Aggregate Proxy Signature and Verifiably Encrypted Proxy Signature

Jin Li^{1} *, Kwangjo Kim^{1}, Fangguo Zhang^{2} and Xiaofeng Chen^{3}

^{1} International Research center for Information Security (IRIS)
Information and Communications University(ICU)
58-4 Hwaam-dong Yusong-ku, Taejon, 305-732, Korea
^{2} Department of Electronics and Communication Engineering
Sun Yat-Sen University, Guangzhou, 510275, P.R.China
^{3} Department of Computer Science
Sun Yat-Sen University, Guangzhou, 510275, P.R.China

Abstract. An aggregate signature is a single short string that convinces any verifier that, for all $1 \leq i \leq n$, signer i signed message m_i, where the n signers and n messages are distinct. The main motivation of aggregate signatures is compactness. In this paper, the concept of aggregate proxy signature (APS) is first proposed to compact the proxy signatures. Furthermore, a concrete APS scheme is constructed, which can be proved to be secure under the security model of APS. Additionally, as an application of APS, the concept of verifiably encrypted proxy signature (VEPS) is also first proposed in this paper, which can be used in contract signing. The VEPS allows the original signer to delegate another to sign the contract on its behalf. Finally, a VEPS construction is derived from the APS, which can be easily proved to be secure from the security of APS.

Keywords: Proxy signature, Aggregate signature, Random oracle, Bilinear pairings

1 Introduction

A proxy signature protocol allows an original signer to delegate its signing power to another entity, called proxy signer, to sign messages on its behalf. The delegated proxy signer can compute a proxy signature that can be verified by anyone with access to the original signer's public key. Proxy signatures have many practical applications such as in distributed system etc. [10] and are one of important cryptographic protocols. The concept of proxy signature was first introduced by Mambo, Usuda, and Okamoto [8] in 1996. After Mambo et al.'s first scheme was published, many various types of proxy signature schemes have been proposed such as short proxy signature scheme [5,7], one-time proxy signatures [16]. Also, there are a lot of proxy signature schemes were found flaws such as [11]. The

* This work was partially supported by the 2nd stage of Brain Korea 21 Project sponsored by the Ministry of Education and Human Resources Development, Korea

main reason is the lack of formal security model. Until 2003, the formal security model was proposed in [1]. In this security model, a public key infrastructure setting (PKI) is also assumed, where each entity holds a public and secret key pair.

The notion of aggregate signature schemes was introduced in 2003 by Boneh, Gentry, Lynn and Shacham [3]. Basically, aggregating signatures means compressing n signatures on n distinct messages from n distinct users into a unique (shorter) signature. This is useful in many real-world applications. For example, certificate chains in a hierarchical PKI of depth n consist of n signatures by n different CAs on n different public keys. By using an aggregate signature scheme, this chain can be compressed down to a single aggregate certificate. After the concept of aggregate signatures was proposed, many types of aggregate signatures have been presented such as identity-based aggregate signatures [4], sequential aggregate signatures [13].

In this paper, the concept of aggregate proxy signature (APS) is first proposed. Consider the following situations: n proxy signers have generated n proxy signatures on n different messages on behalf of the same original signer. To verify these proxy signatures, the ordinary method is to verify them one by one, which costs large storage and computation. Reducing the amount of memory required to store these proxy signatures and the computational time required to verify their validity is the motivation for the concept of APS. An APS is obtained from n different initial proxy signatures, ideally in such a way that: (1) the length of the aggregate proxy signature is smaller than the sum of the length of the n initial proxy signatures; (2) verifying the correctness of the aggregate proxy signature costs less than verifying the n initial proxy signatures one by one. If an aggregate proxy signature is verified as valid, then the receiver is convinced that the n initial signatures are valid. On the other hand, if the aggregate signature is invalid, the receiver is convinced that some initial proxy signature is not valid.

