Efficient Certificateless Online/Offline Signature with tight security*

S. Sharmila Deva Selvi, S. Sree Vivek†
Indian Institute of Technology Madras
Chennai, Tamil Nadu, India
{sharmila, svivek}@cse.iitm.ac.in

Vivek Krishna Pradhan
Indian Institute of Science Eduction and Research
Pune, Maharashtra, India
vivek.k.pradhan@gmail.com

C. Pandu Rangan
Indian Institute of Technology Madras
Chennai, Tamil Nadu, India
prangan@cse.iitm.ac.in

Abstract

Since public key cryptography is usually build using computationally expensive operation, it has been out of reach for resource constrained and low power devices. Today there are a large number of low power devices in use and they perform complex tasks. There is need for light weight cryptography having high security and low communication overhead. Online/Offline schemes are well suited for this purpose since they allow the use of public key cryptosystems in these low power devices. Many cryptosystems that are efficient in terms of number of computational steps may be inefficient if we consider the size of keys that must be used to achieve a acceptable level of security. Especially cryptosystems that have a loose security proof may work with large keys, this increases the communication overhead. In this paper, we show a view of the how Certificateless schemes are constructed. Then, we present a Certificateless Online/Offline Signature (CLOOS) Scheme and give a tight security reduction to the Gap Diffie-Hellman problem in the random oracle model. Even though other schemes exist that are are constructed using less number computational steps, if we take into account the size of keys our scheme will be more efficient. Thus, our scheme is light weight and has a low communication overhead.

Keywords: Certificateless Cryptography, Online/Offline Computation, Signature, Provable Security, Random Oracle Model, Tight Reduction.

1 Introduction

Modern cryptography started with protocols designed in the Public Key Infrastructure(PKI) model. Initially Public Key Infrastructure(PKI) based cryptosystems were proposed. But all PKI based schemes had the additional overhead of verifying certificates of the public keys. The certificates were issued by a trusted third party called the Certification Authority(CA). Adi Shamir[20] proposed Identity Based Cryptography(IBC) to solve the problem of certificate verification, but this brought with it the so called *Key Escrow Problem*. In IBC, the trusted third party called the Public Key Generator(PKG), had great power over the users. The PKG can decrypt all messages for any user and forge signatures of any user as he has the secret keys of all the users of the system.

Journal of Internet Services and Information Security (JISIS), volume: 1, number: 1/2, pp. 115-137

*In this paper, the authors take some parts from their previous version [18] published in JISIS vol. 2, no. 3/4, 2012 under the agreement between the Editor-in-Chief and themselves.

†Corresponding author: TCS Lab, BSB 324, Department of Computer Science and Engineering, IIT Madras, Chennai, India. 600036, Tel: +91-0091-4422575387

Al Riyami and Patterson [1] proposed Certificateless Cryptography(CLC) as a solution to the *Key Escrow Problem*. In CLC the trusted entity called the Key Generation Center(KGC) does not have full knowledge of the secret keys of the user, since once the user receives the partial secret and public keys from the KGC he extends/modifies them before using. Hence the KGC only knows a part of the secret key. This solves the *Key Escrow Problem* but the public keys are no longer publicly computable. Either they have to be sent along with every message and key validation algorithm may have to be executed by the user to know the correctness of the keys.

It is often impossible to provide resource constrained or low power devices with a high security. Even, Goldrich and Micali [6] explored the notion of Online/Offline Cryptography so as to allow the use of public key cryptography on these devices. The idea behind online/offline schemes is simple. The scheme is split into two parts the Offline and the Online part. In the Offline phase all the heavy computations are carried our on a more powerful device and many such Offline tuples are stored in secure storage on the low power device. In the online phase after the message and the recipient is known one offline tuple is used to construct the signature or encryption using small computations like hashing and simple modular arithmetic. Many schemes can operate as Online/Offline schemes, the schnorr signature scheme[17] being one of them. Many low power devices are in use today and hence Online/Offline schemes are very relevant.

Security proofs given for cryptosystems may be loose, close or tight as shown by Micali and Reyzin[14]. If a cryptosystem has a loose security proof then to attain an acceptable level of security it may have to work with keys of large size. For example cryptosystems like the Schnorr Signature Scheme that are proven secure using the forking lemma introduced by David Pointcheval and Jacques Stern[15] have a loose security reduction as shown by Goh et al. [9]. Since loose security reduction forces us to use large keys, even very efficient schemes become impractical. Hence there is a need for designing schemes that have *tight security reductions* to hard problems.

Related Work: Many certificateless signatures have been proposed using bilinear pairings [25, 23, 26, 27, 11, 22, 13, 21, 23, 24]. Out of these schemes only [24] can be naturally used in online/offline form but this scheme is insecure and a forgery can be produced easily as the randomness used in this scheme can be exposed. The only concrete certificateless signature scheme without paring are [8, 18]. These schemes can also be naturally used in online/offline form and their security is proved using forking lemma, i.e. they do not have a tight security reduction. These schemes will work with large keys to attain an acceptable level of security.

In general, Certificateless Signature(CLS) schemes can be thought to be composed of two important parts - the **key construct** used by the KGC to form valid keys and the **signature scheme** used by the user to sign messages using the full secret key. Intuitively, for the CLS scheme to be tightly reduced to the hard problem, both the parts - the key construct and the signature - must have a tight reduction to an underlying hard problem.

For designing the **key construct** there are two strategies - to use the key construct used in some identity based signature or the KGC should use a PKI based signature to sign the identity of the user. The most important identity based signature schemes are, schemes proposed by Cha-Cheon[4], Sakai[16], Barreto[2], Galindo[7] and Javier[12]. Also there are four PKI based signature schemes in existence that have a tight security reduction - the BLS signature scheme[3], schemes proposed by Goh and Jarecki[10], Mames et. al.[5] and Sharmila et. al.[19]. All of these signatures cannot be directly adapted as a key construct for CLS. They must first satisfy some conditions. Since we are looking to construct signatures having a tight security reduction the key construct must also be tightly related to the underlying hard problem. Another property that we are looking for is that the partial private key must be an element of $\mathbb{Z}_p$. This is because most signature schemes use an element of $\mathbb{Z}_p$ as the private key and so we have a

Table 1: Overview of identity based signature schemes

Identity based signature schemes by	Partial private key an element of $\mathbb{Z}_p$?	Tight security reduction?
Cha-Cheon[4]	No	Yes
Sakai[16]	No	Yes
Barreto[2]	No	Yes
Galindo[7]	Yes	No
Javier[12]	Yes	No

Table 2: Overview of PKI based signature schemes

PKI based signature schemes by	Partial private key an element of $\mathbb{Z}_p$?	Tight security reduction?
Goh[10]	**Yes**	**Yes**
Mames[5]	**Yes**	**Yes**
BLS[3]	No	Yes
Sharmila[19]	**Yes**	**Yes**

large selection of schemes that we can use to generate the final signature using the full secret key. Table 1 describes the six identity based signature schemes and their suitability for conversion to a CLS scheme. Table 2 gives an overview of the PKI based signatures and their suitability to be used as a key construct for the CLS scheme. From these tables it is clear that only three schemes[19, 10, 5] are suitable for being used as key constructs. These key constructs are shown and analysed in table 3.

The other part of a CLS scheme uses the full private key derived from the key construct along with a **signature scheme** to compute the final certificateless signature. The PKI based signature used in this part should also be tightly reduced to the underlying hard problem if the CLS scheme to be constructed is to have a tight reduction. Another property that is required is that the signature needs to be online/offline. We again choose from the three PKI based signatures having tight security reduction i.e. Goh and jareki, BLS, Mames and Sharmila. The table 4 compares the properties of these signature schemes. This table clearly shows that the suitable scheme is by Mames et al. Hence in this paper we demonstrate a CLOOS scheme constructed using the scheme by Sharmila et. al.[19](our scheme) as the key construct and the scheme by Mames et al. as the Signature Scheme.

Our Contribution: Above we discussed a view of a certificateless signature scheme as the Key Construct and the Signature Scheme, and discussed how a Certificateless Online/Offline Signature(CLOOS) scheme having a tight security reduction may be constructed. We first review our scheme[18]. We then present a CLOOS scheme having a tight security proof constructed using a suitably chosen Key Construct and Signature Scheme. We prove the security of this scheme is tightly related to the Gap Diffie-Hellman Problem. This scheme is the only CLOOS scheme having a tight security reduction. We compare our scheme with the schemes - [8, 18]. Even though our schemes seems to be computationally more expensive than the schemes in [8, 18], our scheme will be more efficient, because the schemes in [8, 18] work with large keys due to loose reductions.

Table 3: Partial Key Construction using the chosen constructs having a tight security reduction and proper partial private key

Key Construction with the chosen schemes	Goh and Jarecki[10]	Mames et. al.[5]	Sharmila et al.[19]
Setup	$sk = s \in_R \mathbb{Z}_p$ $P_{Pub} = sP$	$sk = s \in_R \mathbb{Z}_p$ $P_{Pub} = sP$	$sk = s_1, s_2 \in_R \mathbb{Z}_p$ $P_{Pub1} = s_1 P, P_{Pub2} = s_2 P$
Signature	$r \in_R \mathbb{Z}$ $H = \mathscr{H}(\mathscr{M}, r) Z = sH$ $k \in_R \mathbb{Z}_p$ $K = kP, X = kH$ $c = \mathscr{H}(P, H, P_{Pub}, Z, K, X)$ $v = k + cs$ $\sigma = \langle Z, r, v, c \rangle$	$k \in_R \mathbb{Z}_p$ $K = kP, H = \mathscr{H}(K)$ $Z = sH, X = kH$ $c = \mathscr{G}(\mathscr{M}, P, H, P_{Pub}, Z, K, X)$ $v = k + cs$ $\sigma = \langle Z, v, c \rangle$	$r \in_R \mathbb{Z}_p$ $U_2 = rP_{Pub2}$ $U_1 = r\mathscr{H}_1(\mathscr{M}, U_2)$ $h_m = \mathscr{H}_2(\mathscr{M}, U_1)$ $v = h_m s_1 + r s_2$ $\sigma = \langle U_1, v \rangle$
Signature Cost	$2\mathscr{H} + 3PM + 1ma + 1mm$	$2\mathscr{H} + 3PM + 1ma + 1mm$	$2\mathscr{H} + 2PM + 1ma + 2mm$
Verification	$H = \mathscr{H}(\mathscr{M}, r)$ $K = sP - cP_{Pub}, X = vH - cZ$ $c \stackrel{?}{=} \mathscr{H}(P, H, P_{Pub}, Z, K, X)$	Compute $K = vP - cP_{Pub}$ $H = \mathscr{H}(u)$ $X = vH - cZ$ $c \stackrel{?}{=} \mathscr{G}(\mathscr{M}, P, H, P_{Pub}, Z, K, X)$	$h_m = \mathscr{H}_2(\mathscr{M}, U_1)$ $U_2 = vP - h_m P_{pub1}$, $\widehat{e}(U_1, P_{Pub2})$ $\stackrel{?}{=} \widehat{e}(\mathscr{H}_1(\mathscr{M}, U_2), U_2)$
Verification Cost	$2\mathscr{H} + 4PM + 2PA$	$2\mathscr{H} + 4PM + 2PA$	$2\mathscr{H} + 2BP + 2PM + 1PA$
Partial private Key Construct	$Q_A = \mathscr{H}(ID_A, r_A)$ $Z = sQ_A, K_A = k_A P$ $X = k_A Q_A$ $c = \mathscr{H}(P, Q_A, P_{Pub}, Z, K_A, X)$ $d_A = k + cs$ $psk = d_A,\ ppk = \langle Z, r_A, c \rangle$	$k_A = k_A P, Q_A = \mathscr{H}(K_A)$ $Z = sQ_A, X = kQ_A$ $c = \mathscr{G}(ID_A, P, Q_A, P_{Pub}, Z, K_A, X)$ $d_A = k + cs$ $psk = d_A, ppk = \langle Z, c \rangle$	$U_2 = r_A P_{Pub2}$ $U_1 = r_A \mathscr{H}_1(ID_A, U_2)$ $q_A = \mathscr{H}_2(ID_A, U_1)$ $d_A = q_A s_1 + r_A s_2$ $psk = d_A, ppk = \langle U_1 \rangle$

