A Practical and Efficient Two-Part Edwards Curve Digital Signature for Mobile Networks

Mingrui Zhang[1,2], Bo Yang[1,2], Hongxia Hou[1,2,3], Meijuan Huang[1,2,4], and Yanwei Zhou[1,2]
(Corresponding author: Bo Yang)

School of Computer Science, Shaanxi Normal University[1]
Xi'an 710119, China
State Key Laboratory of Information Security (Institute of Information Engineering), Chinese Academy of Sciences[2]
Beijing 100093, China
School of Cyberspace Security, Xi'an University of Posts & Telecommunications[3]
Xi'an, Shaanxi 710121, China
School of Mathematics and Information Science, Baoji University of Arts and Sciences[4]
Baoji 721013, China
Email: byang@snnu.edu.cn
(Received Dec. 2, 2019; Revised and Accepted May 15, 2020; First Online May 26, 2021)

Abstract

In the IoT era, people can control their homes with a smartphone easily. To ensure the security of data transmission, digital signature protocols have been applied to the authentication of identity and messages. However, in the traditional method, a user's private key is directly stored on a mobile phone. So that the private key may be disclosed under various malicious attacks. A two-part signature on the Edwards curve is proposed in this paper to improve the security of the private key. A valid signature can be generated without reestablishing the whole private key. The security analysis of the protocol with standard assumptions is also presented. We implement this new protocol on PC and Android smartphones. The evaluation results demonstrate that our protocol runs faster in the key generation phase and has a smaller signature length. Therefore, this scheme can be effectively deployed in practice to protect the private key.

Keywords: EdDSA; Mobile Devices; Mobile Network; Signature; Smart Home

1 Introduction

Due to the development of mobile network and IoT (Internet of Things), living environment of people has been greatly improved. However, owing to constant and fast-paced cyber-attack evolution, The security of mobile network and smart mobile device are raising important. According to [2], in 2020, cyber-attacks will have launched in such a way that: Malicious attacks up to 1001 million, the web attacks 2746 million, and virus and malicious attacks 1585. The business loss caused by network attack will exceed $ 2000 millions. These challenges threaten Iot system security. For example, as a part of Internet of Things, smart home-smartphone system also suffers these security threats. In Smart home-Smartphone system, people can control their smart homes with smartApps on a smartphone through wireless network [4]. All kinds of Smartapps provided by various manufacturers have been able to connect and control smart home conveniently and quickly [10]. As users download and install smart apps from the mobile network, mobile network communication risks may threaten the security of the system.

In this smart home-smartphone system, smart home only executes instructions send by smart phone. In order to prove that instructions really comes from the real user rather than adversary, an effective method is to use authentication technology,such as digital signature. As shown Figure 1. Recently,some new authentication schemes have been proposed [15, 24]. Using smartphones to implement some existing authentication protocols is an effective solution. However, a user's private key always stored in mobile phone directly. There is a potential risk that smartphones could be attacked directly by malicious applications and user's private key may be disclosed. Or, the adversary may obtain the private key by analyzing the memory information of the mobile operating system [1, 17]. If the adversary obtains the key, then the adversary obtains control of the everything of the house.

In order to reduce the risk of the secret key to be stolen, a natural idea is dividing the private key into several parts. A general solution is to use (t, n) threshold secret sharing protocol [21, 26]. In (t, n) threshold secret sharing protocol, the secret is divided into n parts, and no participant less than $t-1$ can recover the complete

secret. Based on the above idea, many threshold based signature algorithms have been proposed [12, 22]. Even though threshold-based signature is a possible solution, there is still a potential risk here. Although the secret key is divided into n parts, the whole private key will still be re-established during the signature process. What's more, if the secret key is divided into too many parts, the signature generation time will be greatly increased.

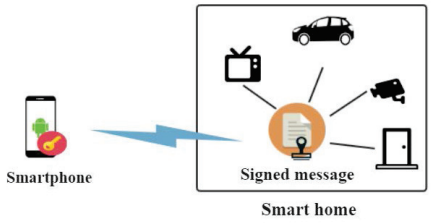

Figure 1: Traditional instructions transmission structure of smart home

To avoid these limitations, one common approach is dividing the private key into two parts, and storing them in different places (*i.e.* smartphone and smart device node). A valid signature generation negotiated by the smartphone and smart server. As shown Figure 2. It requires both parties to participate in the key generation phase and signature phase. Either party cannot recover the private key in the signature process.

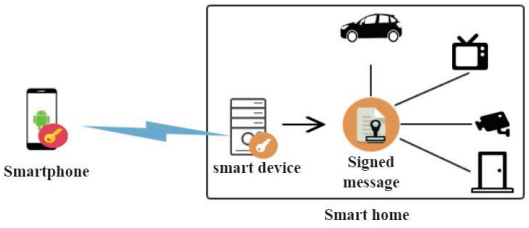

Figure 2: Two-part instructions transmission structure of smart home

To date, many two-party digital signatures have been proposed. A two-part DSA algorithm proposed by MacKenzie and Reiter [11] and the security of the algorithm was proved with random oracle model. Zhang [28] proposed a practical distributed two-party SM2 signature algorithm. SM2 algorithm is issued by the Chinese Government's State Cryptography Administration. A two-part ECDSA protocol proposed by Lindell [18], which is faster than previous protocols. But, many effective side-channel attacks for general NIST elliptic curve on smartphone has been published in [5,23]. He [14] and Zhang [27] proposed two-part identity-based signature protocol in 2018. However, identity based cryptography has a problem of key escrow.

For the sake of improving above deficiency, we present a two-party Edwards-curve digital signature algorithm. Edwards-curve digital signature algorithm developed by Daniel J. Bernstein *et al.*in 2012 [7]. In 2017, this digital signature formally defined in RFC 8032 [16] named EdDSA. EdDSA is more resilient to side-channel attacks, especially the cache-timing attack [3, 6]. The Edward curve [6] and parameter Curve25519 [8] has better efficiency and security than a general NIST elliptic curve [9, 25]. This algorithm is used widely in many softwares such as SSH, TLS, Tor, I2P, *etc.* and blockchains such as Monero, Naivecoin, Siacoin*etc.* [25].

1.1 Our Contribution

In this paper, a two-part siganture on edwards curve is proposed. A valid signature can be generated without reestablishing the whole private key. We implement our protocol on personal computers and mobile device (Android smartphone). Our scheme is more effective, practical and secure in mobile networks. We describe our contribution in detail as follows:

1) A two-part signature on edwards curve is proposed in this paper. According to our knowledge, this is the first two-part cryptography scheme designed for Edwards-curve digital signature algorithm without sacrificing security. This protocol generates a valid signature without reestablishing the whole private key. That is, either party cannot recover the private key in the signature process.

2) The security analysis of the protocol with standard assumptions is also presented. At the same time, in the signature generation phase, our protocol do not use random number generator, which can avoid the additional risks brought by random number generator.

3) This new protocol is implemented by us using Java(11.0.3) on PC and Android smartphone. The evaluation results demonstrate that our protocol has a faster speed in the secret key generation phase and smaller signature length compared with other protocols. Therefore, this scheme can be effectively deployed in practice to protect the private key security.

1.2 Organization of the Paper

This paper is organized as follows: Section 2, we review Edwards-curve digital signature scheme, paillier cryptosystem and zero-knowledge proof. Section 3, we present the protocol for two-part edwards-curve digital signature scheme. Section 4, we describe the security model of two-part Edwards-curve digital signature scheme. Section 5, we present the security analysis. Section 6, we implement our protocol using Java, and compared with other schemes. Section 7 we concludes.

