Utilization of ECDLP for Constructing a New Certificate Based Digital Signature

Leili Abedi-Ostad¹ and Morteza Nikooghadam²

¹, ² Department of Computer Engineering, University of Imam Reza, Mashhad 91735-553, Iran
E-mail: ¹leilaabediostad@imamreza.ac.ir, ²m.nikooghadam@imamreza.ac.ir

ABSTRACT

Digital signatures that are used to achieve the integrity along with the authentication could be classified into various types. PKI based, ID based, certificate based and certificateless digital signatures are the most important types. Regarding advantages of certificate-based signatures (CBS), we want to propose a CBS scheme by means of employing elliptic curve discrete logarithm problem (ECDLP). The proposed scheme’s security is proven under the elliptic curve discrete logarithm assumption in the random oracle model. Results of comparing our scheme with existing pairing-free certificate-based signature schemes, shows that ours has much lower computational cost.

Keywords: Elliptic Curve Discrete Logarithm Problem, Certificate-Based Digital Signature, Random Oracle Model, Pairing-Free, Elliptic Curve Cryptography.

1 INTRODUCTION

In traditional public key cryptography (PKC), the public key of user should be certified by certification authority (CA). This approach has difficulties for managing certificates. For solving this problem, Shamir [1] presented identity based cryptography (IBC), which means user’s public key is made of his/her unique identity. In this scheme, private key of user is created by a private key generator (PKG). Since PKG has private key of all users, he/she can impersonate them. This problem is called key escrow [2].

There are two schemes for solving this problem. One of them is certificateless public key cryptography (CL-PKC). This scheme was presented by Al-Riyami and Paterson [3]. In this scheme, a key generating center (KGC) has to create user’s partial private key. The private key is made of the partial private key and a random secret value that is selected by the user. Since users select their own public key, there is no way for authentication of declared public key. This problem leads to key replacement attack [4]. The other scheme is certificate-based cryptography (CBE) which was presented by Gentry [2]. In this scheme all users make their private and public key. Afterwards CA produces a certificate for each user by using his/her identity and public key.

Certificate-based signature (CBS) was proposed by Kang et al. [5]. In this scheme, similar to CBE, private and public keys are created by the user; then CA creates a certificate based on user’s public key and his/her identity. A signer by knowing his/her certificate and private key can produce a valid signature. In [4-10] many certificate-based signatures based on pairing operation were proposed. In 2000, Koblitz et al. [11] found out that the computational cost of exponentiation operation is much more than the cost of scalar multiplication on the elliptic curve group. In 2007, Chen et al. [12] realized that the computational cost of pairing is twenty times more than scalar multiplication over the elliptic curve group. Since cryptography protocols without pairing have much lower cost than pairing-based protocols, Liu et al. [13] suggested one pairing-free CBS schemes. Zhang [14] demonstrated that the proposed pairing-free CBS scheme in [13] was insecure. In 2009, Ming and Wang [15] and Zhang et al. [16] suggested schemes without pairing. Li et al. [10] suggested two secure CBS schemes against key replacement attack. In 2012, Li et al. [17] suggested a short CBS scheme which had one pairing operation. Li et al. [18] in 2013 proposed a new CBS scheme under the
discrete logarithm assumption and secure in random oracle model.

We want to propose a CBS scheme by employing ECDLP. We will show that our scheme is secure under the elliptic curve discrete logarithm assumption in the random oracle model. Compared to existing CBS schemes, ours has much lower computational cost.

At first, we give some definitions. Then you can see our suggested CBS scheme and its security analysis. Efficiency comparison of our scheme and conclusion are at the end of this paper.

2 STRUCTURE OF CBS

Setup: It gets a security parameter, and gives the system public parameters and the certifier’s master secret key.

UserKeyGen: It gets the system public parameters, and gives a secret key and a public key.

Certify: It gets system public parameters, master secret key, the identity of a user and its public key. Then its output is the user certificate.

Sign: It gets system public parameters, a message, the user’s identity and his/her certificate, public key and secret key. Its output is a signature.

Verify: It gets a message/signature pair, system parameters, user’s public key and his/her identity. Its output is 0 or 1. Value 1 indicates a valid signature, and 0 is for an invalid signature.