Next, we show an application of APS to verifiably encrypted proxy signature (VEPS). It is known that verifiably encrypted signatures can be used in applications such as online contract signing [8]. Suppose Alice wants to show Bob that she has signed a message, but does not want Bob to possess her signature of that message. Alice can achieve this by encrypting her signature using the public key of a trusted third party, and sending this to Bob along with a proof that she has given him a valid encryption of her signature. Bob can verify that Alice has signed the message, but cannot deduce any information from her signature. Later, in the protocol, if Alice is unwilling or unable to reveal her signature, Bob can ask the third party to reveal Alice's signature.

However, consider the following situation: If either Alice or Bob is busy, they can delegate their signing power to the other party, which is called as proxy signer, to sign the contract on behalf of him or her. So, the concept of VEPS is first presented in this paper to solve this problem. In this case, the proxy signer of Alice, for example, wants to show Bob that it has signed a message on behalf of Alice, but does not want Bob to possess its proxy signature on that message. The proxy signer can achieve this by encrypting its proxy signature using the public key of a trusted third party, and sending this to Bob along with a proof that it has given him a valid encryption of its proxy signature. Bob can verify that the proxy signer has signed the message on behalf of Alice, but cannot deduce any information from the encrypted signature. Later, in the protocol, if the proxy signer is unwilling or unable to reveal its signature, Bob can ask the third party to reveal its proxy signature.

Contributions. In this work we introduce the notion and security model of APS. Roughly speaking, the new concept allows to efficiently manage multiple proxy signatures addressed to a specific verifier. Furthermore, a concrete construction is presented, which can be proved to be secure in the security model. Additionally, the concept of VEPS is first proposed in this paper, which can be used in contract signing. It allows the original signer to delegate another to sign the contract on its behalf. A VEPS construction is also derived from the APS, which can be easily proved to be secure from the security of APS.

2 Preliminaries

2.1 Definition

Definition 1. (APS) *An APS scheme consists of 7 algorithms: (KeyGen, (D,P), PSign, PVerify, Aggregate, Verify). The algorithms are specified as follows:*

- *KenGen The key generation algorithm, on input security parameter 1^k, outputs user's public key pk and corresponding secret key sk.*
- *(D,P) is a pair of interactive algorithms forming the proxy-designation protocol. The input to each algorithm includes two public keys pk_o, pk_i. D also takes as input the secret key sk_o, and P also takes as input the secret key sk_i. As result of the interaction, the expected local output of P is sk_p, a proxy signing key that user pk_i uses to produce proxy signatures on behalf of user pk_o.*
- *PSign The proxy signature generation algorithm, that takes as input a secret key sk_p, a message m, returns the signature σ.*
- *PVerify The proxy signature verification algorithm, that takes input public key pk_o, pk_i, a message m and a proxy signature σ, outputs 1 if it is a valid proxy signature for m relative to pk. Otherwise, output 0.*
- *Aggregate The aggregate algorithm, that takes as input n different proxy signatures $\sigma_1, \cdots, \sigma_n$ of distinct messages $m_1, \cdots, m_n$ correctly signed by different users $pk_1, \cdots, pk_n$, outputs an aggregate proxy signature σ;*
- *Verify The aggregate proxy signature verification algorithm, that takes as input pk_o, $pk_1, \cdots, pk_n$, n messages $m_1, \cdots, m_n$ and σ, returns 1 or 0 for accept or reject, respectively.*

2.2 Security Requirements

Adversary's attack capabilities are modelled by providing it access to certain oracles. We now introduce the oracles we will need and provide the adversary with different subsets of this set of oracles.

– $\mathcal{APS}$ Oracle: The aggregate proxy signing oracle, on input message m_1, $\cdots$, m_n, pk_o, $\mathcal{L} = \{y_1, \cdots, y_n\}$ for aggregate proxy signature, returns an aggregate proxy signature σ such that $\mathsf{APV}(pk_o, \mathcal{L}, m_1, \cdots, m_n, \sigma) = 1$.
– $\mathcal{KR}$ Oracle: The key registration oracle, on input key pair (pk, sk), first checks if sk is indeed the secret key of pk. Then it stores (pk, sk) as a valid registered key pair if it is. Otherwise, reject and output a special symbol $\perp$.
– $\mathcal{DE}$ Oracle: The delegation oracle, on input any registered public key pk_i, and original public key pk_o, its secret key sk_o, returns a delegation on the public key pk_i.
– $\mathcal{RA}$ Oracle: The random oracle, on input m_i, outputs a randomly value r_i chosen in the domain of the hash function.

