# Concurrent Signatures without Random Oracles

Xiao Tan[1], Qiong Huang[2], and Duncan S. Wong[1]

[1] xiaotan4@gapps.cityu.edu.hk, duncan@cityu.edu.hk
Department of Computer Science,
City University of Hong Kong, Hong Kong S.A.R., China
[2] csqhuang@alumni.cityu.edu.hk
College of Informatics,
South China Agricultural University, Guangzhou, China

**Abstract.** Concurrent signatures provide a way to exchange digital signature among parties in an efficient and fair manner. To the best of our knowledge, all the existing solutions are all proven secure in the random oracle model, which is only heuristic. How to build an efficient concurrent signature scheme in the standard model has been remaining an open problem since the introduction in EUROCRYPT 2004. In this paper we answer the problem affirmatively and propose a novel construction of concurrent signature, the security of which does not rely on the random oracle assumption. The ambiguity of our scheme is slightly different from the existing schemes, requiring that any one can produce indistinguishable ambiguous signatures using merely public information. Security of the new scheme is based on Computational Diffie-Hellman (CDH) assumption in the standard model, which is a rather standard and well-studied assumption in cryptography.

**Keywords**: Concurrent Signatures, Without Random Oracles, Ambiguity

## 1 Introduction

The fair exchange of digital signatures is a protocol which ensures the atomicity of signature exchange among a group of parties so that the exchange is performed in an all-or-nothing manner, either all the parties get the others' signatures or none of them does. One of the electronic commerce applications of fair exchange is contract signing [11]. There are two main approaches for doing fair exchange. One is the timed release technique [9, 10] in which the signatures are computed as segments and exchanged in a polynomial number of rounds such that in any round, the amount of one party's knowledge on the counter-party's signature is nearly the same as that of the other party. This technique requires a number of communication rounds, and its fairness depends on the assumption that the parties have comparable computing power. The other technique relies on a semi-trusted offline arbitrator which only gets involved in the exchange when a dispute between the parties occurs. This type of fair exchange is referred to as Optimistic Fair Exchange (OFE) [1, 3, 12].

Concurrent signature, introduced by Chen, Kudla and Paterson [7], is the third approach for performing fair exchange of signatures. In a concurrent signature scheme, two parties $A$ and $B$ produce some signatures called ambiguous signatures $\sigma_A$ and $\sigma_B$, respectively. The signatures are ambiguous in a way that no outsider can distinguish the signers of them until an extra secret called *keystone* is released by $A$. Once the keystone is released, $\sigma_A$ and $\sigma_B$ will become binding to their signers *concurrently* and anyone can verify the validity of $\sigma_A$ and $\sigma_B$ with respect to their corresponding signers. Unlike OFE, a concurrent signature scheme does not have a semi-trusted arbitrator. Instead, the initiator $A$ and the matcher $B$ interact in the following three phases: (1) *Keystone Generation Phase*: $A$ generates a keystone $s$ and the corresponding *keystone fix* $f$, and sends $f$ to $B$; (2) *Ambiguous Signature Generation Phase*: $A$ generates and sends $\sigma_A$ to $B$, then $B$ generates and sends $\sigma_B$ back to $A$ after verifying the validity of $\sigma_A$; and (3) *Signature Authorship Binding Phase*: $A$ verifies the validity of $\sigma_B$ and sends $s$ to $B$. The keystone $s$ binds the authorship of $\sigma_A$ and $\sigma_B$ to $A$ and $B$, respectively and concurrently.

A secure concurrent signature should be unforgeable, ambiguous and fair. By Chen et al.'s model [7], unforgeability requires that when a party, say the matcher $B$, receives an ambiguous signature $\sigma_A$, $B$ can be sure that $\sigma_A$ must be generated by the initiator $A$, and vice versa. Ambiguity prevents the public from telling if an ambiguous signature $\sigma_A$ (resp. $\sigma_B$) was generated by $A$ or $B$ before the keystone $s$ is released. In addition, no one including $B$ (resp. $A$) should be able to convince anyone else that $\sigma_A$ (resp. $\sigma_B$) was generated by $A$ (resp. $B$). Fairness requires that once $s$ is released, the authorship of $\sigma_A$ to $A$ and that of $\sigma_B$ to $B$ should be bound concurrently. If $s$ is not released, none of $\sigma_A$ or $\sigma_B$ will bind to any signer.

Concurrent signature facilitates very efficient solutions to many electronic commerce applications. Generally, these applications assume that $B$ will receive the keystone $s$ when $B$'s binding signature $(s, f, \sigma_B)$ is to be used by $A$. This assumption can be easily guaranteed by existing mechanisms or in certain scenarios [7]. For instance, $A$ needs $B$'s signature to acquire goods or service from $B$, then $B$ can always ask $A$ to show the keystone before providing service to $A$.

*Motivation*: Several concurrent signature schemes were proposed in the literature [17, 19, 16, 13, 8, 18, 21]. These works either focus on improving the ambiguity model [17, 13], or fixing potential attacks against fairness [19], or extending concurrent signatures to multi-party setting [16, 18] or identity based setting [8], or balancing the capability of controlling the keystone between the initiator and the matcher [21]. However, none of the schemes are proven secure without random oracles. As we know, the random oracle model [4] is often criticized as a heuristic methodology, assuming that all the parties have oracle access to some truly random functions. Canetti et al. proved that there does not exist any equivalent function ensemble in the real world that can replace random oracles without security loss [6]. Therefore, it is more desirable to construct concurrent signatures in the standard model for reliable security.

We also observed a restriction in existing ambiguity models [7, 17]: given the ambiguous signatures, the public knows that at least one of the legitimate parties must have been involved. This leaked information harms privacy of the corresponding parties who prefer to remain anonymous until the fair exchange has been done successfully. A natural scenario is, when $A$ and $B$ sign a contract or coalition agreement of sensitive content, it is probably undesirable for others to know that either $A$ or $B$ are involved until the contract is validated by both of them. In this case, we need an ambiguity model to ensure that nothing about the authorship of ambiguous signatures are revealed before the keystone is released.

*Contributions*: Our contributions in this paper are two-fold:

- We propose a new ambiguity model for concurrent signatures that overcomes the restriction above. This model renders an arbitrarily large signer space to protect the anonymity of the corresponding parties.
- We present a concrete concurrent signature scheme in the new model, and prove its security without random oracles. It is the first concurrent signature scheme in the standard model.

### 1.1 Our Idea

To show the difficulty of finding a construction in the standard model, a useful observation is that most of the concurrent signatures are based on ring signatures built from Schnorr signature [15]. As a result, these schemes have to apply the forking lemma [14] to prove the unforgeability in the random oracle model.

In order to get rid of this barrier, we start from an efficient two-user ring signature in the standard model [5]. Instead of directly applying the ring signature as the building block, we twist it such that the ring is formed by one of the two parties and a keystone fix which can be treated as a virtual user. More precisely, the initiator $A$'s ambiguous signature $\sigma_A$ is produced upon the ring $(pk_A,\ f)$ using $sk_A$, and the matcher $B$'s ambiguous signature $\sigma_B$ is produced upon the ring $(pk_B,\ f)$ using $sk_B$. If $A$ follows the scheme to generate the keystone fix $f$, then releasing $s$ in the binding phase will result in decomposition of the rings $(pk_A,\ f)$ and $(pk_B,\ f)$ simultaneously. That is, $(s, f, \sigma_A)$ and $(s, f, \sigma_B)$ become bound to $A$ and $B$ respectively as two concurrent signatures. The overall idea is shown by Figure 1.

On the other hand, our construction achieves ambiguity in a flavor of anonymity which ensures that any person can produce $f$ by another means, such that it will be capable of generating indistinguishable $\sigma_A$, $\sigma_B$ as honestly computed ones. The indistinguishability holds even against the matcher $B$, say $B$ can not identify the authorship of the received ambiguous signature $\sigma_A$ when running a concurrent signature protocol. Hence, this strong ambiguity is mutually exclusive to the conventional unforgeability model. Fortunately, our scheme ensures that the concurrent signatures are still unforgeable after the keystone is released. In particular, $\sigma_A$, $\sigma_B$ can be bound to $A$, $B$ respectively only if $f$, $\sigma_A$, $\sigma_B$ are produced honestly by following the scheme.

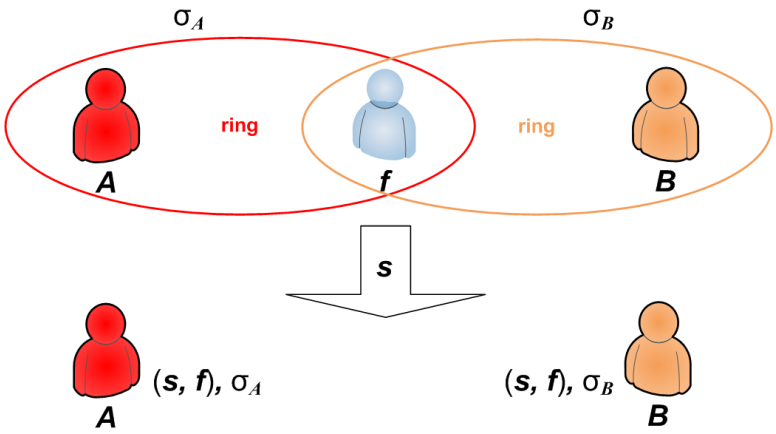


**Fig. 1.** An Illustration on the Idea of Our Concurrent Signature Construction

## 2 Definitions

Our definition for concurrent signatures is slightly different from previous formalizations. In particular, we need an additional algorithm KFVer for the matcher to verify the keystone. Formally, a concurrent signature scheme consists of the following probabilistic polynomial time (PPT) algorithms.

