J. Zhou, J. A. Onieva, and J. Lopez, "A Synchronous Multi-Party Contract Signing Protocol Improving Lower Bound of Steps", *21st International Information Security Conference (IFIP SEC06)*, LNCS, pp. 221-232, 2006.
NICS Lab. Publications: https://www.nics.uma.es/publications

A Synchronous Multi-Party Contract Signing Protocol Improving Lower Bound of Steps

Jianying Zhou[1], Jose A. Onieva[2], and Javier Lopez[2]

[1] Institute for Infocomm Research
21 Heng Mui Keng Terrace, Singapore 119613
jyzhou@i2r.a-star.edu.sg

[2] Computer Science Department
University of Malaga
29071 - Malaga, Spain
{onieva,jlm}@lcc.uma.es

Abstract. Contract signing is a fundamental service in doing business. The Internet has facilitated the electronic commerce, and it is necessary to find appropriate mechanisms for contract signing in the digital world. A number of two-party contract signing protocols have been proposed with various features. Nevertheless, in some applications, a contract may need to be signed by multiple parties. Less research has been done on multi-party contract signing. In this paper, we propose a new synchronous multi-party contract signing protocol that, with n parties, it reaches a lower bound of $3(n-1)$ steps in the all-honest case and $4n-2$ steps in the worst case (i.e., all parties contact the trusted third party). This is so far the most efficient synchronous multi-party contract signing protocol in terms of the number of messages required. We further consider the additional features like timeliness and abuse-freeness in the improved version.

Keywords: multi-party contract signing, security protocol design, secure electronic commerce.

1 Introduction

The Internet has facilitated the electronic commerce. Many business transactions have been shifted to the Internet. The motivation for such a trend is the efficiency and cost-saving. However, as new risks may arise in the digital world, sufficient security measures should be taken. This will help users to establish the confidence for doing business on the Internet.

Contract signing is a fundamental service for business transactions, and has been well practiced in the traditional paper-based business model. Now, it is necessary to find appropriate mechanisms for contract signing in the digital world. Consider several parties on a computer network who wish to exchange some digital items but do not trust each other to behave honestly. *Fair exchange* is

a problem of exchanging data in a way that guarantees either all participants obtain what they want, or none does. From a designing point of view, contract signing is a particular form of fair exchange, in which the parties exchange commitments to a contract (typically, a text string spelling out the terms of a deal). That is, a contract is a non-repudiable agreement on a given text such that after a contract signing protocol instance, either *each* signer can prove the agreement to any verifier *or none* of them can. If several signers are involved, then it is a *multi-party contract signing* (MPCS) protocol.

There are some two-party contract signing protocols in the literature. Nevertheless, less research has been done on multi-party contract signing. In this paper, we propose a new synchronous multi-party contract signing protocol that, with n parties, it reaches a lower bound of $3(n-1)$ steps in the all-honest case and $4n-2$ steps in the worst case (i.e., all parties contact the trusted third party). This is so far the most efficient synchronous multi-party contract signing protocol in terms of the number of messages required. We further consider the additional features like timeliness and abuse-freeness in the improved version.

The rest of this paper is organized as follows. In Section 2, we review the previous work related to contract signing, outline the properties to be satisfied when designing an optimistic contract signing protocol, and give explicit definitions for some terms used along the descriptions of these protocols. In Section 3, we describe a simple synchronous protocol for multi-party contract signing, then improve the simple version to an optimal multi-party contract signing protocol. After that, we further consider the additional features like timeliness and abuse-freeness in Section 4, and conclude the paper in Section 5.

2 Related Work

As contract signing is a particular case of fair exchange, any fair exchange protocol found in the literature in which digital signatures are exchanged can be considered as the related work. In all practical schemes, contract signing involves an additional player, called *trusted third party* (TTP). This party is (at least to some extent) trusted to behave correctly, thus playing the role of a notary in paper-based contract signing and somehow sharing the legal duties the former ones have. In fact, designing and implementing a contract signing protocol using an on-line TTP should not be a complicated task. In this case, if Alice and Bob wish to enter into a contract, they each sign a copy of the contract and send it to the TTP through a secure channel. The TTP will forward the signed contracts only when it has received valid signatures from both Alice and Bob.

