Efficient and Provably-Secure Certificateless Short Signature Scheme from Bilinear Pairings

Hongzhen Du[1,2] and Qiaoyan Wen[1]

[1] School of Science, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2] Mathematics Department, Baoji University of Arts and Sciences, Baoji 721007, China
duhongzhen@gmail.com

Abstract. In this paper, we present a certificateless signature (CLS) scheme that is proved to be secure in the random oracle model under the hardness assumptions of k-CAA and Inv-CDHP. Our scheme upholds all desirable properties of previous CLS schemes, and requires general cryptographic hash functions instead of the MapToPoint hash function which is inefficient and probabilistic. Furthermore, our scheme requires less computation cost and significantly more efficient than all known CLS schemes, and the size of signatures generated by our scheme is approximate 160 bits, which is the shortest certificateless signatures so far. So it can be used widely, especially in low-bandwidth communication environments.

Keywords: certificateless signature, k-CAA, bilinear pairings

1 Introduction

In a traditional public key cryptosystem (PKC), anyone who wants to send messages to others must obtain their authorized certificates that contain the public key. However, this requirement brings lots of certificate management problems in practice. In order to avoid the problems and the cost of distributing the public keys, Shamir [1] firstly introduced the concept of identity based public key cryptosystem in 1984, which allows a user to use his identity information such as name, Email address, IP address or telephone number *et al* as his own public key . It means that there is no need for a user to keep a public key directory or obtain other users' certificates before communication. However, in an ID-based public key cryptosystem, there inherently exists a drawback called private key escrow problem. Since this cryptosystem involves a Key Generation Center (KGC), which is responsible for generating user's private key based on his identity, that is, the private key of a user is known to the KGC. As a result, the KGC can literally decrypt any ciphertext and forge any user's signature on any message. To avoid the inherent key escrow problem in ID-based public key cryptosystem, Al-Riyami and Paterson [2] introduced a new approach called certificateless public key cryptography (CLPKC) in 2003. The CLPKC is intermediate between traditional PKC and ID-based cryptosystem. In a certificateless cryptosystem, a user's private key is not generated by the Key Generation Center (KGC) alone. Instead, it consists of partial private key generated by the KGC and some secret value chosen by the user. So, the KGC is unable to obtain the user's private-key. In such a way that the key escrow problem of ID-based cryptosystem can be solved. In addition, certificateless public key cryptosystems are not purely ID-based, and there exists an additional public key for each user. Fortunately, this public key does not need to be certified by any trusted authority since only a user with the valid ID can obtain the partial private key from the KGC, which ensures that the public key can be verified without a certificate.

Following the pioneering work due to Al-Riyami and Paterson in [2], several certificateless signature (CLS) schemes [3, 4, 5, 6, 7] have been proposed. Yum and Lee [3] come up with generic construction of CLS schemes, and their construction leads to good security reduction, but it results in inefficient schemes. Li, Chen and Sun [4] propose another CLS scheme based on bilinear pairings. However, their

scheme is costly, since the verification algorithm requires four expensive pairing computations. Literature [5] also needs four pairing computations in verification algorithm and [6] three pairing computations. Yap, Heng and Goi [7] present an efficient CLS scheme, which the signing algorithm does not require pairing computation and the verification algorithm only needs two bilinear pairing computations. So, it is more efficient than the existing CLS schemes. Unfortunately, Park [8] claims that their scheme [7] is insecure against a key replacement attack.

In the CLS schemes [2, 3, 4, 5, 6, 7], a special hash function called MapToPoint function which is used to map an identity information into a point on elliptic curve is required. However, the hash function is inefficient and probabilistic although there has been much discussion on the construction of such hash algorithm [9, 10], and there is no deterministic polynomial time algorithm for it so far. Therefore, using general cryptographic hash function instead of the MapToPoint function can improve the efficiency of CLS schemes.

At present, many short signatures schemes in public key cryptosystem have been proposed since Boneh, Lynn and Shacham [10] construct a short signature called BLS signature, which is just half the size of the signature in DSA (320-bit) with comparable security. Because of the small size of short signatures, they are needed in environments with stringent bandwidth constraints, such as bar-coded digital signatures on postage stamps. Nevertheless, certificateless signatures generated by [2, 3, 4, 5, 6, 7] have approximately 320-bit size if using an elliptic curve on $F_{3^{97}}$. To our best knowledge, no short CLS schemes have been found so far.