2 Preliminaries

2.1 Edwards-Curve Digital Signature

In this section, we describe the Edwards-curve digital signature. This algorithm has four phases which include set

up, key generation, signature and verify.

2.1.1 EdDSA Setup

1) Construct a twisted Edwards curve over $\mathbb{F}_q$ as follows:

$$ax^2 + y^2 = 1 + dx^2y^2$$

$\mathbb{F}_q$ is a finite field where q is an odd prime. These two numbers a and b are described in [8]. If a point p satisfied this equation, then point $p\,(x_0, y_0)$ on the curve.

2) L is the number of points on the curve.

3) H is a hash function $H : \{0,1\}^* \to \{0,1\}^{512}$.

4) B is a generator of the elliptic curve group.

5) An integer b which is length of EdDSA public key.

6) Two functions $ENC(p)$ and $DEC(i)$. $ENC(p)$ can compress a point p to a integer. $DEC(i)$ can restore a compressed point from integer i.

2.2 EdDSA Key Generation

In this phase, the public key and private key is generated.

1) Choose an EdDSA private key which is a b-bit string k.

2) Calculate the hash $H(k) = (h_0, h_1, ..., h_{2b-1})$.

3) Define a scalar $s = 2^{b-2} + \sum_{3 \leqslant i \leqslant b-3} 2^i h_i$, compute $A = s \cdot B$.

4) Store the compressed value $ENC(A)$ as the public key.

5) The bits $h_b, ..., h_{2b-1}$ are used below during signing.

6) Output the key pair $(s, ENC(A))$.

2.2.1 EdDSA Signature

In this phase, signature (R, S) with message m is generated.

1) Define $r = H(h_b||...||h_{2b-1}||M)$.

2) Compute $R' = r \cdot B$.

3) Calculate $S = (r + H(ENC(R')||ENC(A)||H(M)) \cdot s) \bmod L$.

4) compute $R = ENC(R')$

5) Output EdDSA signature (R, S).

2.2.2 EdDA Verifiy

If following equation is satisfied, it means the signature is valid.

$$8S \cdot B = 8 \cdot R + 8Hash\,(R||ENC(A)||Hash(M)) \cdot A$$

or,

$$S \cdot B = R + Hash(R||ENC(A)||Hash(M)) \cdot A$$

2.3 Zero-Knowledge Proof

In this paper, we mainly use a technology ideal zero-knowledge functionality F_{zk}. The standard ideal zero knowledge functionality is defined by $((x, w), \lambda) \to (\lambda, (x, R(x, w)))$. λ is defined as empty string.

The zero knowledge functionality F_{zk}^R for relation R: Upon receiving $(prove, sid, x, w)$ from party $P_i (i \in 1, 2)$; if $(x, w) \notin R$ or sid has been previously used then ignore the message. Otherwise, send $(proof, sid, x)$to party P_{3-i} [18]. This zero-knowledge proof functionality can be implemented in the random oracle model [13].

The committed non-interactive ideal zero knowledge functionality F_{zk}^{R-com} for relation R. It works as follows: On receiving $(com\text{-}prove, sid, x, w)$ from a party $P_i (i \in 1, 2)$, ignore this message if $(x, w) \notin R$ or sid has been previously used. Store (sid, i, x) and send $(proof\text{-}receipt, sid)$ to P_{3-i}. Upon receiving $(decom\text{-}proof, sid)$ from a party $P_i (i \in 1, 2)$, if (sid, i, x) has been stored then send $(decom\text{-}proof, sid, x)$ to P_{3-i}.

In our two-part EdDSA protocol, we use the following ideal zero-knowledge functionalities: $F_{zk}^{R_p}$, $F_{zk}^{R_{DL}}$, $F_{zk}^{R_{PDL}}$, F_{zk}^{R-com}, which are defined in [18]. There are three relations here:

1) R_p is used to prove that a Paillier homomorphic encryption public key is successfully generated. $R_p = \{(N, (p_1, p_2))|N = p_1 \cdot p_2$ and p_1, p_2 are prime$\}$ It is a vaild Paillier public key, its proof in the [19].

2) Define the relation R_{DL} to prove knowledge of the discrete log of an Elliptic-curve point. $R_{DL} = \{(\mathbb{G}, G, q, P, w)|P = w \cdot G\}$ In our two-part EdDSA protocol, we use the Schnorr proof [20].

3) R_{PDL} is used to prove that a discrete log is encrypted by Paillier algorithm: $R_{PDL} = \{(c_{Paillier}, pk, Q_1, \mathbb{G}, q), (s_1, sk), s_1 = Dec_{sk}(c_{Paillier}), Q_1 = s_1 \cdot G$ and $s_1 \in \mathbb{Z}_q\}$. pk is a Paillier public key and sk is associated private key.

2.4 Paillier Encryption

In our two-part EdDSA protocol, we use Paillier encryption [19]. The Paillier encryption is defined as the following steps:

Key Generation.

1) Choose two random large prime numbers p and q with equal length.

2) Compute $n = pq, k = n + 1, \eta = \phi(n)$ and $\mu = \phi(n)^{-1} \bmod n$,where $\phi(n) = (p-1)(q-1)$.
3) The private key is $sk = (\eta, \mu)$ and the public key is $pk = (n, k)$.

Encryption.

1) Choose a message $m(0 \leqslant m \leqslant n)$ randomly.
2) Select a random v where $\mathbb{Z}^*_{n^2}$.
3) Calculate $c = k^m \cdot v^n \bmod n^2 = Enc_{pk}(m))$.

Decryption.
$m = L(c^\eta \bmod n^2) \cdot \mu \bmod n$, where $L(x) = \frac{x-1}{n}$.

Homomorphic Properties.
Define $c_1 = Enc_{pk}(m_1)$, $c_2 = Enc_{pk}(m_2)$, pk is public key, sk is private key.

1) $Enc_{pk}(m_1) \cdot Enc_{pk}(m_2) \bmod n^2 = Enc_{pk}(m_1 + m_2 \bmod n)$.
2) $Enc_{pk}(m_1)^{m_2} \bmod n^2 = Enc_{pk}(m_1 m_2 \bmod n)$.
3) $Enc_{pk}(m_1)^k \bmod n^2 = Enc_{pk}(km_1 \bmod n)$.

In this paper, we define homomorphic addition symbol is $\oplus$ and define homomorphic multiplication symbol is $\odot$. For example, $c_1 \oplus c_2 = Enc_{pk}(m_1 + m_2)$,$k \odot c_1 = Enc_{pk}(m_1)^k$.

3 Two-Part Edwards Curve Digital Signature

In this part, we show the two-part EdDSA signing protocoal. Two parties P_1, P_2 interact with each other to generate the public key and signature. The protocol has four phases which are setup, two-part key generation and two-part signing. The verify phase is same as EdDSA verify phase.

3.1 Setup

There are parameters:
$params = (a, d, q, L, B, H(x), b, ENC(p), DEC(i))$.
These parameters are the same with those in Section 2.1.

3.2 Two-Part Key Generation

In the two-part key generation phase, P_1 and P_2 interact with each other to generate the two-part EdDSA public key Q.

