Fair Multi-party Contract Signing using Private Contract Signatures

Aybek Mukhamedov Mark D. Ryan

School of Computer Science
The University of Birmingham

{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. Garay and Mackenzie (GM) proposed such protocol based on private contract signatures, but it was later shown to be flawed by Chadha, Kremer and Scedrov (CKS); the authors CKS also provided a fix to the GM protocol by revising one of its sub-protocols.

We show an attack on the revised GM protocol for any number $n > 4$ of signers. Furthermore, we argue that our attack shows that the message exchange structure of GM's main protocol is flawed: whatever the trusted party does will result in unfairness for some signer. This means that it is impossible to define a trusted party protocol for Garay and MacKenzie's main protocol; we call this "resolve-impossibility".

We propose a new optimistic multi-party contract signing protocol, also based on private contract signatures. We present a proof that our protocol satisfies fairness as well as its formal analysis in NuSMV model checker for the case of five signers. The protocol requires $n(n-1)(\lceil n/2 \rceil + 1)$ messages to be sent in the optimistic execution, which is about half the number of messages required by the state-of-the-art Baum-Waidner and Waidner protocol, and in contrast with Baum-Waidner and Waidner, it does not use a non-standard notion of a signed contract.

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

Preprint submitted to Elsevier *11 May 2007*

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 [9]. 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 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* [10] 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 [2,3,19]. Two-party protocols for *asynchronous* networks (where messages may be delayed arbitrarily) have been proposed in [4,10,19]. Later, two protocols for the n-party case were proposed: one by Garay and MacKenzie [11] and the other one by Baum-Waidner and Waidner [6]. Both of them consist of a main ("optimistic") part which does not involve the trusted party, together with subprotocols involving a trusted party used in the case that the signers do not receive expected messages.

Garay and MacKenzie's protocol (which we call GM) allows an arbitrary number $n \geq 2$ of signers to exchange signed contracts. A feature of GM is its use of *private contract signatures* to guarantee abuse-freeness. GM was shown by Chadha, Kremer and Scedrov [7] to fail the fairness property for the case $n \geq 4$. Those authors presented a revised version of the trusted party subprotocol. They verified the original main protocol together with the revised trusted party subprotocol for $n \leq 4$.

Baum-Waidner and Waidner's protocol (BW) was also analysed in [7], and was not found to be flawed. It 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-Waidner 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[5] 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 show that the revised GM protocol presented in [7] also fails the fairness property, for the cases $n \geq 5$. Furthermore, we argue that our attack shows that the protocol can not guarantee fairness for any $n \geq 5$ whatever the trusted authority T does, i.e. we show that no trusted party protocol is possible in order to fix the unfairness and the very idea behind Garay and MacKenzie's main protocol is flawed. A preliminary version of this work has been published as [17].

Next, we propose a new optimistic multi-party contract signing protocol based on private contract signatures that employs the ideas of Chadha, Kremer and Scedrov for the trusted party. In contrast with the state-of-art Baum Waidner and Waidner protocol, our protocol does not use a non-standard notion of a signed contract and achieves improvement in the message complexity of the optimistic 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 [6]. For example, if $n = 6$ our protocol requires 120 messages to "optimistically" sign a contract, whereas the previous scheme requires 210. We also present a proof that our protocol satisfies fairness, and give its formal analysis in NuSMV model checker for the case of five signers. An early version of some of this work was presented in [16].

Outline. Section 2 introduces the basic notions and assumptions about contract signing. In the following two sections we describe the revised GM protocol, and our analysis of it, including the impossibility of providing a resolve protocol. Next, in section 5, we describe our protocol followed by a proof that our protocol satisfies fairness in section 6 and its formal analysis in NuSMV model checker for the case of five signers in section 7. We conclude in section 8.

2 Preliminaries

Let $P_1, \ldots, P_n$ denote signers, who want to sign a contract m and T a trusted third party. Signers may adhere to the protocol, or they may be *dishonest*, i.e. deviate from the protocol. We assume that up to $n - 1$ of signers may be dishonest and are coordinated by a single party, an *adversary*. We assume that signers and their public keys are fixed in advance, all contracts are distinct and include the ordered list of all signers. $S_{P_i}(x)$ denotes P_i's universally-verifiable signature on x.

We shall say that P_i has a *valid* contract m from P_j if it receives P_j's signature on m or on (m, i) for some integer i. When P_i runs a contract signing protocol and acquires a valid contract m, we say "P_i decides 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.

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 Resolve 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 Resolve 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 and reply with a signed contract in order to maintain fairness. Lastly, once a signer receives a reply from T to its abort or resolve request, it quits the protocol. Therefore, a fair protocol must guarantee that 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 a valid contract from P_i, then P_i can obtain a valid contract from P_k 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 or successfully complete the contract signing.*

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

PCS promises.

The protocols in this paper employ a cryptographic primitive known as *private contract signature* [10]. 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 also 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 or P_j.
(3) $PCS_{P_i}(m, P_j, T)$ can be converted into a universally-verifiable signature $S_{P_i}(m)$ 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. The trusted party T has the power to convert a promise by some P to sign m to a proper signature by P on m.

A PCS promise may be *trusted party invisible*, where it is impossible for any recipient of $S_{P_i}(m)$ to distinguish whether it was converted from $PCS_{P_i}(m, P_j, T)$ by P_i or by the trusted party T; it may also be *trusted party accountable* where such distinguishability is possible.

3 GM Protocol revised by [7]

The protocol is an optimistic protocol, and therefore consists of two subprotocols, called Main and Abort/Resolve. Usually signers try to achieve the exchange by executing Main.

