RZ 3078 (#93124) 11/23/1998
Computer Science/Mathematics 11 pages

Research Report

Optimistic Asynchronous Multi-Party Contract Signing

Birgit Baum-Waidner[1], Michael Waidner[2]

[1] Entrust Technologies Europe
CH-8301 Glattzentrum/Zürich
Switzerland
birgit.baum@entrust.com

[2] IBM Zurich Research Laboratory
Säumerstrasse 4
CH-8803 Rüschlikon
Switzerland
wmi@zurich.ibm.com

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

IBM Research Division
Almaden · Austin · Beijing · Haifa · T.J. Watson · Tokyo · Zurich

Optimistic Asynchronous Multi-Party Contract Signing

Birgit Baum-Waidner
Entrust Technologies Europe
baum.baum@entrust.com

Michael Waidner
IBM Zurich Research Laboratory
wmi@zurich.ibm.com

Abstract

A contract is a non-repudiable agreement on a given contract text, i.e., it can be used to prove unanimous agreement between its signatories to any verifier. A contract signing protocol is used to *fairly* compute a contract so that, even if $n-1$ of the n signatories misbehave, either all or none of them obtain a contract.

Optimistic contract signing protocols use a third party to ensure fairness, but in such a way that the third party is not actively involved in case all parties are honest. Since no satisfactory protocols without any third party exist, this seems to be the best one can hope for.

We present the first optimistic multi-party contract signing protocol for *asynchronous* networks. Previous constructions supported synchronous networks only.

We show how any such contract signing protocol can be used to construct an optimistic, *perfectly* fair general function evaluation protocol for dishonest majorities. Previous constructions did not use a third party at all but achieved fairness in a probabilistic sense only.

1 Introduction

A *contract* is a non-repudiable agreement by the contract signatories on a given contract text. A contract signing protocol is supposed to securely compute a contract [Blum81]:

- *Unforgeability.* If at least one of the signatories does not wish to sign then the whole contract shall be void.
- *Fairness.* Even if up to $n-1$ of the n signatories are dishonest, either *all* signatories or *none* of them must obtain a signed contract.

Contract signing has obvious applications in secure electronic commerce. But it is also a useful primitive in the solution of other fairness problems. For instance, the problem of sending certified mail to a set of recipients, simultaneously, can be efficiently reduced to contract signing [ABSW98].

The contract signing problem was first described in [Blum81], for the case of two signatories. The notion naturally generalizes to the n-party case, although the first multi-party protocols have been introduced only recently [AsSW296, ABSW98].

The most obvious contract signing protocol uses a trusted third party T to ensure fairness [Rab183]: Each signatory digitally signs the contract and sends the signature to T. T waits until all signatures arrived, and redistributes them to all signatories. If some signature does not arrive then T might cancel the protocol, after some time. Fairness depends fully on the trustworthiness and availability of T. Therefore most research on contract signing has been focused on getting rid of T as trust and performance bottleneck.

No 2-party protocol can achieve deterministically fair 2-party contract signing [EvYa80], and under reasonable assumptions one can show that each 2-party contract signing protocol must have an error probability at least linear in the number of rounds of communication [BGMR90]. These results apply also to the n-party case, unless one weakens the requirement "any number $t < n$ of the signatories can be dishonest" to "any number $t < n/2$."

Optimistic contract signing protocols are a compromise between both approaches [BGMR90, AsSW97, Mica97]: They depend on a trusted third party T, but in such a way that T is *not* actively involved in case all parties are honest. Only for recovery purposes T might become active. Several optimistic contract signing protocols have been proposed recently, making different assumptions on the network synchronization:

- Protocols for *synchronous* networks have been proposed in [Mica97, AsSW97, BaDM98, PfSW98] for the 2-party case, and in [AsSW296, ABSW98] for the n-party case.

  Basically, *"synchronous"* means that all parties have synchronized real-time clocks, and that there is a known upper bound on the network delays.

- Protocols for *asynchronous* networks have been proposed in [AsSW98, AsSW198, PfSW98]. So far no multi-party protocol was known.

  "Asynchronous" means that we make no assumption on clock synchronization or network delays. We only assume that the network reliably transports messages between honest parties, i.e., all messages sent are delivered, eventually.

In Section 2 we give a precise definition of multi-party contract signing, and in Section 3 we describe the first optimistic protocol for multi-party contract signing on *asynchronous* networks, and prove its security.

Our protocol requires at most $t+5$ rounds of communication and $O(tn^2)$ messages. A variant requires $2t+7$ rounds but only $O(tn)$ messages. The *synchronous* protocol proposed in [ABSW98] needs 6 rounds and $6n-4$ messages only. The additional factor of t in the complexity of our protocol seems to be unavoidable if one wants to support asynchronous networks.

As an application of multi-party contract signing, we investigate how to achieve *perfect* fairness for general secure multi-party function evaluation.

Secure 2-party function evaluation was introduced in [Yao82], and generalized to the n-party case in [GMW87]: There are n parties, party P_i holding a secret value x_i. Cooperatively they want to compute a function $f(x_1, ..., x_n)$ such that each party gets the correct result, y, but dishonest parties neither learn anything about the honest parties' secrets (except what is implicit in the final result, y), nor can they choose their own inputs depending on the inputs of the honest parties. A secure function evaluation protocol can compute *any* such function. (For formal definitions see [Beav591, GoLe91, MiRo92, Cane96].)

