Since certificateless public key cryptosystem can solve the complex certificate management problem in the traditional public key cryptosystem and the key escrow problem in identity-based cryptosystem and the pairing computation is slower than scalar multiplication over the elliptic curve, how to design certificateless signature (CLS) scheme without bilinear pairings is a challenge.

In this paper, we first propose a new pairing-free CLS scheme, and then the security proof is presented in the random oracle model (ROM) under the discrete logarithm assumption. The proposed scheme is more efficient than the previous CLS schemes in terms of computation and communication costs and is more suitable for the applications of low-bandwidth environments.

1. Introduction

In 2003, Al-Riyami and Paterson [1] first introduced the concept of certificateless public key cryptosystem (CL-PKC). The basic idea of CL-PKC is to construct the user's public/private key pair by combining a master key of the key generation center (KGC) with a random secret value generated by the user. Hence, the KGC is unable to compute the user's private key, and each user has one additional random public key and this public key does not need to be certified by a trusted third party in CL-PKC. Thus, CL-PKC not only eliminates the certificates in PKC but also solves the key escrow problem in identity-based public key cryptosystem (ID-PKC). As a classical signature scheme, it should provide existential unforgeability, which ensures that the adversary cannot forge a valid signature. In the formal security model of CLS scheme, two types of adversaries should be considered. Type I adversary is allowed to replace user's public key; however, it cannot access the master key of KGC, while Type II adversary is allowed to know the master key of KGC but cannot replace the target user's public key.

The first CLS scheme was proposed by Al-Riyami and Paterson [1]. Following their works, Huang et al. [2] pointed out that Al-Riyami et al.’s scheme is insecure against Type I adversary. Later, Yum and Lee [3] presented a generic construction of CLS scheme. Hu et al. [4] demonstrated that their scheme is also insecure against Type I adversary. Gorantla and Saxena [5] proposed an efficient CLS scheme, but Cao et al. [6] showed that their scheme is insecure against Type I adversary. Since then, many CLS schemes [7–15] have been proposed. However, only few of them, for example, [10–12], are secure against these two types of adversaries; others are vulnerable to the key replacement attack.

With the rapid development of wireless network technology, more and more users use their mobile devices to deal with the transactions. However, almost all of the above-mentioned schemes cannot be used in the low-bandwidth communication and low-storage and less computation environments. Therefore, many researchers tried to design short CLS schemes. In 2007, Huang et al. [16] proposed the first CLS scheme. After that, Du and Wen [17] and Choi et al. [18] proposed CLS schemes, respectively. Unfortunately, all of them are vulnerable to the key replacement attack launched by a Type I adversary; what is more, they still need bilinear pairing computations. As we know, the computation cost of a pairing is approximately 20 times higher than that of the scalar multiplication over an elliptic curve group [19, 20]. Therefore, how to design a pairing-free CLSS scheme is an attractive topic.

Recently, He et al. [21] proposed an efficient certificateless signature scheme without pairings; Tian and Huang [22] and Tsai et al. [23] pointed out that the scheme cannot withstand...
a Type II adversary's attack. Tsai et al. [23] also proposed an improved new scheme in order to enhance security. Gong and Li [24] pointed out the new scheme is insecure against the super adversary in the random oracle model, and they proposed a real CLS scheme and demonstrated that their scheme is secure against the supper adversary. Yeh et al. [25] proposed a secure certificateless signature scheme without pairings in 2014. In this paper, we propose a new CLS scheme without bilinear pairings, and is provable secure in the random oracle model (ROM) under the discrete logarithm assumption.

The rest of this paper is organized as follows. In Section 2, we present the preliminaries including the elliptic curve, bilinear pairings, some hard problems, and complexity assumptions. A new CLS scheme and the security proof are presented in Sections 3 and 4, respectively. After that, we show the performance comparison among our scheme and other related schemes in Section 5. Finally, we conclude the paper in Section 6.

2. Preliminaries

In this section, we present some definitions and assumptions which are needed in the rest of the paper.

