Identity Based Self Delegated Signature - Self Proxy Signatures

S. Sharmila Deva Selvi, S. Sree Vivek, S.Gopinath, C. Pandu Rangan
TCS Lab, Department of CSE,
Indian Institute of Technology Madras (IITM)
Chennai, India
Email: {sharmila,svivek,gopinath,prangan}@cse.iitm.ac.in

***Abstract*—A proxy signature scheme is a variant of digital signature scheme in which a signer delegates his signing rights to another person called proxy signer, so that the proxy signer can generate the signature of the actual signer in his absence. Self Proxy Signature (SPS) is a type of proxy signature wherein, the original signer delegates the signing rights to himself (Self Delegation), there by generating temporary public and private key pairs for himself. Thus, in SPS the user can prevent the exposure of his private key from repeated use. In this paper, we propose the first identity based self proxy signature scheme. We give a generic scheme and a concrete instantiation in the identity based setting. We have defined the appropriate security model for the same and proved both the generic and identity based schemes in the defined security model.**

Keywords: Identity Based Cryptography, Self-Proxy Signatures, Delegation, Random Oracle Model.

I. Introduction

The notion of proxy signature schemes dates back to 1996, proposed by Mambo, Usuda and Okamoto in their seminal paper [14]. In proxy signature scheme, a user Alice, called the original signer delegates her signing rights to another user Bob, called the proxy signer. A verifier can distinguish between a normal signature and a proxy signature but then is convinced that the message is authenticated by Alice. Proxy signatures have a number of applications, including e-commerce, mobile agents and distributed shared objects. The original signer Alice sends a signature on the message warrant which consists the rules governing the delegation to Bob the proxy signer. Bob can now generate a new proxy private key with the help of Alice and sign on behalf of Alice. In 1998, Oded Goldreich, Birgit Pfitzmann and Ronald L. Rivest [5] introduced delegation schemes where a user delegates certain rights to himself. Their motivation was, even though a user has a long-term permanent key, which is used to receive some personalized access rights, the user may wish to delegate these rights to a new temporary possibly short-term keys which he creates to use on his laptop when on travel, to avoid having to store his primary secret key on the vulnerable laptop. They have succeeded without relying on special-purpose (e.g., tamper-proof) hardware installed in the laptop and have proposed several schemes. However, their schemes work for signatures in the Public Key Infrastructure (PKI) setting. In Self Proxy Signature (SPS), a user delegates his signing rights to himself, i.e. the user can generate multiple pairs of temporary public and private keys. The lifetime of the temporary keys can be controlled by creating proper message warrants, depending on the application.

A. Motivation

Self proxy signatures are used in scenarios where the user wants to create new key pairs from the existing key pair. The newly generated key pair is called as "*Temporary Key*" pair and the existing key pair from which the Temporary key pair is generated is called as "*Permanent Key*" pair. It is important to note that while permanent key pair is generated by PKG and it is done only once per user, the temporary key pair is generated by the user and can be done any number of times. We explain three situations, where self delegation is useful and these situations commonly arise in practice.

To reduce the probability of exposure of the permanent private key: Nowadays, numerous internet services such as, internet banking, home trading, on-line payments, electronic commercial services and other secure online transactions rely on Public Key Cryptography. Public key cryptography plays an important role in the authentication of users in these systems. This causes a potential security threat, namely "increase in the probability of the permanent private key being exposed". For instance, if the permanent private key is used in an insecure computing environment such as a public PC or a friend's PDA, a malicious program can plunder the private key by searching the memory where the private key is stored, or by hijacking the password for decrypting the enciphered private key. Moreover, this gives room to access any other online services of the user, which rely on this plundered permanent private key. It is to be noted that this situation was discussed in [12].

To create weak Temporary Keys means less number of bits: This situation is common in secure communication protocols such as SSL/TLS, where the session key exchange between a server and its client is done using Public Key Cryptography. When SSL was designed, United States export regulations limited RSA encryption key lengths to 512 bits for exportable applications. Unfortunately, a 512-bit permanent RSA key presents an attractive target for attack. Thus a server who wish to communicate with both domestic and exportable clients would like to have two keys one with 1024 bits and another key with 512 bits. This feature is called ephemeral RSA and this allows communication between an exportable client and domestic server with permanent strong key. In this scenario, the server generates a temporary 512 bit key which is signed with its strong permanent key.

To significantly improve amortized signature generation and verification cost: Besides the two reasons mentioned above, we show a significant reduction in the total signing cost during the period of validity of temporary keys. For instance, for signing n messages in our scheme we may use $(2+n)$ or $(3+n)$ scalar point multiplications. While direct deployment would incur a cost of $2n$ or $3n$ point multiplications. More detailed comparisons are done towards the end of this paper.

Related Work: In 1996, Mambo et al introduced the concept of a proxy signature scheme [14]. Since then, many proxy signature schemes have been proposed [9][11]. The first multi-proxy signature scheme was proposed in 2000 [7]. In a multi-proxy signature scheme, an original signer could authorize a proxy signing group as his proxy agent. The proxy signature on a message on behalf of the original signer can be generated by the group members only if all the members in the proxy signing group cooperate. A contrary concept, proxy multi-signature was introduced by Yi et al. in 2000 [17]. A proxy multi-signature scheme is one in which a designated proxy signer can generate the signature on behalf of a group of original signers. Another kind of proxy signature scheme is multi-proxy multi-signature scheme, proposed by Hwang in [8] .

The concept of identity based cryptography was introduced by Adi Shamir in his seminal work [16] in the year 1984. The core idea of identity based cryptography is to use any arbitrary string that uniquely identifies a user as his public key. Identity based cryptography serves as an efficient alternative to Public Key Infrastructure (PKI) based systems, where the certificate management and verification of the validity of a user public key are too cumbersome. Although, the concept of self proxy was touched upon by Boldyreva et al. [2], they are not using any temporary keys for carrying out delegations. Only the permanent keys were used for both original and proxy signing and verification. This is a PKI based system. However, this system is shown to have weaknesses by Malkin et al. [13] and they proposed a new scheme based on key insulated signature schemes. To the best of our knowledge, there is no identity based self delegated signature scheme available in the literature and ours is the first attempt in this direction. Kim et al. [10] have proposed a PKI based self proxy signature scheme.

Our Contribution: Our contribution in this paper is three fold. First, we give a formal security model for identity based self proxy signatures. Next, we show that the scheme by Kim et al. [10] is existentially forgeable and finally, we propose a generic identity based self proxy signature scheme and a concrete instantiation of the same. We formally prove the security of both the generic and concrete schemes in the newly proposed security model. Both our proofs rely on the random oracle assumption.

II. Preliminaries

We review the basic requirements and assumptions used in our paper in this section.

A. Bilinear Pairing

Let $\mathbb{G}_1$ be an additive cyclic group generated by P, with prime order q, and $\mathbb{G}_2$ be a multiplicative cyclic group of the same order q. A bilinear pairing is a map $\hat{e}:\mathbb{G}_1\times\mathbb{G}_1\rightarrow\mathbb{G}_2$ with the following properties.