- $\mathsf{Setup}(1^k)$. On input a security parameter $k \in \mathbb{N}$, it outputs a set of system parameters denoted by param including the message space $\mathcal{M}$, the keystone space $\mathcal{S}$, the keystone fix space $\mathcal{F}$.
- $\mathsf{SKGen}(\mathsf{param})$. On input param, it outputs a public key pair $(pk, sk)$.
- $\mathsf{KFGen}(\mathsf{param})$. On input param, it outputs a keystone $s \in \mathcal{S}$ and a keystone fix $f \in \mathcal{F}$.
- $\mathsf{KFVer}(\mathsf{param}, f)$. On input param, a keystone fix $f$, it outputs 1 (accept) or 0 (reject).
- $\mathsf{ASign}(\mathsf{param}, pk_i, pk_j, f, m, sk_i)$. On input param, two public keys $pk_i$, $pk_j$, a keystone fix $f$, a message $m \in \mathcal{M}$, and a private key $sk_i$, it outputs an ambiguous signature $\sigma_i$.
- $\mathsf{AVer}(\mathsf{param}, pk_i, pk_j, f, \sigma_i, m)$. On input param, $pk_i$, $pk_j$, $f$, an ambiguous signature $\sigma_i$, and a message $m$, it outputs 1 (accept) or 0 (reject).
- $\mathsf{Ver}(\mathsf{param}, pk_i, pk_j, s, f, \sigma_i, m)$. On input param, $pk_i$, $pk_j$, $f \in \mathcal{F}$, $s \in \mathcal{S}$, an ambiguous signature $\sigma_i$, and a message $m$, it outputs 1 (accept) or 0 (reject).

The *correctness* requires that for all $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$, any two public keys $pk_i$, $pk_j$ where $(pk_i, sk_i) \leftarrow \mathsf{SKGen}(\mathsf{param})$, $(pk_j, sk_j) \leftarrow \mathsf{SKGen}(\mathsf{param})$, and a message $m \in \mathcal{M}$:

1. If $(s, f) \leftarrow \mathsf{KFGen}(\mathsf{param})$, then $\mathsf{KFVer}(\mathsf{param}, f) = 1$.

2. If $\sigma_i \leftarrow \mathsf{ASign}(\mathsf{param}, pk_i, pk_j, f, m, sk_i)$, where $(s, f) \leftarrow \mathsf{KFGen}(\mathsf{param})$, then $\mathsf{AVer}(\mathsf{param}, pk_i, pk_j, f, \sigma_i, m) = 1$, and $\mathsf{Ver}(\mathsf{param}, pk_i, pk_j, s, f, \sigma_i, m) = 1$. This condition also holds for $\sigma_j$.

### 2.1 The Protocol

We now describe how the concurrent signature scheme is to be carried out between two users $A$ and $B$ with public keys $pk_A$ and $pk_B$, respectively. Suppose $A$ is the initiator, and $B$ is the matcher. They run the protocol in the following three rounds:

R1 : $A$ computes $(s, f) \leftarrow \mathsf{KFGen}(\mathsf{param})$ and $\sigma_A \leftarrow \mathsf{ASign}(\mathsf{param}, pk_A, pk_B, f, m, sk_A)$, and sends $\langle f, \sigma_A \rangle$ to $B$.
R2 : $B$ checks if $\mathsf{KFVer}(\mathsf{param}, f) = 1$ and $\mathsf{AVer}(\mathsf{param}, pk_A, pk_B, f, \sigma_A, m) = 1$. If the verifications hold, $B$ computes $\sigma_B \leftarrow \mathsf{ASign}(\mathsf{param}, pk_B, pk_A, f, m, sk_B)$, and sends $\sigma_B$ to $A$.
R3 : $A$ checks if $\mathsf{AVer}(\mathsf{param}, pk_B, pk_A, f, \sigma_B, m) = 1$. If the verification holds, $A$ sends $s$ to $B$. Then, everyone can verify the concurrent signatures $(s, f, \sigma_A)$ and $(s, f, \sigma_B)$ by checking if $\mathsf{Ver}(\mathsf{param}, pk_A, pk_B, s, f, \sigma_A, m) = 1$ and $\mathsf{Ver}(\mathsf{param}, pk_B, pk_A, s, f, \sigma_B, m) = 1$.

## 3 Security Models

### 3.1 Unforgeability

The unforgeability is defined in the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

- **Setup Phase**: On input a security parameter $k \in \mathbb{N}$, $\mathcal{C}$ runs $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$, and generate key pairs for the participants in the system by running $(pk_i, sk_i) \leftarrow \mathsf{SKGen}(\mathsf{param})$ of multiple times, and sends $\mathsf{param}$, $PKSet = \{pk_i\}$ to $\mathcal{A}$.
- **Query Phase**: $\mathcal{A}$ can make queries to the following oracles.
  - $\mathsf{CorruptOracle}$. It takes a public key $pk_i \in PKSet$ and returns the corresponding private key $sk_i$.
  - $\mathsf{ASignOracle}$. It takes two public keys $pk_i, pk_j \in PKSet$, a keystone fix $f \in \mathcal{F}$, a message $m \in \mathcal{M}$, returns $\sigma_i \leftarrow \mathsf{ASign}(\mathsf{param}, pk_i, pk_j, f, m, sk_i)$.
- **Output Phase**: $\mathcal{A}$ outputs two public keys $pk_{i^*}, pk_{j^*} \in PKSet$, a message $m^* \in \mathcal{M}$, a keystone $s^* \in \mathcal{S}$, a keystone fix $f^* \in \mathcal{F}$, and an ambiguous signature $\sigma_{i^*}$. $\mathcal{A}$ wins the game if both the following conditions hold: (1) $\mathsf{Ver}(\mathsf{param}, pk_{i^*}, pk_{j^*}, s^*, f^*, \sigma_{i^*}, m) = 1$; (2) $\mathcal{A}$ has never queried $\mathsf{CorruptOracle}(pk_{i^*})$ or $\mathsf{ASignOracle}(pk_{i^*}, pk_{j^*}, f^*, m^*)$.

Note that $\mathcal{A}$ can generate keystones and keystone fixes by simply running the $\mathsf{KFGen}$ algorithm. $\mathcal{A}$ is not challenged by any keystone or keystone fix in this game, so it does not need oracles for generating or revealing keystones.

**Definition 1 (Unforgeability).** *A concurrent signature scheme is said to be* $(\epsilon, t, q_c, q_s)$-unforgeable *if for any probabilistic polynomial-time adversary* $\mathcal{A}$ *which runs in time* $t$ *making* $q_c$ *queries to* CorruptOracle *and* $q_s$ *queries to* ASignOracle*, its probability of winning in the game above is bounded by* $\epsilon$*.*

Our definition of unforgeability ensures that no PPT adversary $\mathcal{A}$ can produce a valid concurrent signature bound to a public key $pk_i$ if $\mathcal{A}$ does not know $sk_i$. This formalization refers to the "unforgeability of binding signatures", which differs from the original unforgeability model proposed by Chen et al. [7] referring to the "unforgeability of ambiguous signatures" in the view of the two corresponding parties. However, we address that our model is sufficient for most scenarios of fair exchange of signatures. The reason is, even if the ambiguous signatures are forgeable by others besides the legitimate parties, our unforgeability model ensures that only the legitimate parties can make the ambiguous signatures become binding. Note that [21] formalizes unforgeability of concurrent signatures in a similar way as our model.

### 3.2 Ambiguity

Informally, the ambiguity prevents anyone from telling if two ambiguous signatures together with the keystone fix are generated honestly by user $i$ and $j$, or merely simulated by anyone using the public information, before the keystone is released. In the formalization below, we define an algorithm ASignSim called *ambiguous signature simulator*, which models the capability of an outsider who outputs two simulated ambiguous signatures $\sigma_i$, $\sigma_j$ together with a keystone fix $f$.

- $\mathsf{ASignSim}(\mathsf{param}, pk_i, pk_j, m)$. On input param, two public keys $pk_i, pk_j$, a message $m$, the algorithm outputs a keystone fix $f$ and two ambiguous signatures $\sigma_i, \sigma_j$.

Note that ASignSim needs to be specified explicitly in the scheme description. The ambiguity game is defined as follows where a challenger $\mathcal{C}$ runs the game against an adversary $\mathcal{A}$.

