Attribute-based concurrent signatures

BaoHong Li, Guoqing Xu and Yinliang Zhao

Department of Electronics and Information Engineering, Xi'an Jiaotong University
Xi'an, 710049 - China
[e-mail: bhli@mail.xjtu.edu.cn]

Abstract. In this paper, we introduce the notion of attribute-based concurrent signatures. This primitive can be considered as an interesting extension of concurrent signatures in the attribute-based setting. It allows two parties fairly exchange their signatures only if each of them has convinced the opposite party that he/she possesses certain attributes satisfying a given signing policy. Due to this new feature, this primitive can find useful applications in online contract signing, electronic transactions and so on. We formalize this notion and present a construction which is secure in the random oracle model under the Strong Diffie-Hellman assumption and the eXternal Diffie-Hellman assumption.

Keywords: Computer security, Network security, Cryptography, fair exchange, concurrent signatures.

1. Introduction

1.1 Motivation

Fair exchange of digital signatures is a fundamental problem in E-commerce, and concurrent signatures, introduced by Chen, Kudla and Paterson [1], provide a novel solution to this problem. This signature scheme allows two parties named as an initial signer and a matching signer to produce and exchange two *ambiguous* signatures until an extra piece of information (called keystone) is released by one of two parties. More specifically, before the keystone is released, two signatures are ambiguous in the sense that, from a third party's point of view, each signature may be generated by either of two parties. Once the keystone is released, however, both signatures are binding to their respective signers concurrently, and anybody can publicly verify who signed which signature.

In some applications, it is preferred that both sides of exchange possess certain attributes, such as title, age and nationality, in order for the exchange to occur correctly. We will illustrate this by following two exemplary scenarios.

Online contract signing. Suppose Company A wants to sign an agreement with Company B for software development. Company A requires the signer from Company B has to be either a senior manager within the development department or, at a minimum, a junior manager in the cryptography team. Meanwhile, Company B requires the signer from Company A must be the financial executive.

Electronic transactions. Broker Bob wants to sell a restricted class game to Customer Carl. According to relevant laws, Bob should be a licensed retailer for the U.S. market, while Carl must be over 18 years old and reside in U.S. [2].

Unfortunately, existing concurrent signature schemes do not take into account signers' attributes, hence can not be applied directly in above scenarios. A trivial solution would be that, prior to the exchange, two parties run an attribute-based authentication protocol [3] to confirm that they fulfill the attribute requirements. However, this solution is not only inefficient but also vulnerable to collusion attacks, that is, an unqualified signer may be able to produce and exchange a signature after the authentication protocol has been executed by his/her partner, who posseses enough attributes to fulfill the policy.

1.2 Contribution

In this paper, we introduce the notion of *attribute-based concurrent signatures* (ABCS) as a practical solution to aforementioned scenarios. It can be thought as an interesting extension of concurrent signatures in the attribute-based setting. In ABCS, every user obtains a set of attributes from the authority. To start an exchange, both parties define signing policies which the opposite side should satisfy. Then each of them can produce an *anonymous* signature if his/her attributes satisfy the signing policy. Anonymity is a stronger security notion than ambiguity, since it could be produced by anyone, not just limited to both parties of exchange. After two anonymous signatures have been exchanged, the release of a keystone can concurrently convert them to ordinary signatures such that they can be publicly verified.

We formalize this notion and present a formal security model for ABCS. We also present a construction which is secure in the random oracle model under the Strong Diffie-Hellman assumption [4] and the eXternal Diffie-Hellman assumption [5].

1.3 Technical Approach

We start from the *attribute-based group signature* scheme due to Khader [6], which provides the basis for our ABCS scheme in terms of the anonymity and utility of attribute validation. We also use a bottom-up approach presented by Emura and Miyaji [7] to efficiently construct dynamic access trees. The main challenge is how to maintain the anonymity when we extend it to concurrent signatures. Most concurrent signatures schemes [1, 8-12] are constructed from Schnorr-based ring signatures [14] where the ring consists of the involved two parties. Because of the anonymity and unforgeability of ring signatures, this kind of constructions provides a natural way to obtain ambiguity and fairness for concurrent signatures. Given such a ring signature, any third party can not tell which party of the ring is the signer, but the opposite party surely know who produces the signature. If we add such a ring in our ABCS scheme, however, the ambiguity may be compromised, since the signing policy attached with the signature may help a third party to deduce the signer's identity, especially when the signing policy is only fulfilled by one party.

In [15], Nguyen proposed a method to construct concurrent signatures which is independent of the ring signature concept. In this method, *promises* of signatures are used as concurrent signatures. A promise of signature is anonymous because it could be produced by anyone using solely public information, but the release of the keystone will convert it to an ordinary signature. His signature scheme is *asymmetric* in the sense that different signature schemes, namely the Schnorr signatures and the Schnorr-like signatures, have to be used for the initial signer and the matching signer, respectively, so that the matching signer is also able to produce a promise of signature even without the knowledge of the keystone. Although this method provides proper anonymity which meets the requirement of our ABCS scheme, we can not directly apply it to our construction since its Schnorr-like signatures can not be used to generate a SPK of the discrete logarithms for a public key which has the form of $y = g_1^{x_1} g_2^{x_2}$. We address this limitation by presenting a new variant of Schnorr signatures (see Sect. 2.2). This signature scheme is quite suitable for our construction since it has the same form of responses as Schnorr signatures.

1.4 Related Work

Several concepts are related to ours ABCS. We briefly review them as follows.

Concurrent signatures. Chen, Kudla and Paterson [1] introduced the notion of concurrent signatures in Eurocrypt 2004. This primitive has been considered to be more practical than some other fair exchange techniques, such as gradual secret releasing [16] or optimistic fair exchange (OFE) [17], since it does not rely on a highly interactive protocol or a (semi-trusted) third party. In ICICS 2004, Susilo, Mu and Zhang [8] introduced the notion of *perfect concurrent signatures* in order to strengthen the ambiguity of concurrent signatures. Unfortunately, their scheme was shown to have a flaw in fairness by Wang, Bao and Zhou [9], and thus a modified construction was presented. Subsequently, various efforts have been made to add new features to concurrent signatures or strengthen their security. For example, Sherman, Chow and Susilo [10] presented the first generic construction of identity-based perfect concurrent signatures. Tonien, Susilo and Safavi-Naini [11] extended concurrent signatures to the multi-party setting. Yuen et al. [12] added a feature of negotiable binding control into concurrent signatures, and Tan, Huang and Wong [13] presented the first concurrent signature scheme secure in the standard model.

Most concurrent signature schemes are based on Schnorr-based ring signatures [14], except

the asymmetric concurrent signatures of Nguyen [15], where promises of Schnorr signatures and Schnorr-like signatures are used as concurrent signatures.

Attribute-based signatures. Maji, Prabhakaran and Rosulek [18] presented the first attribute-based signatures (ABS) in 2008. This primitive allows a signer to convince a verifier that he/she holds a set of attributes satisfying a given signing policy and has endorsed the message. Since their construction is only secure in generic group model, several ABS schemes [19-21] were presented that are secure in the standard model. However, they are only *selectively secure*, a weaker notion of unforgeability than adaptive security. Maji, Prabhakaran and Rosulek [22] presented the first ABS which is adaptively secure in the standard model, but their scheme is much less efficient in signature size since it employed the Groth-Sahai NIZK system as building blocks. Okamoto and Takashima [23] presented the first ABS which allows non-monotone predicates to express signing policies. They further extended their ABS to the multi-authority setting [24].

Attribute-based group signatures. The notion of attribute-based group signatures (ABGS) was first introduced by Khader [6]. This primitive can be considered as an extension of group signatures such that a group member can produce a signature if he/she holds sufficient attributes satisfying a given signing policy. Subsequently, Emura, Miyaji and Omote [7] presented a dynamic ABGS which is efficient when access trees have to change frequently. Ali and Amberker [25] presented the first ABGS which is secure in the standard model. They also addressed the issue of attribute anonymity, which may be desirable in some applications.

