Concurrent Signatures from a Variety of Keys

George Teşeleanu[1,2]

[1] Advanced Technologies Institute
10 Dinu Vintilă, Bucharest, Romania
tgeorge@dcti.ro
[2] Simion Stoilow Institute of Mathematics of the Romanian Academy
21 Calea Grivitei, Bucharest, Romania

Abstract. Concurrent signatures allow two entities to produce two ambiguous signatures that become binding once an extra piece of information (called the keystone) is released. Such a signature is developed by Chen *et al.*, but it restricts signers to using the same public parameters. We describe and analyse a new concurrent signature that allows users to sign documents even if they use different underlying hard problems when generating their public parameters.

1 Introduction

The fair exchange of signatures between two mutually distrustful parties is a fundamental and well-studied problem in cryptography. Ideally, we would like the exchange of signatures to be done in fair way, *i.e.* each participant receives the other's signature, or neither does. We would also like to have some sort of guarantee that is impossible for one party to terminate the protocol and to leave the other participant committed when they are not.

To achieve a form of the properties mentioned above, several authors have put forth three main categories:

– Gradual release schemes: Using the idea of time release, the output is gradually revealed (*e.g.* bit per bit). Usually, this solution is highly interactive and may not work if the adversary is more computationally powerful [8, 10, 16].
– Optimistic schemes: Using a trusted third party, this approach can restore fairness if a dispute rises. In some cases, the infrastructure requirements and trusting a third party are not appropriate [2, 5, 14].
– Concurrent or legally fair schemes: The exchanged signatures become binding only when an extra piece of information (the keystone) is revealed. To enforce a signed contract, a participant has to present it in a court of law. Note that the keystone offers enough information to restore fairness. This approach does not require a trusted arbitrator or a high degree of interaction between parties [6, 7, 12].

Chen *et al.* [6] constructed their concurrent signature protocol based on a 1-out-of-n signature scheme proposed by Abe *et al.* [1]. An 1-out-of-n signature is constructed so that once a signature is computed, then any verifier is convinced that the signature was generated by one of n signers. Hence, using a slight modification of Abe *et al.* signature, Chen *et al.* are able to guarantee ambiguity before revealing the keystone.

In their paper, Abe *et al.* presented both a non-separable scheme where all n key pairs correspond to the same scheme, and a separable scheme where each key pair can be generated by a different scheme, under a different hardness assumption. For the discrete logarithm assumption, the authors of [1] also propose an non-separable schemes that is more efficient than the generic one. The concurrent signature proposed by Chen *et al.* was based on the efficient non-separable variant. Hence, it is based on the discrete logarithm assumption. Furthermore, the security of this protocol can be proven in the random oracle model, assuming the hardness of computing discrete logarithms in a cyclic group of prime order. Using a variation of Abe *et al.*'s 1-out-of-n signature with key separation, we introduce a concurrent signature in the separable model.

The efficient 1-out-of-n signature without key separation proposed in [1] is an adaptation of the Schnorr signature [17]. Maurer [13] introduced a construction that unifies the Schnorr zero-knowledge protocol [17] and the Guillou-Quisquater protocol [11]. A consequence of Maurer's construction is the introduction of other novel protocols whose security is based on other hardness assumptions[3]. Based on Maurer's approach, we describe a generic 1-out-of-n signature that can be seen as an adaptation of the signature described in [12]. Based on this signature we also generalize Chen *et al.*'s signature.

Note that in [1] the authors also describe a 1-out-of-n signature with key separation based on the full domain RSA signature scheme [4]. We chose to use only the zero-knowledge version, since working in a general framework[4] may reduce implementation errors, and save application development and maintenance time.

Remark that concurrent signatures are not abuse-free in the sense of [3, 9], since the party *Bob* who holds the keystone can always determine whether to complete the protocol or not. But, there are situations where it is not in *Bob*s interest to try and cheat *Alice*. One interesting scenario is that of fair tendering of contracts. Suppose *Alice* has a building contract that she wishes to put out to tender. Also, suppose that *Bob* and *Charlie* are two competing companies that put forward signed proposals to win the contract. If *Alice* whats to accept *Bob*'s offer, she returns a signed payment instruction to *Bob* and he in turn releases the keystone. A common form of abuse is to show *Charlie Bob*'s proposal and thus enabling *Charlie* to make a better offer. But, in the case of concurrent signatures, *Charlie* sees an ambiguous signature that might have been crafted by *Alice*. Hence, *Alice* gains no advantage in revealing *Bob*'s proposal.

Our Contributions. In their paper, Chen *et al.* [6] claim that their scheme can be extended to other ring signatures as long as the scheme is compatible to the keystone idea. Hence, different hard problems could be used to construct such

[3] different from the discrete logarithm and e^{th}-root assumptions
[4] Guillou-Quisquater's signature is also included in this framework.

schemes. Also, they claim that concurrent signatures that work in the separable model can be build using the techniques developed in [1]. Unfortunately, they do not provide such examples. Our aim is to fill this gap. Hence, the main contributions of our paper are the following:

– Adjusting the construction of Chen *et al.* to support signatures with separable keys. To achieve this, we first introduce a modification to Abe *et al.*'s separable 1-out-of-n signature.
– Generalizing the non-separable 1-out-of-n signature of Abe *et al.* to other hardness assumptions. We also implicitly prove the security of Abe *et al.*'s signature[5].
– Generalizing Chen *et al.*'s original concurrent signature to support additional hardness assumptions.

Structure of the paper. We introduce notations, definitions, schemes and protocols used throughout the paper in Section 2. We present a variation of Abe *et al.*'s signature scheme in Section 3. In Section 4 we present our main result, namely a concurrent signature in the separable model. We conclude in Section 5. In Appendices A and B we generalize the non-separable signature from [1] and the concurrent signature from [6].

2 Preliminaries

Notations. Throughout the paper, the notation $|S|$ denotes the cardinality of a set S. The action of selecting a random element x from a sample space X is denoted by $x \xleftarrow{\$} X$, while $x \leftarrow y$ represents the assignment of value y to variable x. The probability of the event E to happen is denoted by $Pr[E]$. The subset $\{0, \ldots, s-1\} \in \mathbb{N}$ is denoted by $[0, s)$. Note we further consider that all of $\mathcal{N}$'s subsets are of the form $[0, s)$ and have more than one element. A vector v of length n is denoted either $v = (v_0, \ldots, v_{n-1})$ or $v = \{v_i\}_{i \in [0,n)}$. Also, we use the notations C_k^n to denote binomial coefficients and exp to denote Euler's constant.

2.1 Groups

Let $(\mathbb{G}, \star)$ and $(\mathbb{H}, \otimes)$ be two groups. We assume that the group operations $\star$ and $\otimes$ are efficiently computable.

Let $f : \mathbb{G} \to \mathbb{H}$ be a function (not necessarily one-to-one). We say that f is a homomorphism if $f(x \star y) = f(x) \otimes f(y)$. Throughout the paper we consider f to be a one-way function, *i.e.* it is infeasible to compute x from $f(x)$. To be consistent with [13], we denote by $[x]$ the value $f(x)$. Note that given $[x]$ and $[y]$ we can efficiently compute $[x \star y] = [x] \otimes [y]$, due to the fact that f is a homomorphism.

[5] The original authors give an idea of how to prove that their signature is secure, but do not provide a concrete proof.

2.2 1-out-of-n Signatures

Definition 1 (1-out-of-n Signature). *An 1-out-of-n signature scheme is a digital signature comprised of the following algorithms*

Setup(λ): On input a security parameter λ, this algorithm outputs the private and public keys (sk_i, pk_i) of all the participants and the public parameters $pp = (\mathcal{M}, \mathcal{S})$, where $\mathcal{M}$ is the message space and $\mathcal{S}$ is the signature space.

Sign(m, sk_k, L): A PPT algorithm that on input a message $m \in \mathcal{M}$, the private key sk_k and a list of public keys L such that $pk_k \in L$, outputs a signature σ.

Verify(m, σ, L) An algorithm that on input a message m, a signature σ and a list of public keys L outputs either `true` *or* `false`*.*

We further present the security models presented in [1] for 1-out-of-n signature schemes.