Main protocol. The main protocol for n participants is divided into n levels. For each level, a different strength of promise is used. An i-level promise from A to B is implemented as $PCS_A((m,i),B,T)$. Intuitively, the higher i is, the more A is committed to signing m, and hence, the stronger the promise is. We use $S_P(m)$ to denote the message m signed by P. The protocol for P_i $(1 \leq i \leq n)$ is described in Table 1.

(Revised) Abort and Resolve protocols. The original Abort and Resolve protocols [11] were shown to be flawed by Chadha, Kremer and Scedrov[7]. Those authors also proposed revised versions of the Abort and Resolve protocols, which they analysed and showed to be error-free for values of $n \leq 4$. We recall their revised versions here.

When P_i requests resolve from T, it sends evidence to T which consists of promises at various levels from the other participants. This evidence is designed so that T can infer the promises that an *honest* participant would have sent when it launched the resolve protocol (note that a participant may have dishonestly sent other promises). When P_i requests resolve, it sends the message

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

where τ_j is computed as following:

(1) If P_i runs the resolve protocol in step 5 of the main protocol (see table 1), then $\tau_j = 1$ for $j > i$ and $\tau_j = i - 1$ for $j < i$.
(2) In step 6.2 of the main protocol, $\tau_j = a - 1$ for $1 \leq j \leq a-1, j \neq i$ and $\tau_j = 1$ for $j > a - 1$.
(3) In step 6.4 of the main protocol, $\tau_j = a - 1$ for $j < i$, $\tau_j = a$ for $i < j \leq a$ and $\tau_j = 1$ for $j > a$.
(4) In step 7 of the main protocol, $\tau_j = n$ for all j.
(5) In step 9 of the main protocol, $\tau_j = n$ for all $j < i$ and $\tau_j = n+1$ for all $j > i$.

T maintains the set $S(m)$ of indices of participants that contacted T in the past and received an abort token. For each participant P_i in the set $S(m)$, T also maintains two integer variables $h_i(m)$ and $l_i(m)$ that it calculates on the basis of the promises that P_i provides in its resolve request. Intuitively, h_i corresponds to the highest level promise an honest P_i could have sent to any higher indexed participant before it contacted T. l_i corresponds the highest level promise an honest P_i could have sent to a lower indexed participant before it contacted T. The protocol for T works as follows:

- If T ever replies with a signed contract for m, then T responds with the contract for any further request from any participant.

Table 1 GM multi-party contract-signing protocol—Main for P_i

Wait for all higher recursive levels to start

1. $P_j \rightarrow P_i$: $PCS_{P_j}((m,1),P_i,T)$ $(n \geq j > i)$

   If P_i does not receive 1-level promises from $P_n \ldots P_{i+1}$ in a timely manner, then P_i simply quits.

Start recursive level i

2. $P_i \rightarrow P_j$: $PCS_{P_i}((m,1),P_j,T)$ $(i > j \geq 1)$

Wait for recursive level i-1 to finish

3. $P_j \rightarrow P_i$: $PCS_{P_j}((m,i-1),P_i,T)$ $(i > j \geq 1)$

   If P_i does not receive (i-1)-level promises from $P_{i-1} \ldots P_1$ in a timely manner, then P_i requests abort.

Send i-level promises to all lower-numbered signers

4. $P_i \rightarrow P_j$: $PCS_{P_i}((m,i),P_j,T)$ $(i > j \geq 1)$

Finish recursive level i when i-level promises are received

5. $P_j \rightarrow P_i$: $PCS_{P_j}((m,i),P_i,T)$ $(i > j \geq 1)$

   If P_i does not receive i-level promises from $P_{i-1} \ldots P_1$ in a timely manner, then P_i requests resolve.

Complete all higher recursive levels

For $a = i+1$ to n, P_i does the following:

> 6.1. $P_i \rightarrow P_a$: $PCS_{P_i}((m,a-1),P_a,T)$
>
> 6.2. $P_j \rightarrow P_i$: $PCS_{P_j}((m,a),P_i,T)$ $(a \geq j > i)$
>
> If P_i does not receive a-level promises from $P_a \ldots P_{i+1}$ in a timely manner, then P_i requests resolve.
>
> 6.3. $P_i \rightarrow P_j$: $PCS_{P_i}((m,a),P_j,T)$ $(i > j \geq 1)$
>
> 6.4. $P_j \rightarrow P_i$: $PCS_{P_j}((m,a),P_i,T)$ $(i > j \geq 1)$
>
> If P_i does not receive a-level promises from $P_{i-1} \ldots P_1$ in a timely manner, then P_i requests resolve.
>
> 6.5. $P_i \rightarrow P_j$: $PCS_{P_i}((m,a),P_j,T)$ $(a \geq j > i)$

Wait for signatures and (n+1)-level promises from higher-numbered signers

7. $P_j \rightarrow P_i$: $PCS_{P_j}((m,n+1),P_i,T), S_{P_j}(m,1)$ $(n \geq j > i)$

   If P_i does not receive signatures and (n+1)-level promises from $P_n \ldots P_{i+1}$ in a timely manner, then P_i requests resolve.

Send signatures and (n+1)-level promises to signers

8. $P_i \rightarrow P_j$: $PCS_{P_i}((m,n+1),P_j,T), S_{P_i}(m,1)$ $(j \neq i)$

Wait for signatures from lower-numbered signers

9. $P_j \rightarrow P_i$: $PCS_{P_j}((m,n+1),P_i,T), S_{P_j}(m,1)$ $(i > j \geq 1)$

   If P_i does not receive signatures and (n+1)-level promises from $P_{i-1} \ldots P_1$ in a timely manner, then P_i requests resolve.

