Improved Multi-party Contract Signing

Aybek Mukhamedov and Mark Ryan

School of Computer Science
University of Birmingham, UK
{A.Mukhamedov, M.D.Ryan}@cs.bham.ac.uk

Abstract. A multi-party contract signing protocol allows a set of participants to exchange messages with each other with a view to arriving in a state in which each of them has a pre-agreed contract text signed by all the others. "Optimistic" such protocols allow parties to sign a contract initially without involving a trusted third party T. If all signers are honest and messages are not arbitrarily delayed, the protocol can conclude successfully without T's involvement. Signers can ask T to intervene if something goes amiss, for example, if an expected message is not received. Two multi-party contract signing protocols have been proposed so far.

One solution to this problem was proposed by Garay and MacKenzie (DISC'99) based on private contract signatures, but it was subsequently shown to be fundamentally flawed (it fails the fairness property). Another more efficient protocol was proposed by Baum-Waidner and Waidner (ICALP'00). It has not been compromised, but it is based on a non-standard notion of a signed contract.

In this paper, we propose a new optimistic multi-party contract signing protocol based on private contract signatures. It does not use a non-standard notion of a signed contract and has half the message complexity of the previous solution.

1 Introduction

A contract signing protocol allows a set of participants to exchange messages with each other with a view to arriving in a state in which each of them has a pre-agreed contract text signed by all the others. An important property of contract signing protocols is *fairness*: no participant should be left in the position of having sent another participant his signature on the contract, but not having received signatures from the other participants.

One way in which this can be achieved is by employing a trusted party T. All the signers of the contract send their signatures to T. When T has them all, he sends them out to each of the signers. It would be desirable to have a protocol which does not require a trusted party, but this is known to be impossible for deterministic protocols [7]. This has led to the invention of "optimistic protocols", which employ a trusted party only in the case that something goes wrong. If all the signers are honest and there are no adverse network delays which prevent the protocol from completing, the trusted party is not needed. But if a participant

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-540-77366-5_37
S. Dietrich and R. Dhamija (Eds.): FC 2007 and USEC 2007, LNCS 4886, pp. 179–191, 2007.

of the protocol has sent messages which commit him to the contract and has not received corresponding commitment from the other participants, he can contact the trusted party who will intervene.

As well as fairness, there are other desired properties of contract signing protocols. *Timeliness* ensures that every signer has some recourse to prevent endless waiting. A third property called *abuse-freeness* guarantees that a signer is not able to prove to an external observer that she is in a position to choose between successfully completing the protocol and aborting it. This property is desirable because being in such a position would give the signer an unfair advantage.

Optimistic contract signing protocols have been first described for *synchronous* networks in [1,2,11,13]. 2-party protocols for *asynchronous* networks, have been proposed in [3,8,13], where all messages are eventually delivered, but without upper bounds on network delays. Later, two protocols for n-party case have been proposed: one by Garay and MacKenzie [9], and the other one by Baum-Waidner and Waidner [5]. Chadha, Kremer and Scedrov in [6] revealed and claimed to have fixed a flaw in the trusted party's protocol of Garay and MacKenzie's scheme. Later, we showed that Garay and MacKenzie's main protocol is flawed for $n > 4$ and fairness can not be restored whatever the trusted party does [12].

Baum-Waidner and Waidner's protocol requires $(n+1)n(n-1)$ messages in the "optimistic" execution, where n is the number signers and the number of dishonest signers can be up to $n-1$. However, their protocol is based on a non-standard notion of a signed contract: a contract on a text m signed by an agent A is defined to be a tuple $(m, n+1)$ digitally signed by A. Any other digitally signed (m, i) with $i < n+1$ is not considered to be a signed contract; it is merely A's promise to sign the contract. Such a notion has undesirable side-effects. The validity of the contract produced by Baum-Wainder and Waidners's protocol depends on the integer it is tupled with. Hence, when a party is presented with such contract it must be able to reliably establish $n+1$ (which could, for instance, be embedded in the body of the contract m) and compare with the integer that the contract is tupled with.

Baum-Waidner[4] further reduced the complexity of the previous scheme. This was achieved by adjusting trusted party T's protocol with an assumption that T knows in advance the number of dishonest signers (and sets the parameters of its protocol accordingly) and fairness is guaranteed provided *all* honest signers continue the protocol (i.e. if some honest signer decides to quit, when the protocol requires it to participate, fairness can not be guaranteed for other honest signers).

