Fair Exchange of Short Signatures without Trusted Third Party

Philippe Camacho
Dept. of Computer Science, University of Chile,
Blanco Encalada 2120, 4to piso, Santiago, Chile.
pcamacho@dcc.uchile.cl

November 18, 2012

Abstract

We propose a protocol to exchange Boneh-Boyen short signatures in a fair way and without relying on a trusted third party. Our protocol is quite practical and is the first of the sort to the best of our knowledge. Our construction uses a new non-interactive zero-knowledge (NIZK) argument to prove that a commitment is the encryption of a bit vector. We also design a NIZK argument to prove that a commitment to a bit vector $v = (b_1, b_2, ..., b_{\kappa})$ is such that $\sum_{i \in [\kappa]} b_i 2^{i-1} = \theta$ where θ is the discrete logarithm of some public value $D = g^{\theta}$. These arguments may be of independent interest.

Key words: Fair exchange, short signatures, gradual release of a secret.

1 Introduction

Nowadays it is more and more common to trade digital goods on the web: E-books, software licenses, avatar-games currencies like *Ultima Online*¹ to cite a few. Whether these goods are exchanged on *E-bay* through *Paypal* or bought directly to their provider *Amazon* or *Microsoft*, the transaction to be secure requires a trusted third party (TTP). Though it works quite well in practice, enabling totally distributed and at the same time secure transaction systems is of clear interest: It would avoid some security issues due to the presence of single points of failure, and also allow smoother electronic commercial transactions that would not rely on some intermediary. A lot of these transactions may be captured by the exchange of digital signatures. Suppose for example you want to buy a software license to some independent developer: Indeed exchanging the software license as well as the money transfer (digital check) can be modeled by signed messages. However we face a non-trivial problem. Given that the transaction is made on-line, a malicious participant may fool his counterpart by not sending his signature or sending some garbage information. A protocol that prevents such a behavior from a corrupted party is called *fair*: This means that at the end of the execution of protocol either both parties obtain the signature they expected or none does.

¹http://en.wikipedia.org/wiki/Ultima_Online

There are two main approaches to solve this problem. On the one hand, one can assume that both players interact through a TTP. Though this solution does not fit our goal, it is important to note that an important line of research has focused on designing protocols where the TTP is only required when "something goes wrong". These protocols are said to be *optimistically fair*: See [1, 25] and [22] for some recent work.

On the other hand, if no TTP exists and we assume that both participants have exactly the same computational resources, then it is impossible in general to achieve complete fairness [10]. In [2, 14] was proposed a way to relax the notion of fairness in order to overcome Cleve's impossibility result. The idea is to assume that both players have roughly the same amount of time, so we can achieve partial fairness. Several secure multi-party computations and specific protocols, like [6, 11, 12, 5, 16], were built on top of this security notion. The recurrent idea behind these constructions consists in enabling each player to release their secret bit by bit in alternation. Thus, if a player aborts, the other participant will have "only one bit of disadvantage". Formalizing this idea is not an easy task though, in particular because it is hard to reason on the specific amount of time for the players. This issue was noticed in [19] where authors point out that (1) assigning more time to the honest party in order to allow him recover his value is somehow artificial as it does not depend on the participant himself, and (2) implementing such definitions seems to imply the use of strong assumptions related to the exact time required to solve some computational problem.

In this work we propose a new security definition that still captures the intuition of partial fairness for the exchange of digital signatures, but without forcing the participants to have access to almost equal computational resources as proposed in [16]. The idea of our definition is to compare the probabilities of computing valid signatures on the agreed messages at the end of the protocol. More precisely, if the adversary aborts the protocol, the honest participant² will compute the expected signature by choosing randomly a value from the space of signature candidates, which is defined by the remaining bits to be obtained. The adversary will keep running its own algorithm and also output a signature candidate. We say the protocol is secure if the probabilities that each participant output a valid signature only differ by a polynomial factor. Note that this definition, like previous ones that circumvent Cleve's impossibility result [10], allows the adversary to get some advantage, but it guarantees that this advantage is polynomially bounded. With that definition in hand we can prove the security of our protocol without having to rely on the strong assumptions mentioned above. Our protocol is designed to exchange short signatures [4] without the presence of a TTP. We use bilinear maps as the underlying signature scheme, and also the idea of releasing gradually each bit of some secret θ that will enable to recover the signature. The security of our construction relies on complexity assumptions for bilinear maps, namely the κ -Strong Diffie-Hellman [4], and the κ -Bilinear Diffie-Hellman assumptions [3] and holds in the common reference string model. As we use non-interactive zero-knowledge proofs of knowledge (ZKPoK) in order to make the protocol simpler and more efficient, we require the use of random oracle [15] or some non-black box assumptions [20]. If we like, we can use interactive ZKPoK at a minor expense of round efficiency.

OUR CONTRIBUTIONS.

1. We propose a practical protocol for exchanging short signatures [4] without relying on a TTP. To the best of our knowledge this is the first construction that meets such a goal.

²Note that we need to consider that at least one participant is honest, as otherwise we cannot really avoid that one of the two adversaries, which are arbitrary polynomial time algorithms, wins.

Operation	# Exp	# Mult	# BM	# Div	# Hash
Step 2 EncSigGen	1				
Bit Vector commitment	κ	κ			
BV-NIZK	4κ	2κ			
BE-NIZK	2κ	$2\kappa - 3$		1	
ZKPoK	$2\kappa + 1$	κ			$\kappa + 1$
Check BV-NIZK		κ	4κ		
Check BE-NIZK	1	$\kappa - 1$	4	2	
Check ZKPoK	$3\kappa + 2$	$2\kappa + 1$			$\kappa + 1$
Step 7 EncSigCheck	2	2	2		
KeyBitCheck	κ	κ			
EncSigDecrypt	1				
Sum	$13\kappa + 8$	$11\kappa - 1$	$4\kappa + 6$	3	$2\kappa + 2$

Figure 1: **Time complexities of the fair exchange protocol.** This figure shows the number of cryptographic operations performed by each participant during the whole protocol. The first block corresponds to the algorithm EncSigGen, the second block to the algorithm EncSigCheck. BV-NIZK stands for the NIZK argument to prove a commitment is the encryption of a bit vector as depicted in Fig. 2. BE-NIZK stands for the NIZK argument to prove the equivalence between a commitment to a bit vector and the discrete logarithm of $D = g^{\theta}$, as depicted in Fig. 3. #Exp, #Mult, #BM, and #Div correspond respectively to the number of group exponentiations, group multiplications, bilinear map applications and group inversions. #Hash is the number of hash evaluations.

The number of rounds of our protocol is $\kappa + 1$, where κ is the security parameter. The communication complexity is $16\kappa^2 + 12\kappa$ bits. The protocol requires a linear number of group exponentiations, group multiplications, bilinear map applications, hash computations and also a constant number of group divisions (see Fig. 1 for more details).

- 2. We introduce a new non-interactive zero-knowledge (NIZK) argument to prove that a commitment is the encryption of a bit vector. This protocol may be of independent interest.
- 3. We introduce another NIZK argument to prove that a commitment to a bit vector corresponds to the binary decomposition of some value θ which is hidden as the discrete logarithm of some group element. We think this argument may lead to other interesting applications.
- 4. As stated earlier, we propose a new security definition for partial fairness in the context of the exchange of digital signatures. This definition is simple and avoids the issue of involving the *exact* running time of the participants.

OUR APPROACH. Let $\kappa \in \mathbb{N}$ be the security parameter. Let $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BMGen}(1^{\kappa})$ be the public parameter where $p = |\mathbb{G}| = |\mathbb{G}_T|$ is prime, \mathbb{G}, \mathbb{G}_T are cyclic groups, $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is the bilinear map and g is a random generator. Let s be a random element in \mathbb{Z}_p , we consider the following common reference string: $(g, g^s, g^{s^2}, ..., g^{s^{\kappa}}) = (g_0, g_1, g_2, ..., g_{\kappa})$. In practice this common reference string can be computed using generic multi-party computation techniques (see [9] for an

efficient implementation) so that the secret s is randomly generated and remains unknown to all the participants. Another alternative is to rely on a TTP that would "securely delete" the secret after the generation of the common reference string. Obviously the intention of this work is to avoid the use of a TTP, but note however that even in this case, the TTP would be required only once

Our construction can be summarized as follows. The prover chooses a secret $\theta \in \mathbb{Z}_p$, then commits each bit of this secret into a Pedersen [28] commitment, where the bit b_i in position i with randomness $r_i \in \mathbb{Z}_p$ will be committed with respect to the base (g,g_i) : That is $\mathsf{Commit}(b_i,r_i,i) = g^{r_i}g_i^{b_i}$. Then we use a NIZK argument 3 to prove this commitment really encrypts a bit. The next step is to publish $D = g^{\theta}$ and show, using another NIZK argument, that θ , the discrete logarithm of D, is "equivalent" to the bit vector committed in $\vec{C} = (\mathsf{Commit}(b_i,r_i,i))_{i\in[\kappa]}$. More precisely, the argument proves that $\theta = \sum_{i\in[\kappa]} b_i 2^{i-1}$. Now if we consider some signature σ , the prover will blind it using θ to obtain $\tilde{\sigma} = \sigma^{\theta}$. Using bilinear maps it is straightforward to verify that $\tilde{\sigma}$ contains a valid signature σ which is blinded in the exponent by θ , the discrete logarithm of D. The other verifications will consist simply in checking the NIZK arguments. Finally, we need to provide zero-knowledge proofs of knowledge for the representation of each bit commitment in order to be able to simulate the execution of the protocol even if the adversary aborts. By releasing each bit in turn, both players will reconstruct their own blinding factor θ and obtain the signature.

RELATED WORK. Among the abundant literature on the topic of gradual release and fair exchange for digital signatures, [12] is probably the work that is the most similar to ours: It describes a practical fair exchange protocol for digital signatures based on gradual release of a secret. The protocol described in [12] works for Rabin, RSA and El Gamal signatures. The number of rounds of the protocol described in [12] is roughly 2κ for RSA and Rabin signatures and κ for El Gamal signatures.