Definition 2 (Signer Ambiguity). *Let $\mathcal{L} = \{pk_i\}_{i \in [0,n)}$, where pk_i are generated by the Setup algorithm. An 1-out-of-n signature is perfectly signer ambiguous if for any message m, any $L \subseteq \mathcal{L}$, any $sk_k \in L$ and any signature σ generated by the Sign(m, sk_k, L), any unbound adversary $\mathcal{A}$ outputs an sk such that $sk = sk_k$ with probability exactly $1/|L|$.*

Definition 3 (Existential Unforgeability against Adaptive Chosen Message and Chosen Public Key Attacks - EUF-CMCPA). *The notion of unforgeability for signatures is defined in terms of the following security game between the adversary $\mathcal{A}$ and a challenger:*

1. *The Setup algorithm is run and all the public parameters are provided to $\mathcal{A}$.*
2. *For any message and any subset of $\mathcal{L} = \{pk_i\}_{i \in [0,n)}$, $\mathcal{A}$ can perform signature queries to the challenger.*
3. *Finally, $\mathcal{A}$ outputs a signature (m, σ, L), where $L \subseteq \mathcal{L}$.*

$\mathcal{A}$ wins the game if Verify$(m, \sigma, L) =$ `true`*, $L \subseteq \mathcal{L}$ and $\mathcal{A}$ did not query the challenger on (m, L). We say that a signature scheme is unforgeable when the success probability of $\mathcal{A}$ in this game is negligible.*

Note that when $n = 1$ Definitions 1 and 3 are equivalent with the classical signature definition and, respectively, the existential unforgeability against adaptive chosen message attack.

We further introduce the notions of a Boolean matrix and of a heavy row in such a matrix [15]. These definitions are then used in stating the heavy row lemma [15].

Definition 4 (Boolean Matrix of Random Tapes). *Let us consider a matrix M whose rows consist of all possible random choices of an adversary and the columns consist of all possible random choices of a challenger. Its entries are 0 if the adversary fails the game and 1 otherwise.*

Definition 5 (Heavy Row). *A row of M is heavy if the fraction of 1's along the row is at least $\varepsilon/2$, where ε is the adversary's success probability.*

Lemma 1 (Heavy Row Lemma). *The 1's in M are located in heavy rows with a probability of at least $1/2$.*

2.3 Concurrent Signatures

Definition 6 (Concurrent Signature). *A concurrent signature scheme is a digital signature comprised of the following algorithms*

Setup(λ)*: On input a security parameter* λ*, this algorithm outputs the private and public keys* (x_i, y_i) *of all participants and the public parameters* $pp = (\mathcal{M}, \mathcal{S}, \mathcal{K}, \mathcal{F}, KeyGen)$*, where* $\mathcal{M}$ *is the message space,* $\mathcal{S}$ *is the signature space,* $\mathcal{K}$ *is the keystone space,* $\mathcal{F}$ *the keystone fix space and* $KeyGen : \mathcal{K} \to \mathcal{F}$ *is a function.*

aSign(y_i, y_j, x_i, f, m)*: On input the public keys* $y_i \neq y_j$*, the private key* x_i *corresponding to* y_i*, an element* $f \in \mathcal{F}$ *and a message* $m \in \mathcal{M}$*, this algorithm outputs an ambiguous signature* $\sigma = \langle s, e, f \rangle$*, where* $s \in \mathcal{S}$ *and* $e \in \mathcal{F}$*.*

aVerify(σ, y_i, y_j, m) *On input an ambiguous signature* $\sigma = \langle s, e, f \rangle$*, public keys* y_i*,* y_j *and a message* m *this algorithm outputs a boolean value.*

Verify(k, σ, y_i, y_j, m) *On input* $k \in \mathcal{K}$ *,* $\sigma = \langle s, e, f \rangle$*, public keys* y_i*,* y_j *and message* m*, this algorithm checks whether* $KeyGen(k) = f$ *and outputs* `false` *if not; otherwise it outputs the result of* $aVerify(\sigma, y_i, y_j, m)$*.*

Concurrent signatures are used by two parties *Alice* and *Bob* as depicted in Figure 1. At the end of the protocol, both $\langle k, \sigma_A \rangle$ and $\langle k, \sigma_B \rangle$ are binding, and accepted by the *Verify* algorithm.

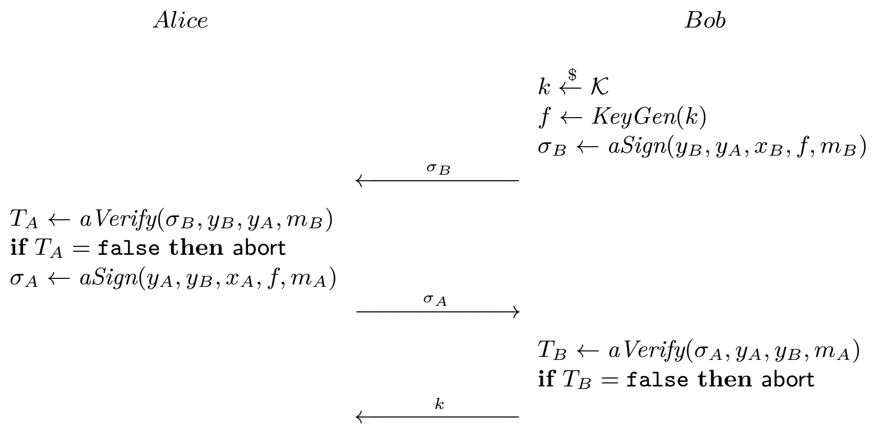
Alice
Bob
$k \xleftarrow{\$} \mathcal{K}$
$f \leftarrow KeyGen(k)$
$\sigma_B \leftarrow aSign(y_B, y_A, x_B, f, m_B)$
σ_B
$T_A \leftarrow aVerify(\sigma_B, y_B, y_A, m_B)$
if $T_A =$ `false` **then** abort
$\sigma_A \leftarrow aSign(y_A, y_B, x_A, f, m_A)$
σ_A
$T_B \leftarrow aVerify(\sigma_A, y_A, y_B, m_A)$
if $T_B =$ `false` **then** abort
k

Fig. 1. The concurrent signature of messages m_A and m_B.

According to the security notions presented in [6], a PPT adversary $\mathcal{A}$ for a concurrent signature can perform the following queries

- *KeyGen* queries: $\mathcal{A}$ can request the value $f \leftarrow KeyGen(k)$, where k is selected by the challenger $\mathcal{T}$. If $\mathcal{A}$ whants to choose his own k, he can compute $KeyGen(k)$ by himself.

- *KeyReveal* queries: $\mathcal{A}$ can requests $\mathcal{T}$ to reveal the keystone k associated to f. If f was not previously computed by the challenger, then $\mathcal{T}$ outputs invalid, otherwise he returns k.
- *aSign* queries: $\mathcal{A}$ can request a valid *aSign* signature σ for two public keys $y_i \neq y_j$, an element $f \in \mathcal{F}$ and a message m of his choosing. Note that using *aSign* queries in conjunction with *KeyGen* queries, the adversary can obtain concurrent signatures for messages and pairs of users of his choice.
- *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.

The following definition captures the notion of unforgeability in the concurrent context.

Definition 7 (Concurrent Signature Unforgeability - EUF-CS). *The notion of unforgeability for concurrent signatures is defined in terms of the following security game between the adversary $\mathcal{A}$ and a challenger $\mathcal{T}$:*

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

$\mathcal{A}$ wins the game if aVerify$(s, e, f, y_C, y_D, m) =$ `true` 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 either (y_C, y_D, f, m) or (y_D, y_C, f, m).*
- *$\mathcal{A}$ did not query SKExtract on y_D, and did not query aSign on (y_D, y_i, f, m) for any $y_i \neq y_D$ and $\mathcal{A}$ produces a keystone k such that KeyGen$(k) = f$.*
- *$\mathcal{A}$ did not query SKExtract on y_C, and did not query aSign on (y_C, y_i, f, m) for any $y_i \neq y_C$ and f was a previous output from a KeyGen query.*

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

Note that in Definition 7 the first output condition corresponds to an outside attacker that tries to create a forgery without knowing the secret keys of the participants. Hence, in this case *Alice* is convinced that the signature originates from *Bob*. The second and third conditions correspond to the case where the attacker and one of the participants are one and the same.