Our contribution. We propose a new optimistic multi-party contract signing protocol based on private contract signatures. It does not use a non-standard notion of a signed contract and achieves improvement in the message complexity of the optimisitc execution without assuming that T or any signer know the total number of dishonest signers. Our scheme requires $n(n-1)\lceil n/2 \rceil + 1$ messages, which is about half the complexity of the previous protocol by Baum-Waidner and Waidner [5]. For example, if $n = 6$ our protocol requires 120 messages to "optimistically" sign a contract, whereas the previous scheme requires 210.

2 Model and Definitions

Let $P_1, \ldots, P_n$ denote signers, who want to sign a contract m and T a trusted third party. Signers may be honest, in which case they execute the protocol faithfully or *dishonest*, i.e. they deviate from the protocol. We assume that up to $n-1$ of signers may be dishonest and are coordinated by a single party, the *adversary*. We assume that the ordered list $(P_1, P_2, \ldots, P_n)$ of signers is fixed in advance and included in the text m of the contract, and that all signers reliably know each others public key, and all contracts are distinct. $S_{P_i}(m)$ denotes P_i's universally-verifiable signature on m.

We shall say that P_i has a *valid* contract m, if it receives all signers' signatures on m. When P_i runs a contract signing protocol and acquires a valid contract m, we shall say "P_i decided signed". Otherwise, if it quits or receives an abort token from T we say "P_i decides failed".

We consider an asynchronous communication model with no global clocks, where messages can be arbitrarily delayed. However, the communication channels between signers and the trusted party T are assumed to be *resilient*, viz. the messages are guaranteed to be delivered eventually. The adversary is allowed to schedule and insert its own messages into the network. The protocol is expected not to fail, whatever such adversary does.

An optimistic contract signing protocol consists of two protocols, one executed by signer (*Main*), and another by trusted party T (*Abort* or *Resolve*). Usually signers try to achieve the exchange by executing Main. They contact T using Abort or Recovery only if something goes amiss in Main. Once a participant contacts T, it no longer takes part in Main. A request to T via Abort or Recovery can result in T sending back an abort token or a signed contract. The decision of whether to reply with an abort token or a signed contract is taken by T on the basis of the evidence included in the request, and also the previous requests that have been made by other participants. T has the property that if it decides to send back a signed contract, it sticks to that decision when answering further requests from other participants. However, if it issues an abort, it may later overturn that abort in order to maintain fairness. An honest participant (namely, one who adheres to the protocol) will not receive an abort and later have it overturned.

An optimistic contract signing protocol is expected to guarantee *fairness*. It is also desirable for the protocol to guarantee *abuse-freeness* and *timeliness*:

Definition 1. *An optimistic contract signing protocol is said to be* fair *for an honest signer P_i if whenever some signer P_j obtains $S_{P_i}(m)$ then P_i obtains $S_{P_k}(m)$ for all $1 \leq k \leq n$.*

Definition 2. *An optimistic contract signing protocol is said to be* abuse-free *if it is impossible for any set of signers at any point in the protocol to be able to prove to an outside party that they have the full power to terminate (abort) or successfully complete the contract signing.*

Definition 3. *An optimistic contract signing protocol is said to satisfy* timeliness *if each signer has recourse to stop endless waiting.*

3 The Protocol

The following is an optimistic multi-party contract signing protocol. The Main protocol, consists of $\lceil n/2 \rceil + 1$ rounds. In each round a signer P_i waits for promises from lower numbered signers (*below*), sends its promise to higher numbered signers (*above*), waits for promises from signers above and then send its promise to signers below. In the last round signers exchange actual signatures, together with their promises. If a signer does not receive some of the messages, it either quits the protocol or asks T to intervene.

PCS promises. Our protocol employs a cryptographic primitive known as *private contract signature* [8]. A private contract signature by P_i for P_j on text m with respect to trusted party T, denoted $PCS_{P_i}(m, P_j, T)$, is a cryptographic object with the following properties:

1. $PCS_{P_i}(m, P_j, T)$ can be created by P_i, and faked by P_j.
2. Each of P_i, P_j and T (but no-one else) can tell the difference between the versions created by P_i and faked by P_j.
3. $PCS_{P_i}(m, P_j, T)$ can be converted into a regular universally-verifiable signature by P_i, and by T; and by no-one else.