- If the first request to T is a resolve request, then T sends back a signed contract.
- If the first request is an abort request, then T aborts the contract. T may overturn this decision in the future if it can deduce that all the participants in $S(m)$ have behaved dishonestly. T deduces that a participant P_i in $S(m)$

is dishonest when contacted by P_j if

(1) $j > i$ and P_j presents to T a k-level promise from P_i such that $k > h_i(m)$, or

(2) $j < i$ and P_j presents to T a k-level promise from P_i such that $k > l_i(m)$.

Abort and Resolve are described in detail in tables 2 and 3.

Table 2 Revised GM multi-party contract-signing protocol—Abort for P_i

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

```
1. $P_i \rightarrow$ T: $S_{P_i}(m, P_i, (P_1, \ldots, P_n), abort)$
   if not validated($m$) then
      if $S(m) = \emptyset$ then $T$ stores $S_T(S_{P_i}(m, P_i, (P_1, \ldots, P_n), abort))$
      $S(m) = S(m) \cup \{i\}$
      $l_i = 1$
      2. $T \rightarrow P_i$: $S_T(S_{P_j}(m, P_j, (P_1, \ldots, P_n), abort))$
   else (validated($m$)=true)
      3. $T \rightarrow P_i$: $\{S_{P_i}((m, \tau_i))\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$
      where $\tau_j$ is the level of the promise from $P_j$ that was converted to a
      universally-verifiable signature during the resolve protocol.
```

4 Analysis

Notation. $PCS^{\tau}_{i,j}$ is a shorthand for $PCS^{\tau}_{i}, PCS^{\tau}_{j}$. These are τth level promises (i.e. private contract signatures) on m issued by agents P_i and P_j to an agent whose identity is clear from the context. For example, when we say "P_1's request contains $PCS^{4}_{2,3}$", we mean that P_1's request contains 4th level promises issued by agents P_2 and P_3 to P_1. Of course, upon reception of a `resolve` request from P_i, T must check that all promises in it were issued to P_i.

4.1 An attack on fairness against the Revised GM protocol

We demonstrate an attack against *fairness* on the revised version of the protocol that involves five participants. Later, we generalise it to show that the protocol can not guarantee fairness for any $n \geq 5$ whatever the trusted authority T does. This shows that there is no Resolve sub-protocol for T that would fix the flaw, and thus, the structure of the message exchange in GM's

Table 3 Revised GM multi-party contract-signing protocol—Resolve

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

```
1. $P_i \rightarrow T$: $S_{P_i}(\{PCS_{P_j}((m,\tau_j),P_i,T)\}_{j\in\{1,\dots,n\}\setminus\{i\}}, S_{P_i}((m,1)))$
$T$ checks that the format of this message is one of the five permitted formats mentioned in the text.
if $i \in S(m)$ then
    $T$ ignores the message
else if $\texttt{validated}(m)$ then
    2. $T \rightarrow P_i$: $\{S_{P_j}((m,\tau_j))\}_{j\in\{1,\dots,n\}\setminus\{i\}}$
    where $\tau_j$ is the level of the promise from $P_j$ that was converted to a
    universally-verifiable signature.
else if $S(m) = \emptyset$ then
    $\texttt{validated}(m)$:=true
    3. $T \rightarrow P_i$: $\{S_{P_j}((m,\tau_j))\}_{j\in\{1,\dots,n\}\setminus\{i\}}$
else ($\texttt{validated}(m)$=false $\wedge$ $S(m) \neq \emptyset$)
    a) If there is some $p < i$ in $S(m)$ such that $\tau_p \leq h_p(m)$, or if there is some $p > i$
    in $S(m)$ such that $\tau_p \leq l_p(m)$, then $T$ sends back the stored abort
    $S_T(S_{P_j}(m, P_j, (P_1,\dots,P_n), abort))$ to $P_i$. $T$ adds $i$ to $S(m)$, and computes
    $h_i(m)$ and $l_i(m)$ as follows
    $(h_i(m), l_i(m))$= $(\tau_{i+1}, 0)$,      if $i = 1$ (intuitively, $P_1$ has contacted $T$ in either step 6.2 of
                                   the main protocol with $a = \tau_{i+1}+1$ or in step 7 of the main
                                   protocol),
                 $= (0, i)$,            if $1 < i$ and $\tau_{i-1} = i - 1$ (intuitively, $P_i$ has contacted $T$ in
                                   step 5 of the main protocol),
                 $= (\tau_{i-1}, \tau_{i-1})$,      if $1 < i < n, i \leq \tau_{i-1} < n$ and $\tau_{i+1} \leq \tau_{i-1}$ (intuitively, $P_i$ has
                                   contacted $T$ in step 6.2 of the main protocol with $a = \tau_{i-1}$+1),
                 $= (\tau_{i-1}, \tau_{i-1}+1)$,if $1 < i < n, i \leq \tau_{i-1} < n$ and $\tau_{i+1} > \tau_{i-1}$ (intuitively, $P_i$ has
                                   contacted $T$ in step 6.4 of the main protocol with $a = \tau_{i-1}$+1),
                 $= (n, n)$,            if $1 < i < n$ and $\tau_{i-1} = \tau_{i+1} = n$. (intuitively, $P_i$ has con-
                                   tacted $T$ in step 7 of the main protocol).
                 $= (n+1, n+1)$,      if $1 < i < n, \tau_{i-1} = n$ and $\tau_{i+1} = n+1$. (intuitively, $P_i$ has
                                   contacted $T$ in step 9 of the protocol).
                 $= (0, n+1)$,        if $i = n$ and $\tau_{i-1} = n$. (intuitively, $P_n$ has contacted $T$ in step
                                   9 of the main protocol).
    b) Otherwise, T sends $\{S_{P_j}((m,\tau_j))\}_{j\in\{1,\dots,n\}\setminus\{i\}}$ to $P_i$, stores all the
    signatures, and sets $\texttt{validated}(m)$ to true.
```

Main protocol is flawed. We call it a resolve-impossibility result for T and it is applicable to both the original and the revised versions of the protocol.