The next definition captures the notion of ambiguity for concurrent signatures. Note that the security notion is slightly weaker than Definition 2 due to the fact f is generated by *KeyGen* that in practice approximates as best as possible a random oracle.

Definition 8 (Concurrent Signature Ambiguity). *The notion of ambiguity for concurrent signatures is defined in terms of the following security game between the adversary $\mathcal{A}$ and a challenger $\mathcal{T}$:*

1. *The Setup algorithm is run and all the public parameters are provided to $\mathcal{A}$.*
2. *$\mathcal{A}$ can perform any number of queries to the challenger, as described above.*
3. *$\mathcal{A}$ selects a message m and two public keys y_C and y_D.*
4. *In response, the challenger randomly computes $\sigma \leftarrow aSign(y_C, y_D, x_C, f, m)$ or $\sigma \leftarrow aSign(y_D, y_C, x_D, f, m)$, where $k \overset{\$}{\leftarrow} \mathcal{K}$ and $f \leftarrow KeyGen(k)$, and sends σ to $\mathcal{A}$.*
5. *Finally, $\mathcal{A}$ guesses $\mathcal{T}$'s choice.*

A concurrent signature is signer ambiguous if $\mathcal{A}$ cannot guess $\mathcal{T}$'s choice with a probability non-negligible greater than $1/2$.

The following definition captures the intuitive notion of fairness. More precisely, that the person that generated the keystone is the only one that can use it to create a binding signature and that any ambiguous signature produced using the same keystone fix f will all become binding. Note that the definition does not guarantee that *Alice* will receive the necessary keystone k.

Definition 9 (Fairness). *The notion of fairness for concurrent signatures is defined in terms of the following security game between the adversary $\mathcal{A}$ and a challenger $\mathcal{T}$:*

1. *The Setup algorithm is run and all the public parameters are provided to $\mathcal{A}$.*
2. *$\mathcal{A}$ can perform any number of queries to the challenger, as described above.*
3. *$\mathcal{A}$ chooses two public keys y_C and y_D and outputs a tuple (m, y_C, y_D, σ, k), where $\sigma = \langle s, e, f \rangle$ and $f = KeyGen(k)$.*

The adversary wins the game if $aVerify(s, e, f, y_C, y_D, m) =$ `true` and either of the following holds

– *f was a previous output from a KeyGen query, $\mathcal{A}$ did not query KeyReveal on f and (k, σ) is accepted by the Verify algorithm.*
– *$\mathcal{A}$ also outputs a tuple (m, y_D, y_C, σ'), where $\sigma' = \langle s', e', f \rangle$, such that aVerify accepts σ' and Verify accepts (k, σ), but (k, σ') is not accepted by Verify.*

We say that a concurrent signature scheme is fair when the success probability of $\mathcal{A}$ in this game is negligible.

3 1-out-of-n Signatures with Key Separation

3.1 Description

We present a variation of the 1-out-of-n signature scheme presented in [1]. This variation will be used later to develop a concurrent signature protocol that allows users with different flavors of public keys (*i.e.* discrete logarithm based, e^{th} root problem based) to produce two binding and ambiguous signatures. In practice, each user can generate their own public parameters and key pair. To simplify description we present the *Setup* algorithm as a centralized algorithm. We will denote the following signature with $1n$-KSS.

Setup(λ): Let $i \in [0, n)$. Choose for each user two groups $\mathbb{G}_i$, $\mathbb{H}_i$, a homomorphism $[\cdot]_i : \mathbb{G}_i \to \mathbb{H}_i$ and a hash function $H_i : \{0,1\}^* \to \mathcal{C} \subseteq \mathbb{N}$. Note that we require that $|\mathbb{G}_i| \geq 2^\lambda$. Choose $x_i \overset{\$}{\leftarrow} \mathbb{G}_i$ and compute $y_i \leftarrow [x_i]_i$. Output the public key $pk_i = y_i$. The secret key is $sk_i = x_i$.

Listing(): Collect the public keys and randomly shuffle them. Store the result into a list $\mathcal{L} = \{y_j\}_{j \in [0,n)}$ and output $\mathcal{L}$.

Sign($m, sk_k, \mathcal{L}$): To sign a message $m \in \{0,1\}^*$, first generate two random elements $\alpha \overset{\$}{\leftarrow} \mathbb{G}_k$, $\beta \overset{\$}{\leftarrow} \mathcal{C}$ and compute $c_{k+1} \leftarrow H_{k+1}(\mathcal{L}, m, [\alpha]_k)$ and $c'_{k+1} \leftarrow c_{k+1} - \beta \bmod c$, where $|\mathcal{C}| = c$. For $j \in [k+1, n) \cup [0, k)$, select $s_j \overset{\$}{\leftarrow} \mathbb{G}_j$ and then compute $c_{j+1} \leftarrow H_{j+1}(\mathcal{L}, m, [s_j]_j \otimes_j y_j^{c'_j})$ and $c'_{j+1} \leftarrow c_{j+1} - \beta \bmod c$. Compute $s_k \leftarrow \alpha \star_k x_k^{-c'_k}$. Output the signature $(c_0, \beta, \mathcal{S})$, where $\mathcal{S} = \{s_j\}_{j \in [0,n)}$.

Verify($m, c_0, \beta, \mathcal{S}, \mathcal{L}$): For $j \in [0, n)$, compute $e_j \leftarrow [s_j]_j \otimes_j y_j^{c_j - \beta}$ and then $c_{j+1} \leftarrow H_{j+1}(\mathcal{L}, m, e_j)$ if $j \neq n-1$. Output `true` if and only if $c_0 = H_0(\mathcal{L}, m, e_{n-1})$. Otherwise, output `false`.

Correctness. If the pair $(c_0, \beta, \mathcal{S})$ is generated according to the scheme, it is easy to see that the c_j values from the *Sign* and *Verification* coincide when when $j \neq k$. When $j = k$ we observe that

$$\begin{aligned} e_k &= [s_k]_k \otimes_k y_k^{c_k - \beta} = [\alpha \star_k x_k^{-c_k + \beta}]_k \otimes_k y_k^{c_k - \beta} \\ &= [\alpha]_k \otimes_k [x_k]_k^{-c_k + \beta} \otimes_k y_k^{c_k - \beta} = [\alpha]_k \end{aligned}$$

and thus we obtain the same c_k as in the signing phase.

Remark 1. In practice hash functions have $\mathcal{C} = \{0,1\}^\kappa$, where κ is for example 256, 384 or 512. So, if the users from $\mathcal{L}$ have hash functions with different output sizes the simplest method for obtaining a common challenge space $\mathcal{C}$ is to lengthen the function's output by running it several times[6] until we obtain c bits. If efficiency is desired, another method for obtaining a common $\mathcal{C}$ is to truncate all the outputs to the smallest size. Note this method decreases security for some users.

3.2 Security Analysis

Theorem 1. *The* 1n*-KSS scheme is perfectly signer ambiguous.*

Proof. Note that all s_j are taken randomly from $\mathbb{G}_j$, except for s_k. Since α is a random element from $\mathbb{G}_k$, then s_k is also randomly distributed in $\mathbb{G}_k$. Also, β is a random element from $\mathcal{C}$. Hence, for a fixed $(m, \mathcal{L})$ the probability of $(\beta, \mathcal{S})$ is always $1/(|\mathcal{C}| \cdot \prod |G_i|)$, regardless of the closing point s_k. The remaining c_0 is uniquely determined from $(m, \mathcal{L})$ and $(\beta, \mathcal{S})$. □

[6] *e.g.* for each run we can add a different prefix to the message

Theorem 2. *If the following statements are true*

– *an* EUF-CMCPA *attack on the* $1n$*-KSS has non-negligible probability of success in the ROM,*
– *an* $\ell \in \mathbb{Z}$ *is known such that* $\gcd(c_0 - c_1, \ell) = 1$ *for all* $c_0, c_1 \in \mathcal{C}$ *with* $c_0 \neq c_1$*,*
– *for all* i *values,* $u_i \in \mathbb{G}_i$ *are known such that* $[u_i]_i = y_i^\ell$*,*

then at least a homomorphism $[\cdot]_i$ *can be inverted in polynomial time.*

Proof. Let $\mathcal{A}$ be an efficient EUF-CMCPA attacker for $1n$-KSS that requests at most q_s and q_h signing and, respectively, random oracle queries. Also, let ε be its success probability and τ its running time.

