TRIPARTITE CONCURRENT SIGNATURES

Willy Susilo and Yi Mu
Centre for Information Security Research
School of Information Technology and Computer Science
University of Wollongong
Wollongong 2522, Australia
Email: {wsusilo, ymu}@uow.edu.au

Abstract: Fair exchange in digital signatures has been considered as a fundamental problem in cryptography. The notion of concurrent signatures was introduced in the seminal paper of Chen, Kudla and Paterson in Eurocrypt 2004 Chen et al., 2004. In this paper, we partially solve an open problem proposed in Chen et al., 2004. We extend the notion of two party concurrent signatures to tripartite concurrent signature schemes. In tripartite concurrent signatures, three parties can exchange their signatures in such a way that their signatures will be binding *concurrently*. We present a model of tripartite concurrent signatures together with a concrete scheme based on bilinear pairings. It was noted in Chen et al., 2004 that extending concurrent signatures to a multi-party scheme, where there are three or more participants, cannot be achieved by trivially modifying their construction in Chen et al., 2004.

Key words: Tripartite Concurrent Signatures, Multi-party Fair Exchange.

1. INTRODUCTION

Fair exchange in digital signatures has been considered as a fundamental problem in cryptography. Fair exchange is a necessary feature in many applications for electronic commerce. Typical applications include contract signing where two parties need to exchange their signature on a contract.

Two party fair exchange has been studied extensively in the literature. In general, the method can be broadly divided into two types, namely *with* or *without* a trusted party *TTP*. It was believed that fair exchange without a TTP is not practical, since it requires a large number of communication rounds, until the recent work of Chen, Kudla and Paterson in Chen et al., 2004 that shows a *weaker* version of two party fair exchange can be done efficiently *without* any involvement of a TTP. In concurrent signatures, two parties can produce two signatures in such a way that from any third party's point of view, both signatures are ambiguous. However, after additional information, called the *keystone*, is released by one of the parties, both signatures are binding concurrently. It was noted in Chen et al., 2004 that this type of signature scheme falls just short of providing a full solution to the problem of fair exchange of signatures. In the same paper, they questioned the existence of multi party concurrent signatures. They noted that if multi party concurrent signatures can be constructed and modeled correctly, this will move closer to the full solution of multi party fair exchange. They also mentioned that their scheme *cannot* be trivially extended to include multiple matching signers, since the fairness of the scheme will not be achieved.

Our Contribution

In this paper, we present a novel model of tripartite concurrent signatures that allows three parties to exchange their signatures in a fair way. Our model guarantees *fairness* as in the seminal paper of Chen et al., 2004. We also provide a concrete scheme that satisfies our model, based on bilinear pairings. We provide a set of security analysis for our concrete scheme.

1.1 Related Work

In Rivest et al., 2001, the notion of *ring signatures* was formalized and an efficient scheme based on RSA was proposed. This signature can be used to convince any third party that one of the people in the group (who know the trapdoor information) has authenticated the message on behalf of the group.

The authentication provides *signer ambiguity*, in the sense that no one can identify who has actually signed the message.

Designated Verifier Proofs were proposed in Jakobsson et al., 1996. The idea is to allow signatures to convince only the intended recipient, who is assumed to have a public-key. As noted in Rivest et al., 2001, ring signature schemes can be used to provide this mechanism by joining the verifier in the ring. However, it might not be practical in the real life since the verifier might not have any public key setup. In Desmedt, 2003, Desmedt raised the problem of generalizing the designated verifier signature concept to a multi designated verifier scheme. This question was answered affirmatively in Laguillaumie and Vergnaud, 2004, where a construction of multi designated verifiers signature scheme was proposed.

2. PRELIMINARIES

2.1 Basic concepts on Bilinear Pairings

Let G_1, G_2 be cyclic additive groups generated by P_1, P_2, respectively, whose order are a prime q. Let G_M be a cyclic multiplicative group with the same order q. We assume there is an isomorphism $\Psi : G_2 \to G_1$ such that $\Psi(P_2) = P_1$. Let $\hat{e} : G_1 \times G_2 \to G_M$ be a bilinear mapping with the following properties:

1. *Bilinearity*: $\hat{e}(aP, bQ) = \hat{e}(P,Q)^{ab}$ for all $P \in G_1, Q \in G_2, a, b \in Z_q$.
2. *Non-degeneracy*: There exists $P \in G_1, Q \in G_2$ such that $\hat{e}(P,Q) \neq 1$.
3. *Computability*: There exists an efficient algorithm to compute $\hat{e}(P,Q)$ for all $P \in G_1, Q \in G_2$.

For simplicity, hereafter, we set $G_1 = G_2$ and $P_1 = P_2$. We note that our scheme can be easily modified for a general case, when $G_1 \neq G_2$.

Bilinear pairing instance generator is defined as a probabilistic polynomial time algorithm $\mathcal{IG}$ that takes as input a security parameter ℓ and returns a uniformly random tuple $param = (p, G_1, G_M, \hat{e}, P)$ of bilinear parameters, including a prime number p of size ℓ, a cyclic additive group G_1 of order q, a multiplicative group G_M of order q, a bilinear map $\hat{e} : G_1 \times G_1 \to G_M$ and a generator P of G_1. For a group G of prime order, we denote the set $G^* = G \setminus \{\mathcal{O}\}$ where $\mathcal{O}$ is the identity element of the group.