Nevertheless, in our continuous search for speeding up our daily life activities, it is desirable not using a TTP in a contract signing protocol. Additionally, if the TTP is not involved, the notary fee could be avoided. Some protocols appear in the literature trying to eliminate the TTP's involvement using *gradual exchange* of signatures [9, 10]. But these solutions are not deterministic, thus may not be accepted by signatories. Our objective is to focus on contract signing protocols that necessarily use a TTP only in those cases in which an exception

occurs (i.e., a network communication failure or a dishonest party's misbehavior). Otherwise (all-honest-case), the TTP will not be contacted, and parties will bring the protocol to its end by themselves. In the literature, these protocols are called *optimistic contract signing* protocols [2–4, 14–17].

Some properties extracted from the different previous work on optimistic contract signing are summarized as follows.

- *Effectiveness* - if each party behaves correctly, the TTP will not be involved in the protocol.
- *Fairness* - no party will be in an advantageous situation at the end of the protocol.
- *Timeliness* - any party can decide when to finish a protocol run without loosing fairness.
- *Non-repudiation* - no party can deny its action.
- *Verifiability of TTP* - if the TTP misbehaves, all harmed parties will be able to prove it.
- *Transparency of TTP* - if the TTP is contacted to resolve the protocol, the resulting contract will be similar to the one obtained in case the TTP is not involved.
- *Abuse-Freeness* - it is not possible for an attacker (either a legitimate participant or an outsider) to show a third party that the contract final state is under its control.

In [8], Ben-Or *et al.* presented an optimistic contract signing protocol based on a *probabilistic approach*. Such a contract signing protocol is said to be (ν, ϵ)-fair if for any contract C, when signer A follows the protocol properly, if the probability that signer B is privileged to validate the contract with the TTP's help is greater than ν, the conditional probability that "A is not privileged", given that "B is privileged", is at most ϵ.

Previous work in which several signatories are involved in a contract can be found in [1, 6, 11, 12]. Only Asokan *et al.* addressed the MPCS problem in synchronous networks [1]. As they stated, this solution clearly improves the efficiency of those asynchronous protocols previously presented with respect to the number of messages; $4(n-1)$ messages in the all-honest-case and $6n-4$ messages in the worst case. This is possible due to a better reliability of the underlaying network as we can see in Definition 1 below.

Some authors considered the abuse-freeness property in [13, 7]. Baum-Waidner proposed new protocols in [6] that improve the solutions presented for asynchronous networks in [7] such that the number of rounds is significantly reduced in the case that the number of dishonest participants t is considerably less than the total number of participants n - the smaller t is, the better results the new protocols achieve.

Definition 1. A "synchronous" contract signing protocol is used in synchronous networks in which there is a limited time for a message to reach its destination (otherwise it has been lost and the appropriate transport layer manages these events) even if an attack occurs. Thus a party can determine that a

message has not been sent by other party if it did not arrive within the limited time. Users' clocks are assumed to be synchronized.

Definition 2. An "asynchronous" contract signing protocol is used in asynchronous networks in which there is no limited time for a message to reach its destination. Loss and unsorted arrival of messages are possible and have to be managed by the contract signing protocol itself. Clocks are not assumed to be synchronized among users.

A number of protocols exist in the literature which use an asynchronous model of network (i.e., messages can be reordered and lost) with deadline parameters. But when a deadline is introduced, and thus, synchronized clocks among users are assumed (at least when the deadline is approaching), these protocols are converted into synchronous protocols.

In the literature, MPCS protocols make use of either a ring or a matrix topology. Throughout these solutions, authors use the terms *round* and *step* without clearly defining them, which often brings on confusion with respect to the metric to be used for its efficiency evaluation. For this reason we explicitly define these terms as follows:

- *Round* is understood as the existing time slot in which messages are distributed in synchronous networks. In asynchronous networks, the entities need to wait for a local timeout before going to the next round (in case the round is not completed).
- *Step* refers to the action of sending or receiving a message. It is the operation performed by a participating entity. Each round means one step (when all the messages from all entities are distributed or broadcasted in the same time slot, usually in matrix topologies) or several steps (when messages from the same round are distributed from one entity to another, usually in ring topologies).