The idea is that $PCS_{P_i}(m, P_j, T)$ acts as a promise by P_i to P_j to sign m. But P_j cannot prove to anyone except T that he has this promise, since he can create it himself and only T can tell the difference between one created by P_i and one created by P_j.

In our protocol, agents exchange several such promises before issuing a signed contract. A promise issued by a signer at a later stage of the protocol signifies its stronger commitement to the contract as well possession of certain promises from other signers. Hence, τ-level promise of a signer P_i to P_j on m is a message $PCS_{P_i}((m, \tau), P_j, T)$, where $\tau \geq 0$ expresses the temporal ordering of P_i promises.

3.1 Main Protocol for Signer P_i

Each signer waits for 1-level promises from the signers below. On receipt of these, it sends its 1-level promises to the signers above it. Then it waits for 1-level promises from above, and on receipt, sends 1-level promises below. This sequence is repeated for r-level promises, for r ranging from 2 to $\lceil n/2 \rceil$, as shown in Figure 3.1. Finally, in the last round, $\lceil n/2 \rceil + 1$-level promises and signatures are exchanged. The protocol is defined formally in Table 1.

If expected messages are not received, a participant P_i may simply quit the protocol, or request abort or resolve from T, depending on where P_i is in the main protocol.

When P_i requests abort it sends to T the message:

$$S_{P_i}((m, P_i, (P_1, \ldots, P_n), abort))$$

For the resolve requests P_i sends

$$S_{P_i}(\{PCS_{P_j}((m,\tau_j),P_i,T)\}_{j\in\{1,\ldots,n\}\setminus\{i\}},S_{P_i}(m,0))$$

to T, where for $j>i$, τ_j is the maximum level of promises received from all signers P'_j with $j'>i$, and for $i>j$, τ_j is the maximum level of promises received from all signers P'_j with $i>j'$:

$$\tau_j=\begin{cases}\max\{\tau \mid \forall j'>i, \quad P_i \text{ has received } PCS_{P_{j'}}((m,\tau),P_i,T)\} & \text{if } j>i\\ \max\{\tau \mid \forall j'<i, \quad P_i \text{ has received } PCS_{P_{j'}}((m,\tau),P_i,T)\} & \text{if } j<i\end{cases}$$

(For example, if the maximum level promises P_4 receives from P_1 and P_2 is 3, and from P_3 it is 2, then P_4 would send 2-level promises for signers below.)

Table 1. Main protocol for signer P_i

Round 1

1. For each $j<i$, wait for promise $PCS_{P_j}((m,1),P_i,T)$ from P_j.
   If any of them is not received in a timely manner, then quit.
2. For each $j>i$, send promise $PCS_{P_i}((m,1),P_j,T)$ to P_j.
3. For each $j>i$, wait for promise $PCS_{P_j}((m,1),P_i,T)$ from P_j.
   If any of them is not received in a timely manner, then request abort.
4. For each $j<i$, send promise $PCS_{P_i}((m,1),P_j,T)$ to P_j.

For $r=2$ to $\lceil n/2\rceil$: Round r

5. For each $j<i$, wait for promise $PCS_{P_j}((m,r),P_i,T)$ from P_j.
   If any of them is not received in a timely manner, then request resolve.
6. For each $j>i$, send promise $PCS_{P_i}((m,r),P_j,T)$ to P_j.
7. For each $j>i$, wait for promise $PCS_{P_j}((m,r),P_i,T)$ from P_j.
   If any of them is not received in a timely manner, then request resolve.
8. For each $j<i$, send promise $PCS_{P_i}((m,r),P_j,T)$ to P_j.

Round $\lceil n/2\rceil+1$

9. For each $j<i$, wait for promise $PCS_{P_j}((m,\lceil n/2\rceil+1),P_i,T)$ and signature $S_{P_j}(m)$ from P_j.
   If any of them is not received in a timely manner, then request resolve.
10. For each $j\neq i$, send promise $PCS_{P_i}((m,\lceil n/2\rceil+1),P_j,T)$ and signature $S_{P_i}(m)$ to P_j.
11. For each $j>i$, wait for promise $PCS_{P_j}((m,\lceil n/2\rceil+1),P_i,T)$ and signature $S_{P_j}(m)$ from P_j.
    If any of them is not received in a timely manner, then request resolve.