- ***Bilinearity.*** For all $P,Q,R\in_R\mathbb{G}_1$ and $a,b\in_R\mathbb{Z}_q^*$
  - $\hat{e}(P+Q,R)=\hat{e}(P,R)\hat{e}(Q,R)$
  - $\hat{e}(P,Q+R)=\hat{e}(P,Q)\hat{e}(P,R)$
  - $\hat{e}(aP,bQ)=\hat{e}(P,Q)^{ab}$
- ***Non-Degeneracy.*** There exist $P,Q\in\mathbb{G}_1$ such that $\hat{e}(P,Q)\neq I_{\mathbb{G}_2}$, where $I_{\mathbb{G}_2}$ is the identity element of $\mathbb{G}_2$.
- ***Computability.*** There exists an efficient algorithm to compute $\hat{e}(P,Q)$ for all $P,Q\in\mathbb{G}_1$.

B. Computational Assumptions

In this section, we review the computational assumptions related to bilinear maps that are relevant to the protocols we discuss.

1) Computation Diffie-Hellman Problem (CDHP):: Given $(P,aP,bP)\in\mathbb{G}_1^3$ for unknown $a,b\in\mathbb{Z}_q^*$, the CDH problem in $\mathbb{G}_1$ is to compute abP.

Definition. The advantage of any probabilistic polynomial time algorithm $\mathcal{A}$ in solving the CDH problem in $\mathbb{G}_1$ is defined as

$$Adv_{\mathcal{A}}^{CDH}=Pr\left[\mathcal{A}(P,aP,bP)=abP\mid a,b\in\mathbb{Z}_q^*\right]$$

The *CDH Assumption* is that, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{CDH}$ is negligibly small.

C. Notations:

Q_A*:* An identity based public key of a user with identity ID_A.
D_A*:* An identity based private key of a user with identity ID_A.
P_A*:* A non-identity based public key of a user with identity ID_A and also Temporary public of the same user.
U_A*:* A non-identity based private key of a user with identity ID_A and also Temporary private of the same user.
P_{pub}*:* Master public key used by the PKG of the identity based system.

σ_{war}: An identity based warrant signature.
σ_{sp}: A non-identity based proxy signature. σ: Self proxy signature which is a combination of $\langle\sigma_{war}, \sigma_{sp}\rangle$.

III. Review and Weakness of Self Proxy Signature Scheme By Kim et al. [10]

We review the scheme due to Kim et al. [10] and propose the weakness of the scheme in this section. Let (x_a, y_a) be the original public and private key of the signer Alice. The relation between (x_a, y_a) is y_a=g^{x_a}.

Self Proxy Key Generation: The signer Alice chooses $k, x_t \in \mathbb{Z}_q^*$ and computes $r = g^k \ mod \ p$ and $y_t = g^{x_t} \ mod \ p$. Alice computes $x_p = k + (x_a + x_t)H(m_w) \ mod \ q$ as the temporary self proxy private key and computes $y_p = g^{x_p} \ mod \ p$ as the corresponding public key.

Signing: The signer Alice chooses $k' \in \mathbb{Z}_q^*$ randomly and computes r'=$g^{k'} \ mod \ p$, $s' = k' + x_pH(m) \ mod \ q$ and sends $(m, (r', s'), r, m_w, y_t)$ to the verifier Bob.

Verification: The verifier Bob recovers the self proxy public key y_p as $y_p = r(y_a y_t)^{H(m_w)} \ mod \ p$ and checks whether $g^{s'} \stackrel{?}{=} r' y_p^{H(m)} \ mod \ p$. If the equality holds, the verifier Bob accepts (r', s') as the valid self proxy signature.

A. Weakness of Self Proxy Signature Scheme By Kim et al.

The forger $\mathcal{F}$ can produce any number of forged signatures by using a single signature on a message m signed by the original signer. This is shown below:

Let $s' = k' + x_pH(m) \ mod \ q$ be the signature on message m, where $r' = g^{k'} \ mod \ p$, this signature is obtained during the training phase of the forgery game.

The forger $\mathcal{F}$ divides the signature component s' with $H(m)$ and multiplies it with $H(m^*)$ and thus obtains $s^* = k'\frac{H(m^*)}{H(m)} + x_pH(m^*)$. $\mathcal{F}$ computes $r^* = r'^{\frac{H(m^*)}{H(m)}} \ mod \ p$. Now, $(m^*, (r^*, s^*), r, m_w)$ is a valid signature on m^*. Hence a forgery.

IV. Generic Framework and Security Model for Identity Based SPS (IBSPS)

In this section, we give the generic framework and the security model for identity based self proxy signature scheme. The basic idea behind the construction of an Identity Based Self Proxy Signature (IBSPS) scheme is to extract an identity based private key from the PKG and construct a temporary private key / public key pair using the identity based permanent private key and the system parameters. It is to be noted that the temporary key pairs are generated by user without any interaction from the PKG. The PKG works once for each user and generates permanent key of the user. The temporary public key and the warrant details for the session are signed with the identity based private key of the user and the message is signed with the corresponding temporary private key. Thus, the signature on the message consists of two components now; (a) the signature on the message with the temporary private key (b) the identity based signature on the temporary public key and the warrant details.

A. Generic Framework for IBSPS

An identity based self proxy signature scheme consists of the following nine algorithms: **Setup**, **Extract**, **GenTempKey**, **WarrantSign**, **WarrantVerify**, **ProxySign**, **ProxyVerify**, **SPSSign** and **SPSVerify**. The algorithms are described below:

Setup: This is a combination of an identity based system setup and a non-identity based initialize algorithm. The input to this algorithm is the security parameter 1^κ. The PKG run's this algorithm to produce the public parameters $params$, which is published globally and the master private key Msk, kept secret by the PKG. The public parameters include a master public key P_{pub}, cryptographic hash functions and the definition of the groups used in the scheme.

Extract: This algorithm is executed by the PKG. It is executed once for each user at the time of registration with the PKG. The PKG takes the master private key Msk and the identity ID_A of user A as input and computes the private key D_A corresponding to the identity ID_A.

GenTempKey: The user who wants to generate temporary private / public key pairs for various sessions executes this algorithm. The algorithm takes $params$ as input and produces the temporary (private key, public key) pair (U_A, P_A).

WarrantSign: This algorithm is executed by each user in the system to generate the signature on the message warrant m_w, which is publicly verifiable (m_w consists of the details regarding the duration of the delegation and the public key for the current duration). The algorithm when executed by user A, takes the user identity ID_A, the message warrant m_w, temporary public key P_A, the corresponding private key D_A, and $params$ as input and outputs the identity based signature σ_{war} on the message warrant m_w.

WarrantVerify: In order to verify the validity of the message warrant m_w, a verifier executes this algorithm. The input to this algorithm are $params$, the signer identity ID_A, the message warrant m_w and the signature σ_{war} on m_w by the signer. This algorithm returns `True` if σ_{war} is a valid signature on m_w, otherwise returns `False`.