Some confusion exists in the literature with respect to the term 'round'. Some authors explain that when the next message to be sent depends on the previous one, that is a different round. But we claim two different cases can be found (1) message to be sent depends on the previous one because the entity needs to compute/verify it before sending the next one or (2) message to be sent depends on the previous one because there is a distribution order to be respected (as in ring topologies). We consider a round occurs in Case (1).

All of previous solutions to the asynchronous multi-party contract signing problem reach the lower bound on the number of rounds described in Theorem 3 given in [13]:

> *Any complete and optimistic* asynchronous *contract-signing protocol with n participants requires at least n rounds in an optimistic run.*

Describing the theorem, Garay *et al.* stated that for each party P_i, when it sends a message that can be used (together with other information) by other entities to obtain a valid contract, as the protocol is fair, it must have received

in a previous round, a message from the rest of participants in order to be able to get a valid contract too (probably with the TTP's help), no matter how others behave. By an inductive argument, they showed the number of rounds is at least n.

Ferrer's asynchronous protocol presented in [11] with only three rounds is an exception. It claimed to use a number of rounds independent from the number of participants. However, the protocol is flawed [18].

In our proposal to be presented below, we use a synchronous model, in which we assume messages sent among participants can be lost in the network, but a message from a participant reaches the TTP in a finite and known amount of time. Attackers can insert, delete and modify messages, but it is assumed that attackers cannot break the clock synchronization of the network and cannot forge digital signatures. Under this model, the number of rounds can be made independent of the number of participants.

3 A New Synchronous MPCS Protocol

Here we first present a simple synchronous protocol for multi-party contract signing. As stated before, the only protocol in a synchronous model that we can compare with is Asokan's approach [1]. Our approach is also based on two differentiated phases: a promise to sign, and a real signature that a party releases only after receiving all promises from the rest of participants. Again, in the same manner, we reach a lower bound of $4(n-1)$ steps in the all-honest case and $5n-3$ steps in the worst case that all parties contact the TTP. This result will be further improved in the optimal version by reducing the number of steps to $3(n-1)$ in the all-honest case and $4n-2$ in the worst case.

3.1 A Simple Version

Let us consider the following simple solution which uses verifiable encryption of signatures based on a ring architecture for achieving *transparency* of the TTP. Assume that the channel between any participant and the TTP is functional and not disrupted. The following notation is used in the protocol description.

- $C = [M, P, id, t]$: a contract text M to be signed by each party $P_i \in P(i = 1, \cdots, n)$, a unique identifier id for the protocol run, and a deadline t agreed by all parties to contact the TTP.
- $e_P(X)$: encryption of message X with P's public key.
- $S_P(X)$: P's digital signature on X.
- $Cert_i$: a certificate with which anyone can verify that the ciphertext is the correct signature of the plaintext, and can be decrypted by the TTP (see CEMBS - *Certificate of an Encrypted Message Being a Signature* in [5]).

A simple linear protocol for multi-party contract signing is sketched as follows:

$$
\begin{array}{lll}
1. & P_1 \rightarrow P_2 & : m_1[= C, e_{TTP}(S_{P_1}(C)), Cert_1] \\
2. & P_2 \rightarrow P_3 & : m_1, m_2[= C, e_{TTP}(S_{P_2}(C)), Cert_2] \\
n-1. & P_{n-1} \rightarrow P_n & : m_1, .., m_{n-1}[= C, e_{TTP}(S_{P_{n-1}}(C)), Cert_{n-1}] \\
n. & P_n \rightarrow P_{n-1} & : m_n[= C, e_{TTP}(S_{P_n}(C)), Cert_n] \\
n+1. & P_{n-1} \rightarrow P_{n-2} & : m_{n-1}, m_n \\
2(n-1). & P_2 \rightarrow P_1 & : m_2, m_3, .., m_n \\
2n-1. & P_1 \rightarrow P_2 & : S_{P_1}(C) \\
2n. & P_2 \rightarrow P_3 & : S_{P_1}(C), S_{P_2}(C) \\
3(n-1). & P_{n-1} \rightarrow P_n & : S_{P_1}(C), S_{P_2}(C), .., S_{P_{n-1}}(C) \\
3n-2. & P_n \rightarrow P_{n-1} & : S_{P_n}(C) \\
3n-1. & P_{n-1} \rightarrow P_{n-2} & : S_{P_{n-1}}(C), S_{P_n}(C) \\
4(n-1). & P_2 \rightarrow P_1 & : S_{P_2}(C), S_{P_3}(C), .., S_{P_n}(C)
\end{array}
$$

