Efficient and Provably-secure Certificateless Strong Designated Verifier Signature Scheme without Pairings

Meijiao DUAN, Jianming ZHU, Yang LI

Abstract: Strong designated verifier signature (generally abbreviated to SDVS) allows signers to obtain absolute control over who can verify the signature, while only the designated verifier other than anyone else can verify the validity of a SDVS without being able to transfer the conviction. Certificateless PKC has unique advantages comparing with certificate-based cryptosystems and identity-based PKC, without suffering from key escrow. Motivated by these attractive features, we propose a novel efficient CL-SDVS scheme without bilinear pairings or map-to-point hash operations. The proposed scheme achieves all the required security properties including EUF-CMA, non-transferability, strongness and non-delegatability. We also estimate the computational and communication efficiency. The comparison shows that our scheme outperforms all the previous CL-(S)DVS schemes. Furthermore, the crucial security properties of the CL-SDVS scheme are formally proved based on the intractability of SCDH and ECDL assumptions in random oracle model.

Keywords: certificateless cryptography; existential unforgeability; non-delegatability; strong designated verifier signature; without pairings

1 INTRODUCTION

Traditional digital signature scheme with the public-verifiable property does not fit the requirement of various scenarios, such as deniable authentication, e-bidding, e-voting. To settle this problem, Jakobsson [1] put forward a new concept called designated verifier signature (generally abbreviated to DVS). The most attractive property of DVS is called non-transferability, which ensures signers to obtain absolute control over who can verify the signature. In addition, the designated verifier cannot convince any third entity of the fact the DVS is indeed signed by the genuine signer. Meanwhile, a variant of DVS called Strong designated verifier signature (SDVS) is also proposed. None other than the designated verifier can verify the SDVS. We call this property the strongness property, which is usually achieved by forcing the verification process using the designated verifier’s secret. The formalized definition was first formed by Saeednia [2] and extended by Laguillaumie [3]. Later on, Lipmaa et al. [4] pointed out that most previous (S)DVS schemes [2, 5-6] suffer from a new-type attack named delegatability, in which the signer could delegate his/her signing ability to any third entity by transferring a common value without revealing the secret key. To capture this issue, the security notion of non-delegatability was proposed and formalized. Due to these unique properties above, the SDVS can help resolve the conflict between authentication and privacy protection in digital signatures, thus fit into various cryptographic applications and some new fields, such as IoT communications [7], biometric authentication and identity-management [8], privacy preserving cloud computing, the next-generation network infrastructure [9], and so on.

Following Saeednia et al.’s work, many identity-based (S)DVS schemes were proposed [10-13]. And recently Shooshatri et al. [14] presented a code-based SDVS scheme against quantum attacks. Unfortunately, all the above identity-based (S)DVS schemes suffer from the inherent problem of key escrow in ID-based cryptosystems. Certificateless public key cryptography, which was originally introduced by Al-Riyami et al. [15], can address this problem. It relies on a semi-trusted KGC who owns the master secret key. The full secret key of arbitrary user was deduced from two parts: partial private key D_i supplied from KGC, and secret value x_i chosen by himself. Thus, it has unique advantages comparing with certificate-based cryptosystems and identity-based PKC, without suffering from the problem of key escrow.

Motivated by this attractive feature, Huang [16] presented the first construction of certificateless DVS scheme. However, their scheme is figured out to be vulnerable to malicious-but-passive KGC attacks. Later on, some efficient short certificateless DVS schemes were independently designed [17-19] and claimed to be secure. However, most of them [17-18] still need several pairing operations and the previous one does not resist the public key replace attack. Recently, several certificateless strong DVS schemes (CL-SDVS for short) have been proposed [20-24]. Unfortunately, some of them cannot achieve all the security properties as they claimed. According to [25], the scheme in Reference [22] is insecure against the malicious PKC attack and public key replacement attack. The CL-SDVS scheme proposed in Reference [24] is pointed vulnerable to concrete attack which shows the signature is forgeable [26]. Recalling the proofs of previous insecure schemes [16-18, 22], we find that there is an inappropriate restriction in their security models of unforgeability. The adversary cannot query target signer or the target verifier to sign arbitrary message, nor to check the validity of any signature. With these restrictions, they actually ignore such a practical attack in which the adversary can query arbitrary signature on arbitrary message with respect to arbitrary signer and verifier, or the adversary can eavesdrop all communications between signer and verifier. Thus, the previous models are not adequately considered and essentially limit the adversary’s abilities. Therefore, we consider a strictly stronger security model in Section 4. Moreover, expensive operations such as bilinear pairing, pairing-based exponentiation, and map-to-hash operations are required in all existing schemes. And the computation cost is still maintained at a high level.

This paper presents a novel efficient certificateless SDVS scheme with shorter signature size, achieving all the required security properties of strongness, non-transferability, non-delegatability and EUF-CMA against
both Type I and Type II forgers. The unforgeability for our scheme is formally proved based on Square Computational Diffie-Hellman (SCDH) assumption and Elliptic Curve Discrete Logarithm (ECDL) assumption in random oracle model, of which the SCDH assumption is equivalent to the CDH assumption [27]. And the proof of non-delegatability is strictly following the formalized definition originally introduced by Lipmaa [4]. Furthermore, our scheme does not need expensive pairing operations, which are making it more efficient than all the existing CL-SDVS schemes proposed so far. Due to its security and efficiency, our CL-SDVS scheme can be deployed in various power-constrained applications, such as sensor networks, IoT and cyber-physical systems.

The rest of this paper is organized as follows. The next section introduces the formal definition of certificateless SDVS scheme. We also describe the formalized model of crucial security properties, including unforgeability and non-delegatability. In Section 3, we propose a novel efficient certificateless SDVS scheme. The performance and security are analyzed in the next two sections. The last section concludes this paper.

2 MODELS OF CL-SDVS SCHEMES

2.1 Definition of CL-SDVS

A certificateless strong designated verifier signature (CL-SDVS) scheme consists of eight algorithms shown as the following:

- **Setup**: This PPT algorithm runs by the KGC. Taking the security parameter \( \kappa \) as input, it outputs the public parameters \( \text{params} \) and the master secret key \( \text{msk} \).

