Legally Fair Contract Signing Without Keystones

Houda Ferradi, Rémi Géraud, Diana Maimut, David Naccache, David Pointcheval

Legally Fair Contract Signing Without Keystones

Houda Ferradi, Rémi Géraud, Diana Maimuț,
David Naccache, and David Pointcheval

École normale supérieure
45 rue d'Ulm, F-75230 Paris CEDEX 05, France
given_name.family_name@ens.fr

Abstract. In two-party computation, achieving both fairness and guaranteed output delivery is well known to be impossible. Despite this limitation, many approaches provide solutions of practical interest by weakening somewhat the fairness requirement. Such approaches fall roughly in three categories: "gradual release" schemes assume that the aggrieved party can eventually reconstruct the missing information; "optimistic schemes" assume a trusted third party arbitrator that can restore fairness in case of litigation; and "concurrent" or "legally fair" schemes in which a breach of fairness is compensated by the aggrieved party having a digitally signed cheque from the other party (called the keystone).
In this paper we describe and analyse a new contract signing paradigm that doesn't require keystones to achieve legal fairness, and give a concrete construction based on Schnorr signatures which is compatible with standard Schnorr signatures and provably secure.

1 Introduction

When mutually distrustful parties wish to compute some joint function of their private inputs, they require a certain number of security properties to hold for that computation:

- *Privacy*: Nothing is learnt from the protocol besides the output;
- *Correctness*: The output is distributed according to the prescribed functionality;
- *Independence*: One party cannot make their inputs depend on the other parties' inputs;
- *Delivery*: An adversary cannot prevent the honest parties from successfully computing the functionality;
- *Fairness*: If one party receives output then so do all.

Any multi-party computation can be securely computed [5, 7, 16, 17, 30] as long as there is a honest majority [22]. In the case where there is no such majority, and in particular in the two-party case, it is (in general[1]) impossible to achieve both fairness and guaranteed output delivery [9, 22].

[1] See [19] for a very specific case where completely fair two-party computation can be achieved.

Weakening Fairness. To circumvent this limitation, several authors have put forth alternatives to fairness that try and capture the practical context (*e.g.* contract-signing, bank transactions, etc.). Three main directions have been explored:

1. *Gradual release models*: The output is not revealed all at once, but rather released gradually (*e.g.* bit per bit) so that, if an abort occurs, then the adversary has not learnt much more about the output than the honest party. This solution is unsatisfactory because it is expensive and may not work if the adversary is more computationally powerful [13, 18, 19, 24].
2. *Optimistic models*: A trusted server is setup but will not be contacted unless fairness is breached. The server is able to restore fairness afterwards, and this approach can be efficient, but the infrastructure requirements and the condition that the server be trusted limit the applicability of this solution [3,6,23]. In particular, the dispute-resolving third party must be endowed with functions beyond those usually required of a normal certification authority.
3. *Legally fair, or concurrent model*: The first party to receive output obtains an information dubbed the "keystone". The keystone by itself gives nothing and so if the first party aborts after receiving it, no damage has been done – if the second party aborts after receiving the result (say, a signature) then the first party is left with a useless keystone. But, as observed in [8] for the signature to be enforced, it needs to be presented to a court of law, and legally fair signing protocols are designed so that this signature *and* the keystone give enough information to reconstruct the missing data. Therefore, if the cheating party wishes to enforce its signed contract in a court of law, it by doing so reveal the signature that the first party should receive, thereby restoring fairness [8]. Legal fairness requires neither a trusted arbitrator nor a high degree of interaction between parties.

Lindell [22] also introduces a notion of "legally enforceable fairness" that sits between legal fairness and optimistic models: a trusted authority may force a cheating party to act in some fashion, should their cheating be attested. In this case the keystone consists in a digitally signed cheque for an frighteningly high amount of money that the cheating party would have to pay if the protocol were to be aborted prematurely and the signature abused.

Concurrent Signatures. Chen *et al.* [8] proposed a legally fair signature scheme based on ring signatures [2, 27] and designated verifier signatures [21], that is proven secure in the Random Oracle Model assuming the hardness of computing discrete logarithms.

Concurrent signatures rely on a property shared by ring and designated verifier signatures called "ambiguity". In the case of two-party ring signatures, one cannot say which of the two parties produced the signature – since either of two parties could have produced such an ambiguous signature, both parties can deny having produced it. However, within the ring, if A receives a signature then she knows that it is B who sent it. The idea is to put the ambiguity-lifting

information in a "keystone". When that keystone is made public, both signatures become simultaneously binding.

Concurrent signatures schemes can achieve legal fairness depending on the context. However their construction is not *abuse-free* [4,12]: the party A holding the keystone can always determine whether to complete or abort the exchange of signatures, and can demonstrate this by showing an outside party the signature from B with the keystone, before revealing the keystone to B.

Our Results. In this work we describe a new contract signing protocol that achieves legal fairness and abuse-freeness. This protocol is based on the well-known Schnorr signature protocol, and produces signatures *compatible* with standard Schnorr signatures. For this reason, and as we demonstrate, the new contract signing protocol is provably secure in the random oracle model under the hardness assumption of solving the discrete logarithm problem. Our construction can be adapted to other DLP schemes, such as most[2] of those enumerated in [20], including Girault-Poupard-Stern [14] and ElGamal [10].

2 Preliminaries

We assume the reader to be familiar with Schnorr signatures, that we recall in Appendix A.

2.1 Concurrent Signatures

Let us give a more formal account of legal fairness as described in [8,22] in terms of concurrent signatures. Unlike classical contract-signing protocol, whereby contractors would exchange full-fledged signatures (*e.g.* [15]), in a concurrent signature protocol there are "ambiguous" signatures that do not, as such, bind their author. This ambiguity can later be lifted by revealing some additional information: the "keystone". When the keystone is made public, both signatures become simultaneously binding.

Let $\mathcal{M}$ be a message space. Let $\mathcal{K}$ be the keystone space and $\mathcal{F}$ be the keystone fix space.

Definition 1 (Concurrent signature). *A concurrent signature is composed of the following algorithms:*

- $\mathsf{Setup}(\ell)$*: Takes a security parameter* ℓ *as input and outputs the public keys* (y_A, y_B) *of all participants, a function* $\mathsf{KeyGen} : \mathcal{K} \to \mathcal{F}$*, and public parameters* pp *describing the choices of* $\mathcal{M}, \mathcal{K}, \mathcal{F}$ *and* KeyGen*.*

[2] In a number of cases, *e.g.* DSA, the formulae of s do not lend themselves to security proofs.

– $\mathsf{aSign}(y_i, y_j, x_i, h_2, M)$*: Takes as input the public keys* y_1 *and* y_2*, the private key* x_i *corresponding to* y_i*, an element* $h_2 \in \mathcal{F}$ *and some message* $M \in \mathcal{M}$*; and outputs an "ambiguous signature"*

$$\sigma = \langle s, h_1, h_2 \rangle$$

where $s \in \mathcal{S}, h_1, h_2 \in \mathcal{F}$*.*

– $\mathsf{aVerify}(\sigma, y_i, y_j, M)$*: Takes as input an ambiguous signature* $\sigma = \langle s, h_1, h_2 \rangle$*, public keys* y_i *and* y_j*, a message* M*; and outputs a boolean value, with the constraint that*

$$\mathsf{aVerify}\,(\sigma', y_j, y_i, M) = \mathsf{aVerify}\,(\sigma, y_i, y_j, M)$$

where $\sigma' = \langle s, h_2, h_1 \rangle$*.*

– $\mathsf{Verify}(k, \sigma, y_i, y_j, M)$*: Takes as input* $k \in \mathcal{K}$ *and* σ, y_i, y_j, M *as above; and checks whether* $\mathsf{KeyGen}(k) = h_2$*: If not it terminates with output* False*, otherwise it outputs the result of* $\mathsf{aVerify}(\sigma, y_i, y_j, M)$*.*