The above main protocol is divided into two phases. The parties first exchange their commitments in an "in-and-out" manner. Note that P_1 can choose t in the first message (and others can halt if they do not agree). Only after the first phase is finished at step $2(n-1)$, the final signatures are exchanged. Following this simple approach, only $4(n-1)$ steps are needed.

If there is no exception (e.g., network failure or misbehaving party), the protocol will not need the TTP's help. Otherwise, the following resolve sub-protocol helps to drive the contract signing process to its end. P_i can contact the TTP before the deadline t.

$$
\begin{array}{lll}
1. \ P_i \rightarrow TTP & : resolve_{P_i} = C, m_1, .., m_n, S_{P_i}(C, m_1, .., m_n) \\
2. \ TTP & : \text{IF } resolve_{P_i} \text{ is received before } t \text{ THEN} \\
 & \quad \text{decrypts } m_1..m_n \\
 & \quad \text{publishes } S_{P_1}(C), .., S_{P_n}(C)
\end{array}
$$

If the main protocol is not completed successfully, some parties may not hold all the commitments $(m_1, .., m_n)$. Then, they just wait until the deadline t and check with the TTP whether the contract has been resolved by other parties. If not, the contract is cancelled. Otherwise, they get the valid contract $(S_{P_1}(C), .., S_{P_n}(C))$ from the TTP.

If a party has all the commitments when the main protocol is terminated abnormally, it could initiate the above sub-protocol. Then the TTP will help to resolve the contract if the request is received before the deadline t, and the contract will be available to all the participants (even after the deadline t). After the deadline, the TTP will not accept such requests any more. In other words, the status of the contract will be determined the latest by the deadline t.

3.2 Security Analysis

Here we informally analyze our protocol regarding the security properties outlined in Section 2.

- *Effectiveness*: If all parties send all the needed messages correctly, the TTP will not have to decrypt any commitment since after the $4(n-1)$ steps all parties have the contract signed (with n signatures).
- *Fairness*: No party will be in an advantageous situation at the end of the protocol. That is, either all of them possess the contract (or have access to it), or none of them obtains it.
- *Timeliness*: The status of a contract will be finalized either at the end of the main protocol or the latest by a pre-defined deadline t. As the participants not holding all the commitments cannot determine the status of the contract before the deadline t, the property of timeliness is not satisfied. We will further discuss timeliness in Section 4.2.
- *Non-repudiation*: No party can deny its action since each message it sent bears its digital signature.
- *Transparency of TTP*: We use a cryptographic primitive (CEMBS), which allows the users to verify that a bit string is actually the encryption (with the TTP's public key) of the sender's digital signature over the contract C. If the TTP is invoked, it only decrypts the digital signatures and makes them available to all participants. Therefore, after a successful protocol instance, no evidence of the TTP's participation exists.
- *Verifiability of TTP*: Let us identify the possible dishonest behaviors of the TTP: (1) the TTP simply does not reply to participants' requests, or replies with invalid messages; (2) the TTP resolves the protocol but does not publish the contract.

  In the first case, some parties could be beneficiated if they got the contract from the main protocol while others did not. A possible solution is using multiple TTPs and a secure media storage. TTPs have only the write privilege over the media storage but do not control it while participants in the contract signing protocol have only the read privilege over the media storage. A participant can multi-cast his request to the TTPs before the deadline t. As long as one of the TTPs does not misbehave, the correct response will be available from the secure media storage.