2.2 Complexity Assumptions

Definition 1. Computational Diffie-Hellman (CDH) Problem.

Given two randomly chosen $aP, bP \in G_1$, *for unknown* $a, b \in Z_q$, *compute* $Z = abP$.

Definition 2. Computational Diffie-Hellman (CDH) Assumption.

If $\mathcal{IG}$ *is a CDH parameter generator, the advantage* $\mathrm{Adv}_{\mathcal{IG}}(\mathcal{A})$ *that an algorithm* $\mathcal{A}$ *has in solving the CDH problem is defined to be the probability that the algorithm* $\mathcal{A}$ *outputs* $Z = abP$ *on inputs, where* $(G_1, G_M, \hat{e})$ *is the output of* $\mathcal{IG}$ *for sufficiently large security parameter* ℓ, P *is a random generator of* G_1 *and* a, b *are random elements of* Z_q. *The CDH assumption is that* $\mathrm{Adv}_{\mathcal{IG}}(\mathcal{A})$ *is negligible for all efficient algorithms* $\mathcal{A}$.

2.3 Signature of Knowledge

The first signature based on proof of knowledge (SPK) was proposed in Camenisch, 1998 ; Camenisch, 1997. We will use the following definition of SPK from Camenisch, 1998.

Let q be a large prime and p=2q+1 be also a prime. Let G be a finite cyclic group of prime order p. Let g be a generator of Z_p^* such that computing discrete logarithms of any group elements (apart from the identity element) with respect to one of the generators is infeasible. Let $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$ denote a strong collision-resistant hash function.

Definition 3. *A pair* $(c, s) \in \{0,1\}^\ell \times Z_q$ *satisfying* c=H $(g||y||g^s y^c||m)$ *is a signature based on proof of knowledge of discrete logarithm of a group element* y *to the base* g *of the message* $m \in \{0,1\}^*$ *and is denoted by* $SPK\{\alpha : y = g^\alpha\}(m)$.

A $SPK\{\alpha : y = g^\alpha\}(m)$ can only be computed if the value (secret key) $\alpha = \log_g(y)$ is known. This is also known as a non-interactive proof of the knowledge α.

Definition 4. *A pair* (c, s) *satisfying* c=$H(h||g||z||y||h^s z^c||g^s y^c||m)$ *is a signature of equality of the discrete logarithm problem of the group element* z *with respect to the base* h *and the discrete logarithm of the group element* y *with respect to the base* g *for the message* m. *It is denoted by* $SPKEQ\{\alpha : y = g^\alpha \wedge z = h^\alpha\}(m)$.

This signature of equality can be seen as two parallel signatures of knowledge $SPK\{\alpha : y = g^\alpha\}(m)$ and $SPK\{\alpha : z = h^\alpha\}(m)$, where the exponent for the commitment, challenge and response are the same. It is straightforward to see that this signature of equality can be extended to show

the equality of n parallel signatures of knowledge *SPK* using the same technique. This technique can be applied to elliptic curve domain. For completeness, we illustrate the technique as follows.

Definition 5. *A pair* (c,s) *satisfying* $c=H(P||Q||sP+cQ||m)$ *is a signature based on proof of knowledge of elliptic curve discrete logarithm of a group element* Q *to the base* P *of the message* $m \in \{0,1\}^*$ *and is denoted by* $ECSPK\{\alpha : Q = \alpha P\}(m)$.

We note that $ECSPK\{\alpha : Q = \alpha P\}(m)$ can only be computed iff the value of a, where $Q=aP$, is known. It can be computed as follows. Firstly, select a random $z \in Z_q^*$ and compute $c=H(P||Q||zP||m)$, and then, compute $s = z - ca \pmod q$. Using the same technique, the following definition can be derived.

Definition 6. *A pair (c,s) satisfying* $c=H(U||P||S||Q||sU+cS||sP+cQ||m)$ *is a signature of equality of the elliptic curve discrete logarithm problem of the group element S with respect to the base U and the discrete logarithm of the group element Q with respect to the base P for the message m. It is denoted by* $ECSPKEQ : \{\alpha : Q = \alpha P \wedge S = \alpha U\}(m)$.

3. FORMAL DEFINITIONS

3.1 Tripartite Concurrent Signature Algorithms

In this section, we provide a formal definition of a tripartite concurrent signature scheme. In our system, the three participants are polynomially bounded in the security parameter ℓ.

Definition 7. *A tripartite concurrent signature scheme is a digital signature scheme that consists of the following algorithms.*

- SETUP: *A probabilistic algorithm that on input a security parameter* ℓ*, outputs descriptions of the set of participants* $\mathcal{U}$*, the message space* $\mathcal{M}$*, the signature space* $\mathcal{M}$*, the keystone space* $\mathcal{K}$*, the keystone fix space* $\mathcal{F}$ *and a function* $KGEN : \mathcal{K} \rightarrow \mathcal{F}$ *.The algorithm also outputs the public parameters* **param***, together with all public keys of the participants* $\{\mathcal{P}_i\}$*, where each participant retaining their private key* s_i.
- ASIGN: *A probabilistic algorithm that on inputs* $(m, f, \mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k, s_i)$*, where* $f \in \mathcal{F}, m \in \mathcal{M}$. $\mathcal{P}_i, \mathcal{P}_j$ *and* $\mathcal{P}_k$ *are the participants' public keys and* s_i *is the associated secret key for public key* $\mathcal{P}_i$*, outputs an ambiguous signature* $\sigma \in S$ *on m.*