2.1. Elliptic Curve. Let the symbol $E/F_p$ be an elliptic curve $E$ over a prime finite field $F_p$; an equation $y^2 = x^3 + ax + b$, $a, b \in F_p$, with the discriminant $\Delta = 4a^3 + 27b^2 \neq 0$. The point on $E/F_p$ together with an extra point $O$, called the point at infinity, forms a group $G = \{(x, y) : x, y \in F_p, E(x, y) = 0\} \cup \{O\}$. We define $tP = P + P + \cdots + P$ ($t$ times) as scalar multiplication. Let $q$ be the order of $G$.

2.2. Bilinear Pairings. Let $G$ be a cyclic additive group of prime order $q$ and let $G_2$ be a cyclic multiplicative group with the same order $q$; $P$ is a generator. A bilinear pairing is a map $e : G \times G \rightarrow G_2$ with the following properties.

1. Bilinearity: if $P, Q, R \in G$, then $e(P + Q, R) = e(P, R) e(Q, R)$.
2. Nondegeneracy: there exists a $P \in G$ such that $e(P, P) \neq I_{G_2}$, where $I_{G_2}$ is the identity element of $G_2$.
3. Computability: there exists an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in G$.

2.3. Hard Problems and Complexity Assumptions. Discrete logarithm problem (DLP): let $P$ be a generator of group $G$. Given a tuple $(P, xp) \in G$ for $x \in Z_q^\ast$, it is hard to compute $x$.

DL Assumption. There exists no algorithm running in expected polynomial time, which can solve the DLP with nonnegligible probability.

2.4. Security Model. The security model of CLS can be referred to [3, 7, 8]. There are two types of adversaries for a CLS scheme, a Type I adversary $A_1$ and a Type II adversary $A_2$. $A_1$ represents an attacker who is not allowed access to the master key of KGC but he may replace public keys. $A_2$ represents an attacker who is allowed access to the master key of KGC but he cannot replace public keys.

In general, we use two games to define the existential unforgeability of a CLS scheme against a Type I adversary $A_1$ and a Type II adversary $A_2$.

Game 1. A challenger $C$ takes a security parameter $k$ and generates a master private key $s$ and public parameter $\text{params}$ and then sends $\text{params}$ to $A_1$, and keeps $s$ secret. $A_1$ executes the game according to the following steps.

Create($ID$). On input an identity $ID \in \{0, 1\}^*$, if $ID$ has already been created, nothing is to be carried out. Otherwise, $C$ generates the public/private key pair $(PK_{ID}, SK_{ID})$.

Public-Key($ID$). On input an identity $ID$, $C$ outputs the public key $PK_{ID}$ to $A_1$.

Partial-Private-Key-Extract. On input an identity $ID$, $C$ outputs the partial key $D_{ID}$.

Set-Secret-Key. $C$ outputs a master private key $s$ and public parameter $\text{params}$, and then sets $PK_{ID}$ as the new public key of this participant. $C$ will record this replacement, which will be used later.

Sign($ID$, $m$). On input $(ID, m, PK_{ID})$, $C$ uses the private key $(x_{ID}, D_{ID})$ to compute the signature $\sigma$ and returns it to $A_1$. If the public key $PK_{ID}$ has been replaced by $A_1$, then $C$ cannot find $(x_{ID}, D_{ID})$; the answer of the signing oracle may be incorrect. In this situation, we assume that $C$ submits a secret value $r$ corresponding to the replaced public key $PK_{ID}$ to the signing oracle.

At the end, $A_1$ outputs a signature $\sigma$ on the message $m$ corresponding to the public key $PK_{ID}$, for an identity $ID^*$, which is the challenge identity. $A_1$ wins the game if the following conditions hold.

1. Verify$(\text{params}, ID, m, PK_{ID}, \sigma) = 1$.
2. $(ID^*, m)$ has never been submitted to the oracle Sign.
3. $ID^*$ has never been submitted to the Partial-Private-Key-Extract query or Set-Secret-Key query.

