A Revocable Online-Offline Certificateless Signature Scheme without Pairing

Karthik Abinav* Saikrishna Badrinarayanan† C. Pandu Rangan‡
S. Sharmila Deva Selvi§ S. Sree Vivek¶ Vivek Krishna Pradhan‖

Theoretical Computer Science Lab,
Department of Computer Science and Engineering,
Indian Institute of Technology, Madras

Abstract

Certificateless Public key Cryptography is a widely studied paradigm due to its advantages of not having the key-escrow problem and the lack of use of certificates. Online-Offline signature schemes are extremely relevant today because of their great practical applications. In an online-offline signature scheme all the heavy computation is done on powerful processors and stored securely in the offline phase, and the online component requires only light computation. Hence, it is widely used in several low-resource devices like mobile phones, etc. Revocation is another important problem of wide interest as it helps to keep a check on misbehaving users. Currently, there are very few revocable certificateless signature schemes in the literature. We have addressed some of the limitations of the previously existing schemes and designed a new model for the same that involves periodic time generated keys. We present a revocable online-offline certificateless signature scheme without pairing. Pairing, though a very useful mathematical function, comes at the cost of heavy computation. Our scheme is proved secure in the random oracle model using a tight security reduction to the computational Diffie-Hellman problem.

Keywords
Certificateless cryptography, Online/Offline, Revocable, Tight security, Random oracle.

*Email: kabinav@cse.iitm.ac.in
†Email : bsai@cse.iitm.ac.in
‡Email : prangan@cse.iitm.ac.in
§Email : sharmila@cse.iitm.ac.in
¶Email : svivek@cse.iitm.ac.in
‖Email : vivek.k.pradhan@gmail.com

Contents

1 Introduction

The traditional public key infrastructure(PKI) based systems use a certificate to bind a public key with it's user's identity. The drawback is that we need a trusted third-party for the purpose of storing and issuing certificates. The next paradigm introduced was identity based cryptography(IBC) by Adi Shamir[13]. In this model, the user's public key is generated using the user's identity by a trusted authority called the Private Key Generator(PKG). Though this helps in doing away with the need for a trusted authority to issue certificates, the drawback is that a lot of power is vested upon the PKG, who can impersonate any user. This is called the "Key-Escrow" problem.

Certificateless Cryptography
Certificateless public key cryptography(CLPKC) first proposed by Al Riyami and Patterson[1] tries to resolve the key escrow problem while keeping the implicit certification property of IBC. The user first gets a "partial public key" and a "partial private key" from a trusted authority called the Key Generating Center(KGC). The user then adds some secret information on his own, to create his public and private keys. The user's public key is similar to the traditional PKI as it is generated by the user. However, it does not need to be explicitly certified as it has been generated using some "partial private key" obtained from the trusted authority. The KGC does not know the users' private keys as they contain some secret information, which are generated by the users themselves, thereby removing the Key-Escrow problem in IBC. Therefore, CLPKC somewhat lies between PKI and IBC.

The notion of online/offline schemes was first introduced by Even, Goldrich and Micali[5]. These schemes are relevant in the case of low-resource devices like mobile phones, which cannot perform heavy computations. In such situations, the signature consists of two parts - an offline part and an online part. The offline signature component involves heavy computations and is done on a powerful processor. Several such tuples are created and stored securely. This is done without the knowledge of any message. When a signature is to be generated on a low-resource device, an offline signature component is retrieved, followed by inexpensive computation to generate the online part, thus forming the full signature. The generation of the online component is done after the message is known. This widely used paradigm thus requires a high level of security.

Currently, there are only four certificateless online-offline signature schemes in literature - namely [16], [6] , [11] and [12]. The scheme in [16], which can naturally be converted into the online/offline setting, has been proved insecure in [19]. The schemes in [6] and [11] are proved secure using the forking lemma. One problem with such a proof is that it does not offer tight security and hence, for the scheme to be truly effective, large keys have to be used which is not desirable. Though the scheme in [12] has a tight security reduction, it involves pairing, which is computationally expensive, and hence not preferable.

The concept of revocability becomes a major issue when signature protocols are used in real life. Sometimes, the private keys of users could get compromised or the user could misbehave and we should have an efficient revocation mechanism to overcome that.

Initially, one solution to revocation in CLPKC([17], [9], [18]) was to introduce an online mediator called the Security Mediator(SEM). Here, the user's partial private key is split into two components; one is given to the user and the other to the SEM. However, the requirement of the SEM for the creation of each signature results in overburdening it. Also, the SEM has to maintain a large amount of keys, giving more scope for an attacker to compromise a key.

Another way to tackle this issue is by having a time generated component in the partial key. A time generated key is issued for a specific interval of time and generated at regular time periods ([1], [17]). To revoke a user, the KGC stops giving the user the partial keys, thereby preventing the user from being able to generate the full public key and the full private key for future time periods. For example, the time period could be for a day. Hence, for the next day the user will be issued a new partial key. This is similar to the daily token system available at amusement parks. It could be for other time periods also. For example, applications like banking could require time periods of a few weeks too. Hence, for a misbehaving user, the KGC can revoke it by not sending the partial keys for future intervals, thereby preventing the user from signing further. In [1], formal proof of security for such a system isn't stated. In [17], the use of pairing operation makes it expensive.

A time-interval based certificateless revocable signature scheme has been proposed in [14]. There are several limitations in their scheme. Firstly, there is no key sanity check for the partial keys got by the user from the KGC. If the KGC sends the time-updated key over a public channel as stated in the paper, it may be intercepted and modified by an adversary. The lack of a key sanity check prevents the user from knowing whether it was sent by the KGC. Secondly, there is no key sanity check for each user's public key. Additionally, the size of the partial private keys are quite big as they are group elements. Also, the use of pairing makes the computations very expensive.

1.1 Our Contribution

We propose a time-interval based revocable online-offline certificateless signature scheme that does not use pairing. We prove our scheme secure using a tight security reduction to the Computational Diffie Hellman problem in the random oracle model. In this scheme, we give key sanity checks for both user verification and public verification. The size of our partial private keys are small as they are elements of the field $\mathbb{Z}_q$. Unlike in [14], signatures can be generated at any time instant providing greater flexibility. Our scheme has a tight security reduction and doesn't make use of the pairing operation, thereby distinguishing it from the other online-offline certificateles signature schemes in the literature.

2 Preliminaries

2.1 Computational Assumptions

2.1.1 Discrete Logarithmic Problem

Let P, $aP \in \mathbb{G}$ with generator P and $a \in \mathbb{Z}_p^*$, such that a is unknown. The Discrete Log Problem(DLP) in $\mathbb{G}$ is to compute the value of a. The DLP is assumed to be a computationally hard problem for certain groups $\mathbb{G}$. This means that for any probabilistic polynomial time algorithm, the advantage of the algorithm in computing a is negligibly small.

2.1.2 Decision Diffie-Hellman Problem

Let P, aP, bP, $Q \in \mathbb{G}$ with generator P and $a, b \in \mathbb{Z}_p^*$, such that a, b are unknowns. The Discrete Diffie-Hellman(DDH) Problem in $\mathbb{G}$ is to decide if $Q = abP$. The DDH problem is assumed to be a computationally hard problem for some groups $\mathbb{G}$. This means that for any probabilistic polynomial time algorithm, the advantage of the algorithm in deciding it is negligibly small.

2.1.3 Computational Diffie-Hellman Problem

Let P, aP, $bP \in \mathbb{G}$ with generator P and $a, b \in \mathbb{Z}_p^*$, such that a, b are unknowns. The Computational Diffie-Hellman(CDH) Problem in $\mathbb{G}$ is to compute $Q = abP$. The CDH problem is assumed to be a computationally hard problem for certain groups $\mathbb{G}$. This means that for any probabilistic polynomial time algorithm, the advantage of the algorithm in deciding it is negligibly small.

Note: Throughout this paper, wherever we refer to a group $\mathbb{G}$, we refer to such a group in which DLP, DDH and CDH are computationally hard.

2.2 A Revocable Online-Offline Certificateles Signature Scheme

A certificateless online/offline scheme[1] will contain the following eight probabilistic polynomial time algorithms - Setup, Partial Extract, Set Secret Value, Public Key Generation, Private Key Generation, Offline Signature, Online Signature, Verification.
Here, a particular user is denoted as U_A and his identity as ID_A. Also, time keys are provided for a fixed time quantum in the system. We denote this time quantum with the symbol α. Additionally, we use the following naming scheme: UPK - User Public Key, FPK - Full Public Key, PPK - Partial Public Key, USK - User Secret Key, FSK - Full Secret Key, PSK - Partial Secret Key.