3 SECURITY MODEL

According to [4, 5 and 18], we should consider adversary and adversary. Adversary is a malicious user who can be anyone except the CA. He can’t gain the certificate of the other users but he can change their public keys. He can’t gain the CA’s master secret key, either. Adversary is a malicious CA who has a master secret key but is not able to change the user’s public key. We use the same security model in [18] for analyzing security of proposed scheme.

4 SUGGESTED CBS SCHEME

Setup: This algorithm gets security parameter k and outputs system parameters and master key. CA proceeds as follows:

1. Selects a k-bit prime p and determines the tuple 
   \( \{F_p, \frac{E}{F_p}, G, P, H_0, H_1, H_2\} \).
2. Selects the master private key \( x \in Z_n^* \) and calculates the master public key \( y = xP \).
3. Selects three cryptographic secure hash functions 
   \( H_0 : \{0,1\}^* \times G \times G \rightarrow Z_n^* \), \( H_1 : \{0,1\}^* \times G \times G \rightarrow Z_n^* \) and \( H_2 : \{0,1\}^* \times \{0,1\}^* \times G \times G \rightarrow Z_n^* \).

Publishes \( \{F_p, \frac{E}{F_p}, G, P, Q, H_0, H_1, H_2\} \) as system parameters and preserves the master key x.

UserKeyGen: This algorithm gets parameters, chooses \( x_{ID} \in Z_n^* \) randomly as the user private key and then calculates \( PK_{ID} = x_{ID}P \) as the user public key.

Certify: This algorithm gets parameters, master secret key x, user public key \( PK_{ID} \) and user identity \( ID \in \{0,1\}^* \). Randomly picks \( s \in Z_n^* \) and computes \( W = sP, \ h_0 = H_0(ID, PK_{ID}, W) \) and \( R = s + x_{ID} \mod n \). The output is the user’s certificate \( Cert_{ID} = \{R, W\} \).

User will validate his/her certificate by checking the equation \( R.P = W + y.H_0(ID, PK_{ID}, W) \).

Sign: It gets parameters, user identity ID, user private key \( x_{ID} \), user certificate \( Cert_{ID} \) and message \( m \in \{0,1\}^* \). The algorithm works as follows:

1. Chooses \( r \in Z_n^* \) randomly and computes \( U = r.P \).
2. Calculates \( h_1 = H_1(m, PK_{ID}, U, W) \) and \( h_2 = H_2(m, ID, PK_{ID}, U, W) \).
3. Calculates \( z = (R + x_{ID}h_1 + r.h_2) \mod n \).
4. The signature is \( \sigma = \{U, W, z\} \).

Verify: Takes parameters, user public key \( PK_{ID} \) and message/signature pair \( (m, \sigma) \) and computes 

\( h_0 = H_0(ID, PK_{ID}, W), \ h_1 = H_1(m, PK_{ID}, U, W), \ h_2 = H_2(m, ID, PK_{ID}, U, W) \).

This algorithm examines the equation:

\( z.P = W + y.h_0 + PK_{ID}h_1 + U.h_2 \) \hspace{1cm} (1)

If the equality holds, the output is 1; if not, the output is 0.

The reason that the verification equation holds for valid signatures is:

\( W + y.h_0 + PK_{ID}h_1 + U.h_2 = (s + x_{ID}h_0).P + x_{ID}h_1.P + r.h_2.P \)

\( = (s + x_{ID}h_0).P + x_{ID}h_1.P + r.h_2.P \)

\( = R.P + x_{ID}h_1.P + r.h_2.P \)

\( = (R + x_{ID}h_1 + r.h_2).P \)

\( = z.P \) \hspace{1cm} (2)
In Figure 1, Setup, UserKeyGen and Certify steps and in Figure 2, Sign and Verify steps are shown.

