be signed.[4] $\mathit{decs} \in \{\mathsf{sign}, \mathsf{reject}\}$ denotes the user's initial

[2] Actually reliable communication between signatories is needed only to ensure that T is not involved if all parties are honest. Otherwise it is sufficient if each signatory can reliably communicate with T.

[3] verify[] never involves T, i.e., it is always a 2-party protocol only.

[4] *tid* and *contr* must be chosen before the protocol can be started. *tid* might be randomly chosen by one party, say, P_1, and proposed to all others. In case the set of parties involved is not clear anyway, *tid* also specifies this set.

decision to sign or to reject the contract. Upon termination sign[] produces an output $(tid, contr, d_i)$ for P_i, with $d_i \in \{\mathsf{signed}, \mathsf{failed}\}$. We will simply say "$P_i$ decides d_i."[5]

A player P_i who wishes to start verify[] with a verifier V enters $(\mathsf{show}, tid, contr)$. If the verifier, V, wishes to start verify[] as well it enters $(\mathsf{verify}, tid, contr)$ where $tid, contr$ must be the same values as those used by P_i.

Upon termination verify[] produces an output $(tid, contr, d_V)$ with $d_V \in \{\mathsf{signed}, \mathsf{failed}\}$ for V. We will simply say "V decides d_V on tid." No output is produced by P_i.

The following conditions must be satisfied:

1. *Correct Execution.* If all parties P_i are honest and successfully started with $(\mathsf{sign}, tid, contr, decs = \mathsf{sign})$ then all parties will decide signed.

2. *Unforgeability of contract.* If honest P_i never entered $(\mathsf{sign}, tid, contr, decs)$ with $decs = \mathsf{sign}$ then any honest verifier that enters $(\mathsf{verify}, tid, contr)$ will decide failed. (Note that this does not assume an honest T.)

3. *Verifiability of valid contracts.* If honest P_i decides signed on input $(\mathsf{sign}, tid, contr, decs)$, and later inputs $(\mathsf{show}, tid, contr,)$ and honest V inputs $(\mathsf{verify}, tid, contr)$ then V will decide signed.

4. *No surprises with invalid contracts.* If T is honest and honest P_i entered successfully $(\mathsf{sign}, tid, contr, decs)$ with $decs = \mathsf{sign}$ but decided failed then for any $contr_V$ no honest verifier V entering $(\mathsf{verify}, tid, contr_V)$ will decide signed.

5. *Termination of* sign[]. If T is honest then each honest P_i that enters sign will terminate in a limited number of rounds.

6. *Termination of* verify[]. Each honest V that enters verify and each honest P_i that enters show will terminate in a limited number of rounds. ◊

Definition 2. (Optimistic Protocol)

A protocol for n regular parties $P_1, ..., P_n$ and a third party T is called *optimistic* if in the all-honest case the protocol terminates without T ever sending or receiving any messages. ◊

3 Synchronous Optimistic Contract Signing

The following protocol implements optimistic multi-party contract signing on synchronous networks.

In the all-honest case only two rounds of communication are needed: In the first round, each party who wishes to sign the contract broadcasts a signed "promise to sign" (= message $m_{1,i}$). In the second round each party who received all n promises from the first round actually signs the contract and broadcasts its real signature (= message $m_{2,i}$). Obviously this works if all parties wish to sign. If at least one party does not wish to sign this party will not send the signed promise, and thus no party will sign in Round 2. Thus, in the all-honest case the protocol stops after 2 rounds.

If some party cheats some honest parties might not end up with a signed contract in Round 2, while some others do. This inconsistency problem is solved by adding two more rounds to the protocol where everybody who has n signed promises from Round 1 can get them converted into a valid contract, by T. If T issues an affidavit it broadcasts it to all parties, in Round 4. Thus, each party who did not receive all n promises in Round 1 waits until Round 4. If it receives an affidavit from T the decision is signed, otherwise failed.