Due to Cleve's impossibility result [10], the question of building complete fair protocols with dishonest majority seemed to be closed. However, Gordon et al. showed that non-trivial functions can be computed fairly in the two-party model [18], and left the question of finding a tight characterization of these functions open. In particular it is not known whether functions with a non-polynomial size domain and that return multiple bits as output (like computing a signature) can be computed fairly in Cleve's setting.

In [19] is proposed a definition for partial fairness that may exhibit some similarities with ours (both definitions involve a $Q(\kappa)$ factor where Q is a polynomial). However our definition and approach differs quite from [19]. First, the setting in [19] is more general than our specific construction to exchange digital signatures. Secondly, in their protocol, the number of rounds is variable and defines the level of fairness, whereas in our construction fairness only depends on the computational power of the participants.

Our NIZK argument to prove that a commitment encrypts a bit vector is inspired by [21, 20]. We remark that, though [16] uses the idea of gradual release, the construction proposed is not practical as it requires to code the functionality (signing in our case) as an arithmetic circuit.

ORGANIZATION OF THE PAPER. In Section 2 we introduce notations and recall some definitions and standard techniques we use in this work. In Section 3 we describe the bit vector commitment scheme. The argument for proving the equivalence between a bit vector commitment $(C_i)_{i \in [\kappa]}$ and the discrete logarithm θ of g^{θ} is introduced in Section 4. The fair exchange protocol is shown in

³We recall in Section 2.2 standard definitions for zero-knowledge protocols.

2 Preliminaries

2.1 Notations

For $m, n \in \mathbb{N}$ with m < n, [m..n] means the set of integers $\{m, m+1, ..., n-1, n\}$ and [n] means the set of integers $\{1, ..., n\}$. If $\kappa \in \mathbb{N}$ is the security parameter then 1^{κ} denotes the unary string with κ ones. We will use p to denote a prime number of κ bits. A function $\nu : \mathbb{N} \to [0, 1]$ is said to be negligible in κ if for every polynomial $q(\cdot)$ there exists κ_0 such that $\forall \kappa > \kappa_0 : \nu(\kappa) < 1/q(\kappa)$. In the following, neg will denote *some* negligible function in κ . An algorithm is called PPT if it is probabilistic and runs in polynomial time in κ . We write $x \overset{R}{\leftarrow} X$ to denote an element x chosen uniformly at random from a set X. $x \leftarrow v$ means that the variable x is assigned the value v.

A vector of n components and values v_i is denoted $\vec{v} = (v_i)_{i \in [n]}$. If the vector contains elements of \mathbb{Z}_p we may also write $B[\cdot] = (B[1], B[2], ..., B[n])$. Let $\theta \in \mathbb{Z}_p$, we denote by $\theta[\cdot]$ the binary decomposition (vector) of θ . That is $\theta[\cdot] = (\theta[1], ..., \theta[\kappa])$ and in particular $\theta = \sum_{i \in [\kappa]} \theta[i] 2^{i-1}$. $P(\cdot)$ will stand for a formal polynomial with coefficients in \mathbb{Z}_p , and $P[\cdot]$ for the vector of its coefficients: Thus if d = deg(P) is the degree of polynomial $P(\cdot)$ then we have: $P(X) = \sum_{i \in [d+1]} P[i] X^{i-1}$.

2.2 Zero-Knowledge Protocols

2.2.1 Trapdoor commitments

Let \mathcal{R} be the space of randomness, \mathcal{C} the set where commitments belong and \mathcal{M} the space for messages. A trapdoor commitment scheme is composed by the following algorithms: \mathcal{K} , Commit, Verify, TCommit, TOpen. $\mathcal{K}(1^{\kappa})$ is a randomized algorithm that generates the common reference string CRS and an associated trapdoor τ . Commit(CRS, m, r) is a deterministic algorithm that computes a commitment C to value $m \in \mathcal{M}$ using $r \in \mathcal{R}$. Verify(CRS, C, m, r) returns 1 if and only if C = Commit(CRS, m, r), otherwise returns 0. We will sometime use the notation open to denote the opening of the commitment C, that is open = (m, r). TCommit(τ) is a randomized algorithm that returns an equivocal commitment C along with an equivocation key ek given the trapdoor τ . TOpen(ek, C, m) is a deterministic algorithm that returns the randomness $r \in \mathcal{R}$ of C with respect to message $m \in \mathcal{M}$. In order to simplify the notation, in the following the common reference string CRS will be an implicit argument of algorithms Commit and Verify.

We say the commitment scheme is *computationally binding* if for all non-uniform stateful PPT adversary \mathcal{A} we have

$$\Pr\left[\begin{array}{c} (\mathsf{CRS},\tau) \leftarrow \mathcal{K}(1^\kappa); (m_0,m_1,r_0,r_1) \leftarrow \mathcal{A}(\mathsf{CRS}): \\ m_0 \neq m_1 \wedge \mathsf{Commit}(m_0,r_0) = \mathsf{Commit}(m_1,r_1) \end{array}\right] = \mathsf{neg}(\kappa)$$

The scheme is said to be perfectly hiding if for all non-uniform stateful adversaries A we have

$$\Pr\left[(\mathsf{CRS}, \tau) \leftarrow \mathcal{K}(1^{\kappa}); (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{CRS}); C \leftarrow \mathsf{Commit}(m_0, r_0) : \mathcal{A}(C) = 1 \right] \\ = \Pr\left[(\mathsf{CRS}, \tau) \leftarrow \mathcal{K}(1^{\kappa}); (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{CRS}); C \leftarrow \mathsf{Commit}(m_1, r_1) : \mathcal{A}(C) = 1 \right]$$

A commitment scheme is *perfectly trapdoor* for any stateful PPT adversary we have:

$$\Pr\left[\begin{array}{c} (\mathsf{CRS},\tau) \leftarrow \mathcal{K}(1^\kappa); \\ m \leftarrow \mathcal{A}(\mathsf{CRS}); \\ r \overset{\mathcal{R}}{\leftarrow} \mathcal{R}; \\ C \leftarrow \mathsf{Commit}(m,r): \\ \mathcal{A}(m,r) = 1 \end{array}\right] = \Pr\left[\begin{array}{c} (\mathsf{CRS},\tau) \leftarrow \mathcal{K}(1^\kappa); \\ m \leftarrow \mathcal{A}(\mathsf{CRS}); \\ (C,ek) \leftarrow \mathsf{TCommit}(\tau); \\ r \leftarrow \mathsf{TOpen}(ek,C,m): \\ \mathcal{A}(m,r) = 1 \end{array}\right]$$

As a commitment is perfectly indistinguishable from an equivocal commitment we have that a perfect trapdoor commitment scheme is also perfectly hiding.

Our construction relies on a slight variation of the Pedersen⁴ commitment scheme [28] which we recall here. Let $\mathbb G$ be a cyclic group of prime order $p \in \mathbb N$. We consider the common reference string composed by $g \in \mathbb G$ and $h \in \mathbb G$ where g,h are chosen randomly and the discrete logarithm s of h in base g remains secret. To commit to a message $m \in \mathbb Z_p$ with randomness $r \in \mathbb Z_p$ we compute⁵ Commit $(m,r)=g^rh^m$. We denote by open =(m,r) the opening of the commitment. As shown in [28], this scheme is perfectly hiding (in fact it is a trapdoor commitment where $\tau=s$) and computationally binding, under the assumption that computing the discrete logarithm in $\mathbb G$ is hard.

2.2.2 Non-Interactive Zero-Knowledge Arguments

We are interested in statements that are efficiently verifiable⁶. Let R be a NP relation such that $(C, w) \in R$ means the statement is true and this can be verified with the witness w. We will consider R_N , the subset of R where the statements are of size $N = \kappa^{O(1)}$. For relation R we define a non-interactive argument in the following way. An algorithm KeyGen(1^{κ}, N) generates the common reference string CRS. Then the prover P given as input (CRS, C, w), checks first that $(C, w) \in R_N$. If this is not the case he outputs \bot . Otherwise he outputs an argument π . The verifier V using CRS, C and π returns 1 in case it accepts the argument and 0 otherwise.

In our case, C will be a commitment and w its opening (the message and the randomness). We will consider non-interactive zero-knowledge (NIZK) argument (proof) systems (KeyGen, \mathcal{P}, \mathcal{V}) for the relation R_N with the following properties.

PERFECT COMPLETENESS. The argument is perfectly complete if a honest prover can convince a honest verifier with probability 1 in case the statement is true. For any PPT adversary \mathcal{A} we have

$$\Pr\left[\begin{array}{c} \mathsf{CRS} \leftarrow \mathsf{KeyGen}(1^\kappa, N); (C, w) \leftarrow \mathcal{A}(\mathsf{CRS}); \pi \leftarrow \mathcal{P}(\mathsf{CRS}, C, w) : \\ \mathcal{V}(\mathsf{CRS}, C, \pi) = 1 \land (C, w) \in R_N \end{array}\right] = 1$$

COMPUTATIONAL SOUNDNESS. The argument is said to be sound if no adversary can convince a verifier of a false statement. For any PPT adversary \mathcal{A} we have

$$\Pr\left[\begin{array}{c}\mathsf{CRS} \leftarrow \mathsf{KeyGen}(1^\kappa, N); (C, \pi) \leftarrow \mathcal{A}(\mathsf{CRS}): \\ \mathcal{V}(\mathsf{CRS}, C, \pi) = 1 \land \nexists w: (C, w) \in R_N\end{array}\right] = \mathsf{neg}(\kappa)$$

⁴Note that this kind commitment was introduced earlier in [8](see page 98).

⁵Note that we change a bit the convention as the message is "stored" as the exponent of h, instead of g.

⁶We follow the notations of [20].