[1]Definitions based on [11]

- **Setup(K)**: This algorithm is run by the KGC. It generates the master secret key(MSK) first and then the public parameters(params), given a security parameter K as the input. Along with the other information, params additionally contains α. The KGC publishes params and keeps the MSK secret.
- **Partial Extract(params, ID_A, t)**: This algorithm is run by the KGC. Given params, user identity ID_A and the start of the time interval under consideration t, this algorithm generates the Partial Secret Key(PSK) and the Partial Public Key(PPK) of a user U_A and sends them to the user. This can be sent over a public or private channel.
- **Set Secret Value(params, K, t)**: This algorithm is run by each user to generate his user secret key. The input to this algorithm is params, the security parameter K and the start of the time interval under consideration t. For a user U_A, the user secret key is denoted by t_A. This value is not revealed to anyone.
- **Public Key Generation(params, ID_A, USK, PPK, t)**: This algorithm is performed by the user. The input to this algorithm is params, the user identity ID_A corresponding to the user U_A, his user secret key, his partial public key and the start of the time interval under consideration t. The output of this algorithm is the user public key. This step is independent of the Private Key Generation and hence it can be performed even before knowing the full secret key. The full public key is the partial public key together with the user public key.
- **Private Key Generation(params, ID_A, PSK, USK, t)**: This algorithm is run by each user to generate his full private key. The input to this algorithm is params, the user identity ID_A corresponding to user U_A, his partial secret key, his user secret key and the start of the time interval under consideration t. The output is his full secret key. This is kept secret by the user and even KGC does not have full knowledge about it.
- **Offline Signature(params, FSK, t)**: The signer generates the offline component ϕ using this algorithm. He does not have any information about the message. The input to this algorithm are params, the full secret key and the start of the time interval under consideration. The output is the offline component of the signature. The offline signatures are usually pre-computed and a large number of them are stored securely for later use in the online phase. In this case, for a time interval under consideration, the offline signatures are pre-computed and stored in a secure and trusted location.
- **Online Signature(params, ID_A, M, FSK, ϕ, t')**: Given a message M, params, the user identity ID_A corresponding to the user U_A, the full secret key, the offline component of the signature, and the current time instant t', the signer runs the algorithm in the online phase to generate the certificateless signature σ. For each signature computation, a fresh offline signature must be retrieved and used. Note that the time t' is the current instant and not necessarily the start of the time interval under consideration. It is part of σ.
- **Verification(params, ID_A, M, σ, FPK, t)**: This algorithm is run by a verifier to determine whether the given signature is valid or not. The signature verification can be done by anyone using params, the signer's identity ID_A, the message M, the signer's public key, the start of the time interval under consideration t, the signing timestamp t' and FPK. (t' is part of σ). First, the verification algorithm is run to check if the signature is valid. After that, it is verified that t' lies in the interval $(t, t+\alpha)$, where t is the beginning of the time interval under consideration. If both the above conditions are satisfied, the algorithm outputs that the signature is valid. If either or both of them fail, the algorithm outputs that the signature verification failed.

Key Sanity check:
Key sanity check is done at two different places

- **User Verification**: Whenever the KGC gives the user a PPK and PSK, he runs a key sanity check to verify if the keys given by the KGC are of the correct mathematical form.
- **Public Verification**: A different user($\neq U_A$), who intends to use the public key of user U_A to verify this user's signatures must first ensure that the public key he receives is valid.

2.3 Security Models

For any certificateless crypto system[2], there are two types of adversaries A_I and A_{II}. A_I denotes a dishonest user who can replace other users' public keys but has no knowledge about the master secret key. A_{II} represents the malicious KGC who has knowledge of MSK but is trusted not to replace the public keys. Additionally, for a revocable certificateless crypto system, there is a third kind of adversary A_{III}. A_{III} represents a revoked user - i.e. a user whose partial public key and partial private key have been revoked by the KGC. He cannot replace other users' public keys too.

2.3.1 Type I adversary game

Setup: The challenger starts the game by setting the public parameters(params) and sends it across to A_I. The MSK is kept secret.
A_I denotes a dishonest user who can replace other users' public keys but has no knowledge about the master secret key.
A type I adversary can perform the following operations

Training Phase:

- **Hash queries**: The adversary has access to all the hash oracles.
- **Partial Extract queries**: These can be made for all identities except for those in the set of target identities. Also, the adversary cannot query the partial extract oracle for those identities for which he has replaced the public key.
- **Private Key Generation queries**: These can be made for all identities except for those in the set of target identities.. However, private key generation queries cannot be made on those adversaries for which the public key replacement has been made.
- **Public Key Generation queries**: These can be made by the adversary for all identities.
- **Public Key Replace**: A_I sends a new public key to replace the previous public key for some identity. The challenger verifies that this public key is valid and then replaces it if so. All signing and verification done after this will use the new public key.
- **Signature queries**: These can be made by A_I for all identities. The output represents the full signature after the online phase. We do not give a separate offline signature oracle, as the offline signatures are assumed to be securely stored on a storage device and hence cannot be revealed to the adversary.

Forgery: After the training phase, the adversary outputs a forgery for one of the target identities. He wins the type I game if he outputs a valid forgery i.e. it passes the signature verification test and wasn't the output of a signature oracle query during the training phase.

2.3.2 Type II adversary game

Setup: The challenger starts the game by setting the public parameters(params) and sends it across to A_{II}. The MSK is also given in this case.
A_{II} represents the malicious KGC who has knowledge of MSK but is trusted not to replace the public keys.
A type II adversary can perform the following operations

Training Phase:

- **Hash queries**: The adversary has access to all the hash oracles.
- **Partial Extract queries**: These oracle is not provided since A_{II} already has the MSK and he can compute the PPK and PSK.
- **Private Key Generation queries**: These can made for all identities except for any identity in the set of target identities.
- **Public Key Generation queries**: These can be made by the adversary for all identities.

[2]Security Game for Type 1 and Type 2 adversary based on [11].

- **Signature queries**: These can be made by A_{II} for all identities. The output represents the full signature after the online phase. We do not give a separate offline signature oracle as the offline signatures are assumed to be securely stored on a storage device and hence cannot be revealed to the adversary.

Forgery: After the training phase, the adversary outputs a forgery for one of the target identities. He wins the type II game if he outputs a valid forgery i.e. it passes the signature verification test and wasn't the output of a signature oracle query during the training phase.

2.3.3 Type III adversary game

A_{III} denotes a revoked user, i.e. a user for which his partial private keys have been revoked. It represents a user who earlier was functioning properly, but whose keys were revoked for whatever reasons. He currently has no active keys(i.e. in the time period under consideration) and he acts as an adversary in the system. In this game, we give the adversary training till the time he has been revoked. So, the game goes as follows: The adversary gets training upto the beginning of an interval $t^{\#}$. This represents the interval in which he gets revoked. In the training phase, he gets access to a lot of information which is listed below. He cannot get keys for any identity for any time period after the time when he has been revoked. After the training phase, he performs a forgery for one of the target identities(which are randomly chosen by the challenger), for a time instant $t^{'*} > t^{\#}$ (after he has been revoked). In the revoked period, he has access to no new information(i.e. after the training).

Setup: The challenger starts the game by setting the public parameters(params) and sends it across to A_{III}. The MSK is kept secret.

The things he has accesss to are listed below. A type III adversary can perform the following operations

Training Phase:

- **Hash queries**: The adversary has access to all the hash oracles. The inputs can contain a time instant even after the beginning of the challenge time period.
- **Partial Extract queries**: These can be made for any identity, for any time period before the challenge time period.
- **Private Key Generation queries**: These can be made for all identities before the challenge time period.
- **Public Key Generation queries**: These can be made by the adversary for all identities before the challenge time period.
- **Signature queries**: These can be made by A_{III} for all identities for any time instant before the challenge time period. The output represents the full signature after the online phase. We do not give a separate offline signature oracle, as the offline signatures are assumed to be securely stored on a storage device and hence, cannot be revealed to the adversary.

Note: Except in the case of hash queries, none of the other queries can be made for a time instant after the beginning of the challenge period.

Forgery:

After the training phase, the adversary outputs a forgery for one of the target identities for some instant $t^{'*}$ such that $t^{'*} > t^{\#}$. He wins the type III game if he outputs a valid forgery i.e. it passes the signature verification test.

2.4 Definition of Tight security

The scheme is said to have a tight security reduction to an underlying hard problem if the advantage of the challenger in breaking the hard problem is just negligibly smaller than the advantage of the adversary in breaking the scheme. In the case of our scheme, we have a tight security reduction meaning that each of the three games satisfy the above definition.

In our analysis, we have used the technique by Coron in [4]. We assign a probability of p to each identity as being a target identity, and choose p suitably so that it maximises the value of the advantage probability.

3 Our Scheme

- **Setup(K)**: Given K as security parameter, the key generating center(KGC) chooses a group $\mathbb{G}$ of order q and generator of this group P. Then x is chosen randomly from $\mathbb{Z}_q^*$. The KGC then sets the master secret key(MSK) as x and sets $P_3 = xP$. The KGC then chooses 7 hash functions defined below:
  - $\mathbb{H}_1$: $\{0,1\}^* \times \mathbb{G} \times \{0,1\}_t^* \rightarrow \mathbb{Z}_q^*$
  - $\mathbb{H}_2$: $\{0,1\}^* \times \mathbb{G} \rightarrow \mathbb{Z}_q^*$
  - $\mathbb{H}_3$: $\{0,1\}^* \times \mathbb{G} \rightarrow \mathbb{G}$
  - $\widehat{\mathbb{H}_3}$: $\{0,1\}^* \times \mathbb{G} \times \{0,1\}_t^* \rightarrow \mathbb{G}$
  - $\mathbb{H}_4$: $\mathbb{G} \rightarrow \mathbb{G}$
  - $\mathbb{H}_5$: $M \times \{0,1\}^* \times \mathbb{G}^5 \times \{0,1\}_t^* \rightarrow \mathbb{Z}_q^*$
  - $\mathbb{H}_6$: $\mathbb{G}^6 \rightarrow \mathbb{Z}_q^*$

  The KGC also chooses the value of the time quantam α. The KGC keeps the MSK secret and makes params public, where params = (K,P,P_3,$\mathbb{H}_1$,$\mathbb{H}_2$,$\mathbb{H}_3$, $\widehat{\mathbb{H}_3}$,$\mathbb{H}_4$,$\mathbb{H}_5$,$\mathbb{H}_6$, α).