A valid concurrent signature is a tuple $\langle k, \sigma, y_i, y_j, M \rangle$ that is accepted by the Verify algorithm. Concurrent signatures are used by two parties A and B in the following way:

1. A and B run Setup to determine the public parameters of the scheme. We assume that A's public and private keys are y_A and x_A, and B's public and private keys are y_B and x_B.
2. Without loss of generality, we assume that A initiates the conversation. A picks a random keystone $k \in \mathcal{K}$, and computes $f = \mathsf{KeyGen}(k)$. A takes her own public key y_A and B's public key y_B and picks a message $M_A \in \mathcal{M}$ to sign. A then computes her ambiguous signature to be

$$\sigma_A = \langle s_A, h_A, f \rangle = \mathsf{aSign}(y_A, y_B, x_A, f, M_A).$$

3. Upon receiving A's ambiguous signature σ_A, B verifies the signature by checking that

$$\mathsf{aVerify}(s_A, h_A, f, y_A, y_B, M_A) = \mathsf{True}$$

If this equality does not hold, then B aborts. Otherwise B picks a message $M_B \in \mathcal{M}$ to sign and computes his ambiguous signature

$$\sigma_B = \langle s_B, h_B, f \rangle = \mathsf{aSign}(y_B, y_A, x_B, f, M_B)$$

then sends this back to A. Note that B uses the same value f in his signature as A did to produce σ_A.

4. Upon receiving B's signature σ_B, A verifies that

$$\mathsf{aVerify}(s_B, h_B, f, y_B, y_A, M_B) = \mathsf{True}$$

where f is the same keystone fix as A used in the previous steps. If the equality does not hold, then A aborts. Otherwise A sends keystone k to B.

At the end of this protocol, both $\langle k, \sigma_A \rangle$ and $\langle k, \sigma_B \rangle$ are binding, and accepted by the Verify algorithm.

Remark 1. Note that A has an the upper hand in this protocol: Only when A releases the keystone do both signatures become simultaneously binding, and there is no guarantee that A will ever do so. Actually, since A controls the timing of the keystone release (if it is released at all), A may only reveal k to a third party C but withhold it from B, and gain some advantage by doing so. In other terms, concurrent signatures can be *abused* by A [4, 12].

Chen *et al.* [8] argue that there are situations where it is not in A's interest to try and cheat B, in which abuse-freeness is not necessary. One interesting scenario is credit card payment in the "four corner" model. Assume that B's signature is a payment to A. To obtain payment, A must channel *via* her acquiring bank C, which would communicate with B's issuing bank D. D would ensure that B receives both the signature and the keystone — as soon as this happens A is bound to her signature. Since in this scenario there is no possibility for A to keep B's signature private, fairness is eventually restored.

Example 1. A concurrent signature scheme based on the ring signature algorithm of Abe *et al.* [2] was proposed by Chen *et al.* [8]:

- Setup: On input a security parameter ℓ, two large primes p and q are selected such that $q|p-1$. An element $g \in \mathbb{Z}_p^\times$ of order q is selected. The spaces $\mathcal{S} = \mathcal{F} = \mathbb{Z}_q$ and $\mathcal{M} = \mathcal{K} = \{0,1\}^*$ are chosen. Two cryptographic hash functions $H_1, H_2 : \{0,1\}^* \to \mathbb{Z}_q$ are selected and we set KeyGen $= H_1$. Private keys x_A, x_B are selected uniformly at random from $\mathbb{Z}_q$ and the corresponding public keys are computed as $g^{x_i} \bmod p$.
- aSign: The algorithms takes as input y_i, y_j, x_i, h_2, M, verifies that $y_i \neq y_j$ (otherwise aborts), picks a random value $t \in \mathbb{Z}_q$ and computes

$$\begin{aligned} h &= H_2\left(g^t y_j^{h_2} \bmod p \| M\right) \\ h_1 &= h - h_2 \bmod q \\ s &= t - h_1 x_i \bmod q \end{aligned}$$

  where $\|$ denotes concatenation. The algorithm outputs $\langle s, h_1, h_2 \rangle$.
- aVerify: This algorithm takes as input s, h_1, h_2, y_i, y_j, M and checks whether the following equation holds:

$$h_1 + h_2 = H_2\left(g^s y_i^{h_1} y_j^{h_2} \bmod p \| M\right) \bmod q$$

The security of this scheme can be proven in the Random Oracle model assuming the hardness of computing discrete logarithms in $\mathbb{Z}_p^\times$.

2.2 Legal Fairness for Concurrent Signatures

A concurrent signature scheme is secure when it achieves existential unforgeability, ambiguity and fairness against an active adversary that has access to a signature

oracle. We define these notions in terms of games played between the adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In all security games, $\mathcal{A}$ can perform any number of the following queries:

- KeyGen queries: $\mathcal{A}$ can receive a keystone fix $f = \mathsf{KeyGen}(k)$ where k is chosen by the challenger[3].
- KeyReveal queries: $\mathcal{A}$ can request that $\mathcal{C}$ reveals which k was chosen to produce a keystone fix f in a previous KeyGen query. If f was not a previous KeyGen query output then $\mathcal{C}$ returns $\perp$.
- aSign queries: $\mathcal{A}$ can request an ambiguous signature for any message of his choosing and any pair of users[4].
- SKExtract queries: $\mathcal{A}$ can request the private key corresponding to a public key.

Definition 2 (Unforgeability). *The notion of existential unforgeability for concurrent signatures is defined in terms of the following security game:*

1. *The* Setup *algorithm is run and all public parameters are given to* $\mathcal{A}$.
2. $\mathcal{A}$ *can perform any number of queries to* $\mathcal{C}$*, as described above.*
3. *Finally,* $\mathcal{A}$ *outputs a tuple* $\sigma = \langle s, h_1, f\rangle$ *where* $s \in \mathcal{S}, h_1, f \in \mathcal{F}$*, along with public keys* y_C, y_D *and a message* $M \in \mathcal{M}$.

$\mathcal{A}$ *wins the game if* aVerify *accepts* σ *and either of the following holds:*

- $\mathcal{A}$ *did not query SKExtract on* y_C *nor on* y_D*, and did not query aSign on* (y_C, y_D, f, M) *nor on* (y_D, y_C, h_1, M).
- $\mathcal{A}$ *did not query aSign on* (y_C, y_i, f, M) *for any* $y_i \neq y_C$*, and did not query SKExtract on* y_C*, and* f *is the output of* KeyGen*: either an answer to a KeyGen query, or* $\mathcal{A}$ *can produce a* k *such that* $k = \mathsf{KeyGen}(k)$.

The last constraint in the unforgeability security game corresponds to the situation where $\mathcal{A}$ knows one of the private keys (as is the case if $\mathcal{A} = A$ or B).

Definition 3 (Ambiguity). *The notion of ambiguity for concurrent signatures is defined in terms of the following security game:*

1. *The* Setup *algorithm is run and all public parameters are given to* $\mathcal{A}$.
2. *Phase 1:* $\mathcal{A}$ *can perform any number of queries to* $\mathcal{C}$*, as described above.*
3. *Challenge:* $\mathcal{A}$ *selects a challenge tuple* (y_i, y_j, M) *where* y_i, y_j *are public keys and* $M \in \mathcal{M}$*. In response,* $\mathcal{C}$ *selects a random* $k \in \mathcal{K}$*, a random* $b \in \{0, 1\}$ *and computes* $f = \mathsf{KeyGen}(k)$*. If* $b = 0$*, then* $\mathcal{C}$ *outputs*

$$\sigma_1 = \langle s_1, h_1, f\rangle = \mathsf{aSign}(y_i, y_j, x_i, f, M)$$

Otherwise, if $b = 1$ *then* $\mathcal{C}$ *computes*

$$\sigma_2 = \langle s_2, h_2, f\rangle = \mathsf{aSign}(y_j, y_i, x_i, f, M)$$

but outputs $\sigma_2' = \langle s_2, f, h_2\rangle$ *instead.*

[3] The algorithm KeyGen being public, $\mathcal{A}$ can compute $\mathsf{KeyGen}(k)$ for any k of her choosing.

