Efficient and provably-secure certificateless signature scheme without bilinear pairings

He Debiao*, Chen Jianhua, Zhang Rui
School of Mathematics and Statistics, Wuhan University, Wuhan, Hubei, China, 430072

Abstract: Many certificateless signature schemes using bilinear pairings have been proposed. But the relative computation cost of the pairing is approximately twenty times higher than that of the scalar multiplication over elliptic curve group. In order to improve the performance we propose a certificateless signature scheme without bilinear pairings. With the running time being saved greatly, our scheme is more practical than the previous related schemes for practical application.
Key words: Certificateless public key cryptography; Certificateless signature; Bilinear pairings; Elliptic curve

1. Introduction

Public-key cryptography(PKC) has become one of the essential techniques in providing security services in modern communications. In traditional public-key cryptosystems, a pair of public/private keys should be computed by each user. Since the public key is a string of random bits, a digital certificate of the public key is required to provide public-key authentication. 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 simplify the public-key authentication, Shamir [1] introduced the concept of identity-based (ID-based) cryptosystem problem. In this system, each user needs to register at a key generator centre (KGC) with identify of himself before joining the network. Once a user is accepted, the KGC will generate a private key for the user and the user’s identity (e.g. user’s name or email address) becomes the corresponding public key. In this way, in order to verify a digital signature or send an encrypted message, a user only needs to know the “identity” of his communication partner and the public key of the KGC. However, this cryptosystem involves a KGC, which is responsible for generating a user's private key based on his identity. As a result, the KGC can literally decrypt any ciphertext or 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). 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 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 can be solved. Intuitionally, CLPKC has nice features borrowed from both ID-based cryptography and traditional PKC. It alleviates the key escrow problem in ID-based cryptography and at the same time reduces the cost and simplifies the use of the technology when compared with traditional

*Corresponding author.
E-mail: hedebiao@163.com, *Tel*:+0086015307184927

PKC.

Following the pioneering work due to Al-Riyami and Paterson [2], several certificateless signature (CLS) schemes [3-10] have been proposed. All the above CLS schemes may be practical, but they are from bilinear pairings and the pairing is regarded as the most expensive cryptography primitive. The relative computation cost of a pairing is approximately twenty times higher than that of the scalar multiplication over elliptic curve group [11]. Therefore, CLS schemes without bilinear pairings would be more appealing in terms of efficiency.

In this paper, we present a CLS scheme without pairings. The scheme rests on the elliptic curve discrete logarithm problem (ECDLP).With the pairing-free realization, the scheme's overhead is lower than that of previous schemes [3-10] in computation.

2. Preliminaries

2.1.Background of elliptic curve group

Let the symbol E/F_p denote an elliptic curve E over a prime finite field F_p, defined by an equation

$$y^2 = x^3 + ax + b\ ,\quad a,b \in F_p \tag{1}$$

and with the discriminant

$$\Delta = 4a^3 + 27b^2 \neq 0\ . \tag{2}$$

The points on E/F_p together with an extra point O called the point at infinity form a group

$$G = \{(x,y): x, y \in F_p, E(x,y) = 0\} \cup \{O\}\ . \tag{3}$$

Let the order of G be n. G is a cyclic additive group under the point addition "+" defined as follows: Let $P, Q \in G$, l be the line containing P and Q (tangent line to E/F_p if $P = Q$), and R, the third point of intersection of l with E/F_p. Let l' be the line connecting R and O. Then P "+" Q is the point such that l' intersects E/F_p at R and O and P "+" Q. Scalar multiplication over E/Fp can be computed as follows:

$$tP = P + P + \cdots + P(t\ times) \qquad (4).$$

The following problem defined over G is assumed to be intractable within polynomial time.

Eliptic curve discrete logarithm problem(ECDLP): For $x \in_R Z_n^*$ and P the generator of

G , given $Q = x \cdot P$ compute x .

2.2.Certificateless signatures

A CLS scheme consists of seven algorithms[2]: *Setup*, *Partial-Private-Key-Extract*, *Set-Secret-Value*, *Set-Private-Key*, *Set-Public-Key*, *Sign* and *Verify*.

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

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

Set-Secret-Value: This algorithm $params$ and a user's identity ID as input and 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 as inputs and outputs the full private key sk_{ID} .

Set-Public-Key: This algorithm takes $params$ and a user's secret value r as inputs and generates a public key pk_{ID} for the user.

Sign: This algorithm takes $params$, a message m, a user's identity ID , and the user's private key sk_{ID} as inputs and outputs a signature S .

