Round-optimal and Abuse-free Optimistic Multi-Party Contract Signing*

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

[1] Entrust Technologies (Switzerland), birgit.baum@entrust.com
[2] IBM Zurich Research Laboratory, wmi@zurich.ibm.com

Abstract. We present the first optimistic n-party contract signing protocol for asynchronous networks that tolerates up to $n-1$ dishonest signatories and terminates in the minimum number of rounds ($O(n)$). We also show how to make this protocol abuse-free by using standard cryptographic primitives (digital signatures, public-key encryption) only. Previous solutions required $O(n^2)$ rounds of communication, and non-standard cryptographic primitives for abuse freeness.

1 Introduction

A *contract signing protocol* is a protocol that allows n signatories to sign a contract text such that, even if up to $n-1$ of them are dishonest, either *all* honest signatories obtain a signed contract, or nobody obtains it [2, 8].

Dishonest signatories can arbitrarily deviate from their protocol (Byzantine model). We assume an *asynchronous* network: all messages are eventually delivered, but there are no upper bounds on network delays.[1]

Multi-party contract signing has obvious applications in secure electronic commerce, and is the basis for the solution of many related fairness problems, like multi-party certified mail [1].

By using a *third party,* T, that is always honest, the problem can be trivially solved [18]: T collects a digital signature from each of the n signatories, and either redistributes the n signatures, or aborts the contract in case not all n signatures arrive. Security depends fully on T; therefore most research has been focused on getting rid of T as trust and performance bottleneck.

Unfortunately, one cannot get rid of T completely: For $n = 2$ no deterministic protocol without third party exists [11], and each probabilistic protocol has an error probability at least linear in the number of rounds [7]. These results apply also to the n-party case, provided a majority of all signatories might be dishonest.

Therefore our goal is to minimize the involvement of T as much as possible: *Optimistic* protocols depend on a third party T, but in such a way that T is *not* actively involved in case all signatories are honest; only for recovery purposes T might become active [3, 7, 16].

* 27th International Colloquium on Automata, Languages and Programming (ICALP 2000), LNCS 1853, Springer-Verlag, Berlin 2000, 524–535.

[1] Actually we do not need to require eventual delivery for all messages; see Sect. 2.

Optimistic contract signing protocols have been first described for *synchronous* networks [2, 3, 16, 17]. 2-party protocols for *asynchronous* networks have been described in [4, 12, 17]. The n-party case was first investigated in [5]. Independently, but subsequently, protocols for the 3-party and n-party case were proposed in [12] and [13], resp. In Sect. 3 we give a precise definition of the asynchronous n-party contract signing problem.

In Sect. 4 we describe and prove our new asynchronous optimistic n-party contract signing protocol. The protocol is optimistic in case all signatories agree to sign the contract. It requires $t + 4$ rounds of communication in the worst case, $t + 2$ rounds in the optimistic case, and $O(tn^2)$ messages, where $t < n$ is a protocol parameter describing the maximum number of dishonest signatories. A variant requires $2t + 6$ rounds but only $O(tn)$ messages. For $t = n - 1$ any such protocol requires $O(n)$ rounds [13], thus, our protocol is asymptotically *round-optimal.* No previous protocol achieved this bound; the protocol proposed in [13] requires $O(n^2)$ rounds and $O(n^3)$ messages.

In Sect. 5 we add *abuse freeness* to the protocol of Sect. 4: A protocol can be *abused* if at some point in time the dishonest signatories fully control the result (i.e., depending on how they behave the result will be *signed* or *failed*, irrespective of the honest signatories' behavior) and can *prove* to an outside party that they can force the result to be *signed* [12].

Two-party abuse-free protocols have been introduced in [12] and, although without mentioning this feature, in [4]. The 3-party and n-party cases were considered in [12] and [13], respectively.

Our construction for abuse freeness is significantly simpler than the construction in [12, 13]: it uses standard cryptographic primitives only (public-key encryption, digital signatures), i.e., unlike [12, 13] we do not need specific primitives. The construction preserves the round optimality.