- **Setup Phase**: The same as the setup phase in Section 3.1, except that $\mathcal{C}$ also sends the private keys $SKSet = \{sk_i\}$ of all the participants to $\mathcal{A}$.
- **Query Phase**: $\mathcal{A}$ can query the following oracles.
  - KFGenOracle. It returns a keystone fix $f$ by running $(s, f) \leftarrow \mathsf{KFGen}(\mathsf{param})$.
  - KSReleaseOracle. It takes a keystone fix $f$ which is produced by KFGenOracle, returns the corresponding keystone $s$.
- **Challenge Phase**: $\mathcal{A}$ sends to $\mathcal{C}$ two public keys $pk_i, pk_j \in PKSet$ and a message $m \in \mathcal{M}$. $\mathcal{C}$ tosses a random bit $b \in \{0, 1\}$:
  - If $b = 0$: $\mathcal{C}$ runs $(s, f) \leftarrow \mathsf{KFGen}(\mathsf{param})$, then $\sigma_i \leftarrow \mathsf{ASign}(\mathsf{param}, pk_i, pk_j, f, m, sk_i)$ and $\sigma_j \leftarrow \mathsf{ASign}(\mathsf{param}, pk_j, pk_i, f, m, sk_j)$.
  - If $b = 1$: $\mathcal{C}$ runs $(f, \sigma_i, \sigma_j) \leftarrow \mathsf{ASignSim}(\mathsf{param}, pk_i, pk_j, m)$.

Then $\mathcal{C}$ responses $\mathcal{A}$ with $(f, \sigma_i, \sigma_j)$.
- **Output Phase**: $\mathcal{A}$ continues querying the oracles above and outputs a guessing $b' \in \{0, 1\}$ at the end of the game, under the restriction that $\mathcal{A}$ has never queried $\mathsf{KSReleaseOracle}(f)$. $\mathcal{A}$ wins the game if $b' = b$. The advantage of $\mathcal{A}$ is defined as $\mathsf{Adv}_{\mathsf{CS}}^{\mathsf{AMBI},\mathcal{A}} = |\Pr[b' = b] - 1/2|$.

Since $\mathcal{A}$ knows all the private keys, it can produce any users' signatures by itself. Hence, $\mathcal{C}$ does not provide $\mathsf{CorruptOracle}$ and $\mathsf{ASignOracle}$ in this game.

**Definition 2 (Ambiguity).** *A concurrent signature scheme is said to be* $(\epsilon, t, q_f, q_r)$-ambiguous *if for any probabilistic polynomial-time adversary* $\mathcal{A}$ *which runs in time* $t$ *making* $q_f$ *queries to* $\mathsf{KFGenOracle}$ *and* $q_r$ *queries to* $\mathsf{KSReleaseOracle}$*, its probability of winning in the game above is bounded by* $\epsilon$*.*

In the literature, two ambiguity models were proposed for concurrent signatures. One is the original ambiguity model proposed by Chen et al. [7] (we denote it by $\mathsf{AMB}_O$), the other is the perfect ambiguity model proposed by Susilo et al. [17, 19] (we denote it by $\mathsf{AMB}_P$). Below we compare these two models with our ambiguity model which has a flavor of anonymity, denoted by $\mathsf{AMB}_A$.

[17, 19] has shown that $\mathsf{AMB}_P$ is a stronger model than $\mathsf{AMB}_O$, in the sense that it is more difficult for an adversary to identify the authorship of ambiguous signatures before the keystone is released. Given a pair of ambiguous signatures $(\sigma_A, \sigma_B)$ on a keystone fix $f$, $\mathsf{AMB}_P$ captured the following four possibilities of the signers while $\mathsf{AMB}_O$ only covers the first three: (1) $\sigma_A$ is produced by $A$ and $\sigma_B$ is produced by $B$; (2) both $\sigma_A$ and $\sigma_B$ are produced by $A$; (3) both $\sigma_A$ and $\sigma_B$ are produced by $B$; (4) $\sigma_A$ is produced by $B$ and $\sigma_B$ is produced by $A$. As we can see, in the previous ambiguity models, the probability is at least $1/4$ for an adversary to correctly guess the signers. Furthermore, both $\mathsf{AMB}_O$ and $\mathsf{AMB}_P$ leak the following information to the public: given the ambiguous signatures, at least one of the corresponding parties ($A$ or $B$) must have been involved. This leaked information harms the privacy of the concerned parties before their exchange is successfully completed. On the other hand, $\mathsf{AMB}_A$ renders an arbitrarily large signer space that totally overcomes the above restriction, since everyone can use $\mathsf{ASignSim}$ to produce indistinguishable ambiguous signatures as generated by honest participants. Obviously, Case (1)(2)(3) above are already covered by $\mathsf{AMB}_A$, but it does not cover Case (4), since that when $\sigma_A$ and $\sigma_B$ are produced by $\mathsf{ASignSim}$, the issuer of both the ambiguous signatures must be the same person. However, there seems no realistic scenario fall in Case (4). The reason is, Case (4) implies that both $A$ and $B$ are malicious, which contradicts with the general assumption in a cryptographic protocol that at least one participant is honest. Table 1 compares the cardinality of signer space among the existing ambiguity models.

Nguyen proposed a similar property called anonymity for concurrent signatures [13]. It requires that the ambiguous signatures are indistinguishable from random distribution. We address that Nguyen's model is not well defined: even if the ambiguous signatures are uniformly distributed in the signature space, it is still possible for an adversary to distinguish whether signatures are issued

**Table 1.** Comparison of Ambiguity Models

| **Models** | **# Signer Space of ($\sigma_A$, $\sigma_B$)** |
|---|---|
| $\mathsf{AMB}_O$ | $\lvert\{(A,B),(A,A),(B,B)\}\rvert = 3$ |
| $\mathsf{AMB}_P$ | $\lvert\{(A,B),(A,A),(B,B),(B,A)\}\rvert = 4$ |
| $\mathsf{AMB}_A$ | $\lvert\{(A,B),(X,X)\}\rvert = \infty$ |

* Signer Space denotes the set of possible signers given the ambiguous signatures, e.g. ($A$, $B$) represents that $A$ signs $\sigma_A$ and $B$ signs $\sigma_B$. # denotes the cardinality. $X$ denotes an arbitrary person who knows the system parameters and $pk_A$, $pk_B$.

by honest parties. In Appendix A, we show that Nguyen's concurrent signature scheme is not anonymous in our ambiguity model, implying that Nguyen's model is strictly weaker than ours.

### 3.3 Fairness

Concurrent binding is the key security requirement toward fairness. There are two aspects to satisfy: (1) Binding after Releasing: an adversary should not be able to produce a valid ambiguous signature $\sigma_i$ bound to the public key $pk_i$, until $s$ is published; and (2) Concurrency of Binding: an adversary should not be able to generate two ambiguous signatures $\sigma_i$ and $\sigma_j$ such that $\sigma_i$ is bound to the public key $pk_i$ while $\sigma_j$ is not bound to the public key $pk_j$, after the keystone $s$ is released.

The fairness game is defined as below, where a challenger $\mathcal{C}$ simulates the game against an adversary $\mathcal{A}$.

- **Setup Phase**: The same as the Setup Phase in Section 3.2.
- **Query Phase**: The same as the Query Phase in Section 3.2.
- **Output Phase**: $\mathcal{A}$ generates and sends ($pk_{i^*}$, $pk_{j^*}$, $f^*$, $s^*$, $m^*$, $\sigma_{i^*}$) back to $\mathcal{C}$, where $pk_{i^*}, pk_{j^*} \in PKSet$, $f^* \in \mathcal{F}$, $s^* \in \mathcal{S}$, $m^* \in \mathcal{M}$, and $\sigma_{i^*}$ is an ambiguous signature. $\mathcal{A}$ wins the game if either of the following conditions holds: (1) $\mathsf{Ver}(\mathsf{param}, pk_{i^*}, pk_{j^*}, s^*, f^*, \sigma_{i^*}, m^*) = 1$ under the restriction that $f^*$ is returned by $\mathsf{KFGenOracle}$ and $\mathcal{A}$ has never queried $\mathsf{KSReleaseOracle}(f^*)$; (2) $\mathcal{A}$ outputs another ambiguous signature $\sigma_{j^*}$, satisfying: $\mathsf{AVer}(\mathsf{param}, pk_{i^*}, pk_{j^*}, f^*, \sigma_{i^*}, m^*) = 1$, $\mathsf{AVer}(\mathsf{param}, pk_{j^*}, pk_{i^*}, f^*, \sigma_{j^*}, m^*) = 1$, $\mathsf{Ver}(\mathsf{param}, pk_{i^*}, pk_{j^*}, s^*, f^*, \sigma_{i^*}, m^*) = 1$, but $\mathsf{Ver}(\mathsf{param}, pk_{j^*}, pk_{i^*}, s^*, f^*, \sigma_{j^*}, m^*) = 0$.

**Definition 3 (Fairness).** *A concurrent signature scheme is said to be* $(\epsilon, t, q_f, q_r)$-fair *if for any probabilistic polynomial-time adversary* $\mathcal{A}$ *which runs in time* $t$ *making* $q_f$ *queries to* $\mathsf{KFGenOracle}$ *and* $q_r$ *queries to* $\mathsf{KSReleaseOracle}$*, its probability of winning in the game above is bounded by* $\epsilon$*.*

The winning condition (1) ensures that a concurrent signature becomes bound only if the keystone is released or compromised. The restrictions in condition (1)

forbid $\mathcal{A}$ from winning the game in an obvious way: $\mathcal{A}$ produces a forged ambiguous signature $\sigma_{i^*}$ using $sk_{i^*}$, then binds $\sigma_{i^*}$ to $pk_{i^*}$ by querying $\mathsf{KSReleaseOracle}$ to get $s^*$. The condition (2) ensures that the binding of the signatures to their respective signers occurs concurrently.