- AVERIFY: *A deterministic algorithm that on inputs* $(m, f, \sigma, \mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k)$, *where* $f \in \mathcal{F}, m \in \mathcal{M}$. $\mathcal{P}_i, \mathcal{P}_j$ *and* $\mathcal{P}_k$ *are the participants' public keys and* $\sigma \in S$, *outputs* accept *or* reject.
- RELEASE: *A deterministic algorithm that accepts* $f \in \mathcal{F}$ *and a set of valid signatures* $\{\sigma_i, \sigma_j, \sigma_k\}$ *for message* $\{m_i, m_j, m_k\}$ *and outputs the correct* $\hat{k} \in \mathcal{K}$ *used in the* $f = KGEN(\hat{k})$ *function, together with some necessary information,* info, *to confirm the published signatures.*
- PROOF-VERIFY: *A deterministic algorithm that accepts a keystone* $\hat{k} \in \mathcal{K}$, *a keystone fix* $f \in \mathcal{F}$, *some required information produced by the* RELEASE *algorithm,* info. *This algorithm verifies the correctness of the keystone together with* info. *If they are correct, then output* accept. *Otherwise, output* reject.
- VERIFY: *A deterministic algorithm that accepts* $(f, \hat{k}, \sigma)$ *and some necessary information produced by the* RELEASE *algorithm,* info, *and executes* PROOF-VERIFY $(\hat{k}, f,$ info$)$ *and* AVERIFY $(m, f, \sigma_u, \mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k)$ *algorithm, for* $u \in (i, j, k)$, *to produce* accept *or* reject, *respectively.*
- DENY: *A probabilistic algorithm that accepts* $(m, f, \sigma_1, \sigma_2, \mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k, s_i)$ *where* $m \in \mathcal{M}$, $f \in \mathcal{F}$, s_i *is the associated secret key for* $\mathcal{P}_i$ *and* $\sigma_1, \sigma_2 \in S$ *are signatures on m, and tests whether both signatures are valid, i.e. pass* AVERIFY *test, and confirm that one of them is a forgery. If forgery happens, then output* accept. *Otherwise, output* reject.

3.2 Tripartite Concurrent Signature Protocol

We will describe a tripartite concurrent signature protocol among three parties, Alice, Bob and Charlie (or *A*, *B* and *C*, respectively). One of the three parties needs to create a keystone and send the first ambiguous signature to the other two parties. We call this party the *initial signer*. Then, another party will respond to this initial signature by creating another ambiguous signature with the same keystone fix. We call the second party as a *first matching signer*. Finally, the third party will respond to the first two signatures by creating his own ambiguous signature. We call this party a *second matching party*. Without losing generality, we assume *A* to be the initial singer, *B* the first matching signer and *C* the second matching signer. From here on, we will use subscripts *A*, *B* and *C* to describe initial signer *A*,

first matching signer B and second matching signer C. The signature works as follows.

A, B and C run SETUP algorithm to determine the public parameters of the scheme. We assume that participants i's secret and public keys are indicated by $\mathcal{P}_i$ and s_i, respectively, for $i \in (A, B, C)$. Hence, A's public key is $\mathcal{P}_A$ and her secret key is s_A and so forth.

1. A picks random keystone $\hat{k} \in \mathcal{K}$ and computes $f = KGEN(\hat{k})$. A takes her own public key $\mathcal{P}_A$, together with the other parties' public key, $\mathcal{P}_B, \mathcal{P}_C$ and picks a message $m_A \in \mathcal{M}$ to sign. A then computes her ambiguous signatures as $\sigma_A = \mathsf{ASIGN}(m_A, f, \mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C, s_A)$ and sends this to B and C.
2. Upon receiving A's ambiguous signature σ_A, B and C verifies the signature by testing whether $\mathsf{AVERIFY}(m_A, f, \sigma_A, \mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C) = \mathsf{accept}$ holds with equality. If not, B and C abort. Otherwise, B picks a message $m_B \in \mathcal{M}$ to sign and computes his ambiguous signature $\sigma_B = \mathsf{ASIGN}(m_B, f, \mathcal{P}_B, \mathcal{P}_A, \mathcal{P}_C, s_B)$ using the same keystone fix $f \in \mathcal{F}$ and sends this to A and C.
3. Upon receiving B's ambiguous signature σ_B, A and C verifies the signature by testing whether $\mathsf{AVERIFY}(m_B, f, \sigma_B, \mathcal{P}_B, \mathcal{P}_A, \mathcal{P}_C) = \mathsf{accept}$ holds with equality. If not, A and C abort. Otherwise, C picks a message $m_C \in \mathcal{M}$ to sign and computes his ambiguous signature $\sigma_C = \mathsf{ASIGN}(m_C, f, \mathcal{P}_C, \mathcal{P}_A, \mathcal{P}_B, s_C)$ using the same keystone fix $f \in \mathcal{F}$ and sends this to A and B.
4. Upon receiving C's ambiguous signature σ_C, A and B verifies the signature by testing whether $\mathsf{AVERIFY}(m_C, f, \sigma_C, \mathcal{P}_C, \mathcal{P}_A, \mathcal{P}_B) = \mathsf{accept}$ holds with equality. If not, A and B abort. Otherwise, A executes VERIFY algorithm to release the keystone $\hat{k}$ (together with several other confirmation messages, info, whenever necessary) to B and C, and all signatures are binding concurrently.