  Note: In the hash functions - $\{0,1\}_t^*$ indicates the time.
- **Partial Extract**(params, ID_A, t): Given an identity ID_A and the start of a time interval t, the KGC does the following to generate the partial public key(PPK) and the partial secret key(PSK).
  - Choose randomly $s \in_R \mathbb{Z}_q^*$
  - Compute $P_2 = sP$
  - Compute $\widehat{P_2} = s\widehat{\mathbb{H}_3}(ID, P_2, t)$
  - Compute $d_A = s + x\mathbb{H}_1(ID, P_2, t)$
  - Choose a random $k_1 \in_R \mathbb{Z}_q^*$.
  - Compute $u = k_1 P$ and $v = k_1\widehat{\mathbb{H}_3}(ID, P_2, t)$
  - Choose c_1 as $\mathbb{H}_6(u, v, P_2, \widehat{P_2}, P, \widehat{\mathbb{H}_3}(ID, P_2, t))$
  - Compute $s_1 = k_1 + c_1 s$
  - Return PSK = $< d_A, t >$ and $(P_2, \widehat{P_2}, s_1, c_1, t)$ as the PPK.

  Note: Here, t is also sent along with the keys to indicate to the user which time interval the keys are for, thereby preventing confusion. It could be removed from the partial private key to make it more efficient

  Key Sanity Check For User Verification

  Now, the user can verify whether the partial keys received were valid for the time interval under consideration using the following check:

  - Compute $u = s_1 P - c_1 P_2$ and $v = s_1\widehat{\mathbb{H}_3}(ID, P_2, t) - c_1\widehat{P_2}$.
  - Compute $\widehat{c_1} = \mathbb{H}_6(u, v, P_2, \widehat{P_2}, P, \widehat{\mathbb{H}_3})$.
  - Check if $c_1 = \widehat{c_1}$.
  - Check if $d_A P = P_2 + H_1(ID, P_2, t)P_3$
  - Deduce that it is valid if both the above conditions are true.
    (i.e $c_1 = \widehat{c_1}$ and $d_A P = P_2 + P_3 H_1(ID, P_2, t)$)

  This is a check done once in every new time interval.
- **Set Secret Value**(params, t): The user U_A having an identity ID_A performs the following operation to generate the User secret key(USK):
  - Choose randomly $t_A \in_R \mathbb{Z}_q^*$ as the USK
- **Public Key Generation**(params, ID_A, USK, PPK, t): The user U_A perfoms the following operation:
  - Compute $P_1 = t_A P$.
  - Compute $\widehat{P_1} = t_A \mathbb{H}_3(ID_A, P_1)$.
  - Choose a random $k_2 \in_R \mathbb{Z}_q^*$.

- Compute $u = k_2 P$ and $v = k_2\widehat{\mathbb{H}_3}(ID, P_1, t)$
- Choose c_2 as $\mathbb{H}_6(u, v, P_1, \widehat{P_1}, P, \widehat{\mathbb{H}_3}(ID, P_1, t))$
- Compute $s_2 = k_2 + c_2 t_A$
- The full public key FPK is $(P_1, P_2, \widehat{P_1}, \widehat{P_2}, s_1, c_1, s_2, c_2, t)$.

Key Sanity Check For Public Verification

A different user, who intends to use this public key to verify this user's(U_A) signatures must first ensure that the public key he receives are valid. This can be done by the following check:

- Compute $u_1 = s_1 P - c_1 P_2$ and $v_1 = s_1\widehat{\mathbb{H}_3}(ID, P_2, t) - c_1\widehat{P_2}$.
- Compute $\widehat{c_1} = \mathbb{H}_6(u_1, v_1, P_2, \widehat{P_2}, P, \widehat{\mathbb{H}_3})$.
  Compute $u_2 = s_2 P - c_2 P_1$ and $v_2 = s_2\widehat{\mathbb{H}_3}(ID, P_1, t) - c_2\widehat{P_1}$.
- Compute $\widehat{c_2} = \mathbb{H}_6(u_2, v_2, P_1, \widehat{P_1}, P, \widehat{\mathbb{H}_3})$.
- Deduce that it is valid if and only if $c_1 = \widehat{c_1}$ and $c_2 = \widehat{c_2}$.

Note: In the full public key, only the components P_1, P_2 are used to verify signatures.
t is there for the receiver to know the time interval during which the public key he receives are to be used,i.e they are valid for the interval from t to $t + \alpha$.
The other components are present for the verifier to check that the public key of the signer that he received is valid, and is that of the intended user. This is just a one-time check. Rather, it needs to be validated once every time-period when he receives a new public key for that time period.

- **Private Key Generation**(params, ID_A, USK, PSK, t): The user U_A perfoms the following operation:
  - Compute $n_A = d_A + t_A \mathbb{H}_2(ID, P_1)$
  - The FSK is $< n_A, t_A >$.
  - This value is kept secret.
- **Offline Signature**(params, FSK, t): The offline components of the signature are calculated as below:
  - Choose $k \in_R \mathbb{Z}_q^*$.
  - Compute $H = \mathbb{H}_4(kP)$.
  - Compute $Z_1 = n_A H, Z_2 = kH, Z_3 = kP$.
  - Return $\phi = \ < k, H, Z_1, Z_2, Z_3, t >$ as the offline signature.
- **Online Signature**(params, ID_A, M, FSK, ϕ, t'): To generate the full signature for a message M at a time t' a fresh offline signature tuple ϕ is taken. Then, the following operations are performed as below:
  - Compute $c = \mathbb{H}_5(M, ID_A, Z_1, Z_2, Z_3, P_1, P_2, t')$
  - Compute $v = k + cn_A$
  - $\sigma = \ < Z_1, v, c, t' >$.

  Note: Each time an online signature has to be generated, a fresh offline signature tuple is retrieved.
- **Signature Verification**(params, ID_A, M, σ, FPK): To verify the given signature a verifier does the following:
  - Compute $N_A = P_2 + \mathbb{H}_1(ID, P_2, t)P_3 + \mathbb{H}_2(ID, P_1)P_1$
  - Compute $Z_3 = vP - cN_A$
  - Compute $H = \mathbb{H}_4(Z_3)$
  - Compute $Z_2 = vH - cZ_1$ (v,c are part of σ)
  - Verify that $c = \mathbb{H}_5(M, ID_A, Z_1, Z_2, Z_3, P_1, P_2, t')$.

  Here, t' is the time instant at which the signature was generated. It is part of the signature. t is the start of the time interval at which it was generated, i.e the start of the time interval during which it is valid.

  Even if the above verification holds, the verifier also checks that t' lies in the interval t to $t + \alpha$. I.e check that $t <= t' <= t + \alpha$.

Only if this also holds, the signature can be verified to be a valid one.

Note: The verifier first verifies that the signer's public keys are valid (using the key sanity check for public verification). And this validation is a one-time process for each time interval.

4 Security Proof

In the following proofs,all the hash functions are modeled as random oracles.

4.1 Proof for Type I Adversary

Theorem 1: If there exists an adversary A_I that can forge a signature for the above scheme with probability ϵ in time t_{adv}, then there exists a challenger C who can solve the CDH problem with probability atleast ϵ' in time t_{ch} , such that

$$\epsilon' \geq \epsilon \left[\frac{1}{q_{fse} + q_{pe} + 1} \left[\frac{q_{fse} + q_{pe}}{q_{fse} + q_{pe} + 1} \right]^{q_{fse}+q_{pe}} .(1 - \frac{1}{q}) \right]$$

$t_{ch} = S + t_{adv} + (q_1 + q_2 + q_3 + \widehat{q_3} + q_4 + q_5 + q_6 + q_{psq} + q_{fsq} + q_{fpq} + q_{sq} + q_{pkr})O(1)$. q_{id} = number of distinct identities queried by the adversary, q = is the order of the group, $\mathbb{G}$ in which the hard problem can be solved by adversary to break the system, q_{pe} = number of partial extract queried, q_{fse} = number of full secret key extracts.
q_i = number of queries to the H_i hash oracle(where $i = 1, 2..6$), $\widehat{q_3}$ = number of queries to the $\widehat{H_i}$, q_{psq} = number of partial extract queries, q_{fsq} = number of full secret key queries, q_{fpq} = number of full public key queries , q_{sq} = number of signature queries, q_{pkr} is the numbe of public key replacements made and S represents the time taken for the calculations performed by the challenger after the adversary gives a forgery.

Proof: Let C be given an instance of the CDH problem, (P, aP, bP). Suppose there exists a type I adversary, who is capable of breaking the signature scheme above, then C's aim is to find the value of abP .

Setup: The challenger C must set up the system exactly as given in the scheme. C sets $P_3 = aP$ implicitly setting MSK as a, where a is unknown to C. C then chooses seven hash functions, $\mathbb{H}_i$, where $i = 1, 2..6$, along with $\widehat{\mathbb{H}_3}$ and models them as random oracles. C chooses a random value α and sets it as the time quantam.Also C maintains a list l_i for each hash function to maintain consistency. C also maintains l_{id} for storing all the keys. Each entry of the l_{id} is of the form, $< ID, FPK, PSK, USK, FSK, t, X_i >$, where the bit X_i is used to determine wheter the public key has been replaced or not.

Training Phase: In this phase the adversary A_1 makes use of all the oracles provided by C. The system is simulated in such a way that A_1 cannot differentiate between a real and a simulated system that is provided by C.

Choosing the target identity: In the oracle $O_{\mathbb{H}_1(ID_i,(P_2)_j)}$. The adversary asks q_{h_1} queries and expects a response from the challenger for each of them. Since the adversary can query on the same ID and different $(P_2)_j$'s, the number of distinct identities queried is different from q_{h_1}. Let that number be q_{id}. $1 \leq q_{id} \leq q_{h_1}$. The challenger uses a biased coin, with probability of heads as p. We define the value of p later. For each identity queried, the challenger tosses a coin, and sets it as a target identity if the outcome is a head. i.e each identity has a probabilitiy of p of being a target identity.

Let's denote ID_{ch} to represent the set of target identities.
Oracle $O_{\mathbb{H}_1}(ID_i, (P_2)_i, t)$:
A list l_{h_1} is maintained of the form $< ID_i, (P_2)_j, t_j, H_j >$. C responds as follows:

- If $< ID_i, (P_2)_j, t_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple, $< ID_i, (P_2)_j, t_j, h_j >$ to the list.