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

2.3.Security model for certificateless signatures

In CLS, as defined in [2], there are two types of adversaries with different capabilities, we assume *Type 1 Adversary*, $\mathscr{A}1$ acts as a dishonest user while *Type 2 Adversary*, $\mathscr{A}2$ acts as a malicious KGC:

Type 1 Adversary: Adversary $\mathscr{A}1$ does not have access to the master key, but $\mathscr{A}1$ can replace the public keys of any entity with a value of his choice, since there is no certificate involved in CLS.

Type 2 Adversary: Adversary $\mathscr{A}2$ has access to the master key, but cannot replace any user's public key.

Definition 1. Let $\mathscr{A}1$ and $\mathscr{A}2$ be a Type1Adversaryanda Type2Adversary, respectively. We consider two games Game 1 and Game 2 where $\mathscr{A}1$ and $\mathscr{A}2$ interact with its 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 $\mathcal{A}1$ and $\mathcal{A}2$ is negligible.

Game 1: This is the game where $\mathcal{A}1$ interacts with its challenger $\mathcal{C}$:

Setup: The challenger $\mathcal{C}$ takes a security parameter k and runs ***Setup*** to generate master key and $params$, then sends $params$ to $\mathcal{A}1$. $\mathcal{A}1$ acts as the following oracle queries:

Hash Queries: $\mathcal{A}1$ can request the hash values for any input.

Extract Partial Private Key: $\mathcal{A}1$ is able to ask for the partial private key d_{ID} for any ID except the challenged identity ID. $\mathcal{C}$ computes the partial private key d_{ID} corresponding to the identity ID and returns d_{ID} to $\mathcal{A}1$.

Extract Private Key: For any ID except the challenged identity ID, $\mathcal{C}$ computes the private key sk_{ID} corresponding to the identity ID and returns sk_{ID} to $\mathcal{A}1$.

Request Public Key: Upon receiving a public key query for any identity ID, $\mathcal{C}$ computes the corresponding public key pk_{ID} and sends it to $\mathcal{A}1$.

Replace Public Key: For any identity ID, $\mathcal{A}1$ 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, $\mathcal{C}$ uses the private key sk_{ID} corresponding to the identity ID to compute the signature S and sends it to $\mathcal{A}1$. If the public key pk_{ID} has been replaced by $\mathcal{A}1$, then $\mathcal{C}$ cannot find sk_{ID} and thus the signing oracle's answer may be incorrect. In such case, we assume that $\mathcal{A}1$ additionally submits the secret value r' corresponding to the replaced public key sk_{ID} to the signing oracle.

Finally, $\mathcal{A}1$ outputs a signature S^* on a message m^* corresponding to a public key pk_{ID^*} for an identity ID^* which is the challenged identity ID. $\mathcal{A}1$ wins the game if $Verify(params, ID^*, m^*, pk_{ID^*}, S^*) = 1$ 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 $\mathcal{A}2$ interacts with its challenger $\mathcal{C}$.

Setup: $\mathcal{C}$ runs Setup to generate a master key and $params$. $\mathcal{C}$ gives both $params$ and the master key to $\mathcal{A}2$.

Extract Private Key: For any identity ID except the challenged ID, $\mathcal{C}$ computes the private key sk_{ID} corresponding to the identity ID and returns sk_{ID} to $\mathcal{A}2$.

Request Public Key: Upon receiving a public key query for any ID, $\mathcal{C}$ computes the corresponding public key pk_{ID} and sends it to $\mathcal{A}2$.

Signing Queries: On receiving such a query on a identity ID, $\mathcal{C}$ uses the private key sk_{ID} corresponding to the identity ID to compute the signature S and sends it to $\mathcal{A}2$.

Finally, $\mathcal{A}2$ outputs a signature S^* on a message m^* corresponding to a public key pk_{ID^*} for an identity ID^* which is the challenged identity ID. $\mathcal{A}2$ wins the game if the following conditions hold:

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

3. Our scheme

3.1.Scheme Description

In this section, we present an ID-based signature scheme without pairing. Our scheme consists of four algorithms: *Setup*, *Extract*, *Sign*, and *Verify*.

Setup: This algorithm takes a security parameter k as input, returns system parameters and a master key. Given k, KGC does as follows.