  In the second case, the TTP could collude with some parties and resolve the contract for them but does not publish the contract for other parties. That means some parties not holding all the commitments will not get the valid contract. To detect the TTP's cheating, the TTP is required to sign the contract when it is resolved, but this overrides the TTP's transparency. It is difficult to reach a trade-off between transparency and verifiability.
- *Abuse-freeness*: In our protocol, the last participant (P_n) in the ring can decide whether to resolve the protocol after receiving all the commitments from other parties However, as stated in [7] it is not possible to avoid this participant to control whether the normal flow of the protocol continues or not, but all we can aim to is to avoid that it is able to provide evidence to an outsider about its control over the result of the contract. So, for P_n holding $m_1, \cdots, m_{n-1}$, due to the presence of $Cert_i$ in m_i that anyone can verify, it is possible for P_n to *abuse* about the state of the contract. Nevertheless, we show in Section 4.1 that the property of abuse-freeness can be achieved.

3.3 An Optimal Version

The protocol in Section 3.1 has two clearly differentiated phases: exchange of commitments and exchange of digital signatures. The number of steps can be further reduced if we send more available information at each step and thus merge both phases. This will result in an improvement to the previous simple version protocol.

Using the same notation, an optimal synchronous protocol for multi-party contract signing is outlined as follows:

$$
\begin{array}{lll}
1. & P_1 \rightarrow P_2 & : m_1[= C, e_{TTP}(S_{P_1}(C)), Cert_1] \\
2. & P_2 \rightarrow P_3 & : m_1, m_2[= C, e_{TTP}(S_{P_2}(C)), Cert_2] \\
n-1. & P_{n-1} \rightarrow P_n & : m_1, .., m_{n-1}[= C, e_{TTP}(S_{P_{n-1}}(C)), Cert_{n-1}] \\
n. & P_n \rightarrow P_{n-1} & : m_n[= C, e_{TTP}(S_{P_n}(C)), Cert_n], S_{P_n}(C) \\
n+1. & P_{n-1} \rightarrow P_{n-2} & : m_{n-1}, m_n, S_{P_{n-1}}(C), S_{P_n}(C) \\
2(n-1). & P_2 \rightarrow P_1 & : m_2, m_3, .., m_n, S_{P_2}(C), S_{P_3}(C), .., S_{P_n}(C) \\
2n-1. & P_1 \rightarrow P_2 & : S_{P_1}(C) \\
2n. & P_2 \rightarrow P_3 & : S_{P_1}(C), S_{P_2}(C) \\
3(n-1). & P_{n-1} \rightarrow P_n & : S_{P_1}(C), S_{P_2}(C), .., S_{P_{n-1}}(C)
\end{array}
$$

The resolve sub-protocol used by participants to request the TTP's help is the same as presented in Section 3.1. Note that even though the two phases are merged, no party releases its plaintext signature of the contract without having first received all the commitments. If any party decides to quit after releasing its plaintext signature of the contract, the rest of participants can obtain the plaintext signatures of the contract with the TTP's help. As the protocol is similar to the previous one, the same security properties are fulfilled.

This optimal version permits overlapping the dispatch of promises with real signatures without loosing fairness. It improves the simple version presented in Section 3.1 by reducing the number of steps to $3(n-1)$ in the all-honest case and $4n-2$ in the worst case. Note that for $n=2$, three messages are sufficient, as shown in [17].

4 Further Discussions

The MPCS protocol presented in the previous section improved the lower bound of steps. However, as we pointed out in the security analysis that it does not satisfy the properties of abuse-freeness and timeliness. Here we further improve our MPCS protocol to address these properties.

4.1 Achieving Abuse-Freeness

Although it is not possible to force a participant to keep on following the steps of the protocol, we can design the protocol in such a manner that it has no way to demonstrate to an outsider the contract is under its control. For this purpose, we use a *blind commitment* that only the TTP can verify. With this

concept of design in mind, we modify the previous protocol to eliminate the *illustrative* information. The main protocol remains the same, but $Cert_i$ is not included in m_i. Instead, the evidence of origin of the blind commitment $Commit_i$ is generated:

$$Commit_i = S_{P_i}(h(C), e_{TTP}(S_{P_i}(C)))$$

where $h(C)$ is the hash value of C to be used to establish a unique link between $Commit_i$ and C.