Secure function evaluation protocols have been proposed that tolerate any dishonest *minority* [GMW87, BeGW88, RaBe89, HiMa197]. Necessarily a *majority* of dishonest parties can prevent any such protocol from yielding a result. But like for contract signing, one might expect that such a protocol is at least *fair:* the dishonest parties should not get any information about the result (beyond what is implicitly known from their own inputs) unless all honest parties receive this result completely.

Contract signing can be seen as a special instance of this problem. This implies that there is no perfectly fair secure function evaluation protocol. Approximations of fairness are described in [BeGo190, GaHY88, GoLe91].

In Section 4 we describe how to add *perfect* fairness to any general protocol for secure function evaluation that ensures correctness and secrecy and tolerates dishonest majorities. The result is an optimistic protocol for secure function evaluation.[1]

2 Definition of Multi-Party Contract Signing

2.1 Model and Notation

Let $P_1, ..., P_n$ denote the parties directly involved in the contract signing, T a trusted third party, and V any verifier. For simplicity we do not distinguish between a party and his or her protocol machine. We simply say a "party makes an input" ("receives an output") if the corresponding user makes the input to (receives an output from) the corresponding machine.

We consider a static *adversary* who can a priori choose to corrupt up to t (for a given $t < n$)[2] of the n parties, and for some requirements also T. The adversary has full control over the behavior of corrupted parties. The case where all n parties $P_1, ..., P_n$ are honest is called the *all-honest case.*

Each party is able to digitally *sign* messages, and to verify signatures of any other party [DiHe76, GoMR88]. The signature on message m associated with P_X is denoted by $\text{sign}_X(m)$. We assume that the length of $\text{sign}_X(m)$ does not depend on the length of m (e.g., because it is hashed before signing [MeOV97]). We abstract from the error probabilities introduced by cryptographic signature schemes and assume that the adversary cannot forge signatures.

We assume a *reliable network:* all sent messages are delivered eventually.[3] The adversary can read all messages from all channels, and can insert additional messages into all channels. But he can neither alter nor delete sent messages.

We do not require any particular level of synchronization, nor do we assume that messages are delivered in order. The decision on which of all sent messages to deliver next is taken by a fictitious machine called *scheduler.* The adversary has full control over the scheduler, i.e., we do not assume anything more about it than fairness of message delivery [Cane96].

Some security requirements assume that "there exists a scheduler such that a certain condition is satisfied." This means more precisely that if this scheduler is chosen to schedule the network then, regardless of how the adversary behaves otherwise, this condition is satisfied.

If we say that "a player P can decide to stop waiting for a certain event, E," we mean more precisely the following: before P starts waiting for E it sends a message $\textit{timeout}(E)$ to itself, and starts waiting for both, E and $\textit{timeout}(E)$. Whichever of both happens first causes P to stop waiting for the other.

[1] Independently and concurrently to our work, Silvio Micali proposed an optimistic protocol for perfectly fair, secure *2-party* function evaluation. Both protocols were presented at the 1998 Weizmann Workshop on Cryptography, June 16-18th, Weizmann Institute of Science, Rehovot, Israel. There are no proceedings of this workshop. Micali's proposal does not cover the n-party case, and is more complex than ours.

[2] For real-life contract signing, only $t = n - 1$ makes sense. We decided to keep t as a variable since the complexity of our contract signing protocol scales with t, and some of the protocols that are enabled by ours might tolerate only smaller numbers of t.

[3] Reliable communication between *signatories* is needed only to ensure that T is not involved if all parties are honest. Otherwise it is sufficient if each signatory can reliably communicate with T.

Disclaimer. The following protocols are idealized, in several ways. In particular we omit all protocol and message identifiers in messages, all public key certificates, and even parameters such as the current values of t and n and the identity of T.

2.2 Definitions

Definition 1. (Multi-party Contract Signing)
A *Multi-Party Contract Signing Protocol* (MPCS) consists of two protocols, sign$[P_1, ..., P_n]$ and verify$[P_i, V]$. sign[] might involve an additional party, T, in which case we call it an *MPCS with third party.*[4]

The machine V does *not* keep state between different executions of verify[] but always starts with the state it had immediately after initialization of the signature scheme.[5]

A party P_i who wishes to start sign[] enters $(\mathsf{sign}, \mathit{tid}, \mathit{contr}, \mathit{decs})$. tid is a transaction identifier unique for all executions of sign[]. contr is the contract to be signed.[6] $\mathit{decs} \in \{\mathsf{sign}, \mathsf{reject}\}$ denotes the user's initial decision to sign or to reject the contract. Upon termination sign[] produces an output $(\mathit{tid}, \mathit{contr}, d_i)$ for P_i, with $d_i \in \{\mathsf{signed}, \mathsf{failed}\}$. We will simply say "$P_i$ decides d_i."[7]

A player P_i who wishes to start verify[] with a verifier V enters $(\mathsf{show}, \mathit{tid}, \mathit{contr})$. If the verifier, V, wishes to start verify[] as well it enters $(\mathsf{verify}, \mathit{tid}, \mathit{contr})$ where $\mathit{tid}, \mathit{contr}$ must be the same values as those used by P_i. Upon termination verify[] produces an output $(\mathit{tid}, \mathit{contr}, d_V)$ with $d_V \in \{\mathsf{signed}, \mathsf{failed}\}$ for V. We will simply say "V decides d_V." No output is produced by P_i.