2 Model and Notation

Parties. Let $P_1, \ldots, P_n$ denote the *signatories,* T a *third party,* and $V_1, \ldots, V_{n'}$ the potential *verifiers* of signed contracts. Each *party* represents a machine that executes a certain protocol and that servers a specific user (e.g., human being or another protocol). Parties can receive *inputs* from their users (e.g., a signatory the command to sign a certain contract) and can generate certain *outputs* for their users (e.g., the result of the protocol).

The signatories P_i and third party T are able to *digitally sign* messages, and all parties are able to verify their signatures [14]. The signature on message m associated with P_X (T) is denoted by $\mathsf{sign}_X(m)$ ($\mathsf{sign}_T(m)$). In our complexity analyses we assume that $\mathsf{sign}(m)$ has constant length, independent of the length of m (e.g., because m is hashed before signing [15]). We abstract from the error probabilities introduced by cryptographic signature schemes and assume that signatures are unforgeable (the translation into a cryptographic model is straightforward).

Adversary model. We assume that up to t (for a given $t < n$) of the n signatories, and for some requirements also T and all verifiers might be *dishonest.* Dishonest parties can behave arbitrarily and are coordinated by a single party, called the *adversary.* Security properties must always hold for *all* adversaries.

Network. All messages sent to or from T or any V_i are reliably delivered, eventually. 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 the adversary. The adversary can read all messages from, and can insert additional messages into all channels.

All-honest case. This is a special case where we assume that *all* n signatories are honest and *all* messages sent are reliably delivered, eventually. In Sect. 5 we assume, additionally, that in this case, the adversary does not insert any messages into the channels between signatories.[2]

We make use of a few conventions, in order to simplify presentation:

Timeouts. If we say that *"party X waits for a certain set of messages, M, but can stop waiting any time"* we mean more precisely the following: X accepts a special input wakeup from its user, for each protocol execution. "Waiting for M" means that X continues to receive messages but does not proceed in the protocol run until either all messages in M have been received, or wakeup is input. (In case wakeup is input *before* X starts waiting for M it proceeds immediately.)

Initial inputs and protocol parameters. We assume that the parties and their public keys are known and fixed in advance.

3 Definitions

Definition 1 (Asynchronous Multi-party Contract Signing). An *Asynchronous Multi-Party Contract Signing Scheme* (asynchronous MPCS) consists of two protocols:

- $\mathsf{sign}[P_1, \ldots, P_n]$, for signing a contract with signatories $P_1, \ldots, P_n$. $\mathsf{sign}[]$ might involve an additional party, T, in which case we call it an *MPCS with third party.*
- $\mathsf{verify}[P_i, V_j]$, for showing a signed contract to any verifier, V_j, $j \in \{1, \ldots, n'\}$. $\mathsf{verify}[]$ never involves T, i.e., it is always a 2-party protocol only.

A signatory P_i starts $\mathsf{sign}[]$ on input $(\mathsf{sign}, tid_i, contr_i)$.[3] tid_i is a transaction identifier unique for all executions of $\mathsf{sign}[]$. $contr_i$ is the contract text to be signed.

[2] This assumption is not needed for a weaker version of abuse freeness; see [6].

[3] If the user of P_i does not wish to sign then P_i does not participate in $\mathsf{sign}[]$. One could also allow an additional input $decision_i \in \{\mathsf{sign}, \mathsf{reject}\}$, indicating whether the user wants to sign or not, and assume that in the all-honest case each signatory receives an input and participates in $\mathsf{sign}[]$. This would enable faster termination in case one signatory gets input reject.

Upon termination $\mathsf{sign}[]$ produces an output $(tid_i, contr_i, d_i)$ for P_i, with $d_i \in \{\mathsf{signed}, \mathsf{failed}\}$. We will simply say "$P_i$ decides d_i." P_i may receive an input (wakeup, tid_i) any time.

