# Efficient Bitcoin Password-protected Wallet Scheme with Key-dependent Message Security

Liyan Wang[1], Juntao Gao[1], and Xuelian Li[2]
*(Corresponding author: Juntao Gao)*
State Key of Laboratory of Integrated Services Networks, Xidian University[1]
School of Mathematics and Statistics, Xidian University[2]
Xian,Shaanxi, 710126, China
(Email: jtgao@mail.xidian.edu.cn)
*(Received Jan. 17, 2018; Revised and Accepted June 15, 2018; First Online Apr. 4, 2019)*

## Abstract

In Bitcoin financial system, a user is required to generate a fresh key pair to sign a new transaction for protecting privacy. Therefore, a large volume of key pairs need to be stored in Bitcoin wallets. The password-protected wallet, however, is vulnerable to computer crash and ransomware attack. On the other hand, the password itself could be contained in the wallet which would result in the password information leakage by the so-called Key-Dependent Message (KDM).To address these two problems, we propose a new password-protected wallet scheme in this paper. Users can upload the encrypted backup files to the cloud. When the local wallet is lost or damaged, users can recover it via the backup files. To resist against the KDM attack, we use a KDM secure scheme to encrypt wallet files.We prove that our scheme is KDM-CCA secure and the semi-trust cloud server cannot get any information of the backup files. The simulation results show that our scheme is efficient and practical.

*Keywords: Bitcoin; Key-dependent Message (KDM); Password-protected Wallet; Privacy Protection*

## 1 Introduction

Bitcoin has possessed the world's largest trading volume of virtual currencies in recent years. According to economic statistics of the cryptocurrency market (http://coinmarketcap.com/), by January 2018, there are a total of 1381 kinds of cryptocurrencies in the world with a total market capitalization of more than 644 billion dollars. Among them, the market cap of Bitcoin accounts for about 36%. Bitcoin originated in a groundbreaking paper−"Bitcoin: a peer-to-peer electronic cash system" [25]. Instead of depending on a specific central organization, Bitcoin system establishes trust mechanism using a distributed peer-to-peer(P2P) network. All nodes participate in a consensus process called Proof of Work (PoW) to verify and record transactions. The nature of the P2P network is very suitable for establishing an anonymous reputation model [21, 26, 28]. The technological support of Bitcoin is blockchain. Blockchain is a decentralized distributed shared ledger built on P2P networks, which achieves extremely high security in a highly redundant way and creates trust by mutual cooperation [12,22,23]. Trading in Bitcoin, transaction fees are cheaper and cross-border fund transfers become more convenient. Bitcoin transactions can be publicly verified, and the underlying cryptographic mechanisms ensure that records cannot be tampered with. The Bitcoin network is robust enough so that the amount of CPU/GPU needed to control 51% computing power of the network would be astronomical. Due to all these distinguishing features, Bitcoin has recently aroused widespread concerns by academics, financial institutions, government departments, and so on [8,17,20].

Of course, the most appealing feature is that Bitcoin transactions can provide anonymity [27]. In order to ensure the anonymity and increase the transaction security, the current mechanism is that a user has to employ a fresh key pair, *i.e.*, a new pair of public key and private key, to sign each transaction [18,27]. Although time consuming and cumbersome, this is the only available solution currently. As we all know, each transaction requires a valid signature for verification. Only a valid private key can produce a valid digital signature, so one who owns the private key has controlled over the corresponding Bitcoin. If the private key is leaked out, the user's Bitcoin will be lost. All those key pairs are stored in the user's personal wallet. Therefore, how to provide security for personal wallet needs a comprehensive consideration.