The following requirements must be satisfied:

(R1) *Correct execution.* There exists a scheduler such that if all parties P_i are honest and successfully started with (sign, tid, contr, $\mathit{decs} = \mathsf{sign}$) then all parties will decide signed.

(R2) *Unforgeability of contract.* If honest P_i never entered $(\mathsf{sign}, \mathit{tid}, \mathit{contr}, \mathit{decs})$ with $\mathit{decs} = \mathsf{sign}$ then any honest verifier that enters $(\mathsf{verify}, \mathit{tid}, \mathit{contr})$ will decide failed. (Note that this does not assume an honest T.)

(R3) *Verifiability of valid contracts.* There exists a scheduler such that if honest P_i decides signed on input (sign, tid, contr, decs), and later inputs (show, tid, contr) and honest V inputs (verify, tid, contr) then V will decide signed.

(R4) *No surprises with invalid contracts.* If T is honest, and honest P_i entered successfully (sign, tid, contr, $\mathit{decs} = \mathsf{sign}$) but decided failed then for any contr_V no honest verifier V entering $(\mathsf{verify}, \mathit{tid}, \mathit{contr}_V)$ will decide signed.

(R5) *Termination of* sign[]. If T is honest then each honest P_i that enters sign will terminate eventually.

(R6) *Termination of* verify[]. Each honest V that enters verify and each honest P_i that enters show will terminate eventually. ◊

Definition 1 implies that MPCS is a consensus problem [Lync96]: The protocol *terminates.* All honest parties *agree* on a common decision. It satisfies a *non-trivial validity* requirement, namely, if at least one honest party starts with reject then all parties will decide failed, and there is a scheduler such that if all parties are honest and start with sign they will all terminate with signed.

But contract signing is more than just a new variant of distributed consensus: it requires that the decision signed is *verifiable* by any third party. (failed is not verifiable, and it is easy to see that not both decisions can be verifiable if $t > n/2$.)

[4] verify[] never involves T, i.e., it is always a 2-party protocol only.

[5] This models the fact that P_i shall be able to convince *any* verifier, i.e., V does not need to have any a priori knowledge about the contract.

[6] tid and contr must be chosen before the protocol can be started. tid might be randomly chosen by one party, say, P_1, and proposed to all others. In case the set of parties involved is not clear anyway, tid also specifies this set.

[7] In MPCS with third party, T does neither get an input nor produces an output, as its behavior is completely determined by the protocol.

Definition 2. (Optimistic Protocol)

A protocol for n regular parties $P_1, ..., P_n$ and a third party T is called *optimistic* if there exists a scheduler such that in the all-honest case the protocol terminates without T ever sending or receiving any messages.

A multi-party contract signing protocol is called *optimistic on agreement* if it is optimistic in case all players agree on $(tid, contr, decs = \mathsf{sign})$. It is called *optimistic on disagreement* if it is optimistic in case some players disagree initially. ◇

The following theorem shows that for contract signing it is sufficient to ensure being optimistic on agreement:

Theorem 1

Any MPCS that is optimistic on agreement can be transformed into an MPCS that is optimistic in all cases. The transformation adds one round to the protocol. ◇

Proof.

Assume any MPCS that is optimistic on agreement. We add one round at the beginning where each player signs its input and sends it to all other players. Each player waits for the full vector of n signed inputs but might decide to stop waiting at any point of time. If a player stopped waiting before having seen the full vector, or if it has received the full vector but can conclude that the final decision based on it would be failed then it sets $decs_i := \mathsf{reject}$ and behaves like a machine that never received an input $(\mathsf{sign}, tid, contr', decs')$ for any $contr', decs'$. Otherwise it stays with the original input, $(\mathsf{sign}, tid, contr, decs_i = \mathsf{sign})$.

Eventually all honest players will finish this first round, and those with $decs_i = \mathsf{sign}$ will start the real contract signing. Obviously there exists a scheduler such that if all parties are honest and agree initially they will all start the MPCS with $decs_i = \mathsf{sign}$, and if some disagree they will all stop after the first round with $decs_i = \mathsf{reject}$, i.e., they will not start the MPCS at all. □

3 Asynchronous Optimistic Contract Signing

3.1 Protocol

The following Protocol 1 solves the multi-party contract signing problem, and is optimistic on agreement (which is sufficient according to Theorem 1).

Ignoring all details, the protocol works as follows: It consists of $t + 2$ locally defined rounds. In Round 1 each party that starts with sign signs a "promise" to sign the contract and broadcasts this promise. In each subsequent round each party collects all signatures from the previous round, countersigns this set of n signatures, and broadcasts it.[8]The result of the $(t + 2)$-nd round becomes the real contract.

A party who becomes tired of waiting for some signatures in some round can call the third party. T analyses the situation and decides either failed or signed:

If the first request received by T comes from a party in the *first* round then T must decide failed – T cannot know whether some parties might have started the protocol with reject. If T receives a request from a party in the *last* round then T must decide signed, as other parties might already have the signed contract.