- **Partial-Private-Key-Extract**: This PPT algorithm runs by KGC. Taking \( \text{params} \), \( \text{msk} \) and \( ID \in \{0,1\}^* \) (which is the identity of a user) as input, it returns the corresponding partial private key \( d_{ID} \).

- **Set-Secret-Value**: This PPT algorithm runs by a user with identity \( ID \). Taking \( \text{params} \) and \( ID \) as input, this algorithm randomly selects a secret value \( x_{ID} \) and returns to the user.

- **Set-Public-Key**: This PPT algorithm runs by a user with identity \( ID \). On input \( x_{ID} \), the algorithm calculates the public key \( PK_{ID} \).

- **Set-Private-Key**: This PPT algorithm executes by a user with identity \( ID \). On input \( x_{ID} \), this algorithm produces private key \( SK_{ID} \).

- **Sign**: This PPT algorithm runs by a signer. It takes \( (ID_{S}, SK_{ID_{S}}, ID_{V}, PK_{ID_{V}}) \) and \( m \in \mathcal{M} \) as input, and returns \( \sigma = \text{Sign}(SK_{ID_{S}}, ID_{S}, ID_{V}, PK_{ID_{V}}, m) \).

- **Verify**: This deterministic algorithm runs by the designated verifier. Taking \( ID_{V}, SK_{ID_{V}}, ID_{S}, PK_{ID_{S}}, m \) and the purported signature \( \sigma \) as input, it returns 1 (accept) or 0 (reject).

- **Simulation**: This algorithm runs by the designated verifier. Taking \( ID_{V}, SK_{ID_{V}}, ID_{S}, PK_{ID_{S}} \) and \( m \) as input, it returns a simulated signature \( \sigma' \) on \( m \), i.e. \( \sigma' = \text{Simulation}(SK_{ID_{S}}, ID_{S}, ID_{V}, PK_{ID_{V}}, m) \), which should be indistinguishable from signatures constructed by the real signer.

2.2 Security Model of CL-SDVS Schemes

A secure certificateless SDVS scheme must achieve four crucial security properties: unforgeability, strongness, non-transferability, and non-delegatability.

**Unforgeability.** This property ensures that no one can forge a valid CL-SDVS with non-negligible probability without obtaining the private key of either the signer or the designated verifier. Depending on the different capabilities, we can divide the forger in certificateless SDVS schemes into two types. The description and formalization of these forgers is given below.

- **Type I Forger**: Let \( \mathcal{F}_I \) denote this type of forger. Forger \( \mathcal{F}_I \) can request public keys, and even substitute public keys for some values he selects, but cannot obtain the master key \( \text{msk} \). Forger \( \mathcal{F}_I \) simulates a normal third-party attacker against the CL-SDVS scheme. To formalize the attack model, we demonstrate the following game performed between the challenger \( \mathcal{C} \) and the forger \( \mathcal{F}_I \).

- **Setup**: \( \mathcal{C} \) runs the Setup algorithm to generate \( msk \) and \( \text{params} \). Then the forger \( \mathcal{F}_I \) can obtain \( \text{params} \), the identity and public keys of the signer \( S \) and the specified verifier \( V \).

- **Queries**: The forger \( \mathcal{F}_I \) can get access to the following oracles.
  o **ExtrPartSK**\((ID)\): On input of any user’s identity \( ID \), \( \mathcal{C} \) produces the corresponding partial private key \( d_{ID} \).
  o **ExtrFullSK**\((ID)\): On input \( ID \), \( \mathcal{C} \) returns the corresponding full private key \( SK_{ID} \).
  o **ReqestPK**\((ID)\): On input \( ID \), \( \mathcal{C} \) returns the corresponding public key \( PK_{ID} \).
  o **ReplacePK**\((ID)\): The forger \( \mathcal{F}_I \) can select a substitutional public key \( PK_{ID} \) and replace the original public key \( PK_{ID} \).
  o **SIGN**\((m, ID_{S}, ID_{V})\): On input \( m \) with \( ID_{S} \) and \( ID_{V} \), it responds a valid signature \( \sigma \) on \( m \).
  o **VER**\((\sigma, m, ID_{S}, ID_{V})\): On input of a signature \( \sigma \) on \( m \) with \( ID_{S} \) and \( ID_{V} \), it executes the verify algorithm and returns True (valid) or False (invalid).
  o **SIMUL**\((m, ID_{S}, ID_{V})\): On input of a message \( m \) with designated verifier’s identity \( ID_{V} \) and signer’s identity \( ID_{S} \), it responds a valid transcript simulation \( \sigma’ \) on \( m \).

- **Output**: Eventually, \( \mathcal{F}_I \) produces a forgery \( \left( m^*, \sigma^* \right) \) with the target signer \( ID_{S} \) and designated verifier \( ID_{V} \). The forger \( \mathcal{F}_I \) wins this game if:
  1. Algorithm **Verify** \((\sigma^*, m^*, ID_{S}^*, ID_{V}^*)\) outputs accept;
  2. \( \mathcal{F}_I \) has never queried **ExtrPartSK** \((ID_{V})\) or **ExtrFullSK** \((ID_{S}^*)\);
3. \( \mathcal{F}_I \) has never queried \( \text{ExtrPartSK}(ID^*_S) \) or \( \text{ExtrFullSK}(ID^*_S) \); 
4. \( \mathcal{F}_I \) has never queried \( \text{SIGN}(m, ID^*_S, ID^*_V) \) or \( \text{SIMUL}(m, ID^*_S, ID^*_V) \) to get the signature or a simulated one on \( m^* \) with \( ID^*_S \) and \( ID^*_V \); 
5. \( \mathcal{F}_I \) has never queried \( \text{SIGN}(m, ID^*_V, ID^*_V) \) or \( \text{SIMUL}(m, ID^*_V, ID^*_V) \) to get the signature or a simulated one on \( m^* \) with \( ID^*_V \) and \( ID^*_V \).