Attribute-based optimistic fair exchange. Wang, Au and Susilo [2] have identified the necessity of attributes in the area of fair exchange of signatures, and introduced the notion of attribute-based optimistic fair exchange (ABOFE) to address this issue. As an extension of optimistic fair exchange [17], ABOFE allows each party of exchange to obtain the full signature from the opposite side only if he/she holds sufficient attributes satisfying a given signing policy. They also presented a generic construction of ABOFE from OFE and ciphertext-policy attribute-based encryption (CP-ABE). The intuition behind their construction is quite simple: prior to the exchange, each party encrypts his/her signature by CP-ABE so that the opposite party can decrypt the signature only if he/she posseses sufficient attributes. Since ABOFE is built on OFE, one of its problems is the requirement for a dispute resolving third party, which may be undesirable in some applications. More seriously, their construction is vulnerable to collusion attacks, that is, an unqualified user could have the signature decrypted by his/her qualified partner. Hence, our ABCS scheme can be considered as an improvement of ABOFE in terms of applicability and security.

The remainder of the paper is organized as follows. In Section 2, we review some definitions and complexity assumptions. We also present a variant of Schnorr signature scheme which serve as basic building block in our ABCS scheme. Section 3 defines ABCS and formulizes its security. In Section 4, we present our construction of ABCS and prove its security. Finally, Section 5 concludes the whole paper.

2. Preliminaries

2.1 Bilinear Groups and Complexity Assumptions

Definition 1 (**Bilinear group**): Consider two (multiplicative) cyclic groups G_1 and G_2 of prime order p where g_1 and g_2 are respective generators of G_1, G_2. We say G_1 and G_2 are

bilinear groups if there exist a group G_T and an efficiently computable function $\hat{e}: G_1 \times G_2 \rightarrow G_T$ with the following properties:

(1) For $\forall u \in G_1, \forall v \in G_2$ and $\forall a, b \in \mathbb{Z}_p$, $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$.

(2) $\hat{e}(g_1, g_2)$ is a generator of G_T.

Additionally, we require a computable isomorphism ψ from G_2 to G_1, with $g_1 = \psi(g_2)$.

The security of our construction relies on the Strong Diffie-Hellman assumption [4] and the eXternal Diffie-Hellman assumption [5]. They have been used to construct anonymous authentication [26], group signature [27], anonymous credentials [28], zero-knowledge proof system [29], to name a few. We briefly review them as follows.

Assumption 1 (*q*-SDH). The *q*-Strong Diffie-Hellman problem in (G_1, G_2) is defined as follows: On input of a (q + 2)-tuple $\langle g_1, g_2, g_2^{\gamma}, g_2^{(\gamma^2)}, \ldots, g_2^{(\gamma^q)} \rangle \in G_1 \times G_2^{q+1}$, output a pair $\langle g_1^{1/(\gamma+x)}, x \rangle \in G_1 \times \mathbb{Z}_p$. We say that the *q*-SDH assumption in (G_1, G_2) holds if no PPT algorithm has non-negligible probability in solving the *q*-SDH problem.

Given a *q*-SDH instance $\langle g_1, g_2, g_2^{\gamma}, g_2^{(\gamma^2)}, \ldots, g_2^{(\gamma^q)} \rangle$, by applying the Boneh-Boyen's Lemma in [4], we can obtain $\langle g_1, g_2, w = g_2^{\gamma} \rangle$ and $(q-1)$ SDH pairs (A_i, x_i) such that $\hat{e}(A_i, w g_2^{x_i}) = \hat{e}(g_1, g_2)$. Any SDH pair besides these $(q-1)$ ones can be transformed into a solution to the original *q*-SDH instance.

Assumption 2 (XDH). Given bilinear groups (G_1, G_2) with a computable isomorphism ψ from G_2 to G_1 and $g_1 = \psi(g_2)$. We say that the eXternal Diffie-Hellman assumption holds if the DDH problem is hard in G_1.

2.2 Access Tree and the bottom-up approach

Definition 2 (**Access Tree** [30]): An access tree is a tree structure representing an (monotone) access structure, where threshold gates are defined on each interior node, and each leaf is associated with an attribute.

An access tree is originally built from top to down. This approach has a drawback that it is impossible to build a new access tree from an existing one. In [7], Emura and Miyaji proposed a bottom-up approach which allows dynamic access tree construction. In this approach, a central access tree is built first and secret values are assigned to attributes associated with leaves. Then different access trees can be obtained by simplifying the central access tree, and these secret values need not to be updated. As a result, it is not necessary to re-issue attribute public keys whenever access structures change.

Our ABCS scheme uses this bottom-up approach to build an access tree, and we denote these two steps by following two algorithms

–**BuildCTree**(*Att*). Takes the universe of attributes *Att* as input, outputs a central access tree *CT* and some secret values.

–**SimplifyCTree**(*CT*, ϕ). Take as input a central tree *CT* and an attribute set $\phi \in Att$, outputs an access tree *T* where attributes in ϕ are leaves.

We refer to Appendix A for a self-contained presentation of these two algorithms.

2.3 A variant of Schnorr signature scheme and promises of signatures

Our ABCS scheme uses Schnorr signatures to produce promises of initial signatures. We also need a signature scheme to produce promises of matching signatures such that they can be produced without the knowledge of keystones. We present as follows a variant of Schnorr signature scheme for this purpose.

–Setup. Choose primes p, q of appropriate size such that $q \mid p-1$, and let g be the generator for the subgroup in $\mathbb{Z}_p^*$ of order q. Choose a hash function $H: \{0,1\}^* \to \mathbb{Z}_q$.

–Key Generation. Pick random $x \in \mathbb{Z}_q$ and set $z = g^{1/x}$. The public key is $\langle p, q, g, z \rangle$ and the secret key is x.

–Sign. Pick random $r \in \mathbb{Z}_q$ and compute $c = H(M, z^r)$, $s = r + cx$. The signature is $\langle c, s \rangle$.

–Verify. Check $c = H(M, z^s g^{-c})$.

Different from the Schnorr-like signature scheme presented in [15], Our Schnorr-like signature scheme has the same form of response (i.e. s) as the original Schnorr signature scheme. This feature is critical to our ABCS construction since it allows generating a SPK of discrete logarithms for a public key with the form of $y = g_1^{x_1} g_2^{x_2}$. Using essentially the same method for the original Schnorr signature scheme [31], we can prove the following lemma.

Lemma 1. This Schnorr-like signature scheme is existentially unforgeable against adaptive chosen message attacks in random oracle model, under the Discrete Logarithm assumption.

Definition 3 (**promises of signatures** [15])). Let *func* be some cryptographic function. The value $\sigma = \langle s, \rho \rangle$ is said to be a valid promise of signature $\omega = \langle k, \rho \rangle$ on some message M if the following conditions hold:

–Publicly Verifiable: given σ, everyone is convinced that if there exists $k = func^{-1}(s)$ then $\omega = \langle k, \rho \rangle$ is a valid ordinary signature.

–Anonymity: without the knowledge of $k = func^{-1}(s)$, σ is indistinguishable from random elements of the signature space.

Given a value $f = g^k$, we can obtain a promise of signature $\sigma = \langle f', c, s_1 \rangle$ by picking random $r \in \mathbb{Z}_q$ and computing $f' = f^{1/x} = z^k$, $c = H(M, z^r f')$, $s_1 = r + cx$. This promise of signature can be verified by checking $c = H(M, z^{s_1} g^{-c} f')$, and the release of k can convert it to an ordinary Schnorr-like signature $\omega = \langle c, s_1 + k \rangle$.

Lemma 2. Let $\omega = \langle c, s \rangle$ be a Schnorr-like signature. The value $\sigma = \langle f', c, s_1 \rangle$ is a valid promise of the signature ω, where $c = H(M, z^{s_1} g^{-c} f')$ and $f' = z^k$ for some k.

Proof. Given the value $k = \log_z f'$, from $c = H(M, z^{s_1} g^{-c} f')$ we have $c = H(M, z^{s_1 + k} g^{-c})$. It implies that $\omega = \langle c, s_1 + k \rangle$ is a valid Schnorr-like signature. In addition, the verification of $c = H(M, z^{s_1} g^{-c} f')$ requires only public information. Hence the publicly verifiable condition is satisfied.