[4] Note that with this information and using KeyGen queries, $\mathcal{A}$ can obtain concurrent signatures for any message and any user pair.

4. *Phase 2:* $\mathcal{A}$ *can perform any number of queries to* $\mathcal{C}$*, as described above.*
5. *Finally,* $\mathcal{A}$ *outputs a guess bit* $b' \in \{0, 1\}$.

$\mathcal{A}$ *wins the game if* $b = b'$ *and if* $\mathcal{A}$ *made no KeyReveal query on* f*,* h_1 *or* h_2.

Definition 4 (Fairness). *The notion of fairness for concurrent signatures is defined in terms of the following security game:*

1. *The* Setup *algorithm is run and all public parameters are given to* $\mathcal{A}$.
2. $\mathcal{A}$ *can perform any number of queries to* $\mathcal{C}$*, as described above.*
3. *Finally,* $\mathcal{A}$ *chooses two public keys* y_C, y_D *and outputs* $k \in \mathcal{K}$ *and* $S = (s, h_1, f, y_C, y_D, M)$ *where* $s \in \mathcal{S}$*,* $h_1, f \in \mathcal{F}$*,* $M \in \mathcal{M}$.

$\mathcal{A}$ *wins the game if* $\mathsf{aVerify}(S)$ *accepts and either of the following holds:*

- f *was output from a KeyGen query, no KeyReveal query was made on* f*, and* Verify *accepts* $\langle k, S \rangle$.
- $\mathcal{A}$ *can output* $S' = (s', h_1', f, y_D, y_C, M')$ *where* $\mathsf{aVerify}(S')$ *accepts and* $\mathsf{Verify}(k, S)$ *accepts, but* $\mathsf{Verify}(k, S')$ *rejects.*

This definition of fairness formalizes the idea that B cannot be left in a position where a keystone binds his signature to him while A's initial signature is not also bound to A. It does not, however, guarantee that B will ever receive the necessary keystone.

3 Legally Fair Co-Signatures

3.1 Legal Fairness Without Keystones

The main idea builds on the following observation: Every signature exchange protocol is plagued by the possibility that the last step of the protocol is not performed. Indeed, it is in the interest of a malicious party to get the other party's signature without revealing its own. As a result, the best one can hope for is that a trusted third party can eventually restore fairness.

To avoid this destiny, the proposed paradigm does *not* proceed by sending A's signature to B and vice versa. Instead, we construct a *joint signature*, or *co-signature*, of both A and B. By design, there are no signatures to steal — and stopping the protocol early does not give the stopper a decisive advantage. More precisely, the contract they have agreed upon is the best thing an attacker can gather, and if she ever wishes to enforce this contract by presenting it to a court of law, she would confirm her own commitment to it as well as the other party's. Therefore, if one can construct co-signatures without intermediary individual signatures being sent, legal fairness can be achieved without keystones.

Since keystones can be used by the party having them to abuse the other [8], the co-signature paradigm provides an interesting alternative to concurrent signatures.

3.2 Schnorr Co-signatures

To illustrate the new paradigm, we now discuss a legally fair contract-signing protocol built from the well-known Schnorr signature protocol, that produces signatures *compatible* with standard Schnorr signatures. This contract signing protocol is provably secure in the random oracle model under the hardness assumption of solving the discrete logarithm problem.

The construction can be adapted to other DLP schemes, such as most[5] of those enumerated in [20], including Girault-Poupard-Stern [14] and ElGamal [10].

– Setup: An independent (not necessarily trusted) authority generates a classical Schnorr parameter-set p, q, g which is given to A and B. Each user U generates a usual Schnorr public key $y_U = g^{x_U}$ and publishes y_U on a public directory $\mathcal{D}$ (see Figure 1). To determine the co-signature public-key $y_{A,B}$ of the pair $\langle A, B\rangle$, a verifier consults $\mathcal{D}$ and simply computes $y_{A,B} = y_A y_B$. Naturally, $y_{A,B} = y_{B,A}$.

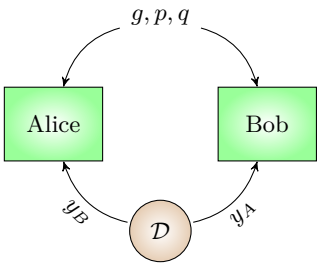

Fig. 1. Public directory $\mathcal{D}$ distributing the public keys.

– Cosign: To co-sign a message m, A and B compute a common r and a common s, one after the other. Without loss of generality we assume that B initiates the co-signature.
  - During the first phase (Figure 2), B chooses a private random number k_B and computes $r_B \leftarrow g^{k_B}$. He commits to that value by sending to A a message digest $\rho \leftarrow H(0\|r_B)$. A chooses a private random number k_A, computes $r_A \leftarrow g^{k_A}$ and sends r_A to B. B replies with r_B, which A checks against the earlier commitment ρ. Both parties compute $r \leftarrow r_A r_B$, and $e \leftarrow H(1\|m\|r)$, where m is the message to be co-signed.
  - During the second phase of the protocol, B sends $s_B \leftarrow k_B - ex_B \bmod q$ to A. A replies with $s_A \leftarrow k_A - ex_A \bmod q$. Both users compute $s \leftarrow s_A + s_B \bmod q$.

– Verify: As in the classical Schnorr signature, the co-signature $\{r, s\}$ is checked for a message m by computing $e \leftarrow H(m\|r)$, and checking whether $g^s y^e = r$

[5] In a number of cases, *e.g.* DSA, the formulae of s do not lend themselves to security proofs.

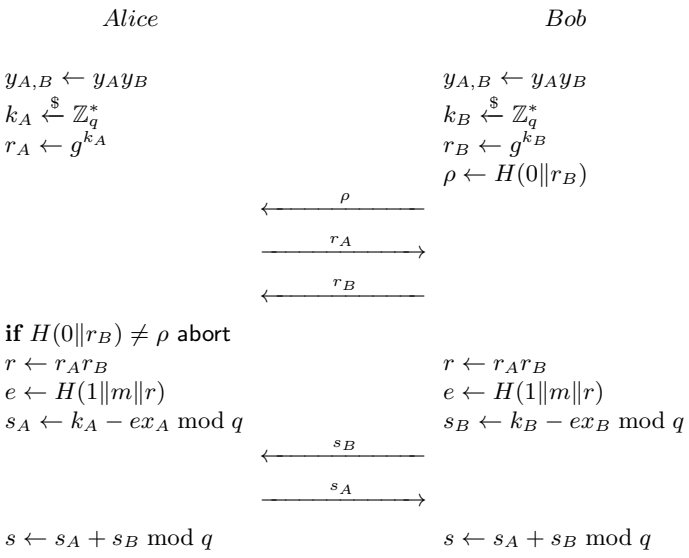

Fig. 2. Generating the Schnorr co-signature of message m.

(Figure 3). If the equality holds, then the co-signature binds both A and B to m; otherwise neither party is tied to m.