# 4 A Concurrent Signature Scheme without Random Oracles

In this section, we propose an efficient concurrent signature scheme and prove its security under the games we defined above. The underlying complexity assumptions are reviewed in Appendix B.

## 4.1 Proposed Scheme

Below are the details of our concurrent signature scheme.

- $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$. The setup algorithm selects two cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$ of prime order $q \geq 2^k$ and an admissible pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. It also picks a generator $g \in \mathbb{G}$ and $h \stackrel{R}{\leftarrow} \mathbb{G}$. Set the message space $\mathcal{M} = \{0,1\}^*$, the keystone space $\mathcal{S} = \mathbb{Z}_q$, the keystone fix space $\mathcal{F} = \mathbb{G} \times \mathbb{G}$. Let $H : \{0,1\}^* \rightarrow \{0,1\}^l$ be a collision-resistant hash function, where the output length $l$ is given as an auxiliary input to the $\mathsf{Setup}$ algorithm. Define $\mathsf{param} := (\mathbb{G}, \mathbb{G}_T, e, q, g, h, \mathcal{M}, \mathcal{S}, \mathcal{F}, H)$.
- $(pk, sk) \leftarrow \mathsf{SKGen}(\mathsf{param})$. The key generation algorithm picks $\alpha \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and $u', u_1, u_2, \cdots, u_l \stackrel{R}{\leftarrow} \mathbb{G}$, computes $\delta = g^\alpha$, and outputs $pk = (\delta, u', u_1, u_2, \cdots, u_l)$, $sk = \alpha$.
- $(s, f) \leftarrow \mathsf{KFGen}(\mathsf{param})$. The keystone fix generation algorithm picks a keystone $s \stackrel{R}{\leftarrow} \mathbb{Z}_q$, computes $\rho = h^s$, $\tau = g^{s^{-1}}$, and sets the keystone-fix as $f = (\rho, \tau)$.
- $1/0 \leftarrow \mathsf{KFVer}(\mathsf{param}, f)$. The keystone fix verification algorithm parses $f = (\rho, \tau)$, and outputs 1 if $\rho, \tau \in \mathbb{G}$ and the following equation holds:

$$e(\rho, \tau) = e(g, h) \tag{1}$$

  Otherwise, it outputs 0.
- $\sigma_i \leftarrow \mathsf{ASign}(\mathsf{param}, pk_i, pk_j, f, m, sk_i)$. The ambiguous signature generation algorithm parses $f = (\rho, \tau)$ and $pk_i = (\delta_i, u'_i, u_{i1}, u_{i2}, \cdots, u_{il})$, picks $r_i \stackrel{R}{\leftarrow} \mathbb{Z}_q$, computes $\eta_i = g^{r_i}$, $M_i = H(pk_i \| pk_j \| f \| m)$, $\zeta_i = \rho^{sk_i}(u'_i \prod_{t \in \tilde{M}_i} u_{it})^{r_i}$ where $\tilde{M}_i$ is the set of indices $t$ such that the string $M_i$'s $t$-th bit $M_i[t] = 1$, and outputs $\sigma_i = (\zeta_i, \eta_i)$.
- $1/0 \leftarrow \mathsf{AVer}(\mathsf{param}, pk_i, pk_j, f, \sigma_i, m)$. The ambiguous signature verification algorithm parses $f = (\rho, \tau)$, $pk_i = (\delta_i, u'_i, u_{i1}, u_{i2}, \cdots, u_{il})$, $\sigma_i = (\zeta_i, \eta_i)$, computes $M_i = H(pk_i \| pk_j \| f \| m)$, and outputs 1 if the following equation holds:

$$e(g, \zeta_i) = e(\delta_i, \rho) e(\eta_i, u'_i \prod_{t \in \tilde{M}_i} u_{it}) \tag{2}$$

Otherwise, it outputs 0.

– $1/0 \leftarrow \mathsf{Ver}(\mathsf{param}, pk_i, pk_j, s, f, \sigma_i, m)$. The signature verification algorithm parses $f = (\rho, \tau)$, and outputs 1 if all the following conditions hold:
(1) $\rho = h^s$ and $\tau = g^{s^{-1}}$;
(2) $\mathsf{AVer}(\mathsf{param}, pk_i, pk_j, f, \sigma_i, m) = 1$.
Otherwise, it outputs 0.

The correctness of the scheme above is straightforward and therefore is omitted here. The following is the ambiguous signature simulation algorithm $\mathsf{ASignSim}$ which is needed in the proof of ambiguity. Let $pk_i = (\delta_i, u'_i, u_{i1}, u_{i2}, \cdots, u_{il})$, $pk_j = (\delta_j, u'_j, u_{j1}, u_{j2}, \cdots, u_{jl})$.

– $(f, \sigma_i, \sigma_j) \leftarrow \mathsf{ASignSim}(\mathsf{param}, pk_i, pk_j, m)$. The ambiguous signature simulator performs the following computations:
(1) Pick $s \overset{R}{\leftarrow} \mathbb{Z}_q$, compute $\rho = g^s$, $\tau = h^{s^{-1}}$, and set the keystone-fix as $f = (\rho, \tau)$.
(2) Pick $r_i \overset{R}{\leftarrow} \mathbb{Z}_q$, compute $\eta_i = g^{r_i}$, $M_i = H(pk_i \| pk_j \| f \| m)$, $\zeta_i = {\delta_i}^s (u'_i \prod_{t \in \tilde{M}_i} u_{it})^{r_i}$, and $\sigma_i = (\zeta_i, \eta_i)$.
(3) Pick $r_j \overset{R}{\leftarrow} \mathbb{Z}_q$, compute $\eta_j = g^{r_j}$, $M_j = H(pk_j \| pk_i \| f \| m)$, $\zeta_j = {\delta_j}^s (u'_j \prod_{t \in \tilde{M}_j} u_{jt})^{r_j}$, and $\sigma_j = (\zeta_j, \eta_j)$.
(4) Output $f, \sigma_i, \sigma_j$ as required.

### 4.2 Security Proofs

We start with a brief security analysis of the proposed scheme on the high level. For unforgeability, just note that any PPT adversary can not compute $(s^*, f^*, \sigma_{i^*})$ with non-negligible probability if it does not know either $sk_{i^*}$ or the logarithm of $h$ to base $g$, due to the hardness of CDH problem. On the other hand, the ambiguity is enforced since that the keystone can be produced in either one of two indistinguishable patterns by $\mathsf{KFGen}$ or $\mathsf{ASignSim}$. The scheme is fair because both $\sigma_{i^*}$ and $\sigma_{j^*}$ are produced on the same keystone $s^*$ of which the releasing leads to concurrent binding, and the DL assumption in $\mathbb{G}$ ensures that it is infeasible to compute $s^*$ from $f^*$ with noticeable advantage for any PPT adversary.

Below we give the formal proof with respect to unforgeability (Def. 1), ambiguity (Def. 2) and fairness (Def. 3).

**Theorem 1.** *The proposed concurrent signature scheme is $(\epsilon, t, q_c, q_s)$-unforgeable, if $(\epsilon', t')$-Computational Diffie Hellman (CDH) assumption holds in $\mathbb{G}$:*

$$\epsilon' \geq \frac{\epsilon}{4q_s(l+1)p(k)}, \qquad t' \geq t + O(q_s t_m(l+2) + (p(k) + 4q_s)t_e),$$

*where $p(k)$ denotes the number of participants in the system[1], $t_m$ and $t_e$ denote the time needed for one multiplication and one exponentiation on $\mathbb{G}$ respectively.*

*Proof.* We construct a PPT algorithm $\mathcal{S}$ that simulates the oracles in the unforgeability game, and produces a valid solution for a CDH instance if there exists an $(\epsilon, t, q_c, q_s)$-adversary $\mathcal{A}$ who successfully outputs a forgery. Let the CDH instance be $(\mathbb{G}, q, g, A = g^a, B = g^b)$ where $g, A, B \in \mathbb{G}$ of order $q$.

**Setup**: At the start, $\mathcal{S}$ sets the parameters $\mathbb{G}, q, g$ as given in the CDH instance and $h = A$. The other parameters are set as in Setup algorithm. $\mathcal{S}$ sets $n = 2q_s$ and we assume that $n(l+1) < q$. $\mathcal{S}$ guesses a value $1 \leq \hat{i} \leq p(k)$ that $\mathcal{A}$ will forge the concurrent signature of the user $i^* = \hat{i}$. For $1 \leq i \leq p(k)$ where $i \neq \hat{i}$, $\mathcal{S}$ runs SKGen for $p(k)-1$ times to produce key pairs $\{(pk_i, sk_i)\}$ for all the users except user $\hat{i}$. To generate user $\hat{i}$'s public key, $\mathcal{S}$ performs the following computations:

1. Pick $w \xleftarrow{R} \mathbb{Z}_l$, $\mu', \mu_1, \mu_2, \cdots, \mu_l \xleftarrow{R} \mathbb{Z}_n$, and $\nu', \nu_1, \nu_2, \cdots, \nu_l \xleftarrow{R} \mathbb{Z}_q$.
2. Set $u'_{\hat{i}} = B^{-nw+\mu'} g^{\nu'}$, and $u_{\hat{i}t} = B^{\mu_t} g^{\nu_t}$ for $1 \leq t \leq l$.
3. Set $pk_{\hat{i}} = (B, u'_{\hat{i}}, \cdots, u_{\hat{i}t})$.