Suppose that agents $P_1, \dots, P_5$ decide to sign a contract m using the Revised GM protocol. They optimistically execute the Main sub-protocol up to a point, where P_4 sends its signature and 6th-level promise on m to all participants (step 9 in Table 1). Now suppose P_1 and P_3 do not send their signatures on m to P_4. Hence, according to the protocol P_4 sends a `resolve` request to T, but we suppose that it is delayed by the intruder long enough, until the following sequence of events is completed:

- P_5 requests `abort` from T. `Abort` is granted as no other request was made to T regarding m (i.e. $S(m) = \emptyset$), which results in $l_5 = 1$;
- P_1 requests `resolve` from T with a message that contains $PCS^4_{2,3,4}, PCS^1_5$. As P_5 was previously granted abort (i.e. $S(m) = \{P_5\}$) and $\tau_5 = l_5 = 1$, T does not overturn its previous `abort` decision, but sends `abort` to P_1 and sets $h_1 = 4$ (T presumes that P_1 requested `resolve` at the step 6.2 of the *main* sub-protocol).
- P_3 requests `resolve` from T with a message containing $PCS^4_{1,2}, PCS^5_{4,5}$ that also results in an `abort` reply, as $\tau_1 = h_1 = 4$ and $S(m) = \{P_5, P_1\}$. T sets $l_3 = 5$ (thinking that P_3 is at the step 6.4).
- P_2 sends a `resolve` request to T that has $PCS^5_{1,3,4,5}$. However, once more, T replies with `abort` since $\tau_3 = l_3 = 5$ and $P_3 \in S(m)$, and sets $h_2 = 5$ (T presumes that P_2 is in step 7).

  Note that although P_2 received an abort from T, he is not in an unfair state, since he did not send his signature on m to any signatory.

Now P_4's `resolve` request containing $PCS^5_{1,2,3}, PCS^6_5$ reaches T, but it results in an `abort` reply, since $\tau_2 = h_2 = 5$ and $P_2 \in S(m)$. Recall that P_4 has already sent his signature on m to all participants and P_1 and P_3 in particular. Therefore, this is an attack on fairness: P_1, P_3 and P_2 have P_4's signature on the contract m, whereas P_4 doesn't have any of theirs.

We describe our attack as *abort propagation*, since the attack is based on a legitimate abort being carried through a number of participants until it becomes an unfair abort. Although P_2 is dishonest in our attack, it is possible to re-order the events so that the same attack takes place but P_2 is honest. T receives evidence of the dishonesty only of P_5, P_1 and P_3 during the attack, but not of P_2. The resolve request of P_2 is justified because P_3 and P_5 would not send their signature and 6th level promises to P_2. The attack sequence could have happened before P_4 sent out his signature and 6th level promises to other signatories.

There are other instantiations of this idea. For example, for the five signers case P_3 could be left in unfair state if P_1, P_4 and P_5 group up together and act dishonestly in the similar way as in our attack.

4.2 Resolve-impossibility for T

We call the attack on fairness that we described above *abort chaining*: intuitively, malefactors group together to propagate T's *abort* decision. When an honest signatory sends out his signature on a contract, but does not receive signed contracts back, and then asks T to `resolve`, he receives an `abort` decision. This is not due to a fault in T's `abort` or `resolve` protocols – a closer examination of the attack reveals that T could not have overturned any previous `abort` decision when presented with the `resolve` requests, since the most recent agent he sent the `abort` to could have been honest. The following is a more rigorous explanation of this intuition.

4.2.1 Generalising the attack

We show that one can derive an attack against the protocol involving any $n \geq 5$ number of signers using the same pattern of attack as was shown for the five signers case. For example, if we have $n \geq 5$ signers, where P_1, P_3 and P_n are dishonest ones, and P_4 is the victim, then an attack can proceed in a similar way. The signers optimistically execute the *main* sub-protocol up to a point, where P_4 sends its signature and $(n+1)$th-level promise on m to all participants. Again, we suppose that P_1 and P_3 do not send their signatures on m to P_4, and therefore P_4 sends a `resolve` request to T. As before, we suppose this is delayed by the intruder long enough so that the following sequence is completed:

(1) P_n requests `abort` from T. `Abort` is granted as $S(m) = \emptyset$, and T sets $l_n = 1$.
(2) P_1 requests `resolve` from T with $PCS^{n-1}_{2,\ldots,n-1}, PCS^1_n$. Since $S(m) = \{P_n\}$ and $\tau_n = l_n = 1$, T sends `abort` to P_1 and sets $h_1 = n-1$.
(3) P_3 requests `resolve` from T with $PCS^{n-1}_{1,2}, PCS^n_{4,\ldots,n}$ that also results in an `abort` reply, as $\tau_1 = h_1 = n-1$ and $S(m) = \{P_5, P_1\}$. T sets $l_3 = n$.
(4) P_2 sends a `resolve` request to T that has $PCS^n_{1,\ldots,n}$. Once more, T replies with `abort` since $\tau_3 = l_3 = n$ and $P_3 \in S(m)$, and sets $h_2 = n$.

Now P_4's `resolve` request with $PCS^n_{1,2,3}, PCS^{n+1}_{5,\ldots n}$ reaches T, but it results in an `abort` reply, since $\tau_2 = h_2 = n$ and $P_2 \in S(m)$. As before, this is an attack on fairness: P_1, P_3 and P_2 have P_4's signature on the contract m, whereas P_4 doesn't have any of theirs. As before, a similar attack in which P_2 is honest can also be constructed.

4.2.2 Resolve impossibility

We show that there is no way to adapt the Abort and Resolve protocols to fix this problem. More formally, we prove that for all protocols for the trusted party T, there exists an execution for the attacker which makes the protocol unfair for an honest participant.