To prove that this promise of signature is anonymous, we first show that it could be simulated by using any public key: given any public key z, the simulator picks random r, s_1, computes $c = H(M, z^r z^{s_1})$ and sets $f' = g^c z^r$. We have a promise of signature $\sigma = \langle f', c, s_1 \rangle$ where $c = H(M, z^{s_1} g^{-c} f')$ holds.

Next we prove that, in the random oracle model and under the DDH assumption, this simulated promise of signature is indistinguishable from an honestly-generated promise of signature.

Given a DDH instance $\langle g, g^{a_1}, g^{a_2}, g^{a_3} \rangle$, we set $f = g^{a_1}$, $z = g^{a_2}$, $f' = g^{a_3}$. Then we pick random c, s_1, set $c = H(M, z^{s_1} g^{-c} f')$ and return the simulated promise of signature $\sigma = \langle f', c, s_1 \rangle$. If $a_1 a_2 = a_3$, we have $k = a_1$, $x = 1/a_2$, $f' = z^{a_1}$ and $r = s_1 - cx$, thus σ is an honestly-generated promise of signature. Otherwise, σ is a simulated promise of signature. If there exists an algorithm can distinguish an honestly-generated promise of signature from a simulated promise of signature with non-negligible advantage, clearly it could be used to solve the DDH problem.

3. Formal Definitions of ABCS

In this section, we presents the formal definition of ABCS and a concrete protocol to carry out this signature scheme. We also present the security model for ABCS by adapting the concurrent signatures model to the attribute-based setting.

3.1 Attribute-based concurrent signatures

Definition 4 (**ABCS**): An attribute-based concurrent signature scheme among an authority and a set of users $U = \{1, \ldots, n\}$ is a signature scheme consisted of following algorithms.

–Setup(1^λ). On input security parameter 1^λ, this algorithm defines the keystone space $\mathcal{K}$, the keystone fix space $\mathcal{F}$ and a universe of attribute *Att*. It also runs the algorithm BuildCTree(*Att*) to build a central access tree. Finally, it outputs a master secret key *MSK* for the authority and some public parameters *PK*.

–KeyGen(i, att_i, *MSK*, *PK*). This algorithm takes as input a user index $i \in U$, an attribute set $att_i \subseteq Att$ possessed by the user, *MSK* and *PK*, outputs a public/secret key pair $\langle upk_i, usk_i \rangle$.

–KfGen(k). This algorithm takes as input a keystone $k \in \mathcal{K}$, outputs a keystone fix $f \in \mathcal{F}$.

–KfTran(f, usk_i). This algorithm takes as input a keystone fix $f \in \mathcal{F}$ and a secret key usk_i, outputs a new keystone fix $f_i \in \mathcal{F}$.

–KfVer(k, f_i, upk_i).Takes as input a keystone $k \in \mathcal{K}$, a public key upk_i and a keystone fix $f_i \in \mathcal{F}$ output by KfTran, this algorithm outputs *accept* if f_i is valid or *reject* otherwise.

–Isign($upk_i, usk_i, T_i, PK, M_i$). This algorithm takes as input a public/secret key pair, an access tree T_i, *PK* and a message M_i, outputs a promise of initial signature $\sigma_i = \langle f, \rho_i \rangle$ and a keystone $k \in \mathcal{K}$ where f = KfGen(k).

–IVerify($\sigma_i, upk_i, T_i, PK, M_i$). This algorithm takes as input a promise of initial signature σ_i on a message M_i, a public key upk_i, an access tree T_i and *PK*, outputs accept if σ_i is valid or reject otherwise.

–Msign($f, upk_j, usk_j, T_j, PK, M_j$). This algorithm takes as input a keystone fix f, a public/secret key pair, an access tree T_j, *PK* and a message M_j, outputs a promise of matching signature $\sigma_j = \langle f_j, s_j, \rho_j \rangle$ where f_j = KfTran(f, usk_j).

–MVerify($\sigma_j, upk_j, T_j, PK, M_j$). This algorithm takes as input a promise of matching signature σ_j on a message M_j, a public key upk_j, an access tree T_j and *PK*, outputs accept if σ_j is valid or reject otherwise.

–Verify(k, σ_i, upk_i, T_i, PK, M_i). This algorithm takes as input a keystone k, a promise of signature $\sigma_i = \langle f, \rho_i \rangle$ or $\langle f_i, s_i, \rho_i \rangle$ on a message M_i, a public key upk_i, an access tree T_i and PK. Depending on the type of σ_i, it outputs accept if either of the following two cases hold:

(1) If $\sigma_i = \langle f, \rho_i \rangle$, $f = \mathsf{KfGen}(k) \wedge \mathsf{IVerify}(\sigma_i, upk_i, T_i, PK, M_i) = \text{accept}$, or

(2) If $\sigma_i = \langle f_i, s_i, \rho_i \rangle$, $\mathsf{KfVer}(k, f_i, upk_i) = \text{accept} \wedge \mathsf{MVerify}(\sigma_i, upk_i, T_i, PK, M_i) = \text{accept}$.

3.2 A Signature Exchange Protocol

Given an attribute-based concurrent signature scheme, we define a signature exchange protocol between an initial signer Alice and a matching signer Bob. Before the exchange phase, The authority runs $\mathsf{Setup}(1^\lambda)$ to establish the system and a central tree CT. It also runs KeyGen to generate public/secret key pairs for Alice and Bob, respectively. Then the protocol works as follows:

(1) Alice and Bob select attribute sets ϕ_B and ϕ_A that the opposite side should satisfy. Then they run the algorithm $\mathsf{SimplifyCTree}(CT, \phi_B)$, $\mathsf{SimplifyCTree}(CT, \phi_A)$, respectively, and output access trees T_B and T_A.

(2) Alice produces a promise of initial signature $\sigma_A = \langle f, \rho_A \rangle = \mathsf{Isign}(upk_A, usk_A, T_A, PK, M_A)$ on a message M_A and a keystone k such that $f = \mathsf{KfGen}(k)$. Alice sends σ_A to Bob.

(3) Bob validates σ_A by checking $\mathsf{IVerify}(\sigma_A, upk_A, T_A, PK, M_A) = \text{accept}$. If not, Bob aborts the protocol. Otherwise, Bob produces a promise of matching signature $\sigma_B = \mathsf{Msign}(f, upk_B, usk_B, T_B, PK, M_B)$ on a message M_B and sends it to Alice.

(4) Upon receiving $\sigma_B = \langle f_B, s_B, \rho_B \rangle$ from Bob, Alice checks $\mathsf{KfVer}(k, f_B, upk_B) = \text{accept}$ and $\mathsf{MVerify}(\sigma_B, upk_B, T_B, PK, M_B) = \text{accept}$. If so, Alice forwards the keystone k to Bob.

(5) Using the keystone k, Alice and Bob convert σ_B and σ_A to ordinary signatures as $\omega_A = \langle k, \rho_A \rangle$, $\omega_B = \langle s_B + k, \rho_B \rangle$.

3.3 Security Model

Unforgeability, anonymity and fairness are three security properties that our ABCS scheme should satisfy. The unforgeability states that an adversary, without the knowledge of a signer's secret key, can not provide a keystone and a promise of signature, such that it can be converted to an ordinary signature for this signer. Formally, it is defined in the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

–Setup. $\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda)$ to generate the public parameters PK and the master secret key MSK. $\mathcal{C}$ sends PK to $\mathcal{A}$.

–Query. $\mathcal{A}$ can adaptively make following types of queries to $\mathcal{C}$:

(1) **PubkeyGen**. $\mathcal{A}$ requests the public key upk_i for a user index $i \in U$.

(2) **SecKeyGen**. $\mathcal{A}$ requests the secret key pair usk_i for a public key upk_i and an attribute set att_i.

(3) **BuildTree**. $\mathcal{A}$ requests an access tree T_i on an attribute set ϕ_i.

(4) **Isign**. $\mathcal{A}$ requests a promise of initial signature by inputting a user index i, an access tree

T_i and a message M_i.

(5) **Msign**. $\mathcal{A}$ requests a promise of matching signature by inputting a user index i, an access tree T_i, a message M_j and a keystone fix f.