There are two types of unforgeability to consider in APS: *Delegation unforgeability* and *aggregate proxy signature unforgeability*. *Delegation unforgeability* means that even if the adversary asks for polynomial users' delegation, it is still hard to output a forgery delegation that the original signer has not delegated. *Aggregate proxy signature unforgeability* means that, except the proxy signers, anyone else (even if the origin signer) cannot generate valid aggregate proxy signature on behalf of these proxy signers.

2.2.1 Delegation Unforgeability

Delegation unforgeability for aggregate proxy signature is defined as in the following game involving an adversary $\mathcal{A}$.

1. Let $(pk_o, sk_o) \leftarrow \mathsf{KenGen}(1^k)$. $\mathcal{A}$ is given pk_o and the public parameters.
2. $\mathcal{A}$ accesses to $\mathcal{RA}$ Oracle, $\mathcal{DE}$ Oracle, and $\mathcal{KR}$ Oracle.

The adversary $\mathcal{A}$ wins the game if he can output $m_1^*, \cdots, m_n^*$, $\mathcal{L}$=(pk_1, $\cdots$, pk_n,), such that $\mathcal{L}$ includes a public key pk_i that is not equal to any query of $\mathcal{DE}$ oracle and σ^* is a valid aggregate proxy signature with respect to pk_o. The advantage of the adversary is the probability that he wins the game.

Definition 2. *(Delegation Unforgeability) An aggregate proxy signature scheme is delegation unforgeability secure if no probabilistic polynomial time (PPT) adversary has a non-negligible advantage in the above game.*

2.2.2 Aggregate Proxy Signature Unforgeability

We formalize this intuition as the aggregate chosen-key security model. In this model, the adversary $\mathcal{A}$ is given a single proxy signer's public key. His goal is the existential forgery of an aggregate proxy signature. We give the adversary power to choose all public keys except the challenge public key. The adversary is also given access to a proxy signing oracle on the challenge key. His advantage, $Adv^{AggSig}(A)$, is defined to be his probability of success in the following game.

– *Setup*: The aggregate forger $\mathcal{A}$ is provided with the challenge proxy signer's public key pk_1 and original signer's key pair (sk_o, pk_o), generated at random.

- $\mathcal{A}$ requests proxy signatures with pk_1 on behalf of original signer pk_o, adaptively.
- $\mathcal{A}$ accesses to $\mathcal{RA}$ Oracle and $\mathcal{KR}$ Oracle.
- Finally, A outputs $n-1$ additional public keys $pk_2, \cdots, pk_n$, which have been queried to $\mathcal{KR}$ Oracle. Here n is at most N, a game parameter. These keys, along with the initial key pk_1, will be included in $\mathcal{A}$'s forged aggregate. $\mathcal{A}$ also outputs messages $m_1^*, \cdots, m_n^*$, and, finally, an aggregate proxy signature σ^* by the n users on behalf of pk_o, each on his corresponding message. The forger wins if the aggregate signature σ^* is a valid aggregate on messages $m_1^*, \cdots, m_n^*$ under public keys $pk_1, \cdots, pk_n$, and σ^* is nontrivial, i.e., $\mathcal{A}$ did not request a proxy signature on m_1^* under pk_1.

An aggregate forger $\mathcal{A}$ $(t, q_H, q_S, n, \epsilon)$-breaks an n-user APS scheme in the aggregate chosen-key model if: $\mathcal{A}$ runs in time at most t; $\mathcal{A}$ makes at most q_H queries to the random oracle and at most q_S queries to the $\mathcal{APS}$ oracle; $Adv^{AggSig}(A)$ is at least ϵ; and the forged aggregate signature is by at most N users. An aggregate signature scheme is $(t, q_H, q_S, n, \epsilon)$-secure against existential forgery in the aggregate chosen-key model if no forger $(t, q_H, q_S, n, \epsilon)$-breaks it.