**Fig. 1. Interactions between the signer and CA**

**Fig. 2. Interactions between the signer and the verifier**

### Signer

1. Chooses $r \in Z_n^*$ randomly
2. Computes $U = r \cdot P$
3. Computes $h_1 = H_1(m, PK_{ID}, U, W)$ and $h_2 = H_2(m, ID, PK_{ID}, U, W)$
4. The signature is $\sigma = (U, W, z)$

### CA

1. Chooses $x \in Z_n^*$ randomly
2. Computes $y = x \cdot P$ and $H_0, H_1$ and $H_2$
3. Publishes $\{F_p, E, G, P, y, H_0, H_1, H_2\}$
4. Randomly picks $s \in Z_n^*$
5. Computes $W = s \cdot P$
6. Computs $h_0 = H_0(ID, PK_{ID}, W)$
7. Computes $R = s + x \cdot h_0 \mod n$

### Verifier

1. Chooses $h_0 = H_0(ID, PK_{ID}, W)$
2. Computes $h_1 = H_1(m, PK_{ID}, U, W)$
3. Computes $h_2 = H_2(m, ID, PK_{ID}, U, W)$
4. Computes $z, P = W + y \cdot h_0 + PK_{ID} \cdot h_1 + U \cdot h_2$

### SECURITY ANALYSIS

**Theorem 1:** (Game I). Let $A_i$ be a Type I adversary against proposed CBS scheme in random oracle model and runs at most $t$ in polynomial time, makes at most $q_{H_0}$ for $i = 0, 1, 2$ $H_i$ queries, $q_r$ PKReplace queries, $q_e$ certification queries, $q_c$ corruption queries, $q_k$ UserKeyGen queries and $q_s$ sign queries and wins the Game I with a probability $\varepsilon$. An algorithm $B$ can solve the ECDLP with a probability $\varepsilon'$ in polynomial time $t'$, where

\[
\varepsilon' \geq \frac{1}{q_{H_0}} \left(1 - \frac{1}{q_{H_0}}\right)^{q_r + q_k + q_e + q_c + q_s} \varepsilon,
\]

\[
t' \leq 2t + (q_k + 2q_e + 4q_s)H_c + (q_e + q_s)H_m
\]

The multiplication operation in $Z_n^*$ takes time $t_e$ and addition operation in $i = j$ takes time $t_m$ in the random oracle model.

**Proof:** Let $(F_p, \frac{E}{F_p}, G, P, Q = \alpha \cdot P)$ be a random instance of the ECDLP selected by $B$ as input. $B$ wants to output $\alpha$. Hash functions are considered as random oracles. For consistence, $B$ requires keeping five initially empty lists $L_k, L_e, L_0, L_1, L_2$. List $L_k$ keeps the UserKeyGen queries and PKReplace queries; list $L_e$ keeps certification queries and lists $L_0, L_1, L_2$ keep $H_i$ queries. At first $B$ sets the master public key $y = Q = \alpha \cdot P$ and gives system parameters $\frac{E}{F_p}, G, P, y$ to $A_i$.

Then, $B$ randomly selects an index $j$ such that $1 \leq j \leq q_{H_0}$, where $q_{H_0}$ is the number of queries in
the random oracle $H_0$. We note that first $ID_j = ID^*$ where $ID_j$ is the $j$th query to the random oracle $H_0$ and $j$ should be selected. Algorithm B will simulate oracles and interact with the adversary $A_i$ as follows:

**UserKeyGen Query:** This algorithm gets a user’s identity $ID_i$. Then $B$ verifies the list $L_k$ to see whether $ID_i$ has been inserted before or not. If it was not defined, $B$ selects $x_{ID_i} \in \mathbb{Z}_n^*$ randomly and puts $PK_{ID_i} = x_{ID_i}P$. Then $B$ adds $(ID_i, x_{ID_i}, PK_{ID_i})$ to the list $L_k$ and transfers $x_{ID_i}$ to $A_i$. Otherwise, it sends back the defined value.

**Certification Query:** This algorithm gets $ID_i$ and $PK_{ID_i}$, then $B$ responds as follows:

- If $i \neq j$, $B$ verifies the list $L_o$ to see whether $ID_i$ has been inserted before or not. If not, $B$ selects two random numbers $d_i$ and $R_i \in \mathbb{Z}_n^*$ and computes $W_i = R_i - d_i \cdot y$. Then $B$ verifies the list $L_0$ to see whether $(ID_i, PK_{ID_i}, W_i)$ has been inserted before or not. If it was defined before, $B$ must reselect $d_i$ and $R_i \in \mathbb{Z}_n^*$. Otherwise $B$ adds $(ID_i, PK_{ID_i}, W_i, d_i)$ to the list $L_0$, adds $(ID_i, PK_{ID_i}, W_i, R_i)$ to list $L_o$ and sends $Cert_{ID_i} = \{W_i, R_i\}$ to $A_i$. Otherwise, it sends back the defined value. If $i = j$, $B$ aborts.

