Efficient Certificateless Online/Offline Signature

S. Sharmila Deva Selvi
Indian Institute of Technology Madras,
Chennai, Tamil Nadu, India
sharmila@cse.iitm.ac.in

S. Sree Vivek*
Indian Institute of Technology Madras,
Chennai, Tamil Nadu, India
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

Public key cryptography usually is computationally more expensive than symmetric key systems. Due to this low power or resource constrained devices cannot make use of public key cryptosystems easily. There is a need for high security in these devices since many of these devices perform complex tasks which includes interaction with third party cloud infrastructures. These cloud infrastructures are not trusted entities. Hence there is need for light weight public key cryptography which are secure against these cloud administrators. The trusted entity in certificateless schemes cannot compromise the security of the users. Online/offline have two parts, first the computationally heavy part(offline) of the cryptosystem and then the main "online" algorithm for use on resource constrained devices. The heavy computations are done in the offline phase on a more powerful device. Hence, Certificateless online/offline schemes are perfect for low power devices interacting with clouds. In this paper, we present a certificateless online/offline signature scheme. This scheme is the most efficient certificateless signature scheme in existence and also has the added advantage of being online/offline. The scheme is proven secure in the random oracle model.

Keywords: certificateless cryptography, online/offline computation, signature, provable security, random oracle model.

1 Introduction

Initially cryptosystems were constructed based on the Public Key Infrastructure(PKI). These PKI based schemes have an entity called the Certification Authority(CA) which provides certificates for the public keys. The users of the system had to verify this certificate before accepting the public keys. This caused an additional overhead of certificate verification. This problem was solved by Adi Shamir who introduced Identity Based Cryptography(IBC)[1]. In IBC a publicly known parameter like email address, social security number or unique identification number was used as the public key. A trusted party called the Private Key Generator(PKG) generates the private keys of all users using a master secret key. Since the PKG knows all the private keys of the system, he has full power over the system and can decrypt and sign messages of any user. This is known as the *Key Escrow Problem*.

To solve the *Key Escrow Problem* Al Riyami and Patterson proposed Certificateless Cryptography (CLC)[2]. In CLC, the trusted entity called the Key Generation Center(KGC) only produces a partial private key. The user generates the full private key using the partial private key. Thus the KGC does not

Journal of Internet Services and Information Security (JISIS), volume: 2, number: 3/4, pp. 77-92

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

have full knowledge of the users' private keys. Hence CLC is ideal for cloud applications where the third party cloud administrators are not trusted entities. The disadvantage is that the public keys are no longer publicly computable and hence it is sent along with every message or is maintained in a public lookup directory.

Even, Goldreich and Micali first talked about the notion of online/offline schemes[3]. There are many low power or resource constrained devices in use, like PDA's, smartcards or nodes in a wireless sensor network etc. These devices require the same level of security as in other devices but they can handle only the most efficient schemes. Online/offline schemes help complex cryptosystems to be implemented on resource constrained devices since these schemes are divided into two parts offline phase and the online phase. Durning the offline phase, all the heavy computations are carried out on a more powerful device. Many such offline computaion tuples are computed and stored in the resource constrained device. Following this during the online phase after the message to be signed or encrypted is known the resource constrained device computes the signature or encryption using light computations alone. Many schemes have the property of being online/offline, for example, the well known Schnorr signature scheme[4].

In recent years, there has been a huge increase in the use of resource constrained devices like PDAs, smartcards or nodes in a wireless sensor network. These devices heavily interact with clouds. These cloud services are most often provided by untrusted third party providers. These cloud administrators should not have unrestricted access to the users' data. Certificateless schemes are best used in these scenarioes since the KGC in CLC is not a trusted entity. But if Certificateless schemes having heavy computations are provided the resource constrained devices will be unable to use these schemes, hence there is a need for light-weight certificateless schemes for use in resource constrained devices on cloud platforms. So, there is a need for Certificateless Online/Offline Schemes.

Many certificateless signatures have been proposed using bilinear pairings [5, 6, 7, 8, 9, 10, 11, 12, 6, 13]. The only concrete certificateless scheme without parings is by Ge et. al.[14]. This scheme can also be used in online/offline form and its security is proved using forking lemma.

Our Contribution: In this paper we present a new certificateless signature scheme. This scheme is more efficient than Ge et. al.'s scheme[14] and hence is the most efficient certificateless signature scheme in existence. We prove the security of our scheme by reducing the hardness of forging our scheme to be equivalent to forging Galindo and Garcia's Signature(GGS)[15] scheme or the Schnorr signature scheme[4]. We also provide a strong signature oracle in the proofs, which strengthens the security proof.

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 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. Our scheme is shown to be secure as long on GGS scheme and schnorr signature schemes are secure. Both of these schemes are secure assuming the hardness of Discrete Logarithm Problem.

2.1.1 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 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.

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

SetSecretValue(κ, *params*): 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($ID_A, t_A, \langle partial\,keys\rangle$): 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($ID_A, t_A, \langle partial\,keys\rangle, \langle user\,public\,key\rangle$,): This algorithm is executed by the user to generate the full private key. The user computes his full private key n_A using the partial keys k_A, p_A and the user keys t_A, q_A. The full private key n_A is kept secret by the user. Note that the KGC does not have complete knowledge about n_A.

OfflineSignature(*params*): 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($ID_A, \mathscr{M}, n_A, \phi$): This algorithm is run by the signer during the online phase. Given a message $\mathscr{M}$, the full private key n_A 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($ID_A, \mathscr{M}, \langle full\,public\,key\rangle, \sigma$): 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 $\langle full\,public\,key\rangle$. If the signature is valid output `true` else output `false`.

2.3 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.[11] for both the type-I and type-II adversaries.