Many related wallet schemes have been proposed in recent years. In [5], the authors proposed a scheme called "BlueWallet", using a hardware token to authorize transactions. This scheme is essentially to isolate the trading device from the signature device to prevent malicious attacks. However, usability is also reduced since users have to carry the hardware token with them anytime any-

where. In [24], the authors combined random seeds with a passphrase that easy to remember to generate private keys. Therefore, users only need to store the random seeds in the local files. But once the local files are lost, users cannot recover all the keys. In [15], the first threshold signature scheme compatible with Bitcoin's ECDSA signatures was presented. And this cryptographic primitive could be used to build Bitcoin wallets to enhance the security. In [14], the authors analyzed the setback of [15], then presented a threshold-optimal and efficient signature scheme. Taking into account the priority/weight of participants, Dikshit *et al.* proposed a more practical scheme−a weighted threshold ECDSA scheme−to secure the Bitcoin wallet in [9]. Although this scheme uses joint control to eliminate the risk of internal fraud, once multiple participants with different priority/weight are combined to reconstruct the key, the security of this scheme cannot be guaranteed. In [19], a social-network-based wallet-management scheme was proposed. The authors utilize an identity-based hierarchical key-insulated encryption scheme and a secret sharing scheme to achieve time-sharing authorization. However, due to many bilinear pairing operations involved, the efficiency of the whole scheme is reduced.

In general, existing Bitcoin wallets can be divided into four major categories [11], namely: offline storage wallets, hosted wallets, local wallets and password-protected wallets.

Since the key pairs are stored in the offline device, such as a USB thumbdrive, the offline storage wallet is relatively secure but with low accessibility as the user cannot spend funds unless the offline device is nearby. And the wallet will be easily exposed on an online computational device, such as a computer or mobile connected to the network, possibly to malware. The "BlueWallet" we mentioned above is an offline storage wallet.

Hosted wallets use a third-party web service to host users' accounts. In other words, the service is responsible for maintaining the users' private keys, which will make the third-party service vulnerable to malicious attacks. Once the web service crashes, users need to bear the risk of financial losses.

Local wallet stores the key pairs on the device's local storage, generally in a file or a database in a preset file system path. When launching a new transaction, Bitcoin client can read the key in the local wallet directly. For example, the wallet scheme proposed in [24] is a kind of local wallet. Despite the convenience of this kind of wallet, malware may also read the wallet files, which would lead to disclosure of user's key information. Besides, the device has the risk of computer crash, artificial errors and being stolen. Both of these can result in loss of users' assets.

In order to address the potential physical theft of the local storage devices, password-protected wallets are presented where Bitcoin clients allow a local wallet file to be encrypted with a password. When launching a new transaction, a user needs to unlock the wallet by entering the password before Bitcoin client reads the key in the wallet. In brief, this method reduces the usability of the wallet for the mitigation of the physical theft. However, if the wallet file is only stored locally, though the encrypted file can ensure that a malicious attacker cannot obtain the contents of the file, the user still cannot get back his own transaction keys, which is equivalent to losing money. Besides,ransomware attacks have been frequent in recent years. Once the user's local device encounters this attack, the wallet files cannot be recovered unless the user pays a high amount of ransom. And we have noticed that, when encrypting the wallet, since there are many key pairs involved, the user is likely to put the encryption key into the local wallet file, or directly select a transaction key as the encryption key, which will lead to encrypt the key itself. This is the so-called key-dependent message (KDM) [6], or circular encryption [1].

As early as 1984, when Goldwasser and Micali proposed the definition of semantic security [16], it was pointed out that it would be dangerous to encrypt information that an attacker could not have, such as private keys. If the plaintext is associated with the key, the obtained ciphertext will leak key information. Today, with the demand for encryption gradually diversifies, the probability of such situation is increasing. For example, a data backup system may place the backup encryption key on disk, and then encrypt the contents of the entire disk, including the key [2]. It has been found that in BitLocker disk encryption application of the Windows Vista, the disk encryption key is eventually stored on disk and encrypted with the data on the disk. Once this happens, it can result in an entropy leak of the private keys in the user's wallet file, which can also result in financial losses.

