# 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 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,

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

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

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

$$y^2 = x^3 + ax + b\ ,\quad a,b \in F_p \qquad (1)$$

and with the discriminant

$$\Delta = 4a^3 + 27b^2 \neq 0\ . \qquad (2)$$

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

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

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 $x$.

# 3. Our scheme

## 3.1.Scheme Description

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*. Our scheme also consists of seven algorithms. These algorithms are described as follows.

**Setup:** This algorithm takes a security parameter $k$ as in put, and returns system parameters and a master key. Given $k$ , KGC does the following.

1) KGC chooses a $k$ -bit prime $p$ and determines the tuple $\{F_p, E/F_p, G, P\}$ as defined in Section 2.

2) KGC chooses the master private key $x \in Z_n^*$ and computes the master public key $P_{pub} = xP$ .

3) KGC chooses two cryptographic secure hash functions $H_1 : \{0,1\}^* \to Z_n^*$ and $H_2 : \{0,1\}^* \to Z_n^*$.

4) KGC publishes $params = \{F_p, E/F_p, G, P, P_{pub}, H_1, H_2\}$ as system parameters and secretly keeps the master key $s$ .

**Set-Secret-Value**: The user with identity $ID$ picks randomly $x_{ID} \in Z_n^*$ , computes $P_{ID} = x_{ID} \cdot P$ and sets $x_{ID}$ as his secret value.

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

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

2) KGC computes $s_{ID} = r_{ID} + h_{ID} x \bmod n$ and issues $\{s_{ID}, R_{ID}\}$ to the users through secret channel.

The user's s partial private key is the tuple $s_{ID}$ 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-Private-Key**: The user with identity $ID$ takes the pair $sk_{ID} = (x_{ID}, s_{ID})$ as its private key.

**Set-Public-Key**: The user with identity $ID$ takes $pk_{ID} = \{P_{ID}, R_{ID}\}$ as its public key.

**Sign**: This algorithm takes system parameters, user's private key $sk_{ID} = (x_{ID}, s_{ID})$, 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 $h = H_2(m, R, P_{ID}, R_{ID})$.

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

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

5) The resulting signature is ($R, s$).

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

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

Accept if it is equal. Otherwise reject.

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

$$\begin{aligned} s \cdot (R + h \cdot P) &= (l + h)^{-1}(x_{ID} + s_{ID}) \cdot (l \cdot P + h \cdot P) \\ &= (l + h)^{-1}(x_{ID} + s_{ID}) \cdot (l + h) \cdot P = (x_{ID} + s_{ID}) \cdot P \\ &= x_{ID} \cdot P + s_{ID} \cdot P = P_{ID} + R_{ID} + h_{ID} \cdot P_{pub} \end{aligned} \tag{6}$$

Then the correctness of our scheme is proved.

## 3.2.Security Analysis

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

***Type 1 Adversary***: Adversary $\mathcal{A}1$ does not have access to the master key, but $\mathcal{A}1$ 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 $\mathcal{A}2$ has access to the master key, but cannot replace any user's public key.

Let $\mathcal{A}1$ and $\mathcal{A}2$ be a Type1Adversaryanda Type2Adversary, respectively. We consider two games Game 1 and Game 2 where $\mathcal{A}1$ and $\mathcal{A}2$ interact with its challenger in these two games, respectively.

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

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

$Create(ID)$: This allows $\mathcal{A}$ to ask $\mathcal{C}$ to set up a new participant with identity $ID$. On receiving such a query, $\mathcal{C}$ generates the public/private key pair.

$Public-Key(ID)$: $\mathcal{A}$ can request the public key of a participant whose identity is $ID$. In response, $\mathcal{C}$ outputs the public key $pk_{ID}$.

$Partial\text{-}Private\text{-}Key-Extract(ID)$: $\mathcal{A}$ can request the partial private key of a participant whose identity is $ID$. In response, $\mathcal{C}$ outputs the partial private key $s_{ID}$.

$Secret\text{-}Key-Extract(ID)$: $\mathcal{A}$ can request the private key of a participant whose identity is $ID$. In response, $\mathcal{C}$ outputs the private key $s_{ID}$.

$Public-Key-Replacement(ID, pk'_{ID})$: For a participant whose identity is $ID_i$, $\mathcal{A}$ can choose a new public key $pk'_{ID}$ and then set $pk'_{ID}$ as the new public key of this participant. $\mathcal{C}$ will record these replacements which will be used later.

$Sign(ID, m)$: 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. $\mathcal{A}1$ wins the game if the following conditions hold:

- $Verify(params, ID, m, pk_{ID^*}, S) = 1$
- （$ID^*, m$）has never been submitted to the oracle $Sign$.