Definition 2. *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($ID_A, t_A,$ *partial public key*), PrivateKeyGeneration(ID_A, t_A, d_A), PublicKeyReplace($ID_A, \langle$*public key*$\rangle, \langle$*new public key*$\rangle$) 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($ID_i, \langle$*new public key*$\rangle$) $\mathscr{A}_I$ 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 $\langle \mathscr{M}, \sigma^*, ID_{ch}, \langle$*public key*$\rangle\rangle$. $\mathscr{A}_I$ wins if

1. $verify(\mathscr{M}, \sigma^*, ID_{ch}, \langle$*public key*$\rangle) =$ `True`

2. The signature σ^* was not the output of a Signature oracle query during the training phase.

3. 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 3. *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($ID_A, t_A, d_A, \langle partial\ public\ key \rangle$), PrivateKeyGeneration(ID_A, t_A, d_A) 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 $\langle full\ public\ key \rangle$ 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 $\langle \mathscr{M}, \sigma^*, ID_{ch}, \langle public\ key \rangle \rangle$ and $\mathscr{A}_{II}$ wins if

1. $verify(\mathscr{M}, \sigma, ID_{ch}, \langle public\ key \rangle) =$ `True` and

2. The Signature oracle was not queried with $(\mathscr{M}, ID_{ch}, \langle public\ key \rangle)$ 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 throgh the oracle PublicKeyReplace($ID_i, \langle new\ public\ key \rangle$), but $\mathscr{A}_{II}$ cannot change the public keys and hence no such oracle is provided in Definition 3.

3 Our Scheme

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

  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.

Correctness of the Signature algorithm: If $\langle \sigma, u \rangle$ is correctly generated, then $LHS = g^{\sigma} = g^{r+(y_A+sh_1)h_3+t_A h_2 \widehat{h}_3} = u(p_A h^{h_1})^{h_3}(q_A^{h_2})^{\widehat{h}_3} = RHS$ of the verification algorithm.

3.1 Security Proof

3.1.1 Unforgeability against type-I adversary

Theorem 1. *If there is a EUF-CMA type-I adversary $\mathscr{A}_I$ of the above scheme that succeeds with probability ε then there is a challenger $\mathscr{C}$ running in polynomial time that can produce a forgery for the Galindo and Garcia's signature scheme[15] with probability atleast ε.*

Proof: Let $\mathscr{C}$ denote the challenger for our signature scheme and $\mathscr{A}_I$ denote the adversary for our scheme. $\mathscr{C}$ will simultaneously act as an adversary for a Galindo and Garcia Signature(GGS) scheme[15] where the challenger for the GGS scheme is denoted by $\mathscr{B}_I$ as shown in the figure 3.1.1. The aim of $\mathscr{C}$ is to cleverly embed the instance of GGS scheme into the instance of our signature scheme. Finally when the adversary of our scheme $\mathscr{A}_I$ submits a forgery for our scheme, $\mathscr{C}$ will inturn use this forgery to construct a forgery for the GGS scheme and submit to $\mathscr{B}_I$. This is a contradiction since the GGS scheme is known to be secure and so by contradiction our scheme is also secure.

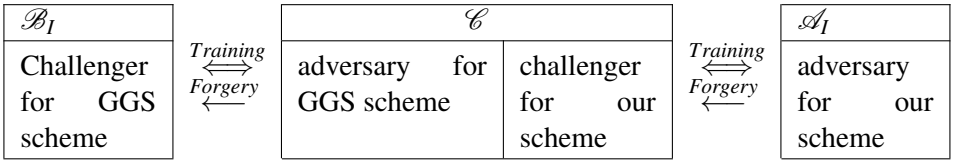


Figure 1: The relationship between $\mathscr{C}$, $\mathscr{A}_I$, $\mathscr{B}_I$

First, $\mathscr{B}_I$ sets up the *params* for the GGS scheme and passes the *params* information to $\mathscr{C}$. Now $\mathscr{C}$ will construct an instance of our signature scheme based on the *params* received and passes on some public information to $\mathscr{A}_I$. The oracle queries from $\mathscr{A}_I$ to $\mathscr{C}$ are answered by $\mathscr{C}$ as follows. For every query from $\mathscr{A}_I$ a suitably defined query will be formulated by $\mathscr{C}$ and $\mathscr{C}$ will pose this newly formed query to $\mathscr{B}_I$. Since $\mathscr{B}_I$ knows all the parrameters, including the secret keys of the GGS scheme, $\mathscr{B}_I$ will give exact answers by running the actual algorithm i.e. $\mathscr{B}_I$ will never abort. The response from $\mathscr{B}_I$ to $\mathscr{C}$ will be suitably modified to construct a response to $\mathscr{A}_I$ by $\mathscr{C}$. This way, $\mathscr{C}$ will provide training to

$\mathscr{A}_I$. Finally, when $\mathscr{A}_I$ produces a forgery to our system, $\mathscr{C}$ will use it to produce a forgery for the GGS scheme. The details are given below. First, we give a small review of the game simulated by $\mathscr{B}_I$.