**Type II Forger:** Let \( \mathcal{F}_u \) denote this type of forger. Forger \( \mathcal{F}_u \) has access to the master-key msk, but could not substitute public keys. Forger \( \mathcal{F}_u \) simulates a malicious KGC against the CL-SDVS scheme. To define the attack model, we demonstrate this game below performed between a challenger \( \mathcal{C} \) and the forger \( \mathcal{F}_u \): 

- **Setup:** \( \mathcal{C} \) executes the Setup algorithm of the CL-SDVS scheme to generate msk and params, which the forger \( \mathcal{F}_u \) also can both obtain.
- **Queries:** The forger \( \mathcal{F}_u \) can get access to the following oracles, which are the same as in Game I: ExtrFullSK(ID), ReqestPK(ID), SIGN(m, ID_s, ID_v), VER(\( \sigma \), m, ID_s, ID_v), and SIMUL(m, ID_s, ID_v).
- **Output:** Eventually, \( \mathcal{F}_u \) outputs his forgery \( (m^*, \sigma^*) \) with the target \( ID^*_S \) and \( ID^*_V \). The forger \( \mathcal{F}_u \) wins this game if: 
  1. Algorithm Verify \( (\sigma^+, m^*, ID^*_S, ID^*_V) \) outputs accept; 
  2. \( \mathcal{F}_u \) has never queried ExtrFullSK \( (ID^*_S) \) or ExtrFullSK \( (ID^*_V) \); 
  3. \( \mathcal{F}_u \) has never queried SIGN \( (m, ID^*_S, ID^*_V) \) or SIMUL \( (m, ID^*_S, ID^*_V) \) to get the signature or a simulated one on \( m^* \) with \( ID^*_S \) and \( ID^*_V \); 
  4. \( \mathcal{F}_u \) has never queried SIGN \( (m, ID^*_V, ID^*_V) \) or SIMUL \( (m, ID^*_V, ID^*_V) \) to get the signature or a simulated one on \( m^* \) with \( ID^*_V \) and \( ID^*_V \).

**Strongness.** The verification process needs to use the specified verifier’s secret, without which any third entity cannot confirm whether a signature is valid.

**Non-transferability.** Given arbitrary message \( m \) and the corresponding signature \( \sigma \) generated by the signer, it is computationally indistinguishable from the simulation \( \sigma^+ \) produced by the specified verifier.

**Non-delegatability.** The signer cannot delegate his/her signing ability to any third entity by transferring a common value without revealing the secret key.

Assume that \( \mathcal{F} \) denotes the delegated entity by the real signer, who can generate a valid signature in time \( \tau \) with a non-negligible probability \( \varepsilon \). Then we can construct an efficient black-box knowledge extractor \( E \) based on the algorithm \( \mathcal{F} \). Suppose \( \mathcal{F} \) can produce a valid signature on \( m \) for \( (PK_{mv}, SK_{dv}) \) and \( (PK_{mv}, SK_{dv}) \), where \( m \) is an arbitrary input message, \( (PK_{mv}, SK_{dv}) \) is the key pair of an arbitrary signer, \( (PK_{mv}, SK_{dv}) \) is the key pair of an arbitrary designated verifier. And the probability of \( \mathcal{F} \) succeed is \( \varepsilon > r \), which is related to the choice of the random coin toss and the hash function \( H \). Let \( \mathcal{F}_u \) represent \( \mathcal{F} \) with \( m \) as its input. Then by taking \( \mathcal{F}_u \) as a subroutine, the knowledge extractor \( E \) can deduce either \( SK_{mv} \) or \( SK_{dv} \) in expected time \( \tau / (\varepsilon - r) \), where \( r \in [0,1] \) is the knowledge error. The time cost making the oracle queries is negligible here. If such an \( E \) exists, the CL-SDVS scheme is \( (\tau, r) \)-non-delegatable.

3. **PROPOSED CL-SDVS SCHEME**

This section presents an efficient Certificateless SDVS scheme without using any expensive pairing computation or map-to-point hash operation. Our scheme satisfies all the required security properties including unforgeability, non-transferability, strongness and non-delegatability. The signature length is also obviously reduced. The construction of our CL-SDVS scheme is as follows:

- **Setup.** Let \( \mathcal{E}_q \) be a finite field and \( E / F_q \) be an elliptic curve \( y^2 = x^3 + ax + b \) over \( F_q \), where \( p \) and \( q \) are two large primes. Let \( G \) be a \( q \)-order subgroup of the additive group of points over \( E / F_q \), \( P \) be a generator of \( G \), and \( H_1(), H_2(): [0, 1] \rightarrow Z_q^\times \) be hash functions.

The trusted KGC selects a random value \( s \in Z_q^\times \) and calculates \( P_{pub} = sP \). Finally, the public parameters \( params = (p, q, E / F_q, G, P, P_{pub}, H_1, H_2) \) is released, and master key msk = s is kept secret by KGC.

- **Partial-Private-Key-Extract.** Given a user identity \( ID \), KGC computes its partial private key as follows. First it chooses a random value \( y \in Z_q^\times \) and sets \( Y_{ID} = yP \). Then it calculates the partial private key \( d_{ID} = y + H_1(P_{pub}, ID, Y_{ID})s \). KGC sends \( d_{ID} \) and \( Y_{ID} \) to the corresponding user via a secure channel. The correctness of the partial private key can be confirmed by verifying \( Y_{ID} = H_1(P_{pub}, ID, Y_{ID}) \) and \( d_{ID} = H_2((P_{pub}, ID, Y_{ID})) \).

- **Set-Secret-Value.** The user with identity \( ID \) sets its secret value by randomly choosing \( x_{ID} \in Z_q^\times \).

- **Set-Private-Key.** Combining \( d_{ID} \) and \( x_{ID} \), the user with identity \( ID \) sets \( sk_{ID} = d_{ID} + x_{ID} = w_{ID} \) as his full private key.

- **Set-Public-Key.** On input \( params \) and \( x_{ID} \), the user with identity \( ID \) calculates \( X_{ID} = x_{ID}P \), and sets \( PK_{ID} = (X_{ID}, Y_{ID}) \) as his public key.