(5) **KsReveal**. $\mathcal{A}$ requests the keystone $k \in \mathcal{K}$ for a keystone fix $f \in \mathcal{F}$ which is an output of previous Isign queries.

–Output. $\mathcal{A}$ outputs a keystone k and a promise of signature $\sigma_i = \langle f, \rho_i \rangle$ or $\langle f_i, s_i, \rho_i \rangle$ on a message M_i, such that Verify(k, σ_i, upk_i, T_i, PK, M_i) = accept. $\mathcal{A}$ succeeds if all of following conditions hold:

(1) No SecKeyGen query has made on the public key upk_i.

(2) No Isign query on $\langle i, T_i, M_i \rangle$ or Msign query on $\langle f, i, T_i, M_i \rangle$ has made.

(3) No KsReveal query has made on f.

Definition 5 (**Unforgeability**). An attribute-based concurrent signature scheme is existentially unforgeable against adaptive chosen message attacks if the success probability of any PPT adversary in the above game is negligible.

The anonymity states that, without the knowledge of the keystone, an adversary can not determine the identity of the signer from a promise of signature. Formally, it is defined in the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

–Setup. It is the same as the unforgeability game.

–Query Phase 1. It is the same as the Query phase in the unforgeability game.

–Challenge. $\mathcal{A}$ provides two indices i_0, i_1, an access tree T_i and a message M_i such that both i_0 and i_1 have secret keys satisfying T_i. Since there are two types of promises of signatures, we consider following two cases.

(1) $\mathcal{C}$ returns a promise of initial signature σ_{i_b} = Isign(upk_{i_b}, usk_{i_b}, T_i, PK, M_i) for a random $b \in \{0, 1\}$.

(2) $\mathcal{A}$ also provides a keystone fix f, with the restriction that f should be an output of previous Isign queries and no KsReveal query has made on it. $\mathcal{C}$ returns a promise of matching signature σ_{i_b} = Msign(f, upk_{i_b}, usk_{i_b}, T_i, PK, M_i) for a random $b \in \{0, 1\}$.

–Query Phase 2. It is the same as Query Phase 1 except that no KsReveal query has made on f.

–Output. Finally, $\mathcal{A}$ outputs a guess b' of b. $\mathcal{A}$ wins the game if $b' = b$.

Definition 6 (**Anonymity**). An attribute-based concurrent signature scheme is anonymous if the advantage of any PPT adversary $\mathcal{A}$ in the above game is negligible, where the advantage of $\mathcal{A}$ is defined as $\Pr[b' = b] - 1/2$.

The fairness states that, given an initial signer A and a matching signer B, B can not provide a keystone k and a promise of initial signature such that it can be converted to an ordinary signature for A, or A can not release a keystone k such that the promise of matching signature is converted to an ordinary signature for B while the promise of initial signature still keeps anonymous. Formally, it is defined in the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

–Setup. It is the same as the unforgeability game.

–Query. It is the same as the Query phase in the unforgeability game.

–Output. $\mathcal{A}$ outputs a keystone k and a promise of signature σ_i such that they are accepted by Verify and no SecKeyGen query has made on upk_i. $\mathcal{A}$ wins the game if either of following two conditions holds:

(1) $\sigma_i = \langle f, \rho_i \rangle$, and no KsReveal Query has made on f or,

(2) $\sigma_i = \langle f_i, s_i, \rho_i \rangle$, and $\mathcal{C}$ also outputs a promise of signature $\sigma_j = \langle f, \rho_j \rangle$ such that it is accepted by IVerify and f_i = KfTran(f, usk_j) but σ_j and k are rejected by Verify.

Definition 3. An attribute-based concurrent signature scheme is fair if the success probability of any PPT adversary in the above game is negligible.

4. Construction and Security Proofs

4.1 Construction

We present our construction of ABCS as follows.

– Setup(1^λ). The algorithm performs the following steps:

(1) Generate a bilinear group (G_1, G_2) of prime order p, with a computable isomorphism ψ: $G_2 \to G_1$. Select a random generator g_2 in G_2, and set $g_1 = \psi(g_2)$. Select random $u, v \in G_1$ and a hash function $H: \{0,1\}^* \to \mathbb{Z}_p^*$. Define the keystone space and the keystone fix space as $\mathcal{K} = \mathbb{Z}_p^*$, $\mathcal{F} = G_1$.

(2) Define a universe of attributes Att = {1,…, $|Att|$}. Run BuildCTree(Att) to obtain the central access tree and secret values $\{t_l\}_{l \in Att}$, $\{d_l\}_{l \in Dum}$, where Dum is the set of dummy nodes.

(3) Compute $g_2^{t_l}$ and $v_l = v^{t_l}$ for each attribute $l \in Att$. Compute $g_2^{d_l}$ for each dummy node $l \in Dum$.

(4) Select random $\gamma \in \mathbb{Z}_p^*$ and output $MSK = \langle \gamma, t_1, \ldots, t_{|Att|} \rangle$.

(5) Output $PK = \langle G_1, G_2, g_1, g_2, w = g_2^\gamma, u, v, g_2^{t_1}, \ldots, g_2^{t_{|Att|}}, v_1, \ldots, v_{|Att|}, g_2^{d_1}, \ldots, g_2^{d_{|Dum|}} \rangle$.

–KeyGen(i, att_i, MSK, PK). The algorithm picks random $x_i \in \mathbb{Z}_p^*$, computes $A_i = g_1^{1/(\gamma + x_i)}$, $y_i = g_1^{x_i}$, $z_i = g_1^{1/x_i}$ and $T_{i,l} = A_i^{t_l}$ for each attribute $l \in att_i$. It outputs $upk_i = \langle y_i, z_i \rangle$, $usk_i = \langle A_i, x_i, T_{i,1}, \ldots, T_{i,|att_i|} \rangle$.

–KfGen(k). The algorithm outputs $f = g_1^k$.

–KfTran(f, usk_i). The algorithm outputs $f_i = f^{1/x_i}$.

–KfVer(k, f_i, upk_i). The algorithm outputs accept if $f_i = z_i^k$.

–Isign($upk_i, usk_i, T_i, PK, M_i$). The algorithm performs the following steps:

(1) Let ϕ_i be the attribute set in T_i. Compute $\triangle_l$ for each $l \in \phi_i$ and set $B = \prod_{l \in \phi_i} (g_2^{t_l})^{\triangle_l}$.

(2) Pick random $\alpha, \beta \in \mathbb{Z}_p^*$ and encrypt A_i and $T_{i,l}$ as:

$$C_1 = u^\alpha,\ C_2 = A_i v^\alpha,\ C_3 = g_2^\beta,\ C_4 = B^\beta,\ CT_l = (T_{i,l} v_l^\alpha)^\beta,\ l \in \phi_i.$$

(3) Set $\delta = x_i\alpha$ and compute $SPK\{(\alpha,\beta,x_i,\delta): y_i = g_1^{x_i} \wedge \frac{\hat{e}(C_2,g_2)^{x_i}}{\hat{e}(v,g_2)^{\delta}\hat{e}(v,w)^{\alpha}} = \frac{\hat{e}(g_1,g_2)}{\hat{e}(C_2,w)} \wedge C_3 = g_2^{\beta}$ $\wedge C_4 = B^{\beta} \wedge \frac{\hat{e}(\prod_{l\in\phi_i} CT_l^{\Delta_l}, g_2)}{\hat{e}(C_2,C_4)} = \frac{\hat{e}(\prod_{l\in\phi_i} v_l^{\Delta_l}, C_3)^{\alpha}}{\hat{e}(v,C_4)^{\alpha}}\}(M_i)$. Concretely, it picks random $r_i, r_\alpha, r_\beta, r_\delta \in \mathbb{Z}_p^*$, and computes:

$R_1 = g_1^{r_i}$, $R_2 = u^{r_\alpha}$, $R_3 = C_1^{r_i} u^{-r_\delta}$, $R_4 = \hat{e}(C_2,g_2)^{r_i}\hat{e}(v,w)^{-r_\alpha}\hat{e}(v,g_2)^{-r_\delta}$

$R_5 = g_2^{r_\beta}$, $R_6 = B^{r_\beta}$, $R_7 = (\hat{e}(\prod_{l\in\phi_i} v_l^{\Delta_l}, C_3)/\hat{e}(v,C_4))^{r_\alpha}$.