A signatory P_i starts $\mathsf{verify}[]$ with verifier V_j on input $(\mathsf{show}, V_j, tid_i, contr_i)$. A verifier V_j starts $\mathsf{verify}[]$ on input $(\mathsf{verify}, P_i, tid_V, contr_V)$. Upon termination $\mathsf{verify}[P_i, V_j]$ produces an output $(tid_V, contr_V, d_V)$ with $d_V \in \{\mathsf{signed}, \mathsf{verify\_failed}\}$ for V_j. We will simply say "V_j decides d_V." No output is produced for P_i. V_j may receive an input (wakeup, tid_V) any time.

The following requirements must be satisfied:

- **(R1)** *Correct execution.* In the all-honest case, if all signatories start with the same input (sign, tid, $contr$), and no signatory receives (wakeup, tid), then all signatories terminate and decide signed.
- **(R2)** *Unforgeability.* If an honest P_i never received input $(\mathsf{sign}, tid, contr)$ then *no* honest V_j that receives input $(\mathsf{verify}, P_{i'}, tid, contr)$, for any $P_{i'}$, will decide signed. (Note that this does not assume an honest T.)
- **(R3)** *Verifiability of valid contracts.* If an honest P_i decides signed on input (sign, tid, $contr$), and later P_i receives input $(\mathsf{show}, V_j, tid, contr)$ and honest V_j receives input $(\mathsf{verify}, P_i, tid, contr)$ and does not receive (wakeup, tid) afterwards then V_j will decide signed.
- **(R4)** *No surprises with invalid contracts.* If T is honest, and an honest P_i received input $(\mathsf{sign}, tid, contr)$ but decided failed then no honest verifier V_j receiving $(\mathsf{verify}, P_{i'}, tid, contr)$, for any $P_{i'}$, will decide signed.
- **(R5)** *Termination of* $\mathsf{sign}[]$. If T is honest then each honest P_i that receives sign and wakeup (for the same tid and $contr$) will terminate eventually.
- **(R6)** *Termination of* $\mathsf{verify}[]$. Each honest V_j that receives verify and then wakeup, for the same tid, and each honest P_i that receives show will terminate eventually.

Definition 2 (Optimistic Protocol). An MPCS with third party T is called *optimistic on agreement* if in the all-honest case, if all signatories receive input (sign, tid, $contr$) and none receives (wakeup, tid) the protocol terminates without T ever sending or receiving any messages.

It is called *optimistic on disagreement* if in the all-honest case, if some signatories do *not* receive input (sign, tid, $contr$) the protocol terminates without T ever sending or receiving any messages.

It is called *optimistic* if it is optimistic on agreement and on disagreement.

Definition 3 (Abuse freeness [12]). An MPCS is called *abuse-free* if it is impossible for the adversary at any point in the protocol to be able to prove to an outside party that he has the power to control the contract, i.e., to terminate the protocol with result signed or failed.

Remark 1. Def. 1 captures *plain* contract signing only. We do not require additional features, like fair exchange of pre-defined signatures [4] or invisibility of T [16]. If required, they can be implemented on top of contract signing.

4 Asynchronous Optimistic Multi-party Contract Signing

The following Scheme 1 solves the multi-party contract signing problem, and is optimistic on agreement.

Ignoring all details, the protocol works as follows: It consists of $t + 2$ locally defined rounds. In Round 1 each signatory that starts the protocol signs a "promise" to sign the contract and broadcasts this promise. In each subsequent round each signatory collects all signatures from the previous round, countersigns this set of n signatures, and broadcasts it.[4] The result of the $(t + 2)$-nd round becomes the real contract.

A signatory who becomes tired of waiting for some signatures in some round sends the information received so far to the honest T, stops sending further messages and simply waits for T's answer. T analyses the situation and decides either failed or signed:

If the first request received by T comes from a signatory in the *first* round then T must decide failed. If T receives a request from the *last* round then T must decide signed, as other signatories might already have the signed contract. Thus if T receives requests from all rounds then somewhere in the middle between the first and the last round T might have to change the decision from failed to signed. But T must do this only if it is clear that all requests that were answered with failed came from dishonest signatories—otherwise two honest signatories might decide differently.

Scheme 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 indicating whether P_i contacted T or not. Both are initialized before executing any rule. Let $M_{0,i} := \mathsf{nil}$, for all i.