Game simulated by the Adversary $\mathscr{B}_I$ (Galindo and Garcia's Signature Scheme)
First let us change the notations from the original paper so that they become similar to the notations used in this paper. $\mathscr{B}_I$ sets up the signature scheme by setting up two Hash functions $\mathscr{H}_1$ and $\mathscr{H}_2$, and chooses a master private key $s \in_R \mathbb{Z}_p^*$. Then he computes $h = g^s$ and sends the values $\langle \mathbb{G}, \mathscr{H}_1, \mathscr{H}_2, g, h \rangle$ to $\mathscr{C}$. He also provides the oracles *Oracle* $O_{H_1}^{\mathscr{B}_I}(ID_i, p_j)$, *Oracle* $O_{H_2}^{\mathscr{B}_I}(\mathscr{M}, ID_i, u)$, *Oracle* $O_{KeyExtract}^{\mathscr{B}_I}(ID_i)$, *Oracle* $O_{Signature}^{\mathscr{B}_I}(ID_i, \mathscr{M})$. The *Oracle* $O_{KeyExtract}^{\mathscr{B}_I}(ID_i)$ simulates the key extract algorithm which is - choose $y_A \in_R \mathbb{Z}_p^*$ and compute $p_A = g^{y_A}$, $k_A = y_A + s\mathscr{H}_1(ID_A, p_A)$ finally return $\langle k_A, p_A \rangle$. The *Oracle* $O_{Signature}^{\mathscr{B}_I}(ID_i, \mathscr{M})$ simulates the signature algorithm which is - choose $r \in_R \mathbb{Z}_p^*$ and compute $u = g^r$, $\sigma = r + k_A\mathscr{H}_2(\mathscr{M}, ID_A, u)$ finally return $\langle \mathscr{M}, \sigma, u, p_A \rangle$. The verification algorithm of the signature does the following: Given $\langle \mathscr{M}, \sigma, u, ID_A, p_A \rangle$ and checks if $g^\sigma \stackrel{?}{=} u(p_A h^{\mathscr{H}_1(ID_A, p_A)})^{\mathscr{H}_2(\mathscr{M}, ID_A, u)}$. Since $\mathscr{B}_I$ knows the master secret key s, $\mathscr{B}_I$ can perfectly answer all the queries by executing exact algorithms.

Setup: $\mathscr{C}$ has to setup the parameters for the game exactly as in our scheme. To Simulate our scheme for the advarsary $\mathscr{A}_I$, $\mathscr{C}$ makes use of the challenger $\mathscr{B}_I$ in the following way: Once $\mathscr{B}_I$ finishes the setup of the signature scheme he sends $\langle \mathbb{G}, \mathscr{H}_1, \mathscr{H}_2, g, h \rangle$ to $\mathscr{C}$. $\mathscr{C}$ chooses the same group $\mathbb{G}$ and the same master public key h for the simulation of our scheme. Note here that implicitly the *msk* of the simulated scheme is same as the *msk* of the Galindo and Garcia's Signature Scheme instance. He then chooses the four hash functions $\mathscr{H}_1, \mathscr{H}_2, \mathscr{H}_3, \widehat{\mathscr{H}_3}$ and then publishes the $Params = \langle \kappa, g, h, \mathscr{H}_1, \mathscr{H}_2, \mathscr{H}_3, \widehat{\mathscr{H}_3} \rangle$
$\mathscr{C}$ models the four hash functions as random oracles $O_{H_1}, O_{H_2}, O_{H_3}$ and $O_{\widehat{H}_3}$. A list is maintained for each hash function since the responses given for each hash query must be consistent. The lists are denoted by $L_{\mathscr{H}_1}$, $L_{\mathscr{H}_2}$, $L_{\mathscr{H}_3}$ and $L_{\widehat{\mathscr{H}_3}}$. The $L_{\mathscr{H}_i}$ will contain the values of the input to the hash function and the value that is set by $\mathscr{C}$ as the hash value. Another list L_{ID} is maintained which will contain the various keys. L_{ID} will be of the form $\langle ID_i, p_i, k_i, q_i, t_i \rangle$. If some of the values are unknown while updating the values for a particular ID that value is left blank.

Training phase:
$\mathscr{A}_I$ makes use of a number of oracles provided by the $\mathscr{C}$ in this phase.
Oracle $O_{H_1}(ID_i, p_j)$: For the j^{th} query $\mathscr{C}$ checks if the value ID_i, p_j exists on the list $L_{\mathscr{H}_1}$. If it is on the list then return the value of a_j from the list, else he queries the *Oracle* $O_{H_1}^{\mathscr{B}_I}(ID_i, p_j)$ and receives a response a_j and sends this value to $\mathscr{A}_I$. $\mathscr{H}_1(ID_i, p_j)$ is set as a_j and the tuple of values $\langle ID_i, p_j, a_j \rangle$ is added to $L_{\mathscr{H}_1}$

Oracle $O_{H_2}(ID_i, p_j, q_j)$: For the j^{th} query $\mathscr{C}$ checks if the value ID_i, p_j, q_j exists on the list $L_{\mathscr{H}_1}$. If it is on the list then return the value of b_j from the list, else he randomly chooses a response $b_j \in_R \mathbb{Z}_p^*$ and sends this value to $\mathscr{A}_I$. $\mathscr{H}_2(ID_i, p_j, q_j)$ is set as b_j and the tuple $\langle ID_i, p_j, q_j, b_j \rangle$ is added to $L_{\mathscr{H}_2}$

Oracle $O_{H_3}(M_j, ID_j, u_j, p_j, q_j)$: For the j^{th} query $\mathscr{C}$ checks if the value M_j, ID_j, u_j, p_j, q_j exists on the list $L_{\mathscr{H}_3}$. If it is on the list then return the value of c_j from the list, else he queries $\widehat{h}_3 =$*Oracle* $O_{\widehat{H}_3}(M_j, ID_j, u_j, p_j, q_j)$ and $h_2 =$*Oracle* $O_{H_2}(ID_i, p_j, q_j)$ and finally queries the *Oracle* $O_{H_2}^{\mathscr{B}_I}(M_j, ID_i, uq_A^{h_2\widehat{h}_3})$ and receives a response c_j and sends this value to $\mathscr{A}_I$. $\mathscr{H}_3(M_j, ID_j, u_j, p_j, q_j)$ is set as c_j and the tuple of values $\langle M_j, ID_j, u_j, p_j, q_j, c_j \rangle$ is added to $L_{\mathscr{H}_3}$