In order to make $\mathcal{A}$ work properly we simulate the random oracles that correspond to each hash function (see Algorithm 1) and the signing oracle (see Algorithm 2). For simplicity we treat all the random oracles as one big random oracle $\mathcal{O}_H$ that takes as input the j-th query (i, L_j, m_j, r_j) and returns a random value corresponding to $H_i(L_j, m_j, r_j)$. Also, to avoid complicated suffixes y_0, for example, refers to the first public key from the current L_j.

Algorithm 1: Hashing oracle $\mathcal{O}_H$ simulation for all H_i.

```
   Input: A hashing query $(i, L_j, m_j, r_j)$ from $\mathcal{A}$
1  if $\exists h_j$ such that $\{L_j, m_j, r_j, h_j\} \in T_i$ then
2      $e \leftarrow h_j$
3  else
4      $e \xleftarrow{\$} \mathcal{C}$
5      Append $\{L_j, m_j, r_j, e\}$ to $T_i$
6  end if
7  return $e$
```

The signing oracle $\mathcal{O}_S$ fails and returns $\perp$ only if we cannot assign c_0 to $(L_j, m_j, e_{|L_j|-1})$ without causing an inconsistency in T_0. This event happens with probability at most q_h/q, where $q = 2^\lambda$. Thus, $\mathcal{O}_S$ is successful with probability at least $(1 - q_h/q)^{q_s} \geq 1 - q_h q_s/q$.

Let Θ and Ω be the random tapes given to $\mathcal{O}_S$ and $\mathcal{A}$. The adversary's success probability is taken over the space defined by Θ, Ω and $\mathcal{O}_H$. Let Σ be the set of $(\Theta, \Omega, \mathcal{O}_H)$ with which $\mathcal{A}$ successfully creates a forgery, while having access to a real signing oracle. Let $(m, c_0, \beta, \{s_i\}_{i\in[0,n')}, L)$ be $\mathcal{A}$'s forgery, where $|L| = n'$. Then T_{i+1} contains a query for (L, m, e_i) for all $i \in [0, n')$ with probability at least $1 - 1/c$, due to the ideal randomness of $\mathcal{O}_H$. Let $\Sigma' \subseteq \Sigma$ be the set of $(\Theta, \Omega, \mathcal{O}_H)$ with which $\mathcal{A}$ successfully creates a forgery, while having access only to the simulated oracle $\mathcal{O}_S$. Then, $Pr[(\Theta, \Omega, \mathcal{O}_H) \in \Sigma'] \geq \varepsilon'$, where $\varepsilon' = (1 - q_h q_s/q)(1 - 1/c)\varepsilon$.

Since the queries form a ring, there exists at least an index $k \in [0, n')$ such that the u query $Q_u = (k+1, L, m, e_k)$ and the v query $Q_v = (k, L, m, e_{k-1})$

Algorithm 2: Signing oracle $\mathcal{O}_S$ simulation.

```
Input: A signature query $(m_j, L_j)$ from $\mathcal{A}$
$c_0, \beta \xleftarrow{\$} \mathcal{C}$
for $i \in [0, |L_j|)$ do
    $s_i \xleftarrow{\$} \mathbb{G}_i$
    $e_i \leftarrow [s_i]_i \otimes_i y_i^{c_i - \beta}$
    if $i \neq |L_j| - 1$ then
        $c_{i+1} \leftarrow H_{i+1}(L_j, m_j, e_i)$
    end if
end for
if $\nexists h$ such that $\{L_j, m_j, e_{|L_j|-1}, h\} \in T_0$ then
    Append $\{L_j, m_j, e_{|L_j|-1}, c_0\}$ to $T_0$
    return $(c_0, \beta, \{s_i\}_{i \in [0,|L_j|)})$
else
    return $\perp$
end if
```

satisfy $u \leq v$. Such a pair (u, v) is called a gap index. Remark that $u = v$ only when $n' = 1$. If there are two or more gap indices with regard to a signature, we only consider the smallest one.

We denote by $\Sigma'_{u,v}$ the set of $(\Theta, \Omega, \mathcal{O}_H)$ that yield the gap index (u, v). There are at most $C_2^{q_h} + C_1^{q_h} = q_h(q_h + 1)/2$ such sets. If we invoke $\mathcal{A}$ with randomly chosen $(\Theta, \Omega, \mathcal{O}_H)$ at most $1/\varepsilon'$ times, then we will find at least one $(\Theta, \Omega, \mathcal{O}_H) \in \Sigma'_{u,v}$ for some gap index (u, v) with probability $1 - (1 - \varepsilon')^{1/\varepsilon'} > 1 - exp(-1) > 3/5$.

We define the sets $GI = \{(u, v) \mid |\Sigma'_{u,v}|/|\Sigma'| \geq 1/(q_h(q_h + 1))\}$ and $B = \{(\Theta, \Omega, \mathcal{O}_H) \in \Sigma'_{u,v} \mid (u, v) \in GI\}$. Then, we have $Pr[B|\Sigma'] \geq 1/2$. Using the heavy row lemma we obtain that a triplet $(\Theta, \Omega, \mathcal{O}_H)$ that yields a successful run of $\mathcal{A}$ is in B with probability at least $1/2$.

Let $\mathcal{O}_{H'}$ be the identical to $\mathcal{O}_H$ except for the Q_v query to which $\mathcal{O}_{H'}$ responds with a random element $c'_k \neq c_k$. Then according to the heavy row lemma, with probability $1/2$, $(\Theta, \Omega, \mathcal{O}_{H'})$ satisfies $Pr[(\Theta, \Omega, \mathcal{O}_{H'}) \in \Sigma'_{u,v}] = \varepsilon''/2$, where $\varepsilon'' = \varepsilon'/(2q_h(q_h + 1))$. Hence, if we run $\mathcal{A}$ at most $2/\varepsilon''$ times, then with probability $1/2 \cdot [1 - (1 - \varepsilon''/2)^{2/\varepsilon''}] > 1/2 \cdot (1 - exp(-1)) > 3/10$ we will find at least one c'_k such that $(\Theta, \Omega, \mathcal{O}_{H'}) \in \Sigma'_{u,v}$. Since Q_u is queried before Q_v, e_k remains unchanged. Therefore we can compute

$$\tilde{x}_k = u_k^a \star_k ({s'_k}^{-1} \star_k s_k)^b,$$

where a and b are computed using Euclid's algorithm such that $\ell a + (c'_k - c_k)b = 1$. Note that

$$\begin{aligned}[s_k'^{-1} \star_k s_k]_k &= [s_k'^{-1}]_k \otimes_k [s_k]_k \\ &= y_k^{c'_k - \beta} \otimes_k ([\alpha]_k)^{-1} \otimes_k [\alpha]_k \otimes_k y_k^{-c_k + \beta} \\ &= y_k^{c'_k - c_k}\end{aligned}$$

and thus

$$\begin{aligned}[\tilde{x}_k]_k &= [u_k^a \star_k (s_k'^{-1} \star_k s_k)^b]_k \\ &= ([u_k]_k)^a \otimes_k ([s_k'^{-1} \star_k s_k]_k)^b \\ &= (y_k^\ell)^a \otimes_k (y_k^{c'_k - c_k})^b \\ &= y_k.\end{aligned}$$

The overall success probability is $9/100 = 3/5 \cdot 1/2 \cdot 3/10$ and $\mathcal{A}$ is invoked at most $1/\varepsilon' + 2/\varepsilon''$ times. $\square$

3.3 Concrete Examples

All Discrete Logarithm Case. Let $p = 2q + 1$ be a prime number such that q is also prime. Select an element $h \in \mathbb{H}_p$ of order q in some multiplicative group of order $p - 1$. The discrete logarithm of an element $z \in \mathbb{H}_p$ is an exponent x such that $z = h^x$. We further describe the parameters of the all discrete logarithm signature.

