Abuse-free Multi-party Contract Signing

JUAN A. GARAY* PHILIP MACKENZIE*

Information Sciences Research Center
Bell Laboratories
600 Mountain Ave
Murray Hill, NJ 07974

Abstract. In the contract-signing problem, participants wish to sign a contract m in such a way that either all participants obtain each others' signatures, or nobody does. A contract-signing protocol is *optimistic* if it relies on a trusted third party, but only uses it when participants misbehave (e.g., try to cheat, or simply crash).
We construct an efficient general multi-party optimistic contract-singing protocol. The protocol is also *abuse-free*, meaning that at no point can a participant prove to others that he is capable of choosing whether to validate or invalidate the contract. This is the first abuse-free optimistic contract-signing protocol that has been developed for $n > 3$ parties. We also show a linear lower bound on the number of rounds of any n-party optimistic contract-signing protocol.

1 Introduction

A contract is a non-repudiable agreement on a given text [7]. Contract signing is an important part of any business transaction, in particular in settings where participants do not trust each other to some extent already. Thus, the World Wide Web is probably the best example of a setting where contract signing schemes are needed. Still, even though a great amount of research has gone into developing methods for contract signing over a network such as the WWW, the case of contract signing by an arbitrary number of signatories was only recently considered, with most of the work concentrating on the case of two parties. In this paper we present a general n-party optimistic contract-signing protocol.

Electronic contract signing. Contract signing is part of the broader problem of fair exchange [8, 11, 17, 21, 2]. More specifically, it can be considered fair exchange of digital signatures [4]. The term "contract signing" was first introduced in [7].

Early work on electronic contract signing, or more generally, fair exchange of secrets/signatures, focused on the gradual release of secrets to obtain simultaneity, [6, 15, 19] (see [12] for more recent results). The idea is that if each party alternately releases a small portion of the secret, then neither party has a considerable advantage over the other. Unfortunately, such a solution has several

* E-mail: {garay,philmac}@research.bell-labs.com.

drawbacks. Apart from being expensive in terms of computation and communication, it has the problem in real situations of uncertain termination: if the protocol stops prematurely and one of the participants does not receive a message, he will never be sure whether the other party is continuing with the protocol, or stopped—and perhaps even engaged in another contract-signing protocol!

More recently, considerable efforts have been devoted to develop efficient protocols that mimic the features of "paper contract signing," especially *fairness*. A contract-signing protocol is *fair* if at the end of the protocol, either both parties have valid signatures for a contract, or neither does. In some sense, this corresponds to the "simultaneity" property of traditional paper contract signing. That is, a paper contract is generally signed by both parties at the same place and at the same time, and thus is fair.

An alternative approach to achieving fairness instead of relying on the gradual release of secrets has been to use a *trusted third party* (TTP). A TTP is essentially a judge that can be called in to handle disputes between contract signers. This was the case in the early work of [14], where the contract is only valid if a "center of cancellation" does not object. The TTP can be *on-line* in the sense of mediating after every exchange as in [11, 13, 17], at the expense of the TTP becoming a potential bottleneck, or *off-line*, meaning that it only gets involved when something goes wrong (e.g., a participant attempts to cheat, or simply crashes, or the communication delays between the participants are intolerably high, etc.). The latter approach has been called *optimistic* [2], and fair contract-signing protocols have been developed in this model [2, 4, 3]. Although the TTP is (by definition) trusted, it is, in some protocols, possible for the TTP to be accountable for his actions. That is, it can be determined if the TTP misbehaved.

However, until recently, "fair" contract-signing protocols had the conceptual drawback of allowing a party (say, Alice) at some point to show to outside parties that she has a *potentially signed contract* (say, with Bob), as well as control over whether the contract will be validated. (In some cases protocols are even asymmetric, in the sense of not giving the other party (Bob) the same ability.) Alice may take advantage of this feature by coercing another party, say Carol, into another contract on the threat of signing the contract with Bob. and then cancelling the contract with Bob. (Bob would probably not appreciate this.) Another example of abuse would be in "We will beat your best deal" scenarios, in which Alice can take a validated price on some merchandise from vendor Val and present it to merchant Mary to achieve a better price, and later renege on the deal with Val.

In [18] this type of attack was pointed out, and an efficient *abuse-free* optimistic contract-signing protocol was presented. Although not mentioned, a protocol for fair exchange of digital signatures presented in [4] is also abuse-free; however, that protocol is inefficient in that it requires cut-and-choose techniques.

In [23] the optimal efficiency of optimistic contract-signing protocols is considered, and they give tight lower bounds on the message and round complexity of 2-party protocols.

Most of the protocols previously developed are for two-party contract signings. In [18] an intricate protocol for three parties is presented. Independently and concurrently to our work, the case of multi-party contract signing was considered in [1, 5]. In [1], an optimistic multi-party contract-signing protocol is given for the case of synchronous networks, while in [5], an optimistic multi-party contract-signing protocol is given for asynchronous networks; these protocols, however, are not abuse-free. To our knowledge, there has been no proposal on how to produce general n-party abuse-free contract signings.

