Abuse-free Optimistic Contract Signing

Juan A. Garay* Markus Jakobsson* Philip MacKenzie*

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

Abstract. We introduce the notion of *abuse-free* distributed contract signing, that is, distributed contract signing in which no party ever can prove to a third party that he is capable of choosing whether to validate or invalidate the contract. Assume Alice and Bob are signing a contract. If the contract protocol they use is *not* abuse-free, then it is possible for one party, say Alice, at some point to convince a third party, Val, that Bob is committed to the contract, whereas she is not yet. Contract protocols with this property are therefore not favorable to Bob, as there is a risk that Alice does not really want to sign the contract with him, but only use his willingness to sign to get leverage for another contract. Most existing optimistic contract signing schemes are not abuse-free. (The only optimistic contract signing scheme to date that does not have this property is inefficient, and is only abuse-free against an off-line attacker.) We give an efficient abuse-free optimistic contract-signing protocol based on ideas introduced for designated verifier proofs (i.e., proofs for which only a designated verifier can be convinced). Our basic solution is for two parties. We show that straightforward extensions to $n > 2$ party contracts do not work, and then show how to construct a three-party abuse-free optimistic contract-signing protocol.
An important technique we introduce is a type of signature we call a *private contract signature*. Roughly, these are designated verifier signatures that can be converted into universally-verifiable signatures by either the signing party or a trusted third party appointed by the signing party, whose identity and power to convert can be verified (without interaction) by the party who is the designated verifier.

1 Introduction

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 contracts 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, there is room for improvement. In this paper, we discuss some shortcomings of the present state of the art in digital contract signing, and propose a solution that

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

we prove has all the desired properties that we typically associate with fairness. Finally, we show how to extend our scheme to three participants without losing these properties.

Recently, considerable efforts have been devoted to develop 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.

Early work on electronic contract signing, or more generally, fair exchange of secrets/signatures, focused on the gradual release of secrets to obtain simultaneity, and thus fairness [9, 25, 30] (see [21] 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 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 has stopped—and perhaps even has engaged in another contract signing protocol!

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 [24], 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 [16, 22, 28], 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, 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.

Most of these protocols have the conceptual drawback of allowing one of the parties at some time to convince a third party he has a *potentially signed contract*. That is, this participant can demonstrate that the other party is committed (to some extent) to the agreement, without himself being committed. Ultimately, of course, we want *neither* of the participants to have this ability until the contract has been signed by both parties.

In this sense, it is possible to distinguish between two types of attacks, which we call *on-line* vs. *off-line*. In an on-line attack, the cheating contract signer and a third party may interact in an arbitrary fashion during the contract signing protocol, in order for the third party to be convinced of the other signer's willingness to sign. In an off-line attack, however, this is not allowed, and the cheating contract signer may only send the third party one message. Clearly, the former is a stronger form of attack. All previously proposed contract signing protocols are vulnerable against the on-line attack, and all proposed optimistic contract signing protocols – but one – are vulnerable against the off-line attack. Our proposed scheme is secure against both types of attacks. A third – and even stronger – attack allows the cheating signer to cooperate with the third party from the "beginning of time". The attack involves the sharing of the cheating signer's secret key by the cheating signer and the third party (i.e., the key is stored in a way such that neither of them can reconstruct it on their own.) Such an attack must be initiated before the certification of the corresponding public key. As in [31], we have to assume that the certification process is done in an "isolated" manner, and that the certified party be required to prove knowledge of his secret key before he becomes certified, thereby preventing successful "identity sharing" of this type. We believe that this is the only defense against such an "ultimate" attack, apart from mere social constraints that make very careful planning and administering of attacks "from the beginning of time" less likely to take place.

Our results. Our contributions are twofold:

- In this paper we introduce the notion of "abuse freeness" to the area of electronic contract signing, and present an efficient abuse-free optimistic contract signing protocol. We also show how to extend the scheme to three parties without losing these properties.
- To build the protocols we use a new type of signature that we call *private contract signatures* (PCS). Roughly, these are designated verifier signatures that can be converted into universally-verifiable signatures by either the signing party or a trusted third party appointed by the signing party, whose identity and power to convert can be verified (without interaction) by the party who is the designated verifier. We give a complete formalization of these signatures, and present an efficient discrete log-based scheme which uses new boolean combinations of proofs of membership and knowledge. We prove the scheme to be as secure as the Decisional Diffie-Hellman problem.

Related work. Contract signing is part of the broader problem of fair exchange [10, 16, 28, 33, 2]. More specifically, it can be considered fair exchange of digital signatures [3]. The different existing approaches to contract signing were already described above. The term "contract signing" was first introduced in [8]. The first optimistic scheme in the sense defined above was based on the gradual increase of privilege [7]: as the computation evolves, the probability of a contract being valid gradually increases from 0 to 1. This solution has several shortcomings, as it requires termination detection by the third party (i.e., synchronous system), and one of the parties might be "privileged," in the sense of only one of them being able to finalize the contract; this happens with a non-negligible probability.