Definition 3. *An APS is secure if $Adv^{AggSig}(A)$ is negligible for any PPT adversary $\mathcal{A}$.*

2.3 Preliminaries

Before present our results, we review the definitions of groups equipped with a bilinear pairings and a related assumption. Let $\mathbb{G}$ be a (multiplicative) cyclic group of prime order p. Let g be a generator of $\mathbb{G}$. We also let $\hat{e}$ be a bilinear map such that $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$ with the following properties:

1. *Bilinearity*: For all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}$, $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$.
2. *Non-degeneracy*: $\hat{e}(g, g) \neq 1$.
3. *Computability*: There exists an efficient algorithm to compute $\hat{e}(u, v)$.

Definition 4. Computational Diffie-Hellman Assumption*: Given g, g^x, $g^y \in (\mathbb{G})^3$ for unknown $x, y \in_R \mathbb{Z}_p^*$, it is hard to compute g^{xy} for any PPT algorithm.*

3 An APS Scheme

Let $\mathbb{G}$ be a bilinear group where $|\mathbb{G}| = p$. Define a bilinear map $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$. Meanwhile, define two collision-resistant hash functions $H_1 : \mathbb{G} \rightarrow \mathbb{G}$ and $H_2 : \{0,1\}^* \rightarrow \mathbb{G}$. The construction of such hash function can be found in [2]. Then the system parameters are params$=(\mathbb{G}, \mathbb{G}_1, \hat{e}, g, H_1, H_2)$.

1. **KenGen.** For original signer, it picks $x_o \in \mathbb{Z}_p$ and outputs $(x_o, y_o = g^{x_o})$ as its key pair. The original signer's secret key is x_o and the public key is y_o. For user i, it chooses $x_i \in \mathbb{Z}_p$ and outputs $(x_i, y_i = g^{x_i})$ as its key pair. The user $i's$ secret key is x_i and the public key is y_i.

2. **D.** In order to delegate his signing capability to user i, the original signer y_o, on input y_i, computes $S_i = [H_1(y_i)]^{x_o}$ as the corresponding delegation.
3. **P.** Given S_i, the user i computes its proxy signing key as $sk_i = (x_i, S_i)$.
4. **PSign.** Assuming the proxy signer i with public key y_i wants to generate signature on message m on behalf of y_o, it computes $H_2(m)^{x_i}$ and outputs the proxy signature $\sigma = S_i \cdot H_2(m)^{x_i}$.
5. **PVerify.** On input the aggregate proxy signature σ, message m and y_o, y_i, accept if $\hat{e}(\sigma, g) = \hat{e}(H_1(y_i), y_o)\ \hat{e}\ (H_2(m), y_i)$.
6. **Aggregate.** On input n proxy signatures $\sigma_1, \cdots, \sigma_n$ on n different messages $m_1, \cdots, m_n$ by n distinct proxy signers $y_1, \cdots, y_n$, output $\sigma{=}\sigma_1 \cdots \sigma_n$ as the aggregate proxy signatures.
7. **Verify.** On input σ on n different messages $m_1, \cdots, m_n$ by n distinct proxy signers $y_1, \cdots, y_n$, accept if $\hat{e}(\sigma, g) = \prod_{i=1}^{n}(\hat{e}(H_1(y_i), y_o)\ \hat{e}\ (H_2(m_i), y_i))$.

3.1 Security Results

Theorem 1. *In random oracle model, the APS scheme is delegation unforgeable if CDH assumption holds in bilinear groups.*

Proof. If there exists an adversary $\mathcal{A}$ breaks the scheme, then we show there exists an algorithm $\mathcal{C}$ that, by interacting with $\mathcal{A}$, solves the CDH problem. Our algorithm $\mathcal{C}$ described below solves CDH problem for a randomly given instance $\{g, g^x, g^y\}$ and asked to compute g^{xy}. The details are as follows.