1) Choose a k-bit prime p and determine the tuple $\{F_p, E/F_p, G, P\}$ as defined in Secttion 2.1.
2) Choose the master private key $x \in Z_n^*$ and compute the master public key $P_{pub} = x \cdot P$.
3) Choose two cryptographic secure hash functions $H_1 : \{0,1\}^* \to Z_n^*$ and $H_2 : \{0,1\}^* \to Z_n^*$.
4) Publish $params = \{F_p, E/F_p, G, P, P_{pub}, H_1, H_2\}$ as system parameters and keep

the master key x secretly.

Partial-Private-Key-Extract: This algorithm takes system parameters, master key and a user's identifier as input, returns the user's ID-based private key. With this algorithm, KGC works as follows for each user with identifier ID_U.

1) Choose at random $r_{ID} \in Z_n^*$, compute $R_{ID} = r_{ID} \cdot P$ and $h_{ID} = H_1(ID, R_{ID})$.

2) Compute $s_{ID} = r_{ID} + h_U x \bmod n$.

The user's s partial private key is the tuple $d_{ID} = \{s_{ID}, R_{ID}\}$ and he can validate her private key by checking whether the equation $s_{ID} \cdot P = R_{ID} + h_{ID} \cdot P_{pub}$ holds. The private key is valid if the equation holds and vice versa.

Set-Secret-Value: The user with identity ID picks randomly $s'_{ID} \in Z_n^*$ sets s'_{ID} as his secret value.

Set-Private-Key: Given $params$, the user's partial private key d_{ID} and his secret value s'_{ID} and output a pair (d_{ID}, s'_{ID}) as the user's private key.

Set-Public-Key: This algorithm takes $params$, the user's secret value s'_{ID} as inputs, and generates the user's public key as $pk_{ID} = s'_{ID} \cdot P$.

Sign: This algorithm takes system parameters, user's partial private key d_{ID}, user's secret value s'_{ID}, and a message m as inputs, returns a signature of the message m. The user does as follows.

1) Choose at random $l \in Z_n^*$ to compute $R = l \cdot P$.

2) Compute $h = H_2(m, R, pk_{ID})$.

3) Verify whether the equation $\gcd(l + h, n) = 1$ holds. If the equation does not hold, return to step 1).

4) Compute $s = (l + h)^{-1}(s_{ID} + s'_{ID}) \bmod n$.

5) The resulting signature is (R_{ID}, R, s).

Verify: To verify the signature (R_{ID}, R, s) for message m and identity ID, the verifier first computes $h_{ID} = H_1(ID, R_{ID})$, $h = H_2(m, R, pk_{ID})$ and then checks whether

$$s \cdot (R + h \cdot P) = pk_{ID} + R_{ID} + h_{ID} P_{pub} \quad (5)$$

Accept if it is equal. Otherwise reject.

Since $R = l \cdot P$, $s_{ID} = r_{ID} + h_U x \bmod n$ and $s = (l+h)^{-1}(s_{ID} + s'_{ID}) \bmod n$, we have

$$\begin{aligned} s \cdot (R + h \cdot P) &= (l+h)^{-1}(s_{ID} + s'_{ID}) \cdot (l \cdot P + h \cdot P) \\ &= (l+h)^{-1}(s_{ID} + s'_{ID}) \cdot (l+h) \cdot P = (s_{ID} + s'_{ID}) \cdot P \\ &= s'_{ID} \cdot P + s_{ID} \cdot P = pk_{ID} + R_{ID} + h_U \cdot P_{pub} \end{aligned} \quad (6)$$

Then the correctness of our scheme is proved.

3.2.Security Analysis

We prove the security of our scheme Σ in the random oracle model which treats H_1 and H_2 as two random oracles [9] using the signature security model defined in [2]. As for the security of Σ, the following theorem is provided.

Theorem 1. Our scheme is secure against existential forgery under adaptively chosen message attacks in the random oracle model with the ECDLP is intractable.

This theorem follows from the following Lemmas 1 and 2.

Lemma 1. Let $\mathscr{A}1$ be a type 1 Adversary in game 1. Assume $\mathscr{A}1$ 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. If $\mathscr{A}1$ can break Σ with probability $\varepsilon \geq 10(q_{H_2}+1)(q_{H_2}+q_S)/2^k$, then the ECDLP can be solved within running time $t \leq 23 q_{H_2} t/\varepsilon$ and with probability $\varepsilon' \geq 1/9$.

Proof: Suppose that there is a type 1 Adversar $\mathscr{A}1$ for an adaptively chosen message attack against Σ. Then, we show how to use the ability of $\mathscr{A}1$ to construct an algorithm $\mathcal{F}$ solving the ECDLP.