Oracle $O_{\widehat{H}_3}(M_j, ID_j, u_j, p_j, q_j)$: For the j^{th} query $\mathscr{C}$ checks if the values $M_j, ID_j, u_j, p_j, q_j, \widehat{c}_j$ is already on the corresponding list. If it is on the list he outputs $\widehat{c}_j$ else he randomly chooses a response $\widehat{c}_j \in_R \mathbb{Z}_p^*$ and sends this value to $\mathscr{A}_I$, then $\widehat{\mathscr{H}_3}(M_j, ID_j, u_j, p_j, q_j)$ is set as $\widehat{c}_j$ and the tuple $\langle M_j, ID_j, u_j, p_j, q_j, \widehat{c}_j \rangle$

is added to $L_{\widehat{\mathscr{H}_3}}$

Oracle $O_{PartialKeyExtract}(ID_i)$*:* $\mathscr{A}_I$ chooses an $ID = ID_i$ and sends to $\mathscr{C}$. $\mathscr{C}$ responds to this query as follows :

- If the values of $\langle p_i, k_i \rangle$ already exists on the list L_{ID} in the entry corresponding to ID_i then output the values from the list
- Else,
  - $\mathscr{C}$ queries the *Oracle* $O^{\mathscr{B}_I}_{KeyExtract}(ID_i)$ and obtains the values $\langle p_i, k_i \rangle$
  - Returns p_i, k_i and adds these values to L_{ID} in the entry corresponding to ID_i.

  These values are valid since they return `true` for the check: $g^{k_i} \stackrel{?}{=} p_i h^{\mathscr{H}_1(ID_i, p_i)}$, since: any valid key extract value in the Galindo and Garcia Signature scheme returns true for the above check.

Oracle $O_{PublicKeyGen}(ID_i)$*:* $\mathscr{A}_I$ chooses an $ID = ID_i$ and sends to $\mathscr{C}$. $\mathscr{C}$ responds to this query as follows :

- if the value of p_i, q_i is already set in the entry corresponding to ID_i return them
- Else,
  - $\mathscr{C}$ chooses $m, t_i \in_R \mathbb{Z}_p^*$
  - then set $q_i = g^{t_i}$ and query *Oracle* $O^{\mathscr{B}_I}_{Signature}(ID_i, \mathscr{M})$ and from the signature returned retrieve the value of p_i.
  - return the values $\langle p_i, q_i \rangle$ as the public key. Update L_{ID} with the values of p_i, q_i, t_i in the entry corresponding to ID_i

Oracle $O_{FullPrivateKeyGen}(ID_i)$*:* $\mathscr{A}_I$ chooses an $ID = ID_i$ and sends to $\mathscr{C}$. $\mathscr{C}$ responds to this query as follows:

- Makes sure that both *Oracle* $O_{PublicKeyGen}(ID_i)$ and *Oracle* $O_{PartialKeyExtract}(ID_i)$ have already been run once.
- From the list L_{ID} in the entry corresponding to ID_i return $\langle k_i, t_i \rangle$

PublicKeyReplace(ID_i, q_i', p_i') $\mathscr{A}_I$ 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 (q_i', p_i') in the list L_{ID}. But both q_i' and p_i' should $\in \mathbb{G}$. Also an extra element is added to the list L_{ID} which will act as a flag for the $\mathscr{C}$ to know that the public key has been replaced. Let this extra element be 1. So the entry in L_{ID} for the identities for which public key has been replaced will be of the form: $\langle ID_i, p_i, k_i, q_i, n_i, 1 \rangle$. This means that all verifications done after this will use the new public key.

Oracle $O_{Signature}(ID_i, \mathscr{M}_j)$*:* $\mathscr{A}_I$ chooses an $ID = ID_i$, message $\mathscr{M}$ and sends to $\mathscr{C}$. $\mathscr{C}$ responds to this query as follows :

- For any $ID = ID_i$, query *Oracle* $O^{\mathscr{B}_I}_{Signature}(ID_i, \mathscr{M}_j)$ to obtain $\langle \sigma^*, u_j, \mathscr{M}_j, ID_j, p_j \rangle$
- Query *Oracle* $O_{PublicKeyGen}(ID_i)$ to get $\langle p_i, q_i \rangle$. Then query $\widehat{h}_3$ = *Oracle* $O_{\widehat{H}_3}(M_j, ID_j, u_j, p_j, q_j)$ and h_2 = *Oracle* $O_{H_2}(ID_i, p_i, q_i)$

- Compute $\sigma = \sigma^* + t_i h_2 \widehat{h}_3$
- Now query *Oracle* $O^{\mathscr{B}_I}_{H_2}(\mathscr{M}, ID_i, u)$ to obtain the value $h_2^{\mathscr{B}_I}$ and set this value as $\mathscr{H}_3(M_j, ID_j, u_j, p_j, q_j)$ by adding $\langle M_j, ID_j, u_j, p_j, q_j, h_2^{\mathscr{B}_I} \rangle$ to the list L_{H_3}. If the corresponding entry already exists on the list we start the steps from the beginning again.
- Return $\langle \sigma, u, \mathscr{M}, p_i, q_i \rangle$ as the signature

Correctness for the Signature Oracle's output:

- The signature output by the *Oracle* $O^{\mathscr{B}_I}_{Signature}(ID_i, \mathscr{M})$, $\langle \sigma^*, u, \mathscr{M} p_i, ID_i \rangle$ satisfies the verification algorithm of the Galindo and Garcia Signature scheme.
- i.e. it satisfies the check $g^{\sigma^*} \stackrel{?}{=} u(p_i h^{\mathscr{H}_1^{\mathscr{B}_I}(ID_i, p_i)})^{\mathscr{H}_2^{\mathscr{B}_I}(\mathscr{M}, ID_i, u)}$.
- But we have set the value of $\mathscr{H}_1^{\mathscr{B}_I} = \mathscr{H}_1 = h_1$ and $\mathscr{H}_3(M_j, ID_j, u_j, p_j, q_j) = \mathscr{H}_2^{\mathscr{B}_I}(M_j, ID_i, u_j) = h_3$
- $g^{\sigma} = g^{\sigma^* + t_i h_2 \widehat{h}_3} = u(p_A h^{h_1})^{h_3} (q_A^{h_2})^{\widehat{h}_3}$
- This is same as our verification algorithm so this is a valid signature.

Forgery: Eventually $\mathscr{A}_I$ returns a forgery $\langle \mathscr{M}, \sigma, u^*, q, p, ID \rangle$. $\mathscr{C}$ computes $u = u^* q^{h_2 \widehat{h}_3}$ and returns $\langle \sigma, \mathscr{M}, u, p_i, ID_i \rangle$ to the challenger $\mathscr{B}_I$ as a forgery to the underlying Galindo and Garcia Signature scheme.

Correctness of the solution:

- We know that the forgery returns `true` for the check: $g^{\sigma} \stackrel{?}{=} u^*(p_A h^{h_1})^{h_3} (q_A^{h_2})^{\widehat{h}_3}$
- But we have set the value of $\mathscr{H}_1^{\mathscr{B}_I} = \mathscr{H}_1 = h_1$, $\mathscr{H}_3(M_j, u_j, q_j, p_j) = \mathscr{H}_2^{\mathscr{B}_I}(M_j, ID_i, u^* q^{h_2 \widehat{h}_3}) = h_2^{\mathscr{B}_I}$ and $u = u^* q^{h_2 \widehat{h}_3}$
- Hence, $g^{\sigma} = u^*(p_A h^{h_1})^{h_3} (q_A^{h_2})^{\widehat{h}_3} = u(p_A h^{\mathscr{H}_1(ID_A, p_A)})^{\mathscr{H}_2(\mathscr{M}, ID_A, u)}$
- This is same as the Galindo and Garcia signature scheme verification algorithm. This shows that the forgery submitted by $\mathscr{C}$ to $\mathscr{B}_I$ is a valid forgery.

3.1.2 Probability Analysis:

$\mathscr{C}$ never fails to give a perfect simulation. So if the adversary $\mathscr{A}_I$ gives a forgery with probability ε then the adversary $\mathscr{C}$ returns a valid forgery to the challenger $\mathscr{B}_I$ with the same probability ε.

3.1.3 Unforgeability against type-II adversary

Theorem 2. *If there is a EUF-CMA type-II adversary $\mathscr{A}_{II}$ of the above scheme that succeeds with probability ε then there is a challenger $\mathscr{C}$ running in polynomial time that can produce a forgery for the Schnorr signature scheme with probability atleast $\varepsilon \left(\frac{1}{e(1+q_{PE})} \right)$ where q_{PE} is the number of partial extract queries.*

Proof: Let $\mathscr{C}$ be a challenger whose aim is to win a game simulated by another challenger $\mathscr{B}_{II}$. In this proof $\mathscr{B}_{II}$ simulates an instance of the Schnorr signature scheme[4], and he challenges $\mathscr{C}$ to give a valid forgery to the scheme. To win this game $\mathscr{C}$ makes use of the the advarsary $\mathscr{A}_{II}$ who is assumed to be capable of producing a forgery to our scheme as defined by the security model. We give a small review of the game simulated by $\mathscr{B}_{II}$. Note that the proof is similar to that of the proof for type-I adversary, the difference is that here we use the Schnorr Signature scheme as the underlying signature scheme for the proof.

Game simulated by the Adversary $\mathscr{B}_{II}$ (Schnorr Signature Scheme)

First let us change the notations from the original paper so that they become similar to the notations used in this paper. $\mathscr{B}_{II}$ sets up the signature scheme by setting up a Hash function $\mathscr{H}$, chooses a group $\mathbb{G}$ and a generator of this group g. He sends the values $\langle \mathbb{G}, \mathscr{H}, g \rangle$ to $\mathscr{C}$. He also provides the oracles *Oracle* $O_H^{\mathscr{B}_{II}}(\mathscr{M}, u)$, *Oracle* $O_{KeyGen}^{\mathscr{B}_{II}}()$,*Oracle* $O_{PublicKey}^{\mathscr{B}_{II}}()$, *Oracle* $O_{PublicKey}^{\mathscr{B}_{II}}()$ just outputs a public key and not the private key, i.e. it outputs q_A alone, *Oracle* $O_{Signature}^{\mathscr{B}_{II}}(\mathscr{M}, q_A)$. The *Oracle* $O_{KeyGen}^{\mathscr{B}_{II}}()$ simulates the key generation algorithm which is - choose $t_A \in_R \mathbb{Z}_p^*$ and compute $q_A = g^{t_A}$ finally return $\langle t_A, q_A \rangle$. The *Oracle* $O_{Signature}^{\mathscr{B}_{II}}(\mathscr{M}, q_A)$ simulates the signature algorithm which is - choose $r \in_R \mathbb{Z}_p^*$ and compute $u = g^r$, $\sigma = r + t_A \mathscr{H}(\mathscr{M}, u)$ finally return $\langle \mathscr{M}, \sigma, u \rangle$. The verification algorithm of the signature is given $\langle \mathscr{M}, \sigma, u, q_A \rangle$ and checks if $g^{\sigma} \stackrel{?}{=} u q_A^{\mathscr{H}(\mathscr{M},u)}$.