The most closely related work to ours is Asokan, Shoup, and Waidner [3]. They present an optimistic protocol for fair exchange of digital signatures. They allow any of a variety of signature schemes, and prove security of their protocol. It is not mentioned, but one of their protocols is off-line abuse-free. However, their protocol is inefficient, due to the use of expensive cut-and-choose techniques. More specifically, to obtain a failure probability of 2^{-k}, they require $2k + 4$ exponentiations for both prover and verifier. In contrast, we require 32 exponentiatons for the prover and the verifier, regardless of the security parameter.[1] Also, their scheme does not allow the trustee to be held accountable for his actions, meaning that if the trustee were to misbehave, it would not be possible for the participants to show to a judge that this must have happened. They present another fair optimistic contract signing protocol that is more efficient, but does not have the abuse-free property, nor the ability to allow any type of signature scheme.

The abuse-free contract signing protocol we present is more efficient (in number of rounds, communication bits, and computation), than the only published off-line abuse-free protocol so far, namely the one in [3], and provides for trustee accountability. Our scheme is both off-line and on-line abuse-free. However, our protocol restricts the participants to a certain type of signature.

The *private contract signatures* that we introduce exhibit some similarities with *designated-confirmer signatures* and *convertible undeniable signatures*, two notions first suggested by Chaum [12]. A designated-confirmer signature is an undeniable signature [13, 15] that can be proven valid by a so-called confirmer, an entity appointed by the signer and given a secret key for verification by the same. A convertible undeniable signature is an undeniable signature that can be converted to a standard (i.e., self-authenticating) signature, This can be done either by the signer or, in the setting of designated confirmer signatures, by a trusted third party. Although Chaum's scheme was broken by Michels [**?**], its conceptual ideas are still intact, and alternative solutions were suggested in [19, **?**].

One major difference between the concept of convertible undeniable signatures and private contract signatures is that the latter allow an optimistic approach in that the receiver knows beforehand that the trusted third party can convert the signature. In contrast, a convertible undeniable signature needs a verification session, such as an interactive undeniable signature verification [13, 15]. Technically, they differ in that the signer of a convertble undeniable signature needs to give a secret key to the trusted third party, whereas a private contract signature is performed relative to the public key of the trusted third party, who does not need to obtain any secret key from the signer. One could say that a private contract signature is a special case of a "designated-confirmer convertible signature," namely, one in which the receiver knows that the contents correspond to a valid publicly-verifiable signature, but can't convince anybody else about this without the help of either the signer or the trusted third party.

[1] Batch computation methods (e.g., [32, 5]) can be applied to both the schemes, and although they can be applied to a larger extent to [3], their scheme remains noticeably more expensive than ours for reasonable security parameters.

Our work is also closely related to work on *designated-verifier proofs*, which were introduced by Jakobsson *et al.* [31]. A designated-verifier proof of a statement Θ is a proof with the property that it will convince nobody but the so-called "designated verifier" (who is selected by the prover) of the correctness of Θ. Furthermore, the designated verifier will accept a valid such proof and reject an invalid such proof with an overwhelming probability[2]. We call a *non-interactive* designated-verifier proof a designated verifier *signature*.

A private contract signature is a designated-verifier signature with the property that it may be converted to a standard, self-authenticating signature, drawing upon the ideas of designated *converter* proofs and convertible undeniable signatures. This conversion can be performed by two participants only: the participant who generated the signature, and a trusted third party, who is appointed by the participant who generated the signature, and whose identity can be determined from the transcript constituting the private contract signature. However, as already mentioned, the initial receiver of the signature need not interact with anybody in order to determine whether the transcript he has received can always be converted into a standard signature.

Finally, we show that natural extensions of an abuse-free protocol to three-party contract signing do not seem to allow abuse-freeness. However, we present a three-party fair abuse-free optimistic contract-signing protocol obtained by careful design of a (much more intricate) protocol.[3] Independently of our work, *multi-party* contract signing protocols were given in [1, 4] While not abuse-free, the protocol in [4] is simpler than ours, resulting in a more elegant and efficient general solution.

2 Model and Definitions

Our basic model is similar to that of [3]. We have 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. We will first concentrate on the case of two participants, A and B.

The network model we consider is the same as in [3]. Namely, an asynchronous communication model with no global clocks, where messages can be delayed arbitrarily [27], but with messages sent between correct participants and the trusted third party guaranteed to be delivered eventually. In general, we assume

[2] Depending on the proof structure, this probability might be a constant fraction, in which case the standard method of repeating the proof can be applied.

[3] In fact, this has been simplified and extended to a general n-party contract signing [29].

an adversary may schedule messages, and possibly insert its own messages into the network.

A and B wish to "sign" a contract m.[4] 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 A and B on m, or valid signatures by A and B on the concatenation of m and the public key of T, signed together by T.

Obviously, in the asynchronous model of [27], 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 corrupted participant (say A^*) to obtain a valid contract without allowing the remaining participant (say 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.

Finally, an optimistic contract signing protocol is *abuse-free* if it is impossible for a single player (say A^*) at any point in the protocol to be able to prove to an outside party that he has 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 and Techniques

For a prime q, we will work 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.