On *asynchronous* networks, the last "otherwise" would not be effective, as a party could not decide whether an affidavit was not sent, or just not delivered yet.

[5]T does neither get an input nor produces an output, as its behavior is completely determined by the protocol.

Protocol 1 (Synchronous Optimistic Multi-Party Contract Signing)

- **Signing:** Let $c := (\mathit{tid}, \mathit{contr})$.
  1. (a) If P_i starts with $\mathit{decs} = \mathsf{reject}$ it decides failed and *stops.*
     (b) Otherwise P_i sends message $m_{1,i} := \mathrm{sign}_i(1, c)$ to all other parties.
  2. Each P_i compiles $M_1 := (m_{1,1}, ..., m_{1,n})$.
     (a) If this succeeds and each $m_{1,j}$ is a valid signature $\mathrm{sign}_j(1, c)$ then P_i sends $m_{2,i} := \mathrm{sign}_i(2, c)$ to all other parties.
     (b) Otherwise P_i waits for a message from T in Round 4.
  3. Each P_i compiles $M_2 := (m_{2,1}, ..., m_{2,n})$.
     (a) If this succeeds and each $m_{2,j}$ is a valid signature $\mathrm{sign}_j(2, c)$ then P_i decides signed and *stops.*
     (b) Otherwise, and if P_i has sent $m_{2,i}$, it sends $m_{3,i} := \mathrm{sign}_i(3, M_1)$ to T.
  4. If T receives at least one message $m_{3,i}$ in Round 3 which contains a full and consistent M_1 then T sends $m_T := \mathrm{sign}_T(M_1)$ to all parties, and each P_i receiving this decides signed. (Otherwise T does not send anything, and is actually not even aware of this protocol run.)

     Each P_i waiting for a message from T in Round 4 decides failed if none arrives, or signed in case m_T is received, and *stops.*

- **Verification:** Any verifier V accepts a contract if it sees a full and consistent M_2 or an m_T containing a full and consistent M_1. $\diamondsuit$

In the all-honest case, each party broadcasts at most 2 messages in 2 rounds. For $n = 2$ Protocol 1 becomes identical to Scheme 2 of [PfSW98]. Using their argument one can trivially conclude that Protocol 1 is round-optimal.

In any case, the protocol needs at most 4 rounds of communication. In the all-honest case $2n$ messages are broadcast to $n - 1$ parties. If some parties cheat one has to add, in the worst case, n messages of type $m_{3,i}$, and one broadcast message from T.

If broadcast is implemented by sending individual messages this amounts to a total of $2n^2 - 2n$ messages in the all-honest case, and $2n^2$ messages in the worst case. The number of messages can be reduced to $4n - 4$ messages in the all-honest case and $6n - 4$ messages in the worst case, by adding two rounds to the optimistic part: We already mentioned that reliability of communication between signatories is not essential for security. Anyway, each party can cause the protocol to decide for failed by just setting $\mathit{decs} := \mathsf{reject}$. Thus it does not reduce security if the full information exchanges of the first two rounds are *relayed* via one party, say P_1:

Protocol 2 (Message-minimized Version of Protocol 1)

The protocol differs from Protocol 1 only in the way messages are sent. Round k, $k = 1, 2$, of Protocol 1, is replaced by the following two rounds:

1. P_i sends $m_{k,i}$ to P_1 only.
2. P_1 collects M_k.
   (a) If this succeeds, P_1 sends M_k to all other parties.
   (b) Otherwise P_1 *stops.* $\diamondsuit$

Theorem 1

Protocols 1 and 2 are optimistic MPCSs for synchronous networks. $\diamondsuit$

Proof. Security-wise Protocols 2 and 1 are identical: the proofs are literally the same for both variants. "Round" always corresponds to the rounds of Protocol 1.