Table 4: Overview of PKI based signature schemes

PKI based signature schemes by	online/offline?	Tight security reduction?
Goh[10]	No	Yes
Mames[5]	**Yes**	**Yes**
BLS[3]	No	Yes
Sharmila[19]	No	Yes

2 Preliminaries

In this section, we describe all the basic definitions required for this paper and also describe the generic certificateless online/offline signature scheme. We also describe the security model we have used to prove our schemes.

2.1 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 $\widehat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with the following properties.

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

2.2 Computational Assumptions

The security proof of a scheme against a well defined adversary is given by using the adversary as a probabilistic polynomial time algorithm, and solving a known hard problem assuming that the adversary exists. This shows that until the hard problem remains as such, the adversary cannot exist. In this section we define the hard problems that the security of our proposed schemes rely on.

2.2.1 Discrete Logarithm Problem

Definition 1. *Discrete Logarithm Problem (DLP): Given $(g, g^a) \in \mathbb{G}_1^2$ for unknown $a \in \mathbb{Z}_q^*$, the Discrete Logarithm problem in $\mathbb{G}_1$ is to compute a.*

The advantage of any probabilistic polynomial time algorithm $\mathscr{A}$ in solving the Discrete Logarithm Problem in $\mathbb{G}_1$ is defined as

$$Adv_{\mathscr{A}}^{DLP} = Pr\left[\mathscr{A}(g, g^a) = a \mid a \in \mathbb{Z}_q^*\right]$$

The *Discrete Logarithm Problem* is computationally hard, i.e. for any probabilistic polynomial time algorithm $\mathscr{A}$, the advantage $Adv_{\mathscr{A}}^{DLP}$ is negligibly small.

2.2.2 Decision Diffie-Hellman Problem (DDHP)

Definition 2. *Given $(P, aP, bP, Q) \in \mathbb{G}_1^4$ for unknown $a, b \in \mathbb{Z}_q^*$, the DDH problem in $\mathbb{G}_1$ is to check if $Q \stackrel{?}{=} abP$.*

The advantage of any probabilistic polynomial time algorithm $\mathscr{A}$ in solving the DDH problem in $\mathbb{G}_1$ is defined as

$$Adv_{\mathscr{A}}^{DDH} = |Pr\left[\mathscr{A}(P, aP, bP, Q) = 1\right]| - |Pr\left[\mathscr{A}(P, aP, bP, abP) = 1\right]| a, b \in_R \mathbb{Z}_q^*$$

The *DDH Assumption* is that, for any probabilistic polynomial time algorithm $\mathscr{A}$, the advantage $Adv_{\mathscr{A}}^{DDH}$ is negligibly small. Here $\mathbb{G}_1$ is a additive group.

2.2.3 Computation Diffie-Hellman Problem (CDHP)

Definition 3. *Given* $(g, g^a, g^b) \in \mathbb{G}_1^3$ *for unknown* $a, b \in \mathbb{Z}_q^*$*, the CDH problem in* $\mathbb{G}_1$ *is to compute* g^{ab}*.*

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

$$Adv_{\mathscr{A}}^{CDH} = Pr\left[\mathscr{A}(g, g^a, g^b) = g^{ab} \mid a, b \in_R \mathbb{Z}_q^*\right]$$

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

2.2.4 Gap Diffie-Hellman Problem (GDHP)

Definition 4. *We call* $\mathbb{G}$ *a gap Diffie-Hellman group if the DDHP can be solved in polynomial time but no probabilistic polynomial time algorithm can solve CDHP with non-negligible advantage. The CDHP in gap diffie-hellman groups is called GDHP.*

2.3 Certificateless Online/Offline Signature

Any certificateless signature scheme consists of seven algorithms namely *Setup, PartialExtract, SetSecretValue, PublicKeyGeneration, PrivateKeyGeneration, Sign* and *Verify*. A certificateless online/offline signature scheme will contain the following eight probabilistic polynomial time algorithms. Here a particular user is denoted as $\mathscr{U}_A$ and his identity as ID_A. Since there are many keys in a Certificateless system we use the following conventions: `UPK` - User Public Key, `FPK` - Full Public Key, `PPK` - Partial Public Key, `USK` - User Secret Key, `FSK` - Full Secret Key, `PSK` - Partial Secret Key.

Setup(κ): This algorithm is run by the KGC. The master private key and the public parameters are generated by executing this algorithm. Given the security parameter κ the KGC first sets the master private key (*msk*) then the public parameters (*params*). The KGC publishes *params* but keeps *msk* secret.

PartialExtract(*params*, ID_A): This algorithm is executed by the KGC. Given an identity ID_A as input, the KGC generates the `PPK` (Partial Private Key) and `PPK` (Partial Public Key) and sends them to the user.

SetSecretValue(*params*, κ): This algorithm is run by each user in the system to generate his user secret value. Let the user be $\mathscr{U}_A$ and the corresponding user secret value of this user be t_A. The value t_A is kept secret by the user.

PublicKeyGeneration(*params*, ID_A, `USK`, `PPK`$\rangle$): This algorithm is executed by the user to generate the full public key corresponding to his identity. The inputs to this algorithm are the identity ID_A, the user private key t_A and the partial public key. The output of this algorithm is the user public key. Note that this step is independent of the PrivateKeyGeneration. The user public key can be set even before knowing the partial private key. The full public key is the partial public key together with the user public key.

PrivateKeyGeneration(*params*, ID_A, `PSK`, `USK`): This algorithm is executed by the user to generate the `FPK`. The user computes his `FPK` using the `PPK` and the `UPK`. The `FPK` is kept secret by the user. Note that the KGC does not have complete knowledge about `FPK`.

OfflineSignature(*params*, `FSK`): To generate a certificateless signature, taking *params* as input the signer generates the offline component ϕ. It should be noted that the signer does not know the message during the offline computation. The offline signature is typically a collection of tuples. The signer pre-computes and stores a large number of offline signature tuples in secure local storage for use in the online phase.

OnlineSignature($params, ID_A, \mathscr{M}, \texttt{FSK}, \phi$)**:** This algorithm is run by the signer during the online phase. Given a message $\mathscr{M}$, the FPK and an offline tuple, the signer generates the certificateless signature σ. Note that for each signature computation in the online phase, a fresh offline signature tuple must be retrieved and used. If there is no look up directory for the public key values then the public key must be sent along with the message and the signature.

Verification($ID_A, \mathscr{M}, \texttt{FPK}, \sigma$)**:** This algorithm is run by the verifier. The signature verification can be done by anyone using *params*, the signer's identity ID_A, the message $\mathscr{M}$ and the signer's public key FPK. If the signature is valid output `true` else output `false`.

2.4 Security Model for Certificateless Online/Offline Signature

In the certificateless setting there are two types of adversaries denoted by, $\mathscr{A}_I$ and $\mathscr{A}_{II}$. $\mathscr{A}_I$ represents a dishonest user who can replace other users' public keys since there is no certificate bound with the public keys. $\mathscr{A}_{II}$ represents a malicious KGC who has knowledge of *msk* but is trusted not to replace public keys. Here we describe the security model for Existential Unforgeability against chosen message attack (EUF-CMA) against $\mathscr{A}_I$ and $\mathscr{A}_{II}$. This model is the strongest security model discussed by Z. Zhang et al.[13] for both the type-I and type-II adversaries.

Definition 5. *A certificateless signature scheme is existentially unforgeable against chosen message attack of type-I (EUF-CMA-I) if any type-I PPT adversary $\mathscr{A}_I$ has negligible advantage in the following game between $\mathscr{A}_I$ and a challenger algorithm $\mathscr{C}$:*

Restrictions: $\mathscr{A}_I$ may request hash queries, PartialExtract(ID_A), PublicKeyGeneration($ID_A, \texttt{USK}, \texttt{PPK}$), PrivateKeyGeneration($ID_A, \texttt{PSK}, \texttt{USK}$), PublicKeyReplace($ID_A, \texttt{New-FPK}$) and Signature oracle queries. $\mathscr{A}_I$ must however stick to the following exception:

1. For any identity, $\mathscr{A}_I$ cannot request the partial private key after replacing the public key.

Setup: $\mathscr{C}$ starts the game by setting the public parameters and gives *params* to $\mathscr{A}_I$. The *msk* is kept secret.