$\mathcal{C}$ runs $\mathcal{A}$ on input $y_o = g^x$ as target user's public key, handling all of $\mathcal{A}$'s requests and answering all $\mathcal{A}$'s queries as follows:

– H-queries: Assume $\mathcal{A}$ makes at most q_{H_1} times to H_1-oracle and q_{H_2} times to H_2-oracle, respectively. When $\mathcal{A}$ queries m_i to H_2-oracle, $\mathcal{C}$ answers $H_2(m_i) = g^{\hat{m_i}}$ for a random $\hat{m_i} \in Z_p$. Furthermore, $\mathcal{C}$ randomly chooses a $s \in [1, q_{H_1}]$ and prepares $t_i \in Z_p$ for $1 \leq i \leq q_{H_1}$. When $\mathcal{A}$ queries y_i to H_1-oracle, $\mathcal{C}$ answers $H_1(y_i) = g^{t_i}$ if $i \neq s$. Otherwise, $H_1(y_s) = g^y$ if $i = s$.
– Key Registration Queries: If $\mathcal{A}$ requests to register a new user i by outputting (x_i, y_i), $\mathcal{C}$ stores these keys as valid registered key pair.
– Delegation Queries: If $\mathcal{A}$ requests to designates i with registered public key y_i, it assumes $\mathcal{A}$ has requested H_1 query on y_i. If $i \neq s$, $\mathcal{C}$ knows the value t_i such that $H_1(y_i) = g^{t_i}$. So cert is $y_o^{t_i}$. Otherwise, it aborts.

Finally, $\mathcal{A}$ outputs a forgery of aggregate proxy signature ($m_1^*, \cdots, m_n^*$, $\mathcal{L}$, σ^*), such that $\mathcal{L}$ includes a public key y^* that is not equal to any query of $\mathcal{DE}$ Oracle and σ^* is a valid aggregate proxy signature with respect to pk_o and $\mathcal{L}$ on message m^*. Assume $\mathcal{L}{=}\{y_1, \cdots, y_n\}$, such that $y_s = y^*$. It satisfies $\hat{e}(\sigma^*, g) = \prod_{i=1}^{n}(\hat{e}(H_1(y_i), y_o)\ \hat{e}\ (H_2(m_i^*), y_i))$, which implies $\sigma^* = \prod_{i=1}^{n} H_1(y_i)^x H_2(m_i^*)^{x_i}$. Because $H_2(m_i) = \hat{m_i}$, $H_1(y^*) = g^y$, and $H_1(y_i) = g^{t_i}$ for $y_i \neq y^*$, we can compute $g^{xy}{=}\sigma^* / \prod_{i=1}^{n} y_i^{\hat{m_i^*}} \prod_{i \in \{1, \cdots, n\} \backslash s} y_o^{t_i}$ and solve the CDH problem.

It is easy to see that if $\mathcal{A}$ outputs a forgery of aggregate proxy signature with probability ϵ, then CDH problem can be solved with probability about $\frac{1}{q_{H_1}} \cdot \epsilon$. So, we can say that the APS scheme is delegation unforgeability secure in the random oracle if CDH assumption holds.

Theorem 2. *In random oracle model, the APS scheme is aggregate proxy signature unforgeable if CDH assumption holds in bilinear groups.*

Proof. We show there exists an algorithm $\mathcal{C}$ that, if there exists an adversary $\mathcal{A}$ breaks the scheme, by interacting with $\mathcal{A}$, solves the CDH problem. Our algorithm $\mathcal{C}$ described below solves CDH problem for a randomly given instance $\{g, g^x, g^y\}$ and asked to compute g^{xy}.