- **Rule S1:** If $raised\_exception = \mathsf{false}$ and $r = 1$ then:
  - P_i sends $m_{1,i} := \mathsf{sign}_i(c_i, 1, \mathsf{prev\_rnd\_ok})$ to all signatories.
  - From all received messages of type $m_{1,j}$ it tries to compile full and consistent vectors $M_{1,i} := (m_{1,1}, \ldots, m_{1,n})$ and $X_{1,i} := M_{1,i}$ with $m_{1,j} = \mathsf{sign}_j(c_i, 1, \mathsf{prev\_rnd\_ok})$. (NB: here we also check that all signatories agree on c_i.) If this succeeds P_i sets $r := 2$.

  At any time P_i can stop waiting for any missing $m_{1,j}$ (i.e., the user of P_i might enter wakeup any time), in which case it sets $raised\_exception := \mathsf{true}$ and sends $resolve_{1,i} := (1, i, \mathsf{sign}_i(m_{1,i}, \mathsf{resolve}))$ to T.
- **Rule S2:** If $raised\_exception = \mathsf{false}$ and $2 \leq r \leq t + 2$ then:
  - P_i sends $m_{r,i} := (\mathsf{sign}_i(M_{r-1,i}, r, \mathsf{vec\_ok}), \mathsf{sign}_i(c_i, r, \mathsf{prev\_rnd\_ok}))$ to all signatories.

[4] The real protocol does this more efficiently, in order to keep the messages short.

- From all received messages of type $m_{r,j}$ it tries to compile full and consistent vectors

  $M_{r,i} = (\mathsf{sign}_1(c_i, r, \mathsf{prev\_rnd\_ok}), \ldots, \mathsf{sign}_n(c_i, r, \mathsf{prev\_rnd\_ok}))$
  $X_{r,i} := (\mathsf{sign}_1(M_{r-1,i}, r, \mathsf{vec\_ok}), \ldots, \mathsf{sign}_n(M_{r-1,i}, r, \mathsf{vec\_ok}))$

  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 stop waiting for any missing $m_{r,j}$. In this case P_i sets $\textit{raised\_exception} := \mathsf{true}$ and sends $\textit{resolve}_{r,i} := (r, i, \mathsf{sign}_i(X_{r-1,i}, \mathsf{resolve}), X_{r-1,i}, M_{r-2,i})$ to T.
- **Rule S3:** If $\textit{raised\_exception} = \mathsf{true}$ then:
  P_i waits for a message from T (without sending or receiving any further messages). This can be any of $\textit{signed}_{r,i} = (\textit{resolve}_{r',j}, \mathsf{sign}_T(c, r', j, \mathsf{signed}))$ or $\textit{aborted}_{r,i} = \mathsf{sign}_T(c_i, r, i, \mathsf{aborted})$. On receiving one P_i decides as indicated by T and *stops*; if it decides signed it sets $C_i := \textit{signed}_{r,i}$.

Protocol "sign" for third party T:

We assume T receives a message $\textit{resolve}_{r,i}$, as defined in the protocol for P_i. Let c be the (unique) pair (*tid*, *contr*) contained in $\textit{resolve}_{r,i}$.

If this is the first time T is asked about this contract (i.e., *tid*) then T initializes a Boolean variable $\textit{signed} := \mathsf{false}$ and two sets $\textit{con} := \emptyset$ and $\textit{abort\_set} := \emptyset$. *signed* indicates T's current decision, signed or failed. *con* is the set of all indices of signatories that contacted T, and *abort_set* is the set of all failed-messages sent by T.

Processing $\textit{resolve}_{r,i}$ cannot be interrupted, i.e., it cannot happen that T processes two different $\textit{resolve}_{r,i}$ concurrently.