1) P_1 chooses a random string k_1 which is cryptographic security. Then, P_1 computes $str_1 = Hash(k_1)$. str_1 is the $P_1's$ private key generating elements. Then, P_1 computes itself private key $sl_1 = str_1[len]...str_1\left[\frac{len}{2}+1\right]$, which len is length of str_1. And then, P_1 gets itself signing parameters $perix1 = str_1[1]...str_1\left[\frac{len}{2}\right]$. P_1 computes $Q_1 = sl_1 \cdot B$. Then, P_1 sends $(prove\text{-}com, 1, Q_1, sl_1)$ to $F_{zk}^{R_{DL}-com}$. At last, P_1 generates a Paillier key-pair (pk, sk). P_1 sends $(prove, 1, N(p_1p_2))$ to $F_{zk}^{R_p}$, where $pk = N = p_1 \cdot p_2$.
2) When P_2 receives $(proof, 1, Q_1)$ from $F_{zk}^{R_{DL}}$ and $(proof, 1, N)$ from $F_{zk}^{R_p}$ successfully. P_2 chooses a random string k_2 which is cryptographic security. Then, P_2 computes $str_2 = Hash(k_2)$, str_2 is the private key generating elements of P_2. Then, P_2 computes itself private key $sl_2 = str_2[len]...str_2\left[\frac{len}{2}+1\right]$. The len is length of str_2. And then, P_2 gets its own signing parameters $perix2 = str_2[1]...str_2\left[\frac{len}{2}\right]$. P_2 computes $Q_2 = sl_2 \cdot B$. And then, P_2 sends $(prove, 2, Q_2, sl_2)$ to $F_{zk}^{R_{DL}}$. Finally, P_2 computes $Q = sl_2 \cdot Q_1$ and stores $(sl_2, Q, pk, perix2)$.
3) When P_1 receives $(proof, 2, Q_2)$ form $F_{zk}^{R_{DL}}$, P_1 computes $Q = sl_1 \cdot Q_2$, $Q' = ENC(Q)$. P_1 stores $(sl_1, Q, (sk, pk), perix1, Q')$, and P_1 sends $(decom, 1)$ to $F_{zk}^{R_{DL}}$. If not received , it abort.
4) Finally, if P_2 receives $(decom, 1, Q_1)$ from $F_{zk}^{R_{DL}}$, computes $Q = sl_2 \cdot Q_1$, $Q' = ENC(Q)$. P_2 stores $(sl_2, Q, pk, perix_2, Q')$. Otherwise, it abort. Obviously, $Q = sl_1 \cdot Q_2 = sl_2 \cdot Q_1 = (sl_1 \cdot sl_2)B$. Then P_1 and P_2 interaction processes are described in Figure 3.

3.3 Two-Part Signature Generation

In two-part signature generation phase, P_1 and P_2 interact with each other to generate the Two-part EdDSA Signature (R, S).

1) P_1 computes $r_1 = Hash(perix1||M) \bmod L$. Using Paillier algorithm encrypt r_1 and sl_1, such that $C_1 = Enc_{pk}(r_1)$, $C_2 = Enc_{pk}(sl_1)$. Then P_1 computes $Q_{r1} = r_1 \cdot B$. Finally, P_1 sends $(prove, 1, (c_1, c_2), (r_1, sk))$ and $(commit, 1, Q_{r1})$ to $F_{zk}^{R_{PDL}-com}$.
2) If P_2 receives $(proof, 1, (Q_r, C_1, C_2))$ and $(commit, 1)$ from $F_{zk}^{R_{PDL}-com}$,
then it executes the following steps ; otherwise, it aborts. P_2 computes $r_2 = Hash(perix2||M) \bmod L$. P_2 computes $Q_2' = ENC(Q_2)$; Then, P_2 computes $n_1 = Hash(perix2||Q_2') \bmod L$ and $n_2 = Hash(perix2||Q') \bmod L$. Then P_2 computes $e = Hash(m) \bmod L$ and $Q_{r2} = r_2 \cdot B$. P_2 sends $(prove, 2, R)$ to $F_{zk}^{R_{DL}}$.
3) When P_1 receives $(proof, 2, Q_2)$ from $F_{zk}^{R_{DL}}$, P_1 sends $(decom, 1)$ to $F_{zk}^{R_{DL}-com}$.
4) When P_2 receives Q_{r1}, P_2 computes part of signature $R = e \cdot r_2 \cdot Q_{r1}$ and $r = ENC(R)$. Next, P_2 computes $k = Hash(r||Q'||m) \bmod L$. According to the homomorphic properties of Paillier cryptosystem, P_2 can compute $s_1 = ((r_2 e) \bmod q + n_1 q) \odot C_1$,

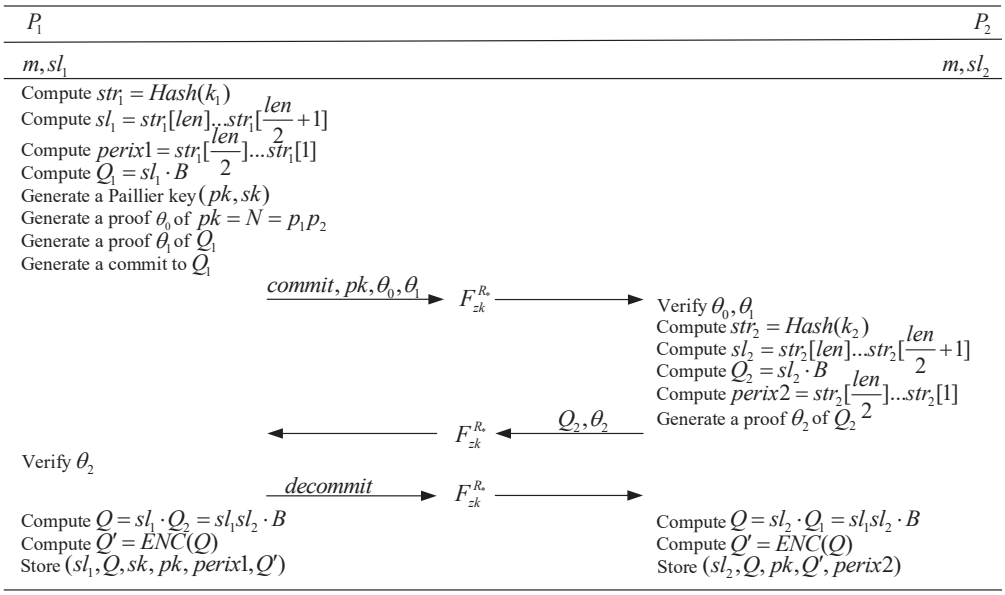

Figure 3: Two-part EdDSA key generation

$s_2 = ((ksl_2) \bmod q + n_2q) \odot C_2$. P_2 computes $s' = s_1 \oplus s_2$. Finally, P_2 sends s' to P_1.

5) If P_1 receives s' from $F_{zk}^{R_{DL}}$, then goes to the following steps; Otherwise, it abort. P_1 can compute the R, P_1 decrypts s' using its private such that $S = Dec_{sk}(s') \bmod L$. Finally, P_1 can verify and output the signature (R, S). Then P_1 and P_2 interaction processes are described in Figure 4.