Setup: $\mathscr{C}$ has to setup the parameters for the game exactly as in our scheme. To Simulate our scheme for the advarsary $\mathscr{A}_{II}$, $\mathscr{C}$ makes use of the challenger $\mathscr{B}_{II}$ in the following way: Once $\mathscr{B}_{II}$ finishes the setup of the signature scheme he sends $\langle \mathbb{G}, \mathscr{H}, g \rangle$ to $\mathscr{C}$. $\mathscr{C}$ chooses the same group $\mathbb{G}$ for the simulation of scheme-I. He then chooses $s \in_R \mathbb{Z}_p^*$ and four hash functions $\mathscr{H}_1, \mathscr{H}_2, \mathscr{H}_3, \widehat{\mathscr{H}_3}$. Computes $h = g^s$, sets $msk = s$ and then publishes the $Params = \langle \kappa, g, h, \mathscr{H}_1, \mathscr{H}_2, \mathscr{H}_3, \widehat{\mathscr{H}_3} \rangle$. Since the type-II adversary models a malicious KGC, $\mathscr{C}$ provides the value of the $msk = s$ to the adversary $\mathscr{A}_{II}$

$\mathscr{C}$ models the four hash functions as random oracles $O_{H_1}, O_{H_2}, O_{H_3}$ and $O_{\widehat{H_3}}$. A list is maintained for each hash function since the responses given for each hash query must be consistent. The lists are denoted by $L_{\mathscr{H}_1}$, $L_{\mathscr{H}_2}$, $L_{\mathscr{H}_3}$ and $L_{\widehat{\mathscr{H}_3}}$. The $L_{\mathscr{H}_i}$ will contain the values of the input to the hash function and the value that is set by $\mathscr{C}$ as the hash value. Another list L_{ID} is maintained which will contain the various keys. L_{ID} will be of the form $\langle ID_i, p_i, k_i, q_i, t_i \rangle$. If some of the values are unknown while updating the values for a particular ID that value is left blank. Another list is maintained having values of the form $\langle ID_i, f_i \rangle$ where each $f_i \in \{0,1\}$ is a generated by a coin toss. This list is denoted by L_{CT}

Training phase:

$\mathscr{A}_{II}$ makes use of a number of oracles provided by the $\mathscr{C}$ in this phase. We assume that the public key queries made by $\mathscr{A}_{II}$ are all distinct since even if any ID is repeated then we look it up in the list and respond with the same value. The different queries are answered in such a way that the adversary cannot differentiate between a real system and the simulated system we are providing.

Oracle $O_{H_1}(ID_i, p_j)$: For the j^{th} query $\mathscr{C}$ checks if the value ID_i, p_j exists on the list $L_{\mathscr{H}_1}$. If it is on the list then return the value of a_j from the list, else randomly chooses a response a_j and sends this value to $\mathscr{A}_{II}$. $\mathscr{H}_1(ID_i, p_j)$ is set as a_j and the tuple of values $\langle ID_i, p_j, a_j \rangle$ is added to $L_{\mathscr{H}_1}$

Oracle $O_{H_2}(ID_i, p_j, q_j)$: For the j^{th} query $\mathscr{C}$ checks if the value ID_i, p_j, q_j exists on the list $L_{\mathscr{H}_1}$. If it is on the list then return the value of b_j from the list, else he randomly chooses a response $b_j \in_R \mathbb{Z}_p^*$ and sends this value to $\mathscr{A}_{II}$. $\mathscr{H}_2(ID_i, p_j, q_j)$ is set as b_j and the tuple $\langle ID_i, p_j, q_j, b_j \rangle$ is added to $L_{\mathscr{H}_2}$

Oracle $O_{H_3}(M_j, ID_j, u_j, p_j, q_j)$: For the j^{th} query $\mathscr{C}$ checks if the value $M_j, ID_j, u_j, p_j, q_j, c_j$ exists on the list $L_{\mathscr{H}_3}$. If it is on the list then return the value of c_j from the list, else he randomly chooses

a response $c_j \in_R \mathbb{Z}_p^*$ and sends this value to $\mathscr{A}_{II}$. $\mathscr{H}_3(M_j, ID_j, u_j, p_j, q_j)$ is set as c_j and the tuple $\langle M_j, ID_j, u_j, p_j, q_j, c_j \rangle$ is added to $L_{\mathscr{H}_3}$

Oracle $O_{\widehat{H}_3}(M_j, ID_j, u_j, p_j, q_j)$: For the j^{th} query $\mathscr{C}$ checks if the value $M_j, ID_j, u_j, p_j, q_j, \widehat{c}_j$ exists on the list $L_{\widehat{\mathscr{H}_3}}$. If it is on the list then return the value of $\widehat{c}_j$ from the list, else he queries $h =$*Oracle* $O_H^{\mathscr{B}_{II}}(M_j, u_j)$ and $h_2 =$*Oracle* $O_{H_2}(ID_i, p_j, q_j)$ and computes the value of $\widehat{c}_j = h/h_2$. $\widehat{\mathscr{H}_3}(M_j, ID_j, u_j, p_j, q_j)$ is set as $\widehat{c}_j$ and the tuple of values $\langle M_j, ID_j, u_j, p_j, q_j, \widehat{c}_j \rangle$ is added to $L_{\widehat{\mathscr{H}_3}}$
Note that this ensures the property that $\mathscr{H}^{\mathscr{B}_{II}}(M_j, u_j) = \widehat{\mathscr{H}_3}(M_j, ID_j, u_j, p_j, q_j)\mathscr{H}_2(ID_i, p_j, q_j)$.