Our results. We give a construction of an optimistic abuse-free contract-signing protocol for an arbitrary number of participants n. The construction is modular, requires $O(n^3)$ messages and $O(n^2)$ rounds of interaction (in an asynchronous environment), and tolerates an arbitrary number of faulty (Byzantine) parties. The protocol uses *private contract signatures*, introduced in [18]. We also show a lower bound of n rounds for any optimistic contract-signing protocol (i.e., even if it is not abuse-free).

Other related work. As pointed out in [23], contract signing bears some resemblance to atomic commitment [25] and Byzantine agreement [22]. Contract signing is to be robust against malicious participants while providing a non-repudiable agreement; on the other hand, atomic commitment tolerates only benign failures (is unattainable otherwise) and does not require that participants reach a decision (also, when they do, *all* decisions—including those of the faulty participants—must be consistent). The failure model of contract signing is similar to that of Byzantine agreement—although in the TTP version the trusted third party is assumed to remain uncorrupted. However, contract signing achieves more than reaching agreement: all the participants should be able to prove it afterwards.

Organization of the paper. The remainder of the paper is organized as follows. Section 2 describes the model and states the relevant definitions. Section 3 describes the cryptographic tools that we need for the design. The multi-party abuse-free contract-signing protocol is presented in Section 4, and the proof of correctness and security is given in Section 5. Section 6 presents the efficiency analysis and the lower bound on the number of rounds. Section 7 offers some final remarks.

2 Model and Definitions

Our basic model is similar to that of [4]. We have a set of participants $\mathcal{S} = \{P_1, P_2, \cdots, P_n\}$, and a trusted third party T. Participants may be correct or faulty (Byzantine). Formally, the participants and the trusted third party are modelled by probabilistic interactive Turing machines. We assume all participants have public/private keys which will be specified later. T acts like a server, responding (atomically) to requests from the participants, with responses defined later. We assume that communication between any participants and T is over a private channel.

The network model we consider is the same as in [4]. Namely, it is an asynchronous communication model with no global clocks, where messages can be delayed arbitrarily, but with messages sent between correct participants and the trusted third party guaranteed to be delivered eventually. In general, we assume an adversary may schedule messages, and possibly insert its own messages into the network.

The participants, $\{P_1, \ldots, P_n\}$, wish to "sign" a contract m.[1] By signing a contract, we mean providing public, non-repudiable evidence that they have agreed to the contract. The type of evidence is either universally agreed upon, or is part of the contract m itself. For instance, valid evidence may include either valid signatures by all participants on m, or valid signatures by all participants on the concatenation of m and the public key of T, signed together by T.

Obviously, in the asynchronous model of [16], an adversary may prevent a contract from being signed simply by delaying all messages between players. Thus in order to force contract-signing protocols to be non-trivial, we specify a completeness condition using a slightly restricted adversary. An optimistic contract-signing protocol is *complete* if the (slightly restricted) adversary cannot prevent a set of correct participants from obtaining a valid signature on a contract. This adversary has signing oracles that can be queried on any message except m, can interact with T, and can arbitrariliy schedule messages from the participants to T. However, it cannot delay messages between the correct participants enough to cause any timeouts.

An optimistic contract-signing protocol is *fair* if

1. it is impossible for a set of corrupted participants, A, to obtain a valid contract without allowing the remaining set of participants, B, to also obtain a valid contract;
2. once a correct participant obtains a cancellation message from the TTP T, it is impossible for any other participant to obtain a valid contract; and
3. every correct participant is guaranteed to complete the protocol.

Effectively, condition 2 provides a *persistency* condition, stating that correct participants cannot have their outcomes overturned by other participants. In previous two-party protocols, this condition was never explicitly discussed, although the condition was trivially satisfied by those protocols. With multi-party protocols, the condition is much more difficult to satisfy.

We assume the corrupted participants have a signing oracle to obtain signatures from the remaining participants B on any message except m. The corrupted participants may also interact with T and the messages from B to T may be arbitrarily scheduled.

[1] In general, each participant might need to sign different pieces of text. Extension to this case is trivial.

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

We say an optimistic contract-signing protocol is *secure* if it is fair, complete, and abuse-free.

3 Cryptographic Tools

3.1 Discrete logs

For a prime q, we will be working in a group G_q of order q with generator g. For a specific example, we could use a group form by integer modular arithmetic as follows. Let p be a prime of the form $lq + 1$ for some value l co-prime to q, and let Z_p^* be the group, with g a generator of order q in Z_p^*. Then G_q would be the subgroup generated by g.

We assume the hardness of the *Decision Diffie-Hellman problem* (DDH). In this problem, it is the goal of the polynomial-time adversary to distinguish the following two distributions with a non-negligible advantage over a random guess:

1. (g_1, g_2, y_1, y_2) with $g_1, g_2, y_1, y_2 \in_R G_q$, and
2. (g_1, g_2, g_1^r, g_2^r) with $g_1, g_2 \in_R G_q$ and $r \in_R Z_q$.

The security of the digital signatures described in the next section relies on the DDH assumption.