PERFECT WITNESS INDISTINGUISHABILITY. The argument is said to be perfectly witness-indistinguishable if the verifier does not learn which witness was used by the prover in order to produce the proof. For all stateful interactive PPT adversaries \mathcal{A} we have

$$\Pr\left[\begin{array}{c} \mathsf{CRS} \leftarrow \mathsf{KeyGen}(1^\kappa, N); (C, w_1, w_2) \leftarrow \mathcal{A}(\mathsf{CRS}); \\ \pi \leftarrow \mathcal{P}(\mathsf{CRS}, C, w_1) : \\ ((C, w_1), (C, w_2)) \in R_N^2 \land \mathcal{A}(\pi) = 1 \end{array}\right]$$

$$= \operatorname{Pr} \left[\begin{array}{c} \mathsf{CRS} \leftarrow \mathsf{KeyGen}(1^\kappa, N); (C, w_1, w_2) \leftarrow \mathcal{A}(\mathsf{CRS}); \\ \pi \leftarrow \mathcal{P}(\mathsf{CRS}, C, w_2) : \\ ((C, w_1), (C, w_2)) \in R_N^2 \wedge \mathcal{A}(\pi) = 1 \end{array} \right]$$

Note that in case there is only one valid witness w for some statement C, then the argument becomes trivially perfectly witness-indistinguishable.

PERFECT ZERO-KNOWLEDGE. We say an argument is zero-knowledge if the verifier learns nothing but the truth of the statement. To formalize this idea we consider two simulators S_1 , S_2 such that S_1 generates the CRS and a trapdoor τ . The simulator S_2 uses the common reference string CRS, the statement C and the trapdoor τ to output a simulated argument π . The argument is said to be perfect zero-knowledge if for any stateful interactive PPT adversary A we have

$$\Pr\left[\begin{array}{c} \mathsf{CRS} \leftarrow \mathsf{KeyGen}(1^\kappa, N); \\ (C, w) \leftarrow \mathcal{A}(\mathsf{CRS}); \\ \pi \leftarrow \mathcal{P}(\mathsf{CRS}, C, w) : \\ (C, w) \in R_N \land \mathcal{A}(\pi) = 1 \end{array} \right] = \Pr\left[\begin{array}{c} (\mathsf{CRS}, \tau) \leftarrow S_1(1^\kappa, N); \\ (C, w) \leftarrow \mathcal{A}(\mathsf{CRS}); \\ \pi \leftarrow S_2(\mathsf{CRS}, C, \tau) : \\ (C, w) \in R_N \land \mathcal{A}(\pi) = 1 \end{array} \right]$$

2.2.3 Non-interactive Zero-Knowledge Proofs of Knowledge

Our protocol for fair exchange uses zero-knowledge proofs of knowledge relative to bit commitments. In order to simplify the description of the fair exchange protocol we will use non-interactive zero-knowledge proofs of knowledge. We note however that interactive ZKPoK would work as well, though adding 2 rounds to our protocol and loosing possibly security guarantees in case the protocol is run in parallel or involves more than 2 players. The most popular way to implement such protocols is by using the Fiat-Shamir heuristic [15], trading non-interaction for a security proof relying on the random oracle model. We mention that our scheme could also be adapted to fit Groth's short non-interactive argument proof system [20]. In this case the security of non-interactive proofs of knowledge would depend on a non-black box assumption and we would get shorter arguments⁷.

Let \mathbb{G} be a cyclic group of prime order p where the discrete logarithm is hard. Let $\mathbb{H}: \mathbb{G} \to \mathbb{Z}_p$ be a randomly chosen function from a CRHF. Let g, h be two random generators of \mathbb{G} such that the discrete logarithm of h in base g is unknown.

We will need a ZKPoK of the discrete logarithm θ of some public value $D = g^{\theta}$. Following the notation of [7], we have that $PK\{\theta:g^{\theta}\} = (c = \mathsf{H}(g^r), z = r - c\theta)$ where $r \stackrel{R}{\leftarrow} \mathbb{Z}_p$. The verifier checks that $c = \mathsf{H}(D^c g^z)$. We will also use the following ZKPoK that convinces a verifier that the prover knows the representation of a commitment $C = g^{\alpha}h^{\beta}$ in base (g,h) where $\alpha, \beta \in \mathbb{Z}_p$. $PK\{(\alpha,\beta): C = g^{\alpha}h^{\beta}\} = (c = \mathsf{H}(g^{r_1}h^{r_2}), z_1 = r_1 - c\alpha, z_2 = r_2 - c\beta)$ where $r_1, r_2 \stackrel{R}{\leftarrow} \mathbb{Z}_p$. The verifier checks that $c = \mathsf{H}(C^c g^{z_1}h^{z_2})$.

⁷Note however that the common reference string would need to be of quadratic size in the size of the statements.

2.3 Bilinear Maps

In this paper we consider bilinear maps which are defined as following: Let \mathbb{G} , \mathbb{G}_T , be cyclic groups of prime order p. We consider a map $e: \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ which is

- bilinear: $\forall a, b \in \mathbb{G}, x, y \in \mathbb{Z}_p : e(a^x, b^y) = e(a, b)^{xy}$.
- non-degenerate: let g be a generator of \mathbb{G} then e(g,g) also generates \mathbb{G}_T .
- efficiently computable: There exists a polynomial time algorithm BMGen with parameter 1^{κ} that outputs $(p, \hat{\mathbb{G}}_T, \hat{e}, g)$ where $\hat{\mathbb{G}}, \hat{\mathbb{G}}_T$ is the representation of the corresponding groups of size p (p being a prime number of κ bits), g is a generator of \mathbb{G} , and \hat{e} is an efficient algorithm to compute the map. For the sake of simplicity, we will not distinguish between $\mathbb{G}, \mathbb{G}_T, e$, and $\hat{\mathbb{G}}, \hat{\mathbb{G}}_T, \hat{e}$.

2.4 Assumptions

Let $N \in \mathbb{N}$. For the following assumptions, the common public parameter is $PP = \langle (p, \mathbb{G}, \mathbb{G}_T, e, g), (g_0, g_1, g_2, \dots, g_N) \rangle$ where s is chosen randomly in \mathbb{Z}_p and $g_i = g^{s^i}$ for $i \in [0..N]$.

Definition 1 N-Diffie-Hellman Inversion (N-DHI) assumption, [26]. The N-Diffie-Hellman Inversion problem consists in computing $g^{\frac{1}{s}}$ given PP. We say the N-DHI assumption holds if for any PPT adversary A we have

$$Adv^{N\text{-}DHI}(\mathcal{A},\kappa,N) = \Pr\left[\,g^{\frac{1}{s}} \leftarrow \mathcal{A}(1^{\kappa},\mathtt{PP})\,\right] = \mathsf{neg}(\kappa)$$

The bilinear variant of the previous assumption was introduced in [3].

Definition 2 N-Bilinear Diffie-Hellman Inversion assumption

(N-BDHI). The N-Bilinear Diffie-Hellman Inversion problem consists in computing $e(g,g)^{\frac{1}{s}}$ given PP. We say the N-BDHI assumption holds if for any PPT adversary A we have

$$Adv^{N\text{-}BDHI}(\mathcal{A},\kappa,N) = \Pr\left[\,e(g,g)^{\frac{1}{s}} \leftarrow \mathcal{A}(1^{\kappa},\operatorname{PP})\,\right] = \operatorname{neg}(\kappa)$$

Definition 3 N-Strong Diffie-Hellman assumption (N-SDH), [4]. The N-Strong Diffie-Hellman (N-SDH) problem consists in computing $(c, g^{\frac{1}{s+c}})$ given PP. We say the N-SDH assumption holds if for any PPT adversary $\mathcal A$ we have

$$Adv^{N\text{-}SDH}(\mathcal{A},\kappa,N) = \Pr\left[\, (c,g^{\frac{1}{s+c}}) \leftarrow \mathcal{A}(1^\kappa, \operatorname{PP}) \, \right] = \operatorname{neg}(\kappa)$$

As mentioned in [4], the N-SDH assumption is equivalent to the N-DHI assumption when c is fixed. The following assumption can be considered as a particular case of the poly-Diffie-Hellman assumption [23], or a generalization of the N+1-Exponent assumption introduced in [30].

Definition 4 N+i-Diffie-Hellman Exponent(N+i-DHE) assumption. The N+i-Diffie-Hellman Exponent problem consists in computing $g^{s^{N+i}}$, for

 $1 \leq i \leq N$ given PP. We say the N+i-DHE assumption holds if for any PPT adversary A we have

$$Adv^{N+i\text{-}DHE}(\mathcal{A},\kappa,N) = \Pr\left[\left.g^{s^{N+i}} \leftarrow \mathcal{A}(1^{\kappa},\operatorname{PP})\right.\right] = \operatorname{neg}(\kappa)$$

In [30], the N-DHI assumption was shown to be equivalent to the N+1-Exponent assumption (N+1-DHE). We state here the following implication.

Proposition 1 N- $BDHI \Rightarrow N+i$ -DHE.

Proof. Let \mathcal{A} be a PPT adversary that breaks the N+i-DHE assumption. We build the following adversary \mathcal{B} . \mathcal{B} receives the challenge tuple $g, g^s, g^{s^2}, ..., g^{s^N}$. He sets $h = g^{s^N}$. Then if we consider $t = \frac{1}{s}$ we have that: $(h, h^t, h^{t^2}, \cdots, h^{t^N}) = (g^{s^N}, g^{s^{N-1}}, g^{s^{N-2}}, \cdots, g)$. \mathcal{B} sends the tuple $(h, h^t, h^{t^2}, \cdots, h^{t^N})$ to \mathcal{A} who outputs $h' = h^{t^{N+i}}$ where $1 \leq i \leq N$. We have that $h' = h^{t^{N+i}} = g^{s^{N-N-i}} = g^{\frac{1}{s^i}}$. Finally \mathcal{B} outputs $e(h', g^{s^{i-1}}) = e(g, g)^{s^{-i+i-1}} = e(g, g)^{\frac{1}{s}}$

2.5 Digital Signatures

STANDARD DIGITAL SIGNATURES. We denote by SSig = (SKG, SSig, SVf) a standard signature scheme. A pair of private / public keys (sk, pk) is created by running $SKG(1^{\kappa})$. Given a message $m \in \{0, 1\}^*$, a signature on m under pk is $\sigma_m = SSig(sk, m)$. A signature σ on m is deemed valid if and only if $SVf(pk, m, \sigma)$ returns valid. Regarding security, we use the standard notion of existential unforgeability under chosen message attack [17].