Oracle $O_{\mathbb{H}_2}(ID_i, (P_1)_j)$: A list l_{h_2} is maintained of the form $< ID_i, (P_1)_j, H_j >$. C responds as follows:

- If $< ID_i, (P_1)_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple, $< ID_i, (P_1)_j, h_j >$ to the list.

Oracle $O_{\mathbb{H}_3}(ID_i, (P_1)_j)$: A list l_{h_3} is maintained of the form $< ID_i, (P_1)_j, H_j, x_j >$. C responds as follows:

- If $< ID_i, (P_1)_j >$ already exists in the list then respond with value h_j from the list.
- Else,
  If $ID \notin ID_{ch}$,choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j P$. Return h_j and add the tuple, $< ID_i, (P_1)_j, h_j, x_j >$ to the list.

  If $ID \in ID_{ch}$, choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j bP$. Return h_j and add the tuple, $< ID_i, (P_1)_j, h_j, x_j >$ to the list.

Oracle $O_{\widehat{\mathbb{H}_3}}(ID_i, (P_2)_j, t_j)$: A list $l_{\widehat{h_3}}$ is maintained of the form $< ID_i, (P_2)_j, H_j, t_j, x_j >$. C responds as follows:

- If $< ID_i, (P_2)_j, t_j >$ already exists in the list then respond with value h_j from the list.
- Else,
  If $ID \notin ID_{ch}$,choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j P$. Return h_j and add the tuple, $< ID_i, (P_2)_j, t_j, h_j, x_j >$ to the list.

  If $ID \in ID_{ch}$, choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j bP$. Return h_j and add the tuple, $< ID_i, (P_2)_j, t_j, h_j, x_j >$ to the list.

Oracle $O_{\mathbb{H}_4}(k_j)$: A list l_{h_4} is maintained of the form $< k_j, h_j, x_j >$. C responds as follows:

- If $< k_j >$ already exists in the list then respond with value h_j from the list.
- Else,choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j bP$. Return h_j and add the tuple, $< k_j, h_j, x_j >$ to the list.

Oracle $O_{\mathbb{H}_5}(M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j)$:
A list l_{h_5} is maintained of the form $< M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j, H_j >$. C responds as follows:

- If $< M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple,
  $< M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j, h_j >$ to the list.

Oracle $O_{\mathbb{H}_6}(A_j, B_j, C_j, D_j, E_j, F_j)$: Here, $A_j, B_j, ..F_j$ are some elements in $\mathbb{G}$. A list l_{h_6} is maintained of the form $< A_j, B_j, C_j, D_j, E_j, F_j, h_j >$. C responds as follows:

- If $< A_j, B_j, C_j, D_j, E_j, F_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple,
  $< A_j, B_j, C_j, D_j, E_j, F_j, h_j >$ to the list.

Oracle Partial Extract: C responds as follows:

- If values corresponding to ID_i for the start of the time interval t already exists on the list l_{id}, then return (d_i, t) as PSK and $(P_2, \widehat{P_2}, s_1, c_1, t)$ as PPK from the list
- Else,
  If $ID \notin ID_{ch}$
  Choose $d_i, q_i \in_R \mathbb{Z}_q^*$. Compute $sP = d_i P - q_i P_3$. Compute $P_2 = sP$. Retrieve $\widehat{x}_{j3}$ from oracle corresponding to H_3 and set $\widehat{P_2} = \widehat{x}_{j3} sP$. Then set, $\mathbb{H}_1(ID_i, P_2, t) = q_i$. Add these values to l_{h_1}. Choose a random $k_1 \in_R \mathbb{Z}_q^*$. Compute $u = k_1 P$ and $v = k_1 \widehat{\mathbb{H}_3}(ID_i, P_2, t)$. Choose c_1 as $\mathbb{H}_6(u, v, P_2, \widehat{P_2}, P, \widehat{\mathbb{H}_3}(ID_i, P_2, t))$. Compute $s_1 = k_1 + c_1 s$. Output (d_i, t) as the PSK and

$(P_2, \widehat{P_2}, s_1, c_1, t)$ as PPK. Add these values to the list l_{id} in the entry corresponding to ID_i.

If $ID \in ID_{ch}$, abort.

Lemma 1: The above oracle outputs valid PSK and PPK

Proof: It can be observed that the outputs given by the oracle, satisfy the condition for a valid PPK, PSK. (They satisfy the key sanity check for user verification given earlier)

Oracle Public Key Generation: Challenger responds as follows:

- If values corresponding to ID_i for the start of the time interval t already exists on the list, then return $< P_1, P_2, \widehat{P_1}, \widehat{P_2}, s_1, c_1, s_2, c_2, t >$ from the list.
- Else,
  If $(P_2, \widehat{P_2}, s_1, c_1, t)$ are already in the list l_{id}, in the entry corresponding to ID, retrieve them. Else run the partial key extract oracle and retrieve those values.

  Choose $t_A \in_R \mathbb{Z}_q^*$. Set $P_1 = t_A P$. Query the oracle $\mathbb{H}_3$ on (ID, P_1) and retrieve its value. Compute $\widehat{P_1} = t_A \mathbb{H}_3$. Choose a random $k_2 \in_R \mathbb{Z}_q^*$.
  Compute $u = k_2 P$ and $v = k_2 \widehat{\mathbb{H}_3}(ID_i, P_1, t)$.
  Choose c_2 as $\mathbb{H}_6(u, v, P_1, \widehat{P_1}, P, \widehat{\mathbb{H}_3}(ID_i, P_1, t))$.
  Compute $s_2 = k_2 + c_2 t_A$. Output $(P_1, \widehat{P_1}, P_2, \widehat{P_2}, s_1, c_1, s_2, c_2, t)$ as the full public key. Add these values and t_A to the list l_{id} in the entry corresponding to ID_i and set $x_i = 0$.

Lemma 2: The above oracle for public key generation outputs a valid full public key.

Proof: It can be observed that the output generated by the oracle passes the key sanity check for user verification mentioned in the scheme. Hence, the oracle generates valid public keys.

Oracle Full Private Key :Challenger responds as follows:

- If values corresponding to ID_i for the start of the time interval t already exists on the list, then return $< n_A, t_A, t >$ from the list.
- Else,
  If $ID \notin ID_{ch}$
  If d_A is already in the list l_{id}, in the entry corresponding to ID, retrieve them.
  Else run the partial key extract oracle and retrieve that value.
  If t_A is already in the list l_{id}, in the entry corresponding to ID, retrieve them.
  Else run the public key generation oracle and retrieve that value.
  Compute $n_A = d_A + t_A \mathbb{H}_2(ID, P_1)$. Output $< n_A, t_A, t >$ as the full private key and add them to the list l_{id}.

  If $ID \in ID_{ch}$, abort.

Oracle Public Key Replace: The adversary sends the
value $< ID, P_1, P_2, \widehat{P_1}, \widehat{P_2}, s_1, c_1, s_2, c_2, t >$ to the challenger C. The challenger runs the public key verification test. If the test succeds it adds these values to the list in the entry corresponding to ID and sets $x_i = 1$ to indicate that the public key has been replaced. Further signatures for this identity use this value of the public key.

Oracle Signature: Given a value of M,ID and a time instant $t' \in (t, t + \alpha)$ by the adversary, the challenger does the following:

- Compute $N_A = P_2 + \mathbb{H}_1(ID, P_2, t)P_3 + \mathbb{H}_2(ID, P_1)P_1$.
- Choose $c, v, \alpha \in_R \mathbb{Z}_q^*$.
- Compute $Z_3 = vP - cN_A$.

- Set $\alpha P = \mathbb{H}_4(Z_3)$ and add $< Z_3, \alpha P, \alpha >$ to the list l_{h_4}
- Compute $Z_1 = \alpha N_A, Z_2 = \alpha Z_3$.
- Set $c = \mathbb{H}_5(M, ID_A, Z_1, Z_2, Z_3, P_1, P_2, t')$ and add it to the list l_{h_4}.
- Output $< Z_1, v, c, t' >$ as the signature

Lemma 3: The above signature oracle produces a valid signature for any valid public key.

Proof: It can be easily observed that the signature produced by the oracle passes the verification given in the scheme.

Forgery: Suppose the adversary outputs a forgery $\sigma^* = (Z_1^*, v^*, c^*, t^{'*})$.
Let $t^{'*}$ belong to a time interval $(t^*, t^* + \alpha)$.
The challenger aborts if the forgery is not for an identity that is within the set of target identities ID_{ch}.

The challenger first checks that the signature is a valid one and passes the verification test.

The challenger computes the solution to the hard problem as follows:

- Compute N_A for the target identity;

$$N_A = P_2 + \mathbb{H}_1(ID, P_2, t^*)P_3 + \mathbb{H}_2(ID, P_1)P_1$$

- Compute $Z_3^* = v^*P - c^*N_A$
- Retrieve x_{j4} from the $\mathbb{H}_4$ oracle on input Z_3^*
- Compute x_{j3} from the $\mathbb{H}_3$ oracle on input (ID^*, P_1^*)
- Compute $\widehat{x}_{j3}$ from the $\widehat{\mathbb{H}_3}$ oracle on input $(ID^*, \widehat{P_1^*}, t^*)$
- Retrieve $h_1 = \mathbb{H}_1(ID, P_2, t^*)$
- Retrieve $h_2 = \mathbb{H}_2(ID, P_1)$
- Compute $\Delta = h_1^{-1}\left(x_{j4}^{-1}Z_1^* - x_{j3}^{-1}h_2\widehat{P_1} - \widehat{x_{j3}}^{-1}\widehat{P_2}\right)$.
- C returns

$$\epsilon' \geq \epsilon\left[\frac{1}{q_{fse}}\left[\frac{q_{fse}-1}{q_{fse}}\right]^{q_{fse}}\right]$$

Δ as the solution to the hard problem.