Training Phase: $\mathscr{A}_I$ is now allowed query the oracles as defined in the model. The queries are subject to the restrictions stated above. The following oracle queries are allowed:

- PartialExtract(ID_i) queries can be made by the $\mathscr{A}_I$ for any identity except ID_{ch}.
- PrivateKeyGeneration(ID_i) queries can be made by $\mathscr{A}_I$ for all identities except ID_{ch}. Note that $\mathscr{A}_I$ need not send t_i or k_i for this query, if they are not yet set. $\mathscr{C}$ should set them before answering the query.
- PublicKeyGeneration(ID_i) queries can be made by $\mathscr{A}_I$ for all identities. Note that $\mathscr{A}_I$ is not required to send t_i to $\mathscr{C}$ for this query. If t_i is not set it should be set by $\mathscr{C}$.
- PublicKeyReplace(ID_i, New-FPK) $\mathscr{A}_I$ sends a new public key to replace the old public key. When $\mathscr{C}$ receives this query, $\mathscr{C}$ replaces the old public key for ID_i with the new one, only if the new public key is valid. This means that all signing and verifications done after this will use the new public key.
- Signature(ID_i, $\mathscr{M}$) query can be made by $\mathscr{A}_I$ for all identities. $\mathscr{A}_I$ is not required to send the full private key to $\mathscr{C}$ for the query. Here the Signature oracle is a combination of the online and offline signatures. We do not separately give the offline and the online signatures as oracles since the offline phase is assumed to be securely stored on the local storage of the device and hence it is not revealed to the adversary.

Note that in our security proof, we provide a strong Signature oracle. A strong oracle for signature means that even if the public key has been replaced for the particular identity during the training phase, the Signature oracle outputs valid signatures.

Forgery: Finally, after taking sufficient training, $\mathscr{A}_I$ outputs a forgery $\langle \mathscr{M}, \sigma^*, ID_{ch}, \texttt{FPK} \rangle$. $\mathscr{A}_I$ wins if

- $verify(\mathscr{M}, \sigma^*, ID_{ch}, \texttt{FPK}) = \texttt{True}$
- The signature σ^* was not the output of a Signature oracle query during the training phase.
- The partial private key of ID_{ch} is not known to $\mathscr{A}_I$.

The advantage of $\mathscr{A}_I$ is defined as the probability that $\mathscr{A}_I$ wins the game.

Definition 6. *A certificateless signature scheme is existentially unforgeable against chosen message attack of type-II (EUF-CMA-II) if any type-II adversary $\mathscr{A}_{II}$ has negligible advantage in the following game between $\mathscr{A}_{II}$ and a challenger algorithm $\mathscr{C}$:*

Restrictions: $\mathscr{A}_{II}$ may request hash queries, PublicKeyGeneration($ID_A, \texttt{USK}, \texttt{PPK}$), PrivateKeyGeneration($ID_A, \texttt{PSK}, \texttt{USK}$) and Signature oracle queries. Let ID_{ch} be the identity for which $\mathscr{A}_{II}$ submits the final forgery.

Setup: $\mathscr{C}$ sets up the system by generating the public parameters *params* and gives it to $\mathscr{A}_{II}$. The *msk* is also sent to $\mathscr{A}_{II}$.

Training Phase: $\mathscr{A}_{II}$ is now allowed to make use of a number of oracles provided by $\mathscr{C}$. With respect to the restrictions stated above, $\mathscr{C}$ provides the following oracles.

- Note that the PartialExtract(ID_i) oracle is not required to be provided to $\mathscr{A}_{II}$ since $\mathscr{A}_{II}$ already has the master private key and can easily compute the partial private key and partial public key.
- PrivateKeyGeneration(ID_i) queries can be made by $\mathscr{A}_{II}$ for all identities except ID_{ch}. Note that $\mathscr{A}_{II}$ need not send t_i or d_i for this query, if they are not yet set $\mathscr{C}$ should set them before answering the query.
- PublicKeyGeneration(ID_i) queries can be made by $\mathscr{A}_{II}$ for any identity. Note that $\mathscr{A}_{II}$ is not required to send t_i to $\mathscr{C}$ for this query. If t_i is not set then $\mathscr{C}$ sets it first and then sends the FPK to $\mathscr{A}_{II}$.
- Signature($ID, \mathscr{M}$) query can be made by $\mathscr{A}_{II}$ for all identities. $\mathscr{A}_{II}$ is not required to send the full private key to $\mathscr{C}$ for the query. Here the Signature oracle is a combination of the online and offline signatures. We do not separately give the offline and the online signatures as oracles since the offline phase is assumed to be securely stored on the local storage of the device and hence it is not revealed to the adversary.

Forgery: Finally, $\mathscr{A}_{II}$ outputs a forgery $\langle \mathscr{M}, \sigma^*, ID_{ch}, \texttt{FPK} \rangle$ and $\mathscr{A}_{II}$ wins if

1. $verify(\mathscr{M}, \sigma, ID_{ch}, \texttt{FPK}) = \texttt{True}$ and
2. The Signature oracle was not queried with $(\mathscr{M}, ID_{ch}, \texttt{FPK})$ as input during the training phase.

The advantage of $\mathscr{A}_{II}$ is defined as the probability that $\mathscr{A}_{II}$ wins the game.

Remark: Observe that while $\mathscr{A}_I$ does not get the master private key(*msk*), $\mathscr{A}_{II}$ gets *msk* from the challenger $\mathscr{C}$. Also while $\mathscr{A}_I$ may change the public key through the oracle PublicKeyReplace(ID_i, New-FPK), but $\mathscr{A}_{II}$ cannot change the public keys and hence no such oracle is provided in Definition 6.

2.5 Review of the Scheme Presented in [18]

This is paper is an expanded version of our work[18] that was presented at MIST-2012. We extend the ideas from CLOOS-MIST here to construct a new Certificateless Online/Offline Signature (CLOOS) having a tight security reduction. Note that CLOOS-MIST described below has two parts - a key construct and a signature scheme. We have incorporated the schnorr signature scheme as main component of both these parts. Note that the schnorr signature scheme does not have a tight security proof, due to this our scheme also does not have a tight security proof.

- **Setup(κ):** Given κ the security parameter as input, the KGC chooses a multiplicative group $\mathbb{G}$ with prime order p, chooses a generator of the group g. Chooses $s \in_R \mathbb{Z}_p^*$ as the master private key. The KGC then computes $h = g^s$, Chooses four hash functions with the following definition:
  - $\mathscr{H}_1 : \mathbb{Z}_p^* \times \mathbb{G} \to \mathbb{Z}_p^*$
  - $\mathscr{H}_2 : \mathbb{Z}_p^* \times \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_p^*$
  - $\mathscr{H}_3, \widehat{\mathscr{H}_3} : \{0,1\}^{|\mathscr{M}|} \times \mathbb{Z}_p^* \times \mathbb{G}^3 \to \mathbb{Z}_p^*$

  The KGC keeps msk secret and sets $Params = \langle \kappa, p, g, h, \mathscr{H}_1, \mathscr{H}_2, \mathscr{H}_3, \widehat{\mathscr{H}_3} \rangle$

  Note: In all the algorithms described below, we consider the identity ID_A corresponds to the user $\mathscr{U}_A$ and all values subscripted with A represent the value corresponding to the user $\mathscr{U}_A$.

- **PartialExtract(ID_A):** Given an identity ID_A the KGC does the following to generate the partial private key:
  - Choose $y_A \in_R \mathbb{Z}_p^*$ and compute the partial public key $p_A = g^{y_A}$.
  - Compute the partial private key $k_A = y_A + s\mathscr{H}_1(ID_A, p_A)$.

  Send p_A and k_A as the partial public key and partial private key respectively to the user $\mathscr{U}_A$. The user may check `true`: $g^{k_A} = p_A h^{\mathscr{H}_1(ID_A, p_A)}$ for the validity and correctness of the received values.

- **SetSecretValue(κ, $params$):** $\mathscr{U}_A$ performs the following to generate the user secret value corresponding to his identity:
  - Choose $t_A \in_R \mathbb{Z}_p^*$ and sets t_A as the user secret value.

- **PublicKeyGeneration(ID_A, t_A, k_A, p_A):** The user performs the following to generate the full public key.
  - Compute the user public key as $q_A = g^{t_A}$.
  - Set full public key as $\langle p_A, q_A \rangle$.

- **PrivateKeyGeneration($ID_A, t_A, k_A, \langle p_A, q_A \rangle$):** The user sets his full private key as follows:
  - Compute the value $w_A = t_A \mathscr{H}_2(ID_A, p_A, q_A)$.
  - The full private key $n_A = \langle k_A, t_A, w_A \rangle$.

- **Offline Signature($params$):** The signer performs the following to generate the offline components which are stored as tuples:
  - Choose $r \in_R \mathbb{Z}_p^*$.

  - Compute $u = g^r$.
  - The offline signature is $\phi = \langle u, r\rangle$.

  Note: It should be noted that these offline components are computed when the device is idle and does not perform any operations with respect to signing. A large set of these pair of values are stored in the local memory of the device. These values are independent of the messages.

- **Online Signature(**$ID_A, \mathscr{M}, n_A, \phi$**):**
  - Obtain a fresh offline signature tuple $\phi = \langle u, r\rangle$ note that $n_A = \langle k_A, t_A, w_A\rangle$.
  - Compute $h_3 = \mathscr{H}_3(\mathscr{M}, ID_A, u, p_A, q_A)$ and $\widehat{h_3} = \widehat{\mathscr{H}_3}(\mathscr{M}, ID_A, u, p_A, q_A)$
  - Compute $\sigma = r + k_A h_3 + w_A \widehat{h}_3$.
  - The online signature is $\langle \sigma, u\rangle$.

- **Signature Verification(**$ID_A, \mathscr{M}, \langle p_A, q_A\rangle, \langle \sigma, u\rangle$**):**
  - Compute $h_1 = \mathscr{H}_1(ID_A, p_A)$, $h_2 = \mathscr{H}_2(ID_A, p_A, q_A)$, $h_3 = \mathscr{H}_3(\mathscr{M}, ID_A, u, p_A, q_A)$ and $\widehat{h}_3 = \widehat{\mathscr{H}_3}(\mathscr{M}, ID_A, u, p_A, q_A)$.
  - Check if $g^{\sigma} \stackrel{?}{=} u(p_A h^{h_1})^{h_3}(q_A^{h_2})^{\widehat{h}_3}$. If the check returns `true` accept the signature else $\langle \sigma, u\rangle$ is invalid.

3 Our Scheme

Note: We now present a new scheme using two signature schemes that have a tight security reduction. We use the scheme given by Sharmila et al.[19] as the key construct and the scheme by Mames et. al.[5] as the signature scheme.