Thus if T receives multiple requests then somewhere in the middle between the first and the last round T might have to change the decision from failed to signed. The problem is that T can do this only if *all* parties that received failed before are provably dishonest. Therefore we need $t + 2$ rounds:

Assume that the dishonest parties call T one after the other, starting with Round 1. Say, for some $s \leq t$ each $P_i, i = 1, ..., s$, is dishonest and calls T in Round i. P_1's request must be answered with failed, as already explained. For $i > 2$, the request by P_i shows that $P_{i-2}, ..., P_1$ are all dishonest (basically, this is Lemma 2): P_i calling in Round i means that P_i has seen *all* messages of Round $i - 1$, as otherwise P_i would have called T already then. Those include messages from $P_{i-2}, ..., P_1$, which could not exist if those would have stopped in Round $i - 2, ..., 1$, as supposed. Thus whenever T receives a request for a certain Round i, $i \geq 2$, such that it has *not* answered a request for Round $i - 1$ yet then it can safely switch from failed to signed.

[8]The real protocol does this a bit more efficiently, in order to keep the protocol messages short. This is the reason why we are using two vectors $M_{i,r}$ and $X_{i,r}$ instead of just one containing the contents of both.

In the worst case $s = t$: If T receives a request in Round $t+2$ then it knows that the first t requesters are exactly all the dishonest parties. If there was no request in Round $t+1$ then T can safely switch to signed. If there was a request in Round $t+1$ then it knows that at least one honest party did not finish this round, thus *no* honest party will finish Round $t+2$ and will receive the normal contract, and thus sticking to failed is safe.

Theorem 3 shows that this reasoning is complete, i.e., results is a secure multi-party signing protocol.

Protocol 1 (Asynchronous Optimistic Multi-party Contract Signing)

Protocol "sign" for honest P_i:

The protocol is given by the following rules; P_i applies them, sequentially, until it *stops*. Let $c_i := (tid_i, contr_i)$.

The protocol will proceed in locally defined rounds. Let $r := 1$ a local round counter for P_i, and let $raised\_exception := \mathsf{false}$ a Boolean variable. Both are initialized before executing any rule. Let $M_{0,i} := \mathsf{nil}$, for all i.

- **Rule S0:** If $r = 1$ and $decs_i = \mathsf{reject}$ then P_i decides failed and *stops*.
- **Rule S1:** If $raised\_exception = \mathsf{false}$ and $r = 1$ and $decs_i = \mathsf{sign}$ then:
  - P_i sends $m_{1,i} := \mathrm{sign}_i(c_i, 1, 1)$ to all parties.
  - From all received messages of type $m_{1,j}$ it tries to compile full and consistent vectors

    $$M_{1,i} := (m_{1,1}, ..., m_{1,n}) \text{ and } X_{1,i} := M_{1,i}$$

    with $m_{1,j} = \mathrm{sign}_j(c_i, 1, 1)$. If this succeeds P_i sets $r := 2$.

  At any time P_i can decide to stop waiting for any missing $m_{1,j}$, in which case it sets $raised\_exception := \mathsf{true}$ and sends $resolve_{1,i} = (1, i, \mathrm{sign}_i(m_{1,i}, \mathsf{resolve}))$ to T.
- **Rule S2:** If $raised\_exception = \mathsf{false}$ and $2 \leq r \leq t+2$ then:
  - P_i sends $m_{r,i} := (\mathrm{sign}_i(M_{r-1,i}, r, 2), \mathrm{sign}_i(c_i, r, 1))$ to all parties.
  - From all received messages of type $m_{r,j}$ it tries to compile full and consistent vectors

    $$M_{r,i} = (\mathrm{sign}_1(c_i, r, 1), ..., \mathrm{sign}_n(c_i, r, 1))$$
    $$X_{r,i} := (\mathrm{sign}_1(M_{r-1,i}, r, 2), ..., \mathrm{sign}_n(M_{r-1,i}, r, 2))$$

    If this succeeds and

    $r < t+2$ then it sets $r := r+1$.
    $r = t+2$ then it decides signed, sets $C_i := M_{r,i}$, and *stops*.

  At any time P_i can decide to stop waiting for any missing $m_{r,j}$. In this case P_i sets $raised\_exception := \mathsf{true}$ and sends $resolve_{r,i} := (r, i, \mathrm{sign}_i(X_{r-1,i}, \mathsf{resolve}), X_{r-1,i}, M_{r-2,i})$ to T.
- **Rule S3:** If $raised\_exception = \mathsf{true}$ then:

  P_i waits for a message from T. This can be any of $signed_{r,i} = \mathrm{sign}_T(c_i, r', j, \mathsf{signed})$ or $aborted_{r,i} = \mathrm{sign}_T(c_i, r, i, \mathsf{aborted})$. On receiving one P_i decides as indicated by T. If it decides signed it sets $C_i := signed_{r,i}$.

Protocol "sign" for third party T:

We assume T receives a message $resolve_{r,i}$, as defined in the protocol for P_i. Let c be the (unique) triple $(tid, contr)$ contained in $resolve_{r,i}$.

If this is the first time T is asked about this contract then T initializes a Boolean variable $signed := \mathsf{false}$ and two sets $con := \emptyset$ and $abort\_set := \emptyset$. Processing $resolve_{r,i}$ cannot be interrupted, i.e., it cannot happen that T processes two different $resolve_{r,i}$ concurrently.