Any third party can be convinced with the authenticity of the signatures by executing VERIFY algorithm.

3.3 Security Requirements

As the original model of concurrent signatures in Chen et al., 2004, we require a tripartite concurrent signature to satisfy *correctness*, *unforgeability*, *ambiguity* and *fairness*. Intuitively, these notions are described as follows.

- *Correctness*: If a signature σ has been generated *correctly* by invoking ASIGN algorithm on a message $m \in \mathcal{M}$, then AVERIFY algorithm will return accept with an overwhelming probability, given a signature σ on m. Moreover, after the keystone $\hat{k} \in \mathcal{K}$ is released, then the output of VERIFY algorithm will be accept with an overwhelming probability.
- *Unforgeability*: There are two different cases that we need to consider. Case 1) When an adversary $\mathcal{A}$ does not have any knowledge of the respective secret key s_i, then no valid signature that will pass the AVERIFY algorithm can be produced. Otherwise, one of the underlying hard problems can be solved by using this adversary's capability. Case 2) Any party cannot *frame* the other party that he/she has indeed signed message. We require that although both signatures are ambiguous, any party who would like to frame (or cheat) the others will not be able to produce a valid keystone with an overwhelming probability.
- *Ambiguity*: We require that given the two ambiguous signatures, any adversary will not be able to distinguish who was the actual signer of the signatures *before* the keystone is released.
- *Fairness*: We require that any valid ambiguous signatures generated using the same keystone will all become binding *after* the keystone is released. Hence, a matching signer cannot be left in a position where a keystone binds his signature to him whilst the initial signer's signature is not binding to her. Additionally, we also require that *only* the party who generates a keystone can use to create a binding signature. We do not require that the matching signers will definitely receive the necessary keystone.

Definition 8 *A tripartite concurrent signature scheme is* secure *if it is existentially unforgeable under a chosen message attack, ambiguous and fair.*

4. A CONCRETE TRIPARTITE CONCURRENT SIGNATURE SCHEME

A tripartite concurrent signature scheme is defined by the following algorithms. Our scheme is developed using the technique proposed in

Laguillaumie and Vergnaud, 2004. In the following, we denote the participants by $\mathcal{U}_i$, $i \in \{A,B,C\}$ for convenience and clarify of the presentation.

- SETUP: On input security parameter ℓ, the algorithm selects a uniformly random tuple $param = (p, G_1, G_M, \hat{e}, P)$ of bilinear parameters, including a prime number q of size ℓ, a cyclic additive group G_1 of order q, a multiplicative group G_M of order q, a bilinear map $\hat{e}: G_1 \times G_1 \rightarrow G_M$ and a generator P of G_1. The algorithm also selects a secret key $s \in Z_q^*$ and computes the associated public key $P_{pub} = sP$, for a random generator $P \in G_1$. The algorithm also publishes two cryptographic hash function $H_0 : \{0,1\}^* \rightarrow G_1$ and $H_1 : \{0,1\}^\ell \rightarrow Z_q^*$. Each user $\mathcal{U}_i \in \mathcal{U}$, $i \in \{A,B,C\}$, selects his/her secret key s_i and publishes his/her public key $\mathcal{P}_i = s_i P$. At the end of the algorithm, the parameter $param = (p, G_1, G_M, \hat{e}, P)$ is published, together with the public key P_{pub} and public key of the participants $\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C$. The algorithm also sets $\mathcal{M} = \mathcal{F} = \mathcal{K} = Z_q$. The *KGEN*(·) function is defined to be $H_1(\cdot)$.
- ASIGN: The algorithm accepts $(m, f, \mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k, s_i)$ as input, where $\mathcal{P}_i = s_i P$, for s_i is $\mathcal{U}_i$'s secret key, $\mathcal{P}_j$ and $\mathcal{P}_k$ are public keys published by $\mathcal{U}_j$ and $\mathcal{U}_k$, $m \in \mathcal{M}$ and $f \in \mathcal{F}$, and performs the following.
  - Select a random $r \in Z_q^*$.
  - Compute $M = H_0(m \| f)$
  - Compute $Q_1 = s_i^{-1}(M - r(\mathcal{P}_j + \mathcal{P}_k))$ and $Q_2 = rP$
  - Output $\sigma = (Q_1, Q_2)$ as the signature on m.
- AVERIFY: The algorithm accepts $(m, f, \sigma, \mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k)$, for $\sigma = (Q_1, Q_2)$, $m \in \mathcal{M}$, $f \in \mathcal{F}$, and verifies whether $\hat{e}(Q_1, \mathcal{P}_i)\, \hat{e}(Q_2, \mathcal{P}_j + \mathcal{P}_k) = \hat{e}(H_0(m \| f), P)$ holds. If it does not hold, then output reject. Otherwise, output accept.
- RELEASE: This algorithm accepts a keystone $\hat{k} \in \mathcal{K}$ together with a valid set of signatures $\{(m_A, (Q_1^A, Q_2^A)), (m_B, (Q_1^B, Q_2^B)), (m_C, (Q_1^C, Q_2^C))\}$ and performs the following. Hereafter, we abuse the notation $\sigma_i = (Q_1^i, Q_2^i)$ to indicate a signature that is produced by $\mathcal{U}_i$. Since $Q_1, Q_2 \in G_1$, where G_1 is an additive group, then this notation is clear from its context.