The proof proceeds as follows. Suppose T is running according to a protocol. Suppose as before that P_1, P_3 and P_n are dishonest and controlled by the attacker, and P_4 is honest. The attacker's strategy is the one described above. We show that, no matter what T does, it is unfair to someone who could be honest at the time of T's action.

Consider P_n's request for abort from T. Since P_n could have not received $(n-1)$ level promises from some of the signers $P_1, \ldots, P_{n-1}$, T must determine that this request is legitimate, and grant the abort. Next, P_1 requests resolve from T with $PCS^{n-1}_{2,\ldots,n-1}, PCS^1_n$. T determines that this request is valid, as P_1 might not have received the nth level promises from some of the signers $P_2, \ldots, P_n$.

- If T resolves, thereby overturning its previous abort decision, this is unfair to P_n, since T has no evidence of any dishonesty of P_n, who could have halted after its abort request to T.
- If T aborts, then we suppose that P_3 requests resolve from T with $PCS^{n-1}_{1,2}$, $PCS^n_{4,\ldots,n}$. T determines that this request is valid, since P_3 may not have received nth level promise from P_1 or P_2 (or both). (At this point T has evidence that P_n dishonestly continued the protocol after requesting abort.)
    - · If T resolves, thereby overturning its two previous abort decisions, this is unfair to P_1, since T has no evidence of any dishonesty of P_1.
    - · If T aborts, then we suppose P_2 sends a resolve request to T that has $PCS^n_{1,\ldots,n}$. T determines that this is a valid request, as P_3 and P_n might not have sent their signature and $(n+1)$-level promises to P_2. (Now T has evidence that P_1 dishonestly continued the protocol after requesting abort.)
        - - If T resolves, thereby overturning its three previous abort decisions, this is unfair to P_3, since T has no evidence of any dishonesty of P_3.
        - - If T aborts, then we suppose that the intruder allows P_4's resolve request to arrive at T.
            - + If T resolves, thereby overturning its four previous abort decisions, this is unfair to P_2, since T has no evidence of any dishonesty of P_2.
            - + If T aborts, this is unfair to P_4, who has honestly sent out his signature.

Thus, no matter what T does, it is unfair to someone who could be honest at the time T takes the decision. The flaw lies in the main protocol and there is no resolve protocol for T that would fix it.

5 Our protocol

In this section we introduce a new protocol, also based on PCs. It also consists of three sub-protocols. 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.

5.1 Main protocol for signer P_i

Our main protocol has a cascading structure for exchanging promises among signers. Intuitively, that allows to commit signers to certain stages of the protocol depending on the promises they send out. For example, for the case of five signers (see Figure 1), when signer P_3 sends out his 2-level promises to signers P_4, P_5, that would imply that P_3 must have received promises from P_1, P_2 in round 2 and promises from P_4, P_5 in round 1. If the trusted party T is requested to restore fairness, it uses such information to deduce dishonest signers who deviated from the protocol. Also, the total number of rounds in the main protocol $\lceil n/2 \rceil + 1$ is chosen such that even if all but one signers are dishonest and collude, they will not be able to propagate abort decision into the last round of the protocol when signatures are released. That is, the total number of rounds outnumbers the total number of requests a coalition of $n-1$ dishonest signers can make in order to propagate trusted party's abort decision.

The protocol begins with each signer waiting 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 5.1. Finally, in the last round, $\lceil n/2 \rceil + 1$-level promises and signatures are exchanged. The protocol is defined formally in Table 4.

If expected messages are not received, a participant P_i may simply quit the protocol, or request abort or $\mathsf{resolve}$ 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 $\mathsf{resolve}$ 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.) The resolve request that P_i sends to T includes $S_{P_i}(m, 0)$ instead of $S_{P_i}(m)$ to simplify a clause in Table 6.

The format of the resolve requests is chosen in such a way that the requesting signer P_i can attest its position in a run of the protocol to the trusted party, as well as provide evidence to T of how far other signers are engaged in the protocol's run. Also, if T replies with an abort token to P_i's resolve request, T infers (and stores) the highest levels of promises that (honest) P_i could have sent to signers below and above from the promises in P_i's resolve request.

Remarks. In our protocol it is possible for any recipient of a signed contract to deduce from its structure whether it was produced by a signer or it was enforced by T. When it is undesirable for an "outsider" to know whether the signed contract was produced optimistically or with the help of the trusted party, the following simple changes can be introduced in our protocol: use trusted party invisible PCS promises; replace the signatures $S_{P_i}(m)$ exchanged in the last round of the Main protocol with $S_{P_i}(m, 1)$; and, add 1-level promises from all signers to all of P_i's resolve requests, so that if T decides to enforce m, it replies with $\{S_{P_i}(m, 1)\}_{j\in\{1,\ldots,n\}}$. With those changes, it is impossible for any party not involved in the protocol to tell whether $S_{P_i}(m, 1)$ was produced optimistically or with the help of the trusted party.

5.2 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` initiated 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 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 and received an abort token. This set is used when T considers whether to overturn its previous abort decision. For each signer P_i such that $i \in S(m)$, T also maintains two integer variables $h_i(m)$ and $l_i(m)$. T sets the values for those variables according to the levels of promises P_i supplies in his request: the promises in P_i's request

Table 4 Our 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.

attest to T how far P_i was involved in a run of the protocol, and from that information T notes the highest levels of promises that honest P_i could have sent to other signers when it stopped executing the Main protocol. 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 [7], even though it does not work for the protocol they consider [17].

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

5.2.1 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 resolved it sends back a signed contract; otherwise, it sends back an abort token (see table 5).

5.2.2 Resolve protocol

The resolve 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 6. 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` 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` to true.
(3) If there has been a positive resolution before, i.e. `validated` is true, T sends back the stored signed contract.