- **Sign.** For arbitrary message \( m \) and the designated verifier \( V \), the signer \( S \) calculates the signature \( \sigma \) using
its private key $SK_{I_D_s} = w_s$ as follows. $S$ gets $V$’s identity $ID_V$ and public key $PK_{I_D_s} = (X_f, Y_f)$, and computes $H_{f} = H_{1}(PK_{pub}, ID_V, Y_f)P_{pub}$. Then $S$ selects three random values $t, r, k \in \mathbb{Z}_q^*$, calculates $T = tP$, $R = rP$ and 
\[
\begin{align*}
c_1 &= (t + w_s)(X_f + Y_f + H_{f}) \\
c_2 &= R + k(X_f + Y_f + H_{f}) \\
h &= H_{2}(ID_S, ID_V, m, T, c_1, c_2) \\
z &= (t + r + h)w_s \mod q
\end{align*}
\]

The CL-SDVS is then $\sigma = (r, k, h, z)$. 

There is a remain issue that should be noticed. Before computing $z$ in the above process, it is necessary to make sure $r + h \neq 0 \mod q$. If this inequation does not hold, $S$ should re-select an alternative random $r' \in \mathbb{Z}_q^*$ which satisfies $r' + h \neq 0 \mod q$. Then the signature can be produced subsequently.

- **Verify.** Given a CL-SDVS signature $\sigma = (r, k, h, z)$ on message $m$, the designated verifier $V$ needs to check whether $r + h \neq 0 \mod q$. If not, $V$ outputs "reject". Otherwise, it calculates 
\[
\begin{align*}
H_S &= H_{1}(PK_{pub}, ID_V, Y_S)P_{pub} \\
T' &= zP - \beta(X_s + Y_s + H_{S}) \\
c_1' &= w_f(T' + X_s + Y_s + H_{S}) \\
c_2' &= aP \\
h' &= H_{2}(ID_S, ID_V, m, T', c_1', c_2') \\
r' &= (\beta - h') \mod q \\
k' &= w_f(a - r') \mod q \\
\end{align*}
\]

Then the simulated CL-SDVS is $\sigma' = (r', k', h', z')$. The correctness of the CL-SDVS scheme is proved following from the fact that 
\[
\begin{align*}
T &= zP - (r + h)(X_s + Y_s + H_{S}) \\
&= [t + (r + h)w_s]P - (r + h)(X_s + Y_s + H_{S}) \\
&= tP + (r + h)w_sP - (r + h)(X_s + d_sP) \\
&= tP + (r + h)w_sP - (r + h)(X_s + d_sP) \\
&= tP \\
c_1 &= w_f(T + X_s + Y_s + H_{S}) \\
&= w_f(T + X_s + Y_s + H_{S} + H_{1}(PK_{pub}, ID_S, Y_s)P_{pub}) \\
&= w_f(t + x_s + d_s)P = (t + w_s)w_fP \\
&= (t + w_s)(X_f + Y_f + H_{1}(PK_{pub}, ID_V, Y_f)P_{pub}) \\
&= (t + w_s)(X_f + Y_f + H_{f}) \\
\end{align*}
\]

and 
\[
\begin{align*}
c_2 &= rP + kw_fP \\
&= R + k(X_f + Y_f + H_{1}(PK_{pub}, ID_V, Y_f)P_{pub}) \\
&= R + k(X_f + Y_f + H_{f}) \\
\end{align*}
\]

4 SECURITY ANALYSIS

The computational assumptions on which the security proof relies are briefly described below.

**Definition 1** (Square Computational Diffie-Hellman (SCDH) Problem): Given a $q$-order cyclic additive group $\mathbb{G}$ generated by an element $P$. On input $P$ and $aP$, computing $a^2P$.

**Definition 2** (SCDH Assumption): Let $\mathcal{A}$ be a SCDH-adversary who can succeed in solving the SCDH problem within time $t$ with the probability of $Adv_{\mathcal{G}}^{scdh}(\mathcal{A})$. If $Adv_{\mathcal{G}}^{scdh}(t) = \max_{\mathcal{A}} \{Adv_{\mathcal{G}}^{scdh}(\mathcal{A})\}$ is negligible, the SCDH assumption holds in $\mathcal{G}$.

**Definition 3** (Elliptic Curve Discrete Logarithm (ECDL) Problem): Let $\mathbb{F}_p$ be a finite field, $\mathbb{E}$ be an elliptic curve defined over $\mathbb{F}_p$. On input $P$ and $Q = aP$, both of them are elements on $q$-order group $\mathbb{E}/\mathbb{F}_p$, compute $a$.

**Definition 4** (ECDL assumption): For arbitrary polynomial time bounded adversary $\mathcal{A}$, the advantage of $\mathcal{A}$’s success in breaking the ECDLP problem is $Adv_{\mathbb{E}}^{ecdl}(t) = \max_{\mathcal{A}} \{Adv_{\mathbb{E}}^{ecdl}(\mathcal{A})\}$. If $Adv_{\mathbb{E}}^{ecdl}(t)$ is negligible, the ECDLP assumption holds in $\mathbb{E}$.

**Theorem 1.** If the SCDH assumption holds, then our proposed CL-SDVS scheme is existential unforgeable against Type I forger under chosen message attacks in the random oracle model.

**Proof.** Given a SCDH instance $(A, B) = (aP, P)$, in which $a \in \mathbb{Z}_q^*$ is an unknown random value and $P \in \mathbb{G}$. Suppose $\mathcal{F}_i$ is a successful forger against our CL-SDVS scheme. Then an algorithm $\mathcal{C}$ can be constructed by running $\mathcal{F}_i$ as a subroutine to break the SCDH problem. The aim of $\mathcal{C}$ is to compute $a^2P$.

Algorithm $\mathcal{C}$ simulates the attack environment. And system parameters are published to $\mathcal{F}_i$, in which the master public key is $P_{pub} = A = aP$. In this formula, $A$ is the master private key and should be kept secret to $\mathcal{C}$. The oracle queries of forger $\mathcal{F}_i$ are simulated as follows:

- $H_{1}(PK_{pub}, ID_a, Y_a)$ query: $\mathcal{C}$ maintains a list $H_{1}^{list}$ (initially empty) with entries of format <$P_{pub}, ID_a, Y_a, e_i$>. When $\mathcal{F}_i$ queries the $H_1$ Oracle with input $(ID_a, Y_a)$, $\mathcal{C}$ responds as below.
• If \((ID_i, Y)\) already consists in list \(H^1_{\text{list}}\) within a tuple \(<P_{\text{pub}}, ID_i, Y, e>\), then \(C\) responds with \(H_i(T_{\text{pub}}, ID_i, Y) = e\).