3.2 Protocol for T

For each contract m with signers $P_1,\ldots,P_n$, when T learns about the contract (through abort or resolve request) it sets up a variable `validated`(m) initiated

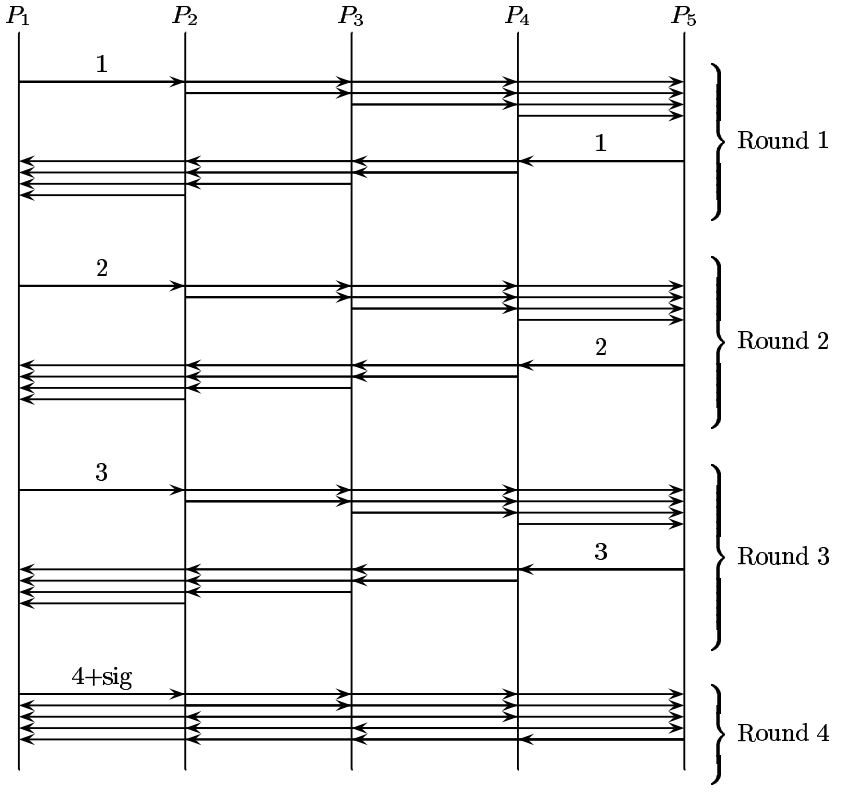

Fig. 1. Messages in the main protocol when $n = 5$

to false, which indicates if T decided to enforce the contract and has a full set of signatures (some converted by T from promises). T must reliably know the position of each signer in the run of the protocol, which can be deduced from the ordered list of signers included in the contract text m. T also maintains a set $S(m)$ of indexes of parties that contacted it in the past: signers are allowed to contact T only once. This set is also used when T considers whether to overturn its previous abort decision. For each signer P_i such that $i \in S(m)$, T also maintans two integer variables $h_i(m)$ and $l_i(m)$. Intuitively, $h_i(m)$ is the highest level promise P_i could have sent to any signer above, and similarly, $l_i(m)$ is the highest level of promise P_i could have sent to a signer below. This construction was inspired by the paper of Chadha, Kremer and Scedrov [6], even though it does not work for the protocol they consider [12].

Depending on the request T executes either Abort or Resolve protocol.

Abort protocol. When T receives an abort message from P_i, it adds i to the set $S(m)$. Then if the protocol has already been successfully recovered it sends back a signed contract; otherwise, it sends back an abort token (see table 2).

Table 2. Abort protocol for T

The first time T is contacted for contract m (either abort or recovery), T initialises $S(m)$ to $\emptyset$ and `validated`(m) to false.