Suppose $\mathcal{F}$ is challenged with a ECDLP instance (P, Q) and is tasked to compute $x \in Z_n^*$ satisfying $Q = x \cdot P$. To do so, $\mathcal{F}$ picks an identity ID_I at random as the challenged ID in this game, and gives $\{F_p, E/F_p, G, P, P_{pub} = Q, H_1, H_2\}$ to $\mathscr{A}1$ as the public parameters. Then $\mathcal{F}$ answers $\mathscr{A}1$'s queries as follows.

***H1*-Queries**: $\mathcal{F}$ maintains a hash list L_{H_1} of tuple ($ID_i, R_{ID_i}, s_{ID_i}, h_{ID_i}$) as explained below. The list is initially empty. When $\mathcal{A}1$ makes a hash oracle query on ID_i, if the query ID_i has already appeared on L_{H_1}, then the previously defined value is returned. Otherwise, $\mathcal{F}$ acts as described in the partial private key extraction queries.

Partial Private Key Extraction Queries: $\mathcal{A}1$ is allowed to query the extraction oracle for an identity ID_i. $\mathcal{F}$ query H_1 oracle, ID_i is on L_{H_1}, then $\mathcal{F}$ response with ($ID_i, R_{ID_i}, s_{ID_i}, h_{ID_i}$). Otherwise, if simulates the oracle as follows. It chooses $a_i, b_i \in Z_n^*$ at random, sets $R_{ID_i} = a_i \cdot P_{pub} + b_i \cdot P$, $s_{ID_i} = b_i$, $h_{ID_i} = H_1(ID_i, R_{ID_i}) \leftarrow -a_i \bmod n$, response with ($ID_i, R_{ID_i}, s_{ID_i}, h_{ID_i}$), and inserts ($ID_i, R_{ID_i}, s_{ID_i}, h_{ID_i}$) into L_{H_1}. Note that ($R_{ID_i}, s_{ID_i}, h_{ID_i}$) generated in this way satisfies the equation $s_{ID} \cdot P = R_{ID} + h_{ID} \cdot P_{pub}$ in the partial private key extraction algorithm. It is a valid secret key.

Public Key Extraction Queries: $\mathcal{F}$ maintains a list L_{pk} of tuple ($ID_i, s'_{ID_i}, pk_{ID_i}$) which is initially empty. When $\mathcal{A}1$ queries on input ID_i, $\mathcal{F}$ checks whether L_{pk} contains a tuple for this input. If it does, the previously defined value is returned. Otherwise, $\mathcal{F}$ picks a random value $s'_{ID} \in Z_n^*$, computes $pk_{ID_i} = s'_{ID_i} \cdot P$ and returns pk_{ID_i}. Then, adds ($ID_i, s'_{ID_i}, pk_{ID_i}$) to the L_{pk}.

Private Key Extraction Queries: For query on input ID_i, If $ID_i = ID_I$, $\mathcal{F}$ stops and outputs "failure". Otherwise, $\mathcal{F}$ performs as follows:

If the L_{H_1} and the L_{pk} contain the corresponding tuple ($ID_i, R_{ID_i}, s_{ID_i}, h_{ID_i}$) and the tuple ($ID_i, s'_{ID_i}, pk_{ID_i}$) respectively, $\mathcal{F}$ sets $sk_{ID_i} = \{d_{ID_i}, s'_{ID_i}\}$ and sends it to $\mathcal{A}1$, where $d_{ID} = \{s_{ID}, R_{ID}\}$. Otherwise, $\mathcal{F}$ makes a partial private key extraction query and a public key extraction query on ID_i, then simulates as the above process and sends $sk_{ID_i} = \{d_{ID_i}, s'_{ID_i}\}$ to $\mathcal{A}1$.

Public Key Replacement:When $\mathcal{A}1$ queries on input ($ID_i, \widetilde{pk_{ID_i}}$), $\mathcal{F}$ checks whether the tuple ($ID_i, s'_{ID_i}, pk_{ID_i}$) is contained in the L_{pk}. If it does, $\mathcal{F}$ sets $pk_{ID_i} = \widetilde{pk_{ID_i}}$ and adds the tuple ($ID_i, \widetilde{s'_{ID_i}}, pk_{ID_i}$) to the L_{pk} . Here we assume that $\mathcal{F}$ can obtain a replacing secret value $\widetilde{s'_{ID_i}}$ corresponding to the replacing public key $\widetilde{pk_{ID_i}}$ from $\mathcal{A}1$. Otherwise, $\mathcal{F}$ executes public key extraction to generate ($ID_i, s'_{ID_i}, pk_{ID_i}$), then sets $pk_{ID_i} = \widetilde{pk_{ID_i}}$ and adds ($ID_i, \widetilde{s'_{ID_i}}, pk_{ID_i}$) to the L_{pk} .