3.4 Correctness

The correctness of the two-part EdDSA is proven as fellows:

$$\begin{aligned} S &= Dec(s') \bmod L = Dec(s_1 \oplus s_2) \bmod L \\ &= Dec_{sk}((((r_2e) \bmod L + n_1L) \odot c_1) \oplus \\ &\quad (((ksl_2) \bmod L + n_2L) \odot c_2)) \bmod L \\ &= ((r_2e) \cdot r_1) + ((ksl_2) \cdot sl_2) \bmod L \\ &= r_1r_2e + ksl_1sl_2 \bmod L \end{aligned}$$

4 Security Model

Definition 1. *Define a security of the digital signature scheme $\pi = (GenKey, Sign, Verify)$, we define an experiment $Sign\text{-}forge_{\mathcal{A},\pi}(1^h)$ consists of the following steps:*

1) $GenKey(1^h) \Rightarrow (pk, sk)$.

2) $\mathcal{A}^{Sign_{sk}(\cdot)}(1^h, pk) \Rightarrow (m^*, \sigma^*)$.

3) *$\mathcal{M}$ is the set of all message which can be queried. An adversary $\mathcal{A}$ can query the oracle to get message. Then, the experiment outputs 1 if $m^* \notin \mathcal{M}$ and $Verify(m^*, \sigma^*) = 1$.*

Definition 2. *If a signature scheme π is existentially unforgeable under chosen message attack for every probabilistic polynomial-time adversary $\mathcal{A}$, then there exist a negligible function η such that for every h.*

$$\Pr[Sign\text{-}forge_{\mathcal{A},\pi}(1^h) = 1] \leqslant \eta(h).$$

Then, we define an experiment $DistSign\text{-}forge^b_{\mathcal{A},\pi}(1^h)$. We define $\prod$ is our two-part signing protocol. In this experiment, an adversary $\mathcal{A}$ can control one of parties. $\prod_b(\cdot,\cdot)$ is an oracle which executes the instructions from party P_{3-b} honestly. The adversary $\mathcal{A}$ can interact with part P_{3-b} to generate signatures with any message. In this protocol, the key generation process is executed only once, and the signature generation process can be executed many times. The adversary $\mathcal{A}$ can query the oracle inputed two parameters , one is a session identifier (sid) and the other is input message or next incoming message. The oracle works as follows:

1) *Upon receiving a query $(0, 0)$ at the first time, P_{3-b} uses the oracle to initialize a machine M in the two-part key generation part of protocol $\prod$. If P_{3-b} sends the first message in this two-part key generation phase, then this message is the oracle's reply.*

2) *When oracle receives a query $(0, m)$, if the two-part key generation has not been completed, then oracle sends the message to the machine M as the next input message, and return M's reply. (If key generation phase has been finished, then oracle reply $\perp$ and exit.)*

3) *When oracle receives a query (sid, m), where $sid \neq 0$ and m is not the empty string. If two-part key generation phase is not completed, then the oracle returns $\perp$.*

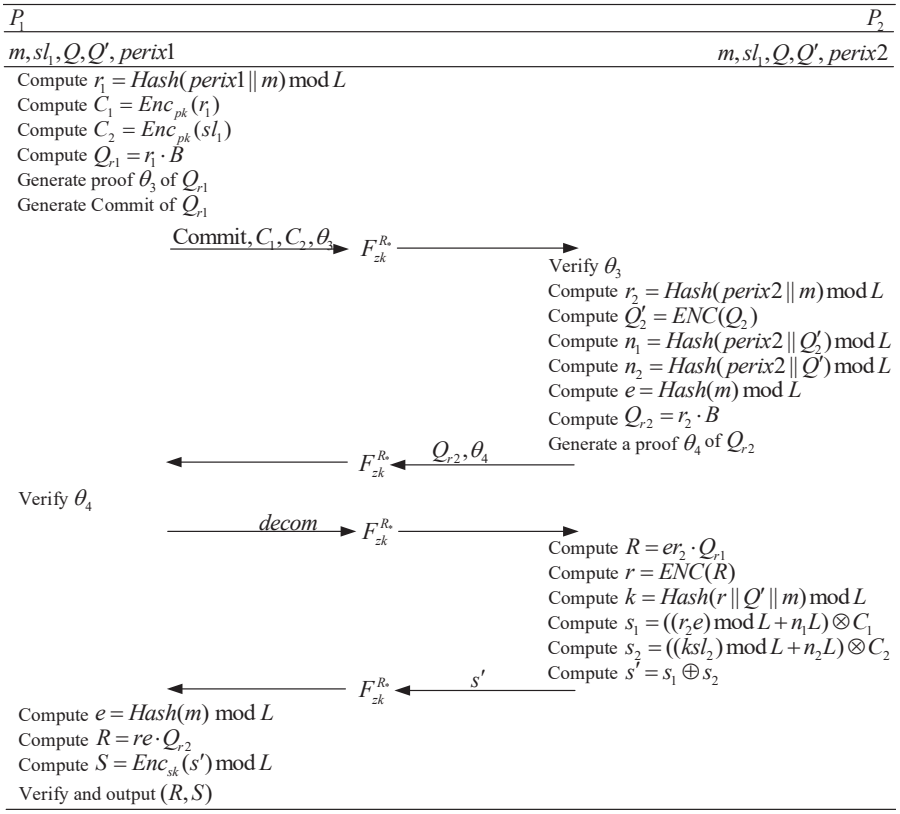

Figure 4: Two-part EdDSA siganture generation

4) *Upon receiving a query (sid, m), the two-part key generation phase has completed and this sid is the first time requested, that means this is a sign require. The oracle call a new machine M_{sid} running the instructions of part P_{3-b} in protocol $\prod$ with (sid, m). M_{sid} is initialized with the key and the parameters stored in M at the end of the two-part key generation phase. If party P_{3-b} sends the first message in the two-part signing phase, then the oracle replies with this message.*

5) *When oracle receives a query (sid, m), the two-part key generation has already finished and sid is not the first time queried, then the oracle sends m to M_{sid}. M_{sid} uses the message m as an input message. The oracle uses M_{sid}'s output as the next message. If M_{sid} completes, then M_{sid}'s output is returned.*

In the above process, an adversary can control part $P_b (b \in 1, 2)$ and can interact with $\prod_b$ in this experiment. If the adversary can forge a signature using a message which has not been queried to the oracle, then the adversary wins. We define the experiment $DistSign\text{-}forge^b_{\mathcal{A},\pi}(1^h)$ and the security definition of protocol $\prod$.

Definition 3. *We define an security experiment $DistSign\text{-}forge^b_{\mathcal{A},\pi}(1^h)$ as follows:*

1) $\mathcal{A}^{\prod_b(\cdot,\cdot)} \Rightarrow (m^*, \sigma^*)$.

2) *Let $\mathcal{M}$ be the set of all messages which adversary $\mathcal{A}$ could query. The adversary can query oracle with form (sid, m). If and only if $m^* \notin \mathcal{M}$ and $Verify_{pk}(m^*, \sigma^*) = 1$, then the experiment outputs 1, where the pk is two-part signing system verification key which output by P_{3-b} in two-part key generation. Verify is an algorithm of $\pi = \{GenKey, Sign, Verify\}$.*

Definition 4. *A protocol $\prod$ is a security two-part protocol for the two-part signature generation for scheme π, if for every probabilistic polynomial-time oracle machine adversary $\mathcal{A}$ and $b \in [1, 2]$, there exist a negligible function η such that for every h:*

$$\Pr[DistSign\text{-}forge^b_{\mathcal{A},\pi}(1^h) = 1] \leqslant \eta(h).$$

Definition 5. *We define a security function $Sign_{EdDSA}$, this function have two subroutines work in the security analysis of protocol $\prod$. These two subroutines as follows:*

1) $(x, Q) \leftarrow Expt\text{-}EdDSA_{keygen}(n)$.