- **Setup(**κ**):** Given κ the security parameter the KGC chooses a group $\mathbb{G}$ of order p and a generator of this group P. s_1 and s_2 are then chosen randomly from $\mathbb{Z}_p^{\star}$. The KGC then sets the master secret key $msk = \langle s_1, s_2\rangle$, and then computes $P_1 = s_1 P$ and $P_2 = s_2 P$
  The KGC then chooses six hash functions with the following definition:

  - $\mathscr{H}_1 : \{0,1\}^{\star} \times \mathbb{G} \rightarrow \mathbb{G}$
  - $\widehat{\mathscr{H}_1} : \{0,1\}^{\star} \times \mathbb{G} \rightarrow \mathbb{G}$
  - $\mathscr{H}_2 : \{0,1\}^{\star} \times \mathbb{G} \rightarrow \mathbb{Z}_p^{\star}$
  - $\mathscr{H}_3 : \{0,1\}^{\star} \times \mathbb{G}^3 \rightarrow \mathbb{Z}_p^{\star}$
  - $\mathscr{H}_4 : \mathbb{G} \rightarrow \mathbb{G}$
  - $\mathscr{H}_5 : \mathscr{M} \times \{0,1\}^{\star} \times \mathbb{G}^7 \rightarrow \mathbb{Z}_p^{\star}$

  The KGC keeps msk secret and makes `params` public where `params` $= \langle \kappa, P, P_1, P_2, \mathscr{H}_1, \mathscr{H}_2, \mathscr{H}_3, \mathscr{H}_4, \mathscr{H}_5\rangle$

- **Partial Extract(**`params`**,** ID_A**):** Given an identity $ID = ID_A$ the KGC does the following to generate the `PPK` (partial public key) and the `PSK` (partial secret key):
  - Randomly chooses $r_A \in_R \mathbb{Z}_p^{\star}$ and then computes $Y_A = r_A P_2$.
  - Computes $H_A = \mathscr{H}_1(ID_A, Y_A)$ and then computes $X_A = r_A H_A$
  - Computes $d_A = s_1 q_A + s_2 r_A$; where $q_A = \mathscr{H}_2(ID_A, X_A)$
  - Finally outputs $\langle d_A\rangle$ as the `PSK` (partial secret key) and $\langle X_A, Y_A\rangle$ as the `PPK` (partial public key).

Any valid partial extract output value will return `true` for the following check: Compute $H_A = \mathscr{H}_1(ID_A, Y_A)$ and $q_A = \mathscr{H}_2(ID_A, X_A)$, then check if (1) $\widehat{e}(X_A, P_2) \stackrel{?}{=} \widehat{e}(H_A, Y_A)$ and (2) $d_A P \stackrel{?}{=} q_A P_1 + Y_A$.

- **Set Secret Value(** `params`**,** κ **)** The user $\mathscr{U}_A$ having identity ID_A performs the following to generate the `USK` (user secret key).
  - Randomly choose $t_A \in_R \mathbb{Z}_p^\star$ as the `USK` (user secret key).
- **Public Key Generation(** `params`**,** ID_A, `USK`, `PPK` **):** The user $\mathscr{U}_A$ runs this algorithm to generate the public key.
  - Compute $T_{A1} = t_A P$
  - Compute $\widehat{H}_A = \widehat{\mathscr{H}_1}(ID_A, T_{A1})$ and then set $T_{A2} = t_A \widehat{H}_A$
  - The `FPK` (Full Public Key) is $\langle X_A, Y_A, T_{A1}, T_{A2} \rangle$
- **Private Key Generation(** `params`**,** ID_A, `USK`, `PSK` **):** The user $\mathscr{U}_A$ runs this algorithm to generate his full secret key. This value is kept secret.
  - Compute $n_A = d_A + t_A \mathscr{H}_3(ID_A, Y_A, T_{A1}, T_{A2})$.
  - The `FSK` (full secret key) is n_A.
- **Offline Signature(** `params`**,** `FSK` **):** The user $\mathscr{U}_A$ runs this algorithm many times to obtain a large number of offline signature tuples and stores them in a secure local storage. For every signature generated in the online phase the user consumes one offline signature tuple. Note that the offline signature tuples are not reused.
  - Randomly choose $k \in_R \mathbb{Z}_p^\star$.
  - Then compute $H = \mathscr{H}_4(kP)$
  - Compute $Z_1 = n_A H$, $Z_2 = kH$ and $Z_3 = kP$
  - The offline signature is $\phi = \langle k, H, Z_1, Z_2, Z_3 \rangle$
- **Online Signature(** `params`**,** $ID_A, \mathscr{M}$, `FSK`, ϕ **):** To generate a signature the user $\mathscr{U}_A$ takes a fresh offline signature tuple ϕ and then:
  - computes $c = \mathscr{H}_5(\mathscr{M}, ID_A, X_A, Y_A, T_{A1}, T_{A2}, Z_1, Z_2, Z_3)$
  - then compute $v = k + c n_A$
  - Output the final signature as $\sigma = \langle Z_1, Z_2, v, c \rangle$
- **Signature Verification(** `params`**,** $ID_A, \mathscr{M}, \sigma$, `FPK` **):** The verifier runs this algorithm to verify if a signature that he had received is indeed a valid signature.
  - Compute $N_A = q_A P_1 + Y_A + \mathscr{H}_3(ID_A, Y_A, T_{A1}, T_{A2}) T_{A1}$
  - Compute $Z_3 = vP - cN_A$
  - Compute $H = \mathscr{H}_4(Z_3)$
  - Compute $H_A = \mathscr{H}_1(ID_A, Y_A)$ and $\widehat{H}_A = \widehat{\mathscr{H}_1}(ID_A, T_{A1})$
  - Check $c \stackrel{?}{=} \mathscr{H}_5(\mathscr{M}, ID_A, X_A, Y_A, T_{A1}, T_{A2}, Z_1, Z_2, Z_3)$

– Check if $vH \stackrel{?}{=} Z_2 + cZ_1$

– Accept the signature if both checks return `true`.

Public Key Verification The FPK (Full Public Key) is $\langle X_A, Y_A, T_{A1}, T_{A2} \rangle$, it can be verified as follows:

– Check if $\widehat{e}(Y_A, H_A) \stackrel{?}{=} \widehat{e}(P_2, X_A)$

– Check if $\widehat{e}(T_{A1}, \widehat{H}_A) \stackrel{?}{=} \widehat{e}(P, T_{A2})$ Only if both these checks return `true` we accept the FPK. This check needs to be performed only once for each user, until the FPK remains the same.

Lemma 1. *The above signature verification algorithm returns* `true` *for valid signatures.*

Proof. Since, the value of n_A is computed as $n_A = d_A + t_A \mathscr{H}_3(ID_A, Y_A, T_{A1}, T_{A2})$, this implies that the value of $n_A P$ is $d_A P + \mathscr{H}_3(ID_A, Y_A, T_{A1}, T_{A2})(t_A P)$. The value of $d_A P$ can be expanded as $q_A P_1 + Y_A$. Hence, $N_A = q_A P_1 + Y_A + \mathscr{H}_3(ID_A, Y_A, T_{A1}, T_{A2}) T_{A1}$ is the first step of the verification algorithm.
Now since $v = k + cn_A$, we have $Z_3 = kP = vP - cN_A$.
All other values inside the hash function used to compute c are computed in the usual way hence any valid signature must return `true` for $c \stackrel{?}{=} \mathscr{H}_5(\mathscr{M}, ID_A, X_A, Y_A, T_{A1}, T_{A2}, Z_1, Z_2, Z_3)$
The check $vH \stackrel{?}{=} Z_2 + cZ_1$ returns `true` since, $LHS = vH = kH + c(n_A H) = z_2 + cZ_1 = RHS$
This shows that a valid signature will return `true` for the verification algorithm. □

Lemma 2. *The above public key verification algorithm returns* `true` *for a valid FPK.*

Proof. Since for any valid FPK,
the $Discretelog_{P_2}(Y_A) = Discretelog_{H_A}(X_A) = r_A$ and $Discretelog_P(T_{A1}) = Discretelog_{H_A}(T_{A2}) = t_A$
$\Rightarrow \widehat{e}(Y_A, H_A) \stackrel{?}{=} \widehat{e}(P_2, X_A)$ and $\widehat{e}(T_{A1}, \widehat{H}_A) \stackrel{?}{=} \widehat{e}(P, T_{A2})$ return `true`. □

3.1 Security Proof

3.1.1 EUF-CMA security against type-I adversary

Theorem 1. *If there exists a EUF-CMA adversary $\mathscr{A}_I$ that can forge the above signature with probability ε then there exists a challenger $\mathscr{C}$ who can solve the GDH problem with probability atleast ε' where,*

$$\varepsilon' \geq \left[\left(\frac{1}{q_{id}} \right) \left(1 - \frac{q_{PE}}{q_{id}} \right) \left(1 - \frac{q_{FSE}}{q_{id}} \right) \varepsilon \right]$$

Proof: Let $\mathscr{C}$ be given an instance of the GDH problem - $\langle P, aP, bP \rangle$. The aim of $\mathscr{C}$ is to find abP. Consider a type-I adversary $\mathscr{A}_I$ capable of breaking the security of the Certificateless Online/Offline Signature scheme. We show that $\mathscr{C}$ can use $\mathscr{A}_I$ to solve the GDH problem.

Setup: The challenger $\mathscr{C}$ must setup the system exactly as in the scheme. $\mathscr{C}$ first chooses $s_2 \in_R \mathbb{Z}_p$ and then sets $P_1 = aP$ and $P_2 = s_2 P$. Note that here the master secret key $msk = \langle a, s_2 \rangle$ where a is unknown to $\mathscr{C}$. $\mathscr{C}$ then chooses six hash functions $\mathscr{H}_i$ where $i = 1, 2...5$ along with $\widehat{\mathscr{H}_1}$ and models them as random oracles O_{H_i}. To maintain consistency of response $\mathscr{C}$ maintains lists L_i for each hash function H_i. Another list L_{id} is maintained for storing all the keys. If any value is unknown while updating the list, then those values are left blank and filled when available. The list L_{id} is of the form $\langle ID_i, Y_i, T_{i1}, T_{i2}, d_i, t_i, n_i, k_i \rangle$ it contains the FPK, PSK, USK, FSK and an extra bit $k_i \in \{0, 1\}$ which acts as a flag to display if the public key has been replaced or not. k_i is set as zero unless some oracle alters its value.

Training Phase: In this phase the adversary $\mathscr{A}_I$ makes use of all the oracles provided by $\mathscr{C}$. Without loss of generality we can assume that the public key queries made by the adversary are distinct. The system is simulated in such a way that $\mathscr{A}_I$ cannot differentiate between a real and a simulated system that is provided by $\mathscr{C}$.