We note that each $(m_i, (Q_1^i, Q_2^i))$ will pass the AVERIFY algorithm.

- Computes $Q_{ij} = s_i Q_2^j$ and $Q_{ik} = s_i Q_2^k$, where Q_2^u denotes Q_2 that was generated by u using ASIGN algorithm (and therefore, it implies that AVERIFY$(m_i, f, \sigma_i, \mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k)$= accept holds with equality, for $\sigma_i = (Q_1^i, Q_2^i)$).
- Produces the following signatures of knowledge.

$$\Gamma = ECSPKEQ\{\alpha : Q_{ij} = \alpha Q_2^i \wedge Q_{ik} = \alpha Q_2^j\}(\varepsilon)$$

- Outputs $(f, \hat{k}, Q_{ij}, Q_{ik}, \Gamma)$, where $\hat{k} \in \mathcal{K}$ and $H_1(f) \stackrel{?}{=} \hat{k}$ holds. Notice that f is only known by the initial signer, and hence, this algorithm can only be performed correctly by the initial signer.

- PROOF-VERIFY: In the following description, the initial signer is denoted by $\mathcal{U}_i$. This algorithm accepts and verifies whether $\Gamma = ECSPKEQ\{\alpha : Q_{ij} = \alpha Q_2^i \wedge Q_{ik} = \alpha Q_2^j\}(\varepsilon)$ holds. If it does not hold, then output reject. Then, it verifies whether the following equations

$$\hat{e}(Q_1^j, \mathcal{P}_j)\hat{e}(Q_{ij}, P)\hat{e}(Q_2^j, \mathcal{P}_k) \stackrel{?}{=} \hat{e}(M, P)$$
$$\hat{e}(Q_1^k, \mathcal{P}_k)\hat{e}(Q_{ik}, P)\hat{e}(Q_2^k, \mathcal{P}_j) \stackrel{?}{=} \hat{e}(M, P)$$

hold with equality. If it does not hold, then output reject. Finally, verify whether $f = KGEN(\hat{k})$ holds. If not, output reject. Otherwise, output accept.

- VERIFY: The algorithm accepts $(m, f, k, \mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k, Q_{ij}, Q_{ik}, \Gamma)$, for $\sigma = (Q_1, Q_2), m \in \mathcal{M}, f \in \mathcal{F}, \hat{k} \in \mathcal{K}$, and performs the following verification steps.
  - test whether $H_1(f) \stackrel{?}{=} \hat{k}$ holds. If not, then output reject.
  - execute PROOF-VERIFY. If not hold, then output reject.
  - execute AVERIFY with parameter $(m, f, \sigma_u, \mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k)$ for the three message-signature pairs, $u \in \{i, j, k\}$. The output of VERIFY is the output of AVERIFY algorithm.

- DENY: This algorithm accepts $(m, f, \sigma_1, \sigma_2, \mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k, s_i)$, where $m \in \mathcal{M}$, $f \in \mathcal{F}$, are signatures on m, u = {1, 2}, s_i is $\mathcal{U}_i$'s secret key associated with the public key $\mathcal{P}_i$ (which implies $\mathcal{P}_i = s_i P$) and $\mathcal{P}_j, \mathcal{P}_k$ are the public keys of $\mathcal{U}_j, \mathcal{U}_k$, respectively. The algorithm performs the following.
  - Test whether AVERIFY accept$(m, f, \sigma_u, \mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k)$, for u = {1, 2}. If it does not hold, then terminate the algorithm and output reject.
  - Compute $\delta_i = s_i Q_1 - M + P$ for $M = H_0(m \| f)$.

- Perform the following verification
$$(\hat{e}(Q_2, \mathcal{P}_j + \mathcal{P}_k)\hat{e}(\delta_1, P) \stackrel{?}{=} \hat{e}(P, P))$$
for both signatures σ_1, σ_2.
- If the result of the above verification for either σ_1 or σ_2 is true, return accept with δ_1 as the proof. Otherwise, return reject.

Correctness.

The correctness of the AVERIFY algorithm is justified as follows.

$$\begin{aligned}
&\hat{e}(H_0(M \| f), P) = \hat{e}(Q_1, \mathcal{P}_i)\hat{e}(Q_2, \mathcal{P}_j + \mathcal{P}_k) \\
&= \hat{e}(s_i^{-1}(M - r(\mathcal{P}_j + \mathcal{P}_k)), \mathcal{P}_i)\hat{e}(rP, \mathcal{P}_j + \mathcal{P}_k) \\
&= \hat{e}(M - r(\mathcal{P}_j + \mathcal{P}_k), P)\hat{e}(r(\mathcal{P}_j + \mathcal{P}_k), P) \\
&= \hat{e}(M - r(\mathcal{P}_j + \mathcal{P}_k) + r(\mathcal{P}_j + \mathcal{P}_k), P) \\
&= \hat{e}(M, P) \\
&= \hat{e}(H_0(M \| f), P)
\end{aligned}$$ □