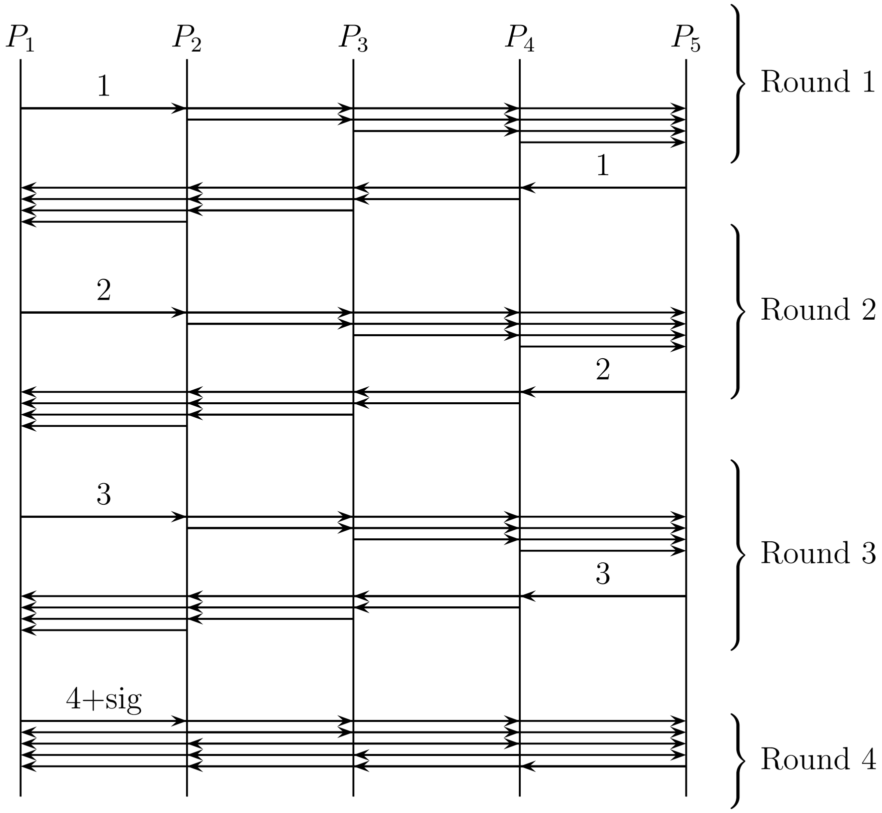

Fig. 1. Messages in our Main protocol when $n = 5$. In order to reduce clutter, each line in the figure represents a set of messages broadcast by a single signer to a set of signers. For example, the first line stands for 1-level promises sent by P_1 to all other signers.

(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 6.

Table 5 Our Abort protocol for T

```
The first time $T$ is contacted for contract $m$ (either abort or resolve), $T$ initialises $S(m)$ to $\emptyset$ and validated 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 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 $\tau_j$ is the level of the promise from $P_j$ that was converted to a
            universally-verifiable signature during the resolve protocol.
```

6 Properties of our 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 lemma.

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".

Table 6 Our Resolve protocol for T

```
The first time $T$ is contacted for contract $m$ (either abort or resolve), $T$ initialises $S(m)$ to $\emptyset$ and validated to false.
If the resolve 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 := true
        Send $\{S_{P_j}(m,\tau_j)\}_{j\in\{1,\ldots,n\}\setminus\{i\}}$ to $P_i$

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

    else // note that validated=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 $S_T(S_{P_j}(m, P_j, (P_1,\ldots,P_n), \mathit{abort}))$ 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 or 11 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 or 9 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 or 9 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 or 11 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 := true
```

$B(r)$**:** 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 Resolve 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 6) 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 [13]).

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 optimised in the same way as it was done by Baum-Waidner and Waidner [6]: 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 [11] 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.

7 Analysis using NuSMV

As an extra check, we have modelled our protocol using the NuSMV model checker [8] for the case of five signers, and proved fairness in that case. Our NuSMV model abstracts away from details of the cryptographic primitives and the Dolev-Yao attacker. Its purpose is to check for situations in which the trusted party fails to respond fairly to honest participants. This includes checking for abort-propagation attacks of the kind described in section 4.1. First, we model a situation in which all five signers have completed rounds 1, 2, 3 of the Main protocol – i.e. they have exchanged all 1, 2 and 3-level promises, but they have not yet sent out any 4-level promises or signatures (see Figure 5.1). No abort should be possible if all signers have got to this point, because the signers are about to release their signatures. In the previous protocol it was possible at this stage for P_1, P_2, P_3 and P_5 to gang up on P_4, by releasing a well-chosen sequence of requests to T and forcing T to issue an abort to P_4, resulting in unfairness. We show that such attacks are impossible in the new protocol. Next, we consider the situation in which an honest signer receives abort, and discontinues the protocol. We show that such aborts will not be overturned.

The NuSMV code models T's behaviour in response to a non-deterministically chosen sequence of requests. A request is modelled by the following parameters:

- `requester`, which may be any of the participants $\{1, 2, 3, 4, 5\}$;

- `reqType`, which may be *abort* or *resolve*;
- `reqTauLeft`, the values of τ_j for $j <$ `requester`;
- `reqTauSame`, a boolean indicating whether the values of τ_j for $j >$ `requester` are equal to or one less than `reqTauLeft`.

We derive `reqTauRight` from these parameters (it is either `reqTauLeft` or `reqTauLeft`-1, according to the value of `reqTauSame`). Note that if `reqType`$=$*abort* then the values of `reqTauLeft` and `reqTauRight` are irrelevant and are ignored, and if `requester`$=1$ (resp. `requester`$=5$) then `reqTauLeft` (resp. `reqTauRight`) is irrelevant and ignored.