Lemma 4: The value of Δ computed above equals abP.

Proof:

- $Z_1^* = x_{j4}bP\,(s + ah_1 + t_Ah_2)$
- $x_{j3}^{-1}h_2\widehat{P_1} = t_Ah_2bP$
- $\widehat{x_{j3}}^{-1}\widehat{P_2} = sbP$

Therefore, $Z_1^* - x_{j3}^{-1}h_2\widehat{P_1} - \widehat{x_{j3}}^{-1}\widehat{P_2} = h_1abP$.

Hence, $\Delta = abP$.

Probability Analysis:

The challenger fails only if any of the following events occur:

- E_1: The adversary returns a forgery for $ID \notin ID_{ch}$.
- E_2: An invalid public key replacement by the adversary was not detected.
- E_3: The adversary queries partial key for an identity $ID \in ID_{ch}$.

- E_4: The adversary queries full private key for an identity $ID \in ID_{ch}$.

$Pr[E_1] = (1-p)$

$Pr[E_2] = \left(\frac{1}{q}\right)$

$Pr[E_3] = 1-(1-p)^{q_{pe}}$

$Pr[E_4] = 1-(1-p)^{q_{fse}}$

Therefore, the probability of the challenger being successful is atleast $Pr[\neg(E_1 \vee E_2 \vee E_3 \vee E_4)]$. And the advantage of the adversary is ϵ.

Thus,

$$\epsilon' \geq \epsilon\{p.(1-p)^{q_{pe}}(1-p)^{q_{fse}}(1-\frac{1}{q})\}$$

Let $X = p.(1-p)^{q_{fse}+q_{pe}}$. X attains maximum for $p_{max} = \frac{1}{q_{fse}+q_{pe}+1}$.
Therefore, the value of p chosen by the adversary is $p_{max} = \frac{1}{q_{fse}+q_{pe}+1}$
And the advantage of the adversary is

$$\epsilon' \geq \epsilon \left[\frac{1}{q_{fse}+q_{pe}+1}\left[\frac{q_{fse}+q_{pe}}{q_{fse}+q_{pe}+1}\right]^{q_{fse}+q_{pe}}.(1-\frac{1}{q})\right]$$

It can be observed that $t_{ch} = S + t_{adv} + (q_1+q_2+q_3+\widehat{q_3}+q_4+q_5+q_6+q_{psq}+q_{fsq}+q_{fpq}+q_{sq}+q_{pkr})O(1)$. where $O(1)$ captures the time taken for the scalar and group operations performed in the course of each query, and the time taken for the calculations made after the forgery is captured in S.

4.2 Proof for Type II Adversary

Theorem 2: If there exists an adversary A_{II} that can forge a signature for the above scheme with probability ϵ in time t_{adv}, then there exists a challenger C who can solve the CDH problem with probability atleast ϵ' in time t_{ch} , such that

$$\epsilon' \geq \epsilon \left[\frac{1}{q_{fse}+1}\left[\frac{q_{fse}}{q_{fse}+1}\right]^{q_{fse}}\right]$$

$t_{ch} = S + t_{adv} + (q_1+q_2+q_3+\widehat{q_3}+q_4+q_5+q_6+q_{fsq}+q_{fpq}+q_{sq})O(1)$.
q_{id} = number of distinct identities queried by the adversary, q = is the order of the group $\mathbb{G}$ in which the hard problem can be solved by adversary to break the system, q_{fse} = number of full secret key extracts, q_i = number of queries to the H_i hash oracle(where $i = 1, 2..6$), $\widehat{q_3}$ = number of queries to the $\widehat{H_i}$, q_{fsq} = number of full secret key queries, q_{fpq} = number of full public key queries , q_{sq} = number of signature queries and S represents the time taken for the calculations performed by the challenger after the adversary gives a forgery.

Proof:

Let C be given an instance of the CDH problem, (P, aP, bP). Suppose there exists a type II adversary, who is capable of breaking the signature scheme above, then C's aim is to find the value of abP .

Setup: The challenger C must set up the system exactly as given in the scheme. C chooses a random $x \in_R \mathbb{Z}_q$. C then chooses seven hash functions, $\mathbb{H}_i$, where $i = 1, 2..6$, along with $\widehat{\mathbb{H}_3}$ and models them as random oracles.C chooses a random value α and sets it as the time quantam. Also C maintains a list l_i for each hash function to maintain consistency. C also maintains l_{id} for storing all the keys. Each entry of the l_{id} is of the form, $< ID, FPK, PSK, USK, FSK, t >$.

Training Phase: In this phase the adversary A_{II} , makes use of all the oracles provided by C. The system is simulated in such a way that A_{II} cannot differentiate between a real and a simulated system that is provided by C.

Choosing the target identity: In the oracle $O_{\mathbb{H}_1(ID_i,(P_2)_j)}$. The adversary asks q_{h_1} queries and expects a response from the challenger for each of them. Since the adversary can query on the same ID and different $(P_2)_j$'s, the number of distinct identities queried is different from q_{h_1}. Let that number be q_{id}. $1 \leq q_{id} \leq q_{h_1}$. The challenger uses a biased coin, with probability of heads as p. We define the value of p later. For each identity queried, the challenger tosses a coin, and sets it as a target identity if the outcome is a head. i.e each identity has a probabilitiy of p of being a target identity.

Let's denote ID_{ch} to represent the set of target identities.

Oracle $O_{\mathbb{H}_1}(ID_i, (P_2)_j, t_j)$: A list l_{h_1} is maintained of the form $< ID_i, (P_2)_j, t_j, H_j >$. C responds as follows:

- If $< ID_i, (P_2)_j, t_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple, $< ID_i, (P_2)_j, t_j, h_j >$ to the list.

Oracle $O_{\mathbb{H}_2}(ID_i, (P_1)_j)$: A list l_{h_2} is maintained of the form $< ID_i, (P_1)_j, H_j >$. C responds as follows:

- If $< ID_i, (P_1)_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple, $< ID_i, (P_1)_j, h_j >$ to the list.

Oracle $O_{\mathbb{H}_3}(ID_i, (P_1)_j)$: A list l_{h_3} is maintained of the form $< ID_i, (P_1)_j, H_j, x_j >$. C responds as follows:

- If $< ID_i, (P_1)_j >$ already exists in the list then respond with value h_j from the list.
- Else,
  Choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j P$. Return h_j and add the tuple, $< ID_i, (P_1)_j, h_j, x_j >$ to the list.

Oracle $O_{\widehat{\mathbb{H}_3}}(ID_i, (P_2)_j, t_j)$: A list $l_{\widehat{h_3}}$ is maintained of the form $< ID_i, (P_2)_j, t_j, H_j, x_j >$. C responds as follows:

- If $< ID_i, (P_2)_j, t_j >$ already exists in the list then respond with value h_j from the list.
- Else,
  If $ID \notin ID_{ch}$,choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j P$. Return h_j and add the tuple, $< ID_i, (P_2)_j, t_j, h_j, x_j >$ to the list.

  If $ID \in ID_{ch}$, choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j bP$. Return h_j and add the tuple, $< ID_i, (P_2)_j, t_j, h_j, x_j >$ to the list.

Oracle $O_{\mathbb{H}_4}(k_j)$: A list l_{h_4} is maintained of the form $< k_j, H_j, x_j >$. C responds as follows:

- If $< k_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j bP$. Return h_j and add the tuple, $< k_j, h_j, x_j >$ to the list.

Oracle $O_{\mathbb{H}_5}(M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j)$:
A list l_{h_5} is maintained of the form $< M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j, H_j >$. C responds as follows:

- If $< M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple,
  $< M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j, h_j >$ to the list.

Oracle $O_{\mathbb{H}_6}(A_j, B_j, C_j, D_j, E_j, F_j)$: Here, $A_j, B_j, ..F_j$ are some elements in $\mathbb{G}$. A list l_{h_6} is maintained of the form $< A_j, B_j, C_j, D_j, E_j, F_j, h_j >$. C responds as follows:

- If $< A_j, B_j, C_j, D_j, E_j, F_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple,
  $< A_j, B_j, C_j, D_j, E_j, F_j, h_j >$ to the list.

Oracle Public Key Generation: Challenger responds as follows:

- If values corresponding to ID_i for the start of the time interval t already exists on the list, then return $< P_1, P_2, \widehat{P_1}, \widehat{P_2}, s_1, c_1, s_2, c_2, t >$ from the list.
- If $ID \notin ID_{ch}$
  If $(P_2, \widehat{P_2}, s_1, c_1, t)$ are already in the list l_{id}, in the entry corresponding to ID, retrieve them. Else, query the partial extract oracle and retrieve them. Choose $t_A \in_R \mathbb{Z}_q^*$. Set $P_1 = t_A P$. Query the oracle $\mathbb{H}_3$ on (ID, P_1) and retrieve its value. Compute $\widehat{P_1} = t_A \mathbb{H}_3$. Choose a random $k_2 \in_R \mathbb{Z}_q^*$.
  Compute $u = k_2 P$ and $v = k_2 \widehat{\mathbb{H}_3}(ID_i, P_1, t)$
  Choose c_2 as $\mathbb{H}_6(u, v, P_1, \widehat{P_1}, P, \widehat{\mathbb{H}_3}(ID_i, P_1, t))$
  Compute $s_2 = k_2 + c_2 t_A$ Output $(P_1, \widehat{P_1}, P_2, \widehat{P_2}, s_1, c_1, s_2, c_2, t)$ as the full public key. Add these values and t_A to the list l_{id} in the entry corresponding to ID_i .