In this paper, we propose a secure and efficient password-protected wallet scheme utilizing backups with key-dependent message security for single user to store and manage personal wallet files. In our scheme, by using a semi-trusted third-party cloud service provider, we store the backups of local encrypted wallet files in the cloud. Once the local device encountered with computer crashes, being stolen or ransomware attacks, the user can recover wallet files from the cloud. Taken the circular encryption§ into account, we use a KDM secure symmetric encryption algorithm [3] to encrypt the wallet files and use HMAC-MD5 algorithm to enhance the security [4] of the scheme. We prove that the proposed scheme can resist active KDM attack and prevent the disclosure of key information. So we will not reveal the private keys of users' to outsiders. In addition, we use a keyword-based searchable encryption algorithm to facilitate the interaction between user and cloud service provider. User can submit the search credential to enable the cloud service provider to return the corresponding backup file instead of downloading all the backup files to local device, making the retrieval time greatly reduced. We utilize one-way trapdoor functions and the learning parity with noise problem which can make sure that the cloud service provider cannot get any detail about the backup files and search

credentials. In our scheme, the adopted algorithms are based on matrix operations or binary bitwise operations and have a low total data volume, which make our scheme more efficient.

The rest of this paper is organized as follows. Section 2 presents some preliminaries used in this paper. Section 3 gives our system model. We give a detailed description of the proposed efficient Bitcoin password-protected wallet scheme in Section 4. The security analysis is presented in Section 5 and the efficiency analysis is presented in Section 6. Finally, in Section 7, we will conclude this paper.

# 2 Preliminaries

We use bold uppercase letters, such as $\boldsymbol{X}$, represent a matrix, and use bold lowercase letters, such as $\boldsymbol{x}$, to represent a vector. If $A$ is a set, then the notation $a \leftarrow A$ denotes randomly and uniformly to choose an element $a$ from $A$. If $A$ is a probabilistic algorithm, then the notation $a \leftarrow A$ denotes that $a$ is computed by $A$. For a positive integer $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, 2, ..., n\}$. For a string $s$, let $s[i]$ denote its $i$-th bit. Let $Ber_{\varepsilon}$ denote the Bernoulli distribution over $\{0, 1\}$ that 1 with probability $\varepsilon$ and 0 with probability $1 - \varepsilon$. And we use $PPT$ to denote probabilistic polynomial time. In addition, for a function $f$ , let $|f|$ denote its output length. We say a function $f$ is negligible in $\eta$ if for any polynomial $p$ there exists a $\eta_0$ such that for all $\eta > \eta_0$, we have $f(\eta) < 1/p(\eta)$.

## 2.1 Key-dependent Message Security

We first review the definition of key-dependent message security (KDM) in the symmetric setting from Black *et al.* [6], and then modify it slightly by restricting the adversary to a special set of functions. We describe the notion of KDM security for a symmetric encryption scheme $\Pi$, which consists of three algorithms $(G, E, D)$ as follows:

- $G(1^{\lambda})$: The key generation algorithm on input of the security parameter $\lambda$ outputs a private key $S$ which we denote by $S \leftarrow G(1^{\lambda})$.
- $E(S, m)$: The encryption algorithm encrypts message $m$ with private key $S$ and outputs the ciphertext $c$. We let $c \leftarrow E(S, m)$ denote this algorithm.
- $D(S, c)$: The decryption algorithm decrypts the ciphertext $c$ with private key $S$ and outputs the message $m$ or an error symbol $\perp$. We let $m \leftarrow D(S, c)$ denote this algorithm.

Correctness: We respectively use $\Omega$ and $M$ to denote the key space and the message space. According to the correctness condition, we require that, for every $S \in \Omega, m \in M, D(S, E(S, m)) = m$.