Oracle $O_{PartialKeyExtract}(ID_i)$: $\mathscr{A}_I$ chooses an $ID = ID_i$ and sends to $\mathscr{C}$. $\mathscr{C}$ responds to this query as follows :

- If entry corresponding to ID_i is already on the list then return $\langle p_i, k_i \rangle$ from the list.
- Else,
  - $\mathscr{C}$ runs the usual partial extract algorithm and returns the values of $\langle p_i, k_i \rangle$ and adds them to the list L_{ID} in the entry corresponding to ID_i.

  These values are valid since we used the regular partial extract algorithm.

Oracle $O_{PublicKeyGen}(ID_i)$: $\mathscr{A}_{II}$ chooses an $ID = ID_i$ and sends to $\mathscr{C}$. $\mathscr{C}$ responds to this query as follows :

- if the value of ID_i, f_i is already on the list L_{CT} then return the values of $\langle p_i, q_i \rangle$ that will already exist on the list L_{ID} in the entry corresponding to ID_i.
- Else set the values of f_i by tossing the coin where $Pr[f_i = 1] = \delta$ and $Pr[f_i = 1] = 1 - \delta$ and update L_{CT} with the values ID_i, f_i. Note that the optimal value of δ will be calculated later. This analysis is similar to analysis by Boneh and Franklin[16]
- If $f_i = 1$, then
  - $\mathscr{C}$ gets the value of p_i by querying *Oracle* $O_{PartialKeyExtract}(ID_i)$.
  - then gets the value of $\langle t_i, q_i \rangle$ by querying *Oracle* $O_{KeyGen}^{\mathscr{B}_{II}}()$.
  - Adds the values t_i, q_i, p_i to the list and returns $\langle p_i, q_i \rangle$.
- If $f_i = 0$, then
  - $\mathscr{C}$ gets the value of p_i by querying *Oracle* $O_{PartialKeyExtract}(ID_i)$.
  - then gets the value of q_i by querying *Oracle* $O_{PublicKey}^{\mathscr{B}_{II}}()$.
  - Adds the values q_i, p_i to the list and returns $\langle p_i, q_i \rangle$.

Note that if $f_i = 1$ the $\mathscr{C}$ knows both the partial private key and the partial public key. Otherwise he knows only the partial public key.
Oracle $O_{FullPrivateKeyGen}(ID_i)$: $\mathscr{A}_{II}$ chooses an $ID = ID_i$ and sends to $\mathscr{C}$. $\mathscr{C}$ responds to this query as follows:

- If ID_i, f_i does not exist on the list L_{CT} then run *Oracle* $O_{PublicKeyGen}(ID_i)$.
- Now if $f_i = 1$, then

– Return $\langle k_i, t_i \rangle$ (if k_i is not on the list then retrieve using *Oracle* $O_{PartialKeyExtract}(ID_i)$)

- If $f_i = 0$, then
  - Abort (since if $f_i = 0$ we do not know the partial private key for that user)

Oracle $O_{Signature}(ID_i, \mathscr{M})$: $\mathscr{A}_I$ chooses an $ID = ID_i$, message $\mathscr{M}$ and sends to $\mathscr{C}$. $\mathscr{C}$ responds to this query as follows :

- For any $ID = ID_i$, query *Oracle* $O^{\mathscr{B}_{II}}_{Signature}(\mathscr{M}, q_i)$ to obtain $\langle \sigma^*, u, \mathscr{M} \rangle$
- Obtain the value of h_3 =*Oracle* $O_{H_3}(M_j, u_j, q_j, p_j)$ and k_i =*Oracle* $O_{PartialKeyExtract}(ID_i)$.
- Compute $\sigma = \sigma^* + k_i h_3$
- Return $\langle \sigma, u, \mathscr{M}, p_i, q_i \rangle$ as the signature

The signature that is given as output by the strong signature oracle is indeed a valid signature as shown below:
Correctness for the Signature Oracle's output:

- The signature $\langle \sigma^*, u, \mathscr{M} \rangle$ output by *Oracle* $O^{\mathscr{B}_{II}}_{Signature}(\mathscr{M}, q_i)$ satisfies the verification algorithm of the Schnorr signature scheme.
- i.e. $g^{\sigma^*} \stackrel{?}{=} u q_A^{\mathscr{H}(\mathscr{M},u)}$, but we have ensured that $\mathscr{H}^{\mathscr{B}_{II}}(M_j, u_j) = \widehat{\mathscr{H}_3}(M_j, u_j, q_j, p_j)\mathscr{H}_2(ID_i, p_j, q_j)$
- $\Rightarrow g^{\sigma^* + k_i h_3} = u(p_A h^{h_1})^{h_3}(q_A^{h_2})^{\widehat{h}_3}$, which is same as the verification algorithm of our scheme. This shows that the signature oracle outputs valid signatures.

Forgery: Eventually $\mathscr{A}_I$ returns a forgery $\langle \mathscr{M}, \sigma^*, ID, u, q, p \rangle$. $\mathscr{C}$ computes $\sigma = \sigma^* - k_i h_3$ and returns $\langle \sigma, u, \mathscr{M}, q \rangle$ to the challenger $\mathscr{B}_{II}$ as a forgery to the underlying Schnorr Signature scheme.