$\mathcal{C}$ chooses x_o and computes $y_o = g^{x_o}$. Then it sends (x_o, y_o) to the adversary. $\mathcal{C}$ runs $\mathcal{A}$ on input $y_1 = g^x$ as target proxy user's public key, handling all of $\mathcal{A}$'s requests and answering all $\mathcal{A}$'s queries as follows:

– H-queries: Assume $\mathcal{A}$ makes at most q_{H_1} times to H_1-oracle and q_{H_2} times to H_2-oracle, respectively. When $\mathcal{A}$ queries y_i to H_1-oracle, $\mathcal{C}$ answers $H_1(y_i) = g^{r_i}$ for a random $r_i \in Z_p$. Furthermore, $\mathcal{C}$ randomly chooses a $s \in [1, q_{H_2}]$. When $\mathcal{A}$ queries m_i to H_2-oracle, $\mathcal{C}$ answers $H_2(m_i) = g^{t_i}$ if $i \neq s$. Otherwise, $H_2(m_s) = g^y$ if $i = s$.
– Key Registration Queries: If $\mathcal{A}$ requests to register a new user by outputting $(x, y = g^x)$, $\mathcal{C}$ stores these keys as valid registered key pair.

Finally, $\mathcal{A}$ outputs a forgery of aggregate proxy signature $(m_1^*, \cdots, m_n^*, \mathcal{L} = \{y_1, \cdots, y_n\}, \sigma^*)$, such that σ^* is a valid aggregate proxy signature with respect to pk_o and $\mathcal{L}$ on message $m_1^*, \cdots, m_n^*$. It satisfies $\hat{e}(\sigma^*, g) = \prod_{i=1}^{n}(\hat{e}(H_1(y_i), y_o)\ \hat{e}\ (H_2(m_i^*), y_i))$. If $m_1^* = m_s$, we have $H_2(m_1^*) = g^y$ and $H_2(m_i^*) = g^{t_i}$ for $m_i \neq m_s$. Finally, $\mathcal{C}$ can compute $g^{xy} = \sigma / (\prod_{i \in \{1,\cdots,n\}} y_o^{r_i} \prod_{i \in \{1,\cdots,n\} \backslash s} y_i^{t_i})$. Otherwise, $\mathcal{C}$ aborts.

It is easy to see that if $\mathcal{A}$ outputs a forgery of APS with probability ϵ, then CDH problem can be solved with probability about $\frac{1}{q_{H_2}} \cdot \epsilon$. So, we can say that the APS scheme is secure in the random oracle if CDH assumption holds.

In this paper, we only deal with the proxy signatures on behalf the same original signer. But, in many applications, the proxy signatures on behalf different signers are also practical. So, we think how to solve this question is also interesting, including its security model and scheme. We do not show details here for space.

4 Verifiably Encrypted Proxy Signature Scheme

Next, we show an application of APS to VEPS. Verifiably encrypted signatures (VES) are used in applications such as online contract signing [8]. However, if one of the two party is busy, they can delegate their signing power to the other party, which is called as proxy signer, to sign the contract on behalf of him or her. So, the concept of VEPS is first presented to solve this problem. From the APS, a VEPS can be easily constructed.

Definition 5. (VEPS)*A VEPS comprises nine algorithms: KeyGen, (D,P), PSign, PVerify, AdjKeyGen, VEPSigCreate, VEPSigVerify, and Adjudicate, provide the verifiably encrypted signature capability. The algorithms are described below. We also refer to the trusted third party as the adjudicator.*

– *KeyGen, (D,P), PSign, and PVerify are the same with their corresponding definitions in APS.*
– *AdjKeyGen. This algorithm generates key pair (ASK, APK) for the adjudicator.*
– *VEPSigCreate. Given a proxy signing key sk_p, message m, adjudicator's public key APK, it outputs the verifiably encrypted proxy signature σ.*
– *VEPSigVerify. Given original public key pk_o, proxy signer's public key pk_i, a message m, an adjudicator's public key APK, and a signature σ, verify if σ is a valid verifiably encrypted proxy signature on m.*
– *Adjudicate. Given an adjudicator's secret key ASK, and a verifiably encrypted proxy signature σ on some message m, extract and output σ', an ordinary proxy signature on m of proxy signer pk_i on behalf of pk_o.*

We require three security properties of VEPS: validity, unforgeability, and opacity, which is similar to [3].