• Otherwise, \(C\) selects a random value \(e \in \mathbb{Z}_q^*\), inserts \(<P_{\text{pub}}, ID_i, Y, e>\) into the list \(H^1_{\text{list}}\) and returns \(e\).

- \(H_2(ID_i, ID_j, m, T, c_1, c_2)\) query: \(C\) maintains a list \(H^2_{\text{list}}\) (initially empty) with entries of format \(<ID_i, ID_j, m, T, c_1, c_2, h>\) and proceeds as below.
  - If a tuple indexed by \((ID_i, ID_j, m, T, c_1, c_2, h)\) already appears on the list \(H^2_{\text{list}}\), then \(C\) returns \(h\).
  - Otherwise, \(C\) randomly selects \(h \in \mathbb{Z}_q^*\), inserts \(<ID_i, ID_j, m, T, c_1, c_2, h>\) into the list \(H^2_{\text{list}}\) and responds with \(h\).

- ExtrPartSK\((ID_i)\) query: \(C\) maintains list \(PS^{\text{list}}\) with entries of format \(<ID_i, Y, d_i, e_i>\) and responds to the queries.
  - If \(ID_i = ID_S^x\) or \(ID_i = ID_Y^x\), then abort the game.
  - Otherwise, \(C\) randomly selects \(d_i, e_i \in \mathbb{Z}_q^*\), sets \(Y = dP - eP_{\text{pub}}\) and inserts tuple \(<ID_i, Y, d_i, e_i>\) into the list \(PS^{\text{list}}\), \(<ID_i, Y, d_i, e_i>\) into the list \(PK^{\text{list}}\), and \(<P_{\text{pub}}, ID_i, Y, e_i>\) into the list \(H^1_{\text{list}}\). Then \(C\) returns \(PS_{ID_i} = d_i\).

- ExtrFullSK\((ID_i)\) query: \(C\) maintains list \(SK^{\text{list}}\) with entries of format \(<ID_i, d_i, x_i, w_i>\) and proceeds as follows to respond to the queries.
  - If \(ID_i = ID_S^x\) or \(ID_i = ID_Y^x\), then abort the game.
  - Otherwise, \(C\) chooses a random \(x \in \mathbb{Z}_q^*\), sets \(X_i = xP\), \(w_i = d_i + x\) and inserts tuple \(<ID_i, d_i, X_i, w_i>\) into the list \(SK^{\text{list}}\). Then \(C\) searches list \(PK^{\text{list}}\) for tuple \(<ID_i, X_i, w_i>\). If such tuple exists, it refreshes this record with \(<ID_i, X_i, Y_i>\).
  - Otherwise, it inserts \(<ID_i, X_i, Y_i>\) into the list \(PK^{\text{list}}\). Then \(C\) returns \(SK_{ID_i} = w_i\).

- RequestPK\((ID_i)\) query: \(C\) maintains list \(PK^{\text{list}}\) with entries of format \(<ID_i, X_i, Y_i>\). When \(f_i\) queries with \(ID_i\), \(C\) proceeds as follows.
  - If \(ID_i\) already consists in the list \(PK^{\text{list}}\) inside a tuple \(<ID_i, X_i, Y_i>\), then \(C\) returns \(PK_{ID_i} = (X_i, Y_i)\).
  - Else if \(ID_i = ID_S^x\), \(C\) randomly chooses \(x, y \in \mathbb{Z}_q^*\) and sets \(X = xP, Y = yP\), stores \(<ID_i, X, Y>\) in list \(PK^{\text{list}}\), and returns \(PK_{ID_i} = (X, Y)\).
  - Else if \(ID_i = ID_Y^x\), \(C\) generates \(X = xP\), \(Y = dP - eP_{\text{pub}}\) by randomly picking \(x, d, e \in \mathbb{Z}_q^*\). Then it stores \(<ID_i, X, Y>\) in list \(PK^{\text{list}}\), inserts \(<P_{\text{pub}}, ID_i, Y, e>\) into the list \(H^1_{\text{list}}\) and returns in output \(PK_{ID_i} = (X, Y)\).
  - Otherwise, \(C\) makes an ExtrFullSK\((ID_i)\) query on \(ID_i\) itself to refresh its list \(PK^{\text{list}}\) and returns in output \(PK_{ID_i} = (X, Y)\).

- ReplacePK\((ID_i, X_i, Y_i)\) query: When \(f_i\) inputs \((ID_i, X_i, Y_i)\) to replace user \(ID_i\) ’s public key, \(C\) searches the list \(PK^{\text{list}}\) to check whether an element \(<ID_i, \cdot, \cdot>\) exists.
  - If it does, \(C\) sets \(X_i = X_i', Y_i = Y_i\) and refreshes the record with \(<ID_i, X_i', Y_i'>\).
  - Otherwise, \(C\) inserts tuple \(<ID_i, X_i', Y_i'>\) into the list \(PK^{\text{list}}\).