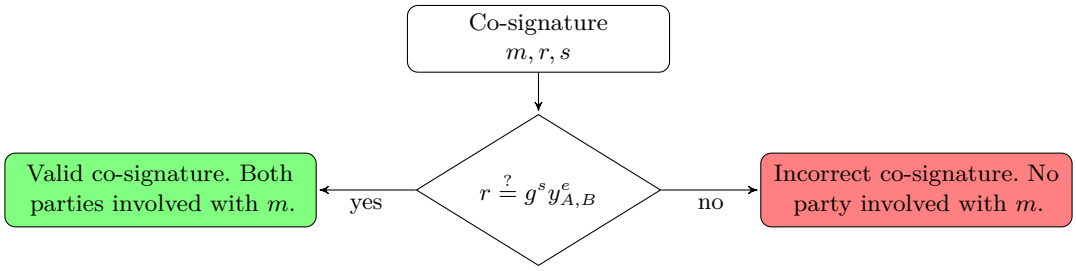

Fig. 3. Verification of a Schnorr co-signature m, r, s.

Remark 2. Note that during the co-signature protocol, A might decide not to respond to B: In that case, A would be the only one to have the complete co-signature. This is a breach of fairness insofar as A can benefit from the co-signature and not B, but the protocol is abuse-free: A cannot use the co-signature as a proof that B, and B alone, committed to m. Furthermore, it is not a breach of legal fairness: If A presents the co-signature in a court of law, she *ipso facto* reveals her commitment as well.

Remark 3. In a general fair-contract signing protocol, A and B can sign different messages m_A and m_B. Using the co-signature construction requires that A and B agree first on the content of a single message m.

3.3 Security Analysis

The security of the co-signature scheme essentially builds on the unforgeability of classical Schnorr signatures. Since there is only one co-signature output, the notion of ambiguity does not apply *per se* — albeit we will come back to that point later on. The notion of fairness is structural in the fact that a co-signature, as soon as it is binding, is binding for *both* parties.

As for concurrent signatures, an adversary $\mathcal{A}$ has access to an unlimited amount of conversations and valid co-signatures, *i.e.* $\mathcal{A}$ can perform the following queries:

- Hash queries: $\mathcal{A}$ can request the value of $H(x)$ for a x of its choosing.
- CoSign queries: $\mathcal{A}$ can request a valid co-signature r, s for a message m and a public key $y_{C,D}$ of its choosing.
- Transcript queries: $\mathcal{A}$ can request a valid transcript $(\rho, r_C, r_D, s_C, s_D)$ of the co-signing protocol for a message m of its choosing, between users C and D of its choosing.
- SKExtract queries: $\mathcal{A}$ can request the private key corresponding to a public key.
- Directory queries: $\mathcal{A}$ can request the public key of any user U.

The following definition captures the notion of unforgeability in the co-signing context:

Definition 5 (Unforgeability). *The notion of unforgeability for co-signatures is defined in terms of the following security game between the adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:*

1. *The* Setup *algorithm is run and all public parameters are provided to $\mathcal{A}$.*
2. *$\mathcal{A}$ can perform any number of queries to $\mathcal{C}$, as described above.*
3. *Finally, $\mathcal{A}$ outputs a tuple $(m, r, s, y_{C,D})$.*

$\mathcal{A}$ wins the game if $\mathsf{Verify}(m, r, s) = \mathsf{True}$ *and there exist public keys $y_C, y_D \in \mathcal{D}$ such that $y_{C,D} = y_C y_D$ and either of the following holds:*

- *$\mathcal{A}$ did not query SKExtract on y_C nor on y_D, and did not query CoSign on $m, y_{C,D}$, and did not query Transcript on m, y_C, y_D nor m, y_D, y_C.*
- *$\mathcal{A}$ did not query Transcript on m, y_C, y_i for any $y_i \neq y_C$ and did not query SKExtract on y_C, and did not query CoSign on m, y_C, y_i for any $y_i \neq y_C$.*

We shall say that a co-signature scheme is unforgeable when the success probability of $\mathcal{A}$ in this game is negligible.

To prove that the Schnorr-based scheme described above is secure we use the following strategy: Assuming an efficient forger $\mathcal{A}$ for the co-signature scheme, we turn it into an efficient forger $\mathcal{B}$ for Schnorr signatures, then invoke the Forking Lemma to prove the existence of an efficient solver $\mathcal{C}$ for the discrete logarithm problem. All proofs hold in the Random Oracle model.

Since the co-signing protocol gives the upper hand to the last-but-one speaker there is an asymmetry: Alice has more information than Bob. Therefore we address two scenarios: When the attacker plays Alice's role, and when the attacker plays Bob's.

Theorem 1. *Let $\{y, g, p, q\}$ be a DLP instance. If $\mathcal{A}$ plays the role of Bob (resp. Alice) and is able to forge in polynomial time a co-signature with probability ϵ_F, then in the Random Oracle model $\mathcal{A}$ can break the DLP instance with high probability in polynomial time.*

Proof. See Appendix B, where this theorem is split in twain depending on whether $\mathcal{A}$ impersonates Bob or Alice. □

4 Concurrent Co-signatures

4.1 Proofs of Involvment

We now address a subtle weakness in the protocol described in the previous section, which is not captured by the fairness property *per se* and that we refer to as the existence of "proofs of involvment". Such proofs are not valid co-signatures, and would not normally be accepted by verifiers, but they nevertheless are valid evidence establishing that one party committed to a message. In a legally fair context, it may happen that such evidence is enough for one party to win a trial against the other — who lacks both the co-signature, and a proof of involvment.

Example 2. In the co-signature protocol of Figure 2, s_B is not a valid Schnorr signature for Bob. Indeed, we have $g^{s_B} y_B^e = r_B \neq r$. However, Alice can construct $s' = s_B + k_A$, so that m, r, s' forms a valid classical signature of Bob *alone* on m.

Example 2 illustrates the possibility that an adversary, while unable to forge a co-signature, may instead use the information to build a valid (mono-) signature. Note that Alice may opt for a weaker proof of involvment, for instance by demonstrating her possession of a valid signature using any zero-knowledge protocol.

A straightforward patch is to refrain from using the public keys y_A, y_B for both signature and co-signature — so that attempts at constructing proofs of involvment become vain. For instance, every user could have a key $y_U^{(1)}$ used for classical signature and for certifying a key $y_U^{(2)}$ used for co-signature[6]. If an

[6] The key $y_U^{(2)}$ may be derived from $y_U^{(1)}$ in some way, so that the storage needs of $\mathcal{D}$ are the same as for classical Schnorr.

adversary generates a classical signature from a co-signature transcript as in Example 2, she actually reveals her harmful intentions.

However, while this exposes the forgery — so that honest verifiers would reject such a signature — the perpetrator remains anonymous. There are scenarios in which this is not desirable, *e.g.* because it still proves that B agreed (with some unknown and dishonest partner) on m.

Note that the existence of proof of involvment is not necessary and depends on the precise choice of underlying signature scheme.

4.2 Security Model

It is important to make extremely clear the security model that we are targeting. In this situation an adversary $\mathcal{A}$ (possibly Alice or Bob) tries to forged signatures from partial and/or complete traces of co-signature interactions, which can be of two kinds :

1. Co-signatures between two parties, at least one of which did not take part in the co-signature protocol;
2. (Traditional) signatures of either party.

$\mathcal{A}$ succeeds if and only if one of these forgeries is accepted, which can be captured as the probability of acceptance of $\mathcal{A}$'s outputs, and the victim (purported mono-signatory, or co-signatory) doesn't have a co-signature with $\mathcal{A}$[7].

Observe that due to the unforgeability of Schnorr signatures, the attacker must necessarily impersonate one of the co-signatories to achieve either of the two forgeries mentioned above (in fact, the strongest position is that of Alice, who has an edge over Bob in the protocol). This is the reason why the victim may have a co-signature of $\mathcal{A}$, so that this security model captures fairness.

In short, we propose to address such attacks in the following way:

1. By using a different key for co-signature and mono-signature;
2. By having Bob store specific co-signature-related information in non-volatile memory.

The reason for (1) is that it distinguishes between mono-signatures, and mono-signatures generated from partial co-signature traces. Thanks to this, it is easy for the verifier to detect a forgery, and perform additional steps.