**Sign Query:** This algorithm gets $ID_i$ and $m_i$, then, $B$ makes UserKeyGen query and Corruption query and gets $PK_{ID_i}$ and $x_{ID_i}$. If $x_{ID_i} = \bot$, $A_i$ should provide the matching secret key $x_{ID_i}$. Otherwise $B$ responds as follows:

- If $i \neq j$, $B$ makes certification query and signs the message $m_i$ by using $(Cert_{ID_j}, x_{ID_i})$.

- If $i = j$, $B$ selects $e_j, c_j, z_j, d_j \in \mathbb{Z}_n^*$ and computes $W_j = -d_j \cdot y$ and $U_j = c_j^{-1} (z_j + P \cdot PK_{ID_j}, e_j)$. Then $B$ signs $H_0(ID_j, PK_{ID_j}, W_j) = d_j$, $H_1(m_j, ID_j, PK_{ID_j}, U_j, W_j) = e_j$ and $H_2(m_j, ID_j, PK_{ID_j}, U_j, W_j) = c_j$.

If hash functions $H_0$, $H_1$, and $H_2$ have been defined before, $B$ reselects the random values. Otherwise, $B$ adds $(ID_j, PK_{ID_j}, W_j, d_j)$ to the list $L_0$, adds $(m_j, ID_j, PK_{ID_j}, U_j, W_j, d_j)$ to the list $L_1$ and adds $(m_j, ID_j, PK_{ID_j}, U_j, W_j, c_j)$ to the list $L_2$. Finally, $(U_j, W_j, z_j)$ is given to $A_i$.

Therefore, $A_i$ gives a forgery signature $\sigma^* = \{U^*, W^*, z^*\}$ on message $m^*$ by considering $(ID^*, PK_{ID^*})$. If $ID^* \neq ID_j$, $B$ aborts. If not, by using the forking lemma [19], $B$ repeats $A_i$ with different oracle $H_0$ but the same random tape.
Then $B$ can get another valid signature $\sigma' = \{U^*, W^*, z'\}$. So,

$$z^*P = W^* + y_h^0 + PK_{ID}^*h_1^* + U^*h_2^*$$  \hspace{1cm} (4)  

$$z^*P = W^* + y_h^0 + PK_{ID}^*h_1^* + U^*h_2^*$$  \hspace{1cm} (5)  

From these two forgeries, $B$ can compute $\alpha = \frac{z^* - z'}{h_0 - h_0}$, so $B$ has solved the ECDLP. $B$ can obtain the value of $\alpha$ if $P_r[E_1 \land E_2 \land E_3]$ where $E_1$: $B$ does not fail while responding oracle queries, $E_2$: $A_1$ wins and $E_3$: If $ID^* = ID$. From the simulation, we have

$$P_r[E_1] \geq 1 - \frac{1}{q_{H_0}}^{q_{H_0}}$$

$$P_r[E_2] = \varepsilon$$

$$P_r[E_3] = \frac{1}{q_{H_0}}$$

Thus the probability of $B$ solving ECDLP is $\varepsilon' \geq \frac{1}{q_{H_0}} \left(1 - \frac{1}{q_{H_0}}\right)^{q_{H_0}} \varepsilon$.