BONEH AND BOYEN SIGNATURE SCHEME [4]. We recall here briefly the short signature scheme [4] introduced by Boneh and Boyen. The setup algorithm $\mathsf{BMGen}(1^\kappa)$ generates the public parameters of the scheme $(p, \mathbb{G}, \mathbb{G}_T, e, g)^8$. The key generation algorithm $\mathsf{SKG}(1^\kappa)$ selects random integers $x, y \overset{R}{\leftarrow} \mathbb{Z}_p$ and sets $u = g^x$ and $v = g^y$. The secret key is sk = (g, x, y) and the public key is pk = (g, u, v). Given a message m and sk, the signing algorithm $\mathsf{SSig}(sk, m)$ works as follows. It selects $r_\sigma \overset{R}{\leftarrow} \mathbb{Z}_p$ such that $r_\sigma - (x + m)/y \neq 0 \mod p$ and return the (randomized) signature $\sigma = (g^{\frac{1}{x+m+yr_\sigma}}, r_\sigma) = (\sigma', r_\sigma)$. Finally, in order to verify a signature σ on message m relative to the public key pk, the algorithm $\mathsf{SVf}(pk, m, \sigma)$ consists in checking that $e(\sigma', ug^m v^{r_\sigma}) = e(g, g)$. The scheme is secure in the standard model under the N-SDH assumption.

2.6 Simultaneous Hardness of Bits for Discrete Logarithm

Our construction relies on the idea of releasing gradually the bits of $\theta \in \mathbb{Z}_p$, the discrete logarithm in base g of $D = g^{\theta}$. A problem that could arise in this situation would be that some θ values are somehow easier to find than others, especially when some of the bits are released. This might help an adversary to retrieve θ much faster (by a factor greater than a polynomial) and thus break the security of our protocol. To overcome this issue we need to introduce the *Simultaneous hardness* of bits of the discrete logarithm assumption which states that a polynomial time adversary cannot distinguish between a random sequence of $l = \kappa - \omega(\log \kappa)$ bits and the first l bits of θ when given

⁸We use symmetric bilinear map for the sake of exposition.

⁹Note that a PPT adversary can easily distinguish both bit strings if $l = \kappa - O(\log \kappa)$ by performing a brute force attack on the remaining bits as $2^{O(\log \kappa)}$ is a polynomial in κ .

$$D = g^{\theta}$$
.

Definition 5 (Simultaneous hardness of bits for discrete logarithm) Let \mathbb{G} be a cyclic group of prime order p. We say that the Simultaneous hardness of bits for discrete logarithm (SHDL) assumption holds, if for every PPT adversary \mathcal{A} and for any $l = \omega(\log \kappa)$, we have that the following quantity is negligible in κ :

$$Adv^{SHDL}(\mathcal{A}, \kappa) = |\Pr \left[\begin{array}{c} \theta \xleftarrow{\mathcal{R}} \mathbb{Z}_p : \\ 1 \leftarrow \mathcal{A}(g^{\theta}, \theta[1..\kappa - l]) \end{array} \right] - \Pr \left[\begin{array}{c} \theta, \alpha \xleftarrow{\mathcal{R}} \mathbb{Z}_p : \\ 1 \leftarrow \mathcal{A}(g^{\theta}, \alpha[1..\kappa - l]) \end{array} \right] |$$

where the probability is taken over the random choices of A.

Schnorr [29] showed that the SHDL holds in the generic group model by computing the following upper bound on the advantage of the adversary:

$$Adv^{SHDL}(\mathcal{A},\kappa) = O(\kappa(\kappa - l)\sqrt{t}(\frac{2^{\kappa - l}}{2^{\kappa}})^{1/4})$$

where t is the number of generic group operations of the adversary. Thus, if we set $l = \omega(\log \kappa)$, we obtain that $Adv^{SHDL}(\mathcal{A}, \kappa) = O(\kappa(\kappa - \omega(\log \kappa))\sqrt{t}(2^{-\omega(\log \kappa)})^{1/4})$, which is negligible.

The recent work [13] by Duc and Jetchev suggests that results applying to groups of integers modulo a safe prime [27, 24] can be extended to elliptic curves so to reduce the SHDL assumption to more standard ones.

3 A New Argument to Prove a Commitment Encrypts a Bit

In this section we describe a commitment scheme to encrypt a vector of values in \mathbb{Z}_p and then provide a NIZK proof that each component of this vector is a bit. Our technique borrows from [21] in the sense we use the idea that if the value b encrypted is a bit then b(b-1) must be equal to 0, and also from [20] by implementing a basic form of the restriction argument.

Our commitment scheme requires to generate a common reference string CRS = $(g, g^s, g^{s^2}, ..., g^{s^N})$ = $(g_0, g_1, ..., g_N)$ where $s \stackrel{\mathbb{R}}{\leftarrow} \mathbb{Z}_p$ is the trapdoor. To commit a bit b_i in position i using randomness $r_i \in \mathbb{Z}_p$, we compute the following slight variation of the Pedersen commitment Commit $(b_i, r_i, i) = C_i = g^{r_i} g_i^{b_i}$. The commitment to the vector $\vec{B} = (b_1, b_2, ..., b_N)$ using the randomness $\vec{r} = (r_i)_{i \in [N]}$ will simply be the vector formed by the commitments for each bit in position i: $\vec{C} = (C_i)_{i \in [N]}$. Abusing a bit our notation, we will write $\vec{C} = \text{Commit}(\vec{B}, \vec{r})$.

We still need a NIZK that each commitment C_i is the encryption of a bit. The prover proceeds as follows: He computes the "translation" of the commitment by N-i positions to the right, by providing the value $A_i = g_{N-i}^{r_i} g_N^{b_i}$. If we compute $e(A_i, C_i g^{-1})$ and try to express this quantity as $e(B_i, g)$, we realize by simple inspection (see correctness proof of Theorem 2) that a factor $g_{N+i}^{b_i(b_i-1)}$ will appear. Obviously the prover does not know g_{N+i} so in case $b_i \notin \{0, 1\}$ he will not be able to provide the second part of the proof, B_i . If b_i is indeed a bit then the prover will compute the proof $\pi_i = (A_i, B_i)$ in order to convince the verifier that C_i is the encryption of a bit relative to position i.

```
Common reference string: Input (1^{\kappa}, N)
     1. (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BMGen}(1^{\kappa})
     2. s \stackrel{R}{\leftarrow} \mathbb{Z}_n
     3. Return CRS =< (p, \mathbb{G}, \mathbb{G}_T, g), (g_0, g_1, g_2, ..., g_N) > where for all i \in [0..N] : g_i = g^{s^i}.
Statement: The statement is formed by a vector of elements of \mathbb{G}: (C_1, C_2, ..., C_N). The claim is that for
each i \in [N] there exists r_i, b_i such that C_i = g^{r_i} g_i^{b_i} where b_i \in \{0, 1\}.
Proof: Input (CRS, \vec{B}, \vec{r})
     1. Check that \vec{B} = (b_1, ..., b_N) \in \{0, 1\}^N. Return \perp if this is not the case.
     2. Check that \vec{r} = (r_1, ..., r_N) \in \mathbb{Z}_p^N. Return \perp if this is not the case.
     3. For each i \in [N] compute an argument \pi_i that C_i is the commitment to a bit in base g_i: \pi_i = (A_i, B_i) where A_i = C_i^{s^{N-i}} and B_i is such that e(A_i, C_i g_i^{-1}) = e(B_i, g).
     4. Return \pi = (\pi_i)_{i \in [N]}.
Verification: Input (CRS, \vec{C}, \pi)
     1. Parse \vec{C} as (C_i)_{i \in [N]}. Check that \vec{C} \in \mathbb{G}^N.
     2. Parse \pi as ((A_i,B_i))_{i\in[N]}. Check that \pi\in(\mathbb{G}\times\mathbb{G})^N.
3. For each i\in[N] check that:
             (a) e(C_i, g_{N-i}) = e(A_i, g).
             (b) e(A_i, C_i g_i^{-1}) = e(B_i, g).
     4. Return valid if and only if all check pass, otherwise return ⊥
```

Figure 2: NIZK proof of a commitment being the encryption of a binary vector.

Proposition 2 The vector commitment scheme described above is perfectly hiding and computationally binding under the N-BDHI assumption.

Proof. Here we have that $\tau = s$. If we define $\mathsf{TCommit}(\tau) = (C = g^r g_i^m, ek = (m, r))$ for $m, r \in \mathbb{Z}_p$, we have that $\mathsf{TOpen}(C, ek, m')$ will return $r' = r + s^i(m - m')$. So the scheme is perfectly trapdoor.

Assume an adversary \mathcal{A} computes $\vec{B}, \vec{B'} \in \mathbb{Z}_p^N$ two vectors of messages and $\vec{r} = (r_1, ..., r_N), \vec{r'} = (r'_1, ..., r'_N) \in \mathbb{Z}_p^N$ two randomness vectors such that $\mathsf{Commit}(\vec{B}, \vec{r}) = \mathsf{Commit}(\vec{B'}, \vec{r'})$ and $B[j] \neq B'[j]$ for (at least) one $j \in [N]$: We obtain the equation $g^{r_j - r'_j} g^{(B[j] - B'[j])s^j} = 1_{\mathbb{G}}$. If we set $X = s^j$, we can deduce that $(r_j - r'_j) + (B[j] - B'[j])X = 0$ and then $X = s^j = \frac{r'_j - r_j}{B[j] - B'[j]} \mod p$. Once s^j is recovered we can compute $g_N^X = g_{N+j}$ and by proposition 1, the N-BDHI assumption is broken.

The proof of the following theorem can be found in Section A.

Theorem 1 The protocol of Fig. 2 is a NIZK proof that the statement $\vec{C} = (C_i)_{i \in [N]}$ is such that for every $i \in [N]$ there exists $(r_i, b_i) \in (\mathbb{Z}_p \times \{0, 1\})$ with $C_i = g^{r_i} g_i^{b_i}$. The NIZK proof has perfect completeness, perfect zero-knowledge and computational soundness under the N-BDHI assumption.