[4] In general, A and B might need to sign two different pieces of text, m and m'. Extension to this case is trivial.

We assume the hardness of the *Diffie-Hellman decision problem* (DDH). In this problem, it is the goal of the p-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 $r \in_R Z_q$.

ElGamal encryption [23] is a public-key encryption scheme, in which it is assumed that the recipient Bob is associated with a public key y and Bob has a private key x such that $g^x = y$. To generate an ElGamal encryption of a value $m \in G_q$ for Bob, a sender Alice generates $r \in_R Z_q$, and sends $(y^r m, g^r)$. Bob decrypts an incoming message (a, b) by computing a/b^x.

The security of ElGamal encryption has been shown to equal the security of the Diffie-Hellman decision problem.

3.1 Non-interactive proofs of knowledge

Our protocols will require certain non-interactive proofs. In this paper, these proofs will basically be the non-interactive versions of Σ-protocols [20, 26] (i.e., Schnorr-like protocols [36]). However, we use a general notion of Σ-protocols [17] which encompass both proofs of knowledge and membership. We refer the reader to [17] for the full technical definitions.

We also use proofs of conjunctions and disjunctions (i.e., ANDs and ORs) of certain statements which have known non-interactive proofs [18]. To simplify notation, we will often state these as conjunctions and disjunctions of the specific *proofs* (instead of the statements), which will indicate proofs of conjunctions and disjunctions of the statements built on those specific proofs. (More formally, a boolean function $b()$ over some non-interactive proofs will denote the non-interactive proof corresponding to the Σ-protocol that is obtained from the boolean function $b()$ applied to the Σ-protocols corresponding to the individual proofs.)

Finally, we use designated verifier proofs/signatures ([31], see also [12]), which are built using conjunctions and disjunctions.

Proof of ciphertext contents Let (a, b) be an ElGamal ciphertext. We want a protocol for proving ciphertext contents, i.e., a protocol for proving that a ciphertext (a, b) corresponds to a plaintext m. We want to consider such proofs that can be disjunctively composed (which excludes some of the simplest approaches), and for this we will use a Σ-protocol.

Specifically, we want to prove that $(a, b) = (y^r m, g^r)$ for some value r and a specified public key y and plaintext m. This can be done by the party who performed the encryption (and who therefore knows the value r) and also by the party who knows the secret key corresponding to y (since this party can decrypt the ciphertext.) In the first case, this is the same as proving that $\log_y(a/m) = \log_g b$, in the second case it is proving that $\log_b(a/m) = \log_g y$. We will call the corresponding non-interactive versions of the Σ-protocols "$m = \mathrm{CC}_E(a, b, y)$"

(meaning "ciphertext contents – by encryptor") and "$m = \mathrm{CC}_D(a, b, y)$" (meaning "ciphertext contents – by decryptor".)

The details of the proof of ciphertext contents by encryptor or decryptor are straightforward and omitted.

DL-based signatures: We can use a variety of signature schemes which consist to proving knowledge of a discrete log (a private key corresponding to a public key) made relative to a message m One such scheme is the Schnorr signature scheme [36]. We will denote such a scheme DL-SIG, and will denote a non-interactive proof by a party with public key y relative to a message m by $\mathrm{DL\text{-}SIG}_y(m)$.

We omit the details of the interactive and non-interactive proofs for the Schnorr scheme.

Designated verifier proofs/signatures A designated verifier proof is a proof that is convincing to the designated verifier, but not to any other party. A proof (such as a witness hiding proof, zero-knowledge proof, or Schnorr proof) of an assertion Θ can be transformed into a a proof for a designated verifier Bob by modifying what is being proved to "Θ or 'I know Bob's private key' ". If Bob shows this proof to any other party, they would not be convinced of the veracity of Θ, since Bob could have constructed the proof himself even if Θ were not true, simply using knowledge of his private key.

An example of a designated verifier signature was given in the previous section: "$\mathrm{DL\text{-}SIG}_y(m) \vee \mathrm{DL\text{-}SIG}_{y'}(m)$." Say Alice knows the private key x associated with y and Bob knows the private key x' associated with y'. If Alice gives this proof to Bob, Bob is convinced of Alice's knowledge of x and signature on m, but if he shows this to any other party, they would not necessarily be convinced that Alice signed m.

4 Private Contract Signatures

An important technique we introduce is a type of signature we call a *private contract signature*. Roughly, these are designated verifier signatures that can be converted into universally-verifiable signatures by either the signing party or a trusted third party appointed by the signing party, whose identity can be verified by the party who is the designated verifier. More formally:

Definition 1. *A* private contract signature (PCS) *scheme Σ is a tuple of probabilistic polynomial-time algorithms* {PCS-Sign, S-Convert, TP-Convert, PCS-Ver, S-Ver, TP-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* $\text{PCS-Sign}_A(m, B, T)$*, outputs a private contract signature, denoted* $\mathrm{PCS}_A(m, B, T)$*. A private contract signature can be verified using* PCS-Ver*, i.e.,*

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

(2) $\mathsf{S\text{-}Convert}$ *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) $\mathsf{TP\text{-}Convert}$ *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* $\mathsf{S\text{-}Ver}$, *and* $\text{TP-Sig}_A(m)$ *can be verified using* $\mathsf{TP\text{-}Ver}$, *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* $\mathsf{FakeSign}$ *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* $\mathsf{S\text{-}Convert}$ *and* $\mathsf{TP\text{-}Convert}$, *respectively, are disjoint.*

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* $\mathsf{S\text{-}Convert}$ *and* $\mathsf{TP\text{-}Convert}$, *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.