- $ID^*$ has never been submitted to $Partial\text{-}Private\text{-}Key-Extract$ query and $Secret\text{-}Key-Extract$ query.

An adversary $\mathcal{A}1$ is said to be an $(\varepsilon, t, q_c, q_s, q_h)$-forger if it has advantage at least $\varepsilon$ in the above game, runs in time at most $t$, and make at most $q_c$, $q_s$ and $q_h$ $Create$, $Sign$ and random oracle queries, respectively. A scheme is said to be $(\varepsilon, t, q_c, q_s, q_h)$-secure against $\mathcal{A}1$ in the sense of unforgeable against chosen message attack if no $(\varepsilon, t, q_c, q_s, q_h)$-forger exists.

**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 $\mathcal{A}2$. $\mathcal{C}$ answers $Create(ID)$, $Public-Key(ID)$, $Secret\text{-}Key-Extract(ID)$, $Partial\text{-}Private\text{-}Key-Extract(ID)$ and $Sign(ID, m)$ from $\mathcal{A}2$ like he does in **Game 1**.

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:

- $Verify(params, ID^*, m, pk_{ID^*}, S) = 1$
- $(ID^*, m)$ has never been submitted to the oracle $Sign$.
- $ID^*$ has never been submitted to $Secret\text{-}Key-Extract$ query.

An Type 2 adversary $\mathcal{A}2$ is said to be an $(\varepsilon, t, q_c, q_s, q_h)$-forger if it has advantage at least $\varepsilon$ in the above game, runs in time at most $t$, and make at most $q_c$, $q_s$ and $q_h$ $Create$, $Sign$ and random oracle queries, respectively. A scheme is said to be $(\varepsilon, t, q_c, q_s, q_h)$-secure against $\mathcal{A}2$ in the sense of unforgeable against chosen message attack if no $(\varepsilon, t, q_c, q_s, q_h)$-forger exists.

We prove the security of our scheme 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 , the following theorems are provided.

**Theorem 1**. The proposed scheme is ( $\varepsilon, t, q_c, q_s, q_h$ )-secure against the adversary $\mathscr{A}1$ in the random oracle model, assuming that the ( $\varepsilon', t'$ )-ECDL assumption holds in $G$ , where $t' = t + O(q_c + q_h)S$ , $\varepsilon' = (1 - \frac{q_h q_c)}{n})(1 - \frac{1}{q})(\frac{1}{q_h})\varepsilon$ and $q_c$ , $q_s$ , $q_h$ are the number of $Create$ , $Sign$ and hashing queries respectively the adversary is allowed to make and $S$ is the time for an scale multiplication operation.

***Proof***: Suppose that there is a type 1 Adversar $\mathscr{A}1$ for an adaptively chosen message attack against our scheme. Then, we show how to use the ability of $\mathscr{A}1$ 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 $x \in Z_n^*$ satisfying $Q = x \cdot P$ . To do so, $\mathscr{F}$ picks an identity $ID^*$ 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 $\mathscr{F}$ answers $\mathscr{A}1$'s queries as follows.

$Create(ID)$ : $\mathscr{F}$ maintains a hash list $L_C$ of tuple ( $ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID}$ ). If $ID$ is on $L_C$ , then $\mathscr{F}$ response with ( $ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID}$ ). Otherwise, $\mathscr{F}$ simulates the oracle as follows. It chooses $a, b, c \in Z_n^*$ at random, sets $R_{ID} = a \cdot P_{pub} + b \cdot P$ , $P_{ID} = c \cdot P$ , $s_{ID} = b$ , $x_{ID} = c$ , $h_{ID} = H_1(ID, R_{ID}, P_{ID}) \leftarrow -a \bmod n$ , response with ( $ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID}$ ), inserts ( $ID, R_{ID}, P_{ID}, h_{ID}$ ) into $L_{H_1}$ . Note that ( $R_{ID}, s_{ID}, h_{ID}$ ) 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.

$H_1 - query$ : $\mathscr{F}$ maintains a hash list $L_{H_1}$ of tuple ( $ID, R_{ID}, P_{ID}, h_{ID}$ ) as explained below. The list is initially empty. When $\mathscr{A}1$ makes a hash oracle query on $ID$ , if the query $ID$ has already appeared on $L_{H_1}$ , then the previously defined value is returned. Otherwise, $\mathscr{F}$ queries $Create(ID_i)$ , gets ( $ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID}$ ) and response with $h_{ID}$ .