– Validity requires that ordinary proxy signature verify, verifiably encrypted proxy signatures verify, and that adjudicated verifiably encrypted signatures verify, i.e., that PVerify(m,PSign(m)), VESigVerify(m,VESigCreate(m)) and PVerify(m,Adjudicate(VESigCreate(m))) hold for all m.
– There are two types of unforgeability, including delegation unforgeability and verifiably encrypted proxy signature unforgeability. Delegation unforgeability requires that it be difficult to forge a valid verifiably encrypted proxy signature of an unauthorized user. Verifiably encrypted proxy signature unforgeability requires that it be difficult to output a verifiably encrypted proxy signature by anyone else, even the original user, except the right proxy signer.
– Opacity requires that it be difficult, given a VEPS, to extract an ordinary proxy signature on the same message, given access to a VEPS creation oracle and an adjudication oracle, maybe along with a hash (random) oracle. The opacity can easily be achieved in our construction based on the assumption that given an APS of n signatures it is difficult to extract the individual proxy signatures.

Let $\mathbb{G}$ be a bilinear group where $|\mathbb{G}| = p$. Define a bilinear map $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$. Meanwhile, define two collision-resistant hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$ and $H_2 : \{0,1\}^* \rightarrow \mathbb{G}$. The system parameters are params=$(\mathbb{G}, \mathbb{G}_1, \hat{e}, g, H_1, H_2)$.

1. **KenGen.** For original signer, it picks $x_o \in \mathbb{Z}_p$ and outputs $(x_o, y_o = g^{x_o})$ as its key pair. The original signer's secret key is x_o and the public key is y_o.
2. **D.** In order to delegate his signing capability to user with registered public key pair $(x, y = g^x)$, then original signer, on input y, computes $S = [H_1(y)]^{x_o}$ as the corresponding delegation.
3. **P.** Given S, the user computes its proxy signing key as $sk_p = (x, S)$.
4. **PSign.** Assume the proxy signer wants to generate proxy signature on message m on behalf of original signer with public key y_o. It computes the proxy signature $\sigma = S \cdot [H_2(m)]^x$.

5. **PVerify.** On input σ, a message m and y_o, y, accept if $\hat{e}(\sigma, g) = \hat{e}(H_1(y), y_o)\ \hat{e}\ (H_2(m), y)$.
6. **AdjKeyGen.** For adjudicator, it picks $x_a \in \mathbb{Z}_p$ and outputs $(x_a, y_a = g^{x_a})$ as its key pair. The adjudicator's secret key is x_a and the public key is y_a.
7. **VEPSigCreate.** Given a proxy signing key $sk_p = (x, S)$, a message $m \in \{0,1\}^*$, and adjudicator's public key y_a, it signs as follows:
   a. Compute $h = H_2(m)$, where $h \in \mathbb{G}$, and $\sigma = h^x \cdot S$.
   b. Select r at random from $\mathbb{Z}_p$, set $u = g^r$ and compute $\sigma' = (y_a)^r$.
   c. Aggregate σ and σ' as $\omega = \sigma\sigma'$.

   Finally, the verifiably encrypted proxy signature is the pair (ω, u). (This can also be viewed as ElGamal encryption of σ under the adjudicator's key.)
8. **VEPSigVerify.** Given public keys y_o, y, a message m, adjudicator's public key y_a, and a verifiably encrypted proxy signature (ω, u), set $h = H_2(m)$; accept if $\hat{e}(\omega, g) = \hat{e}(y_o, H_1(y)) \cdot \hat{e}(y, h) \cdot \hat{e}(u, y_a)$ holds.
9. **Adjudicate.** Given adjudicator's private key x_a, and a verifiably encrypted proxy signature (ω, u) on some message m, ensure that the verifiably encrypted proxy signature is valid by running algorithm VEPSigVerify; then output the proxy signature $\sigma = \omega / u^{x_a}$.

4.1 Security Results

Our VEPS scheme depends on the assumption that given an aggregate signature of k signatures it is difficult to extract the individual signatures. We posit that it is difficult to recover the individual signatures σ_i given their aggregate σ, and the messages. In fact, for the VEPS is only constructed from an aggregate proxy signature of 2 proxy signatures, its security can be reduced to the following problem [3].