Choosing the target identity: In the *Oracle* $\mathscr{O}_{H_1}(ID_i, Y_j)$ the adversary asks q_{H_1} queries of the form $(ID, Y) \in (\{0,1\}^*, \mathbb{G})$ and expects a response for $\mathscr{H}_1(ID_i, y_j)$ from the challenger. The adversary can choose to query the oracle using only one ID but using different values of Y. So the number of unique identities queried is different from q_{H_1}. Let the number of unique identities queried be q_{id}. Then in *Oracle* $\mathscr{O}_{H_1}(ID_i, Y_j)$ there are two indices i and j. Here the index i counts the number of unique identities queried in $\mathscr{H}_1$ and the index j counts the total number of $\mathscr{H}_1$ queries. So $1 \leq i \leq q_{id} \leq q_{H_1}$ and $1 \leq j \leq q_{H_1}$. To set the target identity ID_I the challenger chooses I randomly such that $1 \leq I \leq q_{id}$ and sets the I^{th} unique identity as the target identity ID_I. We also assume that the target identity was decided when it was involved in a query for the first time.

Oracle $\mathscr{O}_{H_1}(ID_i, Y_j)$ To respond to this oracle the list L_{H_1} is maintained of the form $\langle ID_i, Y_j, H_j, \widehat{x}_j \rangle$. $\mathscr{C}$ responds as follows:

- If ID_i, Y_j already exists in the list then respond with value H_j from the list.
- If $ID_i \neq ID_I$ then choose $\widehat{x}_j \in_R \mathbb{Z}_p$ then compute $H_j = \widehat{x}_j P$. Return value of H_j and add the tuple $\langle ID_i, Y_j, H_j, \widehat{x}_j \rangle$ to the list.
- If $ID_i = ID_I$ then choose $\widehat{x}_j \in_R \mathbb{Z}_p$ then compute $H_j = \widehat{x}_j(bP)$. Return value of H_j and add the tuple $\langle ID_i, Y_j, H_j, \widehat{x}_j \rangle$ to the list.

Oracle $\mathscr{O}_{\widehat{H}_1}(ID_i, T_{j1})$ To respond to this oracle the list $L_{\widehat{H}_1}$ is maintained of the form $\langle ID_i, T_{j1}, \widehat{H}_j, \widehat{x}_j \rangle$. $\mathscr{C}$ responds as follows:

- If ID_i, T_{j1} already exists in the list then respond with value $\widehat{H}_j$ from the list.
- If $ID_i \neq ID_I$ then choose $\widehat{x}_j \in_R \mathbb{Z}_p$ then compute $\widehat{H}_j = \widehat{x}_j P$. Return value of $\widehat{H}_j$ and add the tuple $\langle ID_i, T_{j1}, \widehat{H}_j, \widehat{x}_j \rangle$ to the list.
- If $ID_i = ID_I$ then choose $\widehat{x}_j \in_R \mathbb{Z}_p$ then compute $\widehat{H}_j = \widehat{x}_j(bP)$. Return value of $\widehat{H}_j$ and add the tuple $\langle ID_i, T_{j1}, \widehat{H}_j, \widehat{x}_j \rangle$ to the list.

Oracle $\mathscr{O}_{H_2}(ID_i, X_j)$ To respond to this oracle the list L_{H_2} is maintained of the form $\langle ID_i, X_j, q_i \rangle$. $\mathscr{C}$ responds as follows:

- If ID_i, X_j already exists in the list then respond with value q_j from the list.
- Else, choose $q_j \in_R \mathbb{Z}_p$. Return value of q_j and add the tuple $\langle ID_i, X_j, q_j \rangle$ to the list.

Oracle $\mathscr{O}_{H_3}(ID_i, Y_j, T_{j1}, T_{j2})$ To respond to this oracle the list L_{H_3} is maintained of the form $\langle ID_i, Y_j, T_{j1}, T_{j2}, h_j \rangle$. $\mathscr{C}$ responds as follows:

- If $ID_i, Y_j, T_{j1}, T_{j2}$ already exists in the list then respond with value h_j from the list.
- Else, choose $h_j \in_R \mathbb{Z}_p$. Return value of h_j and add the tuple $\langle ID_i, Y_j, T_{j1}, T_{j2}, h_j \rangle$ to the list.

Oracle $\mathscr{O}_{H_4}(K_j)$ To respond to this oracle the list L_{H_4} is maintained of the form $\langle K_j, H_j, y_j \rangle$. $\mathscr{C}$ responds as follows:

- If K_j already exists in the list then respond with value H_j from the list.
- Else choose $y_j \in_R \mathbb{Z}_p$ and respond as: $H_j = y_j(bP)$ Return value of H_j and add the tuple $\langle K_j, H_j, y_j \rangle$ to the list.

Oracle $\mathscr{O}_{H_5}(\mathscr{M}_j, ID_i, X_j, Y_j, T_{j1}, T_{j2}, Z_{1j}, Z_{2j}, Z_{3j})$To respond to this oracle the list L_{H_5} is maintained of the form $\langle \mathscr{M}_j, ID_i, X_j, Y_j, T_{j1}, T_{j2}, Z_{1j}, Z_{2j}, Z_{3j}, q_j \rangle$. $\mathscr{C}$ responds as follows:

- If $\langle \mathscr{M}_j, ID_i, X_j, Y_j, T_{j1}, T_{j2}, Z_{1j}, Z_{2j}, Z_{3j} \rangle$ already exists in the list then respond with value q_j from the list.
- Else, choose $q_j \in_R \mathbb{Z}_p$. Return value of q_j and add the tuple $\langle \mathscr{M}_j, ID_i, X_j, Y_j, T_{j1}, T_{j2}, Z_{1j}, Z_{2j}, Z_{3j}, q_j \rangle$ to the list.

Oracle $\mathscr{O}_{PartialExtract}$: $\mathscr{C}$ responds as follows:

- If values corresponding to ID_i already exists on the list L_{id} then return $\langle d_i \rangle$ and `PSK` and $\langle X_i, Y_i \rangle$ as `PPK` from the list.
- If $ID \neq ID_I$ then:
  - choose $d_i, q_i \in_R \mathbb{Z}_p$. Compute $Y_i = d_i P - q_i(aP)$.
  - Query *Oracle* $\mathscr{O}_{H_1}(ID_i, Y_i)$ and retrieve value of $\widehat{x}_j$ from the list.
  - Compute $X_i = \widehat{x}_j s_2^{-1} Y_i$ then set value of $\mathscr{H}_2(ID_i, X_i) = q_i$ and add these values to L_{H_2}.
  - Output $\langle d_i \rangle$ and `PSK` and $\langle X_i, Y_i \rangle$ as `PPK`.
  - Add these values to list L_{id} in the entry corresponding to ID_i without changing any of the other values.
- If $ID = ID_I$ then Abort.

Lemma 3. *The above Oracle $\mathscr{O}_{PartialExtract}(ID_i)$ outputs valid `PSK` and `PPK`.*

Proof. The `PSK` and `PPK` should return `true` for the following: Compute $H_A = \mathscr{H}_1(ID_A, Y_A)$ and $q_A = \mathscr{H}_2(ID_A, X_A)$, then check if (1) $\widehat{e}(X_A, P_2) \stackrel{?}{=} \widehat{e}(H_A, Y_A)$ and (2) $d_A P \stackrel{?}{=} q_A P_1 + Y_A$.
We set $H_i = \widehat{x}_j P$ and $q_i \in_R \mathbb{Z}_p$.
(1) $LHS = \widehat{e}(X_i, P_2) = \widehat{e}(\widehat{x}_j s_2^{-1} Y_i, s_2 P) = \widehat{e}(\widehat{x}_j r_i s_2 P, s_2^{-1} s_2 P) = \widehat{e}(\widehat{x}_j P, r_i s_2 P) = \widehat{e}(H_i, Y_i) = RHS$
(2) $d_i P \stackrel{?}{=} q_i P_2 + Y_i$ here we had set, $Y_i = d_i P - q_i(aP) \Rightarrow d_i P = q_i P_2 + Y_i$
Hence both checks returns `true`. □

Oracle $\mathscr{O}_{PublicKeyGen}$: $\mathscr{C}$ responds as follows:

- If values corresponding to ID_i already exists on the list L_{id} then return $\langle X_i, Y_i, T_{i1}, T_{i2} \rangle$ as the `FPK` from the list.
- If $ID \neq ID_I$ then:
  - Retrieve the values of X_i and Y_i from the list L_{id}. If they are not on the list then first *Oracle* $\mathscr{O}_{PartialExtract}$ is queried an the above values are retrieved.
  - Choose $t_i \in_R \mathbb{Z}_p$ then compute $T_{i1} = t_i P$. Now query the *Oracle* $\mathscr{O}_{\widehat{H}_1}(ID_i, T_{i1})$ and retrieve value of $\widehat{x}_j$ from the list.

- Compute $T_{i2} = \widehat{x}_j T_{i1}$. Now add $\langle ID_i, X_i, Y_i, T_{i1}, T_{i2}\rangle$ to the list.
- Output $\langle X_i, Y_i, T_{i1}, T_{i2}\rangle$ as the FPK.
- Add these values to list L_{id} in the entry corresponding to ID_i along with the value of t_i without changing any of the other values.

• If $ID = ID_I$.

- Choose $r_I, \widehat{x}_{I1}, t_i, \widehat{x}_{I2} \in_R \mathbb{Z}_p$.
- Compute $Y_I = r_I s_2 P$ then compute $X_I = r_I H_I$.Here $H_I = \widehat{x}_{I1} bP$. Add $\langle ID_I, Y_I, H_I, \widehat{x}_{I1}\rangle$ to list L_{H_1}.
- Compute $T_{I1} = t_I P$ and $T_{I2} = t_I \widehat{H}_I$. Here $\widehat{H}_I = \widehat{x}_{I2} bP$. Add $\langle ID_I, T_{I1}, \widehat{H}_I, \widehat{x}_{I2}\rangle$ to list $L_{\widehat{H}_1}$.
- Output $\langle X_I, Y_I, T_{I1}, T_{I2}\rangle$ as the FPK.
- Add these values to list L_{id} in the entry corresponding to ID_i along with the value of t_I without changing any of the other values.

Lemma 4. *The above Oracle $\mathcal{O}_{PublicKeyGen}(ID_i)$ outputs valid FPK.*

Proof. Any valid FPK returns true for the public key verification algorithm: (1) $\widehat{e}(Y_A, H_A) \stackrel{?}{=} \widehat{e}(P_2, X_A)$ (2) $\widehat{e}(T_{A1}, \widehat{H}_A) \stackrel{?}{=} \widehat{e}(P, T_{A2})$
(1) If $ID \neq ID_I$ then proof same as Lemma 3
If $ID = ID_I$, then $Y_I = r_I P_2, X_I = r_I H_I$. So, $LHS = RHS = \widehat{e}(P_2, H_I)^{r_I}$. Verifies as true.
(2) In both cases $ID \neq ID_I$ and $ID = ID_I$, $T_{i1} = t_i P, T_{i2} = t_i \widehat{H}_I$. So, $LHS = RHS = \widehat{e}(P, \widehat{H}_I)^{t_i}$. Verifies as true. □

Oracle $\mathcal{O}_{FullPrivateKeyGen}$: $\mathcal{C}$ responds as follows:

• If values corresponding to ID_i already exists on the list L_{id} then return $\langle n_i\rangle$ as the FSK form the list.