If the abort message $S_{P_i}(m, P_i, (P_1, \ldots, P_n), abort)$ is received from P_i
Check that the signature is valid

if not `validated`(m) then
if $S(m) = \emptyset$ then store $S_T(S_{P_i}(m, P_i, (P_1, \ldots, P_n), abort))$
$S(m) = S(m) \cup \{i\}$
$h_i(m) := 1$; $l_i(m) = 0$
Send $S_T(S_{P_j}(m, P_j, (P_1, \ldots, P_n), abort))$ to P_i

else
Send $\{S_{P_j}((m, \tau_j))\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$ to P_i
where τ_j is the level of the promise from P_j that was converted to a universally-verifiable signature during the recovery protocol.

Resolve protocol. The recovery messages that T receives are designed so that T can infer what promises an honest signer could have sent and whether all the previous requests were made by dishonest signers. The protocol works is as follows:

1. T checks that all promises and signatures are valid, and promises from above and below are consistent (for details, see Table 3. If any of the checks fail, T ignores the request.
2. If there has been no previous query to T on m, i.e. `validated`(m) is false, it derives a signed contract by converting all the promises contained in the resolve request to universally-verifiable signatures. T puts the signed contract in its database, sends it back in reply to the request, and sets `validated`(m) to true.
3. If there has been a positive resolution before, i.e. `validated`(m) is true, T sends back the stored signed contract.
4. If there has been an abort, T replies with an abort token or overturns its previous abort decision if it deduces that all the previous requests were made by dishonest signers. T deduces that P_j is dishonest from P_i's resolve request if: P_i presents to T a promise made by P_j such which shows that P_j continued the protocol after making a request to T.

The protocol is defined formally in Table 3.

4 Properties of the Protocol

Our protocol respects *timeliness*, since all signers can choose to stop waiting (quit, request abort or resolve) at any time they are waiting to receive a message. In order to prove fairness, we need the following lemmas.

Table 3. Recovery protocol for T

The first time T is contacted for contract m (either abort or recovery), T initialises $S(m)$ to $\emptyset$ and `validated`(m) to false.

If the recovery message $S_{P_i}(\{PCS_{P_j}((m,\tau_j),P_i,T)\}_{j\in\{1,\ldots,n\}\setminus\{i\}}, S_{P_i}(m,0))$ is received
  Check that promises and signature are valid, and promises from above and below are consistent, i.e.:
    for all $j<i$, check that $\tau_j=\tau_{i-1}$
    for all $j>i$, check that $\tau_j=\tau_{i+1}$
    check that $\tau_{i-1}=\tau_{i+1}$ or $\tau_{i-1}=\tau_{i+1}+1$

  if $i\in S(m)$ or one of the above checks failed then
    ignore the message

  else if $S(m)=\emptyset$ then
    `validated`(m) := true
    Send $\{S_{P_j}(m,\tau_j)\}_{j\in\{1,\ldots,n\}\setminus\{i\}}$ to P_i

  else if `validated`(m) then
    Send $\{S_{P_j}(m,\tau_j)\}_{j\in\{1,\ldots,n\}\setminus\{i\}}$ to P_i
      where τ_j is the level of the promise from P_j that was converted to a universally-verifiable signature.

  else // note that `validated`(m)=false $\wedge$ $S(m)\neq\emptyset$
    if $\exists p\in S(m)$ $((p<i\wedge\tau_p\leq h_p(m))\vee(p>i\wedge\tau_p\leq l_p(m)))$ then
      Send the stored abort token $S_T(S_{P_j}(m,P_j,(P_1,\ldots,P_n),abort))$ to P_i
      $S(m):=S(m)\cup\{i\}$
      Compute $h_i(m)$ and $l_i(m)$ as follows:
      if $i=1$
        // P_1 has contacted T in some step 7 of the main protocol
        $(h_i(m),l_i(m))=(\tau_2+1,0)$
      else if $i=n$
        // P_n has contacted T in some step 5 of the main protocol
        $(h_i(m),l_i(m))=(0,\tau_{n-1})$
      else if $1<i<n$ and $\tau_{i+1}=\tau_{i-1}$
        // P_i has contacted T in some step 5 of the main protocol
        $(h_i(m),l_i(m))=(\tau_{i+1},\tau_{i+1})$
      else if $1<i<n$ and $\tau_{i-1}>\tau_{i+1}$
        // P_i has contacted T in some step 7 of the main protocol
        $(h_i(m),l_i(m))=(\tau_{i+1}+1,\tau_{i+1})$
    else
      Convert the promises into signatures $\{S_{P_j}(m,\tau_j)\}_{j\in\{1,\ldots,n\}\setminus\{i\}}$
      Store the signatures
      Send the signatures to P_i
      `validated`(m) := true