Note that the scheme given in [3] that uses verifiable encryption is actually a PCS scheme, with the TP-invisibility property. That scheme is inefficient, requiring cut-and-choose techniques, but it allows for a variety of types of signatures. We now present an efficient PCS scheme in which the signatures are of a specific form.

4.1 An efficient discrete log-based PCS scheme

Say A has public key y_A and private key x_A, where $y_A = g^{x_A}$. Similarly, B has public key/private key pair (y_B, x_B), and T has public key/private key pair (y_T, x_T). The intuition is as follows. In order for A to generate a PCS on m for B, she sends B a proof of the statement

"X is a T-encryption of "1" AND I can sign m as A
OR
X is a T-encryption of "2" AND I can sign m as B"

where X is some value, and T-encryption denotes a message encrypted with T's public key. A can do this because she can perform a T-encryption of "1" to generate X, and she can sign m herself. Upon receiving this proof, B can verify its correctness, but he is not able to transfer it to anybody else, since he could have generated a similar proof himself.

A TP-accountable conversion of the above PCS by A can be done by A giving a proof that X is a T-encryption of "1" (cf. Section 3.1). The TP-accountable conversion of the PCS by T can be done similarly, i.e., by T giving a proof that X is a T-encryption of "1." The accountability will arise from the fact that the proofs given by A and T are of a different form. For a TP-invisible conversion of the PCS, both A and T output the disjunction of the two proofs above. We now define the scheme formally by describing the algorithms.

- $\mathsf{PCS\text{-}Sign}_A(m, B, T)$ works as follows. A generates an ElGamal encryption (a, b) of 1 for T and outputs $\mathrm{PCS}_A(m, B, T) = \langle (a, b), P \rangle$ where P is the following proof:

  $$([1 = \mathrm{CC}_E(a, b, y_T)] \wedge \text{DL-SIG}_{y_A}(m)) \vee ([2 = \mathrm{CC}_E(a, b, y_T)] \wedge \text{DL-SIG}_{y_B}(m)))$$

- $\mathsf{S\text{-}Convert}_A(S)$, where $S = \mathrm{PCS}_A(m, B, T)$, works as follows. For a trustee-invisible signature, A outputs $\text{S-Sig}_A(m) = \langle S, P \rangle$ where P is the following proof:

  $$[1 = \mathrm{CC}_E(a, b, y_T)] \ \vee \ [1 = \mathrm{CC}_D(a, b, y_T)].$$

  For a non-trustee-invisible signature, A outputs $\text{S-Sig}_A(m) = \langle S, P \rangle$ where P is the following proof:

  $$1 = \mathrm{CC}_E(a, b, y_T).$$

– $\mathsf{TP\text{-}Convert}_T(S)$, where $S = \mathrm{PCS}_A(m, B, T)$, works as follows. For a trustee-invisible signature, T outputs $\text{TP-Sig}_A(m) = \langle S, P\rangle$ where P is the following proof:
$$[1 = \mathrm{CC}_E(a, b, y_T)] \ \vee \ [1 = \mathrm{CC}_D(a, b, y_T)].$$
For a non-trustee-invisible signature, T outputs $\text{TP-Sig}_A(m) = \langle S, P\rangle$ where P is the following proof:
$$1 = \mathrm{CC}_D(a, b, y_T).$$

Theorem 1. *Assuming the security of DDH, the above signature scheme is a* PCS *scheme (in the random oracle model).*

Proof Sketch: First we show unforgeability of $\mathrm{PCS}_A(m, B, T) = \langle (a, b), P\rangle$. Say a polynomial-time party F could forge with non-negligible probability. Then we could use F to find x_A or x_B in polynomial-time with non-negligible probability, as follows. Assuming that the challenge length is superpolynomial in k, the Forking Lemma from Pointcheval and Stern [35] shows that in polynomial time we can find two transcripts (a, c, z) and (a, c', z') for the Σ-protocol S corresponding to P. By the definition of S, we can extract x_A or x_B, which contradicts DLA, and hence contradicts DDH.

The designated verifier property follows from the fact that by DDH, it is infeasible to distinguish an ElGamal encryption of 2 from an ElGamal encryption of 1, and the fact that B has a polynomial time algorithm to generate $\mathrm{PCS}_A(m, B, T)$. For the latter, first B generates an ElGamal encryption (a, b) of 2 using public key y_T. Then it can construct the proof P since it knows the secret value used in the encryption and it knows x_B, and thus can play the part of the prover in the Σ-protocol corresponding to the proof P needed for $\mathrm{PCS}_A(m, B, T)$.