2) $(R, S) \leftarrow Expt\text{-}EdDSA_{signing}(m)$.

Define $Expt\text{-}EdDSA_{keygen}(n)$ is EdDSA key generation function, which generate an EdDSA key pair (x, Q) by invoke the EdDSA key generation algorithm. We define n as a counter. Define $Expt\text{-}EdDSA_{signing}(m)$ is EdDSA singing function, which generate an EdDSA signature (S, R) by invoke the EdDSA signing algorithm. We define m is input message.

5 Security Analysis

In this section, we give the security analysis of our two-part protocol.

Theorem 1. *If the EdDSA signature is existentially-unforgeable under a chosen message attack and Paillier encryption scheme is indistinguishable under chosen-plaintext attack, then our two-part signature algorithm is secure.*

Proof. First of all, in our protocol we use zero-knowledge technology to proof data between P_1 and P_2.All parties can prove the authenticity of the data through zero-knowledge proof-of-knowledge. If adversary $\mathcal{A}$ can break these zero-knowledge model with probability ε, then it can break this two-part protocol with probability $\varepsilon \pm \eta(k)$ where $\eta(k)$ is a negligible function. □

In the process of proof, we assume adversary can corrupt P_1 or P_2 in the experiment. We prove respectively that P_1 corrupted and P_2 corrupted. In addition, we construct an adversary $\mathcal{A}_s$ who can forge a vaild EdDSA signature with probability μ_1 in Definition 1. The probability μ_1 is close to the probability that $\mathcal{A}$ forges a vaild signature in Definition 3 negligibly. Adversary $\mathcal{A}_s$ can invoke EdDSA key generation function and EdDSA signing function which describe in Definition 5 with input (sid, m).

Assume that Paillier has CPA security, then for any probabilistic polynomial time algorithm $\mathcal{A}$ and $b \in [1, 2]$ exists a probabilistic polynomial-time algorithm $\mathcal{A}_s$ and a negligible function μ for every h:

$$|\Pr[Sign\text{-}forge_{\mathcal{A},\pi}(1^h) = 1] - \Pr[DistSign\text{-}forge^b_{\mathcal{A},\prod}(1^h) = 1]| \leq \mu(h). \quad (1)$$

Where, π denotes EdDSA signature scheme and $\prod$ denotes the Two-part EdDSA signature scheme. If EdDSA is security, then according to Definition 2 there exists a negligible function μ' for every h, we can conclude $|\Pr[Sign\text{-}forge_{\mathcal{A},\pi}(1^h) = 1]| \leq \mu'(h)$. On the basis of Equation (1), we conclude that $|\Pr[DistSign\text{-}forge^b_{\mathcal{A},\prod}(1^h) = 1]| \leq \mu(h) + \mu'(h)$, obviously $\mu(h) + \mu'(h)$ is negligible, thus according to Definition 4 we conclude $\prod$ is security. Now, we just need to proof Equation (1) holds in the case of $b = 1$ and $b = 2$, respectively.

If $b = 1$, it means adversary $\mathcal{A}$ corrupts part P_1. Let $\mathcal{A}$ be a probabilistic polynomial-time adversary in $DistSign\text{-}forge^b_{\mathcal{A},\prod}$. We construct a probabilistic polynomial-time adversary $\mathcal{A}_s$ in $Sign\text{-}forge^b_{\mathcal{A},\pi}$. $\mathcal{A}_s$ simulates the execution for $\mathcal{A}$ as follows:

1) First, adversary $\mathcal{A}_s$ can invoke $Expt\text{-}EdDSA_{keygen}(n)$ for Q. Q is the public key of EdDSA.

2) Second, $\mathcal{A}_s$ invokes $\mathcal{A}$ by inputing 1^h to simulate oracle $\prod$ for $\mathcal{A}$ in $DistSign\text{-}forge$ as follows:

a. At the beginning of the experiment, $\mathcal{A}$ always replies $\perp$ in the following two cases. The first case is that the key generation phase has not completed the second case is that the adversary $\mathcal{A}$ has not query $(0, 0)$ to $\prod$. After $\mathcal{A}$ queries $(0, 0)$ to $\prod$ that means this experiment is starting.

b. When $\mathcal{A}_s$ receives the first message $(0, m_1)$ from P_1. It is the first message in the two-part key generation phase. $\mathcal{A}_s$ computes the oracle's reply as follow steps:

   i. Adversary $\mathcal{A}_s$ can analyze message $(prove, 1, Q_1, sl_1)$ sent by P_1 to $F_{zk}^{R_{DL}-com}$.
   ii. Adversary $\mathcal{A}_s$ checks $Q_1 = sl_1 \cdot B$. If this equation holds, $\mathcal{A}_s$ calculates $Q_2 = (sl_1)^{-1} \cdot Q$. Otherwise, $\mathcal{A}_s$ chooses a random point as Q_2.
   iii. Adversary $\mathcal{A}_s$ sets the oracle's replay to be $(proof, 2, Q_2)$ and sends it to $\mathcal{A}$.
   iv. Adversary $\mathcal{A}_s$ can analyze message $(prove, 1, N, (p_1, p_2))$ sent by P_1 to $F_{zk}^{R_p}$.
   v. Adversary $\mathcal{A}_s$ checks the equation $N = p_1 \cdot p_2 = pk$, If this equation is not equal, abort.
   vi. Adversary $\mathcal{A}_s$ stores $perix2 = str_2[1] ... str_2\left[\frac{len}{2}\right]$.

c. Adversary receives the second message $(0, m_2)$ from P_1 and works as follows:

   i. $\mathcal{A}_s$ parses m_2 form of $(decom, 1)$ as $\mathcal{A}$ intends to send to $F_{zk}^{R_{DL}-com}$.
   ii. If $Q_1 \neq sl_1 \cdot B$, $\mathcal{A}_s$ generates the oracle response to P_2 and aborts.
   iii. $\mathcal{A}_s$ stores $(Q, perix2, sl_2, pk)$ and the two-part key generation phase is completed.

d. $\mathcal{A}_s$ receives a query (sid, m) where sid has not been queried yet. Adversary $\mathcal{A}_s$ could invoke function $Expt\text{-}EdDSA_{signing}(m)$ with input m. The function will return a vaild EdDSA signature (R, S). Then, $\mathcal{A}_s$ will reply to $\mathcal{A}$ as follows:

   i. When first message (sid, m) received is in the form of $(prove, commit, (C_1, C_2), (Q_1, sl_1), r_1)$. If $Q_{r1} = r_1 \cdot B$ and $C_1 = Enc_{pk}(r_1)$, $C_2 = Enc_{pk}(sl_1)$, then $\mathcal{A}_s$ sets $Q_{r2} = (r_1)^{-1} \cdot R$. Otherwise, $\mathcal{A}_s$ randomly selects a point as Q_{r2}. $\mathcal{A}_s$ sends the message $(proof, Q_{r2}, 2)$ to $\mathcal{A}$ that $\mathcal{A}$ expects to receive.
   ii. Upon receiving a message of the form $(decommit, 1)$ for the second time from $\mathcal{A}$. If $Q_1 \neq sl_1 \cdot B$, then exit. Otherwise, $\mathcal{A}$ chooses a random number $\rho \in \mathbb{Z}$, compute $S' = Enc_{pk}(S + \rho \cdot L)$, where S is received from function $Expt - EdDSA_{signing}(m)$ and then $\mathcal{A}_s$ sets the oracle reply S' to $\mathcal{A}$.

iii. Once $\mathcal{A}$ halts and outputs a pair (m^*, σ^*) , adversary $\mathcal{A}_s$ outputs (m^*, σ^*). At this point, we could prove that Equation (1) holds. Our ultimate purpose here is to prove that $\mathcal{A}$'s view in the simulation by $\mathcal{A}_s$ is identical to its view in a real execution.