ProxySign: The input to this algorithm are $params$, the temporary proxy private key U_A and the actual message to be signed m. This algorithm is executed by the signer to generate a signature (σ_{sp}) on m using the temporary proxy private key generated by the **GenTempKey** algorithm.

ProxyVerify: The input to this algorithm are $params$, the signer identity ID_A, the temporary public key P_A corresponding to user A and the signature σ_{sp} on message m. This algorithm is executed by a verifier who wants to verify the validity of σ_{sp} on m. The output is `True` if σ_{sp} is a valid signature on m, otherwise outputs `False`.

SPSSign: The self proxy signature generation algorithm is executed by the signer with identity ID_A. The input to this algorithm are the identity ID_A, the permanent private / public key pair (D_A, Q_A), the temporary private / public key pair (U_A, P_A), a message warrant m_w and the message m to be signed. It is to be noted that σ_{war} is not executed each time during the generation of an SPS but it is executed once for each session and is reused. The signature generation procedure is given below:

- $\sigma_{war} \leftarrow WarrantSign(m_w, P_A, ID_A, D_A)$.
- $\sigma_{sp} \leftarrow ProxySign(m, P_A, ID_A, U_A)$

The signature $\sigma = \langle m_w, m, \sigma_{war}, \sigma_{sp}\rangle$ is the output of this algorithm.

Note: If the warrant sign was already generated for a session by executing $WarrantSign(m_w, P_A, ID_A, D_A)$ with the temporary private / public key pair (U_A, P_A), the signer can directly call $ProxySign(m, P_A, ID_A, U_A)$ to get σ_{sp}.

SPSVerify: In order to verify the validity of an identity based SPS σ=$\langle m_w, m, \sigma_{war}, \sigma_{sp}\rangle$, the verifier checks whether the message warrant m_w is valid by checking $WarrantVerify(\sigma_{war}, ID_A, m_w) \stackrel{?}{=}$ `True` and whether the signature on message m is valid by checking $ProxyVerify(\sigma_{sp}, ID_A, m, P_A) \stackrel{?}{=}$ `True`. If both the checks are valid this algorithm outputs `True` else it outputs `False`.

B. Security Model for the Unforgeability of IBSPS

Unforgeability is the most general notion of security for any digital signature scheme. Unforgeability ensures that the digital signature scheme is secure against a forger who can forge the signature of a legitimate user. The stronger notion of unforgeability is *existential unforgeability against adaptively chosen messages and identity (only for identity based schemes) attacks*. We propose the security model for identity based SPS in this section. The formal definition for the unforgeability of an IBSPS is defined as a game (EUF-IBSPS-CMA) between a challenger $\mathcal{C}$ and a forger $\mathcal{F}$ described below:

Setup Phase: $\mathcal{C}$ runs the Setup algorithm with the security parameter 1^κ and sends the system parameters $params$ to $\mathcal{F}$.

Training Phase: $\mathcal{F}$ performs polynomially bounded number of queries, as described below in an adaptive manner (i.e., each query may depend on the responses to the previous queries).

Extract query : $\mathcal{F}$ produces an identity ID_A as input to this oracle and obtains the identity based private key D_A corresponding to the identity ID_A from $\mathcal{C}$.

GetTempKey query: $\mathcal{F}$ produces an identity ID_A and receives from $\mathcal{C}$ the temporary private, public keys U_A and P_A corresponding to ID_A.

WarrantSign query: $\mathcal{F}$ gives an identity ID_A, a message warrant m_w and the temporary public key P_A corresponding to ID_A as input. $\mathcal{C}$ computes and returns the warrant sign σ_{war} to $\mathcal{F}$.

ProxySign query: $\mathcal{F}$ submits a message m, a signer identity ID_A and the corresponding temporary public key P_A as input and requests the proxy sign on m. $\mathcal{C}$ generates the proxy signature σ_{sp} only if P_A was not chosen by $\mathcal{F}$ and returns σ_{sp} to $\mathcal{F}$.

Existential Forgery: At the end of the **Training Phase**, $\mathcal{F}$ produces a forgery $\sigma^* = \langle m_w, m, \sigma^*_{sp}, \sigma^*_{war}\rangle$ for an identity ID^* with temporary public key P_{ID^*}. $\mathcal{F}$ wins the EUF-IBSPS-CMA game if the forgery σ^* submitted by $\mathcal{F}$ meets one of the following constraints:

Case 1: - The warrant signature σ^*_{war} is a valid forgery and $\mathcal{F}$ should not have queried the WarrantSign oracle with $(ID^*, m^*_w, P_{ID^*}, Q_{ID^*})$ as input and has not queried the permanent private key corresponding to ID^*.

Case 2: - The proxy signature σ^*_{sp} is a valid forgery and $\mathcal{F}$ should not have queried the ProxySign oracle with (ID^*, m^*, P_{ID^*}) as input and has not queried the temporary private key corresponding to P^*_{ID}.

V. Generic Identity Based SPS Scheme (Gen_IBSPS)

In this section, we propose the generic construction for identity based SPS scheme (Gen_IBSPS) and prove the unforgeability of Gen_IBSPS. We make use of an identity based signature scheme and a non-identity based signature scheme as the basic building blocks for our generic construction. Let the identity based signature scheme be denoted as $IBS = \langle IBS.Setup, IBS.Extract, IBS.Sig, IBS.Ver\rangle$, and the non-identity based signature be denoted as $nonIBS = \langle nonIBS.Initialize, nonIBS.KeyGen, nonIBS.Sig, nonIBS.Ver\rangle$. As security requirements, we require both the schemes, IBS and $nonIBS$ should be existentially unforgeable under adaptive chosen message attack. Examples of IBS can be one of the schemes from [4], [6], [15], [1], [16] and $nonIBS$ can be Schnorr, EC-DSA or BLS [3] signature.

A. The Generic Scheme

As described in the previous section the generic self proxy signature scheme consists of the following nine algorithms:

Setup(1^κ)**:** The PKG publishes the system parameters $params$ after executing $IBS.Setup(1^\kappa)$ and $nonIBS.Initialize(1^\kappa)$ algorithms.

Extract(ID_A)**:** The PKG executes $D_A = IBS.Extract(ID_A)$ and sends the permanent private key D_A to the user A, through a secure channel and the permanent public key Q_A can be computed publicly.

GenTempKey$(params, ID_A)$**:** This algorithm generates the temporary private / public key pair for a given identity ID_A is obtained as $(U_A, P_A) = nonIBS.KeyGen(ID_A)$.