In this paper, we come up with a short CLS scheme that is proved to be secure in the random oracle model under the hardness assumption of k-CAA [11] and Inv-CDH problem. Unlike schemes in [2, 3, 4, 5, 6, 7], our scheme use general cryptographic hash functions, and does not require MapToPoint functions. Furthermore, our scheme requires less computation cost than that of the existing CLS schemes, so it is significantly more efficient than all known CLS schemes. Furthermore, the size of signatures generated by our scheme is reduced to at least half–size compared to all proposed CLS schemes, and is only 154 bits if using an elliptic curve on $F_{3^{97}}$, which is the shortest CLS scheme so far.

The remaining sections are organized as follows: In the next section we give a brief introduction to bilinear pairings and some mathematical theory related to the following schemes. Section 3 provides the framework of CLS schemes. We propose an efficient short CLS scheme, give its efficiency analysis and provide its security proof in the random oracle model in Section 4. Conclusion is drawn in the last section.

2 Preliminaries

In this section, we briefly introduce some mathematical theory related to the following schemes.

2.1 Bilinear Pairings

Let G_1 be a cyclic additive group of prime order q , and G_2 be a cyclic multiplicative group of the same order q . A bilinear pairing is a map $e: G_1 \times G_1 \to G_2$ which satisfies the following properties:

1. Bilinearity

$$e(aP, bQ) = e(P, Q)^{ab} \text{ , where } P, Q \in G_1, a, b \in Z_q^*$$

2. Non-degeneracy

There exists $P, Q \in G_1$ such that $e(P, Q) \neq 1$

3. Computability

There is a computable algorithm to get $e(P, Q)$ for all $P, Q \in G_1$.

As is shown in [12], the modified Tate pairing on a supersingular elliptic curve is such a bilinear pairing.

2.2 k-CAA and Inv-CDHP

Definition1. (k-CAA [11]). For an integer k, and $s \in Z_q^*$, $P \in G_1$. Given $\{ P, sP, e_1, e_2, \cdots, e_k \in Z_q^*$, and $\frac{1}{s+e_1}P, \frac{1}{s+e_2}P, \cdots, \frac{1}{s+e_k}P \}$, to compute $\frac{1}{s+e}P$, where $e \notin \{e_1, e_2, \cdots, e_k\}$. We say that the k-CAA is (t, ε)-hard if for all t-time adversaries *A*, we have

$$Adv_{k-CAA_A} = \Pr\left[\begin{array}{l} A(P, sP, \frac{1}{s+e_1}P, \frac{1}{s+e_2}P, \cdots \frac{1}{s+e_k}P) = \frac{1}{s+e}P \\ \left| s \in Z_q^*, P \in G_1, e_1, e_2, \cdots e_k \in Z_q^*, e \notin \{e_1, e_2, \cdots, e_k\} \right. \end{array} \right] < \varepsilon$$

Until now, k-CAA problem is still hard, which means there is no polynomial time algorithm to solve it with non-negligible probability.

Definition2. (Inv-CDHP) Inverse Computational Diffie-Hellman Problem: For an unknown value $a \in Z_q^*$, given P, aP , to compute $\frac{1}{a}P$.

Inv-CDHP is polynomial time equivalent to CDHP, that is, Inv-CDHP is a hard problem.

3 Framework of Certificateless Signatures

3.1 Definition of CLS

A CLS scheme consists of seven algorithms: Setup, Partial-Private-Key-Extract, Set-Secret-Value, Set-Private-Key, Set-Public-Key, CL-Sign and CL-Verify.

Setup: Taking security parameter k as input and returns the system parameters, *params* and master-key.

Partial-Private-Key-Extract: It takes *params*, master-key and a user's identity ID as inputs. It returns a partial private key d_{ID}.

Set-Secret-Value: Taking as inputs *params* and a user's identity ID, this algorithm generates a secret value *r*.

Set-Private-Key: This algorithm takes *params*, a user's partial private key d_{ID} and his secret value *r*, and outputs the full private key sk_{ID}.

Set-Public-Key: Taking as inputs *params* and a user's secret value *r*, and generates a public key pk_{ID} for that user.

CL-Sign: It takes as inputs *params*, a message m, a user's identity ID, and the user's private key sk_{ID}, and outputs a signature *S*.

CL-Verify: It takes as inputs *params*, a public key pk_{ID} , a message m, a user's identity ID, and a signature *S*, and returns 1 means that the signature is accepted. Otherwise, 0 means rejected.