- **Rule T0:** If $i \in con$ then the message $resolve_{r,i}$ is ignored.
- **Rule T1:** If $i \notin con$ and $signed = \mathsf{false}$ and $r = 1$ then T sets

  $aborted_{r,i} := \mathrm{sign}_T(c, r, i, \mathsf{aborted})$
  $abort\_set := abort\_set \cup \{aborted_{r,i}\}$
  $con := con \cup \{i\}$

  and sends $aborted_{r,i}$ to P_i.
- **Rule T2:**
  If $i \notin con$, $signed = \mathsf{false}$, $r > 1$, and for all $aborted_{s,k} \in abort\_set$ we have $s < r - 1$
  - then: If $con = \emptyset$ then T sets $first\_signed := (resolve_{r,i}, \mathrm{sign}_T(c, r, i, \mathsf{signed}))$. T sets
    $signed_{r,i} := first\_signed$
    $signed := \mathsf{true}$
    $con := con \cup \{i\}$
    and sends $signed_{r,i}$ to P_i.
  - else: T sets
    $aborted_{r,i} := \mathrm{sign}_T(c, r, i, \mathsf{aborted})$
    $abort\_set := abort\_set \cup \{aborted_{r,i}\}$
    $con := con \cup \{i\}$
    and sends $aborted_{r,i}$ to P_i.
- **Rule T3:** If $i \notin con$ and $signed = \mathsf{true}$ then T sets

  $signed_{r,i} := first\_signed$
  $con := con \cup \{i\}$

  and sends $signed_{r,i}$ to P_i.

Protocol "verify":
If P_i wants to show a signed contract on c to verifier V it sends C_i to V. V decides signed if it receives a messages C_i from P_i such that

- **Rule V1:** $C_i = (\mathrm{sign}_1(c, t+2, 1), ..., \mathrm{sign}_n(c, t+2, 1))$, or
- **Rule V2:** $C_i = ((2, j, \mathrm{sign}_j(X_{1,j}, \mathsf{resolve}), X_{1,j}, M_{0,j}), \mathrm{sign}_T(c, 2, j, \mathsf{signed}))$,
  for some j, $X_{1,j} = (\mathrm{sign}_1(c, 1, 1), ..., \mathrm{sign}_n(c, 1, 1))$ and $M_{0,j} = \mathsf{nil}$, or
- **Rule V3:** $C_i = ((r, j, \mathrm{sign}_j(X_{r-1,j}, \mathsf{resolve}), X_{r-1,j}, M_{r-2,j}), \mathrm{sign}_T(c, r, j, \mathsf{signed}))$,
  for some $r > 2$, some j, $X_{r-1,j} = (\mathrm{sign}_1(M_{r-2,j}, r-1, 2), ..., \mathrm{sign}_n(M_{r-2,j}, r-1, 2))$ and $M_{r-2,j} = (\mathrm{sign}_1(c, r-1, 1), ..., \mathrm{sign}_n(c, r-1, 1))$.

Otherwise, or if V decides to stop waiting for C_i, it decides failed. $\diamond$

Lemma 2
Consider Protocol 1: If T receives $resolve_{r,i}$ and finds $aborted_{s,k} \in abort\_set$ for an $s \le r - 2$ then P_k is dishonest. $\diamond$

Proof.
Assume T finds $aborted_{s,k} \in abort\_set$ with $s \le r - 2$. Since $s \ge 1$ we have $r \ge 3$, and therefore $resolve_{r,i}$ includes $\mathrm{sign}_k(M_{r-2,k}, r, 2)$, taken from $m_{r-1,k}$. Thus P_k participated in Round $r - 1$, and since $r - 1 > s$ this means that P_k was still active after having sent $resolve_{s,k}$ to T. Thus P_k is dishonest. $\square$

Theorem 3 (Asynchronous optimistic multi-party contract signing)
Protocol 1 is an MPCS for asynchronous networks for any $t < n$ and honest T. It is optimistic on agreement and terminates in $t+2$ rounds if T is not involved, and in $t+4$ rounds in the worst case. $\diamondsuit$

Proof.
Correct execution, termination of verify and *optimistic on agreement* are all obviously satisfied.
Verifiability. The definitions of C_i in the signing protocol for P_i satisfy the conditions checked by V.
Termination of sign[]. For each player the protocol proceeds in $t+2$ rounds, and each round terminates, either because the full vector of n messages arrived that allows to enter the next round, or because T is asked and will eventually send an answer (which results in a total number of $t+4$ rounds if T is asked in the last round).
No surprises with invalid contracts. Assume honest P_i started with $decs = \mathsf{sign}$, decided failed, and some honest verifier V decides signed.

- Assume V decided because of Rule V1. This implies that P_i sent $\mathrm{sign}_i(c, t+2, 1)$ as part of $m_{t+2,i}$. Thus P_i asked T in Round $t+2$ and received $aborted_{t+2,i}$. According to T's rules this means that there is some party $P_{k_{t+1}}$ that received $aborted_{t+1,k_{t+1}}$. Inductively we can show (using T2) that for all rounds $s \in \{1, ..., t+1\}$ there is one party P_{k_s} that received $aborted_{s,k_s}$. Because of Lemma 2 we know that all parties P_{k_s} with $s \in \{1, ..., t\}$ are dishonest, and as there are at most t dishonest parties we know that $P_{k_{t+1}}$ must be honest. Since $P_{k_{t+1}}$ asked T in Round $t+1$ it did not send its message for Round $t+2$. Thus no party receives all information that is necessary to construct a signed contract, which contradicts our assumption.