$Partial\text{-}Private\text{-}Key - Extract(ID)$ : If $ID = ID^*$ , $\mathscr{F}$ stop the simulation. Otherwise, $\mathscr{F}$ looks up the table $L_C$ . If $ID$ is on $L_C$ , then $\mathscr{F}$ response with $s_{ID}$ . Otherwise, $\mathscr{F}$ queries

$Create(ID)$, gets $(ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID})$. If $ID = ID^*$, $\mathcal{F}$ stops the simulation. Otherwise, $\mathcal{F}$ response with $s_{ID}$.

$Public-Key(ID)$: $\mathcal{F}$ looks up the table $L_C$. If $ID$ is on $L_C$, then $\mathcal{F}$ response with $pk_{ID} = \{R_{ID}, P_{ID}\}$. Otherwise, $\mathcal{F}$ queries $Create(ID)$ with $ID$, gets $(ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID})$ and response with $pk_{ID} = \{R_{ID}, P_{ID}\}$.

$Secret\text{-}Key-Extract(ID)$: If $ID = ID^*$, $\mathcal{F}$ stop the simulation. Otherwise, $\mathcal{F}$ looks up the table $L_C$. If $ID$ is on $L_C$, then $\mathcal{F}$ response with $x_{ID}$. Otherwise, $\mathcal{F}$ queries $Create(ID)$, gets $(ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID})$ and response with $x_{ID}$.

$Public-Key-Replacement(ID, pk'_{ID})$: $\mathcal{F}$ maintains a hash list $L_R$ of tuple $(ID, r_{ID}, R_{ID}, x_{ID}, P_{ID})$, which is initialized empty. When $\mathcal{A}1$ queries on input $(ID, pk'_{ID})$, where $R'_{ID} = r'_{ID} \cdot P$, $P'_{ID} = x'_{ID} \cdot P$ and $pk'_{ID} = (R'_{ID}, P'_{ID})$, $\mathcal{F}$ sets $R_{ID} = R'_{ID}$, $P_{ID} = P'_{ID}$, $s_{ID} = \perp$ and $x_{ID} = x'_{ID}$. At last, $\mathcal{F}$ adds $(ID, r'_{ID}, R'_{ID}, x'_{ID}, P'_{ID})$ to $L_R$.

$H_2-query$: $\mathcal{F}$ maintains a hash list $L_{H_2}$ of tuple $(m, R, ID, P_{ID}, R_{ID}, h)$. When $\mathcal{A}1$ makes *H2* queries for identity $ID$ on the message $(m, R, ID, P_{ID}, R_{ID})$, $\mathcal{F}$ chooses a random value $h \in Z_n^*$, sets $h = H_2(m, R, ID, P_{ID}, R_{ID})$ and adds $(m, R, ID, P_{ID}, R_{ID}, h)$ to $L_{H_2}$, and sends $h$ to $\mathcal{A}1$.

$Sign(ID, m)$: When a signing query on $(ID, m)$ is coming, $\mathcal{F}$ looks up the list $L_R$. If $ID$ is on $L_R$, $\mathcal{F}$ generates random numbers $a, b, c \in Z_n^*$, sets $s = a$, $R = a^{-1} h_{ID} P_{pub}$, $h = H_2(m, R, P_{ID}, R_{ID}) \leftarrow a^{-1}(r_{ID} + x_{ID})$, insert $(m, R, P_{ID}, R_{ID}, h)$ to $L_{H_2}$ and outputs $(R, s)$ as the signature. Otherwise, $\mathcal{F}$ acts like the description of the scheme, since $\mathcal{F}$ knows the private key of the user with identity $ID$.

Finally, $\mathcal{A}1$ stops and outputs a signature $S = \{R, s^{(1)}\}$ 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 $ID \neq ID^*$, $\mathcal{F}$ outputs "failure" and aborts. Otherwise, $\mathcal{F}$ recovers the tuple $(ID, R_{ID}, P_{ID}, h_{ID})$ from $L_{H_1}$, the tuple $(ID, s'_{ID}, pk_{ID})$ from $L_{pk}$ and the tuple $(m, R, ID, P_{ID}, R_{ID}, h^{(1)})$ from $L_{H_2}$.

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 two valid signatures $\{R, s^{(2)}\}$ and $\{R, s^{(3)}\}$. Then we have

$$s^{(i)} \cdot (R + h^{(i)} \cdot P) = P_{ID} + R_{ID} + h_{ID} P_{pub}, i = 1, 2, 3, \quad (7)$$

By $l, x_{ID}, r_{ID}, x$, we now denote discrete logarithms of $R$, $P_{ID}$, $R_{ID}$ and $P_{pub}$ respectively, i.e., $R = lP$, $P_{ID} = x_{ID}P$, $R_{ID} = r_{ID}P$, $P_{pub} = xP$ and.

$$s^{(i)} \cdot (l + h^{(i)} \cdot P) = x_{ID} + r_{ID} + h_{ID} x, i = 1, 2, 3 \quad (8)$$

In these equations, only $l, r_{ID}, x$ are unknown to $\mathcal{F}$. $\mathcal{F}$ solves for these values from the above three linear independent equations, and outputs $x$ as the solution of the discrete logarithm problem.