- **Rule T0:** *(T accepts only one request from each P_i. All other requests are ignored.)* If $i \in \textit{con}$ then the message $\textit{resolve}_{r,i}$ is ignored.
- **Rule T1:** *(If T receives a request from Round $r = 1$ and has not yet decided* signed *then T sends an* abort *to the requester.)* If $i \notin \textit{con}$ and $\textit{signed} = \mathsf{false}$ and $r = 1$ then T sets

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

  and sends $\textit{aborted}_{r,i}$ to P_i.
- **Rule T2:** *(If T receives a request from a Round $r > 1$ and has not yet decided* signed *then T checks whether all previous requests came from dishonest signatories, using Lemma 1. If this is the case then it changes the decision to* signed*, otherwise T sticks to* $\mathsf{aborted}$*.)*
  If $i \notin \textit{con}$, $\textit{signed} = \mathsf{false}$, $r > 1$, and
  if for all $\textit{aborted}_{s,k} \in \textit{abort\_set}$ we have $s < r - 1$
  - then: If $\textit{con} = \emptyset$ then T sets $\textit{first\_signed} := (\textit{resolve}_{r,i}, \mathsf{sign}_T(c, r, i, \mathsf{signed}))$. T sets
    $\textit{signed}_{r,i} := \textit{first\_signed}$

$signed := \mathsf{true}$
$con := con \cup \{i\}$

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

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

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

– **Rule T3:** *(A decision* signed *is always preserved.)* 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 outputs signed if it receives a messages C_i from P_i such that

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

and *stops*. On input wakeup V outputs verify_failed and *stops*.

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

Proof. Assume T finds $aborted_{s,k} \in abort\_set$ with $s \leq r-2$. Since $s \geq 1$ we have $r \geq 3$, and therefore $resolve_{r,i}$ includes $\mathsf{sign}_k(M_{r-2,k}, r, \mathsf{vec\_ok})$, 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. □

Theorem 1 (Security of Scheme 1). *Scheme 1 is an asynchronous MPCS with third party T for any $t < n$. 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.*

Proof. (Sketch)

Correct execution, termination of verify and *optimistic on agreement* are all obviously satisfied.

Unforgeability of contract. All variants of a valid contract contain some pieces signed by all signatories, and these signatures exist only if all signatories started the protocol.

Verifiability. The definitions of C_i in the signing protocol for P_i satisfy the conditions checked by V.

No surprises with invalid contracts. Assume an honest P_i started the protocol, decided failed, and some honest verifier V decides signed.

- Assume V decided because of Rule V1. This implies that P_i sent $\mathsf{sign}_i(c, t+2, \mathsf{prev\_rnd\_ok})$ 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 signatory $P_{k_{t+1}}$ that received $aborted_{t+1,k_{t+1}}$. Inductively we can show (using T2) that for all rounds $s \in \{1, \ldots, t+1\}$ there is one signatory P_{k_s} that received $aborted_{s,k_s}$. Because of Lemma 1 we know that all signatories P_{k_s} with $s \in \{1, \ldots, t\}$ are dishonest, and as there are at most t dishonest signatories 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 signatory receives all information that is necessary to construct a signed contract. This 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 1). But this contradicts Rule T2 of T, i.e., if P_i received $aborted_{s,i}$ then $\mathsf{sign}_T(c, r, i, \mathsf{signed})$ cannot exist. Again we have a contradiction.

Termination of sign[]. For each signatory 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). □

Remark 2. The protocol is optimistic on agreement only: If P_i starts the protocol but P_k does not then P_i will send $resolve_{1,i}$ to T. In [6] we show that any such protocol can be transformed into a protocol that is optimistic in all cases.

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 "sign" of Scheme 1 runs in $t+2$ rounds where each signatory broadcasts one message to all other signatories, resulting in costs of $(t+2)nC_b$. In the worst case each signatory might have one message exchange with T, resulting in $t+4$ rounds and costs of $(t+2)nC_b + 2nC_s$. If one assumes that each broadcast requires $n-1$ single messages we end up with costs of $((t+2)n(n-1) + 2n)C_s = O(tn^2)C_s$. All broadcast messages of Scheme 1 have length $O(\log n)$, all messages sent to or by T have length $O(n)$; thus we need $O(tn^2 \log n)$ bits only.