- *Correct execution, verifiability* and *termination* are obviously satisfied.
- *Strong unforgeability* is given by the fact that a valid contract contains a signature from each party, i.e., all parties must have agreed to sign.
- *No surprises with invalid contracts.* Assume an honest V accepts c as signed. If this happens because of m_T then T has distributed this message to all parties, and all honest parties decide signed. Now assume V accepts because of M_2, and consider some honest participant P_i. As M_2 contains P_i's signature from Round 2, P_i received M_1 in Round 1. If P_i received M_2 in Round 2 it decides signed. Otherwise it sends $m_{3,i}$ to T, which is necessarily answered by m_T (as T is honest), and P_i decides signed. Thus we know that if V accepts then any honest party has accepted the contract.
- *Optimistic.* Assume all parties are honest. If all parties start with $decs = \mathsf{sign}$ and consistent contracts then T will not be involved and everybody decides signed in Round 2. If at least one party P_i starts with $decs_i = \mathsf{reject}$ then $m_{1,i}$ will not be sent, M_1 will be incomplete, no other party can send $m_{2,j}$, thus none will contact T, and in Round 4 all parties that started with $decs_i = \mathsf{sign}$ will decide failed. □

Remark 1. For $n = 2$, Protocol 2 is *not* message optimal: In the all-honest case, Protocol 2 needs 4 rounds and 4 messages. In [PfSW98] it was shown that on synchronous networks 3 rounds and 3 messages are sufficient. ◁

4 Applications

4.1 General Approach

In the following we will discuss three examples of how to construct optimistic protocols for certain other multi-party problems, using MPCS as a building block. All these applications follow a simple pattern, consisting of 3 phases:

1. All parties *prepare* the transaction but do not do any irreversible changes. In the all-honest case this does not involve any third party.
2. An optimistic MPCS is used to sign a contract that exactly specifies the transaction. A party signs only if Phase 1 succeeded.
3. (a) If the MPCS resulted in signed then the transaction is finalized. In the all-honest case this does not involve any third party, but if some parties cheat, T can help to finish the transaction. This help is subject to the existence of a valid, signed contract from Phase 2.
   (b) If the MPCS resulted in failed then the transaction is aborted. This might require to undo some of the effects of Phase 1.

In most examples Phase 3 does never involve T. In this case the pattern results in an asynchronous protocol, provided all the sub-protocols — in particular the MPCS — run asynchronously as well.

In those applications where Step 3b might involve T, the third party needs to verify that the contract is *not* signed before participating. The only way to do this is to ask all parties whether any of them can show a signed contract. If none of them does T decides failed.

Remark 2. It is not possible to construct an *asynchronous* MPCS where a verifier can verify both signed *and* failed: Consider a case where P_1 is the only honest party, and the MPCS terminates with signed, without any involvement by T. Nevertheless the $n-1$ dishonest parties can *simulate* a protocol run where P_1 refused to participate. Necessarily, this simulated protocol run yields failed. Thus, not both results can be verifiable.

On synchronous networks failed is verifiable in the weak sense outlined above: a verifier can ask all parties involved to *show* the contract, and if this does not happen, concludes it is failed.

This is a safe decision: If any party (honest or dishonest) has a signed contract, then *all* parties have a signed contract, and T will receive at least one signed contract. Thus, T decides signed. If any honest party decided failed then *no* party (honest or dishonest) has a signed contract, and thus T will decide failed. $\triangleleft$

4.2 Optimistic Multi-Party Certified Mail

In two-party certified mail the sender P_1 sends a message m to P_2 in exchange for a receipt that can be shown to any verifier V. P_2 has to give the receipt *blindly,* i.e., independent of the contents of m.