- SIGN\((ID_i, ID_j, m)\) query: When \(f_i\) queries with signer \(ID_i\), verifier \(ID_j\) and message \(m\), \(C\) first searches list \(PK^{\text{list}}\) for tuples \(<ID_i, X_i, Y_i>\) and \(<ID_j, X_j, Y_j>\). If such tuples do not exist, it runs the above algorithms to generate keys for user \(ID_i\) and/or \(ID_j\). Then it proceeds as follows.
  - If \((ID_1, ID_2) = (ID_Y^x, ID_Y^x)\) or \((ID_Y^x, ID_Y^x)\), and \(m = m^*\), then it returns \(\bot\).
  - Else if \(ID_1 \neq ID_Y^x\) or \(ID_Y^x\), \(C\) first finds \(<ID_i, w_i>\) in list \(SK^{\text{list}}\) and \(<P_{\text{pub}}, ID_i, Y_i, e_i>\) in list \(H^1_{\text{list}}\). Then it selects \(r, k, h \in \mathbb{Z}_q^*\), calculates \(T = tP\), \(c_1 = (t + w_i)(X_j + Y_j + e_jP_{\text{pub}})\), and obtains \(h\) by making a \(H_2(ID_i, ID_j, m, T, c_1, c_2)\) query. Finally \(C\) calculates \(z = t + (h + r)w_i \mod q\), and responds with \(\sigma = (r, k, h, z)\).
  - Else if \(ID_1 = ID_Y^x\) and \(ID_j \neq ID_Y^x\) or \(ID_j = ID_Y^x\), \(C\) first finds \(<ID_i, w_i>\) in list \(SK^{\text{list}}\) and \(<P_{\text{pub}}, ID_i, Y_i, e_i>\) in list \(H^1_{\text{list}}\). Then \(C\) randomly selects \(z, \beta, \alpha \in \mathbb{Z}_q^*\), and computes \(T = zP - \beta(X_i + Y_i + e_jP_{\text{pub}})\) and \(c_2 = \alpha P\). Then it makes a \(H_2(ID_i, ID_j, m, T, c_1, c_2)\) query itself to get \(h\) and calculates \(r = \beta - h \mod q, k = w_j^t(\alpha - r) \mod q\). Finally \(C\) returns \(\sigma = (r, k, h, z)\).
  - Otherwise, \(C\) generates \(X_i = xP\), \(Y_i = dP - eP_{\text{pub}}\) by randomly picking \(x, d, e \in \mathbb{Z}_q^*\). Then it stores \(<ID_i, X_i, Y_i>\) in list \(PK^{\text{list}}\), inserts \(<P_{\text{pub}}, ID_i, Y_i, e>\) into the list \(H^1_{\text{list}}\) and returns whatever it gets.
• \( \text{VER}(ID_{o}, ID_{s}, m, \sigma) \): When \( \mathcal{F}_i \) queries with input a signature \( \sigma = (r, k, h, z) \), \( \mathcal{C} \) first confirms whether \((ID_{i}, ID_{j}) = (ID_{o}, ID_{s}) \) or \((ID_{i}, ID_{j}) = (ID_{s}, ID_{s}) \) and \( m = m^{*} \) holds.
  
  o If so, it returns “\( \bot \)”.  
  
  o Otherwise, \( \mathcal{C} \) searches list \( \mathcal{PK}^{\text{list}} \) to obtain tuple \(<ID_{o}, X_{o}, Y_{o}>\) and list \( \mathcal{SK}^{\text{list}} \) for tuple \(<ID_{o}, w_{o}>\).  

  Here notice, if \( ID_{j} = ID_{o}^{\prime} \), it knows the corresponding \( w_{j} \). If such tuples do not exist, it queries \( \text{ReuestPK}(ID_{i}) \) and/or \( \text{ExtrFullSK}(ID_{i}) \) to get them. Then it verifies whether \( \sigma \) is valid by running the Signature Verification algorithm, and returns the result it gets.

  Eventually, \( \mathcal{F}_i \) forges a valid signature \((ID_{a}, ID_{B}, m^{*}, \sigma = (r, k, h, z)) \). If \((ID_{i}, ID_{j}) \neq (ID_{s}, ID_{s}^{\prime}) \), \( \mathcal{C} \) outputs “\( \bot \)”. Otherwise, \( \mathcal{C} \) finds \(<P_{pub}, ID_{s}^{\prime}, Y_{s}^{\prime}, e_{s}^{*}>\) in list \( H_{1}^{\text{list}} \). Then \( \mathcal{C} \) executes \( \mathcal{F}_i \) again for a second run, while the input \( A = aP \) remain unchanged, and also keep the same coins used for both \( \mathcal{C} \) and \( \mathcal{F}_i \). The difference between the two runnings is that \( \mathcal{C} \) changes the method to answer \( H_{1} \) queries during the second run. Before a particular query of \( H_{1}(P_{pub}, ID_{s}^{\prime}, Y_{s}^{\prime}) \), \( \mathcal{C} \) responds the queries identically as in the first run. To this query \( H_{1}(P_{pub}, ID_{s}^{\prime}, Y_{s}^{\prime}) \), \( \mathcal{C} \) responds with a new independent value \( e_{o} \in \mathbb{Z}_{q}^{*} \). Subsequent queries to \( H_{1} \) are also reponed by randomly choosing from \( \mathbb{Z}_{q}^{*} \).

  Applying the “forking” technique formalized in Reference [28], \( \mathcal{C} \) can get another valid forged signature tuple \((ID_{a}, ID_{B}, m^{*}, \sigma = (r, k, h, z^{\prime})) \), such that \( h \neq h^{\prime} \). If \( \mathcal{F}_i \) can forge a correct valid signature successfully, then it must request the random oracle \( H_{2} \) for the hash value \( h \), and the input \((T, c_{1}, c_{2}) \) must be correct.

  Thus \( \mathcal{C} \) can find the pair \((T, c_{1}, c_{2}) \) and \((T, c_{1}^{\prime}, c_{2}^{\prime}) \) in list \( H_{2}^{\text{list}} \). Then \( \mathcal{C} \) computes the following values:

\[
W_{1} = (e - e_{o})^{c_{1} - c_{1}^{\prime}} (c_{1} - c_{1}^{\prime})
\]

\[
= (e - e_{o})^{-1} (t + w_{s}^{\ast}) [(P_{s}^{\ast} + eP_{pub}) - (P_{s}^{\ast} + e_{o}P_{pub})]
\]

\[
= (e - e_{o})^{c_{1} - c_{1}^{\prime}} (t + w_{s}^{\ast}) (e - e_{o})P_{pub}
\]

\[
= (t + w_{s}^{\ast}) P_{pub}
\]

\[
W_{2} = (h + r - 1)^{-1} (zP_{pub} - W_{1})
\]

\[
= (h + r - 1)^{-1} [(t + (h + r)w_{s}^{\ast})P_{pub} - W_{1}]
\]

\[
= (h + r - 1)^{-1} (t + (h + r)w_{s}^{\ast} - t - w_{s}^{\ast}) P_{pub}
\]

\[
= (h + r - 1)^{-1} (h + r - 1)w_{s}^{\ast} P_{pub}
\]

\[
= w_{s}^{\ast} P_{pub}
\]

\[
W_{3} = e_{s}^{c_{1} - c_{1}^{\prime}} (W_{2} - (x + y)P_{pub})
\]

\[
= e_{s}^{c_{1} - c_{1}^{\prime}} (W_{2}^{\ast} - xP_{pub} - yP_{pub})
\]

\[
= e_{s}^{c_{1} - c_{1}^{\prime}} [(y + e_{o})aP - yP_{pub}]
\]

\[
= e_{s}^{c_{1} - c_{1}^{\prime}} (e_{s}^{-1}(a_{o}^{2}P))
\]

\[
= a_{o}^{2} P
\]

Finally, \( \mathcal{C} \) produces a correct answer \( W_{3} \) to the SCDH challenge. Thus, if there exists a forger \( \mathcal{F}_i \) who can break our CL-SDVS scheme, then there exists an algorithm who can solve the SCDH problem.