Algorithm $B$'s running time $t'$ is two times of the $A_1$'s running time $t$ and the time required to answer oracle queries and the time to solve the ECDLP. Totally $B$'s running time is $t' \leq 2t + (q_k + 2q_e + 4q_1)n_c + (q_c + q_s)\tau_m$. $\Box$

**Theorem 2:** (Game II). Let $A_{II}$ be a Type II adversary against the proposed CBS scheme in random oracle model and wins the Game II with a probability $\varepsilon$. Then there is an algorithm $B$ which can solve the ECDLP with a probability $\varepsilon'$ in polynomial time $t'$, where

$$\varepsilon' \geq \frac{1}{q_{H_0}} \left(1 - \frac{1}{q_{H_0}}\right)^{q_{H_0}} \varepsilon$$

$$t' \leq 2t + (q_k + 2q_e + 4q_1)n_c + (q_c + q_s)\tau_m$$

**Proof:** Let $(F_p, E_{F_p}, G, P, Q = \alpha.P)$ be a random instance of the ECDLP selected by $B$ as input. $B$ wants to output $\alpha$. At first $B$ selects $s \in Z_n^*$ randomly and sets master public key $y = s.P$ and gives system parameters $F_p, E_{F_p}, G, P, y$ and master secret key $s$ to $A_{II}$. Then, $B$ randomly picks an index $j$ such that $1 \leq j \leq q_{H_0}$, where $q_{H_0}$ is the number of queries to the random oracle $H_0$.

It is noticeable that first $ID_j = ID^*$ where $ID_j$ is the $j$th query to the random oracle $H_0$ and $j$ should be selected. Algorithm $B$ will simulate oracles and interact with the adversary $A_{II}$ as follows:

**UserKeyGen Query:** This algorithm gets a user's identity $ID_i$. Then $B$ verifies the list $L_k$ to see whether $ID_i$ has been inserted before or not. If so, the defined value is sent back. If not, $B$ responds as follows: If $i \neq j$ $B$ chooses $x_{ID_i} \in Z_n^*$ randomly and sets $PK_{ID_i} = x_{ID_i}.P$. Then $B$ adds $(ID_i, x_{ID_i}, PK_{ID_i})$ to the list $L_k$ and transfers $PK_{ID_i}$ to $A_{II}$. If $i = j$ $B$ puts $PK_{ID_i} = Q$, then $B$ adds $(ID_j, \perp, PK_{ID_j})$ to the list $L_k$ and transfers $PK_{ID_j}$ to $A_{II}$.

$H_0, H_1$ and $H_2$ queries are the same as $H_0, H_1$ and $H_2$ queries in theorem 1.

**Corruption Query:** This algorithm gets a user's identity $ID_i$, and then $B$ responds as follows:

If $i \neq j$ $B$ verifies the list $L_k$ to see whether $ID_i$ has been defined before or not. If it was not defined, $B$ selects $x_{ID_i} \in Z_n^*$ randomly and puts $PK_{ID_i} = x_{ID_i}.P$. Then $B$ adds $(ID_i, x_{ID_i}, PK_{ID_i})$ to the list $L_k$ and transfers $x_{ID_i}$ to $A_{II}$. Otherwise, it sends back the defined value. If $i = j$, $B$ aborts.

**Sign Query:** this query is the same as sign query in theorem 1, but interacts with $A_{II}$.

Therefore, $A_{II}$ gives a forgery signature $\sigma^* = \{W^*, U^*, z^*\}$ on message $m^*$ by considering $(ID^*, PK_{ID}^*)$. If $ID^* \neq ID$, $B$ aborts. If not, by using the forking lemma [19], $B$ repeats $A_{II}$ with different oracle $H_1$ but the same random tape. Then $B$ can get another valid signature $\sigma^* = \{W^*, U^*, z^*\}$. So,

$$z^*P = W^* + y\cdot h_0^* + PK_{ID}^*h_1^* + U^*h_2^*$$  \hspace{1cm} (7)  

$$z^*P = W^* + y\cdot h_0^* + PK_{ID}^*h_1^* + U^*h_2^*$$  \hspace{1cm} (8)
Table 1: Time complexity comparison