3.2 Security Model for CLS

In CLS , as defined in [2, 5], there are two types of adversaries with different capabilities, we assume Type 1 Adversary, A_1 acts as a dishonest user while Type 2 Adversary, A_2 acts as malicious key generator centre (KGC):

CLS Type 1 Adversary: Adversary A_1 does not have access to master-key, but A_1 may replace users' public keys.

CLS Type 2 Adversary: Adversary A_2 have access to master-key, but cannot replace any user's public key.

Definition3. Let A_1 and A_2 be a Type 1 Adversary and a Type 2 Adversary, respectively. We consider two games ***Game* 1** and ***Game* 2** where A_1 and A_2 interact with their Challenger in these two games, respectively. We say that a CLS scheme is existentially unforgeable against adaptive chosen message attacks, if the success probability of both A_1 and A_2 is negligible.

Game 1: This is the game where A_1 interacts with its Challenger C:

Setup: The Challenger C takes a security parameter k and runs Setup to generate master Key and *params*, then sends *params* to A_1 . A_1 acts as the following oracle queries:

Hash Queries: A_1 can request the hash values for any input.

Extract Partial Private Key: A_1 is able to ask for the partial private key d_{ID} for any ID except the challenged identity ID. C computes d_{ID} corresponding to ID and returns d_{ID} to A_1 .

Extract Private Key: For any ID except the challenged identity ID, C firstly computes the partial private key d_{ID} and then secret value as well as private key sk_{ID} corresponding to the identity ID and returns it to A_1 .

Request Public Key: A_1 can request the public key for any identity ID. Upon receiving a public key query for any identity ID, C computes the corresponding public key pk_{ID} and sends it to A_1 .

Replace Public Key: For any identity ID, A_1 can pick a new secret value r' and compute the new public pk'_{ID} corresponding to the value r' , and then replace pk_{ID} with pk'_{ID} .

Signing Queries: When a signing query for an identity ID on some message m is coming, C uses the private key sk_{ID} corresponding to the identity ID to compute the signature S and sends it to A_1 . If the public key pk_{ID} has been replaced by A_1 , then C cannot find sk_{ID} and thus the signing oracle's answer may be incorrect. In such case, we assume that A_1 may additionally submit the secret value r' corresponding to the replaced public key pk_{ID} to the signing oracle.

Finally, A_1 outputs a signature S^* on message m^* corresponding to a public key pk_{ID^*} for an identity ID^* which is the challenged identity ID. A_1 wins the game if CL-Verify(***params***, $ID^*, m^*, Pk_{ID^*}, S^*$)=1 and the following conditions hold:

- Extract private key on identity ID^* has never been queried.
- ID^* can not be an identity for which both the public key has been replaced and the partial private key has been extracted.
- Signing query on message m^* for identity ID^* with respect to pk_{ID^*} has never been queried.

Game 2: This is a game in which A_2 interacts with its Challenger C.

Setup: The challenger runs Setup to generate master key and *params*. C gives both *params* and master key to A_2. A_2 can compute partial private key d_{ID} associated with any identity ID since it holds the master key.

Extract Private Key: For any identity ID except the challenged ID, C firstly computes the partial private key d_{ID} and then secret value as well as private key sk_{ID} corresponding to the identity ID and returns it to A_2.

Request Public Key: A_2 can request the public key for any identity ID. Upon receiving a public key query for any ID, C computes the corresponding public key pk_{ID} and sends it to A_2.

Signing Queries: On receiving such a query, the Challenger C uses the private key sk_{ID} corresponding to the ID to compute the signature S and sends it to A_2.

Finally, A_2 outputs a signature S^* on message m^* corresponding to the challenged identity ID^* and a public key pk_{ID^*}. A_2 wins the game if the following conditions hold:

- CL-Verify (***params***, $ID^*, m^*, Pk_{ID^*}, S^*$)=1.
- CL-Sign (ID^*, m^*) with respect to pk_{ID^*} has been never queried.
- Extract Private Key on ID^* has never been queried.

4 An Efficient CLS Scheme

4.1 The Basic Signature Scheme

The proposed CLS scheme consists of the following seven algorithms.

Setup: Given a security parameter k, the PKG chooses two groups G_1 and G_2 of same prime order $q > 2^k$ and a modified Weil pairing map $e: G_1 \times G_1 \to G_2$. P is a generator of groups G_1. Let $g = e(P, P)$, then PKG selects two distinct cryptographic hash functions $H_1 : \{0,1\}^* \to Z_q^*$, $H_2 : \{0,1\}^* \times G_1 \to Z_q^*$, and picks a random number $s \in Z_q^*$ as its master key and computes its public key $P_{pub} = sP \in G_1$. Afterwards, PKG publishes the system parameter list $\text{params} = \{k, G_1, G_2, e, q, P, g, P_{pub}, H_1, H_2\}$, but keeps s secret.