$c = H(M_i, upk_i, C_1, C_2, C_3, C_4, CT_1, \ldots, CT_{|\phi_i|}, R_1, \ldots, R_7)$.

$s_i = (r_i + cx_i)$, $s_\alpha = (r_\alpha + c\alpha)$, $s_\beta = (r_\beta + c\beta)$, $s_\delta = (r_\delta + c\delta)$.

$f = \mathsf{KfGen}(s_i) = g_1^{s_i}$.

(4) Output $k = s_i$, $\sigma_i = \langle f, T_i, upk_i, C_1, C_2, C_3, C_4, CT_1, \ldots, CT_{|\phi_i|}, c, s_\alpha, s_\beta, s_\delta, C_1^{s_i}, C_2^{s_i}\rangle$.

–IVerify($\sigma_i, upk_i, T_i, PK, M_i$). This algorithm performs the following steps:

(1) Compute $B = \prod_{l\in\phi_i}(g_2^{t_l})^{\Delta_l}$ from T_i.

(2) Compute:

$\tilde{R}_1 = fy_i^{-c}$, $\tilde{R}_2 = u^{s_\alpha}C_1^{-c}$, $\tilde{R}_3 = C_1^{s_i}u^{-s_\delta}$ (1–3)

$\tilde{R}_4 = \hat{e}(C_2^{s_i}, g_2)\hat{e}(v,w)^{-s_\alpha}\hat{e}(v,g_2)^{-s_\delta}(\hat{e}(C_2,w)/\hat{e}(g_1,g_2))^{c}$ (4)

$\tilde{R}_5 = g_2^{s_\beta}C_3^{-c}$, $\tilde{R}_6 = B^{s_\beta}C_4^{-c}$ (5–6)

$\tilde{R}_7 = (\hat{e}(\prod_{l\in\phi_i} v_l^{\Delta_l}, C_3)/\hat{e}(v,C_4))^{s_\alpha}(\hat{e}(C_2,C_4)/\hat{e}(\prod_{l\in\phi_i} CT_l^{\Delta_l}, g_2))^{c}$ (7)

(3) Output accept if $c = H(M_i, upk_i, C_1, C_2, C_3, C_4, CT_1, \ldots, CT_{|\phi_i|}, \tilde{R}_1, \ldots, \tilde{R}_7)$.

–Msign($f, upk_j, usk_j, T_j, PK, M_j$). This algorithm performs the following steps:

(1) Compute $B = \prod_{l\in\phi_j}(g_2^{t_l})^{\Delta_l}$, where ϕ_j is the attribute set in T_j.

(2) Pick random $\alpha, \beta \in \mathbb{Z}_p^*$ and encrypt A_j and $T_{j,l}$ as:

$C_1 = u^{\alpha}$, $C_2 = A_j v^{\alpha}$, $C_3 = g_2^{\beta}$, $C_4 = B^{\beta}$, $CT_l = (T_{j,l}v_l^{\alpha})^{\beta}$, $l \in \phi_j$.

(3) Compute $f_j = \mathsf{KfTran}(f, usk_j) = f^{1/x_j}$.

(4) Set $\delta = x_j\alpha$ and compute $SPK\{(\alpha,\beta,x_j,\delta): z_j = g_1^{1/x_j} \wedge \frac{\hat{e}(C_2,g_2)^{x_j}}{\hat{e}(v,g_2)^{\delta}\hat{e}(v,w)^{\alpha}} = \frac{\hat{e}(g_1,g_2)}{\hat{e}(C_2,w)} \wedge C_3 = g_2^{\beta}$ $\wedge C_4 = B^{\beta} \wedge \frac{\hat{e}(\prod_{l\in\phi_j} CT_l^{\Delta_l}, g_2)}{\hat{e}(C_2,C_4)} = \frac{\hat{e}(\prod_{l\in\phi_j} v_l^{\Delta_l}, C_3)^{\alpha}}{\hat{e}(v,C_4)^{\alpha}}\}(M_j)$. To do so, it picks random $r_j, r_\alpha, r_\beta, r_\delta \in \mathbb{Z}_p^*$, and computes:

$R_1 = z_j^{r_j} f_j$, $R_2 = u^{r_\alpha}$, $R_3 = C_1^{r_j} u^{-r_\delta}$, $R_4 = \hat{e}(C_2,g_2)^{r_j}\hat{e}(v,w)^{-r_\alpha}\hat{e}(v,g_2)^{-r_\delta}$

$R_5 = g_2^{r_\beta}$, $R_6 = B^{r_\beta}$, $R_7 = (\hat{e}(\prod_{l\in\phi_j} v_l^{\Delta_l}, C_3)/\hat{e}(v,C_4))^{r_\alpha}$.

$c = H(M_j, upk_j, C_1, C_2, C_3, C_4, CT_1, \ldots, CT_{|\phi_j|}, R_1, \ldots, R_7)$.

$s_j = (r_j + cx_j)$, $s_\alpha = (r_\alpha + c\alpha)$, $s_\beta = (r_\beta + c\beta)$, $s_\delta = (r_\delta + c\delta)$.

(5) Output $\sigma_j = \langle f_j, s_j, T_j, upk_j, C_1, C_2, C_3, C_4, CT_1, \ldots, CT_{|\phi_j|}, c, s_\alpha, s_\beta, s_\delta\rangle$.

–MVerify($\sigma_j, upk_j, T_j, PK, M_j$). This algorithm performs the following steps:

(1) Compute $B = \prod_{l\in\phi_j}(g_2^{t_l})^{\Delta_l}$ from T_j.

(2) Compute:

$\tilde{R}_1 = z_j^{s_j} g_1^{-c} f_j$, $\tilde{R}_2 = u^{s_\alpha}C_1^{-c}$, $\tilde{R}_3 = C_1^{s_j}u^{-s_\delta}$ (8–10)

$\tilde{R}_4 = \hat{e}(C_2,g_2)^{s_j}\hat{e}(v,w)^{-s_\alpha}\hat{e}(v,g_2)^{-s_\delta}(\hat{e}(C_2,w)/\hat{e}(g_1,g_2))^{c}$ (11)

$$\tilde{R}_5 = g_2^{s_\beta} C_3^{-c}, \ \tilde{R}_6 = B^{s_\beta} C_4^{-c} \quad (12\text{–}13)$$

$$\tilde{R}_7 = (\hat{e}(\prod_{l\in\phi_j} v_l^{\Delta_l}, C_3) / \hat{e}(v, C_4))^{s_\alpha} (\hat{e}(C_2, C_4) / \hat{e}(\prod_{l\in\phi_j} CT_l^{\Delta_l}, g_2))^c \quad (14)$$

(3) Output accept if $c = H(M_j, upk_j, C_1, C_2, C_3, C_4, CT_1, \ldots, CT_{|\phi_j|}, \tilde{R}_1, \ldots, \tilde{R}_7)$.

–Verify(k, σ_i, upk_i, T_i, PK, M). Given a keystone k and a promise of initial signature σ_i (or a promise of matching signature σ_i), the algorithm outputs accept if IVerify(σ_i, upk_i, T_i, PK, M_i) = accept and f = KfGen(k) (or MVerify(σ_i, upk_i, T_i, PK, M_i) = accept and KfVer(k, f_j, upk_j) = accept).

4.2 Unforgeability

Theorem 1 (Unforgeability). Suppose there exists an adversary $\mathcal{A}$ that can break the unforgeability of our ABCS scheme with a non-negligible probability ε. Then, in the random oracle model, we can build an algorithm $\mathcal{B}$ that has advantage at least $(\varepsilon / n - 1/q)^2 / (16 q_H)$ in breaking the q-SDH assumption, where q_H is the number of hash function queries made by $\mathcal{A}$.

Proof. We first consider the case where σ_i is a promise of initial signature. We show how to build the algorithm $\mathcal{B}$ as follows.