4.1 Security Analysis

Lemma 1. *If a participant A has signed a message m to generate σ_A, both B and C will be convinced with the authenticity of the signature, but no other third party will.*

Proof. When a signature $\sigma_A = (Q_1, Q_2)$ is generated, firstly either B and C needs to execute the AVERIFY algorithm. If the signature passes this test, then B and C will believe that his signature was indeed generated by A, because they have not colluded to generate this signature. We note that no other third party can be convinced with the authenticity of this signature, since if B and C collude, they can collaboratively compute $Q_1' = rP, Q_2' = (s_B + s_C)^{-1}(M - r\mathcal{P}_A)$, for a random $r \in Z_q^*$, which is valid and indistinguishable signature from any third party's point of view. Hence, the signature cannot be used to convince any other third party other than B and C. □

Theorem 1. *The* DENY *algorithm is correct and sound. This algorithm is used to protect a participant against a collusion of two malicious participants.*

The proof of this theorem is shown in terms of the following lemmas.

Lemma 2. *Any two participants can collude and frame another participant that he has signed a message.*

Proof. To show the correctness of the DENY algorithm, we need to show a successful attack that is launched by a conspiracy of two participants to frame the other participant. Without losing generality,

we assume B will conspire with C to frame A, i.e. to accuse that A had signed a message that he has not signed. The attack is as follows.

- B and C collaboratively perform the following.
  - Select a random $\hat{r} \in Z_q^*$.
  - Compute $Q_1' = \hat{r}P$
    $Q_2' = (s_B + s_C)^{-1}(M - \hat{r}\mathcal{P}_A)$ for $M = H_0(m||f)$
- Output (Q_1', Q_2') as a signature on m.

One can verify that the signature (Q_1', Q_2') will pass the AVERIFY algorithm, due to the following.

$$\begin{aligned}\hat{e}(H_0(m \| f), P) &\stackrel{?}{=} \hat{e}(Q_1', \mathcal{P}_A)\hat{e}(Q_2', \mathcal{P}_B + \mathcal{P}_C) \\ &= \hat{e}(\hat{r}P, \mathcal{P}_A)\hat{e}((s_B + s_C)^{-1}(M - \hat{r}\mathcal{P}_A), \mathcal{P}_B + \mathcal{P}_C) \\ &= \hat{e}(\hat{r}P, \mathcal{P}_A)\hat{e}(M - \hat{r}\mathcal{P}_A, P) \\ &= \hat{e}(\hat{r}\mathcal{P}_A, P)\hat{e}(M - \hat{r}\mathcal{P}_A, P) \\ &= \hat{e}(\hat{r}\mathcal{P}_A + M - \hat{r}\mathcal{P}_A, P) \\ &= \hat{e}(M, P) \\ &= \hat{e}(H_0(m \| f), P)\end{aligned}$$

Lemma 3. *Any collusion attack can be prevented by performing the* DENY algorithm.

Proof. As illustrated in Lemma 2, a valid signature (Q_1', Q_2') can be generated by a collusion of two participants. The signature will be in one of the following forms.

- $(\hat{r}P, (s_B + s_C)^{-1}(M - \hat{r}\mathcal{P}_A))$.
- $(-\hat{r}P, (s_B + s_C)^{-1}(M + \hat{r}\mathcal{P}_A))$.

We denote the above signatures as σ_1 and σ_2. We can easily verify that both signatures will pass the AVERIFY algorithm. Now, we shall demonstrate that A can provide a proof that a forgery has happened, by performing the DENY algorithm. Basically, the algorithm will compute following.

- Compute $\delta_1 = s_A Q_1' - M + P$
- We note that the value of δ_1 will be one of the following.
  $$\delta_1 = \hat{r}\mathcal{P}_A - M + P \text{ or } \delta_1 = P - \hat{r}\mathcal{P}_A - M$$
  depending on the forged signature above.

When the conspiracy happens, one of the following tests will return true.

$$(\hat{e}(Q_2, \mathcal{P}_B + \mathcal{P}_C)\hat{e}(\delta_1, P)) \stackrel{?}{=} \hat{e}(P, P)$$

for σ_1 and σ_2. This is due to

$$\begin{aligned}\hat{e}(P,P) &= (\hat{e}(Q_2, \mathcal{P}_B + \mathcal{P}_C)\hat{e}(\delta_1, P)) \\ &= (\hat{e}((s_B + s_C)^{-1}(M - \hat{r}\mathcal{P}_A), \mathcal{P}_B + \mathcal{P}_C)\hat{e}(\hat{r}\mathcal{P}_A - M + P, P)) \\ &= (\hat{e}(M - \hat{r}\mathcal{P}_A, P)\hat{e}(\hat{r}\mathcal{P}_A - M + P, P)) \\ &= (\hat{e}(P,P))\end{aligned}$$

$$\begin{aligned}\hat{e}(P,P) \stackrel{?}{=} &(\hat{e}(Q_2, \mathcal{P}_B + \mathcal{P}_C)\hat{e}(\delta_1, P)) \\ &= (\hat{e}((s_B + s_C)^{-1}(M + \hat{r}\mathcal{P}_A), \mathcal{P}_B + \mathcal{P}_C)\hat{e}(P - \hat{r}\mathcal{P}_A - M, P)) \\ &= (\hat{e}(M + \hat{r}\mathcal{P}_A, P)\hat{e}(P - \hat{r}\mathcal{P}_A - M, P)) \\ &= (\hat{e}(P,P))\end{aligned}$$