Now we show unforgeability of $\text{S-Sig}_A(m) = \langle S, P\rangle$, where $S = \mathrm{PCS}_A(m, B, T) = \langle (a, b), P_S\rangle$. (The proof of unforgeability of $\text{TP-Sig}_A(m)$ is similar.) First, say a polynomial-time party F' could construct P with non-negligible probability such that (a, b) is not of the form (y_t^α, g^α) (i.e., (a, b) is not the encryption of "1"). Then by the Forking lemma, in polynomial time we can find two transcripts (a, c, z) and (a, c', z') for the Σ-protocol S' corresponding to P. But this is impossible, since the input is not in the language.

Thus, no polynomial-time party F' could construct P unless (a, b) is of the form (y_t^α, g^α). Now say a polynomial-time party F could forge $\text{S-Sig}_A(m) = \langle S, P\rangle$ with non-negligible probability. There are two cases.

1. F has a non-negligible probability of forging $\text{S-Sig}_A(m) = \langle S, P\rangle$ for an S generated by A. In this case, we can break DDH. Given DDH instance (g, h, r_1, r_2), set $y_T = h$, $b = r_1$ and $a = r_2$, and produce $\mathrm{PCS}_A(m, B, T)$ (simulating the random oracle h also, in order to be able to construct the PCS proof). If the DDH instance is a true Diffie-Hellman quadruple, F constructs a valid P with non-negligible probability. Otherwise F cannot.
If F constructs a valid P, we guess that the DDH instance is a true Diffie-Hellman quadruple, and otherwise, with probability 1/2 we guess that it

is a true Diffie-Hellman quadruple. Thus we guess with a non-negligible advantage.
2. F has a non-negligible probability of forging $\text{S-Sig}_A(m) = \langle S, P \rangle$ for an S not generated. Then F has a non-negligible probability of forging a PCS, which is a contradiction to our argument above.

Remark. We note that it may be possible to obtain a similar result using techniques involving encryption of signatures, and proofs that the resulting ciphertexts contain valid signatures. Efficient protocols for proving that the contents of a ciphertext is a valid signature were presented in [19] for ElGamal signatures, and [11] for RSA signatures. Combining this with the methods in [31] may result in alternative implementations.

5 Abuse-free Contract Signing

In this section, we show how to use the PCS scheme to design an abuse-free contract signing protocol. First we give a high level description. The main protocol has an alternating structure, with a previously agreed-upon initiator, say, party A. If no problems occur, A sends a private contract signature (PCS) of the message to be signed to B, and B 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. The basic, failure-free protocol is shown in Figure 1.

This is very similar to the protocol of [3], except that the primitives underlying our protocol are different, and provide different functionality. We now describe the protocol(s) in more detail. We use $[\alpha]_X$, for $X \in \{A, B, T\}$, to denote the string α concatenated with a signature of α under X's public key.

Signatory A		Signatory B
	$\xrightarrow{\text{PCS}_A(m, B, T)}$	If $\neg$ok, quit
If $\neg$ok, abort	$\xleftarrow{\text{PCS}_B(m, A, T)}$	
	$\xrightarrow{\text{S-Sig}_A(m)}$	If $\neg$ok, resolve
If $\neg$ok, resolve	$\xleftarrow{\text{S-Sig}_B(m)}$	

Fig. 1. *The basic abuse-free contract signing protocol for two parties*

Main Protocol

1. A runs $\mathsf{PCS\text{-}Sign}_A((m,1),B,T)$ and sends the result, $\mathrm{PCS}_A((m,1),B,T)$, to B.

2. If B receives a valid $\mathrm{PCS}_A((m,1),B,T)$, it runs $\mathsf{PCS\text{-}Sign}_B((m,2),A,T)$ and sends the result $\mathrm{PCS}_B((m,2),A,T)$, to A. Otherwise B simply quits.

3. If A receives a valid response $\mathrm{PCS}_B((m,2),A,T)$, it runs $\mathsf{S\text{-}Convert}_A(\mathrm{PCS}_A((m,1),B,T))$ and sends the result, $\text{S-Sig}_A((m,1))$, to B. Otherwise A runs the **Abort** protocol.

4. If B receives a valid response $\text{S-Sig}_A((m,1))$, it runs $\mathsf{S\text{-}Convert}_B(\mathrm{PCS}_B((m,2),A,T))$ and sends the result, $\text{S-Sig}_B((m,2))$, to A. Otherwise B runs B-**Resolve**.

5. If A does not receive a valid response $\text{S-Sig}_B((m,2))$, she runs resolve

Protocol Abort: To abort, A sends $[m,A,B,\text{abort}]_A$ to T.

1. If the signature is correct, and neither A nor B have resolved, T sends $[[m,A,B,\text{abort}]_A]_T$ back to A, and stores this.

2. If either A or B has resolved, it sends the corresponding stored value (i.e., $\text{S-Sig}_A((m,1))$ or $\text{S-Sig}_B((m,2))$, resp.—see below).