There are several ways how to generalize this to the multi-party case:

- One natural multi-party version of certified mail is *one-to-many certified mail:* P_1 sends a certified mail m to $P_2, ..., P_n$ and requires to get a receipt from *each* recipient in exchange. Either P_1 receives all receipts, or none of $P_2, ..., P_n$ gets any information about m.
- Similarly one can define *many-to-one certified mail,* where each P_i ($i \in \{2, ..., n\}$) sends a certified mail m_i to P_1, and requires to get a receipt from P_1 in exchange. Either each sender P_i receives a receipt, or P_1 gets no information about any of the m_i.
- One can also define more complicated communication patterns. In the "worst" case each party P_i has a certified mail $m_{i,j}$ for each other party $P_j, i \neq j$, and either all $n(n-1)$ receipts are generated or no information about any of the $m_{i,j}$ is leaked.

We can solve any of these problems using the approach sketched in Section 4.1. For simplicity we describe it for one-to-many certified mail only. As an additional tool we need a public-key encryption scheme secure against adaptive chosen-ciphertext attacks [DoDN91, CrSh98]. Only T needs a key pair. Let nil denote a default message.

Protocol 3 (Optimistic One-to-many Certified Mail)

1. P_1 encrypts (tid, m) under the public key of T, and sends the ciphertext to all parties. Call this ciphertext *cipher*.
2. Each party receiving *cipher* and willing to accept it sets $decs =$ sign. Otherwise $decs =$ reject. We run an optimistic contract signing protocol on $(tid, contr = cipher, decs)$. If $P_i, i = 1, ..., n$, decides failed in the contract signing protocol then it decides failed in the certified mail protocol and *stops.*
3. P_1 sends m to all parties, and proves that (tid, m) was actually the cleartext corresponding to *cipher* (e.g., by sending also all random coins used in computing *cipher*).

   P_1 decides accepted in the certified mail protocol. The receipt is the signed contract.
4. If $P_i, i > 1$, received a correct pair (tid, m) it accepts m and *stops.* Otherwise P_i shows the contract $(tid, cipher, decs)$ to T.
5. If T receives a signed contract $(tid, cipher)$ from P_i then it tries to decrypt *cipher*. Otherwise the request is ignored.

   (a) If decryption succeeds and results in (tid, m) for the given *tid* then T sets $m_T := \text{sign}_T(tid, cipher, m)$.

   (b) Otherwise T sets $m_T := \text{sign}_T(tid, cipher, \text{nil})$.

   T sends m_T to P_i. $\diamondsuit$

Theorem 2 (Security of Protocol 3 – Informally)
Protocol 3 is a secure protocol for optimistic one-to-many certified mail. $\diamondsuit$

Proof. (Sketch)
We have to prove two properties: No information is leaked on m unless P_1 has a receipt. And P_1 cannot get a receipt without revealing m:

- The only way to get information about *cipher* is from P_1 or T – otherwise we could use Protocol 3 to successfully attack the encryption scheme, which we excluded by assumption.

  P_1 reveals m only if he has a signed contract, i.e., a receipt.

  T reveals m only if the condition in Step 5a is satisfied. This means the cleartext of *cipher* and the signed contract agree on *tid*, i.e., correspond to the same protocol run. Thus, P_1 was indeed the creator of *cipher*, and agreed to exchange m for a receipt. Thus, T can safely decrypt *cipher*.

- Since the MPCS is assumed to be secure, the only way to get a valid receipt is if Phase 2 resulted in a signed contract. Based on this contract each P_i will finally get the corresponding message, either from P_1 in Phase 3 or from T in Phase 5. $\square$

Remark 3. Using an asynchronous MPCS [BaWa98] the whole construction would run asynchronously. $\triangleleft$

4.3 Optimistic Multi-Party Fair Exchange of Signatures

In a fair exchange of signatures each party P_i knows a signature s_i on a message m_i. The messages $m_1, ..., m_n$ are known to all parties. At the end either all parties know all signatures, or none of them is revealed (i.e., the adversary's chances to compute a signature on any of the m_i has not been increased).