We note that if the signature is not forged (i.e. generated by A), then the above verification will not return true. Hence, the DENY algorithm will return true iff forgery has happened due to collusion of two participants. □

Theorem 2. (**Ambiguity**) *Before the keystone is released using the* RELEASE *algorithm, both signature are ambiguous.*

Proof. As shown in the proof of Lemma 2, a collusion of two participants can always produce a set of signatures that will pass the verification AVERIFY algorithm. We will illustrate this attack as follows. Without losing generality, we assume A colludes with C. Firstly, A produces (Q_1^A, Q_2^A) by herself, and then collaboratively with C, they can produce $(Q_1^{B'}, Q_2^{B'})$ and claim that this signature was indeed signed by B. Finally, C can produce (Q_1^C, Q_2^C). We note that all the signatures are produced using a legitimate ASIGN algorithm. From any third party's point of view, these signatures are indistinguishable from a set of signatures that are genuinely created by the three participants. In this scenario, before the keystone is released using the RELEASE algorithm, B can always invoke DENY algorithm at any time to deny that he has not signed the message. Hence, the signatures are ambiguous from any third party's point of view. We also note that A and C cannot collude and frame that B has signed a message that he has not signed, as A cannot invoke the RELEASE algorithm correctly since $(Q_1^{B'}, Q_2^{B'})$ are not in the "correct" form, i.e. $(s_B{}^{-1}(M - r(\mathcal{P}_A + \mathcal{P}_C)), rP)$.

Lemma 4. *When the output of* VERIFY *is* accept, *then any third party can be sure who has generated the signature.*

Proof. We note that VERIFY algorithm will test three different components. The first component is to verify whether the keystone $\hat{k}$ is generated correctly, using the ***KGEN*** function. The second verification is to make sure that the published signatures are in the correct form. This is guaranteed with the following test

$$\hat{e}(Q_1^j, \mathcal{P}_j)\hat{e}(Q_{ij}, P)\hat{e}(Q_2^j, \mathcal{P}_k) \stackrel{?}{=} \hat{e}(M, P)$$
$$\hat{e}(Q_1^k, \mathcal{P}_k)\hat{e}(Q_{ik}, P)\hat{e}(Q_2^k, \mathcal{P}_j) = \hat{e}(M, P)$$

We note that the above tests will be satisfied, iff $Q_{ij} = s_i Q_2^j$ and $Q_{ik} = s_i Q_2^k$ hold. Finally, the last verification will confirm that the signatures were indeed generated correctly by each participant, so that they will pass the verification test. Hence, any third party can be convinced with the authenticity of the signature.

Theorem 3. (Fairness) *Any signature that is generated with the same keystone will be binding concurrently when the keystone is released.*

Proof. Suppose any of the participants tries to cheat by signing more than one signature. By testing the VERIFY algorithm, all the signatures will be binding concurrently. This way, the fairness is guaranteed. Moreover, as illustrated in the proof of Theorem 2, two colluding participants cannot frame another participant by generating a forged signature and later on confirm it as if it was signed by the framed participant. This is due to the inability of the initial signer to execute the RELEASE algorithm correctly.

Theorem 4. (Unforgeability) *The scheme presented in this section is existentially enforceable under a chosen message attack in the random oracle model, assuming the hardness of the Computational Diffie-Hellman problem.*

Proof. We use the notion of existential unforgeability against a chosen message attack from Chen et al., 2004. We consider an **EF-CMA** adversary $\mathcal{A}$ that outputs an existential forgery (M^*, σ^*) with probability $\mathrm{Succ}^{\mathcal{A}}_{\mathrm{EF-CMA}}(k)$ within the time t. We denote the number of queries from the random oracle $\mathcal{H}$ by $q\mathcal{H}$ and from the signing oracle Σ by $q\Sigma$. For simplicity, we show a simulation where $\mathcal{A}$ corrupts one of the receiver B or C (but not both of them) to produce a forgery for the initial signer A. The game between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ is defined as follows.

- SETUP: $\mathcal{C}$ runs SETUP to a given security parameter ℓ to obtain descriptions of $\mathcal{U}, \mathcal{M}, \mathcal{S}, \mathcal{K}, \mathcal{F}$ and ***KGEN***: $\mathcal{K} \rightarrow \mathcal{F}$. In addition, SETUP also generates the public key of each participant $\mathcal{P}_i \rightarrow s_i P$, for $i \in \{A, B, C\}$. The associated secret values s_i's

are delivered securely to participant $\mathcal{U}_i, i \in \{A, B, C\}$. The public keys $\{\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C\}$ are published together with the system parameters. Let $\Upsilon = xP$ and $\Psi = xyP$ be the CDH challenge and kept secret by $\mathcal{C}$ at this stage.

