Resolve-Impossibility for a Contract-Signing Protocol[*]

Aybek Mukhamedov Mark D. Ryan
The University of Birmingham
{axm,mdr}@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. Such a protocol was introduced by Garay and MacKenzie in 1999; it consists of a main protocol and a sub-protocol involving a trusted party. Their protocol was shown to have a ¤aw by Chadha, Kremer and Scedrov in CSFW 2004. Those authors also presented a £x – a revised sub-protocol for the trusted party.

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

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 [6]. 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* [7] 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 £rst described for *synchronous* networks in [1, 2, 9, 10]. 2-party protocols for *asynchronous* networks, have been proposed in [3, 7, 10], 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 [8] and the other *round-optimal* one by Baum-Waidner and Waidner [4].

Garay and MacKenzie's protocol (which we call GM) allows an arbitrary number $n \geq 2$ of signers to exchange signed contracts. Their protocol consists of a main ("optimistic") part which does not involve the trusted party, together with a subprotocol used by the signers to contact the trusted party in the case that they do not receive expected messages. A feature of GM is its use of *private contract signatures* to guarantee abuse-freeness. GM was shown by Chadha, Kremer and Scedrov [5] to fail the fairness property for the case $n \geq 4$. Those authors presented a revised version of the trusted party subprotocol. They veri£ed the original main protocol together with the revised trusted party subprotocol for $n \leq 4$.

Our contribution. We show that the revised protocol presented in [5] also fails the fairness property, for the cases

[*] 19th IEEE *Computer Security Foundations Workshop* (CSFW), 2006.

$n \geq 5$. Furthermore, we generalise our attack to show 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 £x the unfairness and the very idea behind Garay and MacKenzie's main protocol is ¤awed .

Outline. In the next section we describe the Revised GM protocol, and introduce some notations for this paper. Next, in section 3, we show our analysis of the protocol, and conclude in section 4.

2. Revised GM Protocol

The protocol allows $n \geq 2$ participants, say $P_1, \ldots, P_n$ to exchange signatures on contract text m. The participants exchange messages with each other. If a participant does not receive an expected message, it can contact the trusted third party T, requesting to abort the protocol, or to recover it.

It is assumed that the ordering of the participants $P_1, \ldots, P_n$, the trusted party T, and the contract text m are all agreed in advance. The text m includes an identi£er that uniquely identi£es each protocol instance. Communication among the participants is assumed to take place on a network channel in control of a "Dolev-Yao" style of attacker. This attacker can intercept and inject messages, but cannot perform cryptographic operations such as encryption or signing unless he has the relevant keys. Communication with T is assumed to take place over a private channel. Messages can be delayed arbitrarily, but with messages sent between correct participants and the trusted third party guaranteed to be delivered eventually. In general, it is assumed that an adversary may schedule messages, and possibly insert its own messages into the network.

The protocol employs a cryptographic primitive known as *private contract signature* [7]. A private contract signature by A for B on text m with respect to trusted party T, denoted $PCS_A(m, B, T)$, is a cryptographic object with the following properties:

1. $PCS_A(m, B, T)$ can be created by A, and by B.
2. Each of A, B and T (but no-one else) can tell the difference between the versions created by A and by B.
3. $PCS_A(m, B, T)$ can be converted in to a regular universally-veri£able signature by A, and by T; and by no-one else.

The idea is that $PCS_A(m, B, T)$ acts as a promise by A to B to sign m. But B 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 A and one created by B.

The protocol consists of three subprotocols, called Main, Abort and Recovery. 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 token, 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.

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 $PCS_A((m, i), B, T)$. 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 Recovery protocols. The original Abort and Recovery protocols [8] were shown to be ¤awed by Chadha, Kremer and Scedrov[5]. Those authors also proposed revised versions of the Abort and Recovery 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 recovery 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 recovery protocol (note that a participant may have dishonestly sent other promises). When P_i requests recovery, 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 $j > a$.

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 £nish

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 aborts.

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 recovers.

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
$P_i$ recovers.
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
$P_i$ recovers.
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 recovers.

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 recovers.

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 $l_i(m)$. Intuitively, h_i is the highest level promise an honest P_i could have sent to any higher indexed participant before it contacted T. l_i is 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.

- If the £rst request to T is a resolve request, then T sends back a signed contract.

- If the £rst 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 Recovery are described in detail in tables 2 and 3.

2.1. Fairness and the attacker model

Fairness. A multi-party contract signing protocol is *fair for an honest participant* P_i if whenever some agent P_j ($j \neq i$) obtains $S_{P_i}(m)$, where m is the pre-agreed contract, then for all $j' \neq i$, it holds that P_i obtains $S_{P_{j'}}(m)$.

Attacker. We assume a "Dolev-Yao" attacker model. This means that the cryptography is assumed to be perfect, i.e. the attacker can only perform cryptographic operations for which he has the appropriate keys. The attacker controls the network and can delay, suppress and insert messages on it. *Honest* participants send only those messages that are allowed by a protocol, whereas *dishonest* participants represent the Dolev-Yao intruder taking part in the protocol under several identities.