**Reduction Cost Analysis**: The simulation of the $Create$ oracle fails if the random oracle assignment $H_1(ID, R_{ID}, P_{ID})$ causes inconsistency. It happens with probability at most $\frac{q_h}{n}$.Hence, the simulation is successful $q_c$ times with probability at least $(1-\frac{q_h}{n})^{q_c} \geq 1-\frac{q_h q_c}{n}$. Due to the ideal randomness of the random oracle, there exists a query $H_2(m, R, ID, P_{ID}, R_{ID})$ with probability at least $1-\frac{1}{n}$. $B$ guesses it correctly as the point of rewind, with probability at least $\frac{1}{q_h}$. Thus, the overall successful probability is

$$(1-\frac{q_h q_c}{n})(1-\frac{1}{n})(\frac{1}{q_h})\varepsilon .$$

The time complexity of $\mathcal{F}$ is dominated by the exponentiations performed in the $Create$ and $Sign$ queries, which is equal to $t + O(q_c + q_h)S$.

**Theorem 2**. The proposed scheme is $(\varepsilon, t, q_c, q_s, q_h)$-secure against the adversary $\mathcal{A}$ 2 in the random oracle model, assuming that the $(\varepsilon', t')$-ECDL assumption holds in $G$, where

$t' = t + O(q_c + q_h)S$, $\varepsilon' = (1 - \frac{q_h q_c)}{n})(1 - \frac{1}{q})(\frac{1}{q_h})\varepsilon$ and $q_c$, $q_s$, $q_h$ are the number of $Create$, $Sign$ and hashing queries respectively the adversary is allowed to make and $S$ is the time for an scale multiplication operation.

***Proof***: Suppose that there is a type 2 Adversar $\mathscr{A}2$ for an adaptively chosen message attack against our scheme. 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 $ID^*$ 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 $\mathscr{A}2$s queries as follows.

$Create(ID)$: $\mathscr{F}$ maintains a hash list $L_C$ of tuple ( $ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID}$ ). If $ID_i$ is on $L_C$, then $\mathscr{F}$ response with ( $ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID}$ ). Otherwise, $\mathscr{F}$ simulates the oracle as follows. If $ID = ID^*$, $\mathscr{F}$ chooses $a, b \in Z_n^*$ at random, sets $R_{ID} = aP$, $P_{ID} = Q$, $h_{ID} = H_1(ID, R_{ID}, P_{ID}) \leftarrow b$, $s_{ID} \leftarrow a$, $x_{ID} \leftarrow \perp$. If $ID \neq ID^*$, $\mathscr{F}$ chooses $a, b, c \in Z_n^*$ at random, sets $R_{ID} = a \cdot P$, $P_{ID} = b \cdot P$, $h_{ID} = H_1(ID, R_{ID}, P_{ID}) \leftarrow c$, $s_{ID} = a + x \cdot h_{ID}$, $x_{ID} = b$. At last $\mathscr{F}$ response with ( $ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID}$ ), inserts ( $ID, R_{ID}, P_{ID}, h_{ID}$ ) into $L_{H_1}$.

$H_1 - query$: $\mathscr{F}$ maintains a hash list $L_{H_1}$ of tuple ( $ID, R_{ID}, P_{ID}, h_{ID}$ ) as explained below. The list is initially empty. When $\mathscr{A}1$ makes a hash oracle query on $ID$, if the query $ID$ has already appeared on $L_{H_1}$, then the previously defined value is returned. Otherwise, $\mathscr{F}$ queries $Create(ID)$, gets ( $ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID}$ ) and response with $h_{ID}$.

$Partial - Private - Key - Extract(ID)$: $\mathscr{F}$ looks up the table $L_C$. If $ID$ is on $L_C$, then $\mathscr{F}$ response with $s_{ID}$. Otherwise, $\mathscr{F}$ queries $Create(ID)$, gets ( $ID, R_{ID}, P_{ID}, s_{ID}, x_{ID}, h_{ID}$ ) and response with $s_{ID}$.