Define $(\mathbb{G}_i, \star_i) = (\mathbb{Z}_{q_i}, +)$ and $\mathbb{H}_i = \langle h_i \rangle$. The one-way group homomorphism is defined by $[x_i]_i = h_i^{x_i}$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, q)$, where q is the smallest q_i from $\mathcal{L}$. Let 1_i be the neutral element of $\mathbb{H}_i$. Then the conditions of Theorem 2 are satisfied for

- $\ell = \prod_{i=0}^{n-1} q_i$, since for all $c \in \mathcal{C}$ we have $c < q \leq q_i$ and q_i are primes,
- for $u = 0$ we have $[u]_i = [0]_i = 1_i = y_i^\ell = (y_i^{\ell/q_i})^{q_i}$ since every element of $\mathbb{H}_i$ raised to the group order q_i is the neutral element 1_i.

All e^{th}-root Case. Let p and q be two safe prime numbers such that $(p-1)/2$ and $(q-1)/2$ are also prime. Compute $N = pq$ and choose a prime e such that $\gcd(e, \varphi(N)) = 1$. An e^{th}-root of an element $z \in \mathbb{Z}_N^*$ is a base x such that $z = x^e$. Note that the e^{th}-root is not unique. We further describe the parameters of the all e^{th}-root signature.

Define $(\mathbb{G}_i, \star_i) = (\mathbb{H}_i, \otimes_i) = (\mathbb{Z}_{N_i}^*, \cdot)$, where $N_i = p_i q_i$ and $\gcd(N_i, N_j) = 1$ for $i \neq j$. The one-way group homomorphism is defined by $[x_i]_i = x_i^{e_i}$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, e)$, where e is the smallest e_i in $\mathcal{L}$. The conditions of Theorem 2 are satisfied for

- $\ell = \prod_{i=0}^{n-1} e_i$, since for all $c \in \mathcal{C}$ we have $c < e \leq e_i$ and e_i are primes,
- for $u_i = y_i^{\ell/e_i}$ we have $[u_i]_i = [y_i^{\ell/e_i}]_i = y_i^\ell$.

Mixture of Discrete Logarithm and e^{th}-root. For simplicity, we consider the case $n = 2$. Let $(\mathbb{G}_0, \star_0) = (\mathbb{Z}_q, +)$, $\mathbb{H}_0 = \langle h \rangle$ and $(\mathbb{G}_1, \star_1) = (\mathbb{H}_1, \otimes_1) = (\mathbb{Z}_N^*, \cdot)$. The one-way group homomorphisms are defined by $[x_0]_0 = h^{x_0}$ and $[x_1]_1 = x_1^e$. The challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, s)$, where s is the smallest of q and e. The conditions of Theorem 2 are satisfied for

- $\ell = eq$, since for all $c \in \mathcal{C}$ we have $c < s \leq e$, $c < s \leq q$ and e, q are primes,
- for $u_0 = 0$ we have $[0]_0 = 1 = (y_0^e)^q$,
- for $u_1 = y_1^q$ we have $[y_1^q]_1 = y_1^\ell$.

All Discrete Logarithm Representation Case. Consider again a group of prime order $\mathbb{H}_p$ and select t elements $h_1, \ldots, h_t \in \mathbb{H}_p$ of order q. A discrete logarithm representation of an element $z \in \langle h_1, \ldots, h_t \rangle$ is a list of exponents $(x_1, \ldots, x_t)$ such that $z = h_1^{x_1} \ldots h_t^{x_t}$. Note that discrete logarithm representations are not unique. We further describe the parameters of the all discrete logarithm representation signature.

We define $\mathbb{G}_i = \mathbb{Z}_{q_i}^{t_i}$ with $\star$ defined as addition applied component-wise and $\mathbb{H}_i = \langle h_{i1}, \ldots, h_{it} \rangle$. Let $x_i = (x_{i1}, \ldots, x_{it})$. The one-way group homomorphism is defined by $[x_i]_i = h_{i1}^{x_{i1}} \ldots h_{it}^{x_{it}}$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, q]$, where q is the smallest q_i from $\mathcal{L}$. Let 1_i be the neutral element of $\mathbb{H}_i$. Then the conditions of Theorem 2 are satisfied for $\ell = \prod_{i=0}^{n-1} q_i$ and for $u = (0, \ldots, 0)$.

Remark that if some t_is are one, we obtain a signature based on mixture of discrete logarithm and discrete logarithm representation problems.

All e^{th}-root Representation Case. Let again $N = pq$ and choose primes $e_1, \ldots, e_t$ such that $\gcd(e_i, \varphi(N)) = 1$, for $1 \leq i \leq t$. An e^{th}-root representation of an element $z \in \mathbb{Z}_N^*$ is a list of bases $(x_1, \ldots, x_t)$ such that $z = x_1^{e_1} \ldots x_t^{e_t}$. Note that e^{th}-root representations are not unique. We further describe the parameters of the all e^{th}-root representation signature.

Let $N_i = p_i q_i$ and $\gcd(N_i, N_j) = 1$ for $i \neq j$. We define $\mathbb{G}_i = (\mathbb{Z}_{N_i}^*)^{t_i}$ with $\star_i$ defined as multiplication applied component-wise and $(\mathbb{H}_i, \otimes_i) = (\mathbb{Z}_{N_i}^*, \cdot)$. The one-way group homomorphism is defined by $[(x_{i1}, \ldots, x_{it})] = x_{i1}^{e_{i1}} \ldots x_{it}^{e_{it}}$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, e)$, where e is a prime such that $\gcd(e, \phi(N_i)) = 1$. Since all exponents are coprime then we can compute integers such that $\alpha_{i1} e_{i1} + \ldots + \alpha_{it} e_{it} = 1$. The conditions of Theorem 2 are satisfied for

- $\ell = 1$,
- for $u_i = (y_i^{\alpha_{i1}}, \ldots, y_i^{\alpha_{im}})$ we have $[u_i]_i = y_i^{\alpha_{i1} e_{i1} + \ldots + \alpha_{it} e_{it}} = y_i$.

4 Concurrent Signatures with Key Separation

4.1 Description

Concurrent signatures allow *Alice* and *Bob* to produce two signatures such that both signatures are ambiguous from the eyes of a third party, but once Alice releases a secret keystone, both signatures become binding to their true signer. Such signatures are useful for contract signing and fair exchange protocols. Based on $1n$-KSS we introduce such a concurrent signature scheme denoted with $1n$-KSCS. Note that when both users use the same group for defining their underlying homomorphisms a more efficient construction is presented in Appendix B.

As before, $\mathcal{C}$ denotes the challenge space and c its cardinality. The $1n$-KSCS scheme uses three cryptographic hash functions $H_k, H_A, H_B : \{0,1\}^* \to \mathcal{C}$. The detailed protocol is presented in Figure 2

Correctness. If the signature $\langle s_A, e_A, f \rangle$ is generated according to the scheme, it is easy to see that

$$[v_A]_A \otimes_A y_B^{h_A} = [u]_A \otimes_A [x_A]_A^{-g_A+f} \otimes y_A^{g_A-f} = [u]_A.$$

Similarly, we can show correctness for *Bob*'s side.

4.2 Security Analysis

The following theorem is a direct consequence of Theorem 1.

Theorem 3. *The $1n$-KSCS scheme satisfies the concurrent signature ambiguity property in the ROM.*

Theorem 4. *If the following statements are true*

- *an* EUF-CS *attack on the $1n$-KSCS has non-negligible probability of success in the ROM,*
- *an $\ell \in \mathbb{Z}$ is known such that $\gcd(c_0 - c_1, \ell) = 1$ for all $c_0, c_1 \in \mathcal{C}$ with $c_0 \neq c_1$,*
- *for $i \in \{A, B\}$, $u_i \in \mathbb{G}_i$ are known such that $[u_i]_i = y_i^{\ell}$,*

then either $[\cdot]_A$ or $[\cdot]_B$ can be inverted in polynomial time.