• If $ID \neq ID_I$ then:

- Retrieve the values of $\langle d_i, t_i\rangle$ from L_{id}. If they are absent run *Oracle* $\mathcal{O}_{PartialExtract}$ and *Oracle* $\mathcal{O}_{PublicKeyGen}$.
- Query *Oracle* $\mathcal{O}_{H_3}(ID_i, Y_i, T_{i1}, T_{i2})$ and retrieve value of h_i from the list.
- Compute $n_i = d_i + h_i t_i$. Output $\langle n_i\rangle$ as the FSK.
- Add this value to list L_{id} in the entry corresponding to ID_i without changing any of the other values.

• If $ID = ID_I$ then Abort.

Oracle $\mathcal{O}_{PublicKeyReplace}$ The adversary sends the values $\langle ID_i, X_i, Y_i, T_{i1}, T_{i2}\rangle$ to the challenger $\mathcal{C}$.$\mathcal{C}$ replaces the current public key for ID_i with the above values in the list L_{id}. In addition the value of $k_i = 1$ is set. This acts as a flag to display that the public key has been replaced. From this point forward signature verification algorithm will use these values as the public key.

Oracle $\mathcal{O}_{Signature}$ The challenger answers this query as follows:

• Find value of $q_A = \mathcal{H}_2(ID_i, X_i)$ and $h_3 = \mathcal{H}_3(ID_i, Y_i, T_{i1}, T_{i2})$ from the corresponding hash oracles .

- Compute $N_i = q_i P_1 + Y_i + h_3 T_{i1}$
- Choose $\alpha, v, c \in_R \mathbb{Z}_p$ then compute $Z_3 = vP - cN_i$
- Set $\alpha P = \mathscr{H}_4(Z_3)$ and add $\langle Z_3, \alpha P, \alpha \rangle$ to the list L_{H_4}, then compute $Z_1 = \alpha N_i$ and $Z_2 = \alpha Z_3$
- Send $\langle Z_1, Z_2, v, c, \mathscr{M} \rangle$

Note: This Signature oracle is a strong Signature oracle since even if the public key has been replaced the signature produced will be a valid one.

Lemma 5 (Strong Signature Oracle)**.** *The above Strong Signature Oracle outputs valid signatures for any valid* FPK *(Even if* FPK *has been replaced).*

Proof. Any valid signature should return `true` to (1) $c \stackrel{?}{=} \mathscr{H}_5(\mathscr{M}, ID_A, X_A, Y_A, T_{A1}, T_{A2}, Z_1, Z_2, Z_3)$ (2) $vH \stackrel{?}{=} Z_2 + cZ_1$ for valid public keys.
(1) N_i, Z_3 is computed in the same way as in the actual verification algorithm. Hence the check $c \stackrel{?}{=} \mathscr{H}_5(\mathscr{M}, ID_A, X_A, Y_A, T_{A1}, T_{A2}, Z_1, Z_2, Z_3)$ will return `true` (2) $RHS = Z_2 + cZ_1 = \alpha(Z_3) + c\alpha N_i = \alpha(vP - cN_i) + c\alpha N_i = v\alpha P = vH = LHS$ also returns `true`

□

Forgery: The adversary outputs a valid forgery: $\sigma^* = \langle Z_1^*, Z_2^*, v^*, c^* \rangle$. The challenger aborts if the forgery is not given for the target identity ID_I. All the relevant public keys are retrieved from L_{id} in the entry corresponding to ID_I
The challenger retrieves the value of y_i from L_{H_4} in the entry corresponding to $Z_3^* = v^*P - c^*N_I$, q_I from L_{H_2} corresponding to ID_I, X_I and h_3 from L_{H_3} corresponding to $ID_I, Y_I, T_{I1}, T_{I2}$. Next he retrieves the values of $\widehat{x}_{j1}$ from L_{H_1} in the entry corresponding to ID_I, Y_I and $\widehat{x}_{j2}$ from $L_{\widehat{H}_1}$ in the entry corresponding to ID_I, T_{I1}.
Finally $\mathscr{C}$ returns $\Delta = q_I^{-1}[y_i^{-1}Z_1^* - s_2\widehat{x}_{j1}^{-1}X_I - h_3\widehat{x}_{j2}^{-1}T_{I2}]$ as the solution to the hard problem.

Lemma 6. *The value of Δ computed in the above way the the solution to the GDHP Problem instance i.e. $\Delta = abP$*

Proof. Since the forgery σ^* is a valid forgery - the element Z_1^* will be of the form: $Z_1^* = n_I H = (aq_I + s_2 r_I + t_I h_3)H$ and H is of the form $y_i(bP)$. So finally:

$$y_i^{-1}Z_1^* = aq_I bP + s_2 r_I bP + t_I h_3 bP$$

The solution of our hard problem i.e. abP, is to be extracted from $y_i^{-1}Z_1^*$. The value of $s_2 r_I bP$ can be computed as $s_2\widehat{x}_{j1}^{-1}X_I = s_2\widehat{x}_{j1}^{-1}\widehat{x}_{j1}r_I bP = s_2 r_I bP$.
The value of $t_I h_3 bP$ can be computed as, $h_3\widehat{x}_{j2}^{-1}T_{I2} = h_3\widehat{x}_{j2}^{-1}\widehat{x}_{j2}t_I bP = t_I h_3 bP$
Hence, we can compute $y_i^{-1}Z_1^* - s_2\widehat{x}_{j1}^{-1}X_I - h_3\widehat{x}_{j2}^{-1}T_{I2} = q_I(abP)$
$\Rightarrow abP = \Delta = q_I^{-1}[y_i^{-1}Z_1^* - s_2\widehat{x}_{j1}^{-1}X_I - h_3\widehat{x}_{j2}^{-1}T_{I2}]$

□

Probability Analysis: $\mathscr{C}$ fails to give a perfect simulation only if the following events occur.

- E_1 : $\mathscr{A}_I$ returns the final forgery for an ID other than the chosen $ID = ID_I$.
- E_2 : $\mathscr{A}_I$ makes a partial key extraction query on ID_I.
- E_3 : $\mathscr{A}_I$ makes a full private key extraction query on ID_I.

So we have $Pr[E_1] = 1 - 1/q_{id}$, $Pr[E_2] = q_{PE}(1/q_{id})$ and $Pr[E_3] = q_{FSE}(1/q_{id})$. Hence the overall probability of not aborting during the simulation phase is: $Pr[\neg E_1 \wedge \neg E_2 \wedge \neg E_3] = \left(\frac{1}{q_{id}}\right)\left(1 - \frac{q_{PE}}{q_{id}}\right)\left(1 - \frac{q_{FSE}}{q_{id}}\right)$ and finally the advantage of the adversary is ε.
Hence the total probability

$$\varepsilon' \geq \left[\left(\frac{1}{q_{id}}\right)\left(1 - \frac{q_{PE}}{q_{id}}\right)\left(1 - \frac{q_{FSE}}{q_{id}}\right)\varepsilon\right]$$

3.1.2 EUF-CMA security against type-II adversary

Theorem 2. *If there exists a EUF-CMA adversary $\mathscr{A}_{II}$ that can forge the above signature with probability ε then there exists a challenger $\mathscr{C}$ who can solve the GDH problem with probability atleast ε' where,*

$$\varepsilon' \geq \left[(\frac{1}{q_{id}})(1 - \frac{q_{FSE}}{q_{id}})\varepsilon\right]$$

Proof: Let $\mathscr{C}$ be given an instance of the GDH problem - $\langle P, aP, bP\rangle$. The aim of $\mathscr{C}$ is to find abP. Consider a type-II adversary $\mathscr{A}_{II}$ capable of breaking the security of the Certificateless Online/Offline Signature scheme. We show that $\mathscr{C}$ can use $\mathscr{A}_{II}$ to solve the GDH problem.

Setup: The challenger $\mathscr{C}$ must setup the system exactly as in the scheme. $\mathscr{C}$ first chooses $s_1, s_2 \in_R \mathbb{Z}_p$ and then sets $P_1 = s_1P$ and $P_2 = s_2P$. Note that here the master secret key $msk = \langle s_1, s_2\rangle$. The challenger gives this value (msk) to $\mathscr{A}_{II}$, since $\mathscr{A}_{II}$ represents a malicious KGC and hence has knowledge of the master secret key. $\mathscr{C}$ then chooses six hash functions $\mathscr{H}_i$ where $i = 1, 2...5$ along with $\widehat{\mathscr{H}_1}$ and models them as random oracles O_{H_i}. To maintain consistency of response $\mathscr{C}$ maintains lists L_i for each hash function H_i. Another list L_{id} is maintained to store the public keys and private keys. The list L_{id} is of the form $\langle ID_i, Y_i, T_{i1}, T_{i2}, d_i, t_i, n_i\rangle$ it contains the `FPK`, `PSK`, `USK`,`FSK`.

Training Phase: In this phase the adversary $\mathscr{A}_{II}$ makes use of all the oracles provided by $\mathscr{C}$. Without loss of generality we can assume that the public key queries made by the adversary are distinct. The system is simulated in such a way that $\mathscr{A}_{II}$ cannot differentiate between a real and a simulated system that is provided by $\mathscr{C}$.
Choosing the target identity: In the *Oracle* $\mathscr{O}_{H_1}(ID_i, Y_j)$ the adversary asks q_{H_1} queries of the form $(ID, Y) \in (\{0,1\}^*, \mathbb{G})$ and expects a response for $\mathscr{H}_1(ID_i, y_j)$ from the challenger. The adversary can choose to query the oracle using only one ID but using different values of Y. So the number of unique identities queried is different from q_{H_1}. Let the number of unique identities queried be q_{id}. Then in *Oracle* $\mathscr{O}_{H_1}(ID_i, Y_j)$ there are two indices i and j. Here the index i counts the number of unique identities queried in $\mathscr{H}_1$and the index j counts the total number of H_1 queries. So $1 \leq i \leq q_{id} \leq q_{H_1}$ and $1 \leq j \leq q_{H_1}$. To set the target identity ID_I the challenger chooses I randomly such that $1 \leq I \leq q_{id}$ and sets the I^{th} unique identity as the target identity ID_I.