Lemma 1. *If a* resolve *request in round* $r > 1$ *results in an abort decision, then:*

1. *for all* r' *such that* $1 < r' < r$ *there are two* resolve *requests in round* r' *that resulted in an abort decision.*
2. *there is an abort request in round 1.*

Proof. 1. We define the following predicates:

$A(r)$**:** there exists a resolve request in round r from some signer P_i that results in an abort decision. P_i's request has $r-1$ level promises from all other signers. We call such requests "type A".

$B(r)$**:** there exists a resolve request in round r from some signer P_i that results in an abort decision. P_i's request has r level promises from signers P_j, where $j < i$ and $r-1$ level promises from P_j where $j > i$. We call such requests "type B".

Point 1 of the lemma states that if $r > 1$ then $A(r) \vee B(r) \rightarrow \forall r'.(1 < r' < r \rightarrow A(r') \wedge B(r'))$. We show this by proving the following: (a) $A(r) \wedge r > 2 \rightarrow B(r-1)$; (b) $B(r) \wedge r > 1 \rightarrow A(r)$.

To show (a): Suppose $A(r) \wedge r > 2$. Let P_i be the signer whose request results in abort. P_i's request has $r-1$ level promises from all other signers. So, there has been a resolve request made by some signer P_k in round $r-1$ (otherwise according to T's protocol any previous abort would be overturned). Moreover, k can be chosen to be less than i, since according to T's protocol, if all such k were greater than i, than P_i's request would have resulted in resolve. Therefore, P_k's resolve request contains $r-1$ level promises from below and $r-2$ level promises from above, since if it had only $r-2$ level promises then P_i's request would overturn the abort received by P_k. Therefore, P_k's request shows $B(r-1)$.

For (b): Suppose $B(r)$ and $r-1$. Let P_i be the signer whose request results in abort. P_i's request has r-level promises from below and $r-1$-level promises from above. Since P_i's request results in abort, there has been a resolve request made by some other signer in round $r' \leq r$. To see this, suppose that the highest r' for which there is a resolve request by a signer P_k other than P_i resulting in abort is less than r.

- if P_k's request is type B, then T sets $h_k(m) = r-1$, $l_k(m) = r-2$.
  - if $k < i$, then P_i's request has an r-level promise from P_k, contradicting $h_k(m) = r-1$. So T overturns P_k's abort.
  - if $k > i$, then P_i's request has an $r-1$-level promise from P_k, contradicting $l_k(m) = r-2$. Again, T overturns P_k's abort.
- if P_k's request is type A, then T sets $h_k(m) = l_k(m) = r-2$. P_i's request has an $r-1$-level promise from P_k contradicting $h_k(m)$ or $l_k(m)$ as above. So T overturns P_k's abort.

Thus, in all cases, the assumption $r' < r$ leads to contradiction; and therefore $r' = r$. P_k's request proves $A(r)$.

2. If there is no abort in round 1, then according to T's protocol, any request by any participant in a later round will result in resolve.

□

Lemma 2. *If T issues abort to P_i in a round $r > 1$ and then later resolve to P_j, then P_i is dishonest.*

Proof. Suppose P_i gets abort at round $r > 1$. The variables h_i and l_i are set according to T's Recovery protocol. We verify that h_i is the highest level promise P_i could have sent to any signer above, and similarly, $l_i(m)$ is the highest level of promise P_i could have sent to a signer below. There are four cases to consider:

- $i = 1$. Then $h_i = \tau_2 + 1 = \cdots = \tau_n + 1$ since P_1 sends out $\tau + 1$-level promises after receiving all τ-level promises, and $l_i = 0$ because P_1 doesn't send any promises to below.
- $i = n$. Then $h_i = 0$ since P_n doesn't send any promises to above, and $l_i = \tau_{n-1} = \cdots = \tau_1$ since P_n has received τ-level promises from everyone before he sends out any τ-level promises.
- $1 < i < n$ and all the τ_k's are equal. P_i has requested resolve while waiting for promises from below, and the evidence it sends are the promises it got in the previous round, which is now complete and it has sent out its promises in that round too. Therefore $h_i = l_i = \tau_k$ for all k.
- $1 < i < n$ and $\tau_1 = \cdots = \tau_{i-1} \neq \tau_{i+1} = \cdots = \tau_n$. Here, P_i's request for resolve is while waiting for promises from above, and its evidence consists of promises it received in two different rounds. The promises it has sent to signers above are $\tau_{i+1} + 1$-level promises, and to below they are τ_{i+1}-level promises, so h_i and l_i are set accordingly.

Now P_j asks for resolve with a request that contains $PCS_{P_i}((m, \tau_i'), P_j, T)$. Since this request does not result in abort, the conditions for abort (which begin "$\exists p$" in Table 3) must fail. Therefore, for all p, $(p < j \rightarrow \tau_p' > h_p) \vee (p > j \rightarrow \tau_p' > l_p)$. Take $p = i$ and we obtain $i < j \wedge \tau_i' > h_i$ or $i > j \wedge \tau_i' > l_i$; each case includes evidence that P_i continued the protocol since its request to T and is therefore dishonest. □

Theorem 1. *The optimistic multi-party contract signing protocol above is fair.*

Proof. Assume P_i is an honest signer participating in the protocol to sign a contract m. Suppose P_i executed the protocol and decided failed, and some signer P_j decided signed. Then P_j has P_i's signature on m, because either: (1) P_i sent it in the last round of the main protocol; or, (2) T converted P_i's promise to P_j into a signed contract for P_j. We consider the two cases in turn.

1. Suppose P_i executed the last round of the protocol and sent out its signature on m. Then $i < n$ since P_n does not send out his signature until he has received everyone else's. Thus, P_i requested resolve from T in the last round with the request

$$S_{P_i}(\{PCS_{P_j}((m, \lceil n/2 \rceil + 1), P_i, T)\}_{j \in \{1,\ldots,i-1\}},$$

$$\{PCS_{P_j}((m, \lceil n/2 \rceil), P_i, T)\}_{j \in \{i+1,\ldots,n\}}, S_{P_i}(m))$$

and received abort. Since $i < n$ and P_i gets abort in the last round, T has evidence to overturn any abort issued in any previous round. Since T does not overturn all previous aborts, there is an abort given to P_k with $k > i$ in the last round. Thus P_i and P_k got abort in the final round ($\lceil n/2 \rceil + 1$). By lemma 1, rounds 2 to $\lceil n/2 \rceil$ have two failed resolve requests and round 1 has an abort request. The total number of requests is thus $2+(\lceil n/2 \rceil - 1)\times 2+1 = 2\lceil n/2 \rceil + 1$. This is at least $n+1$, but there are only n signers and each signer can make at most one request: a contradiction.

2. Suppose T returned a signed contract in response to a resolve request from P_j. There are three cases to consider:
   - If P_i quit the protocol in round 1, T could not have returned a signed contract, since P_i did not release any promises.
   - If P_i requested abort in round 1 from T, then it could have sent 1-level promises to signers above. Hence, T sets $h_i(m) = 1$ and, since P_i is honest, it does not release further promises. According to T's protocol, T could not have returned a signed contract, since any subsequent resolve request would only have $PCS_{P_i}((m, 1), P_k, T)$, where $k > i$.
   - If P_i received an abort decision for its resolve request in some round $1 < r \leq \lceil n/2 \rceil + 1$, and then P_i's promise to P_j got converted to a signature, then by lemma 2 P_i is dishonest.

   In all three cases we reach a contradiction. □

Abuse-freeness. Intuitively, the protocol is abuse-free, because of the use of private-contract signatures. No party has publicly verifiable information about P_i's commitment to the contract until a point from which P_i has the power to acquire a signed contract from all the other participants. (In future work, we intend to investigate our protocol in terms of formal definitions of abuse-freeness, such as that of [10]).

Timeliness. Our protocol also satisfies timeliness, since a participant can give up waiting for a message at any time and take recourse with the trusted party.