In the two-part key generation phase, there is a difference between the simulation and the real execution. The difference is the generation of Q_{r2}. In the real execution, P_2 computes P_2's private key $sl_2 = str_2[len]...str_2\left[\frac{len}{2}+1\right]$. And then, P_2 computes $Q_2 = sl_2 \cdot B$. However, in the simulation, if $\mathcal{A}_s$ not exit, then $\mathcal{A}_s$ computes $Q_2 = (sl_1)^{-1} \cdot Q$, where Q is the public verification key received by $\mathcal{A}_s$ from key generation function $Expt\text{-}EdDSA_{keygen}(n)$. Thus, the distribution over $sl_2 \cdot B$ and $(sl_1) \cdot Q$ are identical. Obviously, the public key Q obtained by executing the simulation process is the same as that obtained by the function $Expt\text{-}EdDSA_{keygen}(n)$ and we can think of Q as random point.

Therefore, in this phase the $\mathcal{A}$'s view in simulation is identical to its view in a real execution.

In the two-part signing phase, there are two differences. The one difference is the generation of R. In the real execution, P_2 computes $r_2 = Hash(perix2||\ m) \bmod L$ and $Q_{r2} = r_2 \cdot B$. And then, P_1 computes $R = er_1 \cdot Q_{r2}$ or P_2 computes $R = er_2 \cdot Q_{r1}$. But in the simulation, P_2 computes $Q_{r2} = (r_1)^{-1} \cdot R$, where R is generated by signing function $Expt\text{-}EdDSA_{signing}(m)$. Obviously the distribution between $Q_{r2} = r_2 \cdot B$ and $Q_{r2} = (r_1)^{-1} \cdot R$ is identical. The other difference is the generation of S' which is the ciphertext of signature. In the simulation, S is an encryption of $S + \rho \cdot L$. However, in real execution, it is an encryption of $s = r_1 r_2 e + ksl_1 sl_2 + (n_1 r_1 + n_2 r_2)q$ where n_1,n_2 are hash value generated by P_2.

Observe that by the definition of EdDSA signature $s = r + kd = r_1 r_2 e + ksl_1 sl_2 \bmod L$, we can imply that $r_1 r_2 e + ksl_1 sl_2 = s \bmod L$, then $r_1 r_2 e + ksl_1 sl_2 = s + l \cdot L$. Therefore, the difference between the real execution and simulation with S is that:

1) Real execution: $s + l \cdot L + \rho \cdot L$.

2) Simulation: $s + \rho \cdot L$.

Since the distribution of S in the real execution and the simulation is statistically close. We prove that Equation (1) holds for $b = 1$.

If $b = 2$, we use a similar method described above, adversary $\mathcal{A}_s$ corrupts part P_2. Let $\mathcal{A}$ be a probabilistic polynomial-time adversary in $DistSign\text{-}forge^{b}_{\mathcal{A},\prod}$ and $\mathcal{A}_s$ be a probabilistic polynomial-time adversary in $Sign\text{-}forge_{\mathcal{A},\pi}$.

In this phase simulation must be designed to work without knowing the paillier private key. That means $\mathcal{A}_s$ should not know the paillier private key. There is a potential problem here, P_2 may send the wrong value to P_1. In order to solve this problem, we assume that $\mathcal{A}_s$ will abort at some random point. $\mathcal{A}_s$ chooses $i \in [1, ..., p(h)+1]$ randomly. $p(h)$ is the upper bound of the number of query. If $\mathcal{A}_s$ chose correctly with probability $\frac{1}{p(h)+1}$ that means $\mathcal{A}_s$ could simulate $\mathcal{A}$'s view with probability $\frac{1}{p(h)+1}$. In this case, we consider S' is right. Thus, $\mathcal{A}$ can forge a signature in $Sign_{EdDSA}$ with probability at least $\frac{1}{p(h)+1}$. The probability of $\mathcal{A}_s$ can forge a signature in $\prod$ also at least $\frac{1}{p(h)+1}$.
The $\mathcal{A}_s$ work as follows:

1) Adversary $\mathcal{A}_s$ receives $(1^n, Q)$ from key generation function $Expt - EdDSA_{keygen}(n)$, where Q is a public key for EdDSA.

2) Let $p(h)$ be upper bound of the query times that $\mathcal{A}$ queries $\prod$, $\mathcal{A}_s$ chooses $i \in \{1, 2, ..., p(h)+1\}$ randomly.

3) $\mathcal{A}_s$ invokes $\mathcal{A}$ and simulates oracle $\prod$ in $DistSign\text{-}forge^{b=2}_{\mathcal{A},\prod}(1^h)$ as follows: In two-part key generation phase.

   a. Before the two-part key generation phase finished, $\mathcal{A}_s$ always replies $\perp$ to all queries. Whatever the content of the query is. Before $\mathcal{A}$ query $(0,0)$ to $\prod$, $\mathcal{A}_s$ always replies $\perp$ to all queries.

   b. After $\mathcal{A}$ queries $(0,0)$ to $\prod$, $\mathcal{A}_s$ generates a valid paillier encryption key-pair (sk, pk) and sets the oracle reply $(proof, 1, N)$ and $(commit, 1)$.

   c. Adversary $\mathcal{A}_s$ receives the first message $(0, m_1)$ from P_2. It is P_2's first message in the two-part key generation phase.

      i. Adversary $\mathcal{A}_s$ can analyze message $(prove, 2, Q_2, sl_2)$ sent by P_2 to $F_{zk}^{R_{DL}}$.

      ii. $\mathcal{A}_s$ checks $Q_2 = sl_2 \cdot B$. If this equation is not equal, then abort.

      iii. $\mathcal{A}_s$ sets the oracle reply to $(decom, 1, Q_1)$, where $Q_1 = (sl_1)^{-1} \cdot Q$.

      iv. Adversary $\mathcal{A}_s$ stores $perix1 = str_1[1]...str_1\left[\frac{len}{2}\right]$.

      v. $\mathcal{A}_s$ stores $(sl_1, pk, sk, perix1, Q)$ two-part key generation phase finished.

In the two-part signing phase:

4) Upon receiving a query of the form (sid, m) where sid has not been queried yet. $\mathcal{A}_s$ queries the Signing function $Expt - EdDSA_{sigiing}(m^*)$ with m and obtains a valid signature (R, S). Then, adversary $\mathcal{A}_s$ interacts with $\mathcal{A}$ in the following steps:

   a. $\mathcal{A}$ receives the first message $(proof, 2, Q_2)$ that $\mathcal{A}$ sends to $F_{zk}^{R_{DL}}$. $\mathcal{A}_s$ checks $Q_2 = r_2 c \cdot B$ and that Q_2 is a vaild point. $\mathcal{A}_s$ computes $Q_{r1} = (r_2)^{-1} \cdot B$. $\mathcal{A}_s$ sends $(decom, Q_1, 1)$ to P_2 as if coming from $F_{zk}^{R_{DL}}$.

b. When receiving second message (sid, m_2). The message m_2 is encrypted signature s'. If and only if this is the i-th call by $\mathcal{A}$ to the oracle $\prod$ it continues. Otherwise, it aborts.