**Theorem 2.** If the SCDH assumption holds, then our proposed CL-SDVS scheme is existential unforgeable against Type II forger under chosen message attacks in the random oracle model.

**Proof.** Given an ECDLP instance \((A, B) = (aP, P)\), in which \( a \in \mathbb{Z}_{q}^{*} \) is an unknown random value and \( P \in \mathbb{E} \).

Suppose \( \mathcal{F}_i \) be a successful forger against our CL-SDVS scheme. Then an algorithm \( \mathcal{C} \) can be constructed by running \( \mathcal{F}_i \) as a subroutine to break the ECDLP problem. The aim of \( \mathcal{C} \) is to compute \( a \).

Algorithm \( \mathcal{C} \) simulates the attack environment as follows. It sets \( msk \) by selecting a random value \( s \in \mathbb{Z}_{q}^{*} \), and caculates master public key \( P_{pub} = sP \). Then public parameters are released, and \( msk = s \) is sent to \( \mathcal{F}_i \), simulates the oracle queries of \( \mathcal{F}_i \). We only describe \( \text{ReuestPK}(ID_{i}) \) oracle below without losing generality, since other oracles are essentially the same as in **Theorem 1**.

• \( \text{ReuestPK}(ID_{i}) \) query: \( \mathcal{C} \) maintains list \( \mathcal{PK}^{\text{list}} \) with entries of format \( ID_{o}, X_{o}, Y_{o} \). When \( \mathcal{F}_i \) makes this query on \( ID_{o} \), \( \mathcal{C} \) proceeds as follows.

  o If \( ID_{o} \) already consists in the list \( \mathcal{PK}^{\text{list}} \) in a tuple \(<ID_{o}, X_{o}, Y_{o}>\), then \( \mathcal{C} \) returns \( PK_{ID_{o}} = (X_{o}, Y_{o}) \).

  o Else if \( ID_{o} = ID_{o}^{\prime} \), \( \mathcal{C} \) generates \( Y = dP - eP_{pub} \) by randomly picking \( d, e \in \mathbb{Z}_{q}^{*} \), and computes \( X = dP \). where \((A, B) \) is the ECDLP challenge instance. Then it stores \(<ID_{o}, X, Y> \) in list \( \mathcal{PK}^{\text{list}} \), inserts \(<P_{pub}, ID_{o}, Y, e> \) into the list \( H_{1}^{\text{list}} \), and returns in output \( PK_{ID_{o}} = (X, Y) \).

  o Else if \( ID_{o} = ID_{o}^{\prime} \), \( \mathcal{C} \) chooses \( x, d, e \in \mathbb{Z}_{q}^{*} \) and caculates \( X = xP, Y = dP - eP_{pub} \). Then stores \(<ID_{o}, X, Y> \) in list \( \mathcal{PK}^{\text{list}} \), inserts \(<P_{pub}, ID_{o}, Y, e> \) into the list \( H_{1}^{\text{list}} \), and returns in output \( PK_{ID_{o}} = (X, Y) \).

  o Otherwise, \( \mathcal{C} \) makes an \( \text{ExtrFullSK}(ID_{i}) \) query on \( ID_{i} \) itself to refresh its list \( \mathcal{PK}^{\text{list}} \) and returns in output \( PK_{ID_{i}} = (X_{i}, Y_{i}) \).

Eventually, \( \mathcal{F}_i \) produces its forgery \((ID_{s}^{\prime}, ID_{s}^{\prime}, m^{*}, \sigma = (r, k, h, z)) \). \( \mathcal{C} \) runs \( \mathcal{F}_i \) again for a second time just as described in the proof of **Theorem 1** to get another valid
forgery \((D_S^1, D_v^1, m^*, \sigma') = (r, k, h', z')\) such that \(z \neq z'\).

Then \(C\) calculates the following values:

\[
W = (h - h')^{-1}(z - z') - es \\
= (h - h')^{-1}[t + (r + h)w_5 - t - (r + h)w_5'^*] - es \\
= (h - h')^{-1}(h - h')w_5^* - es \\
= x_5^* + y_5^* + es - es = x_5^* + y_5^* \\
= a
\]

Finally, \(C\) produces a correct answer \(W\) to the ECDLP challenge. Thus, if there exists a forger \(F_m\) who can break our CL-SDVS scheme, the ECDLP problem would be solved.

**Theorem 3.** Suppose \(F\) be a delegated signer who can generate valid signatures on message \(m \in M\) in polynomial time \(r\). The probability \(e\) that \(F\) succeeds is not negligible. Then the CL-SDVS scheme is \((56\varepsilon / \sqrt{e}, 1/q)\)-non-delegatable under the random oracle model.

**Proof.** If there exists a delegated signer \(F\), then we can construct an efficient knowledge extractor \(E\). On input signature \(\sigma\), \(E\) can get access to \(F\) as a black-box oracle, and produce the private key of either \(ID_S\) or \(ID_V\) in expected time \(t \leq 56\varepsilon / \sqrt{e}\). We assume \(e > k\) where \(k = 1/q\). The probability that \(E\) succeeds is 1.

Suppose \(F_m\) be \(F\) with input \(m\). The extractor \(E\) executes \(F_m\) twice with the same random inputs. Except for the responses to the hash query \(H_2(ID_S, ID_V, m, T, c_1, c_2)\), \(E\) executes \(F_m\) step-by-step in both runs. In the second running, \(E\) returns a different random value as the answer. However, the tuple \((T, c_1, c_2)\) must be equal in both runnings, since it is under the hash-function \(H_2\). If both signatures are valid, we should have \(t = t', i.e. z - (r + h)w_5 = z' - (h' + r')w_5\) mod \(q\), and \(c_2 = c_2', i.e. r + k \cdot w_F = r' + k \cdot w_F\) mod \(q\). Here \((r, k, h, z)\) is the signature in the first run, and \((r', k', h', z')\) is the signature in the second run, where \(h \neq h'\).