T reacts to these queries by updating its state, which is modelled by variables as follows:

- Booleans `requested`i, for $i \in \{1, 2, 3, 4, 5\}$, indicating whether P_i has made a request in the past. The protocol stipulates that each P_i may make only one request, and repeat requests are ignored.
- A boolean `validated`, as required in the protocol.
- Booleans `S1`, `S2`, `S3`, `S4`, `S5`, which model the set S required in the protocol. The boolean `S`i indicates whether $i \in S$.
- Integers `h`i and `l`i, for $i \in \{1, 2, 3, 4, 5\}$, as required by the protocol.

NuSMV models these variables synchronously; that is, there is a global clock, and each tick of the clock causes all of the variables to be updated. In our model, a tick of the clock corresponds to a request being made to T. For each variable, the code contains a clause of the form

```
next(variable) :=
    case
        condition1 : expression1;
        condition2 : expression2;
        . . .
        condition n : expression n;
    esac;
```

This describes how the value of *variable* in the next state is determined. Each of the conditions is evaluated in turn until one of them evaluates to true; in that case, the corresponding expression is evaluated and the result is the next value of *variable*. Values 0 and 1 represent booleans false and true respectively.

Two conditions are required repeatedly in the code; they are given in the DEFINE section. The condition `duplicateRequest` describes whether the current request is from a requester who has already made a request, i.e. whether $\bigvee_{i=1}^{5}($`requester` $= i \wedge$ `requested`$i)$. If this is true, the request is ignored, i.e. no variable is updated in response to the request. The condition `confirmPrevAbort` denotes the condition $\exists p \in S(m)\ ((p < i \wedge \tau_p \leq$

$h_p(m)) \vee (p > i \wedge \tau_p \leq l_p(m)))$ present in the resolve protocol. It is relevant when a resolve request occurs in a situation that an abort has previously been granted, and it determines whether the previous abort will be confirmed or overturned.

Consider, for example, the code describing the evolution of `h1`:

```
 1  next(h1) := case
 2      duplicateRequest : h1;
 3      !(requester=1) : h1;
 4      reqType = abort & !validated : 1;
 5      reqType = abort : h1;
 6      S1 | S_empty | validated : h1;
 7      confirmPrevAbort : case
 8          requester=1 : reqTauRight + 1;
 9          requester=5 : 0;
10          reqTauLeft=reqTauRight : reqTauRight;
11          1 : reqTauRight+1;
12          esac;
13      1: h1;
14      esac;
```

Line 2 indicates that repeat requests are ignored (i.e. the value of `h1` doesn't change). Similarly, `h1` doesn't change if the requester is other than P_1. By the time we get to line 4, we know that the request is not a repeat request and it is from P_1. Line 4 says that if the request was an abort and the contract has not been validated, `h1` is set to 1; otherwise (line 5) if the request was abort, `h1` is left unchanged. Line 6 (which is reached only if the request is a resolve) says that if either `S1` holds (i.e. $i \in S$), or $S = \emptyset$, or `validated` is true, then `h1` is not updated. Here, we are literally following the pseudocode for the trusted party (Tables 5 and 6). In line 7, the condition `confirmPrevAbort` is evaluated, and if it is true, then a further case statement indicates how `h1` is to be updated. The full code is given at [15]. It may be noted that the code is not optimal. For example, in the fragment above, the inner case statement evaluates `requester`, even though we know that `requester` $= 1$ if we get as far as that case statement. This apparent inefficiency, due to the mechanical way the clauses were generated, has no real computational cost for the model checker because at compile time it reduces all these clauses to a normal form, called a binary decision diagram, which is independent of this kind of boolean redundancy.

The queries we put to NuSMV model fairness as two sub-properties:

(1) If an honest signer releases signatures to another signer and later makes a resolve request, the request will be granted. This property guarantees

that the kind of abort propagation attacks in section 4.1 are impossible for our protocol.

(2) If an honest signer receives an abort, then that abort will not be overturned by later requests.

For the first property, we suppose that a non-repeat resolve request from a signer in the last stage of the protocol (i.e. with $\tau_l = \tau_r = 3$) has been made; this must result in `validated` becoming true. This is written in Computational Tree Logic (CTL) as shown below; this formula evaluates to true.

```
SPEC AG (reqType=resolve & !duplicateRequest &
    reqTauLeft=3 & reqTauRight=3 -> AX validated)
```

For the second property, we consider all the possible circumstances in which a honest signer makes a request to T and receives an abort response. The fact that the signer is honest means that it doesn't continue to release promises after it contacts T. This means that certain requests to T from other signers are impossible (they need the later promises from the honest signer).

For example, suppose P_4 requests resolve with $\tau_l = \tau_r = 2$ and gets abort. If P_4 is honest, this means that P_1 cannot later request resolve with $\tau_r = 3$, and none of P_2, P_3, P_4 can request resolve with $\tau_l = \tau_r = 3$, and P_5 cannot request resolve with $\tau_l = 3$. The query for NuSMV is: under these hypotheses, could P_4's abort be overturned? We represent the query as follows: let p represent that P_4 requests resolve with $\tau_l = \tau_r = 2$, i.e.

```
p = requester=4 & !duplicateRequest &
    reqType=resolve & reqTauLeft=2 & reqTauRight=2
```

and let q represent the later requests made impossible by the assumption of P_4's honesty, i.e.

```
q = (requester=1 & reqTauRight=3) |
    (requester>=2 & requester <=4 &
        reqType=resolve & reqTauLeft=3 & reqTauRight=3) |
    (requester=5 & reqType=resolve & reqTauLeft=3)
```

We write v to abbreviate the variable `validated`. We now model the desired property as saying: whenever p occurs and results in abort, then if q remains false then v will remain false. This property is tricky to represent in CTL. In LTL it is much easier: we may write

$$\mathtt{G}\ (\ p\ \&\ \mathtt{X}\ !v \rightarrow\ !v\ \mathtt{W}\ q\ )$$

where `W` is *weak until*. NuSMV can check LTL formulas but it is much more efficient at checking CTL formulas. Using the techniques explained in Chapter 3 of [12], one can translate this LTL formula into the following equivalent CTL formula:

$$\texttt{AG(}p\texttt{ -> AX(!}v \rightarrow \texttt{!E [!}q\texttt{ U (!}q\texttt{ \& }v\texttt{)]))}$$

This formula evaluates to true. Different circumstances of possible abort overturns correspond to different values of p and q. Some examples are shown in the code available at [15].