The reason for (2) is twofold: On the one hand, it enables the verifier to obtain from Bob definitive proof that there was forgery; on the other hand, once the forgery has been identified, it makes it possible for the verifier to re-establish fairness binding the two real co-signatories together. Note that Bob is in charge of keeping this information secure, i.e. available and correct.

[7] In particular, the question of whether Bob "intended" to sign is outside the scope of this security model.

4.3 Concurrent Co-signatures

In the interest of fairness, the best we can ask is that if A tries to incriminate B on a message they both agreed upon, she cannot do so anonymously.

To enforce fairness on the co-signature protocol, we ask that the equivalent of a keystone is transmitted first; so that in case of dispute, the aggrieved party has a legal recourse. First we define the notion of an authorized signatory credential:

Definition 6 (Authorized signatory credential). *The data field*

$$\Gamma_{Alice,Bob} = \{Alice, Bob, k_A, \sigma_{x_A}(g^{k_A} \| Alice \| Bob)\}$$

is called an authorized signatory credential *given by Alice to Bob, where σ_{x_A} is some publicly known auxiliary signature algorithm using Alice's private key x_A as a signing key.*

Any party who gets $\Gamma_{\text{Alice,Bob}}$ can check its validity, and releasing $\Gamma_{\text{Alice,Bob}}$ is *by convention* functionally equivalent to Alice giving her private key x_A to Bob. A valid signature by Bob on a message m exhibited with a valid $\Gamma_{\text{Alice,Bob}}$ is *legally* defined as encompassing the meaning ($\Rrightarrow$) of Alice's signature on m:

$$\{\Gamma_{\text{Alice,Bob}}, \text{signature by Bob on } m\} \Rrightarrow \text{signature by Alice on } m$$

Second, the co-signature protocol of Figure 2 is modified by requesting that Alice provide $t = \sigma_{x_A}(g^{k_A} \| \text{Alice} \| \text{Bob})$ to Bob. Bob stores this in a local non-volatile memory $\mathcal{L}$ along with s_B. For all practical purposes, $\mathcal{L}$ can be simply regarded as Bob's hard disk. Together, t and s_B act as a keystone enabling Bob (or a verifier, *e.g.* a court of law) to reconstruct $\Gamma_{\text{Alice,Bob}}$ if Alice exhibits a (fraudulent) signature binding Bob alone with his co-signing public key.

Therefore, should Alice try to exhibit as in Example 2 a signature of Bob alone on a message they both agreed upon (which is known as a fraud), the court would be able to identify Alice as the fraudster.

The modified signature protocol is described in Figure 4. Alice has only one window of opportunity to try and construct a fraudulent signature of Bob: by stopping the protocol at breakpoint (2) and using the information s_B[8].

Indeed, if the protocol is interrupted before breakpoint (1), then no information involving m was released by any of the parties: The protocol's trace can be simulated without Bob's help as follows

$$\begin{aligned} s_B, r &\xleftarrow{\$} \mathbb{Z}_q \\ e &\leftarrow H(1 \| m \| r \| \text{Alice} \| \text{Bob}) \\ r_B &\leftarrow g^{s_B} y_B^e \\ r_A &\leftarrow r r_B^{-1} \\ t &\leftarrow \sigma_{x_A}(r_A \| \text{Alice} \| \text{Bob}) \\ \rho &\leftarrow H(0 \| r_B) \end{aligned}$$

[8] If Bob transmits a wrong or incorrect s_B, this will be immediately detected by Alice as $r_B \neq g^{s_B} y_B^e$. Naturally, in such a case, Bob never sent any information binding him to the contract anyway.

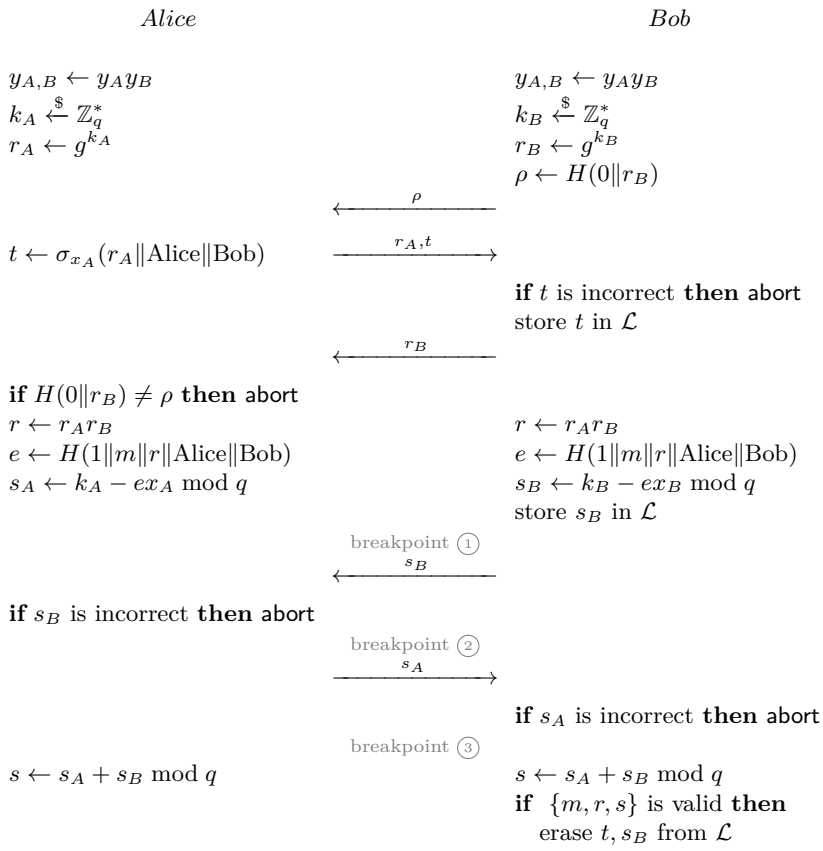

Fig. 4. The legally fair co-signature of message m.

and Bob has only received from Alice the signature of a random integer.

If Alice and Bob successfully passed the normal completion breakpoint ③, *both* parties have the co-signature, and are provably committed to m.

5 Conclusion and Further Work

In this paper we described an alternative construction paradigm for legally fair contract signing that doesn't require keystones, but can be combined with them to provide additional power. The new paradigm produces co-signatures that bind a pair of users, and can be adapted to a number of DLP signature protocols. In the co-signature version of Schnorr's protocol, the resulting co-signatures have the same format as classical (single-user) signature. This paradigm guarantees fairness and abuse-freeness, and can be equipped with keystones to add functionalities such as whistleblower traceability.

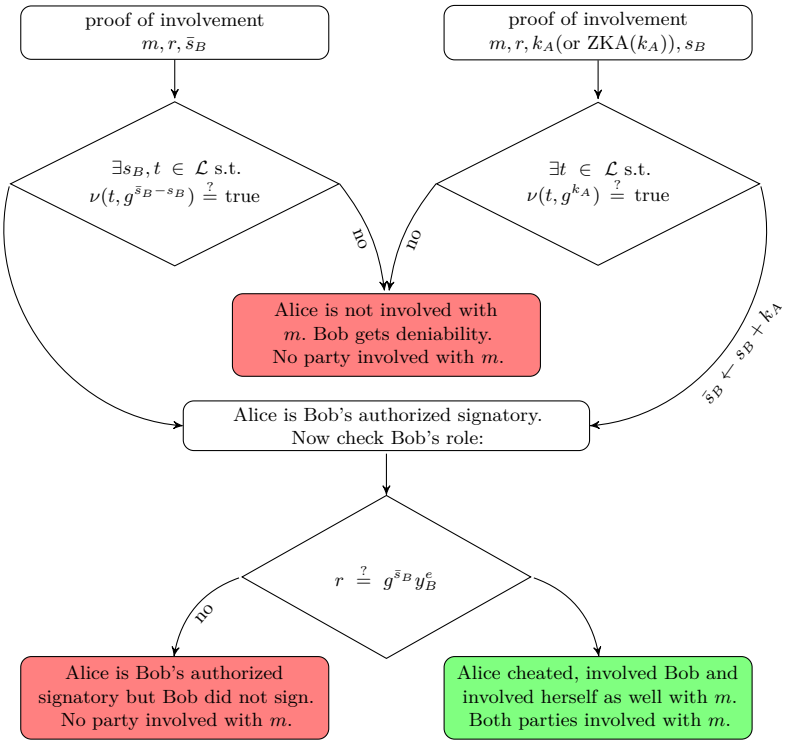

Fig. 5. The verification procedure: **proof of involvement**.