Correctness of the solution:

- We know that the forgery given by $\mathscr{A}_{II}$ $\langle \mathscr{M}, \sigma^*, u, q, p \rangle$, satisfies $g^{\sigma^*} = u(p_A h^{h_1})^{h_3}(q_A^{h_2})^{\widehat{h}_3}$
- $\Rightarrow g^{\sigma^* - k_i h_3} = \dfrac{u(p_A h^{h_1})^{h_3}(q_A^{h_2})^{\widehat{h}_3}}{(p_A h^{h_1})^{h_3}} = u q_A^{h_2 \widehat{h}_3}$
- But since $\mathscr{H}^{\mathscr{B}_{II}}(M_j, u_j) = \widehat{\mathscr{H}_3}(M_j, u_j, q_j, p_j)\mathscr{H}_2(ID_i, p_j, q_j)$ we have $g^{\sigma} \stackrel{?}{=} u q_A^{\mathscr{H}(\mathscr{M},u)}$

3.1.4 Probability Analysis:

$\mathscr{C}$ fails to give a perfect simulation if:

- $\mathscr{A}_{II}$ requests Full Private key of an ID_i for which $f_i = 0$ on the list L_{CT}.
- $\mathscr{A}_{II}$ gives forgery on identity where $f_i = 1$.

Now the probability of not aborting in the q_{FPK} number of queries can be done by having the value of $f_i = 1$. This event happens with probability δ. Here δ is the probability of the coin toss in *Oracle* $O_{PublicKeyGen}(ID_i)$. Hence probability of not aborting in the full private key extract query is $\delta^{q_{FPK}}$. Total probability of not aborting is then $\delta^{q_{FPK}}(1-\delta)$. Maximizing this value gives the optimal value of $\delta_{opt} = 1 - 1/(q_{FPK}+1)$. Using this value we find the total probability of not aborting as $1/e(1+q_{FPK})$. This analysis is similar to the analysis in Boneh and Franklin IBE[16].

So if the adversary $\mathscr{A}_{II}$ gives a forgery with probability ε then the adversary $\mathscr{C}$ returns a valid forgery to the challenger $\mathscr{B}_{II}$ with probability atleast $\varepsilon\left(\frac{1}{e(1+q_{PE})}\right)$ where q_{PE} is the number of partial extract queries.

4 Efficiency Comparison

Many certificateless schemes have been proposed but to the best of our knowledge the only scheme without pairing to be online/offline is by Ge et. al.[14]. Our first scheme is highly efficient. We compare it with the scheme by Ge et. al.[14]. It is the most efficient certificateless signature scheme till date and in addition it is also online/offline. Though there are many certificateless signature schemes with pairing we do not consider them for comparison as our scheme is a scheme without pairing computations.

Table 1: Efficiency Comparison of Certificateless Signature Schemes without pairings

Scheme	Signature Cost	Verification Cost	Remarks
Ge et. al.[14]	$1\mathscr{H} + 2GE + 2mm + 2ma$	$3\mathscr{H} + 7GE + 4GM$	Only online/offline Scheme without pairings
Our Scheme-I	$2\mathscr{H} + 1GE + 2mm + 2ma$	$4\mathscr{H} + 3GE + 3GM + 1mm$	Most efficient Scheme. Additionally Online/Offline

$\mathscr{H}$ - Hash Computation, *GE* - Group Exponentiations, *GM* - Group Multiplications, *mm* - Modular multiplication, *ma* - Modular Addition

5 Conclusions

In this paper we present a Certificateless signature scheme. This scheme is the most efficient certificateless signature scheme in existence. It has an additional property of being online/offline. We have compared this scheme with the only other certificateless signature scheme without pairings in existence and showed that our scheme is more efficient. We prove our scheme in the random oracle model by showing that if a forger exists for our scheme then Galindo and Garcia's Signature(GGS)[15] scheme or the Schnorr signature scheme[4] will also be insecure. Since both these schemes are known to be secure our scheme is also secure in the random oracle model.

References

[1] Adi 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.

[2] Sattam S. Al-Riyami and Kenneth 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.

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

[4] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Proc. of the 9th Annual International Cryptology Conference (CRYPTO'89), Santa Barbara, California, LNCS*, volume 435, pages 239–252. Springer-Verlag, August 1989.

[5] Zhong Xu, Xue Liu, Guoqing Zhang, Wenbo He, Guanzhong Dai, and Weihuan 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.

[6] Changji Wang, Dongyang Long, and Yong Tang. An efficient certificateless signature from pairings. *International Journal of Network Security*, 8(1):96–100, January 2009.

[7] Guoyan Zhang and Shaohui 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.

[8] Lei Zhang and Futai 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.

[9] Lifeng Guo, Lei Hu, and Yong 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.

[10] Changji Wang and Hui 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.

[11] Bessie C. Hu, Duncan S. Wong, Zhenfeng Zhang, and Xiaotie Deng. Certificateless signature: a new security model and an improved generic construction. *Designs, Codes and Cryptography*, 42(2):109–126, February 2007.

[12] Raylin Tso, Xun Yi, and Xinyi 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.

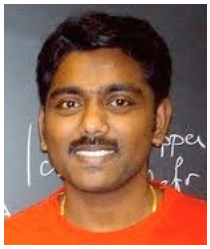
[13] Zhong Xu, Xue Liu, Guoqing Zhang, and Wenbo He. A certificateless signature scheme for mobile wireless cyber-physical systems. In *Proc. of the 28th IEEE International Conference on Distributed Computing Systems (ICDCS'08), Beijing, China*, pages 489–494. IEEE, June 2008.

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

[15] David Galindo and Flavio D. Garcia. A schnorr-like lightweight identity-based signature scheme, gammarth, africa. In *Proc. of the 2nd International Conference on Cryptology in Africa (AFRICACRYPT'09), Gammarth, Tunisia, LNCS*, volume 5580, pages 135–148. Springer-Verlag, June 2009.

[16] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proc. of the 21st Annual International Cryptology Conference (CRYPTO'01), Santa Barbara, California, LNCS*, pages 213–229. Springer-Verlag, August 2001.

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.