If $A_1$ has the advantage at least $\epsilon$ in the above game, runs in time at most $t$, and makes at most $q_C$ Create($ID$) queries, $q_S$ Sign queries, and $q_H$ hash queries, respectively, then $A_1$ is said to be an $(\epsilon, t, q_C, q_S, q_H)$-forger. If there exists no such forger, then a signature scheme is said to be $(\epsilon, t, q_C, q_S, q_H)$-secure against Type I adversary.

Game 2. A challenger $C$ is playing the game with Type II adversary $A_2$.

A challenger $C$ takes a security parameter $k$, generates a master private key $s$ and public parameter $\text{params}$, and then sends $\text{params}$ and $s$ to $A_2$. $C$ answers Create($ID$), Public-Key($ID$), Set-Secret-Key, Partial-Private-Key-Extract, and Sign($ID$, $m$) queries from $A_2$, like does in Game 1.
At the end, $A_2$ outputs a signature $\sigma$ on the message $m$ corresponding to the public key $PK_{ID^*}$ for an identity $ID^*$, which is the challenge identity. $A_2$ wins the game if the following conditions hold.

1. Verify($params$, ID, $m$, PK$_{ID^*}$, $\sigma$) = 1.
2. (ID$^*$, $m$) has never been submitted to the oracle Sign.
3. ID$^*$ has never been submitted to the Set-Secret-Key query.

If $A_2$ has the advantage at least $\epsilon$ in the above game, runs in time at most $t$, and makes at most $q_c$ Create(ID) queries, $q_s$ Sign queries, and $q_h$ hash queries, respectively, then $A_2$ is said to be an $(\epsilon, t, q_c, q_s, q_h)$-forger. If there exists no such forger, then a signature scheme is said to be $(\epsilon, t, q_c, q_s, q_h)$-secure against Type II adversary.

3. The Proposed CLS Scheme

In this section, we propose a new CLS scheme, which consists of seven algorithms: Setup, Partial-Private-Key-Extract, Set-Secret-Key, Set-Private-Key, Set-Public-Key, Sign, and Verify. The details are described as follows.

Setup. On input a security parameter $k$, and return system parameters and master key:

1. KGC generates a cyclic additive group $G$ and a cyclic multiplicative group $G_2$ with the same order $q$.
2. KGC chooses a generator $P \in G$ and two cryptographic secure hash functions $H_1 : \{0, 1\}^* \times G \rightarrow Z_q^*$ and $H_2 : \{0, 1\}^* \rightarrow Z_q^*$.
3. KGC picks a random master private key $x \in Z_q^*$ and computes $P_{pub} = xP$.
4. KGC publishes system parameters $params = \{\varepsilon, P, G, G_2, H_1, H_2, P_{pub}\}$ and keeps $x$ secret.

Partial-Private-Key-Extract. On input $params$, system master key $x$ and a random value $t_A$, user $A$'s identity ID$_A$, compute $R_A = r_AP$, $Q_A = H_1(ID_A, R_A)$, and $D_A = r_A + xQ_A$, output $D_A$ through a secret channel, and publish $R_A$.

Set-Secret-Key. On input the security parameter $k$ and a user's identity ID$_A$, compute $X_A = xAP$ and set $x_A$ as its secret value.

Set-Private-Key. On input $x_A, D_A$ and a user's identity ID$_A$, output the user's private key ($x_A, D_A$).

Set-Public-Key. On input a user's identity ID$_A$, output the user's public key ($X_A, Q_A$).

Sign. On input $params$, ID$_A$, ($x_A, D_A$), and message $m$, perform the following steps.

1. Choose a random number $a \in Z_q^*$ to compute $T_A = aP$.
2. Compute $h = H_2(ID_A||R_A||m||T_A||X_A)$.
3. Compute $s = a/(hx_A + D_A)$.
4. Return $\sigma = (s, h)$ as the signature on the message $m$.