Partial-Private-Key-Extract: Given an identity $ID \in (0,1)^*$, PKG computes $Q_{ID} = H_1(ID)$, $d_{ID} = \frac{1}{s + Q_{ID}} P$, and sends d_{ID} to a user with identity ID as his partial private by a secure channel. The user with identity ID can check its correctness by checking whether $e(d_{ID}, P_{pub} + Q_{ID}P) = g$. For convenience, here we define $T = P_{pub} + Q_{ID}P$.

Set-Secret-Value: The user with identity ID picks randomly $r \in Z_q^*$ sets *r* as his secret value.

Set-Private-Key: Given *params*, the user's partial private key d_{ID} and his secret value *r*, and output a pair (d_{ID} , *r*) as the user's private key. That is, the user's private key $sk_{ID} = (d_{ID}, r)$ is just the pair consisting of the partial private key and the secret value.

Set-Public-Key: Taking as inputs *params* and the user's secret value *r*, and generates the user's public-key as $pk_{ID} = r(p_{pub} + Q_{ID}P) = rT$.

CL-Sign. In order to generate a signature of an identity ID on a message $m \in (0,1)^*$, the user with the identity ID works as follows:

1. sets $h = H_2(m, pk_{ID})$

2. computes $S = \frac{1}{r+h} d_{ID} = \frac{1}{(r+h)(s+Q_{ID})} P \in G_1$.

The signature of an identity ID on message m is $S \in G_1$.

CL-Verify. Given *params*, message m, pk_{ID} and a signature S for an identity ID, the verifier acts as follows:

1. computes $h = H_2(m, pk_{ID})$
2. accepts the signature S and return 1 if and only if the following equation holds.

$$Ver(m, ID, pk_{ID}, S) = 1 \Leftrightarrow e(S, pk_{ID} + hT) = g$$

The correctness of the verification algorithm is proved as follows:

$$e(S, pk_{ID} + hT) = e(S, rT + hT) = e(S, (r(P_{pub} + Q_{ID}P) + h(P_{pub} + Q_{ID}P))$$

$$= e(\frac{1}{(r+h)(s+Q_{ID})} P, (r+h)(P_{pub} + Q_{ID}P))$$

$$= e(\frac{1}{(r+h)(s+Q_{ID})} P, (r+h)(s+Q_{ID})P)$$

$$= e(P, P) = g$$

4.2 Security Analysis

In this section, we give the security proof for our scheme in the random oracle model.

Theorem 1. Our short CLS scheme is secure against existential forgery under adaptively chosen message attacks in the random oracle model with the assumptions that k-CAA *and inv-CDH* in G_1 is intractable.

This theorem follows from the following Lemmas 1 and 2.

Lemma 1. Let A_1 be a type 1 Adversary in game 1 that (t, ε)-breaks the proposed CLS scheme. Assume that, A_1 makes q_{H_i} queries to random oracles H_i (i = 1, 2) and q_E queries to the partial private-key extraction oracle and q'_E queries to the private-key extraction oracle, and q_{pk} queries to the public-key request oracle, and q_s queries to signing oracle. Then, there exists a (ε', t')-algorithm C that is able to solve the K-CAA problem in group G_1 with probability

$$\varepsilon' \geq (\varepsilon - \frac{1}{2^k}) (\frac{q_{H_1} - 1}{q_{H_1}})^{q_E + q'_E + q_s + 1}$$, and time $t' < t + (2q_{pk} + q_s)t_{sm} + q_s t_{inv}$,

where notation t_{sm} and t_{inv} respectively denotes the running time of computing a scalar multiplication in G_1 and the required time for an inversion computation in G_1.

Proof. Suppose that C is given a challenge:

Given P, $R = sP \in G_1, Q_1, Q_2, \cdots, Q_{q_E} \in Z_q^*$, and $\frac{1}{s+Q_1} P, \frac{1}{s+Q_2} P, \cdots,$

$\frac{1}{s+Q_{q_E}} P,$ C's task is to output a pair$(Q^*, \frac{1}{s+Q^*} P)$ for $Q^* \notin \{Q_1, Q_2, \cdots, Q_{q_E}\}$

after interacting with A_1. Now C and A_1 play the role of the challenger and the adversary respectively. C will interact with A_1 as follows:

Setup: C runs algorithm Setup, sets $g = e(P, P)$ and $P_{pub} = sP$, where s is the system master key, which is unknown to C. C picks an identity ID_I at random as the challenged ID in this game, and gives $\{P, g, P_{pub}, H_1, H_2\}$ to A_1 as the public parameters. For simplicity, we assume that for any ID_i, A_1 queries H_1 before ID_i is used as an input of any query to H_2, Partial Private Key Extraction and Private Key Extraction and Signing.