1.	$P_1 \rightarrow P_2$	$: m_1[= C, e_{TTP}(S_{P_1}(C)), Commit_1]$
2.	$P_2 \rightarrow P_3$	$: m_1, m_2[= C, e_{TTP}(S_{P_2}(C)), Commit_2]$
$n-1$.	$P_{n-1} \rightarrow P_n$	$: m_1, .., m_{n-1}[= C, e_{TTP}(S_{P_{n-1}}(C)), Commit_{n-1}]$
n.	$P_n \rightarrow P_{n-1}$	$: m_n[= C, e_{TTP}(S_{P_n}(C)), Commit_n], S_{P_n}(C)$
$n+1$.	$P_{n-1} \rightarrow P_{n-2}$	$: m_{n-1}, m_n, S_{P_{n-1}}(C), S_{P_n}(C)$
$2(n-1)$.	$P_2 \rightarrow P_1$	$: m_2, m_3, .., m_n, S_{P_2}(C), S_{P_3}(C), .., S_{P_n}(C)$
$2n-1$.	$P_1 \rightarrow P_2$	$: S_{P_1}(C)$
$2n$.	$P_2 \rightarrow P_3$	$: S_{P_1}(C), S_{P_2}(C)$
$3(n-1)$.	$P_{n-1} \rightarrow P_n$	$: S_{P_1}(C), S_{P_2}(C), .., S_{P_{n-1}}(C)$

Note each party needs to check whether all the blind commitments it has received are valid before releasing its real signature of the contract. A *valid* blind commitment $Commit_i$ means it is from P_i (by checking its signature), linked to C (by checking $h(C)$), but does not guarantee that $e_{TTP}(S_{P_i}(C))$ in $Commit_i$ matches $S_{P_i}(C)$. $Commit_i$ is *correct* if it is valid and also matches $S_{P_i}(C)$.

If there is no exception (e.g., network failure or misbehaving party), the protocol will not need the TTP's help. Otherwise, a modified resolve sub-protocol helps to drive the contract signing process to its end. P_i can contact the TTP before the deadline t.

1. $P_i \rightarrow TTP$: $resolve_{P_i} = C, m_1, .., m_n, S_{P_i}(C, m_1, .., m_n)$
2. TTP : IF $resolve_{P_i}$ is received before t
   AND all $Commit_i$ are valid THEN
   decrypts & verifies $m_1..m_n$
   IF $S_{P_1}(C), .., S_{P_n}(C)$ ok THEN
   publishes $S_{P_1}(C), .., S_{P_n}(C)$
   ELSE IF $P_i \notin group_f$
   publishes $fail, group_f, S_{TTP}(fail, C, group_f)$

When a party holding all the *valid* blind commitments initiates the above sub-protocol, the TTP will help to resolve the contract if the request is received before the deadline t. The TTP decrypts and verifies $m_1, \cdots, m_n$. If they are all correct, the TTP will publish $S_{P_1}(C), \cdots, S_{P_n}(C)$. Otherwise, the TTP will invalidate the contract by publishing a *fail* token $S_{TTP}(fail, C, group_f)$ where $group_f$ indicates the parties misbehaved in generating their commitments.

The dispute resolution process is changed when the *fail* token is introduced. If a party can show this token, the contract is invalid. Therefore, at the end of the

main protocol, each party needs to check whether $e_{TTP}(S_{P_i}(C))$ in $Commit_i$ matches $S_{P_i}(C)$ for $i = 1, \cdots, n$ (assuming the encryption algorithm is deterministic). If not, it should initiate the above sub-protocol to get the *fail* token. Note, a party P_i cannot get any advantage by providing different $Commit_i$ in the main protocol and the resolve sub-protocol. If P_i provides correct $Commit_i$ in the main protocol but incorrect $Commit'_i$ in the sub-protocol, P_i will not get the *fail* token, i.e., cannot cancel a protocol instance whose final state is signed. On the other hand, if P_i provides incorrect $Commit_i$ in the main protocol but correct $Commit'_i$ in the sub-protocol, P_i may get the signed contract if other parties did not misbehave in generating their commitments, but any other honest party can initiate the resolve sub-protocol to get the *fail* token, thus the contract is still invalid.

The blind commitment does not allow a participant to demonstrate that the protocol state is under its control. In fact, in this case, getting all m_i does not mean being able to solve the protocol as in previous protocols presented in this paper. Thus it provides an abuse-freeness feature. Proof is straightforward, since there is no point in the protocol in which an entity can ensure, even to itself, that the contract is signed till plaintext signatures are obtained. The solution allows to maintain the same number of steps as the optimal protocol in Section 3.3. Furthermore, the TTP is still transparent in this sub-protocol as the signed contract published by the TTP is the same as obtained in the main protocol.