WarrantSign(m_w, P_A, ID_A, D_A)**:** The warrant signature is generated as σ_{war}=$IBS.Sig(m_w,\ P_A,\ ID_A, D_A)$

WarrantVerify$(m_w, ID_A, \sigma_{war})$**:** Verification of the warrant signature (σ_{war}) on the message warrant m_w is performed as {`True`,`False`} $\stackrel{?}{=} IBS.Ver(m_w, P_A, ID_A, \sigma_{war})$

ProxySign(m, ID_A, P_A, U_A)**:** The proxy signature on the message m by user with identity ID_A and temporary private key U_A is generated as $\sigma_{sp} = nonIBS.Sig(m, ID_A, P_A, U_A)$

ProxyVerify$(m, ID_A, P_A, \sigma_{sp})$**:** The verification of the proxy signature σ_{sp} on message m with respect to the identity ID_A and temporary public key P_A is performed as {`True`,`False`} $\stackrel{?}{=} nonIBS.Ver(m, P_A, ID_A, \sigma_{sp})$

SPSSign$(m_w, m, ID_A, P_A, U_A, Q_A, D_A)$**:** This algorithm uses the $WarrantSign$ and $ProxySign$ algorithms to generate the warrant signature σ_{war} and proxy signature σ_{sp}. The self proxy signature on the message m and message warrant m_w can be described as $\sigma = \langle m_w, m, \sigma_{war}, \sigma_{sp}\rangle$

SPSVerify$(m_w, m, Q_A, P_A, \sigma)$**:** The output of this algorithm is `True` if both `True`$\leftarrow WarrantVerify(\sigma_{war})$ and `True`$\leftarrow ProxyVerify(\sigma_{sp})$; else output `False`.

Note: It is to be noted that, the proof of unforgeability of Gen_IBSPS is given in the selective identity model.

B. Proof of Unforgeability of Gen_IBSPS

Theorem 1: *If there exists a forger $\mathcal{F}$, who is capable of breaking the EUF-Gen_IBSPS-CMA security of the Gen_IBSPS scheme with a non-negligible advantage, then we can efficiently construct an algorithm $\mathcal{C}$, which is capable of breaking the EUF-CMA security of the underlying IBS or $nonIBS$ scheme with the same advantage.* **Proof:** The proof for unforgeability of the Gen_IBSPS scheme is viewed as an interactive game between algorithms $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{C}$ and $\mathcal{F}$, as shown in Fig. 1. Algorithm $\mathcal{B}_1$ represents the challenger for the IBS scheme and $\mathcal{B}_2$ represents the challenger for the $nonIBS$ scheme. $\mathcal{B}_1$ and $\mathcal{B}_2$ challenges the algorithm $\mathcal{C}$ to forge the IBS and the $nonIBS$ systems respectively. Let $\mathcal{F}$ be a forger, who is capable of breaking the EUF-Gen_IBSPS-CMA security of the Gen_IBSPS scheme. Algorithm $\mathcal{C}$ can make use of $\mathcal{F}$ to forge either IBS or $nonIBS$. We briefly summarize the roles of $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{C}$ and $\mathcal{F}$ below.

- $\mathcal{B}_1$ and $\mathcal{B}_2$ acts as the challengers for IBS and $nonIBS$ schemes respectively.
- $\mathcal{C}$ acts as forger for both IBS and $nonIBS$ schemes.
- $\mathcal{C}$ also acts as the challenger for the forger $\mathcal{F}$. $\mathcal{F}$ is assumed to be capable of breaking the EUF-Gen_IBSPS-CMA security of Gen_IBSPS scheme.

Setup Phase: During this phase the algorithm $\mathcal{B}_2$ generates a set of private / public key pairs $\langle P_i, U_i\rangle$, for $i = 1$ to n and sends them to the challenger $\mathcal{C}$. $\mathcal{C}$ stores the tuple $\langle i, P_i, U_i\rangle$ in a list F_1. $\mathcal{B}_1$ challenges $\mathcal{C}$ to generate a forgery of an IBS signature for the identity ID^* on any arbitrary message. $\mathcal{B}_2$ challenges $\mathcal{C}$ to generate a forgery of a $nonIBS$ signature for the identity ID^* with temporary public key P_{ID^*}. $\mathcal{C}$ gives ID^* and P_{ID^*} to $\mathcal{F}$. $\mathcal{F}$ should not query the $Extract$ oracle with ID^* as input and should not query the temporary private key U_{ID^*} corresponding to P_{ID^*}.

Training Phase: The following oracle accesses are provided by $\mathcal{B}_1$ to $\mathcal{C}$.

IBS.Extract(ID_i)*:* On input an identity ID_i, $\mathcal{B}_1$ returns the corresponding identity based private key D_i if $ID \neq ID^*$. $\mathcal{B}_1$ maintains the list L_1 to store $\langle ID_i, D_i\rangle$.

IBS.Sig(m, P_i, ID_i, Q_i)*:* With (m, P_i, ID_i, Q_i) as input, $\mathcal{B}_1$ responds with an identity based signature σ_{war} on the message m. $\mathcal{B}_1$ maintains L_2 to store the tuple $\langle m, P_i, ID_i, Q_i, \sigma_{war}\rangle$.

The following oracle access is provided by $\mathcal{B}_2$ to $\mathcal{C}$.

nonIBS.Sig(m, ID_i, P_i)*:* On giving (m, ID_i, P_i) as input, $\mathcal{B}_2$ responds with a non-identity based sign σ_{sp} on message m. $\mathcal{B}_2$ maintains the list L_3 to store the tuple $\langle m, ID_i, P_i, \sigma_{sp}\rangle$.

$\mathcal{F}$ has access to all the following oracles during the EUF-Gen_IBSPS-CMA game, which are controlled by $\mathcal{C}$.

Extract(ID_i)*:* In order to respond to this query by $\mathcal{F}$, $\mathcal{C}$ checks if $ID_i \neq ID^*$, if so then $\mathcal{C}$ queries $\mathcal{B}_1$ with ID_i as input, i.e. queries **IBS.Extract**(ID_i) to $\mathcal{B}_1$. $\mathcal{B}_1$ returns D_i to $\mathcal{C}$. $\mathcal{C}$ stores the tuple $\langle ID_i, D_i\rangle$ into the list F_2 and returns D_i to $\mathcal{F}$.

GenTempKey(ID_i)*:* To respond to this query by $\mathcal{F}$, $\mathcal{C}$ fetches into the list F_1 for the tuple of the form $\langle i, P_i, U_i\rangle$ and returns both (P_i, U_i) to $\mathcal{F}$.

WarrantSign(m_w, P_i, ID_i, Q_i)*:* For this query by $\mathcal{F}$, $\mathcal{C}$ queries $\mathcal{B}_2$ for the identity based signature on m_w with inputs (m_w, P_i, ID_i, Q_i) as **IBS.Sig**(m_w, P_i, ID_i, Q_i). The response of the query, namely the signature σ_{war} is returned to $\mathcal{C}$ and $\mathcal{C}$ sends it to $\mathcal{F}$.

ProxySign(m, ID_i, P_i)*:* For this query by $\mathcal{F}$, $\mathcal{C}$ queries $\mathcal{B}_2$ as **nonIBS.Sig**(m, ID_i, P_i) for the non-identity based signature on m with (m, ID_i, P_i) as input. $\mathcal{B}_2$ returns σ_{sp} to $\mathcal{C}$ which is send to $\mathcal{F}$ as response.