H_1**-Queries:** C maintains a hash list H_1^{list} of tuple (ID_i, Q_i) as explained below. The list is initially empty. When A_1 makes a hash oracle query on ID_i, if the query ID_i has already appeared on the H_1^{list}, then the previously defined value is returned. Otherwise, C acts as follows:

If $ID_i = ID_I$, C returns a random value $Q^* \notin \{Q_1, Q_2, \cdots, Q_{q_E}\}$ to A_1.Otherwise, C randomly picks a value $Q_i \in \{Q_1, Q_2, \cdots, Q_{q_E}\}$ and returns it to A_1. In both cases, C inserts (ID_i, Q_i) in H_1^{list}.

Partial Private Key Extraction Queries: C maintains a list E^{list} of tuple (ID_i, Q_i, d_{IDi}) is initially empty. For any given identity ID_i, C recovers the corresponding tuple (ID_i, Q_i) from the list H_1^{list}, if $ID_i \neq ID_I$, then sets $d_{ID_i} = \frac{1}{s + Q_i} P$ and returns it to A_1 and adds (ID_i, Q_i, d_{ID_i}) to the E^{list}. Otherwise, C aborts and outputs "failure" (denote the event by E_1).

Public Key Extraction Queries: C maintains a list pk^{list} of tuple ($(ID_i, Q_i, pk_{ID_i}, r_i)$ which is initially empty. When A_1 queries on input ID_i, C checks whether pk^{list} contains a tuple for this input. If it does, the previously defined value is returned. Otherwise, C recovers the corresponding tuple (ID_i, Q_i) from the list H_1^{list} and picks a random value $r_i \in Z_q^*$, computes $pk_{IDi} = r_i(p_{pub} + Q_i P)$ and returns pk_{IDi}. Then, adds $(ID_i, Q_i, pk_{ID_i}, r_i)$ to the pk^{list}.

Private Key Extraction Queries: For query on input ID_i, If $ID_i = ID_I$, C stops and out "failure" (denote the event by E_2).Otherwise, C performs as follows:

- If the E^{list} and the pk^{list} contain the corresponding tuple (ID_i, Q_i, d_{ID_i}) and the tuple $(ID_i, Q_i, pk_{ID_i}, r_i)$ respectively, C sets $sk_{ID_i} = (d_{ID_i}, r_i)$ and sends it to A_1.
- Otherwise, C makes a partial private key extraction query and a public key extraction query on ID_i, then simulates as the above process and sends $sk_{ID_i} = (d_{ID_i}, r_i)$ to A_1.

Public Key Replacement (ID_i, pk'_{ID_i})**:** When A_1 queries on input (ID_i, pk'_{ID_i}), C checks whether the tuple $(ID_i, Q_i, pk_{ID_i}, r_i)$ is contained in the pk^{list}. If it does, sets $pk_{ID_i} = pk'_{ID_i}$ and adds $(ID_i, Q_i, pk_{ID_i}, r_i')$ to the pk^{list}.Here we assume that C can obtain a replacing secret value r_i' corresponding to the replacing public key

pk'_{ID_i} from A_1 . Otherwise, C executes public key extraction to generate $(ID_i, Q_i, pk_{ID_i}, r_i)$, then sets $pk_{ID_i} = pk'_{ID_i}$ and adds $(ID_i, Q_i, pk_{ID_i}, r_i^{'})$ to the pk^{list} .

H_2 **-Queries:** C maintains a hash list H_2^{list} of tuple $(m_j, ID_i, Q_i, pk_{ID_i}, h_j)$. When A_1 makes H_2 queries for identity ID_i on the message m_j , C chooses a random number $h_j \in Z_q^*$, sets $h_j = H_2(m_j, pk_{ID_i})$ and sends h_j to A_1 . And then adds $(m_j, ID_i, Q_i, pk_{ID_i}, h_j)$ to the H_2^{list} .