Alternatively we can replace each round of n simultaneous broadcasts by 2 rounds and $2(n-1)$ single messages. This will result in $2t+6$ rounds and costs of $((t+2)2(n-1)+2n)C_s = O(tn)C_s$, and still $O(tn^2 \log n)$ bits: Each broadcast round, say, Round r, is implemented by two mini-rounds: in Mini-Round r_1 each signatory 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 signatories 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, since only in the all-honest case messages sent between honest signatories have to be delivered eventually.

According to [13] any asynchronous optimistic multi-party contract signing protocol tolerating $t = n - 1$ requires a number of rounds at least linear in n.

Corollary 1 (Round optimality).
The number of rounds in Protocol "sign" of Scheme 1 is asymptotically optimal for $t = n - 1$.

5 Abuse-free Asynchronous Multi-party Contract Signing

Scheme 1 is not abuse-free: Consider $n = 2$ and $t = 1$, and assume that P_2 is dishonest. Messages $m_{1,1}$, $m_{1,2}$ and $m_{2,1}$ are delivered. Now the adversary fully controls the result: If he ignores $m_{1,1}$ and $m_{2,1}$ and delivers $resolve_{1,2}$ to honest T then T will decide failed. If he delivers $resolve_{3,2}$ to T then T will decide signed. The message $resolve_{3,2}$ convinces any outside party that T would actually decide signed.

One cannot avoid that the adversary has an advantage in determining whether the protocol ends with signed or with failed, provided all signatories start the protocol. But one *can* avoid that the adversary can prove his ability to force the result signed to an outside party.

In [12, 13] this is achieved by introducing and constructing a new type of digital signature scheme, called private contract signatures.

Our solution is conceptually much simpler, as it uses standard cryptographic primitives only: The basic idea is to use Scheme 1 as it is, but let each signatory P_i generate a *new, fresh* pair of secret and public signature key, (sk_i, pk_i), for each execution of Protocol "sign" of Scheme 1. We call the result of such an execution a *pre-contract.* As long as the adversary cannot prove that a certain fresh key belongs to a certain signatory he cannot prove to an outside party the status of the protocol, and hence the protocol is abuse-free.

We call the fresh keys (sk_i, pk_i) *short-term keys,* in contrast to the *long-term keys* used by P_i to generate signatures under its name. Note that by convention (Sect. 2) all the public long-term keys are fixed in advance! Let $\mathsf{SIGN}_i(x)$ denote P_i's signature with its long-term key, and let $\mathsf{sign}_i(x)$ denote P_i's signature with its short-term key sk_i.

Our construction makes use of a chosen-ciphertext secure public-key encryption scheme [9, 10]. Only T needs to decrypt messages; let $E_T(r; x)$ denote the encryption of x with T's public key using random string r. We assume that each party knows T's public encryption key.

Scheme 2 (Asynchronous Optimistic Abuse-free MPCS)

Protocol "sign" for honest P_i:

– **Phase 1, Distribution of encrypted certificates:** On input $(\mathsf{sign}, tid, contr)$ signatory P_i generates (sk_i, pk_i) and computes $cert_i := \mathsf{SIGN}_i(i, pk_i, tid, contr)$ and $ecert_i := E_T(r_i; cert_i)$ for a randomly chosen r_i. P_i distributes $(pk_i,\ ecert_i,\ tid,\ contr)$ to all signatories.
  Each P_i collects the list of all n tuples. It can stop waiting for a tuple any time, in which case it outputs failed and stops.
– **Phase 2, Pre-contract signing:** Protocol "sign" of Scheme 1 is executed to obtain the pre-contract, using the n fresh keys $(pk_i,\ sk_i)$. The pre-contract to be signed is the concatenation of tid, $contr$, and $((pk_1, ecert_1), \ldots, (pk_n, ecert_n))$.[5]
– **Phase 3, Conversion of pre-contract into real contract:** If P_i gets output signed for the pre-contract then P_i broadcasts $(r_i, cert_i)$ to all other signatories. (This allows any other party to check that $ecert_i$ was an encryption of $cert_i$.)
  If P_i receives input wakeup before having received all n pairs, or if not all pairs match (i.e., there is some j such that $ecert_j \neq E_T(r_j; cert_j)$, or $cert_j$ is not a valid signature by P_j on $(j, pk_j, tid, contr)$) then P_i shows the signed pre-contract to T.
– **Phase 4, Recovery from failed conversion:**
  - If T gets a signed pre-contract it extracts the list $((pk_1, ecert_1), \ldots, (pk_n, ecert_n))$ and tries to decrypt all certificates.
    If this fails, or if some of the decrypted certificates are not valid or not of the proper format, then T signs a "failed" message and sends it back. Otherwise T signs $(tid, contr, cert_1, \ldots, cert_n)$ and sends all certificates and T's signature back. (T's signature certifies that $ecert_j$ was an encryption of $cert_j$ for each j. Note that usually T does not know r_j.)
  - If P_i receives an answer from T it verifies T's signature. If T decided failed then P_i decides failed and stops. Otherwise P_i checks the consistency of the data received from T, and if this succeeds decides signed and stops. If not it decides failed and stops.