Now we show how to extract user’s private key. If \(r = r'\) mod \(q\), then \(h + r \neq h' + r'\) mod \(q\) due to the fact that \(h \neq h'\). Thus we can produce \(w_5 = (z - z')/(h - h')\) mod \(q\), derived from the equation \(z - (r + h)w_5 = z' - (h' + r')w_5\) mod \(q\). If \(r \neq r'\) mod \(q\), one can find \(w_F = (r - r')/(k - k)\) mod \(q\) derived from the equation \(r + k \cdot w_F = r' + k \cdot w_F\) mod \(q\). Thus the private key \(w_5\) or \(w_F\) is successfully extracted with probability 1.

**Theorem 4.** Our CL-SDVS scheme satisfies non-transferability.

**Proof.** By executing \(Sign\) algorithm, the signer \(ID_S\) can produce valid signature \(\sigma = (r, k, h, z)\) on arbitrary \(m\), where \(h = H_2(ID_S, ID_V, m, T, c_1, c_2)\). Meanwhile, by executing \(Simulation\) algorithm, the designated verifier \(ID_V\) can also compute valid signatures. For arbitrary probabilistic polynomial-time distinguisher, it is infeasible to discriminate whether the signature is calculated by a real signer, or simulated by the specified verifier.

**Theorem 5.** Our CL-SDVS scheme satisfies strongness.

**Proof.** The \(Verify\) algorithm in our CL-SDVS scheme needs to use the private key \(SK_{ID_S} = w_5\) of the designated verifier. This would guarantee that any third entity cannot verify the validity or invalidity of the signature, without obtaining the corresponding private key. More precisely, any third party cannot distinguish the signature from a random string with identical length and distribution. Thus our proposed scheme inherently satisfies the security property of strongness.

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Strongness</th>
<th>Unforgeable against Type I forger</th>
<th>Unforgeable against Type II forger</th>
<th>Non-transferability</th>
<th>Nondelegatable</th>
<th>Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our scheme</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>SCDH/EC DL</td>
</tr>
<tr>
<td>H-scheme</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>GBDH</td>
</tr>
<tr>
<td>D-scheme</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>GBDH</td>
</tr>
<tr>
<td>C-scheme</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>GBDH</td>
</tr>
<tr>
<td>Y-scheme</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>GBDH</td>
</tr>
<tr>
<td>X-scheme</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>--</td>
<td>CDH&amp;B BDH</td>
</tr>
<tr>
<td>i-scheme</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>--</td>
<td>--</td>
<td>CDH&amp;B BDH</td>
</tr>
</tbody>
</table>

We provide a security comparison of our proposed scheme with known existing CL-DVS schemes (H-scheme [16], D-scheme [18], C-scheme [17], Y-scheme [23], X-scheme [22] and I-scheme [21]) in Tab. 1, which demonstrates that most existing schemes cannot satisfy all the required security properties, while ours can achieve all.

Tehnički vjesnik 25, 6(2018), 1801-1809
We also provide formal security proofs of unforgeability against both Type I and Type II forgers, non-transferability, strongness and non-delegatability above. To the best of our knowledge, it is the first CL-SDVS scheme without pairings that provides formal security proof of non-delegatability.

5 PERFORMANCE EVALUATION

As shown in Tab. 2, we present a performance comparison of the proposed CL-SDVS with H-scheme [16], C-scheme [17], Y-scheme [23], X-scheme [22] and I-scheme [20]. The analysis incudes the required computational cost of signing or verifying a signature, and the communication cost which is measured by the signature length. Here we omit the cost of conventional hash operations and point addition operations for simplicity. Comparing with bilinear pairings and scalar multiplications, these operations are much more efficient and neglected. Denote \( P \) be bilinear pairing operation, \( E \) be pairing-based scalar multiplication, \( E_x \) be ECC-based scalar multiplication, \( E_y \) be exponentiation in group \( G \), \( H \) be map-to-point hash operation, \( f \) be inverse operation, and "pre" be pre-computed operation. Assume the bit-length of group element in \( G \) is 512 bits, and the group order \( q = 160 \) bits.

To evaluate the computational efficiency, we follow the experimental results made in Reference [29], which estimated the running time of representative cryptographic operations. The time for executing the bilinear pairing operation is approximately equal to 20.01 ms, i.e. \( T_P \approx 20.01 \) ms. Similarly, \( T_E \approx 6.38 \) ms, \( T_E_x \approx 0.83 \) ms, \( T_E_y \approx 11.20 \) ms. Fig. 1 shows the computational cost of signature creation and verification for all the compared schemes, while the time needed to execute the pre-computed operations is neglected. Without using any expensive computations like bilinear pairings, our scheme is the most computationally efficient.

6 CONCLUSIONS

This paper presented the first certificateless strong designated verifier signature scheme without using pairing or map-to-point hash operations. Our scheme satisfies all the required security properties of CL-SDVS, that is unforgeability against both Type I and Type II forgers, non-transferability, strongness and the stronger notion non-delegatability. The formal security proofs were based on the intractability of SCDH and ECDL assumptions in the random oracle model. Furthermore, our CL-SDVS scheme achieves higher efficiency and outperforms all the known CL-(S)DVS schemes in this literature. The signature length is also very short.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61702570 and U1509214, the National Key Research and Development Program of China under Grant No. 2017YFB1400701, and the Foundation of Science and Technology on Information Assurance Laboratory under Grant No.KJ-14-008.

7 REFERENCES


Contact information:

Meijiao DUAN, Dr.
School of Information, Central University of Finance and Economics
39 South College Road, Haidian District, Beijing, P. R. China (100081)
E-mail: duanmeijiao@cufe.edu.cn

Jianming ZHU, Prof.
School of Information, Central University of Finance and Economics
39 South College Road, Haidian District, Beijing, P. R. China (100081)
E-mail: tyzjm65@163.com

Yang LI, Assoc. Prof.
School of Information, Central University of Finance and Economics
39 South College Road, Haidian District, Beijing, P. R. China (100081)
E-mail: liyang@cufe.edu.cn