Proof. Let $\mathcal{A}$ be an efficient EUF-CS attacker for $1n$-KSCS and let ε be its success probability. We split the proof into three cases: $\mathcal{A}$ does not have access to the participants' secret keys, $\mathcal{A} = Bob$ and $\mathcal{A} = Alice$.

First case. The challenger generates a set of participants U, where $|U| = \rho$ and ρ is the result of a polynomial function in λ. Then the challenger chooses $\gamma_A \neq \gamma_B \xleftarrow{\$} [0, \rho)$. For each participant P_i, $i \neq \gamma_a, \gamma_B$, $\mathcal{T}$ selects the associated public parameters (in accordance to the security parameter λ) and generates their secret and public keys (x_i, y_i). For $C = P_{\gamma_A}$ the challenger sets the public parameters to $(\mathbb{G}_A, [\cdot]_A, H_A)$ and the public key $y_{\gamma_A} = y_A$. Similarly for we set $D = P_{\gamma_B}$'s parameters.

To make $\mathcal{A}$ work properly we must simulate all the oracles which $\mathcal{A}$ can query. Hence, the random oracles H_A and H_B can be simulated using Algorithm 1, where we set $\mathcal{L} = \emptyset$, $i = 0$ for A and $i = 1$ for B. We change the list notations from T_0 and T_1 to T_A and T_B. In the case of H_k, the simulation is similar to Algorithm 1. Thus, instead of querying (i, L_j, m_j, r_j), the adversary can query

```
Alice                                                    Bob

                                                         $k \xleftarrow{\$} \{0,1\}^*$
                                                         %KeyGen
                                                         $f \leftarrow H_k(k)$
                                                         %aSign
                                                         $t \xleftarrow{\$} \mathbb{G}_B, s_A \xleftarrow{\$} \mathbb{G}_A$
                                                         $f_A \leftarrow H_B(m_B, [t]_B)$
                                                         $e_A \leftarrow f_A - f \bmod c$
                                                         $f_B \leftarrow H_A(m_B, [s_A]_A \otimes_A y_A^{e_A})$
                                                         $e_B \leftarrow f_B - f \bmod c$
                                                         $s_B \leftarrow t \star_B x_B^{-e_B}$
                                                         $\sigma_B \leftarrow \langle s_A, s_B, e_A, f \rangle$
                          $\xleftarrow{\sigma_B}$
%aVerify
$T_A \leftarrow H_A(m_B, [s_A]_A \otimes_A y_A^{e_A})$
$S \leftarrow T_A - f \bmod c$
$T_B \leftarrow H_B\left(m_B, [s_B]_B \otimes_B y_B^{S}\right)$
if $T_B \neq e_A + f \bmod c$ then abort
%aSign
$u \xleftarrow{\$} \mathbb{G}_A, v_B \xleftarrow{\$} \mathbb{G}_B$
$g_B \leftarrow H_A(m_A, [u]_A)$
$h_B \leftarrow g_B - f \bmod c$
$g_A \leftarrow H_B\left(m_A, [v_B]_B \otimes_B y_B^{h_b}\right)$
$h_A \leftarrow g_A - f \bmod c$
$v_A \leftarrow u \star_A x_A^{-h_A}$
$\sigma_A \leftarrow \langle v_A, v_B, h_A, f \rangle$
                          $\xrightarrow{\sigma_A}$
                                                         %aVerify
                                                         $W_A \leftarrow H_A\left(m_A, [v_A]_A \otimes_A y_A^{h_A}\right)$
                                                         $Z \leftarrow W_A - f \bmod c$
                                                         $W_B \leftarrow H_B\left(m_B, [v_B]_B \otimes_B y_B^{Z}\right)$
                                                         if $W_B \neq h_A + f \bmod c$ then abort
                          $\xleftarrow{k}$
```

Fig. 2. Key separation concurrent signature.

any message M and the algorithm will store its answers in list denoted T_k. When $\mathcal{A}$ makes a *KeyGen* query, $\mathcal{T}$ randomly generates a k and return $f \leftarrow H_k(k)$. Note that the *KeyGen* oracle is actually a sublist of T_k, but the challenger is required to answer *KeyReveal* queries. Hence, when $\mathcal{A}$ requests the keystone associated to an $f \in \mathcal{C}$, we search T_k for a pair $\{k, f\}$ and if it exists we return k, otherwise we return invalid. The simulation of *aSign* queries can be achieved using Algorithm 2 where β is not chosen randomly, but is set to f. Finally, when an *SKExtract* query for a public key is made, $\mathcal{T}$ respond with the associated secret key, except in the case y_C, y_D, when he aborts.

There are two possible situations where our simulation fails. When $\mathcal{O}_S$ causes inconsistencies in $\mathcal{O}_H$ or $\mathcal{A}$ asks the secret keys for user C or user D. The first event does not happen with probability $1 - q_h q_s/q$, where $q = 2^\lambda$, and q_s and q_h are the number of signing queries and random oracle queries to H_A and H_B. The probability for the second event not happening is $1 - 2/\rho$. Let $\varepsilon' = 2/\rho(1 - q_h q_s/q)(1 - 2/\rho)(1 - 1/c)\varepsilon$. Then, if we run $\mathcal{A}$ at most $1/\varepsilon'$ with probability $3/5$ $\mathcal{A}$ will output a forgery $\sigma = \langle s_C, s_D, e, f\rangle$, for a message m.

Note that in this case $\mathcal{A}$ did not make *SKExtract* queries for C and D, and the signature is not produced by an *aSign* query. In other words $\mathcal{A}$ breaks the unforgeability of the $1n$-KSS scheme. Hence, according to Theorem 1 $\mathcal{A}$ inverted either $[\cdot]_A$ or $[\cdot]_B$.

Second case. Now, let us see what happens when $\mathcal{A}$ plays the role of *Alice*. In contrast with the first case, the challenger only chooses $\gamma_B \overset{\$}{\leftarrow} [0, \rho)$ and then sets D's public parameters to $(\mathbb{G}_B, [\cdot]_B, H_B)$ and the public key $y_{\gamma_B} = y_B$.

The probability of $\mathcal{A}$ not asking the secret key for user D is $1 - 1/\rho$. Let $\varepsilon'' = 1/\rho(1 - q_h q_s/q)(1 - 1/\rho)(1 - 1/c)\varepsilon$. Then, if we run $\mathcal{A}$ at most $1/\varepsilon''$ with probability $3/5$ $\mathcal{A}$ will output a forgery $\sigma = \langle s_A, s_D, e_A, f\rangle$, for a message m. According to the heavy row lemma with probability $1/2$ we are on situated on a heavy row $\mathcal{H}$.

Let $T_A \leftarrow H_A(m, [s_A]_A \otimes_A y_A^{e_A})$. Define $\mathcal{O}_{H'}$ as a random oracle identical to $\mathcal{O}_H$ except for the $(0, m, [s_A]_A \otimes_A y_A^{e_A})$ query to which $\mathcal{O}_{H'}$ responds with a random element $T'_A \neq T_A$. We restart $\mathcal{A}$ at most $2/\varepsilon'$ and with a probability of $1/2 \cdot (1 - (1 - \varepsilon'/2)^{2/\varepsilon'}) > 3/10$ we will be situated on $\mathcal{H}$. Hence, we obtain a new forgery $\sigma' = \langle s'_A, s'_D, e'_A, f'\rangle$.

Note that $T_A = e_A + f \neq e'_A + f' = T'_A$. If $e_A = e'_A$ then $f \neq f'$, so these values must have been computed before the relevant H queries and satisfy $f = T_A - e_A$ and $f' = T'_A - e'_A$. However, f is also an output of H_K and the probability that an output from some H_K query and some H query matches is at most $q_h q_k/q$, where q_k is the number of random oracle queries to H_K. Hence, with probability $1 - q_h q_k/q$ we have $f = f'$ and $e_A \neq e'_A$. In this case, using techniques similar to Theorem 2's proof we manage to find a preimage of $[\cdot]_B$.