Then $\mathcal{S}$ sends param and $\{pk_i\}$ for $1 \leq i \leq p(k)$ to $\mathcal{A}$, and keeps $\{sk_j\}$ for $1 \leq j \leq p(k)$, $j \neq \hat{i}$ for answering queries to CorruptOracle later.

Besides, $\mathcal{S}$ defines the following functions where $\tilde{M}$ is the set of indices $t$ such that $M[t] = 1$:

$$J(M) = \mu' + \sum_{t \in \tilde{M}} \mu_t - nw, \qquad L(M) = \nu' + \sum_{t \in \tilde{M}} \nu_t$$

Then we have:

$$u'_{\hat{i}} \prod_{t \in \tilde{M}} u_{\hat{i}t} = B^{J(M)} g^{L(M)}$$

**Query**: $\mathcal{S}$ simulates the ExtractOralce and ASignOracle as follows:

- CorruptOracle: If the corrupted public key $pk_i \neq pk_{\hat{i}}$, $\mathcal{C}$ returns the corresponding $sk_i$ which is produced in Setup phase. Otherwise, $\mathcal{S}$ aborts.
- ASignOracle: If the ASign query $(pk_i, pk_j, f, m)$ is for $pk_i \neq pk_{\hat{i}}$, $\mathcal{C}$ runs $\sigma_i \leftarrow \mathsf{ASign}(\mathsf{param}, pk_i, pk_j, f, m, sk_i)$ using $sk_i$, and returns $\sigma_i$. Otherwise, if $pk_i = pk_{\hat{i}}$, $\mathcal{S}$ performs the following computations:
  1. Compute $M_{\hat{i}} = H(pk_{\hat{i}} \| pk_j \| f \| m)$.
  2. If $J(M_{\hat{i}}) \neq 0 \bmod q$, parse $f = (\rho, \tau)$, randomly pick $r_{\hat{i}} \in \mathbb{Z}_q$, compute:

$$\zeta_{\hat{i}} = \rho^{-L(M_{\hat{i}})/J(M_{\hat{i}})} (u'_{\hat{i}} \prod_{t \in \tilde{M}_{\hat{i}}} u_{\hat{i}t})^{r_{\hat{i}}}, \qquad \eta_{\hat{i}} = \rho^{-1/J(M_{\hat{i}})} g^{r_{\hat{i}}}$$

     and return $\sigma_{\hat{i}} = (\zeta_{\hat{i}}, \eta_{\hat{i}})$.
  3. If $J(M_{\hat{i}}) = 0 \bmod q$, $\mathcal{S}$ simply aborts.

[1] $p$ is a polynomial function of the security parameter $k$. Obviously we have $q_c \leq p(k) - 1$.

The simulation for generating $pk_{\hat{i}}$'s ambiguous signatures is perfect:

$$\zeta_{\hat{i}} = \rho^{-L(M_{\hat{i}})/J(M_{\hat{i}})}(u'_{\hat{i}} \prod_{t\in\tilde{M}_{\hat{i}}} u_{\hat{i}t})^{r_{\hat{i}}} = B^{as}(B^{J(M_{\hat{i}})}g^{L(M_{\hat{i}})})^{-as/J(M_{\hat{i}})}(B^{J(M_{\hat{i}})}g^{L(M_{\hat{i}})})^{r_{\hat{i}}}$$
$$= \rho^{b}(B^{J(M_{\hat{i}})}g^{L(M_{\hat{i}})})^{r_{\hat{i}}-as/J(M_{\hat{i}})} = \rho^{b}(u'_{\hat{i}} \prod_{t\in\tilde{M}_{\hat{i}}} u_{\hat{i}t})^{\tilde{r}_{\hat{i}}}$$
$$\eta_{\hat{i}} = \rho^{-1/J(M_{\hat{i}})}g^{r_{\hat{i}}} = g^{r_{\hat{i}}-as/J(M_{\hat{i}})} = g^{\tilde{r}_{\hat{i}}}$$

where we denote $s = \log_h \rho$ and $\tilde{r}_i = r_{\hat{i}} - as/J(M_{\hat{i}})$.
As $n(l+1) < q$, we have $0 \le nw < q$. Hence $J(M_{\hat{i}}) = 0 \bmod q$ implies $J(M_{\hat{i}}) = 0 \bmod n$, or equivalently $J(M_{\hat{i}}) \neq 0 \bmod n$ implies $J(M_{\hat{i}}) \neq 0 \bmod q$. So the simulation for $\mathsf{ASignOracle}$ does not abort if $J(M_{\hat{i}}) \neq 0 \bmod n$.

**Output**: Suppose the forged concurrent signature output by $\mathcal{A}$ is $\sigma_{i^*} = (\zeta_{i^*}, \eta_{i^*})$ on $(pk_{i^*}, pk_{j^*}, f^*, s^*, m^*)$. When $pk_{i^*} = pk_{\hat{i}}$ and $J(M_{i^*}) = 0 \bmod q$ where $M_{i^*} = H(pk_{i^*}\|pk_{j^*}\|f^*\|m^*)$, $\mathcal{S}$ can solve the CDH instance by computing $g^{ab}$ as:

$$(\frac{\zeta_{i^*}}{\eta_{i^*}^{L(M_{i^*})}})^{s^{*-1}} = (\frac{B^{as^*}(u'_{\hat{i}}\prod_{t\in\tilde{M}_{i^*}} u_{i^*t})^{r_{i^*}}}{(g^{r_{i^*}})^{L(M_{i^*})}})^{s^{*-1}} = (\frac{B^{as^*}(g^{L(M_{i^*})})^{r_{i^*}}}{(g^{r_{i^*}})^{L(M_{i^*})}})^{s^{*-1}} = B^a = g^{ab}$$

Otherwise, $\mathcal{S}$ aborts when $pk_{i^*} \neq pk_{\hat{i}}$ or $J(M_{i^*}) \neq 0 \bmod q$.

Now we analyze the probability that $\mathcal{S}$ outputs $g^{ab}$ without abort. Note that if all the four events below happen, there will be no abort in the simulation:

(1) $P^*$: $J(X^*) = 0 \bmod q$, where $X^* = M_{i^*}$;
(2) $P_\pi$: $J(X_\pi) \neq 0 \bmod n$, for any $X_\pi = M_{\hat{i}} = H(pk_{\hat{i}}\|pk_j\|f\|m) \neq M_{i^*}$ when $(pk_{\hat{i}}, pk_j, f, m)$ is queried to $\mathsf{ASignOracle}$[1];
(3) $Q^*$: $Y^* = pk_{\hat{i}}$, where $Y^* = pk_{i^*}$.
(4) $Q_\xi$: $Y_\xi \neq pk_{\hat{i}}$, for any $Y_\xi = pk_i$ queried to $\mathsf{CorruptOracle}$.

Let $q_m$ be the number of queries made in event (2), then we have $q_m \le q_s$. Obviously the number of queries made in event (4) is $q_c$. Therefore, the probability that $\mathcal{S}$ not aborting in the game is:

$$\mathsf{Pr}[\neg\mathsf{abort}] \ge \mathsf{Pr}[P^* \wedge \bigwedge_{\pi=1}^{q_m} P_\pi \wedge Q^* \wedge \bigwedge_{\xi=1}^{q_c} Q_\xi]$$

Note that $P^* \wedge \bigwedge_{\pi=1}^{q_s} P_\pi$ and $Q^* \wedge \bigwedge_{\xi=1}^{q_c} Q_\xi$ are independent events. Below we separately analyze the probabilities that either of these two events happens.

Since that $X^* \neq X_\pi$, the event $\neg P_\pi$: $J(X_\pi) = 0 \bmod n$ is independent from $P^*$, thus $\Pr[\neg P_\pi | P^*] = 1/n$. Besides, $J(X^*) = 0 \bmod q$ implies $J(X^*) = 0 \bmod n$, so we have:

[1] Note that $\mathcal{A}$ can not query $(pk_{\hat{i}}, pk_j, f, m)$ to $\mathsf{ASignOracle}$ when $M_{\hat{i}} = M_{i^*}$. Since $H$ is collision resistant, we have $pk_{\hat{i}}\|pk_j\|f\|m = pk_{i^*}\|pk_{j^*}\|f^*\|m^*$. So this query is not allowed in the unforgeability game.

$$\begin{aligned}\mathsf{Pr}[P^* \wedge \bigwedge_{\pi=1}^{q_m} P_\pi] &= \mathsf{Pr}[P^*]\mathsf{Pr}[\bigwedge_{\pi=1}^{q_m} P_\pi | P^*] \\ &= \mathsf{Pr}[J(X^*) = 0 \bmod q \wedge J(X^*) = 0 \bmod n](1 - \mathsf{Pr}[\bigvee_{\pi=1}^{q_m} P_\pi | P^*]) \\ &\geq \mathsf{Pr}[J(X^*) = 0 \bmod n]\mathsf{Pr}[J(X^*) = 0 \bmod q | J(X^*) = 0 \bmod n](1 - \sum_{\pi=1}^{q_m} \mathsf{Pr}[\neg P_\pi | P^*]) \\ &\geq \frac{1}{n} \cdot \frac{1}{l+1} \cdot (1 - \frac{q_s}{n}) = \frac{1}{4q_s(l+1)}\end{aligned}$$