4 Base Equivalence Argument

Let $\theta \stackrel{R}{\leftarrow} \mathbb{Z}_p$. Consider the commitment to the bit vector $\vec{C} = (C_i)_{i \in [\kappa]} = (g^{r_i} g_i^{\theta[i]})_{i \in [\kappa]}$ where $r_i \in \mathbb{Z}_p$ for each $i \in [\kappa]$ and also $D = g^{\theta}$. In this section we introduce a NIZK proof to show that indeed each bit commitment in position i, C_i , encrypts the i^{th} bit of θ , which is hidden as the discrete logarithm of D. This argument will allow us to blind the signature with some random factor θ (in the exponent) and then reveal each bit of this exponent gradually without leaking any additional information. The idea is the following. Given $\theta \in \mathbb{Z}_p$ and $\vec{C} = (g^{r_i}g_i^{\theta[i]})_{i \in [\kappa]}$, the prover proceeds in two steps. First he computes $D' = \frac{\prod_{i \in [\kappa]} g^{r_i}g_i^{\theta[i]}}{g^r}$ where $r = \sum_{i \in [\kappa]} r_i$. Here the prover computes some compressed representation of the bit vector commitment and removes the randomness. Observe however that as θ is uniformly random, thus so is D'. The prover will need to convince the verifier that r is indeed the accumulated randomness of the bit vector commitment. To do so he computes $U = D'^{\frac{1}{s}} = (\prod_{i \in [\kappa]} g_i^{\theta[i]})^{\frac{1}{s}} = \prod_{i \in [\kappa]} g_{i-1}^{\theta[i]}$ where we recall that $g_0 = g$. Observe that this value can be computed without knowing s. In order to verify this proof, the verifier will check that $e(\frac{\prod_{i\in[\kappa]}C_i}{g^r},g)=e(U,g_1)$. Intuitively, once the randomness of the bit vector is removed one can move the vector to the left by one position. If r would not be equal to $\sum_{i\in[\kappa]}r_i$, this would not be possible without breaking some assumption. The second step consists in checking that the condensed bit vector commitment $U = \prod_{i \in [\kappa]} g_{i-1}^{\theta[i]}$ is "equivalent" to the simple commitment g^{θ} . This is done by noting that $U = \prod_{i \in [\kappa]} g_{i-1}^{\theta[i]} = g^{P(s)}$ where $P(\cdot)$ is the polynomial $P(X) = \sum_{i \in [\kappa]} \theta[i] X^{i-1}$. This means in particular that $P(2) = \sum_{i \in [\kappa]} \theta[i] 2^{i-1} = \theta$. Thus, we need to prove that $P(s) - P(2) = P(s) - \theta$ is divisible by s - 2. The prover can compute the coefficients of the formal polynomial $W(\cdot)$ such that P(X) - P(2) = W(X)(X-2), then using the common reference string CRS the prover obtains $V=g^{W(s)}$. Verifying the "base equivalence" statement consists in checking that $e(\frac{U}{D},g)=e(V,g_1g^{-2})=e(V,g^{s-2})$. This means that indeed $\theta = P(2)$ and thus the coefficients of $P(\cdot)$ correspond to the binary decomposition of θ . The full protocol is detailed in Fig. 3.

Theorem 2 The protocol in Fig. 3 is a NIZK proof that the bits of the discrete logarithm of D correspond to the bit vector committed in $(C_i)_{i \in [\kappa]}$. The NIZK proof has perfect completeness, perfect zero-knowledge and computational soundness under the κ -SDH assumption.

The proof can be found in Section B.

5 Fair Exchange of Short Signatures without TTP

Our fair exchange protocol for digital signatures works as follows. At the beginning a common reference string CRS is generated. Then each participant runs $\mathsf{FEKeyGen}(1^\kappa)$ to obtain a pair of (public/private) keys (pk, sk) for the signing algorithm. At this point each participant executing $\mathsf{EncSigGen}(\mathsf{CRS}, sk, m)$ will compute an encrypted signature γ for the message m, using the signature σ_m blinded with some factor θ . This value γ will also contain the proofs that relate the signature σ_m with some bit vector commitment to θ .

The rest is straightforward: Each participant sends the encrypted signature. If all the verifications pass, the first participant \mathcal{P}_A will ask to \mathcal{P}_B to open the commitment of the first bit of θ_A .

```
Common reference string: Input (1^{\kappa}, \kappa)
1. (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BMGen}(1^{\kappa}).
       3. Return CRS = \langle (p, \mathbb{G}, \mathbb{G}_T, e), (g_0, g_1, g_2, ..., g_{\kappa}) \rangle where for all i \in [0..\kappa] : g_i = g^{s^i}.
Statement: The statement is formed by a vector of elements of \mathbb{G}: (D, C_1, C_2, ..., C_{\kappa}) where (C_i)_{i \in [\kappa]} is a commitment to a bit vector as defined in Sect. 3. The claim is that the vector formed by the binary
decomposition of the discrete logarithm of D is equal to the bit vector committed in (C_i)_{i\in[\kappa]}.
Proof: Input (CRS, \theta, r_1, ..., r_{\kappa})
       1. Check that D = g^{\theta}. Return \bot if this is not the case.
      2. Compute for every i \in [\kappa]: C_i = g^{r_i} g_i^{\theta[i]}.
       3. Compute r = \sum_{i \in [\kappa]} r_i.
      4. Compute U = (\frac{\prod_{i \in [\kappa]}^{r} C_i}{g^r})^{\frac{1}{s}} using the common reference string CRS and the bit vector \theta[\cdot].
      5. Compute the formal polynomial W(\cdot) such that P(X) - P(2) = W(X)(X - 2) where P(X) = \sum_{i \in [\kappa]} \theta[i] X^{i-1}, and P(2) = \sum_{i \in [\kappa]} \theta[i] 2^{i-1} = \theta. Compute V = g^{W(s)} using the coefficients of the formal polynomial W(\cdot) and the common reference string CRS.
       6. Return \pi = (r, U, V).
Verification: Input (CRS, C, \pi)
      1. Parse C as (D, (C_i)_{i \in [\kappa]}).
2. Parse \pi as (r, U, V).
      3. Check that r \in \mathbb{Z}_p.
       4. Check that (U, V, D, C_1, ..., C_{\kappa}) \in \mathbb{G}^{\kappa+3}.
      5. Compute D' = \frac{\prod_{i \in [\kappa]} C_i}{a^r}.
      6. Check that e(D', g) = e(U, g_1).
7. Check that e(\frac{U}{D}, g) = e(V, g_1 g^{-2}).
      8. Accept if all tests pass in which case return valid otherwise return \perp.
```

Figure 3: NIZK proof that a basic commitment is equivalent to a bit vector commitment.

If the opening is successful, \mathcal{P}_B will do the same for its own blinding factor θ_B . The process is repeated for each bit until all the bits of the blinding factors are recovered. Finally, each player can compute the signature by "canceling out" the blinding factor θ . The abstract syntax of the protocol is described in Fig. 4.

We describe now more in detail how the encrypted signature is constructed, which is the core of our construction. The encrypted signature contains:

- 1. A commitment \vec{C} to the bit string formed by the bits of θ as described in Section 3.
- 2. $\tilde{\sigma}$, the signature of the message m blinded by θ .
- 3. Proofs to guarantee that the bit vector commitment encrypts the binary decomposition of the blinding factor θ .
- 4. A proof in order to convince the verifier that γ is the encryption of σ_m under some blinding factor θ which is hidden in the basic commitment g^{θ} .
- 5. A proof of knowledge of the discrete logarithm of D and a proof of knowledge of the representation of each bit commitment of the vector \vec{C} . These proofs of knowledge will allow us to keep simulating the adversary despite it aborts.

A detailed description of the concrete protocol is given in Fig. 5.

```
\mathcal{P}_A(\mathsf{CRS}, m_A, m_B)
                                                                                                      \mathcal{P}_B(\mathsf{CRS}, m_A, m_B)
                                (sk_A, pk_A) \leftarrow \mathsf{FEKeyGen}(1^\kappa)
2
3
                                                                                                      (sk_B, pk_B) \leftarrow \mathsf{FEKeyGen}(1^\kappa)
          (\theta_A, \vec{r}_A, \gamma_A) \leftarrow \mathsf{EncSigGen}(\mathsf{CRS}, sk_A, m_A)
                                                                                                      (\theta_B, \vec{r}_B, \gamma_B) \leftarrow \mathsf{EncSigGen}(\mathsf{CRS}, sk_B, m_B)
8
10
                   v \leftarrow \mathsf{EncSigCheck}(\mathsf{CRS}, pk_B, m_B, \gamma_B)
11
                                        if v = 0 then ABORT
                                                                                                       v \leftarrow \mathsf{EncSigCheck}(\mathsf{CRS}, pk_A, m_A, \gamma_A)
12
                                                                                                      if v = 0 then ABORT
13
for i = 1 to \kappa:
          \mathsf{open}_{A,i} \leftarrow \mathsf{KeyBitProofGen}(\mathsf{CRS}, \vec{r}_A, \theta_A, i)
14
15
                                                                                                      \mathtt{open}_{B,i} \leftarrow \mathsf{KeyBitProofGen}(\mathsf{CRS}, \vec{r}_B, \theta_B, i)
16
17
                        v_i \leftarrow \mathsf{KeyBitCheck}(\mathsf{CRS}, \mathsf{open}_{B,i}, i)
19
                                      if v_i = 0 then ABORT
20
21
                                                                                                       v_i \leftarrow \mathsf{KeyBitCheck}(\mathsf{CRS}, \mathsf{open}_{A,i}, i)
22
                                                                                                       if v_i = 0 then ABORT
end for
                             \sigma_{m_B} \leftarrow \mathsf{EncSigDecrypt}(\gamma_B, \theta_B)
23
24
                                                                                                      \sigma_{m_A} \leftarrow \mathsf{EncSigDecrypt}(\gamma_A, \theta_A)
```

Figure 4: Abstract fair exchange protocol.

We say that the protocol is perfectly complete¹⁰ if, and only if, both players \mathcal{P}_A and \mathcal{P}_B that

 $^{^{10}\}mathrm{Here}\ complete$ does not refer to fairness.

follow the protocol obtain respectively $\sigma_A = \text{SSig}(sk_B, m_B)$, the signature of message m_B and $\sigma_B = \text{SSig}(sk_A, m_A)$, the signature of message m_A , with probability 1.