3.2 Private contract signatures

Our protocols use *private contract signatures* (PCS), a type of digital signatures that were introduced in [18]. Roughly, the properties of these signatures can be summarized as follows: These signatures are:

- **Undeniable** [9]: they are not self-authenticating or universally verifiable—they cannot be transferred;
- **designated-verifier** [20]: only a selected receiver chosen by the signatory can verify them and convince himself of their validity;
- **designated-converter** [10]: a third party (designated by the signatory) can convert them into self-authenticating signatures; and
- **non-interactive**: the receiver of the signature knows beforehand that the third party can convert the signature, without the need for a verification session.

That is, upon receiving a PCS, a party convinces himself of its validity, but cannot convince anybody else, and he also knows that the third party appointed by the signatory can convert it into a regular (self-authenticating) signature.

For completeness, we include the full formal definition here.

Definition 1. *A* private contract signature (PCS) *scheme Σ is a tuple of probabilistic polynomial-time algorithms* $\{\mathsf{PCS\text{-}Sign}, \mathsf{S\text{-}Convert}, \mathsf{TP\text{-}Convert}, \mathsf{PCS\text{-}Ver}, \mathsf{S\text{-}Ver}, \mathsf{TP\text{-}Ver}\}$ *defined as follows, and having the security properties defined below.*

(1) PCS-Sign *executed by party A on m for B with respect to third party T, denoted* $\mathsf{PCS\text{-}Sign}_A(m, B, T)$*, outputs a private contract signature, denoted* $\mathrm{PCS}_A(m, B, T)$*. A private contract signature can be verified using* PCS-Ver, *i.e.,*

$$\mathsf{PCS\text{-}Ver}(m, A, B, T, S) = \begin{cases} \text{true} & \textit{if } S = \mathrm{PCS}_A(m, B, T); \\ \text{false} & \textit{otherwise.} \end{cases}$$

(2) S-Convert *executed by A on a private contract signature $S = \mathrm{PCS}_A(m, B, T)$ generated by A, denoted* $\mathsf{S\text{-}Convert}_A(S)$*, produces a universally-verifiable signature by A on m,* $\text{S-Sig}_A(m)$.

(3) TP-Convert *executed by T on a private contract signature $S = \mathrm{PCS}_A(m, B, T)$, denoted* $\mathsf{TP\text{-}Convert}_T(S)$*, produces a universally-verifiable signature by A on m,* $\text{TP-Sig}_A(m)$.

(4) $\text{S-Sig}_A(m)$ *can be verified using* S-Ver, *and* $\text{TP-Sig}_A(m)$ *can be verified using* TP-Ver, *i.e.,*

$$\mathsf{S\text{-}Ver}(m, A, T, S) = \begin{cases} \text{true} & \textit{if } S = \text{S-Sig}_A(m); \\ \text{false} & \textit{otherwise;} \end{cases}$$

and

$$\mathsf{TP\text{-}Ver}(m, A, T, S) = \begin{cases} \text{true} & \textit{if } S = \text{TP-Sig}_A(m); \\ \text{false} & \textit{otherwise.} \end{cases}$$

The security properties of a PCS *scheme are:*

(1) **Unforgeability of** $\mathrm{PCS}_A(m, B, T)$**:** *For any m, it is infeasible for anyone but A or B to produce T and S such that* $\mathsf{PCS\text{-}Ver}(m, A, B, T, S) = \text{true}$.

(2) **Designated verifier property of** $\mathrm{PCS}_A(m, B, T)$**:** *For any B, there is a polynomial-time algorithm* FakeSign *such that for any m, A, and T,* $\mathsf{FakeSign}_B(m, A, T)$ *outputs S, where* $\mathsf{PCS\text{-}Ver}(m, A, B, T, S) = \text{true}$.

(3) **Unforgeability of** $\text{S-Sig}_A(m)$ **and** $\text{TP-Sig}_A(m)$**:** *For any m, A, B, T, assuming $P = \mathrm{PCS}_A(m, B, T)$ is known and was produced by* $\mathsf{PCS\text{-}Sign}_A(m, B, T)$*, it is infeasible*

  (3.1) *for anyone but A or T to produce S such that* $\mathsf{S\text{-}Ver}(m, A, T, S) = \text{true}$ *and* $\mathsf{TP\text{-}Ver}(m, A, T, S) = \text{true}$*;*

  (3.2) *for anyone but A to produce S such that* $\mathsf{S\text{-}Ver}(m, A, T, S) = \text{true}$ *and* $\mathsf{TP\text{-}Ver}(m, A, T, S) = \text{false}$*; and*

  (3.3) *for anyone but T to produce S such that* $\mathsf{TP\text{-}Ver}(m, A, T, S) = \text{true}$ *and* $\mathsf{S\text{-}Ver}(m, A, T, S) = \text{false}$.

Definition 2. *A* PCS *scheme Σ is* third party-accountable *if for any* $\mathrm{PCS}_A(m, B, T)$*, the distributions of* $\text{S-Sig}_A(m)$ *and* $\text{TP-Sig}_A(m)$*, produced by* S-Convert *and* TP-Convert*, respectively, are disjoint and efficiently distinguishable.*

That is, it is impossible to have $\mathsf{S\text{-}Ver}(m, A, T, S) = \mathsf{TP\text{-}Ver}(m, A, T, S) = \text{true}$ for any S, and thus it is possible for a verifier to distinguish whether the conversion was performed by the signatory or by the third pary. This property might be useful if the signatories are concerned about the third party's trustworthiness. A somewhat complementary property is the following.