Forgery Phase: At the end of the training phase, $\mathcal{F}$ produces a forgery $\sigma^* = \langle m^*, m^*_w, \sigma^*_{war}, \sigma^*_{sp}\rangle$ and gives it to $\mathcal{C}$. $\mathcal{C}$ verifies the forgery and responds as follows:

Case 1: If $\mathcal{F}$ has not queried the WarrantSign oracle with $m^*_w, P_{ID^*}, ID^*, Q^*$ as input but has query the ProxySign

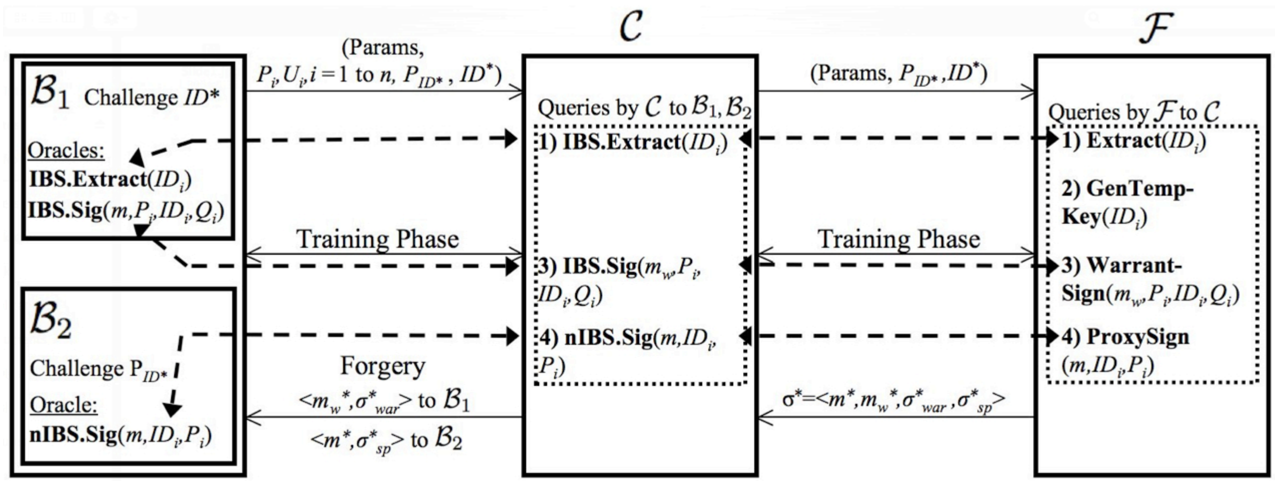

Figure 1. Relation among ($\mathcal{B}_1$, $\mathcal{B}_2$), $\mathcal{C}$ and $\mathcal{F}$

oracle with m^*, ID^*, P_{ID^*} as input then $\mathcal{C}$ submits σ^*_{war} to $\mathcal{B}_1$.

Case 2: If $\mathcal{F}$ has queried the WarrantSign oracle with $m^*_w, P_{ID^*}, ID^*, Q^*$ as input did not query the ProxySign oracle with m^*, ID^*, P_{ID^*} as input then $\mathcal{C}$ submits σ^*_{sp} to $\mathcal{B}_2$.

Analysis: Due to lack of space, we skip the formal probability analysis but the following is easy to see. Suppose ϵ is the advantage of $\mathcal{F}$ in the EUF-Gen_IBSPS-CMA game, which is denoted as $Adv^{wins}_{\mathcal{F}} = \epsilon$.

- If the forgery falls into **Case 1**, then $\mathcal{C}$ is capable of forging the underlying IBS scheme with almost the same advantage ϵ.
- If the forgery falls into **Case 2**, then $\mathcal{C}$ is capable of forging the underlying $nonIBS$ scheme with almost the same advantage ϵ.

VI. A Concrete IBSPS Scheme

In this section, we provide a concrete instantiation of Gen_IBSPS scheme and prove the security of the scheme in the proposed security model. We have used a variant of the IBS signature scheme in [15] and the $nonIBS$ scheme in [3], to construct our concrete IBSPS scheme.

A. The Scheme

The algorithms in the IBSPS scheme are described below:

Setup(1^κ):

Let $\mathbb{G}_1$ be an additive group and $\mathbb{G}_2$ be a multiplicative group both of same prime order q. The PKG chooses a generator $P \in_R \mathbb{G}_1$, picks three cryptographic hash functions defined as $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_2 : \{0,1\}^* \times \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_1$ and $H_3 : \{0,1\}^* \times \mathbb{Z}_q^* \times \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_1$ and chooses a bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. The PKG computes $P_{pub} = sP$, where,$s \in_R \mathbb{Z}_q^*$ is the master private key.

Extract(ID_A): Given a user's identity ID_A as input the PKG computes $Q_A = H_1(ID_A) \in \mathbb{G}_1$ and $D_A = sH_1(ID_A)$ and sends D_A to the user A.

GenTempKey$(params, ID_A)$**:** The signer generates his temporary private / public key pair by choosing $x_A \in_R \mathbb{Z}_q^*$ and computing $P_A = x_AP$. Now, the temporary private / public key pair of the user is $\langle P_A, U_A = x_A\rangle$.

WarrantSign(m_w, P_A, ID_A, D_A)**:** The warrant signature on the warrant m_w is generated as follows.

- Compute $R = rP$, where $r \in_R \mathbb{Z}_q^*$.
- $V_{war} = D_A + rH_2(m_w, R, Q_A, P_A)$.
- $\sigma_{war} = \langle V_{war}, R, P_A\rangle$ is the warrant signature.

Note: Warrant signature is independent of the message and hence need to be computed only once for the entire validity period of the warrant.

WarrantVerify$(m_w, ID_A, \sigma_{war})$**:** Given the identity ID_A of the signer, a message warrant m_w and a warrant signature σ_{war} on m_w, this algorithm returns `True` if the following check holds; otherwise returns `False`.

$$\hat{e}(V_{war}, P) \stackrel{?}{=} \hat{e}(R, H_2(m_w, R, P_A, Q_A))\hat{e}(P_{pub}, Q_A)$$

ProxySign(m, ID_A, P_A, U_A)**:** Given a pair of temporary keys $\langle P_A, U_A\rangle$ the self proxy sign on a message m with warrant m_w by the user with identity ID_A, the proxy signature can be generated as

- $V_{sp} = x_AH_3(m, P_A, \alpha, ID_A)$ where $\alpha \in_R \mathbb{Z}_q^*$ (Note that $x_A = U_A$).
- σ_{sp} is $\langle V_{sp}, \alpha\rangle$

ProxyVerify$(m, ID_A, P_A, \sigma_{sp})$**:** Given the Self Proxy signature σ_{sp} and the temporary public key P_A check whether:

$$\hat{e}(V_{sp}, P) \stackrel{?}{=} \hat{e}(P_A, H_3(m, P_A, \alpha, ID_A))$$

If the check holds, return `True`; otherwise return `False`.

SPSSign$(m, m_w, ID_A, P_A, U_A, Q_A, D_A)$**:** In order to generate a self proxy signature σ, the user with identity ID_A, permanent key pair (Q_A, D_A), temporary key pair (P_A, U_A), a message warrant m_w and a message m to be signed, the user performs the following:

- $\sigma_{war} \leftarrow WarrantSign(m_w, P_A, ID_A, D_A)$.
- $\sigma_{sp} \leftarrow ProxySign(m, P_A, ID_A, U_A)$

Output the self proxy signature $\sigma = \langle\sigma_{war}, \sigma_{sp}\rangle$.