Protocols for the 2-party case were described in [AsSW198, BaDM98]. In [AsSW198] it was shown how to efficiently and fairly exchange signatures from a large class of schemes, including DSS and RSA. The key tool for these protocols is *verifiable encryption* of signatures [Stad96], in the variant defined in [AsSW198]. In principle this is an ordinary public key encryption scheme with an additional property: Let $E_T(x)$ denote the encryption of x under T's public key. Let A and B be two parties. A knows a signature s on message m, and a string w. B knows message m and string w. Let $c := E_T(s, w)$. Verifiable encryption allows A to *prove* to B that c actually includes a signature on m, and string w, *without* revealing s.[6]

Multi-party fair exchange of signatures can be used to implement multi-party contract signing with the nice property that the application can demand how a "signed contract" shall look like (which is called "non-invasive contract signing" in [AsSW98, AsSW198]). For instance, it might be most natural to say that the contract on m shall be any vector of n valid signatures: $(\text{sign}_1(m), ..., \text{sign}_n(m))$, where each P_i is free to select whatever signature scheme suits him or her best. The protocols in Section 3 would not support this requirement, as they define a contract to be either M_2 or m_T.

Using verifiable encryption of signatures we can implement multi-party fair exchange of signatures:

Protocol 4 (Optimistic Multi-Party Fair Exchange of Signatures)

1. Let string w be all information that describes the current exchange. This contains in particular *tid*, and for each P_i the message m_i to be signed and the public key needed to test the expected signature s_i on m_i. w is known to each P_i.

   Let s_i denote the signature by P_i on m_i. Initially s_i is known to P_i only.

[6] Saying that c contains an encryption of s is a slight idealization. In reality one first reduces knowledge of a signature to knowledge of a single discrete logarithm x or of a single e-th root x, and then encrypts this x. B knows not just m but also g^x or x^e, respectively [AsSW198].

Each party P_i sends a verifiable encryption $c_i = E_T(s_i, w)$ to each P_j, and proves to each P_j that this actually includes a signature s_i and w.[7] Each party sets $decs = \mathsf{sign}$ if all these proofs succeed, otherwise $decs = \mathsf{reject}$.

2. We run an optimistic contract signing protocol on $(tid, contr = w, decs)$.

3. If P_i decided signed in Phase 2, it sends s_i to all parties and waits for their signatures. If one such signature, say s_j, does not arrive, P_i can show the signed contract on w and the ciphertext c_j to T.

   Given a valid contract T decrypts c_j and retrieves (s_x, w_x). If $w_x = w$ and s_x is a signature on m_x, where m_x is defined by w, then T sends s_x to the requester.[8] $\diamondsuit$

Theorem 3 (Security of Protocol 4 – Informally)
Protocol 4 is a secure protocol for optimistic multi-party fair exchange of signatures. $\diamondsuit$

We omit the proof of security for Protocol 4 from this extended abstract. On a high level it is very similar to the one for Theorem 2. Note that each ciphertext c_i contains the contract "text" w, which ensures that T decrypts c_i only in case the right contract is shown.

4.4 Optimistic Multi-Party Fair Exchange of Items

In this last section we will sketch a generic construction for multi-party fair exchange.

Informally, most items that could be fairly exchanged belong to one of the following classes [AsSW97, AsSW98]: Let S be the sender of an item, and R the recipient.

- *Revocable items* have the property that the third party T can undo a transfer from S to R, on request by S. For instance, several payment systems support revocability.
- *Generatable items* have the property that after a certain pre-processing step between S and R, the third party can generate the item with R's help, without involving S. In the pre-processing step S can specify a condition that T can later check to distinguish between legitimate and fraudulent requests for help by R.

  For instance, verifiable encryption made signatures (respectively discrete logarithms and e-th roots) generatable. The condition was that there must be a valid contract on w, signed by P_i.
- *Forwardable items* have two transfer protocols, one just between S and R, and a second one involving S, R and T which ensures that T can verify the correctness of the transfer. We call that second transfer protocol *observable*. We assume that transferring an item is idempotent.