- KGen Queries: $\mathcal{A}$ can request that $\mathcal{C}$ selects a keystone $\hat{k} \in \mathcal{K}$ that it used to generate a keystone fix $f \in \mathcal{F}$, by invoking $f = KGEN(\hat{k})$. $\mathcal{A}$ can also select his own keystone $\hat{k} \in \mathcal{K}$ and the compute the keystone fix by himself by running the $\boldsymbol{KGEN}(k)$ function.
- KReveal Queries: $\mathcal{A}$ can request the challenger $\mathcal{C}$ to reveal the keystone $\hat{k}$ that is used to produce a keystone fix $f \in \mathcal{F}$ in a previous KGen Query. If f was not asked before, then C outputs invalid. Otherwise, $\mathcal{C}$ returns $\hat{k} \in \mathcal{K}$.
- Hash Queries: $\mathcal{A}$ can query the random oracle $\mathcal{H}$ at any time. When *(m, f)* are requested, firstly $\mathcal{C}$ checks his H-list. If it exists, then returns the value from the list. Otherwise, $\mathcal{C}$ selects a random $h \in Z_q^*$ at random and computes $M = h\Psi$. The value *(m, f, h, M)* is recorded in the H-list and M is returned to $\mathcal{A}$.
- ASIGN Queries: $\mathcal{A}$ can request an ambiguous signature for any input of the form $(m, f, \mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C)$ for published values $(\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C)$. $\mathcal{C}$ checks the H-list for the existence of m. If it does not exist, then $\mathcal{C}$ calls the ASIGN algorithm to sign the message as usual. However, if m exists, then $\mathcal{C}$ selects $a_2, r_1, r_2 \in Z_q^*$ at random and sets $a_1 = r_1 - a_2 r_2$. Finally $\mathcal{C}$ sets $Q_1 = a_1 P$ and $Q_2 = a_2 P$, and stores $(m, f, r_1, r_1 \Upsilon)$. Return (Q_1, Q_2) to $\mathcal{A}$ as a valid signature.
- AVerify and Verify Queries: $\mathcal{A}$ cannot request an answer for these queries since he can compute them for himself using AVERIFY and VERIFY algorithms.
- Output: Eventually, $\mathcal{A}$ outputs a forgery (m^*, Q_1^*, Q_2^*) and by definition of the existential forgery, there is in the H-list a quadruple (m^*, f^*, h^*, M^*) such that $\Phi = (h^*)^{-1}(Q_1^* + r_2 Q_2^*) = yP$.

We note that the success probability of the attack is defined by

$$\left(\frac{1}{2}\mathrm{Adv}_{\mathrm{EF-CMA}}^{\mathcal{A}} - \frac{q_{\mathcal{H}} q_{\Sigma} + 1}{2^{\ell}}\right)^2 \leq \mathrm{Succ}_{CDH}(\ell)$$

where ℓ is the security parameter. The time to execute the attack is defined by $t' \leq 2(t + q_{\mathcal{H}} + 2q_{\Sigma} + O(1)T_G + q_{\Sigma} T_{GM})$, where T_G denotes the time complexity to perform a scalar multiplication in G and T_{GM} denotes the time complexity to perform an exponentiation in G_M.

Solving two instances of this problem will lead us to a solution to the CDH problem using the technique in Boneh et al., (2003). Hence, we complete the proof. □

Theorem 5. *Our tripartite concurrent signature scheme is secure in the random oracle model, assuming the hardness of the discrete logarithm problem.*

Proof. The proof can be derived from Theorems 1, 2, 3 and 4 and Lemma 4. □

5. OPEN PROBLEMS

In this paper, we presented a tripartite concurrent signature scheme. Using a similar idea, we can obtain a multi party concurrent signature, but we have not defined the notion of fairness in that scenario. Hence, the open problem left in this area is how to define a formal model for a multi party concurrent signature scheme, where there are n parties involved (n>3).

REFERENCES

Abe et al., (2002) Abe, Masayuki, Ohkubo, Miyako, and Suzuki, Koutarou (2002). 1-out-of-n Signatures from a Variety of Keys. *Asiacrypt 2002, LNCS 2501*, pages 415-432.

Boneh et al., (2003) Boneh, Don, Gentry, Craig, Lynn, Ben, and Shacham, Hovav (2003). Aggregate and verifiably encrypted signatures from bilinear maps. *Eurocrypt 2003 LNCS 2656,* pages 416-432.

Camenisch, 1997 Camenisch, Jan (1997). Efficient and generalized group signatures. *Eurocrypt'97, LNCS 1233*, pages 465–479.

Camenisch, 1998 Camenisch, Jan (1998). Group signature schemes and payment systems based on the discrete logarithm problem. *PhD thesis, ETH Zürich.*

Chen et al., 2004 Chen, Liqun, Kudla, Caroline, and Paterson, Kenneth G. (2004). Concurrent signatures. In *Eurocrypt 2004, LNCS 3027*, pages 287–305.

Desmedt, 2003 Desmedt, Yvo (2003). Verifier-Designated Signatures. *Rump Session, Crypto 2003.*

Jakobsson et al., 1996Jakobsson, Markus, Sako, Kazue, and Impagliazzo, Russell (1996). Designated Verifier Proofs and Their Applications. *Eurocrypt'96, LNCS 1070*, pages 143 – 154.

Laguillaumie and Vergnaud, 2004 Laguillaumie, Fabien and Vergnaud, Damien (2004). Multi-Designated Verifiers Signatures. *Sixth Intl Conf on Inf and Comm Security (ICICS 2004).*

Rivest et al., 2001 Rivest, Ronald L., Shamir, Adi, and Tauman, Yael (2001). How to Leak a Secret. *Asiacrypt 2001, LNCS 2248*, pages 552 – 565.