- If $ID \in ID_{ch}$
  If $(P_2, \widehat{P_2}, s_1, c_1, t)$ are already in the list l_{id}, in the entry corresponding to ID, retrieve them. Else, query the partial extract oracle and retrieve them.

  Set $P_1 = aP$. Retrieve x_{3j} from the $\mathbb{H}_3(ID, P_1)$. Set $\widehat{P_1} = x_{3j} P_1$. Choose a random $k_2 \in_R \mathbb{Z}_q^*$.
  Compute $u = k_2 P$ and $v = k_2 \widehat{\mathbb{H}_3}(ID_i, P_1, t)$
  Choose c_2 as $\mathbb{H}_6(u, v, P_1, \widehat{P_1}, P, \widehat{\mathbb{H}_3}(ID_i, P_1, t))$
  Compute $s_2 = k_2 + c_2 t_A$. Output $< P_1, P_2, \widehat{P_1}, \widehat{P_2}, s_1, c_1, s_2, c_2, t >$ as the public key and add it to the list l_{id}.

Lemma 2: The above oracle for public key generation outputs a valid full public key.

Proof: It can be observed that the output generated by the oracle passes the key sanity check for public verification mentioned in the scheme. Hence, the oracle generates valid public keys.

Oracle Full Private Key :Challenger responds as follows:

- If values corresponding to ID_i for the start of the time interval t already exists on the list, then return $< n_A, t_A, t >$ from the list.
- Else,
  If $ID \notin ID_{ch}$
  If d_A is already in the list l_{id}, in the entry corresponding to ID, retrieve them.
  Else run the partial key extract oracle and retrieve that value.
  If t_A is already in the list l_{id}, in the entry corresponding to ID, retrieve them.
  Else run the public key generation oracle and retrieve that value.
  Compute $n_A = d_A + t_A \mathbb{H}_2(ID, P_1)$. Output $< n_A, t_A, t >$ as the full private key and add them to the list l_{id}.

  If $ID \in ID_{ch}$, abort.

Oracle Signature: Given a value of M,ID and a time instant $t' \in (t, t + \alpha)$ by the adversary, the challenger does the following:

- Compute $N_A = P_2 + \mathbb{H}_1(ID, P_2, t) P_3 + \mathbb{H}_2(ID, P_1) P_1$.
- Choose $c, v, \alpha \in_R \mathbb{Z}_q^*$.
- Compute $Z_3 = vP - cN_A$.
- Set $\alpha P = \mathbb{H}_4(Z_3)$ and add $< Z_3, \alpha P, \alpha >$ to the list l_{h_4}
- Compute $Z_1 = \alpha N_A, Z_2 = \alpha Z_3$.
- Set $c = \mathbb{H}_5(M, ID_A, Z_1, Z_2, Z_3, P_1, P_2, t')$ and add it to the list l_{h_4}.
- Output $< Z_1, v, c, t' >$ as the signature

Lemma 3: The above signature oracle produces a valid signature for any valid public key.

Proof: It can be easily observed that the signature produced by the oracle passes the verification given in the scheme.

Forgery: Suppose the adversary outputs a forgery $\sigma^* = (Z_1^*, v^*, c^*, t^{'*})$.
Let $t^{'}$ belong to a time interval $(t, t+\alpha)$. The challenger aborts if its not for one of the target identities in the set ID_{ch}.

The challenger first checks that the signature is a valid one and passes the verification test.

The challenger computes the solution to the hard problem as follows:

- Compute N_A for the target identity;

$$N_A = P_2 + \mathbb{H}_1(ID, P_2, t^*)P_3 + \mathbb{H}_2(ID, P_1)P_1$$

- Compute $Z_3^* = v^*P - c^*N_A$
- Retrieve x_{j4} from the $\mathbb{H}_4$ oracle on input Z_3^*
- Compute $\widehat{x}_{j3}$ from the $\widehat{\mathbb{H}_3}$ oracle on input $(ID^*, \widehat{P_1^*}, t^*)$
- Retrieve $h_1 = \mathbb{H}_1(ID, P_2, t^*)$
- Retrieve $h_2 = \mathbb{H}_2(ID, P_1)$
- Compute $\Delta = h_2^{-1}\left(x_{j4}^{-1}Z_1^* - xh_1bP - \widehat{x_{j3}}^{-1}\widehat{P_2}\right)$.
- C returns Δ as the solution to the hard problem.

Lemma 4: The value of Δ computed above equals abP.

Proof:

- $Z_1^* = x_{j4}bP\,(s + ah_2 + xh_1)$
- $\widehat{x_{j3}}^{-1}\widehat{P_2} = sbP$

Therefore, $Z_1^* - xh_1bP - \widehat{x_{j3}}^{-1}\widehat{P_2} = h_2abP$.

Hence, $\Delta = abP$.

Probability Analysis:

The challenger fails only if any of the following events occur:

- E_1: The adversary returns a forgery for $ID \notin ID_{ch}$.
- E_2: The adversary queries full private key for an identity $ID \in ID_{ch}$.

$Pr[E_1] = (1-p)$

$Pr[E_2] = 1 - (1-p)^{q_{fse}}$

Therefore, the probability of the challenger being successful is atleast $Pr[\neg(E_1 \vee E_2)]$. And the advantage of the adversary is ϵ. Thus,

$$\epsilon' \geq \epsilon\{p.(1-p)^{q_{fse}}\}$$

Let $X = p.(1-p)^{q_{fse}}$. X attains maximum for $p_{max} = \frac{1}{q_{fse}+1}$.
Therefore, the value of p chosen by the adversary is $p_{max} = \frac{1}{q_{fse}+1}$
And the advantage of the adversary is

$$\epsilon' \geq \epsilon\left[\frac{1}{q_{fse}+1}\left[\frac{q_{fse}}{q_{fse}+1}\right]^{q_{fse}}\right]$$

It can be observed that $t_{ch} = S + t_{adv} + (q_1 + q_2 + q_3 + \widehat{q_3} + q_4 + q_5 + q_6 + q_{fsq} + q_{fpq} + q_{sq})O(1)$. where $O(1)$ captures the time taken for the scalar and group operations performed in the course of each query, and the time taken for the calculations made after the forgery is captured in S.

4.3 Proof for Type III Adversary

Theorem 3: If there exists an adversary A_{III} that can forge a signature for the above scheme with probability ϵ in time t_{adv}, then there exists a challenger C who can solve the CDH problem with probability atleast ϵ' in time t_{ch} such that

$$\epsilon' \geq \left[1 - \frac{1}{q}\right] \epsilon$$

$t_{ch} = S + t_{adv} + (q_1 + q_2 + q_3 + \widehat{q_3} + q_4 + q_5 + q_6 + q_{psq} + q_{fsq} + q_{fpq} + q_{sq})O(1)$.
q_{id} = number of distinct identities queried by the adversary, q = is the order of the group $\mathbb{G}$ in which the hard problem can be solved by adversary to break the system, q_{pe} = number of partial extract queried, q_{fse} = number of full secret key extracts, q_i = number of queries to the H_i hash oracle(where $i = 1, 2..6$), $\widehat{q_3}$ = number of queries to the $\widehat{H_i}$, q_{psq} = number of partial extract queries, q_{fsq} = number of full secret key queries, q_{fpq} = number of full public key queries , q_{sq} = number of signature queries and S represents the time taken for the calculations performed by the challenger after the adversary gives a forgery.

Proof: Let C be given an instance of the CDH problem, (P, aP, bP). Suppose there exists a type III adversary, who is capable of breaking the signature scheme above, then C's aim is to find the value of abP .

Setup: The challenger C must set up the system exactly as given in the scheme. C sets $P_3 = aP$ implicitly setting MSK is a, where a is unknown to C. C then chooses seven hash functions, $\mathbb{H}_i$, where $i = 1, 2..6$, along with $\widehat{\mathbb{H}_3}$ and models them as random oracles.C chooses a random value α and sets it as the time quantam. Also C maintains a list l_i for each hash function to maintain consistency. C also maintains l_{id} for storing all the keys. Each entry of the l_{id} is of the form, $< ID, FPK, PSK, USK, FSK, t, X_i >$, where the bit X_i is used to determine wheter the public key has been replaced or not.
Let's say that the adversary was revoked at the time interval beginning at $t^{\#}$
Training Phase: In this phase the adversary A_{III} , makes use of all the oracles provided by C. The system is simulated in such a way that A_{III} cannot differentiate between a real and a simulated system that is provided by C.

Note that Only for the hash oracles, the adversary has the right to query with time instants even after the beginning of the time period when he was revoked(i.e greater than $t^{\#}$)

Choosing the target identity: In the oracle $O_{\mathbb{H}_1(ID_i,(P_2)_j)}$. The adversary asks q_{h_1} queries and expects a response from the challenger for each of them. Since the adversary can query on the same ID and different $(P_2)_j$'s, the number of distinct identities queried is different from q_{h_1}. Let that number be q_{id}. $1 \leq q_{id} \leq q_{h_1}$. The challenger uses a biased coin, with probability of heads as p. We define the value of p later. For each identity queried, the challenger tosses a coin, and sets it as a target identity if the outcome is a head. i.e each identity has a probabilitiy of p of being a target identity.

Let's denote ID_{ch} to represent the set of target identities.

Oracle $O_{\mathbb{H}_1}(ID_i, (P_2)_i, t)$: A list l_{h_1} is maintained of the form $< ID_i, (P_2)_j, t_j, H_j >$. C responds as follows:

- If $< ID_i, (P_2)_j, t_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple, $< ID_i, (P_2)_j, t_j, h_j >$ to the list.

Oracle $O_{\mathbb{H}_2}(ID_i, (P_1)_j)$: A list l_{h_2} is maintained of the form $< ID_i, (P_1)_j, H_j >$. C responds as follows:

- If $< ID_i, (P_1)_j >$ already exists in the list then respond with value h_j from the list.

- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple, $< ID_i, (P_1)_j, h_j >$ to the list.

Oracle $O_{\mathbb{H}_3}(ID_i, (P_1)_j)$: A list l_{h_3} is maintained of the form $< ID_i, (P_1)_j, H_j, x_j >$. C responds as follows:

- If $< ID_i, (P_1)_j >$ already exists in the list then respond with value h_j from the list.
- Else,
  If $ID \notin ID_{ch}$,choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j P$. Return h_j and add the tuple, $< ID_i, (P_1)_j, h_j, x_j >$ to the list.

  If $ID \in ID_{ch}$, choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j bP$. Return h_j and add the tuple, $< ID_i, (P_1)_j, h_j, x_j >$ to the list.

Oracle $O_{\widehat{\mathbb{H}_3}}(ID_i, (P_2)_j, t_j)$: A list $l_{\widehat{h_3}}$ is maintained of the form $< ID_i, (P_2)_j, H_j, t_j, x_j >$. C responds as follows:

- If $< ID_i, (P_2)_j, t_j >$ already exists in the list then respond with value h_j from the list.
- Else,
  If $ID \notin ID_{ch}$,choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j P$. Return h_j and add the tuple, $< ID_i, (P_2)_j, t_j, h_j, x_j >$ to the list.

  If $ID \in ID_{ch}$ and $t < t^{\#}$, choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j P$. Return h_j and add the tuple, $< ID_i, (P_2)_j, t_j, h_j, x_j >$ to the list.

  If $ID \in ID_{ch}$ and $t >= t^{\#}$, choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j bP$. Return h_j and add the tuple, $< ID_i, (P_2)_j, t_j, h_j, x_j >$ to the list.

Oracle $O_{\mathbb{H}_4}(k_j)$: A list l_{h_4} is maintained of the form $< k_j, H_j, x_j >$. C responds as follows:

- If $< k_j >$ already exists in the list then respond with value h_j from the list.
- Else,choose a $x_j \in_R \mathbb{Z}_q^*$. Compute $h_j = x_j bP$. Return h_j and add the tuple, $< k_j, h_j, x_j >$ to the list.

Oracle $O_{\mathbb{H}_5}(M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j)$:
A list l_{h_5} is maintained of the form $< M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j, H_j >$. C responds as follows:

- If $< M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple,
  $< M_j, ID_i, (P_1)_j, (P_2)_j, Z_{1j}, Z_{2j}, Z_{3j}, t_j, h_j >$ to the list.

Oracle $O_{\mathbb{H}_6}(A_j, B_j, C_j, D_j, E_j, F_j)$: Here, $A_j, B_j, ..F_j$ are some elements in $\mathbb{G}$. A list l_{h_6} is maintained of the form $< A_j, B_j, C_j, D_j, E_j, F_j, h_j >$. C responds as follows:

- If $< A_j, B_j, C_j, D_j, E_j, F_j >$ already exists in the list then respond with value h_j from the list.
- Else, choose a $h_j \in_R \mathbb{Z}_q^*$. Return h_j and add the tuple,
  $< A_j, B_j, C_j, D_j, E_j, F_j, h_j >$ to the list.

Oracle Partial Extract: C responds as follows:

- If $t >= t^{\#}$, return "empty".
- Else, if values corresponding to ID_i for the start of the time interval t already exists on the list l_{id}, then return (d_i, t) as PSK and $(P_2, \widehat{P_2}, s_1, c_1, t)$ as PPK from the list.
- Else,
  If $ID \notin ID_{ch}$
  Choose $d_i, q_i \in_R \mathbb{Z}_q^*$. Compute $s = d_i - xq_i$. Compute $P_2 = sP$ and $\widehat{P_2} = s\widehat{\mathbb{H}_3}(ID_i, P_2, t)$. Then set, $\mathbb{H}_1(ID_i, P_2, t) = q_i$. Add these values to l_{h_1}. Choose a random $k_1 \in_R \mathbb{Z}_q^*$. Compute $u = k_1 P$ and $v = k_1\widehat{\mathbb{H}_3}(ID_i, P_2, t)$. Choose c_1 as $\mathbb{H}_6(u, v, P_2, \widehat{P_2}, P, \widehat{\mathbb{H}_3}(ID_i, P_2, t))$. Compute $s_1 = k_1 + c_1 s$. Output (d_i, t) as the PSK and $(P_2, \widehat{P_2}, s_1, c_1, t)$ as PPK. Add these values to the list l_{id} in the entry

corresponding to ID_i.

If $ID \in ID_{ch}$
Choose $d_i, q_i \in_R \mathbb{Z}_q^*$. Compute $sP = d_iP - q_i(aP)$. Set $P_2 = sP$ and $\widehat{P_2} = \widehat{x}_{j3}P_2$, where $\widehat{\mathbb{H}}_3 = \widehat{x}_{j3}P$. Then set, $\mathbb{H}_1(ID_i, P_2, t) = q_i$. Add these values to l_{h_1}. Choose a random $k_1 \in_R \mathbb{Z}_q^*$. Compute $u = k_1P$ and $v = k_1\widehat{\mathbb{H}_3}(ID_i, P_2, t)$. Choose c_1 as $\mathbb{H}_6(u, v, P_2, \widehat{P_2}, P, \widehat{\mathbb{H}_3}(ID_i, P_2, t))$. Compute $s_1 = k_1 + c_1 s$. Output (d_i, t) as the PSK and $(P_2, \widehat{P_2}, s_1, c_1, t)$ as PPK. Add these values to the list l_{id} in the entry corresponding to ID_i.

Lemma 1: The above oracle outputs valid PSK and PPK

Proof: It can be observed that the outputs given by the oracle, satisfy the condition for a valid PPK, PSK. (They satisfy the key sanity check for user verification given earlier)

Oracle Public Key Generation: Challenger responds as follows:

- If $t >= t^{\#}$, return "empty".
- Else

  - If values corresponding to ID_i for the start of the time interval t already exists on the list, then return $< P_1, P_2, \widehat{P_1}, \widehat{P_2}, s_1, c_1, s_2, c_2, t >$ from the list.
  - Else,
    If $(P_2, \widehat{P_2}, s_1, c_1, t)$ are already in the list l_{id}, in the entry corresponding to ID, retrieve them. Else run the partial key extract oracle and retrieve those two values.

    Choose $t_A \in_R \mathbb{Z}_q^*$. Set $P_1 = t_AP$. Query the oracle $\mathbb{H}_3$ on (ID, P_1) and retrieve its value. Compute $\widehat{P_1} = t_A\mathbb{H}_3$. Choose a random $k_2 \in_R \mathbb{Z}_q^*$.
    Compute $u = k_2P$ and $v = k_2\widehat{\mathbb{H}_3}(ID_i, P_1, t)$.
    Choose c_2 as $\mathbb{H}_6(u, v, P_1, \widehat{P_1}, P, \widehat{\mathbb{H}_3}(ID_i, P_1, t))$.
    Compute $s_2 = k_2 + c_2t_A$. Output $(P_1, \widehat{P_1}, P_2, \widehat{P_2}, s_1, c_1, s_2, c_2, t)$ as the full public key. Add these values and t_A to the list l_{id} in the entry corresponding to ID_i and set $x_i = 0$.

Lemma 2: The above oracle for public key generation outputs a valid full public key.

Proof: It can be observed that the output generated by the oracle passes the key sanity check for public verification mentioned in the scheme. Hence, the oracle generates valid public keys.

Oracle Full Private Key :Challenger responds as follows:

- If $t >= t^{\#}$, then return "empty".
- Else, if values corresponding to ID_i for the time interval beginning at t already exists on the list, then return $< n_A, t_A, t >$ from the list.
- Else,
  If d_A is already in the list l_{id}, in the entry corresponding to ID, retrieve them.
  Else run the partial key extract oracle and retrieve that value.
  If t_A is already in the list l_{id}, in the entry corresponding to ID, retrieve them.
  Else run the public key generation oracle and retrieve that value.
  Compute $n_A = d_A + t_A\mathbb{H}_2(ID, P_1)$. Output $< n_A, t_A, t >$ as the full private key and add them to the list l_{id}.

Oracle Signature: Given a value of M,ID and a time instant $t' \in (t, t + \alpha)$ by the adversary, the challenger does the following:

- If $t >= t^{\#}$, then return "empty".

- Else,

  - Compute $N_A = P_2 + \mathbb{H}_1(ID, P_2, t)P_3 + \mathbb{H}_2(ID, P_1)P_1$.
  - Choose $c, v, \alpha \in_R \mathbb{Z}_q^*$.
  - Compute $Z_3 = vP - cN_A$.
  - Set $\alpha P = \mathbb{H}_4(Z_3)$ and add $< Z_3, \alpha P, \alpha >$ to the list l_{h_4}
  - Compute $Z_1 = \alpha N_A, Z_2 = \alpha Z_3$.
  - Set $c = \mathbb{H}_5(M, ID_A, Z_1, Z_2, Z_3, P_1, P_2, t')$ and add it to the list l_{h_4}.
  - Output $< Z_1, v, c, t' >$ as the signature

Lemma 3: The above signature oracle produces a valid signature for any valid public key.

Proof: It can be easily observed that the signature produced by the oracle passes the verification given in the scheme.

Forgery: Suppose the adversary outputs a forgery $\sigma^* = (Z_1^*, v^*, c^*, t^{'*})$.
The challenger aborts if it's not for an identity that is within the set of target identities ID_{ch} or if $t^{'*} < t^{\#}$.