8 Conclusion

We have shown that the revised version of Garay and Mackenzie's protocol presented in [7] fails the fairness property, for the cases $n \geq 5$. We also demonstrated that our attack shows that the protocol can not guarantee fairness for any $n \geq 5$ whatever the trusted authority T does. This means that no trusted party protocol is possible in order to fix the unfairness. The flaw lies in the structure of the message exchange of the Garay and Mackenzie's main protocol.

We also presented a new multi-party contract signing protocol which uses private contract signatures, and we proved that it satisfies fairness and formally analysed in NuSMV model checker for the case of five signers. It also satisfies informal notions of timeliness and abuse-freeness.

Our scheme improves on the state-of-the-art protocol by Baum-Waidner and Waidner [6] in two important aspects. Firstly, our scheme requires only half the number of messages to complete "optimistic" execution. (In contrast with Baum-Waidner's improvement reported in [5], 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.

In future work, we aim to supplement our proof with a formal verification the protocol in the case of arbitrary n against the claimed properties, using a framework such as applied pi calculus [1], strand spaces [20] or higher-order logic [18]. This is a challenging task since state-of-the-art formal analysis methods do not cope well with *open-ended* protocols (i.e. those that may involve arbitrary number of parties in a single run). As our work shows, analysis of such a protocol for a fixed number of signers may not be enough. We also intend to evaluate our protocol against formal definitions of abuse-freeness such as [13].

Acknowledgements. Many thanks to Rohit Chadha, Steve Kremer and Andre Scedrov for their helpful comments about the attack presented in section 2 of this paper; and also allowing us to use their Latex source of the description of their protocol, particularly the source for Tables 1, 2, 3. Thanks also to Steve Kremer for interesting conversations about the meaning of fairness.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In H. R. Nielson, editor, *Proceedings of the 28th ACM Symposium on Principles of Programming Languages*, pages 104–115, London, UK, Jan. 2001. ACM.

[2] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for multi-party fair exchange. Research Report RZ 2892 (# 90840), IBM Research, Dec. 1996.

[3] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In T. Matsumoto, editor, *4th ACM Conference on Computer and Communications Security*, pages 8–17, Zurich, Switzerland, Apr. 1997. ACM Press.

[4] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *Advances in Cryptology—Eurocrypt 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 591–606, 1998.

[5] B. Baum-Waidner. Optimistic asynchronous multi-party contract signing with reduced number of rounds. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12*, volume 2076 of *Lecture Notes in Computer Science*, pages 898–911. Springer, 2001.

[6] B. Baum-Waidner and M. Waidner. Round-optimal and abuse free optimistic multi-party contract signing. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15*, volume 1853 of *Lecture Notes in Computer Science*, pages 524–535. Springer, 2000.

[7] R. Chadha, S. Kremer, and A. Scedrov. Formal analysis of multi-party fair exchange protocols. In R. Focardi, editor, *17th IEEE Computer Security Foundations Workshop*, pages 266–279, Asilomar, CA, USA, June 2004. IEEE Computer Society Press.

[8] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[9] S. Even and Y. Yacobi. Relations among public key signature systems. Technical report, Technion, Haifa, March 1980.

[10] J. A. Garay, M. Jakobsson, and P. D. MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology—Crypto 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 449–466. Springer-Verlag, 1999.

[11] J. A. Garay and P. D. MacKenzie. Abuse-free multi-party contract signing. In P. Jayanti, editor, *International Symposium on Distributed Computing*, volume 1693 of *Lecture Notes in Computer Science*, pages 151–165, Bratislava, Slavak Republic, Sept. 1999. Springer-Verlag.

[12] M. R. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge University Press, 2nd edition, 2004.

[13] D. Kähler, R. Küsters, and T. Wilke. A Dolev-Yao-based Definition of Abuse-free Protocols. In *Proc. 33rd International Colloqium on Automata, Languages, and Programming (ICALP'06)*, volume 4052 of *Lecture Notes in Computer Science*, pages 95–106. Springer, 2006.

[14] S. Micali. Certified E-mail with invisible post offices. Available from author; an invited presentation at the RSA 1997 conference, 1997.

[15] A. Mukhamedov and M. Ryan. NuSMV code for five signers. From Mark Ryan's web page (currently `www.cs.bham.ac.uk/~mdr/`), follow the link for *Contract Signing.*

[16] A. Mukhamedov and M. Ryan. Improved multi-party contract signing. In *Proc. Financial Cryptography*, LNCS. Springer, 2007.

[17] A. Mukhamedov and M. D. Ryan. Resolve-impossibility for a contract-signing protocol. In *19th Computer Security Foundations Workshop (CSFW 2006).* IEEE Comp. Soc. Press, 2006.

[18] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Comput. Secur.*, 6(1-2):85–128, 1998.

[19] B. Pfitzmann, M. Schunter, and M. Waidner. Optimal efficiency of optimistic contract signing. In *Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, pages 113–122, New York, May 1998. ACM.

[20] F. J. Thayer Fabrega, J. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *1998 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, May 1998.