–Setup. Given a q-SDH instance, $\mathcal{B}$ can obtain a tuple of (g_1, g_2, $w = g_2^\gamma$) and a list of $q-1$ SDH pairs $\langle A_i, x_i \rangle$ where $A_i = g_1^{1/(\gamma + x_i)}$. We will instantiate q as $n+1$ or n for different types of forgeries. In the case where $q = n$, $\mathcal{B}$ obtains an additional pair by picking random $\langle A_{i^*}, x_{i^*} \rangle \in G_1 \times \mathbb{Z}_p^*$ for a random $i^* \in \{1, ..., n\}$. Then $\mathcal{B}$ setups the system in the same way as the algorithm Setup(1^λ), except that it implicitly sets γ as the part of MSK. Finally, $\mathcal{B}$ sends PK to $\mathcal{A}$.

–PubkeyGen Queries. Given a user index i, $\mathcal{B}$ chooses the pair $\langle A_i, x_i \rangle$ from the list and returns $upk_i = \langle y_i = g_1^{x_i}, z_i = g_1^{x_i^{-1}} \rangle$.

–SecKeyGen Queries. Given a public key upk_i and an attribute set att_i, $\mathcal{B}$ returns $usk_i = \langle A_i, x_i, T_{i,1}, \ldots, T_{i,|att_i|} \rangle$ if $i \neq i^*$. Otherwise it aborts because $\langle A_{i^*}, x_{i^*} \rangle$ is not a valid SDH pair.

–BuildTree. Given an attribute set ϕ_i, $\mathcal{B}$ returns T_i = SimplifyCTree(CT, ϕ_i).

–Hash Queries. Given a tuple of $\langle M_i, upk_i, C_1\text{–}C_4, CT_1 - CT_{|\phi|}, R_1\text{–}R_7 \rangle$, $\mathcal{B}$ returns a random $c \in \mathbb{Z}_p^*$ and saves it in hash table incase the same query is requested again.

–Isign Queries. If $i \neq i^*$, $\mathcal{B}$ returns a promise of signature σ_i by running Isign. Otherwise, it has to simulate σ_{i^*}. It first picks random $\alpha, \beta \in \mathbb{Z}_p^*$ and sets $C_1 = u^\alpha$, $C_2 = A_{i^*} v^\alpha$, $C_3 = g_2^\beta$, $C_4 = B^\beta$, $CT_j = (C_2)^{t_j \beta}$ for $j = 1, \ldots, |\phi_{i^*}|$. Then it picks random $c, s_i, s_\alpha, s_\beta, s_\delta \in \mathbb{Z}_p^*$, sets $f = g_1^{s_i}$, and computes $R_1 - R_7$ using equations (1) – (7). It is easy to see that σ_{i^*} can be accepted by IVerify and it is indistinguishable from the output of Isign. We ignore the probability that the choice of c could cause a collision in the hash table since it is obviously negligible.

–Msign Queries. If $i \neq i^*$, $\mathcal{B}$ returns a promise of signature σ_i by running Msign. Otherwise, $\mathcal{B}$ simulates σ_{i^*} in the similar way as in the Isign queries except that it sets $f_i = f^{x_{i^*}^{-1}}$ and computes $R_1 - R_7$ using equations (8) – (14).

–KsReveal Queries. Given a keystone fix f produced in Isign queries, $\mathcal{B}$ returns $k = s_i$.

–Output. We divide $\mathcal{A}$ into two types of forgers. A type I forger outputs a forgery σ_i such that we can obtain a new SDH pairs $\langle A_i, x_i \rangle$ for $i \notin \{1,\ldots, n\}$. A type II forger outputs a forgery σ_i such that we can obtain a new SDH pairs $\langle A_i, x_i \rangle$ for $i = i^*$. We treat them in different way.

(1) For a type I forger, we instantiate q as $n+1$. Since $\mathcal{B}$ has n valid SDH pairs, it can perfectly simulate the challenger $\mathcal{C}$ to interact with $\mathcal{A}$, and thus we obtain a Type I forgery with probability ε.

(2) For a type II forger, we instantiate q as n. Since $\mathcal{B}$ has only $n-1$ valid SDH pairs, it will abort when a SecKeyGen query is made on i^*. However, it will not abort for a Isign/Msign query on i^*, since it can simulate a signature which is indistinguishable from the output of Isign/Msign. So we obtain a type II forgery with probability ε/n.

Given a forged promise of initial signature σ_i and the corresponding keystone $k = s_i$, since they are accepted by Verify and $C_1^{s_i}, C_2^{s_i}$ can be computed from C_1, C_2, k, we rewrite σ_i as $\langle T_i, y_i, C_1, C_2, C_3, C_4, CT_1,..., CT_{|\phi|}, c, k, s_\alpha, s_\beta, s_\delta \rangle$. This signature has the right structure for the application of the fork lemma [31]. Hence, by rewinding $\mathcal{B}$ and $\mathcal{A}$, we can obtain the second forged signature $\sigma_i' = \langle T_i, y_i, C_1, C_2, C_3, C_4, CT_1,..., CT_{|\phi|}, c', k', s'_\alpha, s'_\beta, s'_\delta \rangle$ on the same message M_i, with probabilities at least $(\varepsilon - 1/q)^2/(16q_H)$ or $(\varepsilon/n - 1/q)^2/(16q_H)$ for type I and type II forgers, respectively. Let $\triangle c = c - c'$ and similarly for $\triangle k$, $\triangle s_\alpha$, $\triangle s_\beta$ and $\triangle s_\delta$. We show as follows how to extract a new SDH pair from these two forgeries.

–Compute $\tilde{R}_1 - \tilde{R}_4$ and $\tilde{R}_1' - \tilde{R}_4'$ using equation (1) – (4).

–Compute $\tilde{R}_1 / \tilde{R}_1'$ to obtain $y_i = g_1^{\overline{x}_i}$ where $\overline{x}_i = \triangle k / \triangle c$.

–Compute $\tilde{R}_2 / \tilde{R}_2'$ to obtain $C_1 = u^{\overline{\alpha}}$ where $\overline{\alpha} = \triangle s_\alpha / \triangle c$.

–Compute $\tilde{R}_3 / \tilde{R}_3'$ to obtain $C_1^{\triangle k} = u^{\triangle s_\delta}$. It implies $\triangle s_\delta = \overline{\alpha} \triangle k$ from $C_1 = u^{\overline{\alpha}}$.

–Compute $\tilde{R}_4 / \tilde{R}_4'$ to obtain $(\hat{e}(g_1, g_2) / \hat{e}(C_2, w))^{\triangle c} = \hat{e}(C_2, g_2)^{\triangle k} \hat{e}(v, w)^{-\triangle s_\alpha} \hat{e}(v, g_2)^{-\triangle s_\delta}$.

By substituting $\overline{x}_i = \triangle k / \triangle c$, $\overline{\alpha} = \triangle s_\alpha / \triangle c$, $\triangle s_\delta = \overline{\alpha} \triangle k$ and rearranging the last equation, we have $\hat{e}(C_2 v^{-\overline{\alpha}}, w g_2^{\overline{x}_i}) = \hat{e}(g_1, g_2)$, so $\langle \overline{A}_i = C_2 v^{-\overline{\alpha}}, \overline{x}_i \rangle$ is a new SDH pair.

For the case where σ_i is a promise of matching signature, we let $\mathcal{B}$ interact with $\mathcal{A}$ just in the same way and from the fork lemma we also obtain two forgeries σ_i, σ_i' on same message M_i. Since they are produced from same input of $f = g_1^k$ and $z_i = g_1^{1/x_i}$, we know $f_i = f_i' = z_i^k$. By rewriting them as $\sigma_i = \langle f_i, T_i, z_i, C_1, C_2, C_3, C_4, CT_1,..., CT_{|\phi|}, c, s_i + k, s_\alpha, s_\beta, s_\delta \rangle$, $\sigma_i' = \langle f_i, T_i, z_i, C_1, C_2, C_3, C_4, CT_1,..., CT_{|\phi|}, c', s_i' + k, s'_\alpha, s'_\beta, s'_\delta \rangle$, and performing similar computation, we can also extract a new SDH pair as $\langle \overline{A}_i = C_2 v^{-\overline{\alpha}}, \overline{x}_i = \triangle s_i / \triangle c \rangle$.