5) Once $\mathcal{A}$ halts and outputs (m^*, σ), $\mathcal{A}_s$ outputs (m^*, σ) and halts.

When the (sid, m) is first query and it is the $j-th$ query. P_1 does not obtain a valid signature (R, S) with public verification key Q. We consider $i = j$, then the difference between the $\mathcal{A}$'s view in a real execution and the simulated execution by $\mathcal{A}_s$ is C_1 and C_2 . In the real execution, $C_1 = Enc_{pk}(sl_1)$, $C_2 = Enc_{pk}(r_1)$, $Q_1 = sl_1 \cdot B$ and $r_1 = Hash(perix1||M) \bmod L$, however in the simulation sl_1 and r_1 are random value, it is identical. Since $\mathcal{A}_s$ has not hold the pailier private key in the simulation. The indistinguishability of the simulation follows from a reduction of indistinguishability of the encryption scheme under CPA. We can conclude that:

$$|\Pr[Sign\text{-}forge_{\mathrm{A},\pi}(1^h) = 1 | i = j] - \Pr[DistSign\text{-}forge^2_{\mathrm{A},\prod}(1^h) = 1]| \leqslant \eta(h).$$

$$\Pr[DistSign\text{-}forge^2_{\mathrm{A},\prod}(1^h) = 1] \leqslant \frac{\Pr[Sign\text{-}forge_{\mathcal{A}_s,\pi}(1^h) = 1]}{p(h)+1} + \eta(h).$$

$$\Pr[Sign\text{-}forge_{\mathrm{A},\pi}(1^h) = 1] \geq \frac{\Pr[DistSign\text{-}forge^2_{\mathrm{A},\prod}(1^h) = 1]}{p(h)+1} - \eta(h).$$

We can conclude that if adversary $\mathcal{A}$ forges a valid signature in $Sign\text{-}forge_{\mathrm{A},\pi}$ with a non-negligible probability, then $\mathcal{A}_s$ can forge a valid signature in $DistSign\text{-}forge^{b=2}_{\mathrm{A},\prod}$ with a non-negligible probability.

6 Efficiency and Experimental Results

In this section, we will give a comparison of our work with other protocols and show the experiment results.

We implemented our protocol, Zhang's [28] protocol, Lindell's [18] protocol and Zhang's [27] protocol, respectively, using Java(11.0.3) and Jpbc-lib on personal computer with Intel i5-4200H @2.80Ghz processor, 8G bytes memory and Microsoft Windows 10 x64 professional operating system. In [28], [18] and our protocol we use curve25519. We also implemented these four protocols on Android devices(HUAWEI nova3e with Kirin 659 2.36 Ghz processor, 4G bytes memory and Android 9 operating system).

First of all, we compare the running time of the four protocols in different phases on PC and smartphone. The results are illustrated in Figure 5 and Figure 6, respectively, which were obtained in 30 executions. The experimental results show that our protocol runs faster than the other two algorithms [18, 28] in the key generation phase under the same elliptic curve conditions. The two-part identity-based signature scheme [27] runs faster in key generation phase and sign phase. However, it takes more time in the setup phase and have a problem of key escrow.

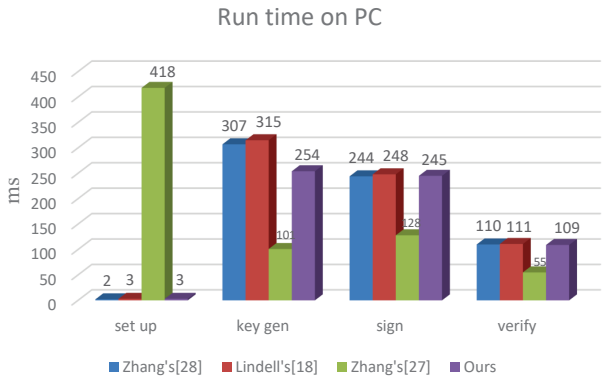

Figure 5: Run time on PC

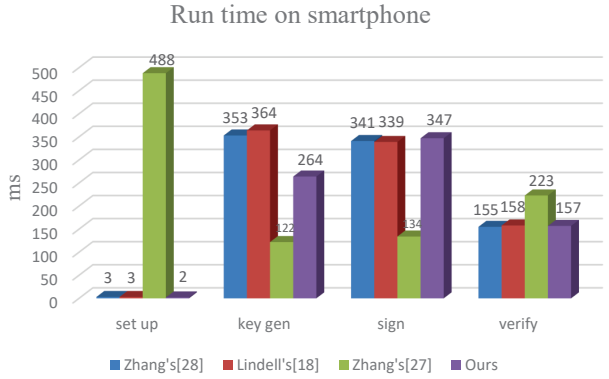

Figure 6: Run time on Smartphone

Secondly, we analyzed the efficiency of each stage of our scheme on PC. As shown in Figure 7. In two-part key generation phase, Keygen_phase1 is defined as the execution process of P_1 before the first message, Keygen_phase2 as the execution process of P_2 between the first message and the second message, and Keygen_phase3 as the execution process after the second message. In two-part signing phase, Sign_phase1 is defined as the execution process of P_1 before the first message, Sign_phase2 as the execution process of P_2 between the first message and the second message, Sign_phase3 as the execution process of P_2 between the second message and the fourth message, and Sign_phase4 as the execution process of P_1 after the fourth message.

Then, we compare the storage space of these three signature protocols in Table 1. Our public key length is one byte shorter than those of other two schemes. Our signature length is 8 bytes shorter than that in Lindell's protocol [18].

Finally, we analyze the size of message as follows. In the two-part key generation phase, the length of all messages of P_1 is $256bits + 1024bits + 512bits = 1792bits$.

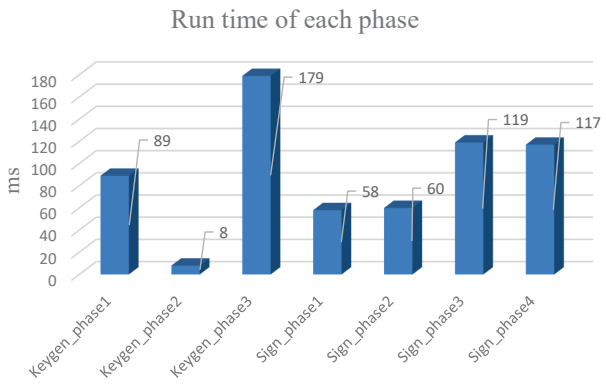

Figure 7: Run time of each phase

Table 1: Length of public key and signature

Protocol	*Public key*	*Signature*
Zhang's [28]	33 bytes	64 bytes
Lindell's [18]	33 bytes	72 bytes
Ours	32 bytes	64 bytes

The length of all messages of P_2 is $256bits$. In the two-part signing phase, the length of all message of P_1 is $256bits + 1024bits + 1024bits = 2304bits$, the length of all messages of P_2 is $2048bits + 256bits = 2304bits$. Therefore, it is efficient for practical application even in restricted environment.