Signing Queries: When a signing query (ID_i, m_j) is coming, C acts as follows:

- If $ID_i = ID_I$, C stops and out "failure" (denote the event by E_3).
- Otherwise, C recovers the tuple (ID_i, Q_i, d_{ID_i}) from the E^{list} and the tuple $(ID_i, Q_i, pk_{ID_i}, r_i)$ from the pk^{list} and the tuple $(m_j, ID_i, Q_i, pk_{ID_i}, h_j)$ from H_2^{list} , computes $S_j = \dfrac{d_{ID_i}}{r_i + h_j} = \dfrac{1}{(r_i + h_j)(s + Q_i)} P$, and S_j is a signature for the identity ID_i on the message m_j . C returns S_j to A_1 as the response of the singing oracle.

Finally, A_1 stops and outputs a signature S^* on the message m^* for the identity ID^* which satisfies the equation $Ver(m^*, ID^*, pk_{ID_i^*}, S^*) = 1$. If $ID^* \neq ID_I$, C outputs "failure" and aborts (denote the event by E_4). Otherwise, C recovers the tuple $(ID^*, Q^*, pk_{ID_i^*}, r_i^*)$ from pk^{list} and the tuple $(m^*, ID^*, Q^*, pk_{ID_i^*}, h^*)$ from H_2^{list} .

Then, we have $e(S^*, pk_{ID^*} + h^*(P_{pub} + Q^* P)) = e(P, P)$, that is,

$$e(S^*, (r^* + h^*)(P_{pub} + Q^* P)) = e(S^*, (r^* + h^*)(s + Q^*)P)$$
$$= e((r^* + h^*)(s + Q^*)S^*, P) = e(P, P)$$

Hence C can successfully compute $\dfrac{1}{s + Q^*} P = (r^* + h^*)S^*$ and output a pair $(Q^*, \dfrac{1}{s + Q^*} P)$ for $Q^* \notin \{Q_1, Q_2, \cdots, Q_{q_E}\}$ as a solution to A_1 's challenge. So, C breaks k-CAA problem in G_1 .

Now analyze the advantage of C in this game.

Note that the responses to A_1 's H_1 and H_2 queries are indistinguishable from the real life. Since each response is uniformly random and independently distributed in Z_q^* . The responses of queries H_1 and H_2 provided for A_1 are all valid. The responses of partial private key extraction queries, private key extraction queries and signing queries are valid if the event E_1 , E_2 and E_3 never happens. Furthermore, if A_1 forges a valid signature and event E_4 does not happen, then C can solve the k-CAA problem. So if none of events E_1 , E_2 , E_3 and E_4 happens, C can solve the k-CAA problem successfully. Now, Let's bound the probability for these events. From the description above, we have $\Pr(\neg E_1 \wedge \neg E_2 \wedge \neg E_3 \wedge \neg E_4) = (\dfrac{q_{H_1} - 1}{q_{H_1}})^{q_E + q_E' + q_s + 1}$. Nevertheless, the

probability that the simulation is not perfect remains to be assessed. The only event where it can happen is that A_1 forges a valid signature without making H_2 queries. It is easy to see that the probability to generate a valid signature without asking H_2 hash oracles is at most $\frac{1}{2^k}$.

Taking the above analysis on these events, we know that challenger C's advantage $\varepsilon' \geq (\varepsilon - \frac{1}{2^k})\ (\frac{q_{H_1}-1}{q_{H_1}})^{q_E+q'_E+q_s+1}$. From the above description of C, we can conclude that the running time of C is bound by $t' < t + (2q_{pk} + q_s)t_{sm} + q_s t_{inv}$.

Lemma 2. Assume that A_2 is a Type 2 Adversary that (t, ε)-breaks our CLS scheme after making q_{H_i} queries to random oracles H_i (i = 1, 2) and q_E queries to the private-key extraction oracle , and q_{pk} queries to the public-key request oracle, and q_s queries to signing oracle. Then, the Inv-CDHP can be solved with probability $\varepsilon' \geq (\varepsilon - \frac{1}{2^k})\ (\frac{q_{H_1}-1}{q_{H_1}})^{q_E+q_s+1}$ and within time $t' < t + (2q_{pk} + q_s)t_{sm} + q_s t_{inv}$, where notation t_{sm} and t_{inv} respectively denotes the running time of computing a scalar multiplication in G_1 and the required time for an inversion computation in G_1.

Proof. Suppose C is given a challenge of a random instance of the Inv-CDH problem: Given $P \in G_1, h^* \in Z_q^*$ and $(r + h^*)P$, where r is unknown to C. C's goal is to output $\frac{1}{r+h^*}P \in G_1$ by interacting with adversary A_2.