4.3 Anonymity

Theorem 2. Under the XDH assumption, our ABCS scheme is anonymous in the random oracle model.

Proof. We prove the anonymity by contradiction. Suppose there exists an adversary $\mathcal{A}$ that wins the anonymity game with a non-negligible advantage ε, we can build an algorithm $\mathcal{B}$ that

has advantage $\varepsilon/2$ in breaking the XDH assumption. We first consider the case where $\mathcal{A}$ is challenged by a promise of initial signature.

Given a XDH instance $\langle g_1, g_1^{a_1}, g_1^{a_2}, g_1^{a_3} \rangle \in G_1^4$, $\mathcal{B}$ setups the system in the same way as the algorithm Setup(1^λ), except that it picks random r, $\beta \in \mathbb{Z}_p^*$ and sets $u = g_1^r$, $v = (g_1^{a_1})^r$. Since $\mathcal{B}$ knows *MSK*, it can perfectly simulate the challenger $\mathcal{C}$ to interact with $\mathcal{A}$. In the challenge phase, when receiving $\langle i_0, i_1, T_i, M_i \rangle$ from $\mathcal{A}$, $\mathcal{B}$ sets $C_1 = (g_1^{a_2})^{\eta}$, $C_2 = A_{i_b} (g_1^{a_3})^{\eta}$, $C_3 = g_2^{\beta}$, $C_4 = B^{\beta}$, $CT_l = (C_2)^{t_l \beta}$. If $a_3 = a_1 a_2$, it is a valid ElGamal encryption of A_{i_b} with the implicit setting of $\alpha = a_2$ and $v = u^{a_1}$. Otherwise it is encryption of a random message. Then $\mathcal{B}$ completes the simulation by choosing random $s_i, s_\alpha, s_\beta, s_\delta, c \in \mathbb{Z}_p^*$ and a random public key y_j for $j \neq i_0, j \neq i_1$, computing $R_1 - R_7$ using equations (1) – (7) and computing $C_1^{s_i}$, $C_2^{s_i}$ from C_1, C_2 and s_i. Clearly, the resultant signature can be accepted by IVerify.

Finally, $\mathcal{A}$ outputs its guess b'. If $b' = b$, $\mathcal{B}$ returns 1 indicating that $a_3 = a_1 a_2$. Otherwise it returns 0 indicating that $a_3 \neq a_1 a_2$. It's easy to see that the advantage of $\mathcal{B}$ in solving the XDH problem is $\Pr[b' = b \mid a_3 = a_1 a_2] + \Pr[b' = b \mid a_3 \neq a_1 a_2] = (1/2 + \varepsilon)/2 + (1/2)/2 = \varepsilon/2$.

In the case where $\mathcal{A}$ is challenged by a promise of matching signature, $\mathcal{B}$ produces the challenging signature in the similar way except that it picks two random values in G_1 as the public key z_i and the keystone fix f_i, and computes $R_1 - R_7$ using equations (8) – (14). From Lemma 2 we know that, from $\mathcal{A}$' point of view, the tuple $\langle f, z_i, f_i \rangle$ is indistinguishable from an honestly-generated one under the XDH assumption. Hence, by using the advantage of $\mathcal{A}$, $\mathcal{B}$ can also obtain an advantage of $\varepsilon/2$ in breaking the XDH assumption.

4.4 Fairness

Theorem 3. Our ABCS scheme is fair in the random oracle model.

Proof. Suppose our ABCS scheme is not fair, by the definition we know that one of two conditions must hold. We will reduce either of cases to a forgery of our ABCS, which contradicts to Theorem 1.

Case 1. Since $\mathcal{A}$ has never made a KsReveal query on f, from the Discrete Logarithm assumption we know that $\mathcal{A}$ must generate the keystone k by itself. Hence such k and σ_i lead to a forgery of our ABCS.

Case 2. If $\mathcal{A}$ produces the signature $\sigma_i = \langle f_i, s_i, \rho_i \rangle$ by itself, it immediately implies a forgery of our ABCS. If $\mathcal{A}$ receives σ_i from $\mathcal{C}$, it means $\mathcal{C}$ must have obtained a promise of initial signature $\sigma_j = \langle f, \rho_j \rangle$ from $\mathcal{A}$. Since k and σ_i are accepted by Verify, we have $f_i = z_i^k = (g_1^k)^{1/x_i}$, and from the condition $f_i =$ KfTran(f, usk_j) we further have $f = g_1^k$. Since $\sigma_j = \langle f, \rho_j \rangle$ is accepted by IVerify, k and σ_j must be accepted by Verify, which contradicts the second condition in the definition.

5. Conclusion

We recognize the importance of attributes in the area of fair exchange of digital signatures, and introduce the notion of attribute-based concurrent signatures. As an an interesting extension of concurrent signatures in the attribute-based setting, this primitive allows two parties fairly exchange their signatures only if each of them can convince the opposite party that he/she possesses certain attributes satisfying a given signing policy. We formalize this notion and present a construction which is secure in the random oracle model under the Strong Diffie-Hellman assumption and the eXternal Diffie-Hellman assumption.

The security of our construction relies on the random oracle model. As denoted in [32], some popular cryptosystems previously proved secure in the random oracle model are actually provably insecure when the random oracle is instantiated by any real-world hashing functions. So it is desirable to construct an ABCS in the standard model for more reliable security. We left it as an interesting open problem.

Acknowledgments

This work is partially supported by NSFC under Grant No .

References

[1] Liqun Chen, Caroline Kudla and Kenneth G. Paterson, "Concurrent Signatures," *Proc. of Advances in Cryptology – EUROCRYPT 2004*, LNCS 3027, Springer-Verlag, pp. 287-305, May 2-6, 2004.

[2] YangWang, ManHo Au and Willy Susilo, "Attribute-based optimistic fair exchange: How to restrict brokers with policies," *Theoretical Computer Science*, vol. 527, no. 3, pp. 83-96, 2014.

[3] Dalia Khader, "Attribute-based Authentication Scheme," *PhD thesis, University of Bath*, 2009.

[4] Dan Boneh and Xavier Boyen, "Short signatures without random oracles and the SDH assumption in bilinear groups," *Journal of Cryptology*, vol. 21, no. 2, pp. 149-177, 2008.

[5] Steven D. Galbraith, "Supersingular curves in cryptography," *Proc. of Advances in Cryptology – ASIACRYPT 2001*, LNCS 2248, Springer-Verlag, pp. 495-513, November 20, 2001.

[6] Dalia Khader, "Attribute based group signatures," *IACR Cryptology ePrint Archive*, *Report 2007/159*, http://eprint.iacr.org/2007/159.

[7] Keita Emura, Atsuko Miyaji and Kazumasa Omote, "A dynamic attribute-based group signature scheme and its application in an anonymous survey for the collection of attribute statistics," *Proc. of International Conference on Availability, Reliability and Security (ARES 2009)*, pp. 487-492, March 16-19, 2009.

[8] Willy Susilo, Yi Mu and Fangguo Zhang, "Perfect concurrent signature schemes," *Proc. of Information and Communications Security (ICICS 2004)*, LNCS 3269, Springer-Verlag, pp. 14-26, October 27-29, 2004.

[9] Guilin Wang, Feng Bao, and Jianying Zhou, "The fairness of perfect concurrent signatures," *Proc. of Information and Communications Security (ICICS 2006)*, LNCS 4307, Springer-Verlag, pp. 435-451, December 4-7, 2006.

[10] Sherman S. M. Chow and Willy Susilo, "Generic construction of (Identity-based) perfect concurrent signatures," *Proc. of Information and Communications Security (ICICS 2005)*, LNCS 3783, Springer-Verlag, pp. 194-206, December 10-13, 2005.

[11] Dongvu Tonien, Willy Susilo and Reihaneh Safavi-Naini, "Multi-party concurrent signatures," *Proc. of 9th International Conference on Information Security (ISC 2006)*, LNCS 4176, Springer-Verlag, pp. 131-145, August 30-September 2, 2006.