Note: If the warrant sign σ_{war} on the tuple (m_w, P_A, ID_A, D_A) was already generated this need not be recomputed for every message, the signer need to call only the algorithm $ProxySign(m, P_A, ID_A, U_A)$ to generate σ_{sp}.

SPSVerify$(m, m_w, Q_A, P_A, \sigma)$**:** If the output of the algorithm $WarrantVerify(m_w, ID_A, \sigma_{war})$ = `True` and $ProxyVerify(m, ID_A,\ P_A, \sigma_{sp})$ = `True` then output `True` else output `False`.

B. Security Proof for IBSPS

Theorem 2: *If there exists a forger $\mathcal{F}$ who is capable of breaking the EUF-IBSPS-CMA security of the IBSPS scheme with non-negligible advantage, then we can efficiently construct an algorithm $\mathcal{C}$, which can solve CDHP with almost the same advantage of $\mathcal{F}$.*

Proof: The challenger $\mathcal{C}$ is given a random instance of CDHP, say (P, aP, bP), $\mathcal{C}$'s aim is to compute abP. Let us assume that there is a forger $\mathcal{F}$ who is capable of breaking the EUF-IBSPS-CMA security of our identity based self proxy signature scheme IBSPS. $\mathcal{C}$ simulates the system with the following oracles H_1, H_2, H_3, $Extract$, $GenTempKey$, $WarrantSign$, $ProxySign$ and $SPSSign$. The forger $\mathcal{F}$ can query these oracles which are controlled by the challenger $\mathcal{C}$. Each oracle maintains a list to maintain the consistency of the replies to the queries.

Setup: $\mathcal{C}$ sets up the system parameters in the following way.

- $\mathcal{C}$ chooses the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ and the generator $P \in \mathbb{G}_1$ as given in the CDHP instance.
- Sets the master public key P_{pub} = bP, which is a part of $CDHP$ instance. It is to be noted that $\mathcal{C}$ does not know b.
- Models all the hash functions as random oracles.
- Selects a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$.
- Delivers $(\hat{e}, \mathbb{G}_1, \mathbb{G}_2, P, P_{pub})$ to $\mathcal{F}$ as $params$.

Training Phase: $\mathcal{F}$ performs polynomially bounded number of queries, in an adaptive manner (i.e., each query may depend on the responses to the previous queries) during this phase.

H_1 *Oracle:* Let q_{H1} be the number of queries asked by $\mathcal{F}$. $\mathcal{C}$ selects a random index γ, where $1 \leq \gamma \leq q_{H1}$. $\mathcal{C}$ doesn't reveal γ to $\mathcal{F}$. When $\mathcal{F}$ generates the γ^{th} query to this oracle, $\mathcal{C}$ decides to fix the corresponding identity (ID_γ) as the target identity for the challenge phase. $\mathcal{C}$ maintains a list L_1 to consistently reply the H_1 oracle queries. $\mathcal{C}$ replies as follows.

- Searches in list L_1 and checks whether a matching tuple corresponding to ID_i exists. If it exists, $\mathcal{C}$ returns the value x_iP.
- Otherwise, $\mathcal{C}$ performs the following:
  - If $ID_i = ID_\gamma$ then, sets $H_1(ID_i) = aP$ and adds tuple $\langle ID_\gamma, aP, \perp, \perp\rangle$ to L_1.
  - If $ID_i \neq ID_\gamma$ then, chooses $x_i \in_R \mathbb{Z}_q^*$, sets $H_1(ID_i) = x_iP$, computes $D_i = x_iP_{pub}$ and adds the tuple $\langle ID_i, x_iP, x_i, D_i\rangle$ to the list L_1.

H_2 *Oracle:*(m_w, R, P_i, Q_i). Let L_2 be the list associated with this oracle. If the oracle was not queried previously with (m_w, R, P_i, Q_i) as input, $\mathcal{C}$ chooses $r \in_R \mathbb{Z}_q^*$, computes $\mathtt{H}_2^i = rP$, adds the tuple $\langle m_w, r, R, P_i, Q_i, \mathtt{H}_2^i\rangle$ to the list L_2 and returns $\mathtt{H}_2^i$ to $\mathcal{F}$. If the tuple $\langle m_w, r, R, P_i, Q_i\rangle$ already exists in L_2 then returns the corresponding $\mathtt{H}_2^i$ to $\mathcal{F}$.

H_3 *Oracle:*(m, ID_i, α, P_i). Let L_3 be the list associated with this oracle. If (m, ID_i, α, P_i) was queried previously, $\mathcal{C}$ returns $\mathtt{H}_3^i$ retrieved from the tuple $\langle m, ID_i, \alpha, \perp, \mathtt{H}_3^i\rangle$ which is already stored in the list L_3. If the tuple does not exist, $\mathcal{C}$ performs the following:

- If $ID_i \neq ID_\gamma$ then, chooses $\mathtt{H}_3^i \in_R \mathbb{G}_1$, adds the tuple $\langle m, ID_i, \alpha, P_i, \perp, \mathtt{H}_3^i\rangle$ to the list L_3 and returns $\mathtt{H}_3^i$ to $\mathcal{F}$.
- If $ID_i = ID_\gamma$ then, chooses $z \in_R \mathbb{Z}_q^*$, computes $\mathtt{H}_3^i$=zbP adds the tuple $\langle m, ID_i, \alpha, P_i, z, \mathtt{H}_3^i\rangle$ to the list L_3 and returns $\mathtt{H}_3^i$ to $\mathcal{F}$.

Extract Oracle(ID_i): For any given identity $ID_i \neq ID_\gamma$, $\mathcal{C}$ searches for the private key D_i in list L_1, corresponding to ID_i and returns it to $\mathcal{F}$. If $ID_i = ID_\gamma$, $\mathcal{F}$ $aborts$.

WarrantSign Oracle(m_w, ID_i, P_i)*:* $\mathcal{F}$ queries the warrant signature on a warrant m_w for a signer with identity ID_i.

- If $ID_i \neq ID_\gamma$, $\mathcal{C}$ responds as per the WarrantSign algorithm by creating new temporary keys $U_i = k_i$, $P_i = k_iP$, where $k_i \in_R \mathbb{Z}_q^*$. $\mathcal{C}$ adds the tuple $\langle ID_i, P_i, k_i\rangle$ to the list L_4.
- If $ID_i = ID_\gamma$, $\mathcal{C}$ responds as follows.
  - Chooses $k, y \in_R \mathbb{Z}_q^*$.
  - Computes $V_{war} = ykP_{pub}$ and $P_i = kaP$.
  - Computes $R = kP_{pub}$ and $\mathtt{H}_2^i = -k^{-1}Q_i + yP$. In this case $\mathcal{C}$ does not query the H_2 oracle, instead it sets the value for the hash computation and stores it in the list.
  - Returns the signature $\sigma_{war} = (m_w, V_{war}, P_i)$ to $\mathcal{F}$.