4.2 Achieving Timeliness

In the previous protocols just presented, a deadline t is selected by the first participant. If other participants disagree with the deadline, they can simply abort the execution of the protocol. Of course, this deadline could be negotiated among the participants before the contract signing protocol is initiated.

If the main protocol is not completed successfully, some participants may hold all the commitments while the others may only hold part of the commitments. For those holding all the commitments, they have the freedom to either resolve the contract with the TTP's help before the deadline t, or take no action and just let the contract being automatically cancelled after the deadline t.

However, for those only holding part of the commitments, they have no options but only wait until the deadline t to know the status of the contract. Obviously, this is unfavorable to these participants in term of timeliness. They should also have the right to decide the status of the contract before the deadline t. As they only hold part of commitments, they are not able to resolve the contract, so they can only choose to cancel the contract. (Note that in our "in-and-out" architecture of commitment exchange, for those participants only holding part of the commitments, even if all of them collaborate, their combined commitments are still incomplete to resolve the contract.)

Here we present a (j, n)-*threshold* cancel sub-protocol. As long as there are at least j out of n participants that wish to cancel the contract before the deadline t, the contract could be cancelled. The cancel sub-protocol is as follows, where *counter* records the number of cancel requests received by the TTP, and $group_c$

records the participants which made cancel requests. For simplicity of description, it is built based on the main protocol in Section 3.3 without considering abuse-freeness.

1. $P_i \rightarrow TTP$: $cancel_{P_i} = C, cancel, S_{P_i}(C, cancel)$
2. TTP : IF $cancel_{P_i}$ is received before t
   AND C is not resolved THEN
   stores $cancel_{P_i}$; $group_c = group_c + P_i$;
   $counter++$;
   IF $counter \geq j$ THEN
   sets C as cancelled
   publishes $cancel, group_c, S_{TTP}(cancel, C, group_c)$

The resolve sub-protocol is modified as follows.

1. $P_i \rightarrow TTP$: $resolve_{P_i} = C, m_1, .., m_n, S_{P_i}(C, m_1, .., m_n)$
2. TTP : IF $resolve_{P_i}$ is received before t
   AND C is not cancelled THEN
   decrypts $m_1..m_n$
   sets C as resolved
   publishes $S_{P_1}(C), .., S_{P_n}(C)$

With the above cancel and resolve sub-protocols, each participant has at least one option to determine the status of the contract before deadline t if the main protocol is not completed successfully. Thus timeliness is achieved, and the extent of timeliness depends on the threshold value j: strong timeliness when $j = 1$, and weak timeliness when $j = n$.

However, the threshold value j should be selected carefully. If j is too small, a few parties may collude to invalidate a contract. If j is too big, it might be hard to establish a valid cancel request among j parties. A possible option is $j = [n/2] + 1$, with a weak majority to "vote" for the validity of a contract.

In the dispute resolution, the *cancel* token issued by the TTP has the top priority. In other words, if a participant presents the *cancel* token, then the contract is invalid. That implies if there are at least j out of n participants who want to cancel the contract before the deadline, even if they have released their plaintext signatures in the main protocol, they together can still change their mind before that deadline. This is a reasonable scenario in the real world because the situation defined in the contract may change with time, even during the process of contract signing, and each participant wishes to pursue the maximum benefit by taking appropriate actions (resolve or cancel).

As the *cancel* token from the TTP has higher priority than the signed contract, those parties that have got the signed contract in the main protocol may need to double check with the TTP about the status of the contract by the deadline t. (Note that the double check does not mean the involvement of the TTP itself, but just a query to a public file maintained by the TTP.) If they do not want to wait until that deadline, they can send the resolve request to the TTP instead, thus blocking other parties to enable the TTP to issue the *cancel* token.

5 Conclusions

Contract signing is a fundamental service for business transactions. Previous work mainly focused on two-party contract signing. In some applications, however, a contract may need to be signed by multiple parties.