- Assume V decides because of Rule V2 or Rule V3, seeing a message $resolve_{r,j}$. Thus P_i asked T in some Round s with $s < r$ and received $aborted_{s,i}$. As P_i is honest we know that $s \geq r-1$ (Lemma 2). But this contradicts Rule T2 of T, i.e., if P_i received $aborted_{s,i}$ then $\mathrm{sign}_T(c, r, i, \mathsf{signed})$ cannot exist. Again we have a contradiction.

Unforgeability of contract. All variants of a valid contract contain some pieces signed by all players, and these signatures exist only if all players started with $decs = \mathsf{sign}$. $\square$

The protocol is optimistic on agreement only: If P_i starts with $decs_i = \mathsf{sign}$ and P_k starts with $decs_k = \mathsf{reject}$ then P_i will send $resolve_{1,i}$ to T. We can transform the protocol into a fully optimistic one as described in Theorem 1.

3.2 Number of Messages and Rounds

Let C_s and C_b be the costs of a single and a broadcast message, respectively.

In the optimistic case, Protocol 1 runs in $t+2$ rounds where each player broadcasts one message to all other players, resulting in costs of $(t+2)nC_b$. In the worst case each party might have one message exchange with T, resulting in $t+4$ rounds and costs of $(t+2)nC_b + 2nC_s$. Finally the transformation of Theorem 1 adds another round of n broadcasts, resulting in $t+5$ rounds and costs of $(t+3)nC_b + 2nC_s$, in the worst case.

If one assumes that each broadcast requires $n-1$ single messages we end up with costs of $((t+3)n(n-1) + 2n)C_s = O(tn^2)C_s$.

Alternatively we can replace each round of n simultaneous broadcasts by 2 rounds and n single messages. This will result in $2(t+3)+2$ rounds and costs of $((t+3)n + 2n)C_s = O(tn)C_s$: Each broadcast round, say, Round r, is implemented by two mini-rounds: in Mini-Round r_1 each party P_i with $i > 1$ sends its message to P_1 only. P_1 collects the full vector of n messages, as it would do in the original protocol. If this succeeds it sends this vector to all other players in Mini-Round r_2, which concludes Round r. As before, everybody, in particular P_1, can stop waiting any time and can send a message to T. This modification does not affect our proof of security: we never used the fact that messages sent between honest players will eventually be delivered.

The message sizes of Protocol 1 are linear in n. The information to be received is the same in both variants, i.e, in any case we need $O(tn^3)$ bits.

4 Optimistic Multi-Party Function Evaluation

The following construction adds perfect fairness to any general secure multi-party function evaluation (GFE) protocol that ensures *correctness* and *secrecy of inputs* even if $n-1$ of all parties are dishonest (e.g., [ChDG88]). For simplicity we will use the notation $\vec{x} := (x_1, ..., x_n)$, and analogously also for other variables.

Giving a formal definition of a GFE is beyond the scope of this paper (see [Beav591, GoLe91, MiRo92, Cane96] for several attempts to do this). Informally:

- *Correctness* means that if P_i receives an output $y \neq \mathsf{nil}$ then $y = f(\vec{x})$, where for honest P_j each x_j is the correct input by P_j. If two honest parties receive $y_i \neq \mathsf{nil}$ and $y_j \neq \mathsf{nil}$ then $y_i = y_j$.
- *Secrecy* means that the GFE does not reveal more information about the honest parties inputs x_i than what is implicit in y. The dishonest parties cannot choose their inputs depending on the inputs of the other, honest parties.

Our construction will preserve exactly the correctness and secrecy provided by the original GFE protocol, but adds *fairness*: Dishonest parties have no advantage over the honest parties in learning y.

In addition we need a *committing* encryption scheme [CDNO97] that is secure against chosen-ciphertext attacks (which implies it is *non-malleable*) [DoDN91, CrSh98]. Let $E_T(r; m)$ denote the encryption of m under T's public encryption key, using r as source of randomness for the probabilistic encryption algorithm.

Intuitively, *committing* means that there is no ciphertext c such that one can find two pairs (r, x) and (r', x') such that $E_T(r; x) = E_T(r'; x')$ but $x \neq x'$. Most encryption schemes are committing, and it is relatively simple to add this property if necessary.

Non-malleable means that an attacker who sees $E_T(r; x)$ but does not know (r, x) cannot compute a new ciphertext $E_T(r'; x')$ such that x and x' satisfy a non-trivial relation. In our case a non-trivial relation an attacker would be interested in is $\mathrm{tail}_k(x) = \mathrm{tail}_k(x')$, where $\mathrm{tail}_k()$ denotes the last k bits in the chosen binary representation of x, for some $k > 1$.

Protocol 2 (Perfectly Fair Secure Multi-Party Function Evaluations)
Let $f()$ denote the function to be computed securely. As for MPCS, each party uses an identifier tid_i to distinguish the current protocol run from all other runs. Let $tid := tid_1$.

The protocol proceeds in four phases:

1. Each P_i chooses two random values w_i and r_i.
   - w_i has the same length as $f(\vec{x})$, interpreted as a string over $\{0, 1\}$, and will be used as a one-time pad for $f(\vec{x})$. Let $w := w_1 \oplus ... \oplus w_n$.
   - r_i serves as source of randomness for the encryption of some data (see below) under T's public key. Let $r := r_1 \oplus ... \oplus r_n$.