Protocol "verify":

If P_i wants to show a signed contract to verifier V it sends to V the pre-contract from Phase 2, signed relative to short-term keys $pk_1, \ldots, pk_n$ and (a) all n pairs $(r_i, cert_i)$ or (b) the vector $(tid, contr, cert_1, \ldots, cert_n)$, signed by T. V outputs signed if all information matches, and *stops*.

On input wakeup V outputs verify_failed and *stops*.

Theorem 2 (Security of Scheme 2).

Scheme 2 is an asynchronous MPCS with third party T for any $t < n$. It is optimistic in all cases and terminates in $t + 4$ rounds if T is not involved, and in $t + 6$ rounds in the worst case.

[5] Note that this pre-contract is *not* the real contract yet, and does not prove anything to an outside party: we will show that the adversary can generate valid looking pre-contracts himself, without any interaction with the honest signatories.

Proof. (Sketch) We prove only three properties:

"Correct execution:" In the all-honest case the channels between signatories ensure eventual delivery and *authenticity* (Sect. 2). Thus the adversary cannot disturb the protocol, and it terminates with decision signed.

"Unforgeability:" Assume some honest P_i did not want to sign, but a verifer V outputs signed. Th. 1 implies that pre-contracts are unforgeable relative to the short-term keys, and Protocol "verify" requires that one of the short-term keys was certified by P_i, which contradicts our assumption. (Note that the set of signatories is fixed in advance; see Sect. 2.)

"No surprises with invalid contracts:" Assume an honest P_i started the protocol, decided failed, and some honest verifier V decides signed.

Th. 1 implies that there was a run of Protocol "sign" of Scheme 1 that yielded a signed pre-contract. P_i participated in this run as one of the short-term keys used was certified by P_i relative to the original $(tid, contr)$, and P_i certifies only fresh keys. Therefore P_i obtained a signed pre-contract as well. As P_i finally decided failed it must have obtained "failed" from T in Phase 4, which means that T was not able to successfully decrypt and check all certificates.

Since T does not distribute inconsistent decisions V accepted the contract because of the result of a successful Phase 3. Thus there must be n pairs $(r_j, cert_j)$ that are valid certificates and consistent with the signed contract. But T was not able to obtain the corresponding n certificates $cert_i$. Since T is honest this contradicts the correctness requirement of the encryption scheme. □

Remark 3. Phase 1 does not guarantee that $ecert_i$ actually contains a valid certificate $cert_i$, i.e., it might happen that there are invalid certificates but Phase 2 is started and terminates successfully.

At first glance this might look dangerous, but actually it is not: If at least one $ecert_i$ is invalid—i.e., decryption does not yield a certificate $cert_i$ on pk_i signed by P_i— then no party will obtain the real contract.

Theorem 3 (Abuse freeness). *Scheme 2 is abuse-free.*

Proof. (Sketch) We prove this property by contradiction: Assume that at a certain point in time the adversary has full control over the contract and can convince an outside party that he can enforce result signed.