Acknowledgments

This work was supported in part by the French ANR Project ANR-12-INSE-0014 SIMPATIC.

References

1. Abdalla, M., An, J.H., Bellare, M., Namprempre, C.: From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In: Knudsen, L.R. (ed.) Advances in Cryptology – EUROCRYPT 2002. Lecture Notes in Computer Science, vol. 2332, pp. 418–433. Springer, Heidelberg, Germany, Amsterdam, The Netherlands (Apr 28 – May 2, 2002)
2. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In: Zheng, Y. (ed.) Advances in Cryptology – ASIACRYPT 2002. Lecture Notes in Computer Science, vol. 2501, pp. 415–432. Springer, Heidelberg, Germany, Queenstown, New Zealand (Dec 1–5, 2002)
3. Asokan, N., Schunter, M., Waidner, M.: Optimistic protocols for fair exchange. In: ACM CCS 97: 4th Conference on Computer and Communications Security. pp. 7–17. ACM Press, Zurich, Switzerland (Apr 1–4, 1997)

4. Baum-Waidner, B., Waidner, M.: Round-optimal and abuse free optimistic multi-party contract signing. In: Montanari, U., Rolim, J.D.P., Welzl, E. (eds.) Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1853, pp. 524–535. Springer (2000)
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing. pp. 1–10. ACM Press, Chicago, Illinois, USA (May 2–4, 1988)
6. Cachin, C., Camenisch, J.: Optimistic fair secure computation. In: Bellare, M. (ed.) Advances in Cryptology – CRYPTO 2000. Lecture Notes in Computer Science, vol. 1880, pp. 93–111. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2000)
7. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing. pp. 11–19. ACM Press, Chicago, Illinois, USA (May 2–4, 1988)
8. Chen, L., Kudla, C., Paterson, K.G.: Concurrent signatures. In: Cachin, C., Camenisch, J. (eds.) Advances in Cryptology – EUROCRYPT 2004. Lecture Notes in Computer Science, vol. 3027, pp. 287–305. Springer, Heidelberg, Germany, Interlaken, Switzerland (May 2–6, 2004)
9. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: Hartmanis, J. (ed.) Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA. pp. 364–369. ACM (1986)
10. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) Advances in Cryptology – CRYPTO'84. Lecture Notes in Computer Science, vol. 196, pp. 10–18. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 1984)
11. Feige, U., Fiat, A., Shamir, A.: Zero knowledge proofs of identity. In: Aho, A. (ed.) 19th Annual ACM Symposium on Theory of Computing. pp. 210–217. ACM Press, New York City„ New York, USA (May 25–27, 1987)
12. Garay, J.A., Jakobsson, M., MacKenzie, P.D.: Abuse-free optimistic contract signing. In: Wiener, M.J. (ed.) Advances in Cryptology – CRYPTO'99. Lecture Notes in Computer Science, vol. 1666, pp. 449–466. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 1999)
13. Garay, J.A., MacKenzie, P.D., Prabhakaran, M., Yang, K.: Resource fairness and composability of cryptographic protocols. In: Halevi, S., Rabin, T. (eds.) TCC 2006: 3rd Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 3876, pp. 404–428. Springer, Heidelberg, Germany, New York, NY, USA (Mar 4–7, 2006)
14. Girault, M., Poupard, G., Stern, J.: On the fly authentication and signature schemes based on groups of unknown order. J. Cryptology 19(4), 463–487 (2006)
15. Goldreich, O.: A simple protocol for signing contracts. In: Chaum, D. (ed.) Advances in Cryptology, Proceedings of CRYPTO '83, Santa Barbara, California, USA, August 21-24, 1983. pp. 133–136. Plenum Press, New York (1983)
16. Goldreich, O.: Foundations of Cryptography: Basic Applications, vol. 2. Cambridge University Press, Cambridge, UK (2004)
17. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th Annual ACM Symposium on Theory of Computing. pp. 218–229. ACM Press, New York City„ New York, USA (May 25–27, 1987)

18. Goldwasser, S., Levin, L.A.: Fair computation of general functions in presence of immoral majority. In: Menezes, A.J., Vanstone, S.A. (eds.) Advances in Cryptology – CRYPTO'90. Lecture Notes in Computer Science, vol. 537, pp. 77–93. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 11–15, 1991)
19. Gordon, S.D., Hazay, C., Katz, J., Lindell, Y.: Complete fairness in secure two-party computation. In: Ladner, R.E., Dwork, C. (eds.) 40th Annual ACM Symposium on Theory of Computing. pp. 413–422. ACM Press, Victoria, British Columbia, Canada (May 17–20, 2008)
20. Horster, P., Petersen, H., Michels, M.: Meta-El-Gamal signature schemes. In: ACM CCS 94: 2nd Conference on Computer and Communications Security. pp. 96–107. ACM Press, Fairfax, Virginia, USA (1994)
21. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated verifier proofs and their applications. In: Maurer, U.M. (ed.) Advances in Cryptology – EUROCRYPT'96. Lecture Notes in Computer Science, vol. 1070, pp. 143–154. Springer, Heidelberg, Germany, Saragossa, Spain (May 12–16, 1996)
22. Lindell, A.Y.: Legally-enforceable fairness in secure two-party computation. In: Malkin, T. (ed.) Topics in Cryptology – CT-RSA 2008. Lecture Notes in Computer Science, vol. 4964, pp. 121–137. Springer, Heidelberg, Germany, San Francisco, CA, USA (Apr 7–11, 2008)
23. Micali, S.: Simple and fast optimistic protocols for fair electronic exchange. In: Borowsky, E., Rajsbaum, S. (eds.) 22nd ACM Symposium Annual on Principles of Distributed Computing. pp. 12–19. Association for Computing Machinery, Boston, Massachusetts, USA (Jul 13–16, 2003)
24. Pinkas, B.: Fair secure two-party computation. In: Biham, E. (ed.) Advances in Cryptology – EUROCRYPT 2003. Lecture Notes in Computer Science, vol. 2656, pp. 87–105. Springer, Heidelberg, Germany, Warsaw, Poland (May 4–8, 2003)
25. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) Advances in Cryptology – EUROCRYPT'96. Lecture Notes in Computer Science, vol. 1070, pp. 387–398. Springer, Heidelberg, Germany, Saragossa, Spain (May 12–16, 1996)
26. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. J. Cryptology 13(3), 361–396 (2000)
27. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) Advances in Cryptology – ASIACRYPT 2001. Lecture Notes in Computer Science, vol. 2248, pp. 552–565. Springer, Heidelberg, Germany, Gold Coast, Australia (Dec 9–13, 2001)
28. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) Advances in Cryptology – CRYPTO'89. Lecture Notes in Computer Science, vol. 435, pp. 239–252. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 1990)
29. Seurin, Y.: On the exact security of Schnorr-type signatures in the random oracle model. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 554–571. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012)
30. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science. pp. 162–167. IEEE Computer Society Press, Toronto, Ontario, Canada (Oct 27–29, 1986)

A Schnorr Signatures

Schnorr digital signatures [28] are an offspring of ElGamal [10] signatures. This family of signatures is obtained by converting interactive identification protocols (zero-knowledge proofs) into transferable proofs of interaction (signatures). This conversion process, implicitly used by ElGamal, was discovered by Feige, Fiat and Shamir [11] and formalized by Abdalla, Bellare and Namprempre [1].