Protocol B-Resolve: For B to resolve, it runs $\mathsf{S\text{-}Convert}_B(\mathrm{PCS}_B((m,2),A,T))$ to produce $\text{S-Sig}_B((m,2))$, and sends the message

$$(\mathrm{PCS}_A((m,1),B,T),\text{S-Sig}_B((m,2)))$$

to T. T checks to make sure the second half corresponds to the first half (i.e., m is the same, etc.), and checks that the second half is valid. If so, then it decides what to do as follows.

1. If A has aborted, it sends the stored copy of $[[m,A,B,\text{abort}]_A]_T$ to B.

2. If A has resolved, it sends the stored copy of $\text{S-Sig}_A((m,1))$ to B.

3. Otherwise, it sends $\mathsf{TP\text{-}Convert}_T(\mathrm{PCS}_A((m,1),B,T))$ to B and stores this.

Protocol A-Resolve: For A to resolve, it runs $\mathsf{S\text{-}Convert}_A(\mathrm{PCS}_A((m,1),B,T))$ to produce $\text{S-Sig}_A((m,1))$, and sends the message

$$(\text{S-Sig}_A((m,1)),\mathrm{PCS}_B((m,2),A,T))$$

to T. T checks to make sure the first half corresponds to the second half (i.e., m is the same, etc.), and checks that the first half is valid. If so, then it decides what to do as follows.

1. If A has already aborted, it sends the stored copy of $[[m,A,B,T]_A]_T$.

2. If B has resolved, it sends the stored copy of $\text{S-Sig}_B((m,2))$.

3. If B has not resolved, it sends $\mathsf{TP\text{-}Convert}_T(\mathrm{PCS}_B((m,2),A,,T))$ to A and stores this.

Recall that the $\mathsf{S\text{-}Convert}$ and $\mathsf{TP\text{-}Convert}$ procedures can result in either TP-invisibility or TP-accountability.

Disputes: Disputes concerning contracts are handled as follows:

1. If one party shows $[[m, A, B, \text{abort}]_A]_T$ and the other shows $(\text{S-Sig}_B^B(m), \text{S-Sig}_A^A(m))$, then the contract is valid, since A must have called **Abort** after sending $\text{S-Sig}_A^A(m)$, and thus after ratifying the contract.
2. If one party shows $[[m, A, B, \text{abort}]_A]_T$ and the other shows a contract with $\text{TP-Sig}_A(m)$ or $\text{TP-Sig}_B(m)$, then if the TP-accountable conversions were used, T must have cheated.

If we assume that TP-invisibility is used and the third party is trusted, only the first case above may occur.

Theorem 2. *Assuming the security of* DDH*, the protocol above is a secure optimistic contract signing protocol, with TTP invisibility.*

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 PCS definition.

Fairness for A: Say A is honest and does not obtain a valid contract. First consider the case in which A calls **Abort** after sending $\text{PCS}_A(m, B, T)$. Since she does not obtain a valid contract, she must receive $[[m, A, B, abort]_A]_T$. Then by the security of PCS, B is not able to construct $\text{S-Sig}_A^A(m)$ or $\text{S-Sig}_A^T(m)$. The only other case is when A sends $\text{S-Sig}_A^A(m)$ to B. But then A either receives $\text{S-Sig}_B^B(m)$ or calls A-**Resolve** to receive $\text{S-Sig}_B^T(m)$, and thus this case is impossible.

Fairness for B: Say B is honest and does not obtain a valid contract. This implies A never sent $\text{S-Sig}_A^A(m)$. Then at some point B must have called B-**Resolve** and obtained $[[m, A, B, abort]_A]_T$. Then A only received $\text{PCS}_B(m, A, T)$, and by the security of PCS, could not have a valid contract.

If T is not honest, then it is easy to see that this can be detected in the TP-accountable version of the protocol, but not necessarily in the TP-invisible version.

Abuse-freeness for A: We must show that B does not obtain publicly verifiable information about (honest) A signing the contract until B is also bound by the contract. This follows by the designated verifier property of $\text{PCS}_A(m, B, T)$, and the fact that once B receives $\text{S-Sig}_A^A(m)$, B must have sent $\text{PCS}_B(m, A, T)$, and B is not able to abort. Thus either A will receive $\text{S-Sig}_B^B(m)$ from B, or A will perform A-**Resolve** to obtain $\text{S-Sig}_B^T(m)$. Either way, A receives a valid contract.

Abuse-freeness for B: We must show that A does not obtain publicly verifiable information about (honest) B signing the contract until A is also bound by the contract. This follows by the designated verifier property of $\text{PCS}_B(m, A, T)$, and the fact that once A receives $\text{S-Sig}_B^B(m)$, A must have sent $\text{S-Sig}_A^A(m, B, T)$, and thus B already has a valid contract.