Definition 3. *A* PCS *scheme Σ is* third party-invisible *if for any* $\mathrm{PCS}_A(m, B, T)$*, the distributions of* $\text{S-Sig}_A(m)$ *and* $\text{TP-Sig}_A(m)$*, produced by* S-Convert *and* TP-Convert*, respectively, are identical.*

In other words, for any S, $\mathsf{S\text{-}Ver}(m, A, T, S) = \mathsf{TP\text{-}Ver}(m, A, T, S)$, meaning no one can determine if a conversion was performed by the original signer or the trustee. This property might be useful in some scenarios where signatories may not want possible "bad publicity" associated with a contract needing to be converted by the third party.

In [18] it is shown:

Theorem 1 ([18]). *Assuming the security of* DDH*, there exists a* PCS *scheme (in the random oracle model).*

4 The Multi-party Contract-Signing Protocol

4.1 Alternating contract-signing protocols

The basic structure of an optimistic 2-party protocol (e.g., [18]) is shown in Figure 1. (The protocol of [4] has the same structure, except that the underlying primitives are different, and provide different functionality.) The protocol has an alternating structure, with a previously agreed-upon initiator, say, party A. If no problems occur, A sends a "promise" (i.e., a private contract signature of Section 3) that she will sign the message to B. B verifies A's PCS and gets convinced of A's intentions—but cannot convince anybody else, and responds with his PCS. Then A converts her PCS into a universally-verifiable signature, and then B does the same.

If problems occur, A or B may run protocols to *abort* or *resolve* the contract signing, depending on what step has been attained in the protocol. These protocols are run between A or B and T, the trusted party. Since A sends her message first and might not hear from B, or the PCS she receives could be wrong, the purpose of abort is to give A the power to terminate the protocol. If resolve is invoked, the participant converts his own PCS into a regular signature, and sends it to T together with what he received from the other participant. Depending on the state of the protocol that it keeps, T grants an abort or converts the PCS into a regular signature and sends it back to the participant who invoked the resolve protocol. (T is able to perform the signature conversions given its designated-converter status.) It is shown in [18] that, assuming the security of DDH, the protocol of Fig. 1 is a secure contract-signing protocol for two parties.

Notice that in the protocol only A can abort the protocol. The reason for this is that if also B could abort, then the abuse freeness condition would be violated.

Specifically, after receiving A's converted signature, S-$\mathrm{Sig}_A(m)$, B could prove to an outside party that he has a valid contract signed by A, as well as the power to terminate it. There is also a reason for the protocol being alternating (first A sends a message, then B does, and so forth), for if it were not (i.e., both parties could start "simultaneosuly"), then the same type of scenario as the one above could occur, with one party being able to prove that he has a valid contract.

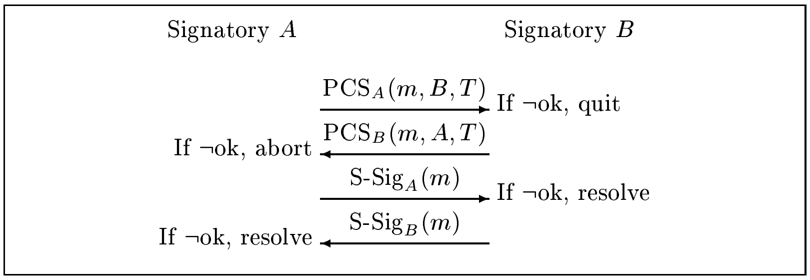

Fig. 1. *A secure contract-signing protocol for two parties*

It was pointed out in [18] that this type of protocol does not extend easily to the multi-party case; specifically, it is unclear whether to let the "middle" parties abort or not. In the next section we present a contract-signing protocol for $n \geq 3$ parties that is secure. The approach we take is to let aborts be overturned, but in a way that avoids the violations of fairness and abuse freeness pointed out in [18]. Naturally, we use the private contract signatures of Section 3 as "promises."

4.2 Protocol description

Our protocol can be described recursively. For i participants P_1 through P_i to sign a contract, P_i indicates its willingness to sign the contract to P_1 through P_{i-1}, then participants P_1 through P_{i-1} come to an agreement about the contract (with promises, not signatures), letting P_i know about this (again with promises). Then all participants exchange promises to sign the contract. Only when all recursive levels are finished do the participants exchange signatures on a contract.

To apply this recursive approach, we use different "strengths" of promises for the different levels of recursion. We denote the strength of a promise by an integer "level." Specifically, an "i-level promise from A to B on a message m" is denoted $\mathrm{PCS}_A((m, i), B, T)$. Our approach is shown in Figure 2. Note that the order of participants is reversed (P_i down to P_1) to match the recursion levels.