Now, we define the KDM security with respect to the fixed set of functions $\Gamma = \{f : S^n \rightarrow M\}$ by using the following game that takes place between a challenger and an adversary $\mathcal{A}$, where $n > 0$ is an integer. We require that for all inputs $\alpha \in S^n$, the output length of function $f \in \Gamma$ is fixed, which means that $|f(\alpha)|$ is the same for every input. The game is defined as follows:

- **Initialization:** The challenger chooses a random bit $b \leftarrow \{0, 1\}$. Select a vector of keys $\boldsymbol{S} = \{S_1, S_2, ..., S_n\}$ where each key $S_i (1 \leq i \leq n)$ is determined by running the key generation algorithm $G(1^{\lambda})$.
- **Queries:** The adversary repeatedly issues queries where each query is of the form $(i, f)$ where $1 \leq i \leq n$ and $f \in \Gamma$. If $b = 0$, the challenger returns $c = E(S_i, f(\boldsymbol{S}))$; if $b = 1$, the challenger returns $c = E(S_i, 0^{|f(\boldsymbol{S}))|}$.
- **Final phase:** Finally, the adversary $\mathcal{A}$ outputs a bit $b^{'} \leftarrow \{0, 1\}$.

We say that $\mathcal{A}$ is a $\Gamma$-KDM adversary and that $\mathcal{A}$ wins the game if $b = b^{'}$. The IND-KDM advantage of an adversary $\mathcal{A}$ is defined as:

$$Adv_{\Pi}^{KDM}(\mathcal{A}) = |Pr[b = b^{'}] - 1/2|. \quad (1)$$

**Definition 1.** *We say that an encryption scheme $\Pi$ is KDM secure with respect to $\Gamma$ if for any PPT adversary $\mathcal{A}$, we have $Adv_{\Pi}^{KDM}(\mathcal{A}) = negl(\lambda)$.*

## 2.2 Searchable Encryption

To prevent information disclosure, the data is generally stored in the cloud in encrypted form. When the user needs to find the specific plaintext and does not want to disclose any information, it is difficult for the cloud service provider to search the corresponding ciphertext. Searchable encryption technology (referred to as SE) is a good solution to this problem. It can reduce computational overhead, and make full use of the huge computing resources of cloud service provider. Formally, a basic searchable encryption scheme based on keyword search [7] consists of four algorithms as follows:

- *Setup*: The data owner selects the corresponding set of keywords based on the contents of all files and creates a keyword dictionary.
- *BuildIndex*: The data owner builds a typical index for each file.
- *GenToken*: The algorithm generates a specific search credential based on the keywords that the user needs to search for. The implementation of this algorithm is performed by the data searcher.
- *Query*: This algorithm is carried out by the cloud service provider. After receiving the search credential, the cloud service provider starts the matching calculation and eventually returns the corresponding search results.

### 2.3 HMAC

HMAC is a kind of key-related message authentication code. It makes use of a hash algorithm, with a key and a message as input, and outputs a message digest [13]. In this paper, we adopt HMAC-MD5 algorithm (hereinafter referred to as HMAC). A HMAC scheme consists of three algorithms as follows:

- *MAC-KeyGen*$(1^\lambda)$: The *PPT* key generation algorithm on input of the security parameter $\lambda$ outputs a key $k_{mac}$ which we denote by $k_{mac} \leftarrow$ *MAC-KeyGen*$(1^\lambda)$.
- *Tag*: The user takes the key $k_{mac}$ and message $\psi$ as input, and outputs the message certification tag $T$. We denote by $T \leftarrow Tag(k_{mac}, \psi)$.
- *Verify*: The verifier takes the key $k_{mac}$, the message certification tag $T$ and the corresponding message $\psi$ as input to verify the legitimacy of this message. If the message was modified, $Verify(k_{mac}, T, \psi) = 0$; otherwise, $Verify(k_{mac}, T, \psi) = 1$.

### 2.4 Learning Parity with Noise (LPN)

For positive integers $n$ and $q(q \geq 2)$, a vector $\boldsymbol{s} \in \mathbb{Z}_q^n$, and a probability distribution $\chi$ on $\mathbb{Z}_q$, define $A_{\boldsymbol{s},\chi}$ to be a distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing a vector $\boldsymbol{a} \in \mathbb{Z}_q^n$ uniformly at random, an error term $x \leftarrow \chi$, and outputting $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + x)$.