Since that it is not allowed to query the target public key $pk_{i^*}$ to $\mathsf{CorruptOracle}$, the event $\neg Q_\xi$: $Y_\xi = pk_i = pk_{\hat{i}}$ could never happen when $Q^*$ happens, so $\Pr[\neg Q_\xi | Q^*] = 0$. And we have:

$$\begin{aligned}\mathsf{Pr}[Q^* \wedge \bigwedge_{\xi=1}^{q_c} Q_\xi] &= \mathsf{Pr}[Q^*]\mathsf{Pr}[\bigwedge_{\xi=1}^{q_c} Q_\xi | Q^*] = \mathsf{Pr}[pk_{i^*} = pk_{\hat{i}}](1 - \mathsf{Pr}[\bigvee_{\xi=1}^{q_c} Q_\xi | Q^*]) \\ &\geq \frac{1}{p(k)}(1 - \sum_{\xi=1}^{q_c} \mathsf{Pr}[\neg Q_\xi | Q^*]) = \frac{1}{p(k)}\end{aligned}$$

And here comes the combined result:

$$\mathsf{Pr}[\neg \mathsf{abort}] \geq \mathsf{Pr}[P^* \wedge \bigwedge_{\pi=1}^{q_s} P_\pi]\mathsf{Pr}[Q^* \wedge \bigwedge_{\xi=1}^{q_m} Q_\xi] \geq \frac{1}{4q_s(l+1)p(k)}$$

Hence, $\mathcal{S}$ solves the given CDH instance with probability at least $\frac{\epsilon}{4q_s(l+1)p(k)}$ when an $(\epsilon, t, q_c, q_s)$-adversary $\mathcal{A}$ outputs a forgery without aborts. In the simulation, it requires 1 exponentiation for producing each key pair, $l+2$ multiplications and 4 exponentiations for each $\mathsf{ASignOracle}$ query. Therefore, the time complexity of $\mathcal{S}$ is bounded by $t + O(q_s t_m(l+2) + (p(k) + 4q_s)t_e)$. □

**Theorem 2.** *The proposed concurrent signature scheme is unconditionally ambiguous.*

**Theorem 3.** *The proposed concurrent signature scheme is $(\epsilon, t, q_f, q_r)$-fair, if $(\epsilon', t')$-Discrete Logarithm (DL) assumption holds in $\mathbb{G}$:*

$$\epsilon' \geq \epsilon(1 - \frac{q_r}{q_f}), \quad t' \geq t + O((2q_f - 1)t_e),$$

*where $t_e$ denotes the time needed for one exponentiation on $\mathbb{G}$.*

The proofs for Theorem 2 and Theorem 3 are given in Appendix C and Appendix D respectively, for lack of space.

**Table 2.** Performance Comparison

| Schemes | Initiator's Cost | Matcher's Cost | Verifier's Cost | Signature Size | Keystone Size |
|---|---|---|---|---|---|
| CS [7] | $2.41E$ | $2.41E$ | $2.5E$ | $3\|q\|$ | $\|q\|$ |
| PCS1 [17] | $9.41E$ | $3.41E$ | $7.98E$ | $3\|q\|$ | $4\|q\| + 2\|p\|$ |
| iPCS1 [19] | $3.41E$ | $4.41E$ | $2.5E$ | $3\|q\|$ | $2\|q\|$ |
| ACS [13]* | $4.16E$ | $3E$ | $4.16E$ | $\|p\| + \|q\|$ | $\|p\|$ |
| CS-FNBC [21] | $5.41E$ | $5.41E$ | $6.5E$ | $3\|q\|$ | $\|p\| + \|q\|$ |
| PCS2 [17] | $6P + 3.41E + 1M$ | $6P + 3.41E$ | $7P + 3.5E$ | $3\|q\|$ | $2\|q\|$ |
| iPCS2 [19] | $6P + 3.41E$ | $6P + 3.41E + 1M$ | $6P + 2.5E$ | $3\|q\|$ | $2\|q\|$ |
| ID-PCS1 [8] | $3P$+$2SM$+$1E$+$2M$ | $3P$+$2SM$+$1E$+$1M$ | $2SM + 2E$ | $\|\mathbb{G}\| + 2\|q\|$ | $2\|q\|$ |
| ID-PCS2 [8] | $3P + 2SM + 2M$ | $3P + 2SM + 1M$ | $4P + 2SM$ | $3\|\mathbb{G}\|$ | $2\|\mathbb{G}\|$ |
| Ours | $3P + 1SM + 3M$ | $5P + 1SM + 1M$ | $6P + 2M$ | $2\|\mathbb{G}\|$ | $2\|\mathbb{G}\|$ |

* ACS defines two signing algorithms for initiator and matcher respectively. For consistency, the table only shows the signature size of initiator in ACS.

## 5 Performance and Comparison

Table 2 compares the performance of the proposed concurrent signature with existing schemes. In the table, "E" denotes a modular exponentiation in $\mathbb{Z}_p^*$ or $\mathbb{G}_T$ of order $q$, "M" denotes a scalar multiplication in $\mathbb{G}$, "SM" denotes a simultaneous scalar multiplication of the form $aP + bQ$ in $\mathbb{G}$, and "P" denotes a computation of the bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. The simultaneous exponentiations of the form $g_1^{x_1} g_2^{x_2}$ and $g_1^{x_1} g_2^{x_2} g_3^{x_3}$ are optimized to about 1.16 and 1.25 single exponentiation operations respectively, by the method of an exponent array [2]. Point additions in $\mathbb{G}$ and hash evaluations are not taken into account since their influence on the time complexity is negligible compared to other operations.

"Initiator's Cost" denotes the total computational cost of the initiator $A$ in one session of concurrent signature protocol. Generally, it covers the cost of generating the keystone and keystone fix $(s, f)$, producing the ambiguous signature $\sigma_A$, and verifying $\sigma_B$ received from $B$. In most of perfect concurrent signature schemes [17, 19, 8], it also includes the cost for validating $B$'s keystone fix $f'$ derived from $f$. "Matcher's Cost" is similarly defined for the matcher $B$, except that it does not necessarily cover the cost of generating keystone. "Verifier's Cost" denotes the cost for verifying the concurrent signatures $(s, f, \sigma_A)$ and $(s, f, \sigma_B)$. "Signature Size" (resp. "Keystone Size") denotes the number of bits of an ambiguous signature (resp. of a keystone fix).

From Table 2, we can see that the proposed scheme without random oracles has comparable (or even better) efficiency to the other pairing based concurrent signature schemes, like PCS2 [17], iPCS2 [19], ID-PCS1, ID-PCS2 [8], which are only proven secure in the random oracle model. The new scheme requires the least computations with respect to Initiator's Cost, and outperforms PCS2, iPCS2 with respect to Matcher's Cost or Verifier's Cost. The signature size is only 342 bits when we set $|q| = 170$ bits and $|\mathbb{G}| = 171$ bits, which is approximately 2/3 as short as the other signatures. Besides, there is no increase in the keystone size. On the other hand, the proposed scheme has a longer public key of size

linear to the security parameter $l$, which is a common feature for constructions that are free of random oracles based on Waters' approach [20].

## 6 Conclusion and Future Work

A novel ambiguity model that provides anonymity for concurrent signatures is proposed in this paper, together with a concrete scheme under the new model. This is the first concurrent signature scheme proven secure without random oracle heuristic.

There are two interesting open problems along this research line: (1) extend our security model for (two-party) concurrent signatures to the multi-party setting where the fair exchange of signatures may take place among more than two parties, and (2) construct efficient concurrent signature schemes in the extended model. We leave these topics as our future work.

## References

1. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of signatures. In: EUROCRYPT 1998. LNCS, vol. 1403, pp. 591–606. Springer (1998)
2. Ateniese, G.: Efficient verifiable encryption (and fair exchange) of digital signatures. In: CCS 1999. pp. 138–146 (1999)
3. Baum-Waidner, B., Waidner, M.: Round-optimal and abuse free optimistic multi-party contract signing. In: ICALP 2000. LNCS, 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 Press (1993)
5. Bender, A., Katz, J., Morselli, R.: Ring Signatures: Stronger Definitions, and Constructions Without Random Oracles. In: TCC 2006. LNCS, vol. 3876, pp. 60–79. Springer (2006)
6. Canetti, R., Goldreich, O., Halevi, S.: The random rracle methodology, revisited. In: STOC 1998. pp. 209–218. ACM Press (1998)
7. Chen, L., Kudla, C., Paterson, K.: Concurrent signatures. In: EUROCRYPT 2004. LNCS, vol. 3027, pp. 287–305. Springer (2004)
8. Chow, S., Susilo, W.: Generic Construction of Identity-Based Perfect Concurrent Signatures. In: ICICS 2005. LNCS, vol. 3783, pp. 194–206. Springer (2005)
9. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Communications of the ACM 28(6), 637–647 (1985)
10. Garay, J., Pomerance, C.: Timed Fair Exchange of Standard Signatures: [Extended Abstract]. In: FC 2003. LNCS, vol. 2742, pp. 190–207. Springer (2003)
11. Goldreich, O.: A simple protocol for signing contracts. In: CRYPTO 1983. pp. 133–136. Springer (1983)
12. Huang, Q., Yang, G.M., Wong, D.S., Susilo, W.: Ambiguous Optimistic Fair Exchange. In: ASIACRYPT 2008. LNCS, vol. 5350, pp. 74–89. Springer (2008)
13. Nguyen, K.: Asymmetric concurrent signatures. In: ICICS 2005. LNCS, vol. 3783, pp. 181–193. Springer (2005)
14. Pointcheval, D., Stern, J.: Security Arguments for Digital Signatures and Blind Signatures. Journal of Cryptology 13(3), 361–396 (2000)