If there is no misbehavior by any participants, there is never a need to contact the trusted third party T. If T is contacted, it needs to store some information to handle the abort and/or resolve messages it may receive. Specifically, if there is a positive resolution (meaning the contract must be honored) it needs to store a completely signed contract (with the promises possibly converted to universal signatures by T itself). If there has not been a positive resolution, T needs to store an “abort list” containing indices of all the parties that have aborted, or tried to resolve unsuccessfully. T will also store the first abort message it received, along with another “forced abort list” that will be explain in the protocol below.

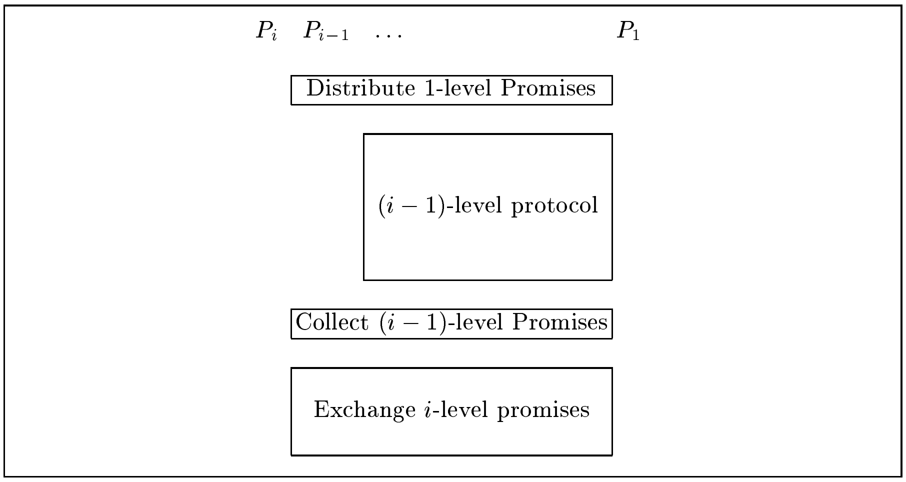

Fig. 2. *Secure contract-signing protocol: ith recursive level*

Protocol for participant P_i:

(1) **(Wait for all higher recursive levels to start)** P_i waits for 1-level promises from $P_{i+1}, \ldots, P_n$ on m. If it does not receive them in a timely manner, P_i simply quits.

(2) **(Start recursive level i)** P_i runs $\mathsf{PCS\text{-}Sign}_{P_i}((m,1), P_j, T)$ for $1 \leq j \leq i-1$ and sends each $\mathrm{PCS}_{P_i}((m,1), P_j, T)$ to P_j.

(3) **(Wait for recursive level $i-1$ to finish)** P_i waits for $(i-1)$-level promises from $P_1, \ldots, P_{i-1}$ on m. If it does not receive them in a timely manner, P_i runs the **Abort** procedure.

(4) **(Send i-level promises to all parties)** P_i runs $\mathsf{PCS\text{-}Sign}_{P_i}((m,i), P_j, T)$ for $1 \leq j \leq i-1$ and sends each $\mathrm{PCS}_{P_i}((m,i), P_j, T)$ to P_j.

(5) **(Finish recursive level i when i-level promises are received)** P_i waits for i-level promises from $P_1, \ldots, P_{i-1}$ on m. If it does not receive them in a timely manner, P_i runs the **Resolve** procedure.

(6) **(Complete all higher recursive levels)** For $a = i+1$ to n, P_i does the following:

(6.1) P_i runs $\mathsf{PCS\text{-}Sign}_{P_i}((m, a-1), P_j, T)$ and sends $\mathrm{PCS}_{P_i}((m, a-1), P_a, T)$ to P_a.

(6.2) P_i waits for a-level promises from $P_{i+1}, \ldots, P_a$ on m. If it does not receive them in a timely manner, P_i runs the **Resolve** procedure.

(6.3) P_i runs $\mathsf{PCS\text{-}Sign}_{P_i}((m,a), P_j, T)$ for $1 \leq j \leq i-1$ and sends each $\mathrm{PCS}_{P_i}((m,a), P_j, T)$ to P_j.

(6.4) P_i waits for a-level promises from $P_1, \ldots, P_{i-1}$ on m. If it does not receive them in a timely manner, P_i runs the **Resolve** procedure.

(6.5) P_i runs $\mathsf{PCS\text{-}Sign}_{P_i}((m,a), P_j, T)$ for $i+1 \leq j \leq a$ and sends each $\mathrm{PCS}_{P_i}((m,a), P_j, T)$ to P_j.

(7) P_i waits for signatures and $n+1$-level promises[2] from $P_{i+1}, \ldots, P_n$ on m. If it does not receive them in a timely manner, P_i runs the **Resolve** procedure.

(8) P_i runs $\mathsf{PCS\text{-}Sign}_{P_i}((m,n{+}1), P_j, T)$ and $\mathsf{S\text{-}Convert}_{P_i}(\mathrm{PCS}_{P_i}((m,1), P_j, T))$ for $1 \leq j \leq n$, and sends each result $(\mathrm{PCS}_{P_i}((m,n{+}1), P_j, T), \text{S-Sig}_{P_i}((m,1)))$ to P_j.