$Public-Key(ID)$ : $\mathcal{F}$ looks up the table $L_C$. If $ID$ is on $L_C$, then $\mathcal{F}$ response with $pk_{ID}=\{R_{ID},P_{ID}\}$. Otherwise, $\mathcal{F}$ queries $Create(ID)$, gets $(ID,R_{ID},P_{ID},s_{ID},x_{ID},h_{ID})$ and response with $pk_{ID}=\{R_{ID},P_{ID}\}$.

$Secret\text{-}Key-Extract(ID)$: If $ID_i=ID^*$, $\mathcal{F}$ stop the simulation. Otherwise, $\mathcal{F}$ looks up the table $L_C$. If $ID$ is on $L_C$, then $\mathcal{F}$ response with $x_{ID}$. Otherwise, $\mathcal{F}$ queries $Create(ID)$, gets $(ID,R_{ID},P_{ID},s_{ID},x_{ID},h_{ID})$ and response with $x_{ID}$.

$H_2-query$ : $\mathcal{F}$ maintains a hash list $L_{H_2}$ of tuple $(m,R,ID,P_{ID},R_{ID},h)$. When $\mathcal{A}2$ akes *H2* queries for identity $ID$ on the message $(m,R,ID,P_{ID},R_{ID})$, $\mathcal{F}$ chooses a random value $h\in Z_n^*$, sets $h_j=H_2(m,R,ID,P_{ID},R_{ID})$ and adds $(m,R,ID,P_{ID},R_{ID},h)$ to $L_{H_2}$, and sends $h$ to $\mathcal{A}2$

$Sign(ID,m)$: When a signing query on $(ID,m)$ is coming , $\mathcal{F}$ acts like the description of the scheme if $ID\neq ID^*$, since $\mathcal{F}$ knows the private key of the user with identity $ID$. If $ID=ID^*$, $\mathcal{F}$ chooses random numbers $a,b\in Z_n^*$, sets $s=a_i$, $R=a^{-1}(R_{ID}+P_{ID})$, $h=H_2(m,R,ID,P_{ID},R_{ID})\leftarrow s^{-1}h_{ID}x$, and response with $(R,s)$ as the signature. It easy to verify the signature is legal since $s\cdot(R+h\cdot P)=P_{ID}+R_{ID}+h_{ID}P_{pub}$.

Finally, $\mathcal{A}2$ stops and outputs a signature $S=\{R,s^{(1)}\}$ 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 $ID\neq ID^*$, $\mathcal{F}$ outputs “failure” and aborts. Otherwise, $\mathcal{F}$ recovers the tuple $(ID,R_{ID},P_{ID},h_{ID})$ from $L_{H_1}$ and the tuple $(m,R,ID,P_{ID},R_{ID},h^{(1)})$ from $L_{H_2}$.

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 $\{R,s^{(2)}\}$. Then we have

$$s^{(i)}\cdot(R+h^{(i)}\cdot P)=P_{ID}+R_{ID}+h_{ID}P_{pub},i=1,2, \qquad (7)$$

By $l, y, r_{ID}, x$, we now denote discrete logarithms of $R$, $P_{ID_i}$, $R_{ID_i}$ and $P_{pub}$ respectively, i.e., $R = lP$, $Q = P_{ID} = yP$, $R_{ID} = r_{ID}P$, $P_{pub} = xP$ and.

$$s^{(i)} \cdot (l + h^{(i)} \cdot P) = y + r_{ID} + h_{ID}x, i = 1, 2 \quad (8)$$

In these equations, only $l, y$ are unknown to $\mathcal{F}$. $\mathcal{F}$ solves for these values from the above three linear independent equations, and outputs $y$ as the solution of the discrete logarithm problem.

**Reduction Cost Analysis**: The simulation of the $Create$ oracle fails if the random oracle assignment $H_1(ID, R_{ID}, P_{ID})$ causes inconsistency. It happens with probability at most $\frac{q_h}{n}$.Hence, the simulation is successful $q_c$ times with probability at least $(1-\frac{q_h}{n})^{q_c} \geq 1-\frac{q_h q_c}{n}$. Due to the ideal randomness of the random oracle, there exists a query $H_2(m, R, ID, P_{ID}, R_{ID})$ with probability at least $1-\frac{1}{n}$. $B$ guesses it correctly as the point of rewind, with probability at least $\frac{1}{q_h}$. Thus, the overall successful probability is $(1-\frac{q_h q_c}{n})(1-\frac{1}{n})(\frac{1}{q_h})\varepsilon$.

The time complexity of $\mathcal{F}$ is dominated by the exponentiations performed in the $Create$ and $Sign$ queries, which is equal to $t + O(q_c + q_h)S$.

# 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 $2$, where $q$ 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.