***H2*-Queries**: $\mathcal{F}$ maintains a hash list L_{H_2} of tuple ($m_j, R_j, ID_i, pk_{ID_i}, h_j$). When $\mathcal{A}1$ makes *H2* queries for identity ID_i on the message m_j , $\mathcal{F}$ chooses a random value $h_j \in Z_n^*$, sets $h_j = H_2(m_j, R_j, pk_{ID_i})$ and adds ($m_j, R_j, ID_i, pk_{ID_i}, h_j$) to L_{H_2} , and sends h_j to $\mathcal{A}1$.

Signing Queries: When a signing query on (ID_i, m_j) is coming, $\mathcal{F}$ acts as follows:

If $ID_i = ID_I$, $\mathcal{F}$ outputs "failure". Otherwise, $\mathcal{F}$ recovers ($ID_i, R_{ID_i}, s_{ID_i}, h_{ID_i}$) from L_{H_1} and ($ID_i, s'_{ID_i}, pk_{ID_i}$) from L_{pk} . Then $\mathcal{F}$ gets the secret key $sk_{ID_i} = \{d_{ID_i}, s'_{ID_i}\}$, he can execute the sign algorithm as described in section 3.1. At last, $\mathcal{F}$ response with (R_{ID_i}, R_j, s_j).

Finally, $\mathcal{A}1$ stops and outputs a signature $S^* = \{R_{ID^*}, R_j, s_j\}$ on the message m^* with respect to the public key pk_{ID^*} for the identity ID^*, which satisfies the following equation $Verify(params, ID^*, m^*, pk_{ID^*}, S^*) = 1$.

If $ID_i \neq ID_I$, $\mathcal{F}$ outputs "failure" and aborts. Otherwise, $\mathcal{F}$ recovers the tuple ($ID_i, R_{ID_i}, s_{ID_i}, h_{ID_i}$) from L_{H_1} , the tuple ($ID_i, s'_{ID_i}, pk_{ID_i}$) from L_{pk} and the tuple ($m_j, R_j, ID_i, pk_{ID_i}, h_j$) from L_{H_2} .

Then, we have

$$s_j \cdot (R_j + h_j \cdot P) = pk_{ID_i} + R_{ID_i} + h_{ID_i} P_{pub} \tag{7}$$

From the forgery lemma[12], if we have a replay of $\mathcal{F}$ with the same random tape but different choice of H_2 will output another valid signatures $\{R_{ID_i}, R_j, s'_j\}$. Then we have

$$s'_j \cdot (R_j + h'_j \cdot P) = pk_{ID_i} + R_{ID_i} + h_{ID_i} P_{pub} , \tag{8}$$

When eliminating R_j in the above two equation, we could have

$$h_{ID_i}(s_j' - s_j)P_{pub} = s_j s_j'(h_j - h_j')P - (s_j - s_j')(pk_{ID_i} + R_{ID_i}) \quad (9)$$

Let $pk_{ID_i} = s_{ID_i}' \cdot P$, $R_{ID_i} = a_i \cdot P_{pub} + b_i \cdot P$, $P_{pub} = Q = x \cdot P$, then we have

$$h_{ID_i}(s_j' - s_j)x = s_j s_j'(h_j - h_j') - (s_j - s_j')(s_{ID_i}' + a_i \cdot x + b_i)\,. \quad (10)$$

Hence, we have

$$(h_{ID_i} - a_i)(s_j' - s_j)x = s_j s_j'(h_j - h_j') - (s_j - s_j')(s_{ID_i}' + b_i)\,. \quad (11)$$

Let $u = ((h_{ID_i} - a_i)(s_j' - s_j))^{-1} \bmod n$ and $v = s_j s_j'(h_j - h_j') - (s_j - s_j')(s_{ID_i}' + b_i)$, then, we get $x = uv \bmod n$. According to [12, Lemma 4], the ECDLP can be solved with probability $\varepsilon' \geq 1/9$ and time $t' \leq 23 q_{H_2} t / \varepsilon$.