Protocol P_i-Abort: To abort, P_i sends $[m, P_i, (P_1, \ldots, P_n), \text{abort}]_{P_i}$ to T.

Protocol P_i-Resolve: For P_i to resolve, it runs $\mathsf{S\text{-}Convert}_{P_i}(\mathrm{PCS}_{P_i}((m,1), P_j, T))$ (for any $j \in \{1, \ldots, n\}$) to produce $\text{S-Sig}_{P_i}((m,1))$, and then sends the message

$$[\{\mathrm{PCS}_{P_j}((m,k_j), P_i, T)\}_{j \in \{1,\ldots,n\} \setminus \{i\}}, \text{S-Sig}_{P_i}((m,1))]_{P_i}$$

to T, where for $j > i$, k_j is the maximum level of a promise received from P_j on m, and for $j < i$, k_j is the maximum level of promises received from *all* participants $P_{j'}$, with $j' < i$. (For example, if the maximum level of the promises received by P_4 from P_3 and P_2 was 6, and the maximum level received by P_4 from P_1 was 5, then it would send the 5-level promises for P_1, P_2 and P_3.)

Protocol for T: For each contract[3] m with participants P_1 through P_n, when T learns about this contract (through an abort or receive message) it sets up a record that stores a boolean variable `validated(`m`)` which is true if the contract has been validated, and T has a full set of signatures (many would be converted by T from promises, most likely). If `validated(`m`)` is false, then T maintains the set S_m of indices of parties that have aborted, or resolved and failed to overturn a previous abort. T also maintains a set F_m of indices of parties that have not themselves aborted, but whom T will force to abort in certain situations. This set F_m is needed because at a particular point in our protocol, standard tests for whether to overturn an abort would allow unfairness. Specifically, they would allow groups of participants to conspire and receive a valid contract after a correct participant already received a cancellation from the T.

The protocol when T receives an abort message from P_i, $[m, P_i, (P_1, \ldots, P_n), \text{abort}]_{P_i}$, is as follows:

(1) If the signature is correct, and there has not been a positive resolution, T does the following. If i is larger than the maximum index in S_m, T clears

[2] We include these to simplify the TTP's protocol.

[3] Without loss of generality, we may assume all contracts are distinct.

F_m. In any case, T stores i in the set S_m. T sends $[[m, P_j, (P_1, \ldots, P_n), \text{abort}]_{P_j}]_T$ and $[m, S_m, \text{abort}]_T$ to P_i. If this is the first player in the list, T sets `validated(`m`)` to false, and stores $[[m, P_i, (P_1, \ldots, P_n), \text{abort}]_{P_i}]_T$.

(2) If there has been a positive resolution, T sends the valid (signed) contract to P_i, i.e., $\{\text{S-Sig}_{P_j}((m, k_j))\}_{j\in\{1,\ldots,n\}}$, where k_j is the level of the promise from P_j that was converted to a universally-verifiable signature.

The protocol when T receives a resolve message from P_i,

$$[\{\text{PCS}_{P_j}((m, k_j), P_i, T)\}_{j\in\{1,\ldots,n\}\setminus\{i\}}, \text{S-Sig}_{P_i}((m, 1))]_{P_i},$$

is as follows:

(1) If $i \in S_m$, T ignores the message.

(2) T checks to make sure all promises and signatures are consistent (i.e., m is the same, etc.) and valid. If not, then it ignores the message.

(3) If there has been no previous query to T on m, it sends

$$[\{\textsf{TP-Convert}_T(\text{PCS}_{P_j}((m, k_j), P_i, T))\}_{j\in\{1,\ldots,n\}\setminus\{i\}}]$$

to P_i, stores all the signatures, and sets `validated(`m`)` to true

(4) If there has been a positive resolution, T sends the stored valid contract to P_i, i.e., $\{\text{S-Sig}_{P_j}((m, k'_j))\}_{j\in\{1,\ldots,n\}}$, where k'_j is the level of the promise from P_j that was converted to a universally-verifiable signature.

(5) If there has not been a positive resolution, T runs as follows:

(5.1) If $i \notin F_m$, then

(5.1.1) if for any $\ell \in S_m$ there is a $j \in S_m$ such that $j > k_\ell$, then T adds i to S_m, and sends the stored abort message $[[m, P_j, (P_1, \ldots, P_n), \text{abort}]_{P_j}]_T$ and $[m, S_m, \text{abort}]_T$ to P_i. Let a be the maximum value in S_m. If $a > i$, then for all j with $k_j = a - 1$, T adds j to F_m. If $a = i$, T clears F_m.