   Let $X_i := (x_i, w_i, r_i, tid_i)$. Using the given GFE protocol we securely compute the following function:

   $$g(\vec{X}) := \begin{cases} (f(\vec{x}) \oplus w, E_T(r; w, tid)) & \text{if } tid_i = tid \text{ for all } i \\ (\mathsf{nil}, \mathsf{nil}) & \text{otherwise} \end{cases}$$

   If the GFE protocol is aborted prematurely, or if it produces the result $(\mathsf{nil}, \mathsf{nil})$ then P_i sets $decs_i := \mathsf{reject}$ and *aborts* the modified GFE protocol. Otherwise P_i sets $decs_i := \mathsf{sign}$, and let (z_i, e_i) the result that P_i obtains.

2. An optimistic MPCS on $(tid, contr_i = (e_i, tid_i))$ is started, with $decs$ as defined in Phase 1. P_i *aborts* the modified GFE protocol if the MPCS decides failed. Otherwise it continues with Phase 3.

3. P_i broadcasts (w_i, r_i) to all parties. It collects all such pairs from all other parties and computes their xor bit-by-bit, w'_i and r'_i, and checks that $e_i = E_T(r'_i; w'_i, tid_i)$.

   (a) If all this succeeds P_i outputs $y_i := z_i \oplus w'_i$.

   (b) Otherwise P_i sends the signed contract on (e_i, tid_i) to T and expects to receive w, from which it computes $y_i := z_i \oplus w$.

4. If T receives a valid, signed contract on (e_i, tid_i) then it decrypts e_i to (w, tid') and checks $tid' = tid_i$. If this succeeds it sends w to the requestor. $\diamondsuit$

The third party T must be trustworthy with respect to ensuring fairness. Privacy does not depend on T, as the secret inputs x_i are only used within the function evaluation protocol itself. T does not even learn the final result y unless it sees value z.

Theorem 4 (Security of Protocol 2)
Protocol 2 is an optimistic protocol for secure and perfectly fair multi-party function evaluation. $\diamondsuit$

Proof.
Correctness. Assume party P_i accepts value y_i in Phase 3. Thus both Phase 1 and Phase 2 must have succeeded. Correctness of the GFE protocol and the fact that P_i did not abort after Phase 1 imply that P_i obtained the *correct* result $(z_i, e_i) = (f(X) \oplus w, E_T(r; w, tid))$, with $tid_j = tid$ for all j.

- *Case 1:* P_i accepts y_i in Phase 3a. Since $E_T()$ is committing we know that w_i is fully determined by e_i. P_i received values r'_i and w'_i with $e_i = E_T(r'_i; w'_i, tid_i)$, thus we know that $w = w'_i$. Thus $z_i \oplus w'_i = z_i \oplus w = f(\vec{x})$.
- *Case 2:* P_i accepts y_i in Phase 3b. Since T is honest (by assumption) we know that w is correctly computed from e_i, and thus $z_i \oplus w = f(\vec{x})$.

Secrecy of inputs. The secrecy property of the GFE protocol implies that Phase 1 does not reveal anything more than the final result. In the best case (from the adversary's point of view) this is $(z, e) = (f(\vec{x}) \oplus w, E_T(r; w, tid))$. We don't mind if the adversary learns $f(\vec{x})$. The other values are either known a priori (tid) or sums of random values (w and r). Thus the adversary can perfectly simulate (z, e) himself. $\vec{x}$ is not used in any of the other phases.

Fairness. This follows from the fairness of the MPCS. Each w_i hides the final result y perfectly, and $e := E_t(r; w, tid)$ is assumed to hide w.

Now assume P_i is honest, and the contract in Phase 2 did not result in signed. Thus P_i will not broadcast w_i, and T will not decrypt any e_j.[9] Therefore no information about y is leaked.

If P_i is honest and decided signed in Phase 2 then it will finally receive the correct w, in Phase 3, either directly or from T. $\square$

Protocol 2 runs on asynchronous networks only if all the sub-protocols run asynchronously. All suitable GFE protocols we are aware of run on *synchronous* networks only; in this case the protocol proposed in [ABSW98] could be used for Phase 2, instead of our Protocol 1.

5 Summary

We presented the first optimistic multi-party contract signing protocol for *asynchronous* networks. We also showed how to construct an optimistic, *perfectly* fair, secure general function evaluation protocol for dishonest majority.

Acknowledgments: We thank *Birgit Pfitzmann, Matthias Schunter* and *Michael Steiner* for interesting discussions.

[9] If the set $\{P_1, ..., P_n\}$ is not clear anyway we assume that tid specifies this set unambiguously. Thus from the contract T could decide who would have to agree on the decryption, and since P_i does not agree, T would refuse decryption.

References

[ABSW98] N. Asokan, Birgit Baum-Waidner, Matthias Schunter, Michael Waidner: Optimistic Synchronous Multi-Party Contract Signing; IBM Research Report RZ 3089 (#93135), IBM Zurich Research Laboratory, Zürich, December 1998.

[AsSW296] N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Multi-Party Fair Exchange; IBM Research Report RZ 2892, IBM Zurich Research Laboratory, Zürich, November 1996.