We say that the protocol is *(partially) fair* if, at the end of the execution of the protocol (be it normal or anticipated by the abortion of the adversary), the probability of both players to recover their corresponding signature differs at most by a polynomial factor in the security parameter κ . As mentioned in the introduction, the advantage of this approach is that it avoids trying to compare the *exact* running time of the participants and thus allows to capture in a simple, but precise manner, the intuition of *partial fairness*.

Definition 6 (Partial fairness) We define the partial fairness of the protocol through the following experiment: The adversary \mathcal{A} plays the role of the corrupted player say w.l.o.g. \mathcal{P}_A . Thus, \mathcal{P}_B is honest and follows the protocol. $\mathcal{O}_{SSig}(\cdot)$ is the signing oracle for the signature scheme SSig relative to the public key pk_B of \mathcal{P}_B .

- 1. A asks for signature computations for arbitrary messages to $\mathcal{O}_{\mathsf{SSig}}(\cdot)$.
- 2. A chooses the messages m_A and m_B on which the fair exchange protocol will be run, with the restriction that m_B must not have been requested before to $\mathcal{O}_{SSig}(\cdot)$.
 - \mathcal{A} computes also its public key pk_A and sends it to \mathcal{P}_B .
- 3. A then interacts in arbitrary way with \mathcal{P}_B .
- 4. If A has aborted before ending the protocol, then let $\theta_A^*[1..i]$ ($0 \le i \le \kappa$) be the partial blinding obtained by \mathcal{P}_B . At this point we assume that \mathcal{P}_B will try to compute $\mathsf{SSig}(sk_A, m_A)$ by choosing at random some element in the remaining space of size $2^{\kappa-i}$. We call this tentative signature σ_B .
- 5. A keeps running its own algorithm and finally outputs a tentative signature σ_A on m_B relative to public key pk_B .

The protocol is said to be partially fair if and only if there exists some polynomial $Q(\cdot)$ such that

$$\frac{\Pr\left[\,\mathsf{SVf}(pk_B,m_B,\sigma_A)=\mathsf{valid}\,\right]}{\Pr\left[\,\mathsf{SVf}(pk_A,m_A,\sigma_B)=\mathsf{valid}\,\right]} \leq Q(\kappa)$$

where the probability is taken over the random choices of A and P_B .

Theorem 3 The protocol described in Fig. 5 is complete. Moreover if the κ -SDH assumption, the κ -BDHI assumption and the SHDL assumption hold, and the underlying signature scheme is secure, and a securely precomputed common reference string is available, then it is secure in the random oracle model according to definition 6.

The proof can be found in Section C.

As the signature scheme presented in [4] is secure under the κ -SDH assumption, we have

Corollary 1 The protocol described in Fig. 5 is complete. Moreover if the κ -SDH assumption, the κ -BDHI assumption and the SHDL assumption hold, and a securely precomputed common reference string is available, then it is secure in the random oracle model according to definition 6.

6 Conclusion and Future Work

In this work we introduced a practical protocol to exchange short signatures [4] fairly without relying on a TTP. It seems our approach can be applicable to other signature schemes or more generally to the exchange of values which are computed from a secret and are publicly verifiable using bilinear maps. Thus, our techniques might be extended in order to obtain a general framework to build practical fair protocols involving bilinear maps.

ACKNOWLEDGMENTS. The author is very grateful to Anna Lysyanskaya for pointing out a gap in the proof of Theorem 3.

References

- [1] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *CCS*, pages 7–17. ACM Press, April 1997.
- [2] Manuel Blum. How to exchange (secret) keys. ACM Transactions on Computer Systems, 1(2):175–193, May 1983.
- [3] Dan Boneh and Xavier Boyen. Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *LNCS*, pages 223–238. Springer Berlin / Heidelberg, 2004.
- [4] Dan Boneh and Xavier Boyen. Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. *Journal of Cryptology*, 21(2):149–177, 2008.
- [5] Dan Boneh and Moni Naor. Timed Commitments. In Mihir Bellare, editor, CRYPTO, volume 1880 of LNCS, pages 236–254. Springer Berlin / Heidelberg, August 2000.
- [6] Ernest Brickell, David Chaum, Jeroen van De Graaf, and Ivan Bjerre Damgård. Gradual and verifiable release of a secret. In Carl Pomerance, editor, CRYPTO, volume 293 of LNCS, pages 156–166. Springer Berlin / Heidelberg, 1987.
- [7] Jan Camenisch and Markus Stadler. Proof Systems for General Statements about Discrete Logarithms. ftp://ftp.inf.ethz.ch/pub/crypto/publications/CamSta97b.ps, 1997.
- [8] David Chaum, Ivan Bjerre Damgård, and Jeroen van de Graaf. Multiparty Computations Ensuring Privacy of Each Party's Input and Correctness of the Result. In David L Chaum, editor, *CRYPTO*, pages 87–119. Springer-Verlag, August 1987.
- [9] Seung Geol Choi, Kyung-Wook Hwang, Jonathan Katz, Tal Malkin, and Dan Rubenstein. Secure Multi-Party Computation of Boolean Circuits with Applications to Privacy in On-Line Marketplaces. In Orr Dunkelman, editor, CT-RSA 2012, volume 7178 of LNCS, pages 416–432. Springer Berlin / Heidelberg, 2012.
- [10] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *STOC*, pages 364–369. ACM Press, November 1986.

- [11] Richard Cleve. Controlled Gradual Disclosure Schemes for Random Bits and Their Applications. In Gilles Brassard, editor, CRYPTO, volume 435 of LNCS, pages 573–588. Springer Berlin / Heidelberg, July 1990.
- [12] Ivan Damgård. Practical and Provably Secure Release of a Secret and Exchange of Signatures. Journal of Cryptology, 8(4):201–222, September 1995.
- [13] Alexandre Duc and Dimitar Jetchev. Hardness of Computing Individual Bits for One-Way Functions on Elliptic Curves. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *LNCS*, pages 832–849. Springer Berlin / Heidelberg, 2012.
- [14] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, June 1985.
- [15] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *LNCS*, pages 186–194. Springer-Verlag, January 1986.
- [16] Juan A. Garay, Philip MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource Fairness and Composability of Cryptographic Protocols. In Ran Canetti, editor, TCC, volume 3876 of LNCS, pages 404–428. Springer, 2006.
- [17] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. SIAM Journal on Computing, 17(2):281, April 1988.
- [18] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete Fairness in Secure Two-Party Computation. *Journal of the ACM*, 58(6):1–37, December 2011.
- [19] S. Dov Gordon and Jonathan Katz. Partial Fairness in Secure Two-Party Computation. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *LNCS*, pages 157–176. Springer Berlin / Heidelberg, 2010.
- [20] Jens Groth. Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In Masayuki Abe, editor, ASIACRYPT, volume 6477 of LNCS, pages 321–340. Springer Berlin / Heidelberg, 2010.
- [21] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect Non-interactive Zero Knowledge for NP. In Serge Vaudenay, editor, EUROCRYPT, volume 4004 of LNCS, pages 339–358. Springer Berlin / Heidelberg, 2006.
- [22] Qiong Huang, Duncan S. Wong, and Willy Susilo. The Construction of Ambiguous Optimistic Fair Exchange from Designated Confirmer Signature without Random Oracles. In Marc Fischlin, J. Buchmann, and Mark Manulis, editors, *PKC*, volume 7293 of *LNCS*, pages 120–137. Springer Berlin Heidelberg, 2012.
- [23] Aniket Kate, Gregory Zaverucha, and Ian Goldberg. Constant-Size Commitments to Polynomials and Their Applications. In Masayuki Abe, editor, ASIACRYPT, volume 6477 of LNCS, pages 177–194–194. Springer Berlin / Heidelberg, 2010.

- [24] Philip Mackenzie and Sarvar Patel. Hard Bits of the Discrete Log with Applications to Password Authentication. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *LNCS*. Springer Berlin Heidelberg, 2005.
- [25] Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC*, pages 12–19, New York, New York, USA, July 2003. ACM Press.
- [26] S Mitsunari, R Sakai, and M Kasahara. A New Traitor Tracing. EICE, E85-A:481-484, 2002.
- [27] Sarvar Patel and Ganapathy S. Sundaram. An efficient discrete log pseudo random generator. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *LNCS*. Springer-Verlag, 1998.
- [28] Torben Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *LNCS*, pages 129–140. Springer Berlin / Heidelberg, 1991.
- [29] C.P. Schnorr. Security of Almost ALL Discrete Log Bits. *Electronic Colloquium on Computational Complexity*, 1998.
- [30] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. An Efficient Signature Scheme from Bilinear Pairings and Its Applications. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *PKC*, volume 2947 of *LNCS*, pages 277–290. Springer Berlin / Heidelberg, 2004.

A Proof of Theorem 1

Proof. Perfect Completeness. Let $i \in [N]$. It's clear that the prover can compute $A_i = C_i^{s^{N-i}}$. Then if $b_i := \vec{B}[i] \in \{0,1\}$ we have $b_i(b_i-1) = 0$ and

$$e(A_{i}, C_{i}g_{i}^{-1}) = e(g^{(r_{i}+b_{i}s^{i})s^{N-i}}, g^{r_{i}}g^{(b_{i}-1)s^{i}})$$

$$= e(g^{(r_{i}s^{N-i}+b_{i}s^{N})(r_{i}+(b_{i}-1)s^{i})}, g)$$

$$= e(g^{r_{i}^{2}s^{N-i}+r_{i}b_{i}s^{N}+r_{i}(b_{i}-1)s^{N}+b_{i}(b_{i}-1)s^{N+i}}, g)$$

$$= e(g^{r_{i}^{2}s^{N-i}+s^{N}r_{i}(2b_{i}-1)}, g)$$

$$= e(B_{i}, g)$$

We can see that the prover can compute B_i as he knows r_i and b_i and the group elements g_{N-i}, g_N are public.

Computational Soundness. Assume there exists some adversary \mathcal{A} that breaks the soundness of the scheme (that is \mathcal{A} is able to open the commitment to some vector that does not contain bits) for at least one i. We build the following adversary \mathcal{B} that breaks the N-BDHI assumption. \mathcal{B} receives the challenge tuple $(g_0, g_1, g_2, ..., g_N)$. \mathcal{B} runs \mathcal{A} using the tuple as the CRS and obtains C_i, r_i, b_i such that $C_i = g^{r_i} g_i^{b_i}$ and $\pi_i = (A_i, B_i)$ where $e(C_i, g_{N-i}) = e(A_i, g)$ (1) and $e(A_i, C_i g_i^{-1}) = e(B_i, g)$ (2).