**Definition 2.** *[3]. For $q = 2$ and an error distribution $\chi = Ber_\varepsilon$, the learning parity with noise problem $LPN_\varepsilon$ is defined as follows: given access to an oracle that produces independent samples from $A_{\boldsymbol{s},\chi}$ for some arbitrary $\boldsymbol{s} \in \mathbb{Z}_2^n$, output $\boldsymbol{s}$ with noticeable probability over all the randomness of the oracle and the algorithm.*

## 3 System Model

As shown in Figure 1, the system model contains three entities: the User, the Cloud service provider (CSP), and the Certificate Authority (CA). The User is an entity who can encrypt his/her own wallet files and upload encrypted backup files to the Cloud service provider. User can also download the specific backup files from the Cloud service provider when recovering the data. The Cloud service provider is an entity that can provide excellent computing service and storage capacity for users. The CA is a credible entity who is responsible for investigating the user's identity and providing key pair generation service.

Because physical theft cannot be blocked, the user encrypts all wallet files using a password in password-protected wallet. In this case, even if the adversary gains physical device, the contents of the wallet files are still not available (due to the lack of an unlocked password). The user who has been stolen also cannot use his/her own Bitcoin, as no one can remember so many key pairs.

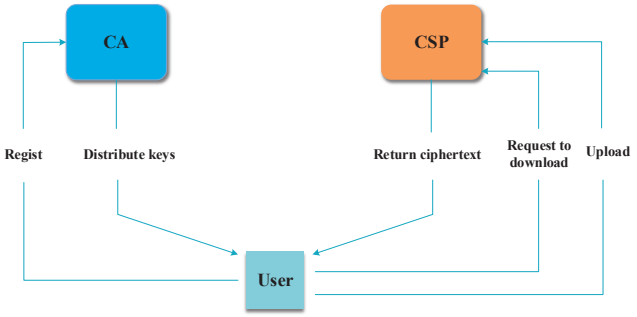


Figure 1: System model

Our model introduces a situation that users can lock their wallet files by encrypting with password and also upload the backup files to the cloud service provider. Once the local wallet files are lost, the user can request the cloud service provider to download the corresponding backup files to recover the original data. In order to prevent the adversary from reading the wallet files, each file should be encrypted with a pre-selected key. And to prevent the occurrence of the key entropy leakage problem caused by circular encryption§, we select the KDM-CPA secure symmetric encryption algorithm to perform the encryption process. When a user uploads the backup files to the cloud service provider, we combine the HMAC scheme to verify the integrity of the data transmitted between the user and the cloud service provider, which can enhance the security of the whole scheme.

Since the third-party CSP is curious, we require to keep the contents of wallet files private from CSP which means CSP cannot access the wallet data throughout the data upload and download process. We utilize a keyword-based searchable encryption algorithm to achieve this goal. In data upload phase, as the backup files are encrypted, the CSP cannot get any message about the private keys. In data download phase, the search credential submitted by the user to the CSP is calculated based on one-way functions, which means that it is impossible for the CSP to recover the original keywords. Therefore the CSP cannot obtain any valuable information from any backup file or any search credential.

As we have presented above, our scheme has two stages, the data processing and data recovery.

1) Data processing: The data processing phase mainly encrypts the wallet files and uploads the backup files to the CSP. It includes four algorithms: system setup, key generation, plaintext processing and data upload. In system setup, system parameters are confirmed according to a given security parameter. Then the communal parameters will be published. In key generation, CA runs key generation algorithm to generate the needed keys. After that, distribute them to the user through a secure channel. In plaintext processing, as shown in Figure 2, there are three steps