[AsSW97] N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Fair Exchange; 4th ACM Conference on Computer and Communications Security, Zürich, April 1997, 6–17.

[AsSW98] N. Asokan, Victor Shoup, Michael Waidner: Asynchronous Protocols for Optimistic Fair Exchange; 1998 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, Los Alamitos 1998, 86–99.

[AsSW198] N. Asokan. Victor Shoup, Michael Waidner: Optimistic Fair Exchange of Digital Signatures; Eurocrypt '98, LNCS 1403, Springer-Verlag, Berlin 1998, 591–606.

[BaDM98] Feng Bao, Robert Deng, Wenbo Mao: Efficient and Practical Fair Exchange Protocols with Off-Line TTP; 1998 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, Los Alamitos 1998, 77–85.

[Beav591] Donald Beaver: Secure Multiparty Protocols and Zero Knowledge Proof Systems Tolerating a Faulty Minority; Journal of Cryptology 4/2 (1991) 75–122.

[BeGo190] D. Beaver, S. Goldwasser: Multiparty Computation with Faulty Majority; 30th Symposium on Foundation of Computer Science (FOCS) 1989, IEEE Computer Society, 1989, 468–473.

[BeGW88] Michael Ben-Or, Shafi Goldwasser, Avi Wigderson: Completeness theorems for non-cryptographic fault-tolerant distributed computation; 20th Symposium on Theory of Computing (STOC) 1988, ACM, New York 1988, 1–10.

[BGMR90] Michael Ben-Or, Oded Goldreich, Silvio Micali, Ronald L. Rivest: A Fair Protocol for Signing Contracts; IEEE Transactions on Information Theory 36/1 (1990) 40–46.

[Blum81] Manuel Blum: Three Applications of the Oblivious Transfer; Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkley, Ca. 94720, September 18, 1981.

[Cane96] Ran Canetti: Studies in Secure Multiparty Computation and Applications; Thesis, Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, June 1995, revised March 1996.

[CDNO97] Ran Canetti, Cynthia Dwork, Moni Naor, Rafail Ostrovsky: Deniable Encryption; Crypto '97, LNCS 1294, Springer-Verlag, Berlin 1997, 90–104.

[ChDG88] David Chaum, Ivan B. Damgård, Jeroen van de Graaf: Multiparty Computations Ensuring Privacy of Each Party's Input and Correctness of the Result; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 87–119.

[CrSh98] Ronald Cramer, Victor Shoup: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack; Crypto '98, LNCS 1462, Springer-Verlag, Berlin 1998, 13–25.

[DiHe76] Whitfield Diffie, Martin E. Hellman: New Directions in Cryptography; IEEE Transactions on Information Theory 22/6 (1976) 644–654.

[DoDN91] Danny Dolev, Cynthia Dwork, Moni Naor: Non-Malleable Cryptography; 23rd Symposium on Theory of Computing (STOC) 1991, ACM, New York 1991, 542–552.

[EvYa80] Shimon Even, Yacov Yacobi: Relations Among Public Key Signature Systems; Technical Report Nr. 175, Computer Science Department, Technion, Haifa, Israel, 1980.

[GaHY88] Z. Galil, S. Haber, M. Yung: Cryptographic computation: secure fault-tolerant protocols and the public-key model; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 135–155

[GMW87] Oded Goldreich, Silvio Micali, Avi Wigderson: How to play any mental game — or — a completeness theorem for protocols with honest majority; 19th Symposium on Theory of Computing (STOC) 1987, ACM, New York 1987, 218–229.

[GoLe91] Shafi Goldwasser, Leonid Levin: Fair Computation of General Functions in Presence of Immoral Majority; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 77–93.

[GoMR88] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM Journal on Computing 17/2 (1988) 281–308.

[HiMa197] Martin Hirt, Ueli Maurer: Complete Characterization of Adversaries Tolerable in Secure Multi-Party Computation; 16th Symposium on Principles of Distributed Computing (PODC), ACM, New York 1997, 25–34.

[Lync96] Nancy A. Lynch: Distributed Algorithms; Morgan Kaufmann, San Francisco 1996.

[MeOV97] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone: Handbook of Applied Cryptography; CRC Press, Boca Raton 1997.

[Mica97] Silvio Micali: Certified E-Mail with Invisible Post Offices; presented at 1997 RSA Conference.

[MiRo92] Silvio Micali, Phillip Rogaway: Secure Computation; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 392–404.

[PfSW98] Birgit Pfitzmann, Matthias Schunter, Michael Waidner: Optimal Efficiency of Optimistic Contract Signing; ACM Principles of Distributed Computing (PODC), Puerto Vallarta, June 1998, 113–122.

[Rab183] Michael O. Rabin: Transaction Protection by Beacons; Journal of Computer and System Sciences 27/ (1983) 256–267.

[RaBe89] Tal Rabin, Michael Ben-Or: Verifiable Secret Sharing and Multiparty Protocols with Honest Majority; 21st Symposium on Theory of Computing (STOC) 1989, ACM, New York 1989, 73–85.

[ScWa98] Matthias Schunter, Michael Waidner: Optimal Efficiency of Optimistic Certified Mail; IBM Zurich Research Laboratory, Zürich, November 1998.

[Yao82] Andrew C. Yao: Protocols for Secure Computations; 23rd Symposium on Foundations of Computer Science (FOCS) 1982, IEEE Computer Society, 1982, 160–164.