Since the adversary has full control still both results are possible. Therefore no party can show a valid pre-contract, as the information contained in it would determine the result of the contract signing. This follows from Th. 2: If one signatory had a signed pre-contract then any other signatory would obtain it as well, and T could unambiguously decide by evaluating this pre-contract. Therefore no honest signatory will enter Phase 3, i.e., no honest signatory will open any encryption.

In this situation we can simulate the adversary's view of the protocol: The simulator generates fresh keys (sk_i, pk_i), exactly like the honest signatories did. But instead of encrypting certificates $cert_i$ on these keys, the simulator encrypts random data of the same length. Since the encryption scheme is secure against adaptive chosen ciphertext attacks the resulting ciphertexts $ecert_i$ are indistinguishable from encrypted certificates. This simulation works as long as none of the ciphertexts must be opened—which is the case as no valid pre-contract exists, yet. Thus, any interaction between the adversary and an outside party can be simulated without any interaction with the honest signatories. Hence, the outside party gets no information about whether the honest signatories actually want to sign a contract, which contradicts our assumption. □

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

References

1. N. Asokan, B. Baum-Waidner, M. Schunter, M. Waidner: Optimistic Synchronous Multi-Party Contract Signing; IBM Research Report RZ 3089 (#93135), Zürich, December 1998.
2. N. Asokan, M. Schunter, M. Waidner: Optimistic Protocols for Multi-Party Fair Exchange; IBM Research Report RZ 2892, Zürich, November 1996.
3. N. Asokan, M. Schunter, M. Waidner: Optimistic Protocols for Fair Exchange; 4th ACM Conf. on Computer and Communications Security, Zürich, April 1997, 6–17.
4. N. Asokan. V. Shoup, M. Waidner: Optimistic Fair Exchange of Digital Signatures; Eurocrypt '98, LNCS 1403, Springer-Verlag, Berlin 1998, 591–606.
5. B. Baum-Waidner, M. Waidner: Asynchronous Optimistic Multi-Party Contract Signing; IBM Research Report RZ3078 (#93124), Zürich, November 1998.
6. B. Baum-Waidner, M. Waidner: Round-optimal and Abuse-free Optimistic Multi-Party Contract Signing; IBM Research Report RZ3199 (#93245), Zürich, Jan 00.
7. M. Ben-Or, O. Goldreich, S. Micali, R. L. Rivest: A Fair Protocol for Signing Contracts; IEEE Transactions on Information Theory 36/1 (1990) 40–46.
8. M. Blum: Three Applications of the Oblivious Transfer; Department of Electrical Engineering and Computer Sciences, University of California at Berkley, September 18, 1981.
9. R. Cramer, V. Shoup: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack; Crypto '98, LNCS 1462, Springer-Verlag, Berlin 1998, 13–25.
10. D. Dolev, C. Dwork, M. Naor: Non-Malleable Cryptography; 23rd Symposium on Theory of Computing (STOC) 1991, ACM, New York 1991, 542–552.
11. S. Even, Y. Yacobi: Relations Among Public Key Signature Systems; Technical Report Nr. 175, Computer Science Department, Technion, Haifa, Israel, 1980.
12. J. Garay, M. Jakobsson, P. MacKenzie: Abuse-free Optimistic Contract Signing; Crypto '99, LNCS 1666, Springer-Verlag, Berlin 1999, 449–466.
13. J. Garay, P. MacKenzie: Abuse-free Multi-party Contract Signing; Intern. Symp. on Distr. Comput. (DISC '99), LNCS 1693, Springer-Verlag, Berlin 1999, 151–165.
14. S. Goldwasser, S. Micali, R. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM J. Comput. 17/2 (1988) 281–308.
15. A. Menezes, P. van Oorschot, S. Vanstone: Handbook of Applied Cryptography; CRC Press, Boca Raton 1997.
16. S. Micali: Certified E-Mail with Invisible Post Offices; 1997 RSA Conference.
17. B. Pfitzmann, M. Schunter, M. Waidner: Optimal Efficiency of Optimistic Contract Signing; ACM PODC '98, Puerto Vallarta 1998, 113–122.

18. M. Rabin: Transaction Protection by Beacons; Journal of Computer and System Sciences 27/ (1983) 256–267.