Lemma 2. Let $\mathscr{A}2$ be a type 2 Adversary in game 2. Assume that, $\mathscr{A}2$ 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. If $\mathscr{A}2$ can break Σ with probability $\varepsilon \geq 10(q_{H_2}+1)(q_{H_2}+q_S)/2^k$, then the ECDLP can be solved within running time $t \leq 23 q_{H_2} t / \varepsilon$ and with probability $\varepsilon' \geq 1/9$.

Proof: Suppose that there is a type 2 Adversar $\mathscr{A}2$ for an adaptively chosen message attack against Σ. Then, we show how to use the ability of $\mathscr{A}2$ to construct an algorithm $\mathscr{F}$ solving the ECDLP.

Suppose $\mathscr{F}$ is challenged with a ECDLP instance (P, Q) and is tasked to compute $y \in Z_n^*$ satisfying $Q = y \cdot P$. To do so, $\mathscr{F}$ randomly picks a value $x \in Z_n^*$ as the system master key, sets $P_{pub} = x \cdot P$, picks an an identity ID_I at random as the challenged ID in this game, and gives the public parameters $\{F_p, E/F_p, G, P, P_{pub}, H_1, H_2\}$ and the system master key x to $\mathscr{A}2$. Then $\mathscr{F}$ answers $\mathscr{A}1$'s queries as follows.

***H1*-Queries**: $\mathscr{F}$ maintains a hash list L_{H_1} of tuple (ID_i, R_{ID_i}, h_{ID_i}). The list is initially empty. When $\mathscr{A}2$ makes a hash oracle query on ID_i, if the query ID_i has already appeared on L_{H_1}, then the previously defined value is returned. Otherwise, $\mathscr{F}$ chooses a random value $h_{ID_i} \in Z_n^*$, sets $h_{ID_i} = H_1(ID_i, R_{ID_i})$ and adds (ID_i, R_{ID_i}, h_{ID_i}) to L_{H_1} , and sends h_{ID_i} to $\mathcal{A}1$.

Public Key Extraction Queries: $\mathcal{F}$ maintains a list L_{pk} of tuple ($ID_i, s'_{ID_i}, pk_{ID_i}$) which is initially empty. When $\mathcal{A}2$ queries on input ID_i , $\mathcal{F}$ checks whether L_{pk} contains a tuple for this input. If it does, the previously defined value is returned. Otherwise, if $ID_i = ID_I$, $\mathcal{F}$ sets $s'_{ID_i} = \perp$, $pk_{ID_i} = Q$, adds ($ID_i, s'_{ID_i}, pk_{ID_i}$) to the L_{pk} . If $ID_i \neq ID_I$, $\mathcal{F}$ picks a random value $s'_{ID} \in Z_n^*$, computes $pk_{ID_i} = s'_{ID_i} \cdot P$ and returns pk_{ID_i} . Then, adds ($ID_i, s'_{ID_i}, pk_{ID_i}$) to the L_{pk} .

Private Key Extraction Queries: $\mathcal{F}$ maintains a list L_{sk} of tuple ($ID_i, R_{ID_i}, s_{ID_i}, h_{ID_i}$) which is initially empty. For query on input ID_i , $\mathcal{F}$ performs as follows:

If the query ID_i has already appeared on L_{sk} , then the previously defined value is returned. Otherwise, $\mathcal{F}$ generates a random number $r_{ID_i} \in Z_n^*$, compute $R_{ID_i} = r_{ID_i} \cdot P$, queries H_1 oracle to get the value $h_{ID_i} = H_1(ID_i, R_{ID_i})$ and adds the tuple ($ID_i, R_{ID_i}, s_{ID_i}, h_{ID_i}$) to L_{sk} . Then $\mathcal{F}$ makes the public key extraction queries and gets the tuple ($ID_i, s'_{ID_i}, pk_{ID_i}$). If $ID_i = ID_I$, $\mathcal{F}$ outputs "failure". Otherwise, $\mathcal{F}$ sets $sk_{ID_i} = \{d_{ID_i}, s'_{ID_i}\}$ and sends it to $\mathcal{A}2$, where $d_{ID} = \{s_{ID}, R_{ID}\}$. Otherwise, $\mathcal{F}$ makes a partial private key extraction query and a public key extraction query on ID_i , then simulates as the above process and sends $sk_{ID_i} = \{d_{ID_i}, s'_{ID_i}\}$ to $\mathcal{A}2$.and adds ($ID_i, r_{ID_i}, R_{ID_i}, s_{ID_i}, h_{ID_i}$) to L_{H_1} , and sends h_{ID_i} to $\mathcal{A}2$.