The correctness of σ_{war} follows from the validity check in the WarrantVerify algorithm : $\hat{e}(V_{war}, P) \stackrel{?}{=} \hat{e}(R, H_2(w, R, P_A, Q_i))\hat{e}(P_{pub}, Q_i)$. Here, the L.H.S $\hat{e}(V_{war}, P) = \hat{e}(ykbP, P)$

R.H.S= $\hat{e}(H,R)\hat{e}(Q_i,P_{pub})$
$= \hat{e}(-k^{-1}Q_i + yP, kbP)\hat{e}(aP,bP)$
= $\hat{e}(k(-k^{-1}Q_i + yP), bP)\hat{e}(aP,bP)$
= $\hat{e}(-aP + ykP, bP)\hat{e}(aP,bP)$
= $\hat{e}(ykP,bP)$=$\hat{e}(ykbP,P)$=L.H.S

Thus, it is clear that σ_{war} generated in this way is a valid signature by user with identity ID_i on warrant m_w.

GenTempKey Oracle(ID_i): $\mathcal{F}$ produces an identity ID_i to $\mathcal{C}$ and queries the corresponding the temporary private, public keys U_i and P_i. $\mathcal{C}$ responds to $\mathcal{F}$ as follows:

- If $ID_i \neq ID_\gamma$ then, $\mathcal{C}$ chooses $k_i \in_R \mathbb{Z}_q^*$, set $P_i = k_iP$, adds the tuple $\langle ID_i, P_i, k_i\rangle$ to list L_4 and returns P_i.
- If $ID_i = ID_\gamma$ then, $\mathcal{C}$ chooses $k_i \in_R \mathbb{Z}_q^*$, set $P_i = k_iaP$, adds $\langle ID_i, P_i, k_i\rangle$ to list L_4 and returns P_i.

ProxySign Oracle(m, ID_i, P_i): $\mathcal{F}$ queries this oracle with (m, ID_i, P_i) where ID_i is signer's identity, P_i is the signer's temporary public key and m is the message to be signed, $\mathcal{C}$ searches L_4 and retrieves the tuple $\langle ID_i, U_i, P_i\rangle$ and responds as follows:

- If the signer's identity $ID_i \neq ID_\gamma$, the challenger $\mathcal{C}$ proceeds as per the $ProxySign$ algorithm.
- If $ID_i = ID_\gamma$, $\mathcal{C}$ performs the following:
  - Chooses $z, \alpha \in_R \mathbb{Z}_q^*$.
  - Computes $V_{sp} = zP_i$
  - Computes $\mathtt{H}_3^i = zP$ and stores the tuple $\langle m, ID_i, \alpha, P_i, \perp, \mathtt{H}_3^i\rangle$ in list L_3. (Note that $\mathcal{C}$ did not query H_3 oracle, instead it sets the value for the hash computation)
  - Returns $\sigma_{sp} = (m, V_{sp},)$

Forgery Phase: At the end of the training phase, $\mathcal{F}$ produces a forgery $\sigma^* = \langle m^*, m_w^*, \sigma_{war}^*, \sigma_{sp}^*\rangle$ on identity ID^* and gives σ^* and ID^* to $\mathcal{C}$. Here m_w^* is the message warrant, m^* is the message, ID^* is the identity of the signer, $\sigma_{war}^* = \langle V_{war}^*, R^*, P_{ID^*}\rangle$ is the warrant signature and $\sigma_{sp}^* = \langle V_{sp}^*, \alpha^*\rangle$ is the proxy signature. $\mathcal{C}$ verifies the forgery and obtains the solution for the CDHP instance in either one of the following cases:

Case 1: Assume that $\mathcal{F}$ has not queried the WarrantSign oracle with $(m_w^*, P_{ID^*}, ID^*, Q^*)$ as input but queried the ProxySign oracle with (m^*, ID^*, P_{ID^*}) as input. $\mathcal{C}$ makes use of σ_{war}^* to solve the CDHP instance as follows:

- $\sigma_{war}^* = \langle V_{war}^*, R^*, P_{ID^*}\rangle$ is the warrant signature.
- $\mathcal{C}$ retrieves the tuple $\langle m, r, R, P_i, Q_i, \mathtt{H}_2^i\rangle$ from the list L_2 and checks whether $\hat{e}(V_{war}^*, P) \stackrel{?}{=} \hat{e}(aP, bP)\hat{e}(\mathtt{H}_2^i, P)$.
- If the above check holds then $\mathcal{C}$ computes $V_{war}^* - rP = abP$.

The above computation is correct because $\mathcal{C}$ has set $P_{pub} = bP$ and the public key of ID_γ as aP. Moreover, $\mathcal{C}$ has set $\mathtt{H}_2^i = rP$ corresponding to the message, which is retrievable from the list L_2. Thus, $V_{war}^* = abP + rP$ and computing $V_{war}^* - rP$ reveals abP.

Case 2: Assume that $\mathcal{F}$ has queried the WarrantSign oracle with $(m_w^*, P_{ID^*}, ID^*, Q^*)$ as input but did not query the ProxySign oracle with (m^*, ID^*, P_{ID^*}) as input. $\mathcal{C}$ makes use of σ_{sp}^* to solve the CDHP instance as follows:

- $\sigma_{sp}^* = \langle V_{sp}^*, \alpha^*\rangle$ is the proxy signature.
- $\mathcal{C}$ knows that $V_{sp}^* = k_izabP$. This is because the GenTempKey oracle has set $P_i = k_iaP$ and the H_3 oracle has set $\mathtt{H}_3^i = zbP$ for the corresponding α^*, when $ID_i = ID_\gamma$.
- $\mathcal{C}$ now retrieves the tuple $\langle ID_i, P_i, k_i\rangle$ from the list L_4 and the tuple $\langle m, ID_i, \alpha, P_i, z, \mathtt{H}_3^i\rangle$ from the list L_3.
- Checks whether $\alpha = \alpha^*$ and $\hat{e}(V_{sp}^*, P) \stackrel{?}{=} \hat{e}(P_i, \mathtt{H}_3^i)$.
- If the above check holds then, $\mathcal{C}$ computes $z^{-1}k_i^{-1}V_{sp}^* = abP$.

Analysis: Let $\mathcal{E}_1$ be the event in which $\mathcal{C}$ $aborts$ when $\mathcal{F}$ queries the private key corresponding to ID^* and $\mathcal{E}_2$ be the event in which ID_γ is not chosen as the target identity by $\mathcal{F}$ for generating the forgery. Suppose $\mathcal{F}$ has made q_{H_1} number of H_1 Oracle queries and q_E number of Extract Oracle queries, then:

$$\Pr[\mathcal{E}_1] = \frac{q_E}{q_{H_1}} \text{ and } \Pr[\mathcal{E}_2] = \frac{1}{q_{H_1} - q_E}.$$

Therefore, $\Pr[\mathcal{F}^{wins}_{EUF-IBSPS-CMA}]=[\neg\mathcal{E}_1 \wedge \mathcal{E}_2]= \left[1 - \frac{q_E}{q_{H_1}}\right] \cdot \left[\frac{1}{q_{H_1} - q_E}\right] = \frac{1}{q_{H_1}}$.