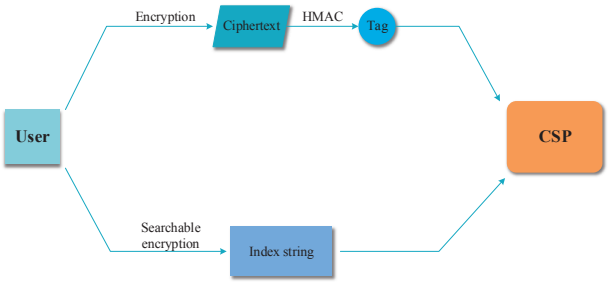


Figure 2: The specific descriptions of plaintext processing

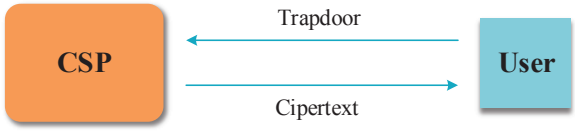


Figure 3: The specific descriptions of data download

to process the wallet files. First, encrypt the wallet file. Second, generate the message authentication code. Third, generate an index string for the wallet file based on the keywords. In data upload, the user needs to share the HMAC key with the CSP firstly, and then upload backup file, message authentication code and the index string to the CSP. CSP checks the integrity of those files.

2) Data recovery: The data recovery phase mainly requests the CSP for downloading the backup file, and decrypts the corresponding ciphertext to obtain plaintext. The data recovery contains one algorithm: data download. In data download, as shown in Figure 3, there are three steps: trapdoor generation, match retrieval and data decryption. First, the user creates the search credential (trapdoor) based on the keywords contained in the backup file and then sends it to the CSP. Second, CSP does a matching search and returns the corresponding ciphertext to the user. Third, the user calls the decryption algorithm to decrypt the ciphertext.

We will give a detailed description for our scheme in next section.

# 4 Our Construction

This section is divided into two parts. We start with the data processing phase which includes four algorithms, respectively system setup, key generation, plaintext processing and data upload. In the second part, we present the data recovery phase which contains one algorithm, that is data download.

## 4.1 Data Processing

System setup: This algorithm takes a security parameter $\lambda$ as input, and respectively generates communal system parameter $prm$ as follows:

1) Choose three arbitrary polynomials of $\lambda$: $l = l(\lambda), N = N(\lambda), m = m(\lambda)$. Employ an efficiently decodable error correcting code, whose binary generator matrix is $\boldsymbol{G}_{m\times l}$.
2) Choose the keyword dictionary parameter $\tau$ and determine the function set $\{P_K(x), F_K(x), J_K(x)\}$. For $K \in \{0,1\}^{\lambda}$, $P_K(x)$ is a family of pseudo-random permutations with domain $\{0,1\}^{\tau}$, $F_K(x)$ is a family of pseudo-random functions mapping $\{0,1\}^{\tau}$ to $\{0,1\}^{\lambda}$, and $J_K(x)$ is a family of pseudo-random functions mapping $[n]$ to $\{0,1\}$.
3) Publish $prm = \{\boldsymbol{G}_{m\times l}, P_K, F_K, J_K, l, N, m\}$ as system parameter.

Key generation: When a user registers with the CA, he/she needs to submit personal identifiable information. If the CA determines that the user is legal, CA will generate keys for the user and issue them through a secure channel; otherwise, denial of service. Given $\lambda$, CA does as follows:

1) Run $G(1^{\lambda})$ algorithm to generate the symmetric encryption key $\boldsymbol{S} \in \mathbb{Z}_2^{\lambda\times N}$.
2) Run $MAC\text{-}KeyGen(1^{\lambda})$ algorithm to generate the HMAC key $k_{mac}$.
3) Distribute $\{\boldsymbol{S}\|k_{mac}\}$ to the user through a secure channel, where $\|$ is a concatenation symbol.