Verify. On input $params$, ID$_A$, $T_A$, $R_A$, ($X_A, Q_A$), $m$, and $s$, compute $Q_A = H_1(ID_A, R_A)$ and $h = H_2(ID_A||R_A||m||T_A||X_A)$. Check whether $H_2(ID_A||R_A||m||s(hX_A + R_A + Q_AP_{pub})||X_A) = h$ holds or not. If the equation holds, output 1, otherwise 0.

We can easily see that the following equation is correct:

$$s \left( hX_A + R_A + Q_AP_{pub} \right) = a \left( hX_A + r_A + xQ_A \right)^{-1} (hX_A + r_A + Q_AxP)$$

Therefore, our CLS scheme is correct.

4. Security Analysis

In this section, we will show that the proposed scheme is secure in the random oracle model under the discrete logarithm assumption.

Theorem 1. If Type I and Type II adversaries can forge a CLS scheme in probabilistic polynomial time $t$ with nonnegligible probability $\epsilon$, then the discrete logarithm problem can be solved with nonnegligible probability $\epsilon' \geq \varepsilon/(q_1^2q_2)$, where $q_i$ ($i = 1, 2$) are denoted by the times of accessing $H_i$ ($i = 1, 2$) oracles, respectively.

Proof. Firstly, we consider Type I attack.

Suppose that there exists a Type I adversary $A_1$ which has a nonnegligible probability $\epsilon$ in attacking our CLS scheme; we construct a challenger $C$ that uses $A_1$ to solve the DLP. Challenger $C$ receives a DL instance $y = vP$ for randomly chosen $v \in Z_q^*$ and $P \in G$ and wants to compute $v$. $C$ runs $A_1$ as a subroutine and simulates its attack situations. $C$ sets $P_{pub} = y$, where $v$ is the master key, which is unknown to $C$, and returns system parameters to $A_1$. $C$ maintains initially empty lists $L_{C}, L_{H_1}, L_{H_2}, L_{D}, L_{SK}, L_{PK}, L_{S}$, and $L_{V}$ in order to simulate the oracle queries of $A_1$ as follows.

Create(ID$_D$). $C$ maintains list $L_C$ of tuple (ID, $R_i$, $P_i$, $D_i$, $x_i$, $h_i$). $C$ responds with (ID, $R_i$, $P_i$, $D_i$, $x_i$, $h_i$) if ID$_D$ is on $L_C$. Otherwise, it chooses three random values $a_i, b_i, x_i \in Z_q^*$, sets $R_i = a_iP + b_iP_{pub}$, $D_i = a_i$, $h_i = H_1(ID_i, R_i)$, $h_i = b_i$, and PK$_i = x_iP$, responds with (ID, $R_i$, PK$_i$, $D_i$, $x_i$, $h_i$), and inserts (ID$_D$, $h_i$) into $L_{H_i}$. It is known that (ID$_D$, $R_i$, $h_i$) satisfies the equation $D_iP = R_i + h_iP_{pub}$ in the Partial-Private-Key-Extract algorithm.

$H_i$ Queries. Suppose that $A_1$ makes at most $q_i$ queries to the oracle $H_i$. $C$ chooses a random number $j \in [1, q_i]$. When it makes a $H_i$ query on ID$_D$ where $1 \leq i \leq q_1$, if $j = i$ (we let ID$_D$ = ID$^*$ at this point), $C$ randomly chooses $u \in Z_q^*$ and returns $u$ and then adds (ID$_D$, $R_i$) to $L_{H_i}$. Otherwise, $C$ picks a random number $h_i \in Z_q^*$, returns $h_i$ to $A_1$, and adds (ID$_D$, $R_i$, $h_i$) to $L_{H_i}$.
H2 Queries. Suppose that A2 makes at most q2 queries to the oracle H2. If the list L_{H2} contains \langle m, r, x, t, ID, h \rangle, C returns h, otherwise, C picks a random number \( h_1 \in \mathbb{Z}_q^{*} \), returns \( h_1 \), and adds \( \langle m, r, x, t, ID, h_1 \rangle \) to L_{H2}.