7 Conclusions

- In this paper, we have proposed a two-part signature on edwards curve. This protocol could be effectively applied in blockchain to reduce the risk of key loss. We describe the steps of this protocol and give an analysis of security. Further, we implemented our algorithm and compare with other schemes.
- This protocol can be applied to many real-world scenarios. For example, the signature of blockchain transactions. The security authentication of users in the Internet of Things and Internet of Vehicle*etc.*
- In the future work, we need implement algorithm in real-world applications to further obtain operational data and improve the real execution efficiency of the algorithm.

Acknowledgments

This research was supported by National Key R and D Program of China (No.2017YFB0802000), the National Natural Science Foundation of China (61802241, 61572303, 61772326, 61802242), the National Cryptography Development Fund during the 13th Five-year Plan Period (MMJJ20180217), the Foundation of State Key Laboratory of Information Security (2017-MS-03), the Fundamental Research Funds for the Central Universities (GK201702004).

References

[1] A. A. Al-khatib and W. A. Hammood, "Mobile malware and defending systems: Comparison study," *International Journal of Electronics and Information Engineering*, vol. 6, no. 2, pp. 116–123, 2017.

[2] W. Ali, G. Dustgeer, M. Awais, and M. A. Shah, "Iot based smart home: Security challenges, security requirements and solutions," in *The 23rd International Conference on Automation and Computing (ICAC'17)*, pp. 1–6, 2017.

[3] K. M. Alonso, *Monero-Privacy in the Blockchain*, 2018. (`https://eprint.iacr.org/2018/535.pdf`)

[4] N. Balta-Ozkan, B. Boteler, and O. Amerighi, "European smart home market development: Public views on technical and economic aspects across the united kingdom, germany and italy," *Energy Research & Social Science*, vol. 3, pp. 65–77, 2014.

[5] P. Belgarric, P. A. Fouque, G. Macario-Rat, and M. Tibouchi, "Side-channel analysis of weierstrass and koblitz curve ECDSA on android smartphones," in *Cryptographers' Track at the RSA Conference*, pp. 236–252, 2016.

[6] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted edwards curves," in *International Conference on Cryptology in Africa*, pp. 389–405, 2008.

[7] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[8] D. J. Bernstein, S. Josefsson, T. Lange, P. Schwabe, and B. Y. Yang, "Eddsa for more curves," *Cryptology ePrint Archive*, pp. 5, vol. 2015, 2015. (`https://ed25519.cr.yp.to/eddsa-20150704.pdf`)

[9] D. J. Bernstein and T. Lange, "Failures in Nist's ECC standards," 2016. (`https://cr.yp.to/newelliptic/nistecc-20160106.pdf`)

[10] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE Symposium on Security and Privacy (SP'16)*, pp. 636–654, 2016.

[11] R. Gennaro, S. Goldfeder, and A. Narayanan, "Two-party generation of DSA signatures," in *Annual International Cryptology Conference*, pp. 137–154, 2001.

[12] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security," in *International Conference on Applied Cryptography and Network Security*, pp. 156–174, 2016.

[13] C. Hazay and Y. Lindell, "Efficient secure two-party protocols: Techniques and constructions," *Information Security and Cryptography*, 2010. ISBN: 978-3-642-14303-8.

[14] D. He, Y. Zhang, D. Wang, and K. K. R. Choo, "Secure and efficient two-party signing protocol for the identity-based signature scheme in the IEEE P1363 standard for public key cryptography," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2018. DOI: 10.1109/TDSC.2018.2857775.

[15] M. S. Hwang, E. F. Cahyadi, C. Y. Yang, and S. F. Chiou, "An improvement of the remote authentication scheme for anonymous users using an elliptic curve cryptosystem," in *IEEE 4th International Conference on Computer and Communications (ICCC'18)*, pp. 1872–1877, 2018.

[16] S. Josefsson and I. Liusvaara, *Edwards-Curve Digital Signature Algorithm (EdDSA)*, RFC 8032, 2017.

[17] K. Karimi and S. Krit, "Smart home-smartphone systems: Threats, security requirements and open research challenges," in *International Conference of Computer Science and Renewable Energies (ICCSRE'19)*, pp. 1–5, 2019.

[18] Y. Lindell, "Fast secure two-party ECDSA signing," in *Annual International Cryptology Conference*, pp. 613–644, 2017.

[19] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 223–238, 1999.

[20] C. P. Schnorr, "Efficient signature generation by smart cards," *Journal of cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[21] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[22] H. M. Sun, N. Y. Lee, and T. Hwang, "Threshold proxy signatures," *IEE Proceedings-Computers and Digital Techniques*, vol. 146, no. 5, pp. 259–263, 1999.

[23] S. Takarabt, A. Schaub, A. Facon, S. Guilley, L. Sauvage, Y. Souissi, and Y. Mathieu, "Cache-timing attacks still threaten IoT devices," in *International Conference on Codes, Cryptology, and Information Security*, pp. 13–30, 2019.

[24] C. Y. Tsai, C. S. Pan, and M. S. Hwang, "An improved password authentication scheme for smart card," in *International Conference on Intelligent and Interactive Systems and Applications*, pp. 194–199, 2016.

[25] L. Wang, X. Shen, J. Li, J. Shao, and Y. Yang, "Cryptographic primitives in blockchains," *Journal of Network and Computer Applications*, vol. 127, pp. 43–58, 2019.

[26] C. C. Yang, T. Y. Chang, and M. S. Hwang, "A (t, n) multi-secret sharing scheme," *Applied Mathematics and Computation*, vol. 151, no. 2, pp. 483–490, 2004.

[27] Y. Zhang, D. He, S. Zeadally, D. Wang, and K. K. R. Choo, "Efficient and provably secure distributed signing protocol for mobile devices in wireless networks," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5271–5280, 2018.

[28] M. Zhang, Y. Zhang, D. He, "A provable-secure and practical two-party distributed signing protocol for SM2 signature algorithm," *Frontiers of Computer Science*, vol. 14, no. 3, pp. 1–14, 2020.

Biography

Mingrui Zhang received the B.S degree from Xi'an University of Posts & Telecommunications in 2018, now he is a M.S. degree candidate in School of Computer Science, Shaanxi Normal University, Xi'an, China. His research interests include secure network protocols and its analysis.

Bo Yang received the B.S. degree from Peking University in 1986, and the M.S. and Ph.D. degrees from Xidian University in 1993 and 1999, respectively. From July 1986 to July 2005, he had been at Xidian University. From 2002, he had been a professor of National Key Lab. of ISN in Xidian University. He has served as a program chair for the fourth China Conference on Information and Communications Security in 2005, the vice-chair for ChinaCrypt 2009, and the general co-chair for the Joint Workshop on Information Security since 2010. He is currently a professor and supervisor of Ph.D. candidates at the School of Computer Science, Shaanxi Normal University, a Bai-Ren project special-term professor of Shaanxi Province, and a member of the Council of Chinese Association for Cryptologic Research. His research interests include information theory and cryptography.

Hongxia Hou Ph.D. candidate, now she is an associate professor in Xi'an University of Posts & Telecommunications. Her main research interests include cryptography and information security.

Meijuan Huang Ph.D. candidate, now she is an associate professor in Baoji University of Arts and Sciences. Her main research interests include cryptography.

Yanwei Zhou is an associate professor in Shaanxi Normal University. His main research interests include cryptography and information security.