Remark: We have not distinguished between off-line and on-line attacks in the above proof sketch. In fact, we only need to consider on-line attacks, as the off-line attack is a special case of the on-line attack. When we refer to the designated verifier property above, we mean the property that unless the secret key of the receiver of a proof is not known by the same, it is under no circumstances possible for this receiver to convince a third party of the validity of the proof, whether they interact or not. Therefore, it follows that the validity of the proof/signature cannot be verified by said third party, but only by the party designated by the prover/signer.

TP Invisibility: Straightforward from the definition.

Similarly, we can show

Theorem 3. *Assuming the security of* DDH*, the protocol variation above is a secure optimistic contract signing protocol with TTP accountability.*

6 Three-party Contracts

In this section we consider contract signing by $n > 2$ parties. Natural extensions to three-party contract signing do not seem to allow abuse-freeness. One problem faced in extensions to three parties is that the two party protocol is asymmetric: A has the power to abort, but B does not. With a straightforward extension to three parties, it is unclear whether to let the "middle" party abort or not, and how to maintain fairness and abuse-freenes in either case. (We leave a more complete discussion to the full version of the paper.)

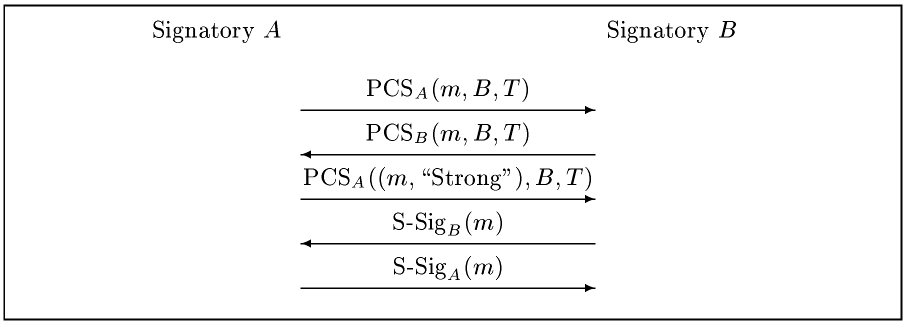

Fig. 2. *Alternative abuse-free contract signing protocol for two parties*

We now present a contract-signing protocol for three parties that is both abuse-free and fair. The approach we take is to let aborts be overturned, but in a way that avoids any violation of fairness and abuse freeness. The protocol is based on the alternative protocol for two parties shown in Figure 2. In the protocol, the roles of A and B are interchanged during the signature phase: B

sends his converted signature first, and A does it after receiving B's signature. However, B only sends his signature after receiving a new message: a "strong" promise that she won't abort the protocol. One consequence of this is that now T might have to overturn a stored abort outcome, as requested by a cheating A, to that of a valid signed contract. In the full paper we show:

Theorem 4. *Assuming the security of* DDH, *the protocol of Figure 2 is a secure optimistic contract signing protocol for two parties (in the random oracle model).*

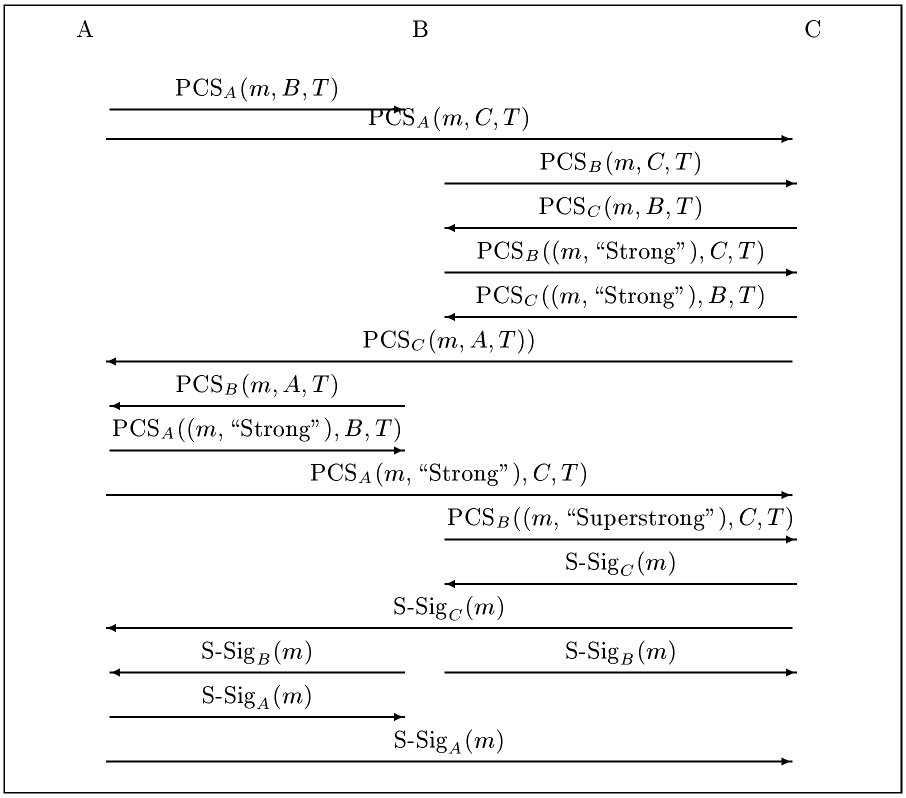

Fig. 3. *Secure contract signing protocol for three parties*

Based on the alternate protocol for two parties, a secure contract signing protocol for three parties is shown in Figure 3. As in the two-party protocol, the protocol also uses "strong" promises, but now with more levels of strength. In the three-party case, B sends also a "Superstrong" promise not to (support an) abort. Intuitively, and as a general rule, a "Strong" promise presented by a party in a call to resolve will not be used by the trustee to overturn an existing abort; on the other hand, a "Superstrong" promise will provide overturning power. All the cases are outlined in the description of the protocol, which will be presented in the full version of this paper.