Definition 6. *Given g^a, g^b, g^x, g^y, and $g^{ax+by} \in \mathbb{G}$, it is hard to output the value g^{ax}.*

In the bilinear aggregate proxy signature scheme, it is difficult to extract individual proxy signatures, under the aggregate extraction assumption [3]. For more details, the reader can be referred to [3]. We can get the following two security results easily from the security of APS with the above aggregate extraction problem [3]:

Theorem 3. *In random oracle model, the VEPS scheme is unforgeable (delegation unforgeable and verifiably encrypted proxy signature unforgeable) if CDH assumption holds in bilinear groups.*

Theorem 4. *In random oracle model, the VEPS scheme achieves opacity if CDH assumption holds in bilinear groups.*

5 Conclusion

In this paper we introduce the notion and security model of APS, which allows to compress the proxy signatures on different messages from different proxy signers into one. Meanwhile, a concrete APS scheme is presented, and it can be proved to be secure in the security model. Additionally, as an application of APS, the concept of verifiably encrypted proxy signature is also proposed in this paper, which can be used in contract signing. It allows the original signer to delegate another to signing the contract. A VEPS construction is also derived from the APS and can be easily proved to be secure from the properties of the corresponding APS.

References

1. A.Boldyreva, A.Palacio, B.Warinschi. Secure Proxy Signature Schemes for Delegation of Signing Rights. Cryptology ePrint Archive, Report 2003/096. Available at http://eprint.iacr.org, 2003.
2. D.Boneh, B.Lynn, H. Shacham. Short Signatures from the Weil Pairing. Asiacrypt 2001, LNCS 2248, Springer-Verlag, pp. 514-532, 2001.
3. D.Boneh, C. Gentry, H.Shacham, B. Lynn. Aggregate and verifiably encrypted signatures from bilinear maps, Eurocrypt'03, LNCS 2656, Springer-Verlag, pp. 416-432, 2003.
4. C. Gentry, Z. Ramzan. Identity-Based Aggregate Signatures, PKC 2006, LNCS 3958, pp. 257-273, Springer-Verlag, 2006.
5. X. Huang, Y. Mu, W. Susilo, F. Zhang, X. Chen. A Short Proxy Signature Scheme: Efficient Authentication in the Ubiquitous World. EUC Workshopspp, pp. 480-489, Springer-Verlag, 2005.
6. B.G. Kang, J.H. Park, S.G. Hahn. A Certificate-Based Signature Scheme, CT-RSA'04, LNCS 2964, pp. 99-111, Springer-Verlag, 2004.
7. J. Li and Y. Wang. A short provably secure proxy signature scheme. Chinese Journal of Electronics, 2006, Vol.15, No. 4: 721-724.
8. M. Mambo, K.Usuda, and E.Okamoto. Proxy signatures for delegating signing operation, Proceedings of the 3rd ACM Conference on Computer and Communications Security (CCS), ACM, pp. 48-57, 1996.
9. S.Malkin, S.Obana, and M.Yung. The hierarchy of key evolving signatures and a characterization of proxy signatures, Eurocrypt'04, LNCS 3027, pp. 306-322, 2004.
10. B.C. Neuman. Proxy based authorization and accounting for distributed systems, Proceedings of the 13th International Conference on Distributed Computing Systems, pp. 283-291, 1993.
11. G.Wang, F. Bao, J. Zhou, R.H. Deng. Security Analysis of Some Proxy Signatures, ICISC 2003, LNCS 2971, Springer-Verlag, pp. 305-319, 2004.
12. H.X.Wang, J.Pieprzyk. Efficient One-time proxy signatures, Asiacrypt 2003, Springer-Verlag, pp. 507-522, 2004.
13. H. Zhu, F. Bao, T. Li, Y.Wu. Sequential aggregate signatures for wireless routing protocols, IEEE WCNC 2005, 2436-2439, 2005.