Plaintext processing: After the user is successfully registered and obtains the keys issued by the CA, he/she can store and manage the wallet files on the basis of our scheme. The steps of the plaintext processing are described as follows:

**Step 1.** Encrypt wallet file:

Firstly, the user needs to encrypt the wallet file $\theta_j$, $1 \le j \le \rho$ ($\rho$ is a positive integer, represents the total number of documents) by performing the following procedures:

1) Divide the plaintext into blocks such as $\boldsymbol{M} \in \mathbb{Z}_2^{l\times N}$.
2) Randomly select a coefficient matrix $\boldsymbol{A} \in \mathbb{Z}_2^{m\times\lambda}$ and a noise matrix $\boldsymbol{E} \in Ber_{\varepsilon}^{m\times N}$.
3) Apply encryption algorithm $E(S, m)$ to encrypt $\boldsymbol{M}$ with $\boldsymbol{S}$. Obtain encrypted block $\boldsymbol{W}$ as follows:

$$\boldsymbol{C} = \boldsymbol{A}\cdot\boldsymbol{S} + \boldsymbol{E} + \boldsymbol{G}\cdot\boldsymbol{M}. \quad (2)$$

$$\boldsymbol{W} = (\boldsymbol{A}, \boldsymbol{C}). \quad (3)$$

4) All the encrypted blocks of the current file $\theta_j$ form a ciphertext file $\psi_j$. We also use $\psi_j$ on behalf of the corresponding backup file.

**Step 2.** Generate the message authentication code:

Then, the user computes authentication tag $T_j$ of ciphertext file $\psi_j$ by applying algorithm $Tag$ and using the HMAC key $k_{mac}$. The expression is as follows:

$$T_j = Tag(k_{mac}, \psi_j). \tag{4}$$

**Step 3.** Generate index string:

Lastly, the user calculates the index string according to the following procedures:

1) Select $s \in \{0,1\}^{\lambda}$,$r \in \{0,1\}^{\lambda}$ uniformly at random and keep them secret.
2) Run the *Setup* algorithm to build a keyword dictionary contains $2^{\tau}$ index-keyword pairs in the form $(i, w_i)$, where the index $i \in [2^{\tau}]$, the keyword $w_i \in \{0,1\}^*$.
3) Set a $2^{\tau}$-bit string $I_j^{'}$. If $\theta_j$ contains $w_i$, set $I_j^{'}[P_s(i)] = 1$; otherwise, set $I_j^{'}[P_s(i)] = 0$.
4) For $r_i = F_r(i)$,$i \in [2^{\tau}]$, compute the index string $I_j$ by

$$I_j[i] = I_j^{'}[i] \oplus J_{r_i}(j). \tag{5}$$

Data upload: After the user finishes processing the plaintext files, the obtained backup files and the related contents can be uploaded to the CSP. The specific interaction process between user and the CSP is as follows.

User:

1) Share the HMAC key $k_{mac}$ with the CSP through a secure channel.
2) Upload $\{\psi_j \| T_j \| I_j\}$ to CSP, where $1 \le j \le \rho$ and $\|$ is a concatenation symbol.

CSP: Run the $Verify$ algorithm to check the integrity for each ciphertext file:

If $Verify(k_{mac}, T_j, \psi_j) = 0$, return an error notification and a request for re-uploading; if $Verify(k_{mac}, T_j, \psi_j) = 1$, return the storage address $V^{'}$.

## 4.2 Data Recovery

**Data download:** Once a local wallet file is lost for some reason, user can recover this wallet file from the CSP. The steps of data download are described as follows:

**Step 1.** Trapdoor generation:

1) The user runs the *GenToken* algorithm to generate the search credential $T_{w_\mu}$ of a specific file which contains the keyword $w_\mu$. $\mu$ is the corresponding index from the dictionary. The expression is as follows:

$$T_{w_\mu} = (p, f) = (P_s(\mu), F_r(P_s(\mu))). \tag{6}$$

2) The user submits the generated search credential $T_{w_\mu}$ and the corresponding storage address $V^{'}$ to the CSP.

**Step 2.** Match retrieval:

1) The CSP runs the *Query* algorithm and computes $I_j^{'}[p] = I_j[p] \oplus J_f(j)$,$j \in [\rho]$ for all files stored in $V^{'}$.
2) If there exists $I_j^{'}[p] = 1$, CSP sends the corresponding ciphertext file $\psi_j$ to the user.