Oracle $\mathscr{O}_{H_1}(ID_i, Y_j)$ To respond to this oracle the list L_{H_1} is maintained of the form $\langle ID_i, Y_j, H_j, \widehat{x}_j\rangle$. $\mathscr{C}$ responds as follows:

- If ID_i, Y_j already exists in the list then respond with value H_j from the list.

- If $ID_i \neq ID_I$ then choose $\widehat{x}_j \in_R \mathbb{Z}_p$ then compute $H_j = \widehat{x}_j P$. Return value of H_j and add the tuple $\langle ID_i, Y_j, H_j, \widehat{x}_j \rangle$ to the list.
- If $ID_i = ID_I$ then choose $\widehat{x}_j \in_R \mathbb{Z}_p$ then compute $H_j = \widehat{x}_j(bP)$. Return value of H_j and add the tuple $\langle ID_i, Y_j, H_j, \widehat{x}_j \rangle$ to the list.

Oracle $\mathscr{O}_{\widehat{H}_1}(ID_i, T_{j1})$ To respond to this oracle the list $L_{\widehat{H}_1}$ is maintained of the form $\langle ID_i, T_{j1}, \widehat{H}_j, \widehat{x}_j \rangle$. $\mathscr{C}$ responds as follows:

- If ID_i, T_{j1} already exists in the list then respond with value $\widehat{H}_j$ from the list.
- Else choose $\widehat{x}_j \in_R \mathbb{Z}_p$ then compute $\widehat{H}_j = \widehat{x}_j P$. Return value of $\widehat{H}_j$ and add the tuple $\langle ID_i, T_{j1}, \widehat{H}_j, \widehat{x}_j \rangle$ to the list.

Oracle $\mathscr{O}_{H_2}(ID_i, X_j)$ To respond to this oracle the list L_{H_2} is maintained of the form $\langle ID_i, X_j, q_j \rangle$. $\mathscr{C}$ responds as follows:

- If ID_i, X_j already exists in the list then respond with value q_j from the list.
- Else, choose $q_j \in_R \mathbb{Z}_p$. Return value of q_j and add the tuple $\langle ID_i, X_j, q_j \rangle$ to the list.

Oracle $\mathscr{O}_{H_3}(ID_i, Y_j, T_{j1}, T_{j2})$ To respond to this oracle the list L_{H_3} is maintained of the form $\langle ID_i, Y_j, T_{j1}, T_{j2}, h_j \rangle$. $\mathscr{C}$ responds as follows:

- If $ID_i, Y_j, T_{j1}, T_{j2}$ already exists in the list then respond with value h_j from the list.
- Else, choose $h_j \in_R \mathbb{Z}_p$. Return value of h_j and add the tuple $\langle ID_i, Y_j, T_{j1}, T_{j2}, h_j \rangle$ to the list.

Oracle $\mathscr{O}_{H_4}(K_j)$ To respond to this oracle the list L_{H_4} is maintained of the form $\langle K_j, H_j, y_j \rangle$. $\mathscr{C}$ responds as follows:

- If K_j already exists in the list then respond with value H_j from the list.
- Else choose $y_j \in_R \mathbb{Z}_p$ and respond as: $H_j = y_j(bP)$ Return value of H_j and add the tuple $\langle K_j, H_j, y_j \rangle$ to the list.

Oracle $\mathscr{O}_{H_5}(\mathscr{M}_j, ID_i, X_j, Y_j, T_{j1}, T_{j2}, Z_{1j}, Z_{2j}, Z_{3j})$To respond to this oracle the list L_{H_5} is maintained of the form $\langle \mathscr{M}_j, ID_i, X_j, Y_j, T_{j1}, T_{j2}, Z_{1j}, Z_{2j}, Z_{3j}, q_j \rangle$. $\mathscr{C}$ responds as follows:

- If $\langle \mathscr{M}_j, ID_i, X_j, Y_j, T_{j1}, T_{j2}, Z_{1j}, Z_{2j}, Z_{3j} \rangle$ already exists in the list then respond with value q_j from the list.
- Else, choose $q_j \in_R \mathbb{Z}_p$. Return value of q_j and add the tuple $\langle \mathscr{M}_j, ID_i, X_j, Y_j, T_{j1}, T_{j2}, Z_{1j}, Z_{2j}, Z_{3j}, q_j \rangle$ to the list.

Oracle $\mathscr{O}_{PartialExtract}$ is not provided since the adversary knows the master secret key and can set any partial extract value that he wants. The adversary should then send the `PSK` and `PPK` to the challenger who will accept it and add it to the list L_{id} if the partial extract values sent by the adversary were valid.

Oracle $\mathscr{O}_{PublicKeyGen}$: $\mathscr{C}$ responds as follows:

- If values corresponding to ID_i already exists on the list then return $\langle X_i, Y_i, T_{i1}, T_{i2} \rangle$ as the `FPK` from the list L_{id}.

- If $ID \neq ID_I$ then:
  - If the values of X_i, Y_i are absent then request $\mathscr{A}_{II}$ to set partial extract for ID_i.
  - Run the usual public key generation algorithm and return values $\langle X_i, Y_i, T_{i1}, T_{i2} \rangle$ as the FPK then add these values to the list L_{id} along with the value of t_i without changing other values.
- If $ID = ID_I$.
  - Choose $r_I, \widehat{x}_{I1} \in_R \mathbb{Z}_p$.
  - Compute $Y_I = r_I s_2 P$ then compute $X_I = r_I \widehat{x}_{I1} P$. Here $H_I = \widehat{x}_{I1} P$. Add $\langle ID_I, Y_I, H_I, \widehat{x}_{I1} \rangle$ to list L_{H_1}.
  - Set $T_{I1} = aP$ and query *Oracle* $\mathscr{O}_{\widehat{H}_1}(ID_I, T_{I1})$ and retrieve $\widehat{x}_I$ from the list.
  - Set $T_{I2} = \widehat{x}_I(aP)$. Output $\langle X_I, Y_I, T_{I1}, T_{I2} \rangle$ as the FPK and add these values to the list L_{id}.
  - Note:- The hard problem instance is injected here in the value of the user private key. In this case we cannot update the value of t_I in L_{id} since in this case we have set $t_I = a$ which is the discrete logarithm of the hard problem instance.

Lemma 7. *The above Oracle $\mathscr{O}_{PublicKeyGen}(ID_i)$ outputs valid FPK.*

Proof. Any valid FPK returns true for the public key verification algorithm: (1) $\widehat{e}(Y_A, H_A) \stackrel{?}{=} \widehat{e}(P_2, X_A)$ (2) $\widehat{e}(T_{A1}, \widehat{H}_A) \stackrel{?}{=} \widehat{e}(P, T_{A2})$
(1) If $ID \neq ID_I$ we run the usual algorithm hence the FPK is valid
If $ID = ID_I$, then $Y_I = r_I P_2, X_I = r_I H_I$. So, $LHS = RHS = \widehat{e}(P_2, H_I)^{r_I}$. Verifies as true.
(2) In both cases $ID \neq ID_I$ and $ID = ID_I$, $T_{i1} = t_i P, T_{i2} = t_i \widehat{H}_I$. So, $LHS = RHS = \widehat{e}(P, \widehat{H}_I)^{t_i}$. Verifies as true. □

Oracle $\mathscr{O}_{FullPrivateKeyGen}$ $\mathscr{C}$ responds as follows:

- If values corresponding to ID_i already exists on the list L_{id} then return n_i from the list.
- If $ID \neq ID_I$ then:
  - Retrieve the values of $\langle d_i, t_i \rangle$ from the list L_{id}. If these values are absent then run *Oracle* $\mathscr{O}_{PartialExtract}$ and *Oracle* $\mathscr{O}_{PublicKeyGen}$.
  - Query *Oracle* $\mathscr{O}_{H_3}(ID_i, Y_i, T_{i1}, T_{i2})$ and retrieve value of h_j from the list.
  - Compute $n_i = d_i + h_j t_i$. Output n_i and add the value to L_{id}.
- If $ID = ID_I$ then Abort.

Oracle $\mathscr{O}_{Signature}$ This oracle is the same as the one given for the type-I Adversary.

Forgery: The adversary outputs a valid forgery: $\langle Z_1^*, Z_2^*, v^*, c^* \rangle$. The challenger aborts if the forgery is not given for the target identity ID_I. Then $\mathscr{C}$ gets the public key values for ID_I from L_{id}.
The challenger retrieves the value of y_i from L_{H_4} in the entry corresponding to $Z_3^* = v^* P - c^* N_I$, q_I from L_{H_2} corresponding to ID_I, X_I and h_3 from L_{H_3} corresponding to $ID_I, Y_I, T_{I1}, T_{I2}$. Next he retrieves the values of $\widehat{x}_{j1}$ from L_{H_1} in the entry corresponding to ID_I, Y_I and q_A from L_{H_2} in the entry corresponding to ID_I, X_I.
Finally $\mathscr{C}$ returns $\Delta = h_3^{-1}[y_i^{-1} Z_1^* - s_2 \widehat{x}_{j1}^{-1} X_I - s_1 q_I(bP)]$ as the solution to the hard problem.

Lemma 8. *The value of Δ computed in the above way the the solution to the GDHP Problem instance i.e. $\Delta = abP$*

Proof. Since the forgery σ^* is a valid forgery - the element Z_1^* will be of the form: $Z_1^* = n_I H = (s_1 q_I + s_2 r_I + a h_3) H$ and H is of the form $y_i(bP)$. So finally:
$y_i^{-1} Z_1^* = s_1 q_I bP + s_2 r_I bP + a h_3 bP = term - I + term - II + term - III$

Here, term-III contains the solution of our hard problem i.e. abP, hence term-II and term-I needs to be removed from $y_i^{-1} Z_1^*$. Term-II can be removed by subtracting $s_2 \widehat{x}_{j1}^{-1} X_I = s_2 \widehat{x}_{j1}^{-1} \widehat{x}_{j1} r_I bP = s_2 r_I bP =$ term-II. Term-I can be removed by subtracting $s_1 q_I (bP)$
$\Rightarrow y_i^{-1} Z_1^* - s_2 \widehat{x}_{j1}^{-1} X_I - s_1 q_I (bP) =$ term-III $= h_3(abP)$
$\Rightarrow abP = \Delta = h_3^{-1}[y_i^{-1} Z_1^* - s_2 \widehat{x}_{j1}^{-1} X_I - s_1 q_I (bP)]$ □

Probability Analysis: $\mathscr{C}$ fails to give a perfect simulation only if the following events occur.

- E_1 : $\mathscr{A}_{II}$ returns the final forgery for an ID other than the chosen $ID = ID_{ch}$.
- E_2 : $\mathscr{A}_{II}$ makes a full private key extraction query on ID_{ch}.