15. Schnorr, C.P.: Effcient signature generation for smart cards. Journal of Cryptology 4(3), 239–252 (1991)
16. Susilo, W., Mu, Y.: Tripartite Concurrent Signatures. In: IFIP/SEC 2005. LNCS, vol. 181, pp. 425–441. Springer (2005)
17. Susilo, W., Mu, Y., Zhang, F.: Perfect concurrent signatures schemes. In: ICICS 2004. LNCS, vol. 3269, pp. 14–26. Springer (2004)
18. Tonien, D., Susilo, W., Safavi-Naini, R.: Multi-party concurrent signatures. In: ISC 2006. LNCS, vol. 4176, pp. 131–145. Springer (2006)
19. Wang, G., Bao, F., Zhou, J.: The fairness of perfect concurrent signatures. In: ICICS 2006. LNCS, vol. 4307, pp. 435–451. Springer (2006)
20. Waters, B.: Efficient identity-based encryption without random oracles. In: EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer (2005)
21. Yuen, T.H., Wong, D.S., Susilo, W., Huang, Q.: Concurrent signatures with fully negotiable binding control. In: ProvSec 2011. LNCS, vol. 6980, pp. 170–187. Springer (2011)

## A Analysis of Nguyen's scheme

Nguyen's asymmetric concurrent signature scheme is defined by the following eight PPT algorithms, essentially the same as the formalization in [13]. Note that the scheme uses distinct ambiguous signature generation algorithms: $\mathsf{IASign}$ for the initiator and $\mathsf{MASign}$ for the matcher. There are no keystone generation algorithm in the scheme, since that the ambiguous signature generation algorithms are also responsible for generating keystones and keystone fixes.

- $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$. The setup algorithm selects two primes $p$ and $q \geq 2^k$ such that $q|(p-1)$, and a generator $g$ for the subgroup $\langle g \rangle$ in $\mathbb{Z}_p^*$ of order $q$. The message space $\mathcal{M} = \{0,1\}^*$, the keystone space $\mathcal{S} = \mathbb{Z}_q$ and the keystone fix space $\mathcal{F} = \langle g \rangle$. Let $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ be a hash function modeled as a random oracle. Define $\mathsf{param} = (p, q, g, \mathcal{M}, \mathcal{S}, \mathcal{F}, H)$.
- $(pk, sk) \leftarrow \mathsf{SKGen}(\mathsf{param})$. The key generation algorithm picks $\alpha \overset{R}{\leftarrow} \mathbb{Z}_q$, and outputs $pk = g^\alpha$, $sk = \alpha$.
- $\sigma_i \leftarrow \mathsf{IASign}(\mathsf{param}, m, sk_i)$. The initiator's ambiguous signature generation algorithm picks $r \overset{R}{\leftarrow} \mathbb{Z}_q$, computes $c = H(g^r, m)$, $f = g^{r+c \cdot sk_i}$, and outputs $\sigma_i = (f, c)$. Here $f$ is the keystone fix for the initiator, the keystone $s = r + c \cdot sk_i$ is implicitly defined and kept secret.
- $\sigma_j \leftarrow \mathsf{MASign}(\mathsf{param}, m, sk_j)$. The matcher's ambiguous signature generation algorithm picks $r' \overset{R}{\leftarrow} \mathbb{Z}_q$, computes $f' = f^{sk_j}$, $c' = H(g^{r'} f', m)$, $k_1 = (r' - c')/sk_j$, and outputs $\sigma_j = (f', k_1, c')$. Here $f'$ is the keystone fix for the matcher.
- $1/0 \leftarrow \mathsf{IAVer}(\mathsf{param}, \sigma_i, pk_i, m)$. The initiator's ambiguous signature verification algorithm parses $\sigma_i = (f, c)$ and outputs 1 if $c = H(pk_i^{-c} f, m)$; otherwise, it outputs 0.
- $1/0 \leftarrow \mathsf{MAVer}(\mathsf{param}, \sigma_j, pk_j, m)$. The matcher's ambiguous signature verification algorithm parses $\sigma_j = (f', k_1, c')$ and outputs 1 if $c' = H(g^{c'} pk_j^{k_1} f', m)$; otherwise, it outputs 0.

- $1/0 \leftarrow \mathsf{IVer}(\mathsf{param}, s, \sigma_i, pk_i, m)$. The initiator's concurrent signature verification algorithm parses $\sigma_i = (f, c)$ and outputs 1 if both $f = g^s$ and $\mathsf{IAVer}(\mathsf{param}, \sigma_i, pk_i, m) = 1$ hold; otherwise, it outputs 0.
- $1/0 \leftarrow \mathsf{MVer}(\mathsf{param}, s, \sigma_j, pk_j, m)$. The matcher's concurrent signature verification algorithm parses $\sigma_j = (f', k_1, c')$ and outputs 1 if both $f' = pk_j^s$ and $\mathsf{MAVer}(\mathsf{param}, \sigma_j, pk_j, m) = 1$ hold; otherwise, it outputs 0.

Below we analyze Nguyen's concurrent signature scheme in our proposed ambiguity model $\mathsf{AMB}_A$.

*Claim.* Nguyen's asymmetric concurrent signature is not anonymous in $\mathsf{AMB}_A$.

*Proof.* Given two ambiguous signatures $(\sigma_A, \sigma_B)$ produced by Nguyen's scheme, we constructs an adversary $\mathcal{A}$ with the knowledge of $sk_B$, in particular, the matcher $B$ or who corrupted $B$, to win the $\mathsf{AMB}_A$ game with advantage of $1/2 - 1/2q$.

In the ambiguity game, $\mathcal{A}$ does not need to make any query to $\mathsf{KFGenOracle}$ or $\mathsf{KSReleaseOracle}$. Suppose the challenger $\mathcal{C}$ outputs $\sigma_A = (f, c)$, $\sigma_B = (f', k_1, c')$. Note that there are two co-related keystone fixes in Nguyen's scheme, where $B$'s keystone fix $f'$ is derived from $A$'s keystone fix $f$. $\mathcal{A}$ simply checks if $f^{sk_B} = f'$. If the checking holds, $\mathcal{A}$ outputs $b' = 0$; otherwise, $\mathcal{A}$ outputs $b' = 1$.

We analyze the success probability of $\mathcal{A}$ by the following two cases:

- *Case 1*: When $(\sigma_A, \sigma_B)$ are honestly produced by $A$ and $B$ respectively, we can see that the checking always holds, referring to the $\mathsf{MASign}$ algorithm.
- *Case 2*: Otherwise, the ambiguous signatures are produced by the $\mathsf{ASignSim}$ algorithm of Nguyen's concurrent signature as below. Note that the keystone fixes $f$, $f'$ are embedded in $\sigma_A$, $\sigma_B$ respectively.
  $(\sigma_A, \sigma_B) \leftarrow \mathsf{ASignSim}(\mathsf{param}, pk_A, pk_B)$
  - Pick $r \overset{R}{\leftarrow} \mathbb{Z}_q$, compute $c = H(pk_A^r, m)$, $f = pk_A^{r+c}$, and $\sigma_A = (f, c)$ where $f$ is the keystone fix of $A$.
  - Pick $r', k_1 \overset{R}{\leftarrow} \mathbb{Z}_q$, compute $c' = H(g^{r'} pk_B^{k_1}, m)$, $f' = g^{r'-c'}$, and $\sigma_B = (f', k_1, c')$ where $f'$ is the keystone fix of $B$.

  So $\mathcal{A}$ actually verifies if $pk_A^{(r+c)sk_B} = g^{r'-c'}$. The equation holds with probability about $1/q$, since $r, r'$ are uniformly picked from $\mathbb{Z}_q$ and $c, c'$ are outputs of the random oracle $H$.

Therefore, the advantage of $\mathcal{A}$ winning the game is non-negligible:

$$\begin{aligned} |\Pr[b' = b] - \frac{1}{2}| &= |\Pr[b' = b | b = 0]\Pr[b = 0] + \Pr[b' = b | b = 1]\Pr[b = 1] - \frac{1}{2}| \\ &= |\frac{1}{2} + \frac{1}{2}(1 - \frac{1}{q}) - \frac{1}{2}| = \frac{1}{2}(1 - \frac{1}{q}) \end{aligned}$$

□

## B Assumptions

The unforgeability and fairness of the proposed scheme are based on the hardness of the Computational Diffie-Hellman (CDH) Problem and the Discrete Logarithm (DL) Problem respectively. Below are the definitions of these two standard complexity assumptions. Note that CDH Assumption implies DL Assumption.