Now C and A_2 play the role of the challenger and the adversary respectively. C will interact with A_2 as follows:

Setup: C runs algorithm Setup, randomly picks a value $s \in Z_q^*$ as the system master key, sets $g = e(P, P)$, $X = rP$ and $P_{pub} = sP$, and picks an identity ID_I at random as the challenged ID in this game, and gives $params\{P, g, P_{pub}, H_1, H_2\}$ and the system master key ***s*** to A_2.

H_1**-Queries:** C maintains a hash list H_1^{list} of tuple (ID_i, Q_i) as explained below. The list is initially empty. When A_2 makes a hash oracle query on ID_i, if the query ID_i has already appeared on the H_1^{list}, then the previously defined value is returned. Otherwise, C randomly picks a value $Q_i \in Z_q^*$ and returns it to A_2. Then, adds (ID_i, Q_i) to H_1^{list}.

Public Key Extraction Queries: C maintains a list pk^{list} of tuple ($(ID_i, Q_i, pk_{ID_i}, r_i)$ which is initially empty. When A_2 queries on input ID_i, C checks whether pk^{list} contains a tuple for this input. If it does, the previously defined value is returned. Otherwise, C works as follows:If $ID_i = ID_I$, C finds the tuple (ID_i, Q_i) in H_1^{list} and sets $pk_{ID_i} = sX + Q_i X$ and sends pk_{ID_i} to A_2, and inserts (ID_i, Q_i, pk_{ID_i}) into pk^{list} .Otherwise, C recovers the tuple (ID_i, Q_i) from H_1^{list} and picks a random value $r_i \in Z_q^*$, computes $pk_{IDi} = r_i(P_{pub} + Q_i P)$ and returns pk_{IDi}. Then, adds $(ID_i, Q_i, pk_{ID_i}, r_i)$ to the pk^{list}.

Private Key Extraction Queries: For query on input ID_i. If $ID_i = ID_I$, C stops and out "failure" Otherwise, C recovers the tuple $(ID_i, Q_i, pk_{ID_i}, r_i)$ from the list pk^{list}, and sends $sk_{ID_i} = (d_{ID_i}, r_i)$ to A_2.

H_2 **-Queries:** C maintains a hash list H_2^{list} of tuple $(m_j, ID_i, Q_i, pk_{ID_i}, h_j)$. When A_2 makes H_2 queries for identity ID_i on the message m_j, C chooses a random number $h_j \in Z_q^*$, sets $h_j = H_2(m_j, pk_{ID_i})$ and sends h_j to A_2. And then adds $(m_j, ID_i, Q_i, pk_{ID_i}, h_j)$ to the H_2^{list}.

Signing Queries: When a signing query (ID_i, m_j) is coming, C does the following:

- If $ID_i = ID_I$, C stops and out "failure".
- Otherwise, C recovers the corresponding tuple $(ID_i, Q_i, pk_{ID_i}, r_i)$ from the pk^{list} and the corresponding tuple $(m_j, ID_i, Q_i, pk_{ID_i}, h_j)$ from the H_2^{list}, computes $S_j = \frac{d_{ID_i}}{r_i + h_j} = \frac{1}{(r_i + h_j)(s + Q_i)} P$, then S_j is a signature for the identity ID_i on the message m_j. C returns S_j to A_2 as the response of the singing oracle.

Finally, A_2 outputs a signature S^* on the message m^* with respect to the public key $pk_{ID_i^*}$ for the identity ID^*, which satisfies $Ver(m^*, ID^*, pk_{ID_i^*}, S^*) = 1$.

- If $ID^* \neq ID_I$, C outputs "failure" and aborts.
- Otherwise, C recovers the corresponding tuple $(ID^*, Q^*, pk_{ID_i^*})$ from pk^{list} and the corresponding tuple $(m^*, ID^*, Q^*, pk_{ID_i^*}, h^*)$ from H_2^{list}. Then, we have $e(S^*, pk_{ID^*} + h^*(P_{pub} + Q^* P)) = e(P, P)$ and $pk_{ID_i^*} = sX + Q^* X$.

  That is, $e(S^*, (r + h^*)(P_{pub} + Q^* P)) = e(S^*, (r + h^*)(s + Q^*)P)$

  $= e((r + h^*)(s + Q^*)S^*, P) = e(P, P)$.

Hence C can successfully compute $\frac{1}{r + h^*} P = (s + Q^*)S^*$ and output $\frac{1}{r + h^*} P$ as a solution to A_2's challenge. So, C breaks Inv-CDH problem in G_1. The analysis of C's advantage and running time is similar to that of the Lemma 1. This completes our proof.