Let the time $t^{'*} \in (t^*, t^* + \alpha)$.
The challenger first checks that the signature is a valid one and passes the verification test.
The challenger computes the solution to the hard problem as follows:

- Compute N_A for the target identity;

$$N_A = P_2 + \mathbb{H}_1(ID, P_2, t^*)P_3 + \mathbb{H}_2(ID, P_1)P_1$$

- Compute $Z_3^* = v^*P - c^*N_A$
- Retrieve x_{j4} from the $\mathbb{H}_4$ oracle on input Z_3^*
- Compute x_{j3} from the $\mathbb{H}_3$ oracle on input (ID^*, P_1^*)
- Compute $\widehat{x}_{j3}$ from the $\widehat{\mathbb{H}_3}$ oracle on input $(ID^*, \widehat{P_1^*}, t^*)$
- Retrieve $h_1 = \mathbb{H}_1(ID, P_2, t^*)$
- Retrieve $h_2 = \mathbb{H}_2(ID, P_1)$
- Compute $\Delta = h_1^{-1}\left(x_{j4}^{-1}Z_1^* - x_{j3}^{-1}h_2\widehat{P_1} - \widehat{x_{j3}}^{-1}\widehat{P_2}\right)$.
- C returns Δ as the solution to the hard problem.

Lemma 4: The value of Δ computed above equals abP.

Proof:

- $Z_1^* = x_{j4}bP\,(s + ah_1 + t_Ah_2)$
- $x_{j3}^{-1}h_2\widehat{P_1} = t_Ah_2bP$
- $\widehat{x_{j3}}^{-1}\widehat{P_2} = sbP$

Therefore, $Z_1^* - x_{j3}^{-1}h_2\widehat{P_1} - \widehat{x_{j3}}^{-1}\widehat{P_2} = h_1abP$.

Hence, $\Delta = abP$.

Probability Analysis:

The challenger fails only if any of the following events occur:

- E_1: The adversary returns a forgery for $ID \notin ID_{ch}$.
- E_2: An invalid public key replacement by the adversary was not detected.
- E_3: The adversary returns a forgery for $t^{'*} < t^{\#}$.

Since the adversary knows when he was revoked, making queries to the key oracles after getting revoked makes the challenger just return "empty" rather than aborting, as it will be treated as an inappropriate request by the adversary.

$Pr[E_1] = 1 - p$

$Pr[E_2] = \left(\frac{1}{q}\right)$

$Pr[E_3] = \left(\frac{t^{\#}}{T}\right)$

Where, T denotes the total possible time, and assuming that the time begins at 0.

Now,the total possible time T is close to infinity. Therefore,$Pr[E_3]$ is close to 0 and so,we can safely assume that $\neg Pr[E_3] = 1$.

Alternately, as the adversary knows when he was revoked, we can also argue that according to the game, he shouldn't produce a forgery for a time $t^{'*} < t^{\#}$ so that way also we can rule out event E_3.

Therefore, the probability of the challenger being successful is atleast $Pr[\neg(E_1 \vee E_2 \vee E_3)]$. And the advantage of the adversary is ϵ.

$$\epsilon' \geq \left[p\left(1 - \frac{1}{q}\right)\right]\epsilon$$

Let $X = \left[p\left(1 - \frac{1}{q}\right)\right]$. X attains maximum for $p_{max} = 1$.

Therefore, the value of p chosen by the adversary is $p_{max} = 1$

And the advantage of the adversary is

$$\epsilon' \geq \left[1 - \frac{1}{q}\right]\epsilon$$

It can be observed that $t_{ch} = S + t_{adv} + (q_1 + q_2 + q_3 + \widehat{q_3} + q_4 + q_5 + q_6 + q_{psq} + q_{fsq} + q_{fpq} + q_{sq})O(1)$. where $O(1)$ captures the time taken for the scalar and group operations performed in the course of each query, and the time taken for the calculations made after the forgery is captured in S.

5 Efficiency

We make a comparison of the size of the ciphertext, the computational cost for signing and verification of our scheme with the scheme proposed in [14].

Table 1: Comparison

Scheme	Ciphertext Size	Cost of signing	Cost of verification
Scheme in [14]	$2\|G\|$	$2s_a + 2H + 3g_m$	$4P + 2H + g_a$
Our scheme	$\|G\| + 2\|F\|$	$2s_a + 2H + 3g_m$	$5g_a + 4H$

Where :

$|G|$ represents size of one group element, $|F|$ represents an element of the field Z_q, s_a denotes a scalar addition, H denotes a hash computation, g_m denotes a group exponentation, P denotes a pairing operation and g_a denotes a group addition.

Additionally, along with both the ciphertexts, the time of signing must also be transmitted.

Ciphertext size: The size of an element in the field Z_q is much smaller than the size of an element of the groups under consideration, therefore the size of the ciphertext in our scheme is smaller.

Cost of signing: Equal in both the schemes.

Cost of verification: The pairing operation being highly expensive, outweighs all the other operations. Therefore, our scheme has a lesser cost of verification.

The above results indicate that the proposed scheme is more efficient than the scheme in [14].

6 Conclusion

In this paper, we have presented a revocable certificateless online-offline signature scheme which does not use pairing and proved its secure in the random oracle model using a tight security reduction to the computational Diffie-Hellman problem. Revocability is a very important property which is relevant in real life. Expiry of cheques is a simple example, where keys need to be expired after a specific time interval. This is the time period in which a particular cheque can be encashed. Due to its relevance in the practical world and the limited availability of such schemes in the literature, the proposed scheme is an important research advancement. We have discussed the limitations of the only previously existing time-interval based revocable certificateless signature scheme[14]. We have come up with our own model of a time-interval based revocable certificateless signature scheme. Our scheme also has the added advantage over the previous scheme in the sense that signatures can be produced at any time instant, and we have also given key sanity checks for user verification and public verification. Our scheme is also computationally extremely efficient, and does not use the costly mathematical pairing operation. In addition, it has the property of being an online/offline signature scheme. Online/offline signature schemes are practically very important in the case of low resource devices. Our scheme is more secure and efficient than previously existing online/offline certificateless signature schemes in the literature.

References

[1] Al-Riyami, S. S., and Paterson, K. G. Certificateless public key cryptography. In *ASIACRYPT* (2003), pp. 452–473.

[2] Baek, J., Safavi-Naini, R., and Susilo, W. Certificateless public key encryption without pairing. In *Proceedings of the 8th international conference on Information Security* (Berlin, Heidelberg, 2005), ISC'05, Springer-Verlag, pp. 134–148.

[3] Chevallier-mames, B., Group, C. S., Vigie, L., Jujubier, A. D., Iv, Z. A., and École Normale Supérieure. An efficient cdh-based signature scheme with a tight security reduction. In *Advances in Cryptology CRYPTO 2005, to appear in Lecture Notes in Computer Science* (2005), SpringerVerlag, pp. 511–526.

[4] Coron, J.-S. On the exact security of full domain hash. In *CRYPTO* (2000), M. Bellare, Ed., vol. 1880 of *Lecture Notes in Computer Science*, Springer, pp. 229–235.

[5] Even, S., Goldreich, O., and Micali, S. On-line/off-line digital signatures. *J. Cryptology 9*, 1 (1996), 35–67.

[6] Ge, A., Chen, S., and Huang, X. A concrete certificateless signature scheme without pairings. In *Proceedings of the 2009 International Conference on Multimedia Information Networking and Security - Volume 02* (Washington, DC, USA, 2009), MINES '09, IEEE Computer Society, pp. 374–377.

[7] Goh, E.-J., and Jarecki, S. A signature scheme as secure as the diffie-hellman problem. In *EUROCRYPT* (2003), pp. 401–415.

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

[9] Ju, H. S., Kim, D. Y., Lee, D. H., Lim, J., and Chun, K. Efficient revocation of security capability in certificateless public key cryptography. In *Proceedings of the 9th international conference on Knowledge-Based Intelligent Information and Engineering Systems - Volume Part II* (Berlin, Heidelberg, 2005), KES'05, Springer-Verlag, pp. 453–459.

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

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

[12] Selvi, S. S. D., Vivek, S. S., Pradhan, V. K., and Rangan, C. P. Efficient Certificateless Online/Offline Signature with tight security. *Journal of Internet Services and Information Security (JISIS) 3*, 1/2 (February 2013), 115–137.

[13] Shamir, A. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology* (New York, NY, USA, 1985), Springer-Verlag New York, Inc., pp. 47–53.

[14] Sun, Y., Zhang, F., Shen, L., and Deng, R. H. A revocable certificateless signature scheme. Cryptology ePrint Archive, Report 2013/053, 2013. `http://eprint.iacr.org/`.

[15] Vivek, S. S., Selvi, S. S. D., and Rangan, C. P. Compact stateful encryption schemes with ciphertext verifiability. In *IWSEC* (2012), pp. 87–104.

[16] Xu, Z., Liu, X., Zhang, G., He, W., Dai, G., and Shu, W. A certificateless signature scheme for mobile wireless cyber-physical systems. In *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems Workshops* (Washington, DC, USA, 2008), ICDCSW '08, IEEE Computer Society, pp. 489–494.

[17] Yap, W.-S., Chow, S. S., Heng, S.-H., and Goi, B.-M. Security mediated certificateless signatures. In *Proceedings of the 5th international conference on Applied Cryptography and Network Security* (Berlin, Heidelberg, 2007), ACNS '07, Springer-Verlag, pp. 459–477.

[18] Yap, W.-S., Heng, S.-H., and Goi, B.-M. An efficient certificateless signature scheme. In *Proceedings of the 2006 international conference on Emerging Directions in Embedded and Ubiquitous Computing* (Berlin, Heidelberg, 2006), EUC'06, Springer-Verlag, pp. 322–331.

[19] Zhang, F., Li, S., Miao, S., Mu, Y., Susilo, W., and Huang, X. Cryptanalysis on two certificateless signature schemes. In *International Journal of Computers Communications and Control* (2010).