Throughout the paper, we will refer to the original Schnorr signature protocol as "classical" Schnorr. This protocol consists in four algorithms:

- $\mathsf{Setup}(\ell)$: On input a security parameter ℓ, this algorithm selects large primes p, q such that $q \geq 2^\ell$ and $p-1 \bmod q = 0$, as well as an element $g \in \mathbb{G}$ of order q in some multiplicative group $\mathbb{G}$ of order p, and a hash function $H : \{0,1\}^* \to \{0,1\}^\ell$. The output is a set of public parameters $\mathsf{pp} = (p, q, g, \mathbb{G}, H)$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters, this algorithm chooses uniformly at random $x \xleftarrow{\$} \mathbb{Z}_q^\times$ and computes $y \leftarrow g^x$. The output is the couple $(\mathsf{sk}, \mathsf{pk})$ where $\mathsf{sk} = x$ is kept private, and $\mathsf{pk} = y$ is made public.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$: On input public parameters, a secret key, and a message m this algorithm selects a random $k \xleftarrow{\$} \mathbb{Z}_q^\times$, computes

$$\begin{aligned} r &\leftarrow g^k \\ e &\leftarrow H(m \| r) \\ s &\leftarrow k - ex \bmod q \end{aligned}$$

  and outputs $\langle r, s \rangle$ as the signature of m.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m, \sigma)$: On input the public parameters, a public key, a message and a signature $\sigma = \langle r, s \rangle$, this algorithm computes $e \leftarrow H(m, r)$ and returns True if and only if $g^s y^e = r$; otherwise it returns False.

The security of classical Schnorr signatures was analysed by Pointcheval and Stern [25, 26] using the Forking Lemma. Pointcheval and Stern's main idea is as follows: in the Random Oracle Model, the opponent can obtain from the forger two valid forgeries $\{\ell, s, e\}$ and $\{\ell, s', e'\}$ for the same oracle query $\{m, r\}$ but with different message digests $e \neq e'$. Consequently, $r = g^s y^{-e} = g^{s'} y^{-e'}$ and from that it becomes straightforward to compute the discrete logarithm of $y = g^x$. Indeed, the previous equation can be rewritten as $y^{e-e'} = g^{s'-s}$, and therefore:

$$y = g^{\frac{s'-s}{e-e'}} \;\Rightarrow\; \mathrm{Dlog}_g(y) = \frac{s'-s}{e-e'}$$

The Forking Lemma for Schnorr signatures is originally stated as follows:

Theorem 2 (Forking Lemma, [26]). *Let $\mathcal{A}$ be an attacker which performs within a time bound t_F an existential forgery under an adaptively chosen-message attack against the Schnorr signature, with probability ϵ_F. Assume that $\mathcal{A}$ makes q_h hashing queries to a random oracle and q_s queries to a signing oracle.*

Then there exists an adversary solving the discrete logarithm problem in subgroups of prime order in polynomial expected time. Assume that $\epsilon_F \geq 10(q_s + 1)(q_s + q_h)/q$*, then the discrete logarithm problem in subgroups of prime order can be solved within expected time less that* $120686\, q_h t_F/\epsilon_F$*.*

This security reduction loses a factor $O(q_h)$ in the time-to-success ratio. Note that recent work by Seurin [29] shows that this is essentially the best possible reduction to the discrete logarithm problem.

B Proof of Theorem 1

Adversary Attacks Bob

Theorem 3. *Let* $\{y, g, p, q\}$ *be a DLP instance. If* $\mathcal{A}_{Alice}$ *plays the role of Alice and is able to forge in polynomial time a co-signature with probability* ϵ_F*, then in the Random Oracle model* $\mathcal{A}_{Alice}$ *can break that DLP instance with high probability in polynomial time.*

Proof. The proof consists in constructing a simulator $\mathcal{S}_{\text{Bob}}$ that interacts with the adversary and forces it to actually produce a classical Schnorr forgery. Here is how this simulator behaves at each step of the protocol.

1. *Key Establishment Phase:*
   $\mathcal{S}_{\text{Bob}}$ is given a target DLP instance $\{y, g, p, q\}$. As a simulator, $\mathcal{S}_{\text{Bob}}$ emulates not only Bob, but also all oracles and the directory $\mathcal{D}$ (see Figure 6).

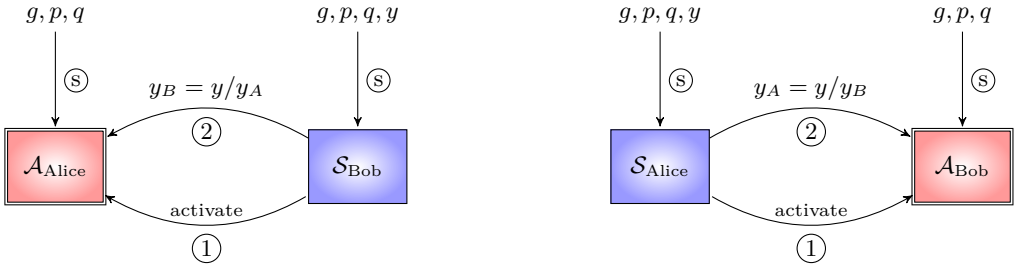

Fig. 6. The simulator $\mathcal{S}_{\text{Bob}}$ (left) or $\mathcal{S}_{\text{Alice}}$ (right) answers the attacker's queries to the public directory $\mathcal{D}$.

$\mathcal{S}_{\text{Bob}}$ injects the target y into the game, namely by posting in the directory the "public-key" $y_B \leftarrow y y_A^{-1}$.
To inject a target DLP instance $y \leftarrow g^x$ into $\mathcal{A}$, the simulator $\mathcal{S}_{\text{Bob}}$ reads $y_{\mathcal{A}}$ from the public directory and poses as an entity whose public-key is $y_{\mathcal{S}} \leftarrow y y_{\mathcal{A}}^{-1}$. It follows that $y_{\mathcal{A},\mathcal{S}}$, the common public-key of $\mathcal{A}$ and $\mathcal{S}$ will be precisely $y_{\mathcal{A},\mathcal{S}} \leftarrow y_{\mathcal{S}} y_{\mathcal{A}}$ which, by construction, is exactly y.

Then $\mathcal{S}_{\text{Bob}}$ activates $\mathcal{A}_{\text{Alice}}$, who queries the directory and gets y_B. At this point in time, $\mathcal{A}_{\text{Alice}}$ is tricked into believing that she has successfully established a common co-signature public-key set $\{g, p, q, y\}$ with the "co-signer" $\mathcal{S}_{\text{Bob}}$.

2. *Query Phase:*
   $\mathcal{A}_{\text{Alice}}$ will now start to present queries to $\mathcal{S}_{\text{Bob}}$. In a "normal" attack, $\mathcal{A}_{\text{Alice}}$ and Bob would communicate with a random oracle $\mathcal{O}$ representing the hash function H. However, here, the simulator $\mathcal{S}_{\text{Bob}}$ will play $\mathcal{O}$'s role and answer $\mathcal{A}_{\text{Alice}}$'s hashing queries.

   $\mathcal{S}_{\text{Bob}}$ must respond to three types of queries: *hashing queries*, *co-signature queries* and *transcript queries*. $\mathcal{S}_{\text{Bob}}$ will maintain an oracle table T containing all the hashing queries performed throughout the attack. At start $T \leftarrow \varnothing$. When $\mathcal{A}_{\text{Alice}}$ submits a hashing query q_i to $\mathcal{S}_{\text{Bob}}$, $\mathcal{S}_{\text{Bob}}$ answers as shown in Algorithm 1.