From (1) we can deduce that $A_i = C_i^{s^{N-i}}$. From (2) (as seen in the correctness proof) we have that $B_i = g^{r_i^2 s^{N-i} + r_i(2b_i-1)s^N + b_i(b_i-1)s^{N+i}} = g_{N-i}^{r_i^2} g_N^{r_i(2b_i-1)} g_{N+i}^{b_i(b_i-1)}$

 \mathcal{B} can compute $X = g_{N-i}^{r_i^2} g_N^{r_i(2b_i-1)}$, so it can obtain $g_{N+i} = (B_i \cdot X^{-1})^{\frac{1}{b_i(b_i-1)}}$ where $b_i(b_i-1) \neq 0$ as $b_i \notin \{0,1\}$, and thus the N+i-DHE assumption is broken. Using proposition 1 we have that the N-BDHI assumption is broken as well.

Perfect zero-knowledge. We justify why the argument is perfectly witness-indistinguishable. Consider a pair of witnesses $(r_i,b_i),(r_i',b_i')$ for some commitment $C_i=g^{r_i}g_i^{b_i}=g^{r_i'}g_i^{b_i'}$. First observe that there is only one possible argument that satisfies equations $A_i=A_i'=C_i^{s^{N-i}}$. Secondly we have that $B_i=g^{r_i^2s^{N-i}+r_i(2b_i-1)s^N}$ and $B_i'=g^{r_i'^2s^{N-i}+r_i'(2b_i'-1)s^N}$ are uniformly random and thus are perfectly indistinguishable.

We describe now the zero-knowledge simulator. It generates the common reference string correctly, and also learns the trapdoor so it can create commitments that can be opened to any value. As the commitment can be opened, it is straightforward to compute an argument A_i , B_i .

Let us justify why the simulator simulates perfectly the real argument. Consider the hybrid stateful algorithm where the simulator generates the trapdoor and the common reference string but opens the commitment to the real bit b_i . Then as the randomness is known, the hybrid algorithm can compute the argument as well. As the commitment is perfectly trapdoor, the real argument is perfectly indistinguishable from the hybrid algorithm. Finally as the argument is perfect witness-indistinguishable the hybrid is perfectly indistinguishable from the simulated argument.

B Proof of Theorem 2

Proof. Perfect Completeness. The reason why the prover can compute $U=(\frac{\prod_{i\in[\kappa]}C_i}{g^r})^{\frac{1}{s}}$ without knowing s is because $U=\prod_{i\in[\kappa]}g_{i-1}^{\theta[i]}$. Indeed U corresponds to the vector $(0,\theta[1],...,\theta[\kappa])$ that is moved by one position to the left. Similarly V can be computed because the prover knows the coefficients W[i] of the polynomial $W(\cdot)$ of degree $\kappa-2$, so we have $V=\prod_{i\in[\kappa-1]}g_{i-1}^{W[i]}$. The rest follows by inspection.

Computational Soundness. Let \mathcal{A} be the PPT adversary that breaks the soundness of the scheme. We build the following adversary \mathcal{B} . \mathcal{B} receives the challenge tuple $(g_0, g_1, g_2, \dots, g_{\kappa})$. This challenge tuple stands for the CRS and is sent to \mathcal{A} . \mathcal{A} returns the following values:

- $\theta \in \mathbb{Z}_p$.
- $\theta^* \in \mathbb{Z}_p$ such that $D = g^{\theta^*}$ and $\theta \neq \theta^*$.
- $(r_i, \theta[i]) \in (\mathbb{Z}_p \times \{0, 1\})^{\kappa}$ for $i \in [\kappa]$ such that $\vec{C} = (C_i)_{i \in [\kappa]}$ where $C_i = g^{r_i} g_i^{\theta[i]}$.
- $\pi = (r, U, V) \in \mathbb{Z}_p \times \mathbb{G} \times \mathbb{G}$.

Assume first that $r \neq \sum_{i \in [\kappa]} r_i \mod p$ then we can deduce that $U = (g^{\sum_{i \in [\kappa]} r_i - r} g^{\theta_i s^i})^{1/s}$. As $g^{\frac{\theta_i s^i}{s}} = g_{i-1}^{\theta_i}$ is easily computable due to the fact that $\theta[i]$ are known, \mathcal{B} can deduce $g^{\frac{\sum_{i \in [\kappa]} r_i - r}{s}}$ and as $\delta = \sum_{i \in [\kappa]} r_i - r \neq 0 \mod p$ is known, \mathcal{B} can compute $g^{\frac{1}{s}} = (\frac{U}{g_{i-1}^{\theta_i}})^{\frac{1}{\delta}}$ and thus the κ -DHI assumption is broken. From now on we assume that $\sum_{i \in [\kappa]} r_i = r$. As the adversary \mathcal{A} wins, this means that there exists some $j \in [\kappa]$ such that $\theta[j] \neq \theta^*[j]$. Moreover as the verification involving V passes we have that $V = g^{\frac{\sum_{i \in [\kappa]} \theta[i] s^{i-1} - \theta^*[i] 2^{i-1}}{s-2}$. As the decomposition in binary is unique we have that $\Delta = \sum_{i \in [\kappa]} 2^{i-1} (\theta[i] - \theta^*[i]) \neq 0 \mod p$. We can rewrite V as $V = g^{\frac{\Delta}{s-2} + \frac{\sum_{i \in [\kappa]} \theta[i] s^{i-1} - \theta[i] 2^{i-1}}{s-2}} = g^{\frac{\Delta}{s-2} + Z(s)}$ where the coefficients of $Z(\cdot)$ are efficiently computable by

 \mathcal{B} because $\forall i \in [\kappa] : s - 2|s^{i-1} - 2^{i-1}$. As $\Delta \in \mathbb{Z}_p$ is also known this means \mathcal{B} can compute $g^{\frac{1}{s-2}} = (\frac{V}{\sigma^{\mathbb{Z}(s)}})^{\frac{1}{\Delta}}$ and thus the κ -SDH assumption is broken.

Perfect zero-knowledge. The simulator works as follows. It generates the common reference string CRS correctly and saves the trapdoor s. Given the statements D and $\vec{C} = (C_i)_{i \in [\kappa]}$ such that \vec{C} is formed by Pedersen commitments to bits in positions $1, ..., \kappa$ and such that D and \vec{C} are equivalent with respect to bases (2, s), the simulator chooses a random $r' \in \mathbb{Z}_p$ and reveals it as the randomness of $\prod_{i \in [\kappa]} C_i$. Then, the simulator sets $U = (\frac{\prod_{i \in [\kappa]} C_i}{g^{r'}})^{\frac{1}{s}}$ and $V = (\frac{U}{D})^{\frac{1}{s-2}}$. To see that r', U, V produced by the simulator are indistinguishable from values of a real experiment, we observe that:

- r' is uniformly distributed as well as r.
- D' is equal to $f_1(r')$ in the simulated experiment and $f_1(r)$ in the real experiment, where $f_1: \mathbb{G} \to \mathbb{G}$ is defined as $f_1(x) = \frac{\prod_{i \in [\kappa]} C_i}{x}$
- U is equal to $f_2(r')$ in the simulated experiment and $f_2(r)$ in the real experiment, where $f_2: \mathbb{G} \to \mathbb{G}$ is defined as $f_2(x) = f_1(x)^{\frac{1}{s}}$.
- V is equal to $f_3(r')$ in the simulated experiment and $f_3(r)$ in the real experiment, where $f_3: \mathbb{G} \to \mathbb{G}$ is defined as $f_3(x) = (\frac{f_2(x)}{D})^{\frac{1}{s-2}}$.

C Proof of Theorem 3 and detailed description of the protocol

Proof. As the underlying NIZK arguments are perfectly complete, we can see by inspection that so is the fair exchange protocol. Let \mathcal{A} be the adversary that breaks the fairness of our protocol.

We distinguish between two types of adversaries A:

- Type I: the adversary \mathcal{A} does not lie: That is he does not forge values for the NIZK arguments and/or the commitments. Said differently, adversary \mathcal{A} follows the protocol, but may abort prematurely.
- Type II: the adversary A lies.

Note that both adversaries may abort during the execution of the protocol. The simulator \mathcal{B} will choose with probability 1/2 to bet that \mathcal{A} is of type I and with probability 1/2 that \mathcal{A} is of type II.