Other de£nitions. $PCS^{\tau}_{i,j}$ 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.

3. Analysis

3.1. Minor issues with the Revised GM Protocol

The presentation of the revised GM in [5] has some easily-resolved problems which make it vulnerable to fairness attacks. For example, the variable l_i is not initialised to 1 when abort is requested by P_i; there are some typographical errors in some of the inequalities; and the authors forgot to mention that trusted party T must check the format of the resolve requests, which makes the protocol vulnerable to simple attacks on fairness.

We have £xed all these weaknesses in our presentation of the revised protocol in the preceding section. However, we now argue that the GM contract-signing protocol is fundamentally ¤awed.

3.2. An attack on fairness against the Revised GM protocol

We demonstrate an attack against *fairness* on the revised version of the protocol that involves £ve participants. Later, we generalise it to show that the protocol can not guarantee fairness for any $n > 4$ whatever the trusted authority T does. This shows that there is no Resolve sub-protocol for T that would £x the ¤aw, and thus, the structure of the message exchange in GM's Main protocol is ¤awed. 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 $P_1, \ldots, P_5$ 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 $l_5 = 1$;

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

```
The £rst time $T$ is contacted for contract $m$ (either abort or recovery), $T$ initializes $S(m)$ to $\emptyset$ and $\mathrm{validated}(m)$ to false.
  1. $P_i \rightarrow$ T: $S_{P_i}(m, P_i, (P_1, \ldots, P_n), abort)$
if not $\mathrm{validated}(m)$ 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 ($\mathrm{validated}(m)$=true)
    3. $T \rightarrow P_i$: $\{S_{P_j}((m, \tau_j))\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$
    where $\tau_j$ is the level of the promise from $P_j$ that was converted to a
    universally-veri£able signature during the recovery protocol.
```

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

```
The £rst time $T$ is contacted for contract $m$ (either abort or recovery), $T$ initializes $S(m)$ to $\emptyset$ and $\mathrm{validated}(m)$ to false.
  1. $P_i \rightarrow T$: $S_{P_i}(\{PCS_{P_j}((m, \tau_j), P_i, T)\}_{j \in \{1,\ldots,n\} \setminus \{i\}}, S_{P_i}((m, 1)))$
$T$ checks that the format of this message is one of the £ve permitted formats mentioned in the text.
if $i \in S(m)$ then
    $T$ ignores the message
else if $\mathrm{validated}(m)$ then
    2. $T \rightarrow P_i$: $\{S_{P_j}((m, \tau_j))\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$
    where $\tau_j$ is the level of the promise from $P_j$ that was converted to a
    universally-veri£able signature.
else if $S(m) = \emptyset$ then
    $\mathrm{validated}(m)$:=true
    3. $T \rightarrow P_i$: $\{S_{P_j}((m, \tau_j))\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$
else ($\mathrm{validated}(m)$=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
    $S_T(S_{P_j}(m, P_j, (P_1, \ldots, P_n), abort))$ 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
                                       contacted $T$ 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 $\{S_{P_j}((m, \tau_j))\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$ to $P_i$, stores all the
    signatures, and sets $\mathrm{validated}(m)$ to true.
```

- 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.

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 justi£ed because P_3 and P_5 would not send their signature and 6th level promises to P_2. The attack sequence could have happenned before P_4 sent out his signature and 6th level promises to other signatories. In our presentation above, the attack was sequentialized to make it easier to verify that it is possible.

There are other instantiations of this idea. For example, for the £ve singers 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.

3.3. 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.

3.3.1 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 £ve 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 $PCS^{n-1}_{2,\dots,n-1}, PCS^1_n$. 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 $PCS^{n-1}_{1,2}, PCS^n_{4,\dots,n}$ 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 $PCS^n_{1,\dots,n}$. 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 $PCS^n_{1,2,3}, PCS^{n+1}_{5,\dots n}$ 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.

3.3.2 Resolve impossibility

We show that there is no way to adapt the Abort and Recovery protocols to £x 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 ¤aw lies in the main protocol and there is no resolve protocol for T that would £x it.

4. Conclusion

We have shown that the revised version of Garay and Mackenzie's protocol presented in [5] fails the fairness property, for the cases $n \geq 5$. We generalised our attack to show 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 £x the unfairness. The ¤aw lies in the structure of the message exchange of the Garay and Mackenzie's main protocol.

Acknowledgments. 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.

References

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

[2] 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, Apr. 1997.

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

[4] 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'00)*, volume 1853 of *LNCS*, pages 524–535. Springer, 2000.

[5] 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.

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

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

[8] 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 *LNCS*, pages 151–165, Sept. 1999. Springer-Verlag.

[9] S. Micali. Certi£ed E-mail with invisible post of£ces. Available from author; an invited presentation at the RSA 1997 conference, 1997.

[10] B. P£tzmann, M. Schunter, and M. Waidner. Optimal ef£ciency of optimistic contract signing. In *Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, pages 113–122, May 1998.