Algorithm 1 Hashing oracle simulation.

Input: A hashing query q_i from $\mathcal{A}$
  if $\exists e_i, \{q_i, e_i\} \in T$ **then**
    $\rho \leftarrow e_i$
  else
    $\rho \xleftarrow{\$} \mathbb{Z}_q^{\times}$
    Append $\{q_i, \rho\}$ to T
  end if
  return ρ

When $\mathcal{A}_{\text{Alice}}$ submits a co-signature query to $\mathcal{S}_{\text{Bob}}$, $\mathcal{S}_{\text{Bob}}$ proceeds as explained in Algorithm 2.

Algorithm 2 Co-signing oracle simulation.

Input: A co-signature query m from $\mathcal{A}_{\text{Alice}}$
  $s_B, e \xleftarrow{\$} \mathbb{Z}_q^*$
  $r_B \leftarrow g^{s_B} y^e$
  Send $H(0 \| r_B)$ to $\mathcal{A}_{\text{Alice}}$
  Receive r_A from $\mathcal{A}_{\text{Alice}}$
  $r \leftarrow r_A r_B$
  $u \leftarrow 1 \| m \| r$
  if $\exists e' \neq e, \{u, e'\} \in T$ **then**
    abort
  else
    Append $\{u, e\}$ to T
  end if
  return s_B

Finally, when $\mathcal{A}_{\text{Alice}}$ requests a conversation transcript, $\mathcal{S}_{\text{Bob}}$ replies by sending $\{m, \rho, r_A, r_B, s_B, s_A\}$ from a previously successful interaction.

3. *Output Phase:*
After performing queries, $\mathcal{A}_{\text{Alice}}$ eventually outputs a co-signature m, r, s valid for $y_{\mathcal{A},\mathcal{S}}$ where $r = r_A r_B$ and $s = s_A + s_B$. By design, these parameters are those of a classical Schnorr signature and therefore $\mathcal{A}_{\text{Alice}}$ has produced a classical Schnorr forgery.

To understand $\mathcal{S}_{\text{Bob}}$'s co-signature reply (Algorithm 2), assume that $\mathcal{A}_{\text{Alice}}$ is an honest Alice who plays by the protocol's rules. For such an Alice, $\{s, r\}$ is a valid signature with respect to the common co-signature public-key set $\{g, p, q, y\}$. There is a case in which $\mathcal{S}_{\text{Bob}}$ aborts the protocol before completion: this happens when it turns out that $r_A r_B$ has been previously queried by $\mathcal{A}_{\text{Alice}}$. In that case, it is no longer possible for $\mathcal{S}_{\text{Bob}}$ to reprogram the oracle, which is why $\mathcal{S}_{\text{Bob}}$ must abort. Since $\mathcal{A}_{\text{Alice}}$ does not know the random value of r_B, such a bad event would only occur with a negligible probability exactly equal to q_h/q (where q_h is the number of queries to the hashing oracle).

Therefore, $\mathcal{A}_{\text{Alice}}$ is turned into a forger for the target Schnorr instance with probability $1 - q_h/q$. Since $\mathcal{A}_{\text{Alice}}$ succeeds with probability ϵ_F, $\mathcal{A}_{\text{Alice}}$'s existence implies the existence of a Schnorr signature forger of probability $\epsilon_S = (1 - q_h/q)\epsilon_F$, which by the Forking Lemma shows that there exists a polynomial adversary breaking the chosen DLP instance with high probability. □

Being an attacker, at some point $\mathcal{A}_{\text{Alice}}$ will output a forgery $\{m', r', s'\}$. From here on we use the Forking Lemma and transform $\mathcal{A}_{\text{Alice}}$ into a DLP solver as described by Pointcheval and Stern in [26, Theorem 14].

Adversary Attacks Alice The case where $\mathcal{A}$ targets A is similar but somewhat simpler, and the proof follows the same strategy.

Theorem 4. *Let $\{y, g, p, q\}$ be a DLP instance. If $\mathcal{A}_{Bob}$ plays the role of Bob and is able to forge a co-signature with probability ϵ_F, then in the Random Oracle model $\mathcal{A}_{Bob}$ can break that DLP instance with high probability in polynomial time.*

Proof (Theorem 4). Here also the proof consists in constructing a simulator, $\mathcal{S}_{\text{Alice}}$, that interacts with the adversary and forces it to actually produce a classical Schnorr forgery. The simulator's behaviour at different stages of the security game is as follows:

1. *The Key Establishment Phase:*
$\mathcal{S}_{\text{Alice}}$ is given a target DLP instance $\{y, g, p, q\}$. Again, $\mathcal{S}_{\text{Alice}}$ impersonates not only Alice, but also $\mathcal{O}$ and $\mathcal{D}$. $\mathcal{S}_{\text{Alice}}$ injects the target y into the game as described in Appendix B. Now $\mathcal{S}_{\text{Alice}}$ activates $\mathcal{A}_{\text{Bob}}$, who queries $\mathcal{D}$ (actually controlled by $\mathcal{S}_{\text{Alice}}$) to get y_B. $\mathcal{A}_{\text{Bob}}$ is thus tricked into believing that it has successfully established a common co-signature public-key set $\{g, p, q, y\}$ with the "co-signer" $\mathcal{S}_{\text{Alice}}$.

2. *The Query Phase:*
   $\mathcal{A}_{\text{Bob}}$ will now start to present queries to $\mathcal{S}_{\text{Alice}}$. Here as well, $\mathcal{S}_{\text{Alice}}$ will play $\mathcal{O}$'s role and will answer $\mathcal{A}_{\text{Bob}}$'s hashing queries.

   Again, $\mathcal{S}_{\text{Alice}}$ must respond to hashing queries and co-signature queries. Hashing queries are answered as shown in Algorithm 1. When $\mathcal{A}_{\text{Bob}}$ submits a co-signature query to $\mathcal{S}_{\text{Alice}}$, $\mathcal{S}_{\text{Alice}}$ proceeds as explained in Algorithm 3.

Algorithm 3 Co-signing oracle simulation for $\mathcal{S}_{\text{Alice}}$.

Input: A co-signature query m from $\mathcal{A}_{\text{Bob}}$
Receive ρ from $\mathcal{A}_{\text{Bob}}$
Query T to retrieve r_B such that $H(0\|r_B) = \rho$
$e, s_A \xleftarrow{\$} \mathbb{Z}_q$
$r \leftarrow r_B g^{s_A} y^e$
$u \leftarrow 1\|m\|r$
if $\exists e' \neq e, \{u, e'\} \in T$ **then**
  abort
else
  Append $\{u, e\}$ to T
end if
$r_A \leftarrow r r_B^{-1}$
Send r_A to $\mathcal{A}_{\text{Bob}}$
Receive r_B from $\mathcal{A}_{\text{Bob}}$; this r_B is not used by $\mathcal{S}_{\text{Alice}}$
Receive s_B from $\mathcal{A}_{\text{Bob}}$
return s_A

   $\mathcal{S}_{\text{Alice}}$ controls the oracle $\mathcal{O}$, and as such knows what is the value of r_B that $\mathcal{A}_{\text{Bob}}$ is committed to. The simulator is designed to trick $\mathcal{A}_{\text{Bob}}$ into believing that this is a real interaction with Alice, but Alice's private key is not used.

3. *Output*:
   Eventually, $\mathcal{A}_{\text{Bob}}$ produces a forgery that is a classical Schnorr forgery $\{m, r, s\}$.

Algorithm 3 may fail with probability $1/q$. Using the Forking Lemma again, we transform $\mathcal{A}_{\text{Bob}}$ into an efficient solver of the chosen DLP instance. □