***H2*-Queries**: $\mathcal{F}$ maintains a hash list L_{H_2} of tuple ($m_j, R_j, ID_i, pk_{ID_i}, h_j$). When $\mathcal{A}2$ makes *H2* queries for identity ID_i on the message m_j , $\mathcal{F}$ chooses a random value $h_j \in Z_n^*$, sets $h_j = H_2(m_j, R_j, pk_{ID_i})$ and adds ($m_j, R_j, ID_i, pk_{ID_i}, h_j$) to L_{H_2} , and sends h_j to $\mathcal{A}2$.

Signing Queries: When a signing query on (ID_i, m_j) is coming, $\mathcal{F}$ acts as follows:

If $ID_i = ID_I$, $\mathcal{F}$ outputs "failure". Otherwise, $\mathcal{F}$ recovers ($ID_i, R_{ID_i}, s_{ID_i}, h_{ID_i}$) from L_{H_1}

and ($ID_i, s'_{ID_i}, pk_{ID_i}$) from L_{pk} . Then $\mathcal{F}$ gets the secret key $sk_{ID_i} = \{d_{ID_i}, s'_{ID_i}\}$, he can execute the sign algorithm as described in section 3.1. At last, $\mathcal{F}$ response with (R_{ID_i}, R_j, s_j).

Finally, $\mathcal{A}2$ stops and outputs a signature $S^* = \{R_{ID^*}, R_j, s_j\}$ on the message m^* with respect to the public key pk_{ID^*} for the identity ID^*, which satisfies the following equation $Verify(params, ID^*, m^*, pk_{ID^*}, S^*) = 1$.

If $ID_i \neq ID_I$, $\mathcal{F}$ outputs "failure" and aborts. Otherwise, $\mathcal{F}$ recovers the tuple ($ID_i, R_{ID_i}, s_{ID_i}, h_{ID_i}$) from L_{H_1} , the tuple ($ID_i, s'_{ID_i}, pk_{ID_i}$) from L_{pk} and the tuple ($m_j, R_j, ID_i, pk_{ID_i}, h_j$) from L_{H_2} .

Then, we have

$$s_j \cdot (R_j + h_j \cdot P) = pk_{ID_i} + R_{ID_i} + h_{ID_i} P_{pub} \quad (7)$$

From the forgery lemma[12], if we have a replay of $\mathcal{F}$ with the same random tape but different choice of H_2 will output another valid signatures $\{R_{ID_i}, R_j, s'_j\}$. Then we have

$$s'_j \cdot (R_j + h'_j \cdot P) = pk_{ID_i} + R_{ID_i} + h_{ID_i} P_{pub} , \quad (8)$$

When eliminating R_j in the above two equation, we could have

$$(s_j - s'_j) pk_{ID_i} = s_j s'_j (h_j - h'_j) P - (s_j - s'_j) R_{ID_i} - h_{ID_i} (s'_j - s_j) P_{pub} \quad (9)$$

Let $pk_{ID_i} = Q = y \cdot P$, $R_{ID_i} = r_{ID_i} \cdot P$, $P_{pub} = x \cdot P$, then we have

$$(s_j - s'_j) y = s_j s'_j (h_j - h'_j) - (s_j - s'_j) r_{ID_i} - h_{ID_i} (s'_j - s_j) x . \quad (10)$$

Let $u = (s_j - s'_j)^{-1} \bmod n$ and $v = s_j s'_j (h_j - h'_j) - (s_j - s'_j) r_{ID_i} - h_{ID_i} (s'_j - s_j) x$, then, we get $x = uv \bmod n$. According to [12, Lemma 4], the ECDLP can be solved with probability $\varepsilon' \geq 1/9$ and time $t' \leq 23 q_{H_2} t / \varepsilon$.

4. Comparison with previous scheme

In this section, we will compare the efficiency of our new scheme with three latest CLS schemes, i.e. Huang et al.'s scheme [8], Tso et al.'s scheme [9] and Du et al.'s scheme [10]. In the computation efficiency comparison, we obtain the running time for cryptographic operations using MIRACAL [13], a standard cryptographic library.

The hardware platform is a PIV 3-GHZ processor with 512-MB memory and a Windows XP operation system. For the pairing-based scheme, to achieve the 1024-bit RSA level security, we use the Tate pairing defined over the supersingular elliptic curve $E/F_p : y^2 = x^3 + x$ with embedding degree $2 \cdot q$ is a 160-bit Solinas prime $q = 2^{159} + 2^{17} + 1$ and p a 512-bit prime satisfying $p + 1 = 12qr$. For the ECC-based schemes, to achieve the same security level, we employed the parameter secp160r1[14], recommended by the Certicom Corporation, where $p = 2^{160} - 2^{31} - 1$. The running times are listed in Table 1 where sca.mul. stands for scalar multiplication.