<table>
<thead>
<tr>
<th>Scheme in [13]</th>
<th>Sign generation phase</th>
<th>Time complexity in $T_{Mul}$</th>
<th>Verification phase</th>
<th>Time complexity in $T_{Mul}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{EXP}+2T_{Mul}+T_{ADD}+T_{HASH}$</td>
<td>242 $T_{Mul}$</td>
<td>$7T_{EXP}+5T_{Mul}+3T_{HASH}$</td>
<td>1685 $T_{Mul}$</td>
<td></td>
</tr>
<tr>
<td>Scheme in [15]</td>
<td>$T_{EXP}+T_{Mul}+2T_{ADD}+T_{HASH}$</td>
<td>241 $T_{Mul}$</td>
<td>$3T_{EXP}+4T_{Mul}+2T_{HASH}$</td>
<td>724 $T_{Mul}$</td>
</tr>
<tr>
<td>Scheme in [16]</td>
<td>$3T_{EXP}+3T_{Mul}+2T_{ADD}+2T_{HASH}$</td>
<td>723 $T_{Mul}$</td>
<td>$7T_{EXP}+5T_{Mul}+4T_{HASH}$</td>
<td>1685 $T_{Mul}$</td>
</tr>
<tr>
<td>Scheme in [18]</td>
<td>$T_{EXP}+2T_{Mul}+2T_{ADD}+2T_{HASH}$</td>
<td>242 $T_{Mul}$</td>
<td>$4T_{EXP}+3T_{Mul}+3T_{HASH}$</td>
<td>963 $T_{Mul}$</td>
</tr>
<tr>
<td>Our scheme</td>
<td>$2T_{Mul}+2T_{ADD}+T_{EC−MUL}+2T_{HASH}$</td>
<td>31 $T_{Mul}$</td>
<td>$3T_{EC−ADD}+4T_{EC−MUL}+3T_{HASH}$</td>
<td>116.36 $T_{Mul}$</td>
</tr>
</tbody>
</table>

From these two forgeries, $B$ can compute $\alpha = z^* - \epsilon'$, so $B$ has solved the ECDLP. $B$ can obtain the value of $\alpha$ if $P_r[E_1 \land E_2 \land E_3]$ where $E_1$: $B$ does not fail while responding oracle queries, $E_2$: $A_H$ wins and $E_3$: If $ID^* = ID_j$. From the simulation, we have $P_r[E_1] \geq \left(1 - \frac{1}{q_{H_0}}\right)^{q_+ q_r}$, $P_r[E_2|E_1] = \epsilon$ thus the success probability of $B$ solving ECDLP is $\epsilon' \geq \frac{1}{q_{H_0}} \left(1 - \frac{1}{q_{H_0}}\right)^{q_+ q_r} \epsilon$.

Algorithm $B$’s running time $t'$ is two times of the $A_H$’s running time $t$ and the time required to respond oracle queries and the time to solve the ECDLP. Totally, $B$ run time is $t' \leq 2t + (q_k + q_c + 4q_s)\epsilon + (q_s)\epsilon_m \Box$

6 EFFICIENCY COMPARISON

You can see the definition of used notations in this paper and their conversions in term of $T_{Mul}$ in the following: [11, 20]

$T_{Mul}$ is time complexity of performing a multiplication operation.

$T_{EXP}$ is time complexity of performing an exponentiation operation. (240$T_{Mul}$)

$T_{ADD}$ is Time complexity of performing an addition operation. (Negligible)

$T_{EC−MUL}$ is time complexity of performing a multiplication of an elliptic curve point. (29$T_{Mul}$)

$T_{EC−ADD}$ is time complexity of performing an addition of two points on elliptic curve. (0.12$T_{Mul}$)

$T_{INV}$ is time complexity of performing an inverse operation. (0.073$T_{Mul}$)

$T_{HASH}$ is time complexity of performing a hash function. (Negligible)

We have compared our scheme’s computational cost with the schemes in [13, 15, 16, and 18]. You can see the results in Table 1. Ming et al. scheme [15] and Liu et al. scheme [13] are not secure [18]. Zhang et al. scheme [16] has no security proof. Li et al. scheme [18] is secure and has less computational cost compared to [13, 16]. Comparing our scheme with mentioned schemes in Table 1 shows that our scheme has much lower computational cost.

7 CONCLUSION

CBS schemes use traditional public key infrastructures and identity-based signatures advantages and have no certificate management problem in PKI and key escrow in IBS. In this paper, a new CBS scheme based on elliptic curve cryptography is proposed. The security of our scheme is proven under the ECDL assumption and in the random oracle model. Comparing our scheme with existing pairing-free CBS schemes shows that ours has less computational cost.
7 REFERENCES