Remarks. Our protocol above works for up to $n-1$ dishonest signers. It can be optimized in the same way as it was done by Baum-Waidner and Waidner [5]: if the number of dishonest signers t is less and is known advance to all honest signers, then we can reduce the number of messages for the Main protocol. For Baum-Waidner, it results in $(t + 2)n(n - 1)$ messages; in our case it is $(\lceil (t + 1)/2 \rceil + 1)n(n - 1)$.

The number of messages of the "optimistic" execution can also be reduced if we allow signers to forward other signers' messages. In particular, a signer P_i instead of broadcasting its promise to all signers above, can now send those messages to P_{i+1}, who will then send P_i's promises intended for other signers (together with his) to P_{i+2}, and so on. Similarly, the same changes are applied when P_i sends promises to signers below. As a result, the number of messages sent in the "optimistic" execution is now $(\lceil n/2 \rceil + 1)2(n - 1)$.

Garay and MacKenzie [9] state that any complete and fair optimistic contract-signing protocol with n participants requires at least n rounds in an optimistic

run. Our result appears to contradict that statement, but it is not clear since they did not define what a round is. Different protocols group messages into rounds in different ways, so the only meaningful comparison is by number of messages in the optimistic execution.

5 Conclusions

We have presented a new multi-party contract signing protocol which uses private contract signatures. The previous multi-party contract signing protocol based on private contract signatures by Garay and MacKenzie [9] has been shown to be incorrect [6,12].

Our scheme improves on the state-of-the-art protocol by Baum-Waidner and Waidner [5] in two important aspects. Firstly, our schemce requires only half the number of messages to complete "optimistic" execution . (In contrast with Baum-Waidner's improvement reported in [4], we do not require the unrealistic assumptions that the number of dishonest signers is known in advance to the trusted party, and that honest signers don't quit the protocol.) Secondly, our scheme does not use a non-standard notion of a signed contract.

References

1. Asokan, N., Schunter, M., Waidner, M.: Optimistic protocols for multi-party fair exchange. Research Report RZ 2892 (# 90840), IBM Research (December 1996)
2. Asokan, N., Schunter, M., Waidner, M.: Optimistic protocols for fair exchange. In: Matsumoto, T. (ed.) 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, pp. 8–17. ACM Press, New York (1997)
3. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 591–606. Springer, Heidelberg (1998)
4. Baum-Waidner, B.: Optimistic asynchronous multi-party contract signing with reduced number of rounds. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 898–911. Springer, Heidelberg (2001)
5. Baum-Waidner, B., Waidner, M.: Round-optimal and abuse free optimistic multi-party contract signing. In: Montanari, U., Rolim, J.D.P., Welzl, E. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 524–535. Springer, Heidelberg (2000)
6. Chadha, R., Kremer, S., Scedrov, A.: Formal analysis of multi-party fair exchange protocols. In: Focardi, R. (ed.) 17th IEEE Computer Security Foundations Workshop, Asilomar, CA, USA, pp. 266–279. IEEE Computer Society Press, Los Alamitos (2004)
7. Even, S., Yacobi, Y.: Relations among public key signature systems. Technical report, Technion, Haifa (March 1980)
8. Garay, J.A., Jakobsson, M., MacKenzie, P.D.: Abuse-free optimistic contract signing. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 449–466. Springer, Heidelberg (1999)
9. Garay, J.A., MacKenzie, P.D.: Abuse-free multi-party contract signing. In: Jayanti, P. (ed.) DISC 1999. LNCS, vol. 1693, pp. 151–165. Springer, Heidelberg (1999)

10. Kähler, D., Küsters, R., Wilke, T.: A Dolev-Yao-based Definition of Abuse-free Protocols. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 95–106. Springer, Heidelberg (2006)
11. Micali, S.: Certified E-mail with invisible post offices. Available from author; an invited presentation at the RSA 1997 conference (1997)
12. Mukhamedov, A., Ryan, M.D.: Resolve-impossibility for a contract-signing protocol. In: CSFW 2006. 19th Computer Security Foundations Workshop, IEEE Computer Society Press, Los Alamitos (2006)
13. Pfitzmann, B., Schunter, M., Waidner, M.: Optimal efficiency of optimistic contract signing. In: Seventeenth Annual ACM Symposium on Principles of Distributed Computing, pp. 113–122. ACM Press, New York (1998)