**Definition 4 (CDH Assumption).** *Let $\mathbb{G}$ be a cyclic group of prime order $q$ where $|q| \geq k$. The CDH problem is that given $g, g^a, g^b \in \mathbb{G}$, compute $g^{ab} \in \mathbb{G}$. We say that $(\epsilon', t')$-CDH assumption holds in $\mathbb{G}$ if no PPT machine can solve this problem in time $t'$ with probability at least $\epsilon'$.*

**Definition 5 (DL Assumption).** *Let $\mathbb{G}$ be a cyclic group of prime order $q$ where $|q| \geq k$. The DL problem is that given $g, g^a \in \mathbb{G}$, compute $a \in \mathbb{Z}_q$. We say that $(\epsilon', t')$-DL assumption holds in $\mathbb{G}$ if no PPT machine can solve this problem in time $t'$ with probability at least $\epsilon'$.*

## C Proof of Theorem 2

*Proof.* We prove that the simulated signatures and keystone fix $(f, \sigma_i, \sigma_j)$ output by $\mathsf{ASignSim}$ are information-theoretically indistinguishable from that generated honestly by participants of $pk_i$ and $pk_j$.

First, it is obvious to see that no matter $\mathcal{C}$ tosses a bit $b = 0$ or 1, the following checking always holds for the challenge $(f, \sigma_i, \sigma_j)$: (1) $\mathsf{KFVer}(\mathsf{param}, f) = 1$; (2) $\mathsf{AVer}(\mathsf{param}, pk_i, pk_j, f, \sigma_i, m) = 1$; (3) $\mathsf{AVer}(\mathsf{param}, pk_j, pk_i, f, \sigma_j, m) = 1$.

Parse $f = (\rho, \tau)$, $\sigma_i = (\zeta_i, \eta_i)$, $\sigma_j = (\zeta_j, \eta_j)$. Denote $s = \log_h \rho$, $s' = \log_g \rho$. From checking (1), we have:

$$e(\rho, \tau) = e(h^s, \tau) = e(g^{s'}, \tau) = e(g, h)$$

Hence, $g = \tau^s$ and $h = \tau^{s'}$. Then $\tau = g^{s^{-1}} = h^{s'^{-1}}$. So $f$ could be either produced by $\mathsf{KFGen}$ using the keystone $s$ as $(h^s, g^{s^{-1}})$, or generated by $\mathsf{ASignSim}$ using $s'$ as $(g^{s'}, h^{s'^{-1}})$. Below we check the consistency of $f$ with $\sigma_i$, $\sigma_j$ for either cases.

From checking (2), we have:

$$e(g, \zeta_i) = e(g^{sk_i}, \rho) e(\eta_i, u_i' \prod_{t \in \tilde{M}_i} u_{it}) = e(\delta_i, g^{s'}) e(\eta_i, u_i' \prod_{t \in \tilde{M}_i} u_{it})$$

Denote $r_i = r_i' = \log_g \eta_i$, we have $\zeta_i = \rho^{sk_i} (u_i' \prod_{t \in \tilde{M}_i} u_{it})^{r_i} = \delta_i^{s'} (u_i' \prod_{t \in \tilde{M}_i} u_{it})^{r_i'}$. So $\sigma_i$ could be either produced by $\mathsf{ASign}$ using $sk_i$ as $(\rho^{sk_i} (u_i' \prod_{t \in \tilde{M}_i} u_{it})^{r_i}, g^{r_i})$, or produced by $\mathsf{ASignSim}$ using $s'$ as $(\delta_i^{s'} (u_i' \prod_{t \in \tilde{M}_i} u_{it})^{r_i'}, g^{r_i'})$. By a similar analysis from checking (3), we have that $\sigma_j$ could be either produced by $\mathsf{ASign}$ using $sk_j$, or produced by $\mathsf{ASignSim}$ using $s'$.

Since that a challenge $(f, \sigma_i, \sigma_j)$ given by $\mathcal{C}$ is a valid output by either the case when $b = 0$ or the case when $b = 1$, the probability for the adversary $\mathcal{A}$ to guess $b' = b$ is no better than $1/2$, say $\mathsf{Adv}_{\mathsf{CS}}^{\mathsf{AMBI},\mathcal{A}} = |\Pr[b' = b] - 1/2| = 0$. □

## D Proof of Theorem 3

*Proof.* Suppose there exists a PPT adversary $\mathcal{A}$ who $(\epsilon, t, q_f, q_r)$-breaks the fairness of the proposed scheme, then it wins the game either by condition (1) or condition (2). We analyze these two cases as below.

– *Case 1*: If $\mathcal{A}$ wins by condition (1), i.e. breaking the Binding after Releasing property of the proposed scheme, we construct an adversary $\mathcal{B}$ who plays the fairness game with $\mathcal{A}$ and outputs the solution of a DL instance $(\mathbb{G}, q, g, A = g^a)$ as follows:

**Setup**: $\mathcal{B}$ sets $(\mathbb{G}, q, g)$ as in the DL instance, sets $h = A^b$ where $b \xleftarrow{R} \mathbb{Z}_q$, and follows $\mathsf{Setup}$ algorithm to generate the other parameters in $\mathsf{param}$. Then $\mathcal{B}$ runs $\mathsf{SKGen}$ to produce key pairs for all the participants, and forwards $(\mathsf{param}, \{(pk_i, sk_i)\})$ to $\mathcal{A}$.
**Query**: At the beginning of this phase, $\mathcal{B}$ guesses a value $1 \leq z \leq q_f$, initiates an empty table $T_f$ and a counter $c = 0$. Then it simulates the oracles as follows:

- $\mathsf{KFGenOracle}$: On each keystone generation query, $\mathcal{B}$ increments $c = c+1$. If $c \neq z$, $\mathcal{B}$ runs $(s_c, f_c) \leftarrow \mathsf{KFGen}(\mathsf{param})$, inserts $(s_c, f_c)$ into $T_f$, then returns $f_c$. If $c = z$, $\mathcal{B}$ sets $f_z = (g^b, A)$.
  The simulation for $f_z$ is perfect, since that:
  $$g^b = (g^{ab})^{a^{-1}} = h^{a^{-1}} = h^s, \quad A = g^a = g^{s^{-1}}$$
  where the keystone corresponding to $f_z$ is $s = a^{-1}$.
- $\mathsf{KSReleaseOracle}$. If the queried keystone fix $f \neq f_z$, $\mathcal{B}$ returns $s$ if there is an entry $(s, f)$ in $T_f$, otherwise returns $\perp$. $\mathcal{B}$ aborts if $f = f_z$.

**Output**: Finally $\mathcal{A}$ outputs a binding signature $\sigma_{i^*}$ on $(pk_{i^*}, pk_{j^*}, s^*, f^*, m^*)$ after making a set of adaptive queries. If $f^* \neq f_z$, $\mathcal{B}$ aborts; if $f^* = f_z$, $\mathcal{B}$ outputs $a = (s^*)^{-1}$.
When both the following events happen, $\mathcal{B}$ can complete the simulation without abort:
(1) $Q^*$: $Y^* = f_z$, where $Y^* = f^*$.
(2) $Q_\xi$: $Y_\xi \neq f_z$, for any $Y_\xi = f$ queried to $\mathsf{KSReleaseOracle}$.
Note that there are totally $q_r$ queries made in event (2). Besides, we have $\Pr[\neg Q_\xi | Q^*] = 0$ since that it is not allowed to query $f = f_z$ to $\mathsf{KSReleaseOracle}$ when $f^* = f_z$.
So the probability that $\mathcal{B}$ does not abort is:

$$\Pr[\neg\mathsf{abort}] = \Pr[\bigwedge_{\xi=1}^{q_r} Q_\xi \wedge Q^*] = \Pr[Q^*]\Pr[\bigwedge_{\xi=1}^{q_r} Q_\xi | Q^*]$$
$$\geq \frac{q_f - q_r}{q_f}(1 - \sum_{\xi=1}^{q_r} \Pr[\neg Q_\xi | Q^*]) = 1 - \frac{q_r}{q_f}$$

In the simulation, exponentiation is the dominant operation for time complexity. For each query to $\mathsf{KFGenOracle}$ when $c \neq z$, it requires 2 exponentiations. When $c = z$, the query to $\mathsf{KFGenOracle}$ requires 1 exponentiation. Therefore, we have $t' = t + O((2q_f - 1)t_e)$.

– *Case 2*: Otherwise, $\mathcal{A}$ breaks the Concurrency of Binding property by winning condition (2). Since $\mathsf{Ver}(\mathsf{param}, pk_{i^*}, pk_{j^*}, s^*, f^*, \sigma_{i^*}, m^*) = 1$, we have $\rho^* = h^{s^*}$ and $\tau^* = g^{(s^*)^{-1}}$ where $f^* = (\rho^*, \tau^*)$. Besides, we have $\mathsf{AVer}(\mathsf{param}, pk_{j^*}, pk_{i^*}, f^*, \sigma_{j^*}, m^*) = 1$, so $\mathsf{Ver}(\mathsf{param}, pk_{j^*}, pk_{i^*}, s^*, f^*, \sigma_{j^*}, m^*) = 1$, according to the construction of the algorithm $\mathsf{Ver}$ in Section 4.1. Hence, even assuming the adversary has unlimited computation power, this case could never happens.

Combining the results in Case 1 and Case 2, the theorem follows. □