So we have $Pr[E_1] = 1 - 1/q_{id}$ and $Pr[E_2] = q_{FSE}(1/q_{id})$. Hence the overall probability of not aborting during the simulation phase is: $Pr[\neg E_1 \wedge \neg E_2] = (\frac{1}{q_{id}})(1 - \frac{q_{FSE}}{q_{id}})$ and the advantage of the adversary is ε.
Hence the total probability

$$\varepsilon' \geq \left[(\frac{1}{q_{id}})(1 - \frac{q_{FSE}}{q_{id}})\varepsilon \right]$$

4 Efficiency Comparison

Many certificateless schemes have been proposed but to the best of our knowledge the only scheme to be online/offline is by Ge et. al.[8] and our scheme - CLOOS-MIST[18]. These schemes do not have a tight security reduction to the underlying hard problem. We have constructed a Certificateless Online/Offline Signature schemes which have a tight security reduction to the underlying hard problem. This is the only CLOOS scheme in existence having a tight security reduction to the underlying hard problem. We compare our scheme with the scheme by Ge et al. and CLOOS-MIST[18] in the table 5. Even though our scheme is built using more number of computational steps, note that our scheme works with much smaller keys since it has a tight security reduction. This makes it more efficient than than the other schemes and also lowers the communication overhead of our scheme.

5 Conclusions

In this paper, we have shown a view of how certificateless schemes are constructed. Then, we have presented a CLOOS scheme and proved its security in the random oracle model. This scheme is the only CLOOS scheme to have a tight security reduction. We have also compared our scheme to the only other CLOOS schemes in existence. We show that even though our scheme is performs more minimal operations, than the other schemes, since our scheme has a tight security it works with much smaller keys and hence will be practically more efficient than the existing schemes. Also since the key size is lowered the communication overhead will also be lesser. Our scheme will allow low power devices to achieve high security requirements.

Table 5: Efficiency Comparison

Scheme	Signature Cost	Verification Cost	Remarks
Our Scheme	$1\mathscr{H} + 1FD\mathscr{H} + 3PM + 1mm + 1ma$	$3\mathscr{H} + 3FD\mathscr{H} + 6PM + 4PA + 4BP$	Scheme with tight reduction (pairings)
CLOOS-MIST[18]	$2\mathscr{H} + 1GE + 2mm + 2ma$	$4\mathscr{H} + 3GE + 3GM + 1mm$	Most efficient Scheme. Additionally Online/Offline
Ge et. al.[8]	$1\mathscr{H} + 2GE + 2mm + 2ma$	$3\mathscr{H} + 7GE + 4GM$	Only online/offline Scheme without pairings

$\mathscr{H}$ - Hash Computation, $FD\mathscr{H}$ - Full Domain Hash Computation, *PM* - Point Multiplications, *PA* - Point Additions, *mm* - Modular multiplication, *ma* - Modular Addition, *BP* - Bilinear Pairing

References

[1] S. S. Al-Riyami and K. G. Paterson. Certificateless public key cryptography. In *Proc. of the 9th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'03), Taipei, Taiwan, LNCS*, volume 2894, pages 452–473. Springer-Verlag, November-December 2003.

[2] P. S. L. M. Barreto, B. Libert, N. McCullagh, and J.-J. Quisquater. Efficient and provably-secure Identity-based signatures and signcryption from bilinear maps. In *Proc. of the 11th international conference on Theory and Application of Cryptology and Information Security (ASIACRYPT'05), Chennai, India, LNCS*, pages 515–532. Springer-Verlag, December 2005.

[3] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[4] J. C. Cha and J. H. Cheon. An Identity-based signature from gap Diffie-Hellman groups. In *Proc. of the 6th International Workshop on Theory and Practice in Public Key Cryptography (PKC'03), Miami, Florida, USA, LNCS*, volume 2567, pages 18–30. Springer-Verlag, Janurary 2003.

[5] B. Chevallier-Mames. An efficient CDH-based signature scheme with a tight security reduction. In *Proc. of the 25th annual international conference on Advances in Cryptology (CRYPTO'05), Santa Barbara, California, USA, LNCS*, volume 3621, pages 511–526. Springer-Verlag, 2005.

[6] S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, September 1996.

[7] D. Galindo and F. D. Garcia. A Schnorr-like lightweight Identity-based signature scheme. In *Proc. of the 2nd African International Conference on Cryptology (AFRICACRYPT'09), Gammarth, Tunisia, LNCS*, volume 5580, pages 135–148. Springer-Verlag, 2009.

[8] A. Ge, S. Chen, and X. Huang. A concrete certificateless signature scheme without pairings. In *Proc. of 2009 International Conference on Multimedia Information Networking and Security (MINES'09), Wuhan, China*, pages 374–377. IEEE, November 2009.

[9] E.-J. Goh and S. Jarecki. A signature scheme as secure as the Diffie-Hellman problem. In *Proc. of the 22nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'03), Warsaw, Poland, LNCS*, volume 2656, pages 401–415. Springer-Verlag, May 2003.

[10] E.-J. Goh, S. Jarecki, J. Katz, and N. Wang. Efficient signature schemes with tight reductions to the Diffie-Hellman problems. *Journal of Cryptology*, 20(4):493–514, 2007.

[11] L. Guo, L. Hu, and Y. Li. A practical certificateless signature scheme. In *Proc. of the 1st International Symposium on Data, Privacy, and E-Commerce (ISDPE'07), Chengdu , China*, pages 248–253. IEEE, November 2007.

[12] J. Herranz. Deterministic Identity-based signatures for partial aggregation. *The Computer Journal*, 49(3):322–330, May 2006.

[13] B. C. Hu, D. S. Wong, Z. Zhang, and X. Deng. Certificateless signature: a new security model and an improved generic construction. *Designs, Codes and Cryptography*, 42(2):109–126, February 2007.

[14] S. Micali and L. Reyzin. Improving the exact security of digital signature schemes. *Journal of Cryptology*, 15(1):1–18, 2002.

[15] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Proc. of the 15th annual international conference on Theory and application of cryptographic techniques (EUROCRYPT'96), Saragossa, Spain, LNCS*, volume 1070, pages 387–398. Springer-Verlag, May 1996.

[16] R. Sakai and M. Kasahara. ID based cryptosystems with pairing on elliptic curve. Cryptology ePrint Archive, Report 2003/054, 2003.

[17] C.-P. Schnorr. Efficient identification and signatures for smart cards. In *Proc. of the 8th Annual International Workshop on the Theory and Application of of Cryptographic Techniques (EUROCRYPT'89), Houthalen, Belgium, LNCS*, volume 435, pages 239–252. Springer-Verlag, 1989.

[18] S. S. D. Selvi, S. S. Vivek, V. K. Pradhan, and C. P. Rangan. Efficient certificateless online/offline signature. *Journal of Internet Services and Information Security (JISIS)*, 2(3/4):77–92, 11 2012.

[19] S. S. D. Selvi, S. S. Vivek, and C. P. Rangan. Identity Based Deterministic Signature Scheme Without Forking-Lemma. Cryptology ePrint Archive, Report 2011/217, 2011.

[20] A. Shamir. Identity-based cryptosystems and signature schemes. In *Proc. of the 4th Annual International Cryptology Conference (CRYPTO'84), Santa Barbara, California, LNCS*, volume 196, pages 47–53. Springer-Verlag, August 1984.

[21] R. Tso, X. Yi, and X. Huang. Efficient and short certificateless signature. In *Proc. of the 7th International Conference on Cryptology and Network Security (CANS'08), Hong Kong, China, LNCS*, volume 5339, pages 64–79. Springer-Verlag, December 2008.

[22] C. Wang and H. Huang. An efficient certificateless signature from pairings. In *Proc. of the 1st International Symposium on Data, Privacy, and E-Commerce (ISDPE'07), Chengdu , China*, pages 236–238. IEEE, November 2007.

[23] C. Wang, D. Long, and Y. Tang. An efficient certificateless signature from pairings. *International Journal of Network Security*, 8(1):96–100, January 2009.

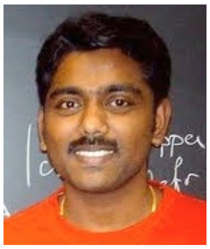
[24] Z. Xu, X. Liu, G. Zhang, and W. He. A certificateless signature scheme for mobile wireless cyber-physical systems. In *Proc. of the 28th IEEE International Conference on Distributed Computing Systems (ICDCS'08), Beijing, China*, pages 489–494. IEEE, June 2008.

[25] Z. Xu, X. Liu, G. Zhang, W. He, G. Dai, and W. Shu. A certificateless signature scheme for mobile wireless cyber-physical systems. In *Proc. of the 28th IEEE International Conference on Distributed Computing Systems Workshops (ICDCS'08 Workshops), Beijing, China*, pages 489–494. IEEE, June 2008.

[26] G. Zhang and S. Wang. A certificateless signature and group signature schemes against malicious PKG. In *Proc. of the 22nd International Conference on Advanced Information Networking and Applications (AINA'08), GinoWan, Okinawa, Japan*, pages 334–341. IEEE, March 2008.

[27] L. Zhang and F. Zhang. A new provably secure certificateless signature scheme. In *Proc. of the 2008 IEEE International Conference on Communications (ICC'08), Beijing, China*, pages 1685–1689. IEEE, May 2008.

S.Sharmila Deva Selvi is a PhD scholar in Indian Institute of Technology - Madras, Chennai, India. She is working under the guidance of Prof C.Pandu Rangan. Her areas of interest are Provable Security of Public Key Cryptosystem, Cryptanalysis of Identity Based and Certificateless cryptosystem and her works are mostly on the provable security of various flavors of signcryption schemes.



S.Sree Vivek is a PhD scholar in Indian Institute of Technology - Madras, Chennai, India. He is working under the guidance of Prof C.Pandu Rangan. His areas of interest are design and analysis of Identity Based Cryptosystem, Cryptanalysis of cryptosystem, Key Agreement and works on signcryption schemes. His thesis research topic is studies on some encryption, signature and signcryption schemes.

Vivek Krishna Pradhan is a graduate student from Indian Institute of Science Education and Research (IISER), Pune. He did his internship at the Theoretical Computer Science Lab in IIT Madras.His research interests are in the fields of Cryptography, Graph Algorithms and Computational Biology.

C.Pandu Rangan is a Professor in the department of computer science and engineering of Indian Institute of Technology - Madras, Chennai, India. He heads the Theoretical Computer Science Lab in IIT Madras. His areas of interest are in theoretical computer science mainly focusing on Cryptography, Algorithms and Data Structures, Game Theory, Graph Theory and Distributed Computing.