Request-Public-Key(ID\_2) Queries. Suppose that A2 makes this query to the Public-Key-Request oracle. C looks up the list L_{PK}. C randomly chooses PK_2 = bP and then adds \( \langle ID_2, ID, x_2, \perp \rangle \) to L_{H2}. Otherwise, C picks a random number \( r_1 \in \mathbb{Z}_q^{*} \), computes \( X_1 = x_1P \), and adds \( \langle ID_2, PK_2, x_1 \rangle \) to L_{PK}.

Partial-Private-Key-Extract(ID\_2) Queries. When A1 makes this query, C does the following steps. If \( ID_2 = ID^* \), C terminates the session. Otherwise, C looks up L_{H2} for the tuple \( \langle ID_2, R_2 \rangle \). If there exists such a tuple, C returns \( D_2 \) to A1. Otherwise, C makes Replace-Public-Key queries on itself and returns D as the response.

Extract-Secret-Key(ID\_2) Queries. C picks a random \( x_i \in \mathbb{Z}_q^{*} \), computes \( X_i = x_iP \), returns \( x_i \) to A1, and adds \( \langle ID_2, X_i, x_i \rangle \) to L_{SK}.

Replace-Public-Key(ID\_2, PK\_2') Queries. If the list L_{PK} contains \( \langle ID_2, PK_2, x_2 \rangle \), C sets \( PK_2 = PK_2' \), \( x_2 = \perp \). Otherwise, C makes Extract-Secret-Key(ID\_2) query on ID\_2 and then sets \( PK_2 = PK_2' \), \( x_2 = \perp \), and adds \( \langle PK_2, x_2, \perp \rangle \) to A1.

Sign(m, ID\_2) Queries. If the list L_{H2} contains \( \langle ID_2, Q, r_2 \rangle \) and the list L_{PK} contains \( \langle ID_2, PK_2, x_2 \rangle \), C does the following.

If \( ID_2 = ID^* \), then C picks three random numbers \( a, x_i, r_1, r_2, x_2 \in \mathbb{Z}_q^{*} \), computes \( T = aP, R = r_1P, X = x_1P, h = H_2(T||X||R||ID_2||m), D_1 = r_1 + vH_1(ID_2, R) = r_1 + v_1h \), and \( s = a/(hX_1 + r_i + v_1h) \), returns \( (h, s) \) to A1, and adds \( \langle m, PK_2, ID_2, r_2, P \rangle \) and \( \langle m, PK_2, ID_2, h \rangle \) to L_{H2}, and L_{H2}' respectively.

The public key PK_2 may be replaced by A_1. The following equation holds because the signature is valid:

\[
\begin{align*}
s \left( hX_1 + r_1P + h_1y \right) &= a \left( hX_1 + r_1 + v_1h \right)^{-1} \left( hX_1P + r_1P + h_1vP \right) = aP = T. \tag{2}
\end{align*}
\]

C computes \( aP - shX_1P - vr_1P)/sh_1 = vP \).

Note that C can solve the DL problem because he knows \( a, s, h, x_i, r_1, P \). Thus, we have \( \epsilon' \geq \epsilon'(q_2^2) \).

Then, we consider Type II attack.

Suppose that there exists a Type II adversary A2 which has a nonnegligible probability \( \epsilon \) in attacking our GLS scheme; we construct a challenger C that uses A2 to solve the DLP challenger C receives a DL instance \( (vP, v) \) for randomly chosen \( v \in \mathbb{Z}_q^{*} \) and \( P \in G \) and wants to compute \( v \). C runs A2 as a subroutine and simulates its attack situation. C sets \( v = vP \), where \( v \) is the master key, and returns system parameters and \( v \) to A2. C maintains initially empty lists L_{C}, L_{H1}, L_{H2}, L_{L2}, L_{PK}, L_{SK}, and L_{V} in order to simulate the oracle queries of A2 as follows.