[12] Tsz H. Yuen, Duncan S. Wong, Willy Susilo and Qiong Huang, "Concurrent signatures with fully negotiable binding control," *Proc. of 5th International Conference on Provable Security (ProvSec 2011)*, LNCS 6980, Springer-Verlag, pp. 170-187, October 16-18, 2011.

[13] Xiao Tan, Qiong Huang and Duncan S. Wong, "Concurrent signature without random oracles," *Theoretical Computer Science*, vol. 562, pp. 194-212, 2015.
[14] Masayuki Abe, Miyako Ohkubo and Koutarou Suzuki, "1-out-of-n Signatures from a Variety of Keys," *Proc. of Advances in Cryptology –ASIACRYPT 2002*, LNCS 2501, Springer-Verlag, pp. 415-432, October 16-18, 2002.
[15] Khanh Nguyen, "Asymmetric concurrent signatures," *Proc. of Information and Communications Security (ICICS 2005)*, LNCS 3783, Springer-Verlag, pp. 181-193, December 10-13, 2005.
[16] Tatsuaki Okamoto and Kazuo Ohta, "How to simultaneously exchange secrets by general assumptions," *Proc. of 2nd ACM Conference on Computer and Communication Security (CCS 1994)*, pp. 184-192, August 30-September 2, 1994.
[17] N. Asokan, Victor Shoup and Michael Waidner, "Optimistic fair exchange of signatures," *Proc. of Advances in Cryptology – EUROCRYPT 1998*, LNCS 1403, Springer-Verlag, pp. 591-606, May 31- June 4, 1998.
[18] Hemanta K. Maji, Manoj Prabhakaran and Mike Rosulek, "Attribute-based signatures: achieving attribute-privacy and collusion-resistance," *IACR Cryptology ePrint Archive*, *Report 2008/328*, http://eprint.iacr.org/2008/328.
[19] Jin Li, Man Ho Au, Willy Susilo and Kui Ren, "Attribute-based signature and its application," *Proc. of 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2010)*, pp. 60-69, April 13-16, 2010.
[20] Siamak F. Shahandashti and Reihaneh Safavi-Naini, "Threshold attribute-based signatures and their application to anonymous credential systems," *Proc. of Progress in Cryptology –ASIACRYPT 2009*, LNCS 5580, Springer-Verlag, pp. 198-216, June 21-25, 2009.
[21] Javier Herranz, Fabien Laguillaumie, Benoît Libert and Carla Ràfols, "Short attribute-based signature for threshold predicates," *Proc. of the Cryptographers' Track at the RSA Conference 2011 (CT-RSA 2012)*, LNCS 7178, Springer-Verlag, pp. 51-67, February 27-March 2, 2012.
[22] Hemanta K. Maji, Manoj Prabhakaran and Mike Rosulek, "Attribute-based signatures," *Proc. of the Cryptographers' Track at the RSA Conference 2011 (CT-RSA 2011)*, LNCS 6558, Springer-Verlag, pp. 376-392, February 14-18, 2011.
[23] Tatsuaki Okamoto and Katsuyuki Takashima, "Efficient attribute-based signatures for non-monotone predicates in the standard model," *Proc. of the 14th Int. Conf. on Public Key Cryptography (PKC 2011)*, LNCS 6571, Springer-Verlag, pp. 35-52, March 6-9, 2011.
[24] Tatsuaki Okamoto and Katsuyuki Takashima, "Decentralized attribute-based signatures," *IACR Cryptology ePrint Archive*, *Report 2011/701*, http://eprint.iacr.org/2011/701.
[25] Syed T. Ali and B.B. Amberker, "Short Attribute-Based Group Signature without Random Oracles with Attribute Anonymity," *Proc. of Int. Symposium on Security in Computing and Communications (SSCC 2013)*, pp. 223-235, August 22-24, 2013.
[26] Keita Emura, Atsuko Miyaji and Kazumasa Omote, "A Selectable k-Times Relaxed Anonymous Authentication Scheme," *Proc. of 10th Int. Workshop on Information Security Applications (WISA 2009)*, LNCS 5932, Springer-Verlag, pp. 281-295, August 25-27, 2009.
[27] Dan Boneh, Xavier Boyen and Hovav Shacham, "Short Group Signatures," *Proc. of Advances in Cryptology – CRYPTO 2004*, LNCS 3152, Springer-Verlag, pp. 41-55, August 15-19, 2004.
[28] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss and Anna Lysyanskaya, "P-signatures and Noninteractive Anonymous Credentials," *Proc. of 5th Theory of Cryptography Conference (TCC 2008)*, LNCS 4948, Springer-Verlag, pp. 356-374, March 19-21, 2008.
[29] Melissa Chase, "Efficient Non-Interactive Zero-Knowledge Proofs for Privacy Applications," *PhD thesis, Brown University*, 2008.
[30] Vipul Goyal, Omkant Pandey, Amit Sahai and Brent Waters, "Attribute-based encryption for fine grained access control of encrypted data," *Proc. of 13th ACM Conference on Computer and Communications Security (CCS 2006)*, pp. 89-98, October 30-November 3, 2006.
[31] David Pointcheval and Jacques Stern, "Security arguments for digital signatures and blind signatures," *Journal of Cryptology*, vol. 13, no. 3, pp. 361-396, 2008.
[32] Ran Canetti Oded Goldreich and Shai Halevi, "The random oracle methodology, revisited," *Proc. of 30th ACM Symposium on Theory of Computing (STOC 1998)*, pp. 209–218, May 23-26, 1998.

Appendix A

For an access tree, let *index*(x) be a function which returns the index of a node x. For an interior node x, let l_x be the number of its children and k_x ($0 < k_x \leq l_x$) be the threshold value on node x. We also use *child*(x) and *dum*(x) to denote the set of attribute children and dummy children of x, respectively.

–BuildCTree(*Att*). Takes $Att = \{1,\ldots, |Att|\}$ as input, this algorithm build a central access tree by performing following steps.

(1) Build a tree *CT* where each attribute in *Att* is a leaf node.

(2) For an interior node x, add $l_x - k_x$ dummy nodes as its children, and change its threshold value from k_x to l_x. Let *Dum* be the set of all dummy nodes.

(3) Assign a unique index for each node in this tree.

(4) Assign a secret value $t_l \in \mathbb{Z}_p^*$ for each attribute leaf $l \in Att$, and a secret value $d_l \in \mathbb{Z}_p^*$ for each dummy node $l \in Dum$.

(5) For an interior node x, select a polynomial q_x of degree $l_x - 1$ such that it passes though (*index*(l), t_l) for each attribute $l \in child(x)$. Set $d_l = q_x(index(l))$ for each dummy node $l \in dum(x)$. Finally, pick a random $r_x \in \mathbb{Z}_p^*$ and set $q_x(0) = r_x$.

(6) Repeat the step (5) up to the root node, and output *CT* and $\{t_l\}_{l \in Att}$, $\{d_l\}_{l \in Dum}$.

–SimplifyCTree(*CT*, ϕ). Take as input the central tree *CT* and an attribute set $\phi \in Att$, this algorithm returns a simplified access tree by performing following steps.

(1) Delete the set of attributes $\{l\}_{l \in Att - \phi}$ from *CT*.

(2) Delete an interior node x along with x's descendants if it has children less than the threshold value l_x. Let T be the resultant access tree and φ be the set of dummy nodes in T.

(3) For all nodes x in of T except the root, compute L_x as follows, where c_x is the set of leaves in the depth 2 subtree with x as leaf node.

$$L_x = \prod_{k \in c_x \setminus \{x\}} \frac{-k}{index(x) - k}$$

(4) For each leaf node $l \in \phi \cup \varphi$, compute $\triangle_l = \prod_{node \in path_l} L_{node}$, where $Path_l = \{l, parent_1, \ldots, parent_n = \text{root}_T\}$ be the set of nodes that appears in the path from l to root node of T.

(5) Output T and $\{\triangle_l\}_{l \in \phi}$, $\{\triangle_l\}_{l \in \varphi}$.

Given an access tree T, we can compute the value r of the root node as:

$$r = \sum_{l \in \phi} \triangle_l t_l + \sum_{l \in \varphi} \triangle_l d_l$$