Using this classification of items we can design a *generic* multi-party fair exchange protocol. For simplicity we assume that each party has exactly one item that he or she wants to transfer. (Other communication patterns can be supported in a similar way.) Let P_R, P_G, P_F be the parties who wish to transfer revocable, generatable or forwardable items, respectively. We assume $|P_R| + |P_G| + |P_F| = n$ and $|P_F| \leq 1$.

Protocol 5 (Optimistic Multi-Party Fair Exchange of Items)

1. Each party $P_r \in P_R$ transfers its item to all other parties. Each party $P_g \in P_G$ transfers those information to all other parties that is necessary in case a recipient would need T's help in generating P_g's item. Each party checks whether all steps were performed correctly, and sets $decs_i$ accordingly.

2. We run an optimistic contract signing protocol on $(tid_i, contr_i, decs_i)$, where $contr_i$ describes exactly what each party is supposed to send, and the information received so far.

[7] In some verifiable encryption schemes the same ciphertext cannot be verified more than once. In this case P_i has to generate $n-1$ versions of c_i, one for each party P_j.

[8] Depending on the application the requesting party might have to prove that it actually participated in the protocol.

3. If Phase 2 ended with signed and $P_F = \{P_f\}$ then P_f transfers its item. If any party P_j does *not* receive P_f's item it asks T for help. T asks P_f to transfer its item again, using the observable transfer protocol. If this fails, T sends an abort message to all parties. (Otherwise everybody has P_f's item now.)

4. (a) If Phase 2 ended with signed and T did not send the abort message in Phase 3 then all $P_g \in P_G$ transfer their items. If any party P_j does not receive a generatable item then it generates it with T's help. In order to get P_g's item generated by T, P_j has to show a valid contract that is signed by P_g. This contract, i.e., *contr*, must be sufficient for T to check that the condition set by P_g is satisfied.

   (b) Otherwise all parties $P_r \in P_R$ ask T to revoke the transfers of their items. This must be bound to the fact that there is *no* signed contract. In order to check this T sends a request to all parties involved to show a signed contract. If no party can show one, all transfers are revoked. $\diamondsuit$

Since we did not provide any definitions for the three types of items we omit the proof of security for Protocol 5 from this extended abstract.

Intuitively the construction is secure as it exactly implements the pattern discussed in Section 4.1:

- No irreversible changes are made in Phase 1. The transfers of the $P_r \in P_R$ can be revoked. The preparations of the transfers of the $P_g \in P_G$ reveal no information, by assumption.
- In Phase 2 either all parties agree to finalize the exchanges, or to abort them.

  If they abort no information on the items is leaked, and T will not co-operate in the generation of any item.

  Otherwise T can complete all transfers of the $P_g \in P_G$ and can prevent all transfers of the $P_r \in P_R$ from being revoked. The only transfer that cannot be enforced by T is the one by $P_f \in P_F$, which is taken care of in Phase 3.
- Phase 3 ensures that either all parties receive P_f's item, or the exchange is aborted.

  (Obviously the construction of this phase works only for a single P_f, and would not work on an asynchronous network.)
- Phase 4 is entered only if the items of all $P_i \in P_R \cup P_F$ are already transferred and the transaction is not aborted (because the contract is signed in Phase 2 and was not aborted in Phase 3).

  None of the revocable transfers will be revoked, as T can always correctly determine whether the contract was signed successfully or not. (Obviously, this would not work on an asynchronous network.)

  Each party P_i will receive the item of each $P_g \in P_G$, either directly or with T's help, as P_i can always show a valid contract which was not aborted in Phase 3.

Remark 4. Unless $P_R = P_F = \emptyset$ Protocol 5 works on synchronous networks only. $\triangleleft$

5 Summary

We presented the first *optimistic* multi-party contract signing protocol for synchronous networks, and proved its security.

In the worst case Protocol 2 requires $6n-4$ messages in 6 rounds, in the all-honest case at most $4n-4$ messages in 4 rounds. The trivial solution using an inline third party requires only $2n$ messages in 2 rounds, but depends on the third party to be involved in *each* transaction.