Create(ID\_2). C maintains list L_{C} of tuple \( \langle ID, R_2, PK_2, D_2, x_i, h \rangle \). C responds with \( \langle ID, R_2, PK_2, D_2, x_i, h \rangle \) if \( ID_2 \) is on L_{C}. Otherwise, if \( ID_2 = ID^* \), C chooses three random values \( r_1, r_2, x_2 \in \mathbb{Z}_q^{*} \) and sets \( R_2 = r_1P, h = H_2(ID_2, R_2), h_1 = b_1, D_1 = r_1 + b_1s \) mod \( q_2 \), and \( P_{pub} = sP, x_1 = \perp \). If \( ID_2 \neq ID^* \), C chooses three random values \( x_1, b_1, r_1, r_2 \in \mathbb{Z}_q^{*} \) and sets \( R_2 = r_1P, h = H_2(ID_2, R_2), h_1 = b_1, D_1 = r_1 + b_1s \) mod \( q_2 \), and \( P_{pub} = sP, \) and \( PK_2 = x_1P \); C responds with \( \langle ID, R_2, PK_2, D_2, x_i, h \rangle \) and inserts \( \langle ID, R_2, h_1 \rangle \) into L_{H2}.

H2 Queries. Suppose that A2 makes at most \( q_2 \) queries to the oracle H2. C chooses a random \( j \in [1, q_2] \). When it makes a H2 query on ID\_2 where \( 1 \leq i \leq q_2 \), if \( i = j \) (we let ID\_2 = ID^* at this point), C randomly chooses \( r_i \in \mathbb{Z}_q^{*} \) and returns \( r_1 \) and then adds \( \langle ID_2, R_2, r_i \rangle \) to L_{H2}. Otherwise, C picks a random number \( h_1 \in \mathbb{Z}_q^{*} \), returns \( h_1 \) to A1, and adds \( \langle ID_2, R_2, h_1 \rangle \) to L_{H2}.

H2 Queries. Suppose that A2 makes at most \( q_2 \) queries to the oracle H2. If the list L_{H2} contains \( \langle m, R_2, T, ID, h \rangle \), C returns \( h \). Otherwise, C picks a random number \( h_1 \in \mathbb{Z}_q^{*} \), returns \( h_1 \), and adds \( \langle m, R_2, T, ID, h_1 \rangle \) to L_{H2}.

Request-Public-Key(ID\_2) Queries. Suppose that A2 makes at most \( q_2 \) queries to the Public-Key-Request oracle. C chooses a random \( j \in [1, q_2] \). If \( i = j \) (we let ID\_2 = ID^* at this point), C randomly chooses \( PK_2 = bP \) and then adds \( \langle R_2, PK_2, x_2 \rangle \) to L_{H2}. Otherwise, C picks a random number \( r_i \in \mathbb{Z}_q^{*} \), returns \( h_1 \) to A1, and adds \( \langle ID, PK_2, x_1 \rangle \) to L_{PK}.

Partial-Private-Key-Extract(ID\_2) Queries. When A2 makes this query, C looks up the tuple \( \langle ID_2, R_2, D_2 \rangle \). If it is the tuple, then C returns \( D_2 \) to A2. Otherwise, C makes the Request-Public-Key(ID\_2) queries and returns \( D_2 \) to A2.

Extract-Secret-Key(ID\_2) Queries. C picks a random \( x_i \in \mathbb{Z}_q^{*} \), computes \( X_i = x_iP \), returns \( x_i \) to A1, and adds \( \langle ID_2, X_i, x_i \rangle \) to L_{SK}. Otherwise, C aborts the simulation.

Sign(m, ID\_2) Queries. If the list L_{H2} contains \( \langle ID_2, Q, r_2 \rangle \) and the list L_{PK} contains \( \langle ID_2, PK_2, x_2 \rangle \), C does the following.