Theorem 5. *Assuming the security of* DDH*, the protocol of Figure 3 is a secure optimistic contract signing protocol for three parties (in the random oracle model).*

References

1. N. Asokan, B. Baum-Waidner, M. Schunter, and M. Waidner. Optimistic synchronous multi-party contract signing. Technical Report RZ3089, IBM Research Report, 1998. to appear in Verlaessliche Informationssysteme 1999.
2. N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In ACM Security'96, pages 6–17.
3. N. Asokan, V. Shoup, and M. Waidner. Fair exchange of digital signatures. Technical Report RZ2973, IBM Research Report, 1998. Extended Abstract in Eurocrypt'98.
4. B. Baum-Waidner and M. Waidner. Optimistic asynchronous multi-party contract signing. Technical Report RZ3078, IBM Research Report, 1998.
5. M. Bellare, J. Garay, and T. Rabin. Fast batch exponentiation for modular exponentiation and digital signatures. In EUROCRYPT'98, pages 236–250.
6. M. Bellare and O. Goldreich. On defining proofs of knowledge. In CRYPTO'92, pages 390–420.
7. M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A fair protocol for signing contracts. *IEEE Trans. Info. Theory*, 36(1):40–46, 1990.
8. M. Blum. Coin flipping by telephone: A protocol for solving impossible problems. In CRYPTO'81, pages 11–15. ECE Report 82-04, 1982.
9. M. Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems*, 1(2):175–193, May 1983.
10. H. Bürk and A. Pfitzmann. Value exchange systems enabling security and unobservability. *Computers and Security*, 9:715–721, 1990.
11. D. Catalano and R. Gennaro. New efficient and secure protocols for verifiable signature sharing and other applications. In CRYPTO'98, pages 105–120.
12. D. Chaum. Designated confirmer signatures. In EUROCRYPT'94, pages 86–91.
13. D. Chaum and H. V. Antwerpen. Undeniable signatures. In CRYPTO'89, pages 212–216.
14. D. Chaum and T. P. Pedersen. Transferred cash grows in size. In EUROCRYPT'92, pages 390–407.
15. D. L. Chaum. Silo watching. In CRYPTO'81, pages 138–139. ECE Report 82-04, 1982.
16. B. Cox, J. D. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *First USENIX Workshop on Electronic Commerce*, pages 77–88, 1995.
17. R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols.* PhD thesis, University of Amsterdam, 1995.
18. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In CRYPTO'94, pages 174–187.
19. I. Damgård and T. Pedersen. New convertible undeniable signature schemes. In EUROCRYPT'96, pages 372–386.

20. I. B. Damgård. On the existence of bit commitment schemes and zero-knowledge proofs. In CRYPTO'89, pages 17–27.
21. I. B. Damgård. Practical and provably secure release of a secret and exchange of signatures. *J. of Crypt.*, 8(4):201–222, Autumn 1995.
22. R. H. Deng, L. Gong, A. A. Lazar, and W. Wang. Practical protocols for certified electronic mail. *J. of Network and Systems Management*, 4(3), 1996.
23. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Trans. Info. Theory*, 31:465–472, 1985.
24. S. Even. A protocol for signing contracts. *ACM SIGACT News*, 15(1):34–39, 1983.
25. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.
26. U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In CRYPTO'89, pages 526–545.
27. M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed commit with one faulty process. *J. ACM*, 32(2), 1985.
28. M. Franklin and M. Reiter. Fair exchange with a semi-trusted third party. In ACM SECURITY'96, pages 1–5.
29. J. Garay and P. MacKenzie. Multi-party contract signing. Manuscript.
30. O. Goldreich. A simple protocol for signing contracts. In CRYPTO'83, pages 133–136.
31. M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In EUROCRYPT'96, pages 143–154.
32. C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In CRYPTO'94, pages 95–107.
33. S. Micali. Certified e-mail with invisible post offices. Presented at the 1997 RSA Security Conference, 1997.
34. M. Michels, H. Petersen and P. Horster. Breaking and repairing a convertible undeniable signature scheme. In ACM SECURITY'96, pages 148–152.
35. D. Pointcheval and J. Stern. Security proofs for signature schemes. In EUROCRYPT'96, pages 387–398.
36. C. P. Schnorr. Efficient identification and signatures for smart cards. In CRYPTO'89, pages 239–252.