4.3 Efficiency

In practice, the size of the element in group G_1 can be reduced by a factor of 2 using compression techniques. So, like BLS signature scheme [10], our signature scheme is a short CLS scheme. If we choose a group and the bilinear map from elliptic curves [10], which results in a group of 160 bits size, signatures generated by our scheme is approximate160 bits length which is half–size compared to all proposed CLS schemes.

Our CLS scheme only requires one scalar multiplication operation in CL-Sign algorithm and one scalar multiplication computation and one pairing operation in CL-Verify algorithm. Obviously, it is faster than all other proposed CLS schemes. Concretely, denote by s a scalar multiplication in G_1 and by p computation of one pairing, other operations are omitted in the following analysis since their computation

cost is trivial, such as the cost of an inverse operation over Z_q^* takes only 0.03ms. The comparison of our CLS scheme's computation cost and that of other proposed schemes is as follows: (We do not consider the pre-computation here)

Scheme	AP[2]	LCS[4]	YHG[7]	GS[6]	Our CLS
Sign	3s+1p	2s	2s	2s	1s
Verify	1e+4p	2s+4p	2p+2s	1s+3p	1s+1p
Public Key Size(bits)	320	320	160	160	160
Signature Size(bits)	320	320	320	320	160

(Table 1)

As is shown in the table 1, one can see that our scheme is the most efficient scheme in terms of the number of pairing operations required and the size of public key and signatures generated by our scheme.

5 Conclusion

In this paper, we come up with a short CLS scheme that is proved to be secure in the random oracle model under the hardness assumption of k-CAA and Inv-CDHP. Our scheme, besides upholding all desirable properties of previous CLS schemes, it is significantly more efficient than all existing CLS schemes. Furthermore, the size of signatures generated by our scheme is the smallest in all proposed CLS schemes. So, it can be used in low-bandwidth, low-power situations such as mobile security applications where the need to transmit and check certificates has been identified as a significant limitation.

References

[1] A. Shamir, "Identity-based cryptosystems and signature schemes," in Proc. Crypto'84, Santa Barbara, CA, pp. 47–53, Aug. 1984.

[2] S. Al-Riyami and K.G. Paterson. Certificateless Public Key Cryptography. *In Proceedings of ASIACRYPT 2003*, LNCS 2894, pp. 452-473, Springer-Verlag, 2003.

[3]D.H. Yum, P.J. Lee. Generic construction of certificateless signature. In ACISP'04, LNCS **3108**, Springer. 2004, pp. 200-211.

[4]. X. Li, K. Chen and L. Sun. Certificateless Signature and Proxy Signature Schemes from Bilinear Pairings. *Lithuanian Mathematical Journal*, Vol 45, pp. 76-83, Springer-Verlag, 2005.

[5] Z. F.Zhang, D. S. Wong, J.Xu, et al. Certificateless Public-Key Signature: Security Model and Efficient Construction. J. Zhou, M. Yung, and F. Bao (Eds.): ACNS 2006, LNCS 3989, pp. 293–308, 2006. Springer-Verlag Berlin Heidelberg 2006.

[6]M.C. Gorantla, A. Saxena. An efficient certificateless signature scheme. Y. Hao et al. (Eds.): CIS 2005, Part II, LNAI 3802, pp. 110–116, 2005.Springer-Verlag Berlin Heidelberg 2005.

[7]W.-S, Yap, S.-H.Heng and B.-M. Goi. An efficient certificateless signature scheme. Proc. Of EUC Workshops 2006, LNCS. Vol. 4097, pp. 322-331, 2006.

[8]J. H. Park. An attack on the certificateless signature scheme from EUC Workshops 2006. eprint, 2007.

[9]P.S.L.M. Barreto and H.Y. Kim, Fast hashing onto elliptic curves over fields of characteristic 3, Cryptology ePrint Archive, Report 2001/098, available at http://eprint.iacr.org/2001/098/.

[10] D. Boneh, B. Lynn, and H. Shacham, Short signatures from the Weil pairing, In C. Boyd, editor, Advances in Cryptology-Asiacrypt 2001, LNCS 2248, pp.514-532, Springer-Verlag, 2001.

[11] S. Mitsunari, R. Sakai and M. Kasahara, A new traitor tracing, IEICE Trans. Vol.E85-A, No.2, pp.481-484, 2002.

[12]I. Blake, G. Seroussi and N. Smart. Advances in elliptic curve cryptography. London Mathematical Society Lecture Note Series. Cambridge University Press, 2005.