(5.1.2) Otherwise, T sends

$$[\textsf{TP-Convert}_T(\text{PCS}_{P_j}((m, k_j), P_i, T))\}_{j\in\{1,\ldots,n\}\setminus\{i\}}]$$

to P_i, stores all the signatures, and sets `validated(`m`)` to true

(5.2) Else ($i \in F_m$)

(5.2.1) Let a be the maximum value in S_m. If for all $i < j \leq a$, $k_j > a$ and for all $j < i$, $k_j \geq a$, then T sends

$$[\textsf{TP-Convert}_T(\text{PCS}_{P_j}((m, k_j), P_i, T))\}_{j\in\{1,\ldots,n\}\setminus\{i\}}]$$

to P_i, stores all the signatures, and sets `validated(`m`)` to true

(5.2.2) Otherwise, T adds i to S_m, and sends the stored abort message $[[m, P_j, (P_1, \ldots, P_n), \text{abort}]_{P_j}]_T$ and $[m, S_m, \text{abort}]_T$ to P_i. Let a be the maximum value in S_m. If $a > i$, then for all j with $k_j = a - 1$, T adds j to F_m. If $a = i$, T clears F_m.

5 Protocol Correctness

Theorem 2. *Assuming the security of* DDH*, the protocol above is a secure optimistic contract-signing protocol (in the random oracle model), with TTP-invisibility if the* PCS *scheme is third-party invisible, and TTP-accountability if the* PCS *scheme is third-party accountable.*

Proof. Recall that we say that an optimistic contract-signing protocol is secure if it is complete, fair and abuse-free.

Complete: Completeness follows directly from the definition of private contract signatures.

Fair: To show fairness for P_i we must prove two results:

(1) If any other party or parties can obtain a valid contract on m, so can P_i, and

(2) If P_i gets a cancellation from T, then no other party or parties can obtain a valid contract.

Result (1) can be seen from the following two cases:

(1) Say P_i has sent out a universally-verifiable signature on m. Then P_i has $(n+1)$-level promises from P_{i+1} through P_n, and n-level promises from P_1 through P_{i-1}, so P_i can contact T and T will overturn any set of aborts S_m, even if $i \in F_m$.

(2) Say P_i has not sent out a universally-verifiable signature on m. Then by the security of the PCS, for some party to obtain a valid contract, some P_j would have to obtain the contract from T. Then P_i can send an abort or resolve message to T and obtain the contract also.

Result (2) can be seen from the following cases:

(1) Say P_i sends an abort to T in Step 3 of the protocol and receives a signed abort message back from T. Since P_i is correct, $i > 1$, parties P_1 through P_{i-1} have a 1-level promise from P_i, and parties P_{i+1} through P_n have no promise from P_i. Since T is honest, T stores i in S_m and will never allow any resolve request to overturn the abort, since any resolve request will have k_i (the level of i's promise) at most 1, and thus $i > k_i$.

(2) Say P_i sends a resolve request to T in Step 5 of the protocol and receives a signed abort message back from T. Since P_i is correct, P_i must have sent $(i-1)$-level promises to T from P_1 through P_{i-1}, and P_1 through P_{i-1} have at most i-level promises from P_i. Since T is honest, there

must be a $j \in S_m$ with $j > i$, so T stores i in S_m and will never allow any resolve request to overturn the abort, since any resolve request will have k_i (the level of i's promise) at most i, and thus $j > k_i$.

(3) Say P_i sends a resolve request to T in Step (6.2) of the protocol and receives a signed abort message back from T. Since P_i is correct, P_i must have sent $(a-1)$-level or a-level promises to T from P_1 through P_{i-1} and P_{i+1} thorugh P_{a-1}, and P_1 through P_{i-1} and P_{i+1} through P_a have at most $(a-1)$-level promises from P_i. Since T is honest, there must be a $j \in S_m$ with $j \geq a$, so T stores i in S_m and will never allow any resolve request to overturn the abort, since any resolve request will have k_i (the level of i's promise) at most $a-1$, and thus $j > k_i$.

(4) Say P_i sends a resolve request to T in Step (6.4) of the protocol and receives a signed abort message back from T. Since P_i is correct, P_i must have sent to T (1) a-level promises from P_1 through P_{i-1} and (2) $(a-1)$-level promises from P_{i+1} thorugh P_{a-1}. Also, P_1 through P_{i-1} have at most a-level promises from P_i, and P_{i+1} through P_a have at most $(a-1)$-level promises from P_i. Since T is honest, there are two cases:

(4.1) There is a $j \in S_m$ with $j > a$, so T stores i in S_m and will never allow any resolve request to overturn the abort, since any resolve request will have k_i (the level of i's promise) at most a, and thus $j > k_i$.

(4.2) There is no $j \in S_m$ with $j > a$, but $a \in S_m$, and there is also a $j' \in S_m$ with $j < i$. In this case, T stores i in S_m and all $j < i$ in F_m, and will never allow any resolve request to overturn the abort, since any resolve request will either have k_i at most $a-1$ (and thus $a > k_i$), or will have $k_i = a$ and be from a party with index in F_m.

Abuse-free: To show abuse-freeness for P_i we must show that no party obtains publicly verifiable information about (honest) P_i signing the contract until P_i has the power to validate the contract over any other parties' aborts. This follows by the designated verifier property of PCS, and the fact that once P_i sends S-Sig$_{P_i}((m, 1))$, P_i will have received $(n+1)$-level promises from P_{i+1} through P_n and n-level promises from P_1 through P_{i-1}, so P_i either has all the signatures from all other parties, or could contact T and obtain a valid contract over any other parties' aborts.