Third case. The proof is similar to the second case and thus is omitted. □

Theorem 5. *The $1n$-KSCS scheme is fair in the ROM.*

Proof. The challenger generates a set of participants U, where $|U| = \rho$ and ρ is a polynomial function in λ. For each participant $\mathcal{T}$ selects the associated public parameters (in accordance to the security parameter λ) and generates their secret and public keys (x_i, y_i). We simulate the adversary's random oracles, and the *KeyGen* and *KeyReveal* algorithms as in Theorem 4's proof. Also, the challenger responds to *aSign* and *SKExtract* queries using its knowledge of the private keys. In the final stage of the fairness game, $\mathcal{A}$ outputs a signature $\sigma = \langle s, e, f\rangle$.

In the first case f was obtained by a *KeyGen* query, but no *KeyReveal* query was made for f. Since H_K is a random oracle, this event happens with probability q_k/q. Thus, is negligible.

In the second case (k, σ) is accepted by the *Verify* algorithm and $\mathcal{A}$ manages to produce a second signature $\sigma' = \langle s', e', f\rangle$ that is accepted by the *aVerify* algorithm, but (k, σ') is rejected by the *Verify* algorithm. Since, (k, σ) is accepted by *Verify*, we have $f = KeyGen(k)$. Since σ and σ' share the value f, we must have that (k, σ') is also accepted by the *Verify* algorithm. This is a contradiction. □

5 Conclusion

Our concurrent signature protocol is the abstraction of a large class of protocols that allow users with independently selected underlying problems to commonly produce an ambiguous signature. We have managed to relate the presented protocol's security to the hardness of inverting one-way homomorphisms. Note that the presented list of homomorphisms examples is by no means exhaustive.

References

1. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n Signatures from a Variety of Keys. In: ASIACRYPT 2002. Lecture Notes in Computer Science, vol. 2501, pp. 415–432. Springer (2002)
2. Asokan, N., Schunter, M., Waidner, M.: Optimistic Protocols for Fair Exchange. In: CCS 1997. pp. 7–17. ACM (1997)
3. Baum-Waidner, B., Waidner, M.: Round-Optimal and Abuse Free Optimistic Multi-party Contract Signing. In: ICALP 2000. Lecture Notes in Computer Science, vol. 1853, pp. 524–535. Springer (2000)
4. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: CCS 1993. pp. 62–73. ACM (1993)
5. Cachin, C., Camenisch, J.: Optimistic Fair Secure Computation. In: CRYPTO 2000. Lecture Notes in Computer Science, vol. 1880, pp. 93–111. Springer (2000)
6. Chen, L., Kudla, C., Paterson, K.G.: Concurrent Signatures. In: EUROCRYPT 2004. Lecture Notes in Computer Science, vol. 3027, pp. 287–305. Springer (2004)
7. Ferradi, H., Géraud, R., Maimuţ, D., Naccache, D., Pointcheval, D.: Legally Fair Contract Signing Without Keystones. In: ACNS 2016. Lecture Notes in Computer Science, vol. 9696, pp. 175–190. Springer (2016)
8. Garay, J., MacKenzie, P., Prabhakaran, M., Yang, K.: Resource Fairness and Composability of Cryptographic Protocols. In: TCC 2006. Lecture Notes in Computer Science, vol. 3876, pp. 404–428. Springer (2006)
9. Garay, J.A., Jakobsson, M., MacKenzie, P.: Abuse-Free Optimistic Contract Signing. In: CRYPTO 1999. Lecture Notes in Computer Science, vol. 1666, pp. 449–466. Springer (1999)
10. Goldwasser, S., Levin, L., Vanstone, S.A.: Fair Computation of General Functions in Presence of Immoral Majority. In: CRYPT0 1990. Lecture Notes in Computer Science, vol. 537, pp. 77–93. Springer (1991)

11. Guillou, L.C., Quisquater, J.J.: A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory. In: EUROCRYPT 1988. Lecture Notes in Computer Science, vol. 330, pp. 123–128. Springer (1988)
12. Maimuţ, D., Teşeleanu, G.: A Unified Security Perspective on Legally Fair Contract Signing Protocols. In: SECITC 2018. Lecture Notes in Computer Science, vol. 11359, pp. 477–491. Springer (2018)
13. Maurer, U.: Unifying Zero-Knowledge Proofs of Knowledge. In: AFRICACRYPT 2009. Lecture Notes in Computer Science, vol. 5580, pp. 272–286. Springer (2009)
14. Micali, S.: Simple and Fast Optimistic Protocols for Fair Electronic Exchange. In: PODC 2003. pp. 12–19. ACM (2003)
15. Ohta, K., Okamoto, T.: On Concrete Security Treatment of Signatures Derived from Identification. In: CRYPTO 1998. Lecture Notes in Computer Science, vol. 1462, pp. 354–369. Springer (1998)
16. Pinkas, B.: Fair Secure Two-Party Computation. In: EUROCRYPT 2003. Lecture Notes in Computer Science, vol. 2656, pp. 87–105. Springer (2003)
17. Schnorr, C.P.: Efficient Identification and Signatures For Smart Cards. In: CRYPTO 1989. Lecture Notes in Computer Science, vol. 435, pp. 239–252. Springer (1989)

A 1-out-of-n Signatures Without Key Separation

A.1 Description

In this section we present a more efficient 1-out-of-n signature. This signature only works when all the participants use the same underlying commutative group. We will denote the following signature with 1n-NKSS.

Setup(λ): Choose two commutative groups $\mathbb{G}$, $\mathbb{H}$, a homomorphism $[\cdot] : \mathbb{G} \to \mathbb{H}$ and a hash function $H : \{0,1\}^* \to \mathcal{C} \subseteq \mathbb{N}$. Note that we require that $|\mathbb{G}| \geq 2^\lambda$. For each user, choose $x_i \xleftarrow{\$} \mathbb{G}$ and compute $y_i \leftarrow [x_i]$. Output the public key $pk_i = y_i$. The secret key is $sk_i = x_i$.

Listing(): Collect the public keys and randomly shuffle them. Store the result into a list $\mathcal{L} = \{y_j\}_{j\in[0,n)}$ and output $\mathcal{L}$.

Sign$(m, sk_k, \mathcal{L})$: To sign a message $m \in \{0,1\}^*$, first generate the random elements $\alpha \xleftarrow{\$} \mathbb{G}$ and $c_j \xleftarrow{\$} \mathcal{C}$, where $j \in [0,n) \setminus \{k\}$. Then compute

$$\begin{aligned}
z &\leftarrow [\alpha] \otimes y_0^{c_0} \otimes \ldots \otimes y_{k-1}^{c_{k-1}} \otimes y_{k+1}^{c_{k+1}} \otimes \ldots \otimes y_{n-1}^{c_{n-1}} \\
c &\leftarrow H(\mathcal{L}, m, z) \\
c_k &\leftarrow c - c_0 - \ldots - c_{k-1} - c_{k+1} - \ldots - c_{n-1} \bmod c \\
s &\leftarrow \alpha \star x_k^{-c_k}.
\end{aligned}$$

Output the signature $(s, \mathcal{W})$, where $\mathcal{W} = \{c_j\}_{j\in[0,n)}$.

Verify$(m, s, \mathcal{W}, \mathcal{L})$: Compute the values $u \leftarrow \sum_{j=0}^{n-1} c_j \bmod c$ and $v \leftarrow [s] \otimes (\otimes_{j=0}^{n-1} y_j^{c_j})$. Output `true` if and only if $u \equiv H(\mathcal{L}, m, v) \bmod c$. Otherwise, output `false`.