If \( ID_2 = ID^* \), then C picks three random \( a, x_i, r_1, r_2, x_2 \in \mathbb{Z}_q^{*} \), computes \( T = aP, X = x_1P, h = H_2(T||X||R||ID_2||m), D_1 = r_1 + vH_1(ID_2, R) = r_1 + v_1h \), and \( s = a/(hX_1 + r_i + v_1h) \), returns \( (h, s) \) to A1, and adds \( \langle m, PK_2, ID_2, r_2, P \rangle \) and \( \langle m, PK_2, ID_2, h \rangle \) to L_{H2} and L_{H2}' respectively. The public key PK_2 may be replaced by A_1. The following equation holds because the signature is valid:

\[
\begin{align*}
s \left( hX_1 + r_1P + h_1y \right) &= a \left( hX_1 + r_1 + v_1h \right)^{-1} \left( hX_1P + r_1P + h_1vP \right) = aP = T. \tag{3}
\end{align*}
\]

C computes \( aP - shX_1P -vr_1P)/sh_1 = vP \).

Note that C can solve the DL problem because he knows \( a, s, h, x_i, bP \). Thus, we have \( \epsilon' \geq \epsilon'(q_2^2) \).
Table 1: Performance comparisons.

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Computational cost</th>
<th>Signature size (bytes)</th>
<th>Secure against Type I</th>
<th>Secure against Type II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choi et al. [18]</td>
<td>1M</td>
<td>1M</td>
<td>64</td>
<td>Yes</td>
</tr>
<tr>
<td>He et al. [21]</td>
<td>1M</td>
<td>1M</td>
<td>64</td>
<td>Yes</td>
</tr>
<tr>
<td>Tsai et al. [23]</td>
<td>1M</td>
<td>1M</td>
<td>64</td>
<td>Yes</td>
</tr>
<tr>
<td>Gong and Li [24]</td>
<td>1M</td>
<td>1M</td>
<td>64</td>
<td>Yes</td>
</tr>
<tr>
<td>Yeh et al. [25]</td>
<td>1M</td>
<td>1M</td>
<td>64</td>
<td>Yes</td>
</tr>
<tr>
<td>Our scheme</td>
<td>1M</td>
<td>1M</td>
<td>64</td>
<td>Yes</td>
</tr>
</tbody>
</table>

5. Performance Comparison

In order to achieve 1024-bit RSA level security, we used the Tate pairing defined over the supersingular elliptic curve $E/F_p : y^2 = x^3 + x$ with embedding degree 2. $q$ is a 160-bit Solinas prime $q = 2^{159} + 2^{17} + 1$ and $p$ is a 512-bit prime such as $p + 1 = 12q^r$. The signature of Chen et al. [20] consists of a point of elliptic curve $E/F_{p^2} : y^2 = x^3 + x$; then, the signature size is $512/8 = 64$ bytes. In order to achieve the same security level, we use the ECC group on Koblitz elliptic curve $y^2 = x^3 + x$ defined on $F_{2^{163}}$ with $b = 163$ bit random prime. The signature size of our scheme is $f(163 + 163/8) = 41$ bytes. The performance comparison among the proposed scheme and some related CLS schemes is given in Table 1. A pairing operation is denoted by $P$, scalar multiplication in the group $G$ by $M$, a modular exponentiation in $G_2$ by $E$, and pairing-based scalar multiplication by PM. Sign and Ver denote the computational costs required for signing and verification processes of CLS scheme. According to Table 1, it is known that our scheme is more efficient than the other CLS schemes.

6. Conclusion

In this paper, we proposed a new CLS scheme without pairings and also showed that the proposed scheme is secure in the random oracle model under the DL assumption. The proposed scheme is more efficient than the previous CLS schemes in terms of computation and communication costs and is more suitable for the applications of low-bandwidth environments.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is partially supported by the Major State Basic Research Development (973) Program of China (no. 2013CB834205), the National Natural Science Foundation of China (nos. 61070153 and 61103209), Natural Science Foundation of Zhejiang Province (no. LZ12F02005), and Education Department Foundation of Zhejiang Province (no. Y201222977).

References


Submit your manuscripts at http://www.hindawi.com