TTP-Invisibility or TTP-Accountability: Straightforward from the definitions.

□

6 Efficiency

By inspection of the recursive construction, it is easy to see that the number of messages required by our protocol is $O(n^3)$, and the number of rounds is $O(n^2)$.

We now show a lower bound in the number of rounds which holds even if the contract-signing protocol is not abuse-free.

Theorem 3. *Any complete and fair optimistic contract-signing protocol with n participants requires at least n rounds in an optimistic run.*

Proof. (Sketch) Since the optimistic protocol is complete, there must be a way for the n participants to obtain a universally-verifiable contract without contacting the TTP. Thus there must be some party, say P_1[4] that sends a message during some round containing information that can be used along with information provided by other participants to validate the contract. At this point, since the protocol is fair, P_1 must have received messages in previous rounds from other participants, say P_2 through P_n, such that regardless of the actions of those participants, P_1 could send a message to the TTP and obtain a validated contract.

Specifically, there must be a previous round in which a participant, say P_2, sends a message to P_1 that allows this. At this point, since the protocol is fair, P_2 must have received messages in previous rounds from participants P_3 through P_n so that regardless of the actions of those participants P_2 could send a message to the TTP and obtain a validated contract. (Otherwise, P_2 could not obtain a validated contract, but P_1 would be able to (possibly much later), and thus the protocol would not be fair for P_2.)

In a similar manner, given that a set of participants, say P_1 through P_i have received messages so that for each $j \in \{1, \ldots, i\}$, participant P_j could send a message to the TTP and obtain a validated contract regardless of the actions of participants P_{j+1} through P_n, there must be a previous round in which a participant, say P_{i+1}, sends the message that allows this to P_i.

Therefore, by an inductive argument, we show the number of rounds is at least n. □

7 Conclusions

In this paper we presented an optimistic abuse-free contract-signing protocol for an arbitrary number of participants n. The protocol tolerates an arbitrary number of faulty (Byzantine) parties. and requires $O(n^3)$ messages and $O(n^2)$ rounds of interaction (in an asynchronous environment). Waidner [26] recently suggested a way to add abuse freeness to any optimistic contract signig protocol using verifiable encryption. Applying this transformation to the protocol of [5] yields an abuse-free optimistic contract signing protocol requiring $O(n)$ rounds.

[4] The indices here are unrelated to the indices used in the protocol of Section 4

References

1. N. Asokan, B. Baum-Waidner, M. Schunter, and M. Waidner. Optimistic synchronous multi-party contract signing. IBM Research Report RZ3089, 1998.
2. N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *ACM Conf. on Computer and Comm. Security '97*, pp. 6–17.
3. N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *IEEE Symp. on Research in Security and Privacy*, pp. 86–99, 1998.
4. N. Asokan, V. Shoup, and M. Waidner. Fair exchange of digital signatures. In *EUROCRYPT '98*, pp. 591–606.
5. B. Baum-Waidner and M. Waidner. Optimistic asynchronous multi-party contract signing. IBM Research Report RZ3078, 1998.
6. M. Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems*, 1(2):175–193, May 1983.
7. M. Blum. Coin flipping by telephone: A protocol for solving impossible problems. In *CRYPTO 81*, pp. 11–15.
8. H. Bürk and A. Pfitzmann. Value exchange systems enabling security and unobservability. *Computers and Security*, 9:715–721, 1990.
9. D. Chaum. Blind signatures for untraceable payments. In *CRYPTO 82*, pp. 199–203.
10. D. Chaum. Designated confirmer signatures. In *EUROCRYPT 94*, pp. 86–91.
11. B. Cox, J. D. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *First USENIX Workshop on Electronic Commerce*, pp. 77–88, 1995.
12. I. Damgård. Practical and provably secure release of a secret and exchange of signatures. *J. of Cryptology*, 8(4):201–222, Autumn 1995.
13. R. Deng, L. Gong, A. Lazar, and W. Wang. Practical protocols for certified electronic mail. *J. of Network and Systems Management*, 4(3), 1996.
14. S. Even. A protocol for signing contracts. *ACM SIGACT News*, 15(1):34–39, 1983.
15. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, June 1985.
16. M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed commit with one faulty process. *Journal of the ACM*, 32(2), 1985.
17. M. Franklin and M. Reiter. Fair exchange with a semi-trusted third party. In *ACM Conf. on Computer and Comm. Security '97*, pp.1–5.
18. J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. To appear in *CRYPTO 99*, August 1999.
19. O. Goldreich. A simple protocol for signing contracts. In *CRYPTO 83*, pp. 133–136.
20. M. Jakobsson, K.Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In *EUROCRYPT 96*, pp. 143–154.
21. S. Micali. Certified e-mail with invisible post offices. Presented at the 1997 RSA Security Conference, 1997.
22. M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.
23. B. Pfitzmann, M.Schunter, and M. Waidner. Optimal efficiency of optimistic contract signing. In *PODC '98*, pp. 113–122.
24. C. P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO'89*, pp. 239–252.
25. A. Silberschatz, H. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, 1997.
26. M. Waidner. Personal communication.