In this paper, we presented a new multi-party contract signing protocol that reaches a lower bound of $3(n-1)$ steps in the all-honest case and $4n-2$ steps in the worst case (i.e., all parties contact the TTP). The result improves the lower bound of $4(n-1)$ steps for the all-honest case and $6n-4$ steps for the worst case in Asokan *et al.*'s protocol [1]. Actually, our protocol is so far the most efficient synchronous multi-party contract signing protocol in terms of the number of messages required.

We further considered the additional features like abuse-freeness and timeliness in our protocol. With no special requirements and more importantly without introducing additional steps in the protocol, we achieved the abuse-freeness property which is very important for contract signing protocols. In addition, by introducing the concept of threshold cancel sub-protocol, we achieved the timeliness property. Achieving the TTP's strong verifiability while keeping its transparency is an open issue to be further investigated. Future work also includes formal security analysis of our protocol.

Acknowledgements

We thank the anonymous reviewers for their valuable comments and suggestions on the improvement of this paper. The second author has been funded by the Consejeria de Innovacion, Ciencia y Empresa (Junta de Andalucia) under the III Andalusian Research Plan.

References

1. N. Asokan, Birgit Baum-Waidner, Matthias Schunter, and Michael Waidner. Optimistic synchronous multi-party contract signing. Technical Report RZ 3089, IBM Zurich Research Lab, 1998.
2. N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for multi-party fair exchange. Technical Report RZ 2892, IBM, Zurich Research Laboratory, 1996.
3. N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *Proceedings of 4th ACM Conference on Computer and Communications Security*, pages 7–17. ACM Press, 1997.
4. N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000.
5. Feng Bao, Robert Deng, and Wenbo Mao. Efficient and practical fair exchange protocols with off-line TTP. In *Proceedings of 1998 IEEE Symposium on Security and Privacy*, pages 77–85. IEEE, May 1998.

6. Birgit Baum-Waidner. Optimistic asynchronous multi-party contract signing with reduced number of rounds. In *Proceedings of 28th International Colloquium on Automata, Languages and Programming*, pages 898–911. Springer, 2001.
7. Birgit Baum-Waidner and Michael Waidner. Round-optimal and abuse-free multi-party contract signing. In *Proceedings of 27th International Colloquium on Automata, Languages and Programming*, LNCS 1853, pages 524–535. Springer, 2000.
8. M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, volume 36, pages 40–46, 1990.
9. M. Blum. Three applications of the oblivious transfer. Technical Report, Department of EECS, University of California, Berkeley, CA, 1981.
10. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, volume 28, pages 637–647, 1985.
11. Josep Lluís Ferrer-Gomila, Magdalena Payeras-Capellà, and Llorenç Huguet-Rotger. Efficient optimistic n-party contract signing protocol. In *Proceedings of 4th International Conference on Information Security*, pages 394–407. Springer, 2001.
12. Josep Lluís Ferrer-Gomila, Magdalena Payeras-Capellà, and Llorenç Huguet-Rotger. Optimality in asynchronous contract signing protocols. In *Proceedings of 1st International Conference on Trust and Privacy in Digital Business*, LNCS 3184. Springer, August 2004.
13. Juan A. Garay and Philip D. MacKenzie. Abuse-free multi-party contract signing. In *Proceedings of 13th International Symposium on Distributed Computing*, pages 151–165. Springer, 1999.
14. N. González-Deleito and O. Markowitch. An optimistic multi-party fair exchange protocol with reduced trust requirements. In *Proceedings of 4th International Conference on Information Security and Cryptology*, LNCS 2288, pages 258–267. Springer, December 2001.
15. O. Markowitch and S. Saeednia. Optimistic fair-exchange with transparent signature recovery. In *Proceedings of Financial Cryptography 2001*. Springer, February 2001.
16. Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *Proceedings of 22nd Annual Symposium on Principles of Distributed Computing*, pages 12–19. ACM Press, 2003.
17. Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Optimal efficiency of optimistic contract signing. In *Proceedings of 17th Annual ACM Symposium on Principles of Distributed Computing*, pages 113–122. ACM Press, 1998.
18. Jose Onieva, Jianying Zhou, and Javier Lopez. Attacking an asynchronous multi-party contract signing protocol. In *Proceedings of 6th International Conference on Cryptology in India*, LNCS 3797, pages 311–321. Springer, December 2005.