It depends on the application which of the two approaches is more appropriate: If the all-honest case is the most likely one, and if communication with T is relatively expensive, then the optimistic approach seems to be most appropriate.

Our main application of multi-party contract signing is as a primitive for secure atomic transactions, similar to distributed commit for ordinary atomic transactions. As an example we showed how to solve several fair exchange problems using multi-party contract signing.

Acknowledgments: We thank *Birgit Pfitzmann, Michael Steiner,* and *Victor Shoup* for interesting discussions.

References

[AsSW296] N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Multi-Party Fair Exchange; IBM Research Report RZ 2892, IBM Zurich Research Laboratory, Zürich, November 1996.

[AsSW97] N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Fair Exchange; 4th ACM Conference on Computer and Communications Security, Zürich, April 1997, 6–17.

[AsSW98] N. Asokan, Victor Shoup, Michael Waidner: Asynchronous Protocols for Optimistic Fair Exchange; 1998 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, Los Alamitos 1998, 86–99.

[AsSW198] N. Asokan. Victor Shoup, Michael Waidner: Optimistic Fair Exchange of Digital Signatures; Eurocrypt '98, LNCS 1403, Springer-Verlag, Berlin 1998, 591–606.

[BaDM98] Feng Bao, Robert Deng, Wenbo Mao: Efficient and Practical Fair Exchange Protocols with Off-Line TTP; 1998 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, Los Alamitos 1998, 77–85.

[BaWa98] Birgit Baum-Waidner, Michael Waidner: Asynchronous Optimistic Multi-Party Contract Signing; IBM Research Report RZ3078 (#93124), IBM Zurich Research Laboratory, Zürich, November 1998.

[BGMR90] Michael Ben-Or, Oded Goldreich, Silvio Micali, Ronald L. Rivest: A Fair Protocol for Signing Contracts; IEEE Transactions on Information Theory 36/1 (1990) 40–46.

[CrSh98] Ronald Cramer, Victor Shoup: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack; Crypto '98, LNCS 1462, Springer-Verlag, Berlin 1998, 13–25.

[DiHe76] Whitfield Diffie, Martin E. Hellman: New Directions in Cryptography; IEEE Transactions on Information Theory 22/6 (1976) 644–654.

[DoDN91] Danny Dolev, Cynthia Dwork, Moni Naor: Non-Malleable Cryptography; 23rd Symposium on Theory of Computing (STOC) 1991, ACM, New York 1991, 542–552.

[EvGL85] Shimon Even, Oded Goldreich, Abraham Lempel: A Randomized Protocol for Signing Contracts; Communications of the ACM 28/6 (1985) 637–647.

[FrTs98] Matt Franklin, Gene Tsudik: Secure Group Barter; Financial Cryptography 1998, *to appear.*

[GoMR88] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM Journal on Computing 17/2 (1988) 281–308.

[KeGa96] Steven P. Ketchpel, Hector Garcia-Molina: Making Trust Explicit in Distributed Commerce Transactions; 16th International Conference on Distributed Computing Systems,1996, IEEE Computer Society, 1996, 270–281.

[Lync96] Nancy A. Lynch: Distributed Algorithms; Morgan Kaufmann, San Francisco 1996.

[Mica97] Silvio Micali: Certified E-Mail with Invisible Post Offices; presented at 1997 RSA Conference.

[PfSW98] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Optimal Efficiency of Optimistic Contract Signing; ACM Principles of Distributed Computing (PODC), Puerto Vallarta, June 1998, 113–122.

[Rab183] Michael O. Rabin: Transaction Protection by Beacons; Journal of Computer and System Sciences 27/ (1983) 256–267.

[Stad96] Markus Stadler: Publicly verifiable secret sharing; Eurocrypt '96, LNCS 1070, Springer-Verlag, Berlin 1996, 190–199.