Table 1. Cryptographic Operation Time(in milliseconds)

Modular exponentiation	Pairing	Pairing-based sca.mul	ECC-based sca.mul.	Map-to-point hash
5.31	20.04	6.38	2.21	3.04

To evaluate the computation efficiency of different schemes, we use the simple method from [15]. For example, the sign algorithm of our scheme requires one ECC-based scale multiplication; thus, the computation time of the sign algorithm is 2.21 × 1 = 2.21 ms; the verify algorithm has to carry out three ECC-based scalar multiplications, and the resulting running time is 2.21 × 3 = 6.63 ms. As another example, in Huang et al.'s scheme[8], the sign algorithm should carry out a pairing-based scalar multiplications and a map-to-point hash computation; thus, the computation time for a client is 6.38 +3.04= 9.42 ms; the verify algorithm has to carry out three pairing, a map-to-point hash computation , then the resulting running time is 20.04 × 3 + 3.04 = 63.16 ms. Table 2 shows the results of the performance comparison.

Table 2. Performance comparison of different schemes

	Running time	
	Sign	Verify
Huang et al.'s scheme [8]	9.42 ms	63.16 ms
Tso et al.'s scheme [9]	5.31 ms	48.39 ms
Du et al.'s scheme [10]	6.38 ms	26.40 ms
Our scheme	2.21 ms	6.63 ms

According to Table 2, the running time of the sign algorithm of our scheme is 23.46% of Huang et al.'s schemes, 41.62% of Tso et al.'s scheme and 34.64% of Du et al.'s scheme, the running time of the verify algorithm of our scheme is 10.50% of Huang et al.'s schemes, 13.70% of Tso et al.'s scheme and 25.12% of Du et al.'s scheme. Thus our scheme is more useful and efficient than the previous schemes[3-10].

5. Conclusion

In this paper, we have proposed an efficient certificateless signature scheme without bilinear pairings. We also prove the security of the scheme under random oracle. Compared with previous scheme, the new scheme reduces both the running time. Therefore, our scheme is more practical than the previous related schemes for practical application.

6. References

[1]. A. Shamir, Identity-based cryptosystems and signature schemes, Proc. CRYPTO1984, LNCS, vol.196, pp.47–53, 1984.

[2]. S. Al-Riyami, K.G. Paterson, Certificateless public key cryptography, Proceedings of ASIACRYPT 2003, LNCS 2894, Springer-Verlag, 2003, pp. 452 – 473.

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

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

[5]. Z.F. Zhang, D.S. Wong, J. Xu, et al., Certificateless public-key signature: security model and efficient construction, in: J. Zhou, M. Yung, F. Bao (Eds.), ACNS 2006, LNCS 3989, Springer-Verlag, Berlin, 2006, pp. 293 – 308.

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

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

[8]. X. Huang, Yi Mu, W. Susilo, D.S. Wong, Certificateless signature revisited, ACISP 2007, LNCS, vol. 4586, Springer-Verlag, 2007, pp. 308 – 322.

[9]. R. Tso, X. Yi, and X. Huang, Efficient and Short Certificateless Signature, CANS 2008, LNCS 5339, pp. 64–79, 2008.

[10]. H. Du, Q. Wen, Efficient and provably-secure certificateless short signature scheme from bilinear pairings, Computer Standards & Interfaces 31 (2009) 390 – 394.

[11]. L. Chen, Z. Cheng, and N.P. Smart, Identity-based key agreement protocols from pairings, Int. J. Inf. Secur, no.6, pp.213–241, 2007.

[12]. P. David, S. Jacque，Security Arguments for Digital Signatures and Blind Signatures, Journal of Cryptology, Vol. 13, No. 3. p. 361-396, 2000.

[13]. Shamus Software Ltd., Miracl library, http://www.shamus. ie/index.php?page=home.

[14]. The Certicom Corporation, SEC 2: Recommended Elliptic Curve Domain Parameters, www.secg.org/collateral/sec2_final.pdf.

[15]. X. Cao, X. Zeng, W. Kou, and L. Hu, Identity-based anonymous remote authentication for value-added services in mobile networks, IEEE Transactions on Vehicular Technology, vol.58, no.7, pp.3508 - 3517, 2009.