What happens in case of an adversary of type I, is that the only way for \mathcal{A} to win his by breaking the signature scheme (without even caring about the values sent by \mathcal{P}_B). This means the simulator \mathcal{B} will try to build an adversary that breaks the security of the signature scheme. The case of adversary of type II is more straightforward: as \mathcal{A} lies, the simulator will be able to break the soundness of some of the arguments involved or the binding property of the commitment scheme.

```
\begin{aligned} \mathsf{FESetup}(1^\kappa) \\ 1. \ \ (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BMGen}(1^\kappa) \end{aligned}
      2. s \stackrel{R}{\leftarrow} \mathbb{Z}_n
      3. Return CRS = \langle (p, \mathbb{G}, \mathbb{G}_T, e, g), (g_0, g_1, g_2, ..., g_{\kappa}) \rangle where for all i \in [0..\kappa] : g_i = g^{s^i}.
\mathsf{FEKeyGen}(1^{\kappa})
      1. (sk, pk) \leftarrow \mathsf{SKG}(1^{\kappa}) where sk = (q, x, y) and pk = (q, u, v) with u = q^x and v = q^y, like described in
           section 2.5.
      2. Return (sk, pk).
\mathsf{EncSigGen}(\mathsf{CRS}, sk, m)
      1. Compute \theta \stackrel{R}{\leftarrow} \mathbb{Z}_p.
      2. Compute D = g^{\theta}
      3. Compute \vec{C} = (C_i)_{i \in [\kappa]} = (g^{r_i} g_i^{\theta[i]})_{i \in [\kappa]}.
      4. Compute \pi_1 that shows that \vec{C} is the encryption of a binary vector as described in figure 2.
      5. Compute \pi_2 that shows that \vec{C} is the encryption of the bits of the binary decomposition of the blinding
           factor \theta as described in figure 3.
      6. Compute PK_{\theta} = PK\{\theta : g^{\theta}\} as described in section ??.
      7. Compute \vec{PK}, a vector where each component at position i is ZKPoK for the representation of C_i in base (g,g_i). \vec{PK} = (PK\{(r_i,\theta[i]):g^{r_i}g_i^{\theta[i]}\})_{i\in[\kappa]} as described in section ??.
      8. Parse sk as (g, x, y).
     9. Set r_{\sigma} \stackrel{R}{\leftarrow} \mathbb{Z}_{n}.
    10. Compute \sigma = (\sigma', r_{\sigma}) \leftarrow \text{SSig}(sk, m) where \sigma' = g^{\frac{1}{x+m+yr_{\sigma}}}.
    11. Set \tilde{\sigma} \leftarrow (\sigma'^{\theta} = g^{\frac{\theta}{x+m+yr\sigma}}, r_{\sigma}) = (\tilde{\sigma}', r_{\sigma}).
    12. Set \gamma \leftarrow (D, \vec{C}, \pi_1, \pi_2, PK_{\theta}, \vec{PK}, \tilde{\sigma}).
    13. Return (\theta, \vec{r}, \gamma), where \vec{r} = (r_i)_{i \in [\kappa]} is the randomness vector of the commitment \vec{C}.
\mathsf{EncSigCheck}(\mathsf{CRS}, pk, m, \gamma)
      1. Parse \gamma as \gamma = (D, \vec{C}, \pi_1, \pi_2, PK_\theta, \vec{PK}, \tilde{\sigma}).
      2. Check \pi_1 as described in figure 2.
      3. Check \pi_2 as described in figure 3.
      4. Check PK_{\theta} using D and PK_{\theta} as described in section ??.
      5. Check the zero-knowledge proof of knowledge \vec{PK} using \vec{C} and \vec{PK} as described in section ??.
      6. Parse pk as pk = (g, u, v).
      7. Check that e(\tilde{\sigma}, ug^m v_{\sigma}^r) = e(D, g).
      8. Return valid if all tests pass, ⊥ otherwise.
KeyBitProofGen(CRS, \vec{r}, \theta, i)
      1. Opens the i^{\text{th}} commitment of \vec{C}, that is (\theta[i], r_i) such that C_i = g^{r_i} g_i^{\theta[i]}.
      2. Return open \leftarrow (\theta[i], r_i).
\mathsf{KeyBitCheck}(\mathsf{CRS}, \mathsf{open}, i)
      1. Parse open as open =(b,r)
      2. Check that C_i = g^r g_i^b and b \in \{0, 1\}.
\mathsf{EncSigDecrypt}(\gamma, \theta)
      1. Parse \gamma as \gamma = (D, \vec{C}, \pi, PK_{\theta}, \vec{PK}, \tilde{\sigma}).
      2. Parse \tilde{\sigma} as \tilde{\sigma} = (\tilde{\sigma}', r_{\sigma}).
      3. Compute \sigma' = \tilde{\sigma}'^{\frac{1}{\theta}}.
      4. Return \sigma = (\sigma', r_{\sigma})
```

Figure 5: Implementation of the fair exchange protocol.

For adversary \mathcal{A} of type I, we will consider two scenarios: In the first case the simulator \mathcal{B} will try to fool the \mathcal{A} by computing commitments without knowing the signature on message m_B , σ_{m_B} . Thus if \mathcal{A} outputs the signature the simulator will have been able to break the signature scheme. Note however that \mathcal{A} could detect that he is simulated due to the fact that the simulator \mathcal{B} does not know the signature and thus has no access either to the discrete logarithm of D. This means that at some point of the execution of the protocol \mathcal{A} might abort the simulation and make the reduction fail. To handle this case, the simulator \mathcal{B} will change his strategy by using a known signature but trying to distinguish between a sequence of $\kappa - \omega(\log \kappa)$ random bits and the $\kappa - \omega(\log \kappa)$ heading bits of some discrete logarithm.

We now detail both strategies. \mathcal{B} will bet on each strategy with probability 1/4.

Type I - first strategy: \mathcal{B} generates himself the CRS, thus knowing the trapdoor s. When \mathcal{A} chooses the message m_B to be signed, \mathcal{B} will not compute the signature because he wants to break the signature scheme. More concretely \mathcal{B} will

```
    Run BMGen(1<sup>κ</sup>) to obtain (p, G, G<sub>T</sub>, e, g).
    Set s ← Z<sub>p</sub>.
    Compute CRS ←< (p, G, G<sub>T</sub>, e, g), (g<sub>0</sub>, g<sub>1</sub>, ..., g<sub>κ</sub>) > where ∀i ∈ [0..κ]: g<sub>i</sub> = g<sup>s<sup>i</sup></sup>.
    Set θ<sub>B</sub> ← Z<sub>p</sub>.
    Set r<sub>σ</sub> ← Z<sub>p</sub>.
    Set D = (g<sup>m<sub>B</sub></sup> uv<sup>r<sub>σ</sub></sup>)<sup>θ<sub>B</sub></sup>.
    Set σ̃ = (g<sup>θ<sub>B</sub></sup>, r<sub>σ</sub>).
    Set (r<sub>1</sub>, r<sub>2</sub>, ..., r<sub>κ</sub>) ← Z<sub>p</sub><sup>κ</sup>.
    C = (C<sub>i</sub>)<sub>i∈[κ]</sub> = (g<sup>r<sub>i</sub></sup> g<sub>i</sub><sup>θ<sub>B</sub>[i]</sup>)<sub>i∈[κ]</sub>.
```

Note that all the proofs can be computed because \mathcal{B} knows s. Moreover \mathcal{A} is fooled because we have $e(\tilde{\sigma}, g^{m_B}uv^{r_{\sigma}}) = e((g^{m_B}uv^{r_{\sigma}})^{\theta_B}, g) = e(D, g)$ and all values computed by the simulator \mathcal{B} are perfectly indistinguishable from those of a real experiment.

Let σ_B be the signature that is output by \mathcal{B} . As \mathcal{A} does not lie, if he aborts the simulation where only $O(\log \kappa)$ bits remain we have that

 $\Pr[\mathsf{SVf}(pk_A, m_A, \sigma_B) = \mathsf{valid}] \ge \frac{1}{2^{\log(R(\kappa))+1}}$ (+1 because \mathcal{A} has got one bit of advantage). We can deduce that $\frac{\Pr[\mathsf{SVf}(pk_B, m_B, \sigma_A) = \mathsf{valid}]}{\Pr[\mathsf{SVf}(pk_A, m_A, \sigma_B) = \mathsf{valid}]} \le \frac{1}{2^{-(\log(R(\kappa)+1)}} = 2^{\log(R(\kappa))+1} = 2R(\kappa)$ which is still a polynomial, thus \mathcal{A} did not win, so our reduction is still valid. The case where the adversary \mathcal{A} aborts the simulation when $\omega(\log \kappa)$ bits remain is treated in the second strategy. Consider now the case where \mathcal{A} does not abort in the simulation. That is he may abort the protocol but will finally output the signature σ_A , which is an attempt to be a valid signature for message m_B . This means that the simulator, by outputting σ_A computed by adversary \mathcal{A} , managed to break the signature scheme.

Type I - second strategy: \mathcal{B} receives as input some element E which discrete logarithm is unknown and some sequence α of $\kappa - \omega(\log \kappa)$ bits. He also generates himself the CRS and computes the private key for the signature scheme as well. When \mathcal{A} chooses the message m_B to be signed, \mathcal{B} will commit to the bit vector that starts by α and ends with random bits. As \mathcal{B} knows the trapdoor he will be able to compute proofs and make all the tests pass. To summarize \mathcal{B} will

1. Run BMGen(1^{κ}) to obtain $(p, \mathbb{G}, \mathbb{G}_T, e, g)$.

```
2. Set s \stackrel{R}{\leftarrow} \mathbb{Z}_p.
```

3. Compute CRS
$$\leftarrow <(p, \mathbb{G}, \mathbb{G}_T, e, g), (g_0, g_1, ..., g_{\kappa}) > \text{where } \forall i \in [0..\kappa]: g_i = g^{s^i}.$$

- 4. Generate random $x, y \in \mathbb{Z}_p$ and set $u = g^x, v = g^y$ as the public key for the signature scheme.
- 5. Set $\theta_B \leftarrow \alpha ||\alpha'|$ where α' is a random bit sequence with $|\alpha| + |\alpha'| = \kappa$
- 6. Set $r_{\sigma} \stackrel{R}{\leftarrow} \mathbb{Z}_p$.
- 7. Set D = E.
- 8. Set $\tilde{\sigma} = (D^{\frac{1}{x+m_B+yr_{\sigma}}}, r_{\sigma}).$
- 9. Set $(r_1, r_2, ..., r_{\kappa}) \stackrel{R}{\leftarrow} \mathbb{Z}_p^{\kappa}$.
- 10. $\vec{C} = (C_i)_{i \in [\kappa]} = (g^{r_i} g_i^{\theta_B[i]})_{i \in [\kappa]}.$

These values are indistinguishable from a real execution of the protocol. If \mathcal{A} aborts the simulation (not only the protocol), that is he does not output his tentative signature σ_A then this means that α was not the sequence of the heading bits of the discrete logarithm of E. Thus the SHDL assumption is broken.

Type II: For adversary \mathcal{A} of type II, the simulator \mathcal{B} does the following: He asks \mathcal{A} to compute γ_A which is parsed as $(D_A, \vec{C}_A, \pi_{A,1}, \pi_{A,2}, PK_{\theta A}, P\vec{K}_A, \tilde{\sigma}_A)$. For the proofs of knowledge \mathcal{A} will use $\mathsf{H}(\cdot)$ a random oracle which is controlled by \mathcal{B} . So \mathcal{B} will rewind \mathcal{A} and obtain $PK_{\theta'A}$ from which he will obtain θ_A^* and $P\vec{K}_A'$ and will extract $(r_{Ai}, \theta_A[i])$ for each $i \in [\kappa]$. As \mathcal{A} lied, \mathcal{B} will be able to break the soundness of one of the NIZK argument or the binding property of the commitment scheme.