Correctness. If the pair $(s, \mathcal{W})$ is generated according to the scheme, it is easy to see that

$$v = [s] \otimes (\otimes_{j=0}^{n-1} y_j^{c_j}) = [\alpha] \otimes [x_k]^{-c_k} \otimes (\otimes_{j=0}^{n-1} y_j^{c_j}) = z$$

and

$$u \equiv \sum_{j=0}^{n-1} c_j \equiv c \equiv H(\mathcal{L}, m, z) \equiv H(\mathcal{L}, m, v) \bmod c.$$

A.2 Security Analysis

Theorem 6's proof is similar to Theorem 1's proof and thus is omitted.

Theorem 6. *The* $1n$*-NKSS scheme is perfectly signer ambiguous.*

Theorem 7. *If the following statements are true*

- *an* EUF-CMCPA *attack on the* $1n$*-NKSS has non-negligible probability of success in the ROM,*
- *an* $\ell \in \mathbb{Z}$ *is known such that* $\gcd(c_0 - c_1, \ell) = 1$ *for all* $c_0, c_1 \in \mathcal{C}$ *with* $c_0 \neq c_1$*,*
- *for all* i *values,* $u_i \in \mathbb{G}$ *are known such that* $[u_i] = y_i^{\ell}$*,*

then the homomorphism $[\cdot]$ *can be inverted in polynomial time.*

Proof (sketch). In order to make $\mathcal{A}$ work properly we simulate the random oracle that correspond to the hash function (see Algorithm 1 with i always set to 0) and the signing oracle (see Algorithm 3). Note that $\mathcal{A}$ requests at most q_s and q_h signing and, respectively, random oracle queries.

The signing oracle $\mathcal{O}_S$ fails and returns $\perp$ only if we cannot assign c to (L_j, m_j, e) without causing an inconsistency in T_0. Thus, $\mathcal{O}_S$ is successful with probability at least $(1 - q_h/q)^{q_s} \geq 1 - q_h q_s/q$. The success probability of $\mathcal{A}$ in the simulated environment is $(1 - q_h q_s/q)\varepsilon$, where ε is $\mathcal{A}$'s success probability.

Let $(m, s, \{c_i\}_{i \in [0,n')}, L)$ be $\mathcal{A}$'s forgery, where $|L| = n'$. Define $z \leftarrow [s] \otimes (\otimes_{i=0}^{n'-1} y_i^{c_i})$. Due to the ideal randomness of $\mathcal{O}_H$, $\mathcal{A}$ queries $\mathcal{O}_H$ on (L, m, z) with probability $1 - 1/c$. Let $k \in [0, n')$ be the index of the user associated with the forgery. Then, according to Theorem 6, $\mathcal{A}$ will guess k with a probability of $1/n'$. If we invoke $\mathcal{A}$ at most $1/\varepsilon'$ times, where $\varepsilon' = n'(1 - q_h q_s/q)(1 - 1/c)\epsilon$, then we will find at least one $(\Theta, \Omega, \mathcal{O}_H)$ for which $\mathcal{A}$ knows k with probability 3/5. According to the heavy row lemma we are situated on a heavy row $\mathcal{H}$ with probability 1/2.

Define $\mathcal{O}_{H'}$ as a random oracle identical to $\mathcal{O}_H$ except for the (L, m, z) query to which $\mathcal{O}_{H'}$ responds with a random element $c' \neq c$. We rewind the simulation and run $\mathcal{A}$ at most $2/\varepsilon'$ times, but with access to $\mathcal{O}_{H'}$ instead of $\mathcal{O}_H$. We will be situated on $\mathcal{H}$ with a probability of 3/10. Now we can compute

$$\tilde{x}_k = u^a \star ({s'}^{-1} \star s)^b,$$

where a and b are computed using Euclid's algorithm such that $\ell a + (c' - c)b = 1$. As in Theorem 2's proof, we obtain $[\tilde{x}_k] = y_k$.

The overall success probability is $9/200 = 3/5 \cdot 3/10$ and $\mathcal{A}$ is invoked at most $3/\varepsilon'$ times. □

Algorithm 3: Signing oracle $\mathcal{O}_S$ simulation.

```
   Input: A signature query $(m_j, L_j)$ from $\mathcal{A}$
1  for $i \in [0, |L_j|)$ do
2      $s_i \xleftarrow{\$} \mathbb{G}$
3      $c_i \xleftarrow{\$} \mathcal{C}$
4      $e_i \leftarrow [s_i] \otimes y_i^{c_i}$
5  end for
6  $s \leftarrow s_0 \star \ldots \star s_{|L_j|-1}$
7  $c \leftarrow c_0 + \ldots + c_{|L_j|-1}$
8  $e \leftarrow e_0 \otimes \ldots \otimes e_{|L_j|-1}$
9  if $\nexists h$ such that $\{L_j, m_j, e, h\} \in T_0$ then
10     Append $\{L_j, m_j, e, c\}$ to $T_0$
11     return $(s, \{c_i\}_{i \in [0,|L_j|)})$
12 else
13     return $\perp$
14 end if
```

B Same Group 1-out-of-n Concurent Signature

B.1 Description

Based on the $1n$-NKSS signature we introduce a more efficient concurrent signature ($1n$-NKSCS) in the non-separable model. In this case, the scheme only uses two cryptographic hash functions $H_1, H_2 : \{0,1\}^* \to \mathcal{C}$.

Correctness. If the signature $\langle s_A, e_A, f \rangle$ is generated according to the scheme, it is easy to see that

$$[s_A] \otimes y_A^{e_A} \otimes y_B^f = [t_A] \otimes [x_A]^{-e_A} \otimes y_A^{e_A} \otimes y_B^f = [t_A] \otimes y_B^f.$$

Similarly, we can show correctness for *Bob*'s side.

B.2 Security Analysis

Theorem 8 is a direct consequence of Theorem 6 and Theorems 9 and 10's proofs are omitted due to their similarity to Theorems 4 and 5's proofs.

Theorem 8. *The $1n$-NKSCS scheme satisfies the concurrent signature ambiguity property in the ROM.*

Theorem 9. *Let $i \in \{A, B\}$. If the following statements are true*

- *an* EUF-CS *attack on the $1n$-NKSCS has non-negligible probability of success in the ROM,*
- *an $\ell \in \mathbb{Z}$ is known such that $\gcd(c_0 - c_1, \ell) = 1$ for all $c_0, c_1 \in \mathcal{C}$ with $c_0 \neq c_1$,*
- *for all i values, $u_i \in \mathbb{G}$ are known such that $[u_i] = y_i^\ell$,*

then the homomorphism $[\cdot]$ can be inverted in polynomial time.

Theorem 10. *The $1n$-NKSCS scheme is fair in the ROM.*

Alice		*Bob*
		$k \xleftarrow{\$} \{0,1\}^*$
		$f \leftarrow H_1(k)$
		$\% aSign$
		$t_B \xleftarrow{\$} \mathbb{G}$
		$f_B \leftarrow H_2\left(m_b, [t_B] \otimes y_A^f\right)$
		$e_B \leftarrow f_B - f \bmod c$
		$s_B \leftarrow t_B \star x_B^{-e_B}$
		$\sigma_B \leftarrow \langle s_B, e_B, f \rangle$
	$\xleftarrow{\quad\sigma_B\quad}$	
$\% aVerify$		
$T_A \leftarrow H_2\left(m_B, [s_B] \otimes y_B^{e_B} \otimes y_A^f\right)$		
if $T_A \not\equiv e_B + f \bmod c$ **then** abort		
$\% aSign$		
$t_A \xleftarrow{\$} \mathbb{G}$		
$f_A \leftarrow H_2\left(m_A, [t_A] \otimes y_B^f\right)$		
$e_A \leftarrow f_A - f \bmod c$		
$s_A \leftarrow t_A \star x_A^{-e_A}$		
$\sigma_A \leftarrow \langle s_A, e_A, f \rangle$		
	$\xrightarrow{\quad\sigma_A\quad}$	
		$\% aVerify$
		$T_B \leftarrow H_2\left(m_A, [s_A] \otimes y_A^{e_A} \otimes y_B^f\right)$
		if $T_B \not\equiv e_A + f \bmod c$ **then** abort
	$\xleftarrow{\quad k\quad}$	

Fig. 3. Same group concurrent signature.