Thus the challenger $\mathcal{C}$ solves the $CDHP$ instance with almost the same probability as the forger $\mathcal{F}$ wins the EUF-IBSPS-CMA game. □

VII. Conclusion

We have introduced the notion of identity based self proxy signature scheme, wherein a signer creates temporary private key / public key pair which is controlled by a corresponding message warrant. The message warrant and the temporary public key are signed with the permanent identity based private key of the signer and the signature on the message is signed with the temporary private key. The temporary private key is revoked in appropriate time intervals. We have given a generic construction for identity based SPS, proposed the formal security model, given a concrete instantiation and proved it in the random oracle model. Several specific schemes can be constructed by choosing a specific identity based scheme for warrant signing and a non-identity based signature scheme for message signing.

For example, in section VI-A, we have used a variant of the IBS signature scheme in [15] for warrant signing and the $nonIBS$ scheme in [3] for message signing. Other possible combinations are, we may use CC [4], FH [6], SOK [15] for warrant signing and Schnorr, EC-DSA, BLS [3] signatures for message signing. The table in **Fig. 4.** summarizes the complexity figures for each of this combination. We observe that, if self proxy signatures are generated in a

Scheme	One Signature				*n* Signatures (During a Session)			
	Sign		Verify		Sign		Verify	
	PM	BP	PM	BP	PM	BP	PM	BP
CC [4]	2	-	1	2	2n	-	n	2n
FH [6]	3	1	1	2	3n	n	n	2n
SOK [14]	2	-	-	3	2n	-	-	3n

Figure 2. Complexity Figure for IBS Schemes per Signature and per Session

Scheme	One Signature			
	Sign		Verify	
	PM	BP	PM	BP
Schnorr	1	-	2	-
EC-DSA	1	-	2	-
BLS [3]	1	-	-	2

Figure 3. Complexity Figure for nonIBS Schemes per Signature

Scheme		One Signature				*n* Signatures (During a Session)			
Warrant Sign (IBS)	Proxy Sign (nIBS)	Sign		Verify		Sign		Verify	
		PM	BP	PM	BP	PM	BP	PM	BP
CC [4]	**Schnorr**	2+1	-	1+1	2+0	2+n	-	1+n	2+0
	EC-DSA	2+1	-	1+1	2+0	2+n		1+n	2+0
	BLS [3]	2+1	-	1+0	2+2	2+n		1+0	2+2n
FH [6]	**Schnorr**	3+1	1	1+1	2+0	3+n	1	1+n	2+0
	EC-DSA	3+1	1	1+1	2+0	3+n	1	1+n	2+0
	BLS [3]	3+1	1	1+0	2+2	3+n	1	1+0	2+2n
SOK [14]	**Schnorr**	2+1	-	-	3+0	2+n	-	-	3+0
	EC-DSA	2+1	-	-	3+0	2+n	-	-	3+0
	BLS [3]	2+1	-	-	3+2	2+n	-	-	3+2n

Figure 4. Complexity Figure of IBSPS per Signature and per Session

Legend - [**BP** - *Bilinear Pairing*, **PM** - *Scalar Point Multiplication*] session the total operation count turns out to be $n + c$ for some constant c for our schemes. For example, if signatures on warrant is generated using CC [4] and the proxy signature is generated by Schnorr scheme, then the total number of scalar point multiplication is $2+n$ for signing and $1+n$ for verifying (See row 1 of **Fig. 4.**). However, direct application of CC [4] scheme results in $2n$ scalar point multiplication during signing and n during verification. Thus our method has significantly reduced the computational complexity from $2n$ to $n+2$ during signing. The complexity figure for other combination of schemes in our generic scheme is given in **Fig. 4.**

REFERENCES

[1] Paulo S. L. M. Barreto, Benoît Libert, Noel McCullagh, and Jean-Jacques Quisquater. Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. In *Advances in cryptology ASIACRYPT05*, volume 3788 of *Lecture Notes in Computer Science*, pages 515–532. Springer, 2005.

[2] Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure proxy signature schemes for delegation of signing rights. Cryptology ePrint Archive, Report 2003/096, 2003. eprint.iacr.org.

[3] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[4] Jae Choon Cha and Jung Hee Cheon. An identity-based signature from gap diffie-hellman groups. In *Public Key Cryptography - PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 18–30. Springer, 2003.

[5] Oded Goldreich, Birgit Pfitzmann, and Ronald L. Rivest. Self-delegation with controlled propagation - or - what if you lose your laptop. In *Advances in Cryptology - CRYPTO 1998*, volume 1462 of *Lecture Notes in Computer Science*, pages 153–168. Springer, 1998.

[6] Florian Hess. Efficient identity based signature schemes based on pairings. In *Selected Areas in Cryptography, SAC - 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 310–324. Springer, 2003.

[7] SHIN-JIA Hwang and CHIU-CHIN Chen. New multi-proxy multi-signature schemes. *Applied mathematics and computation, Elsevier, New York*, vol. 147(no. 1):57–67, 2004.

[8] SHIN-JIA Hwang and C. Shi. A simple multi-proxy signature scheme. In *Proceedings of the* $10^t h$ *National Conference on Information Security*, pages 134–138, 2000.

[9] Seungjoo Kim, Sangjoon Park, and Dongho Won. Proxy signatures, revisited. In *Information and Communication Security (ICICS - 1997)*, volume 1334 of *Lecture Notes in Computer Science*, pages 223–232. Springer, 1997.

[10] Young-Seol Kim and Jik Hyun Chang. Self proxy signature scheme. *IJCSNS International Journal of Computer Science and Network Security*, vol. 7(no. 2):335–338, 2007.

[11] Byoungcheon Lee, Heesun Kim, and Kwangjo Kim. Strong proxy signature and its applications. In *The 2001 Symposium on Cryptography and Information Security - SCIS 2001*, 2001.

[12] Younho Lee, Heeyoul Kim, Yongsu Park, and Hyunsoo Yoon. A new proxy signature scheme providing self-delegation. In *Information Security and Cryptology - ICISC 2006*, volume 4296 of *Lecture Notes in Computer Science*, pages 328–342. Springer, 2006.

[13] Tal Malkin, Satoshi Obana, and Moti Yung. The hierarchy of key evolving signatures and a characterization of proxy signatures. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 306–322. Springer, 2004.

[14] Masahiro Mambo, Keisuke Usuda, and Eiji Okamoto. Proxy signatures for delegating signing operation. In *ACM Conference on Computer and Communications Security ACMCCS-96*, pages 48–57, 1996.

[15] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosysytems based on pairing. In *In Symposium on Cryptography and Information Security - SCIS 2000*, 2000.

[16] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO - 1984*, pages 47–53. Springer, 1984.

[17] Lijang Yi, Guoqiang Bai, and Guozhen Xiao. Proxy multi-signature scheme: A new type of proxy signature scheme. *Electronic letters*, vol. 36(Issue. 6):527–528, 2000.