**Step 3.** Data decryption:

1) Divide the ciphertext into blocks such as $\boldsymbol{W}$.
2) The user calls the decryption algorithm $D(S, c)$ to decrypt $\boldsymbol{W}$ with $\boldsymbol{S}$.
   Obtain the matrix as follows:

$$\boldsymbol{W} = (\boldsymbol{A}, \boldsymbol{C}), \tag{7}$$

$$\boldsymbol{Q} = \boldsymbol{C} - \boldsymbol{A} \cdot \boldsymbol{S}. \tag{8}$$

3) Obtain the plaintext by applying the decodable error correcting code to decode each columns of the matrix $\boldsymbol{Q}$.

# 5 Security Analysis

In this section, we give a detailed description of the security analyses of the whole scheme.

This section is divided into four parts. First, we define security for our scheme in the sense of IND-CCA security, KDM-CCA security and trapdoor indistinguishability. Second, we present our security models. Third, we give a complete proof of our scheme according to the above security definitions and security models. Finally, we compare the usability and security of our scheme with other related schemes by using the evaluation framework proposed in [11].

## 5.1 Security Definitions

The IND-CCA security guarantees that no adversary, given an encryption of a message randomly chosen from a two-element message space determined by the adversary, can identify the message choice with probability significantly better than that of random guessing even if the decryption training was carried out in advance.

KDM-CCA security guarantees that the scheme can resist active key-dependent message attack, that is, the adversary cannot distinguish between the encryption of the key-dependent message and the encryption of a random message even if the decryption training was carried out in advance.

The trapdoor indistinguishability guarantees that an adversary cannot distinguish between the trapdoors of two challenge keywords.

## 5.2 Security Models

Let $\mathcal{A}$ be an adversary whose running time is bounded by $t$ which is polynomial in security parameter $\lambda$ and $C$ be a challenger. We consider the following three models:

1) IND-CCA Model: $\mathcal{A}$ is assumed to be an IND-CCA attacker. This model depicts the indistinguishability of our scheme under the chosen-ciphertext attack (IND-CCA).

   **Setup.** After the system is established, the generation algorithm $G(1^\lambda)$ is run by the challenger $C$. System parameter $prm$ and the symmetric encryption key $\boldsymbol{S}$ are then generated. $prm$ is given to the adversary $\mathcal{A}$ while $\boldsymbol{S}$ is kept secret from $\mathcal{A}$. $\mathcal{A}$ queries a number of arbitrary ciphertexts $c$ to the challenger $C$ and gets the corresponding decryption results.

   **Query.** $\mathcal{A}$ outputs a target plaintext pair $(M_0, M_1)$ to the challenger $C$ (Notice that none of $M_0$ nor $M_1$ has been given as an answer in Setup phase). The challenger $C$ selects a random bit $\beta \leftarrow \{0,1\}$ and creates a target ciphertext $\psi_\beta = E(\boldsymbol{S}, M_\beta)$ and returns it to the adversary $\mathcal{A}$.

   **Challenge.** $\mathcal{A}$ outputs its guess $\beta' \leftarrow \{0,1\}$.

   We define the adversary $\mathcal{A}$'s advantage in this model by $Adv^{IND-CCA}(\mathcal{A}) = |Pr[\beta = \beta'] - 1/2|$.

2) KDM-CCA Model: $\mathcal{A}$ is assumed to be a KDM-CCA attacker. This model depicts the indistinguishability of our scheme under key-dependent message chosen-ciphertext attack (KDM-CCA).

   **Setup.** After the system is established, the two generation algorithms $G(1^\lambda)$ and $MAC - KeyGen(1^\lambda)$ are run by the challenger $C$. System parameter, a fixed set of affine function class, the symmetric encryption key and the HMAC key, which we denoted by $prm$, $\Gamma$, $\boldsymbol{S} = \{S_1, S_2, ..., S_N\}$ and $k_{mac}$ are then generated respectively. $prm$ and $\Gamma$ are given to the adversary $\mathcal{A}$ while $\boldsymbol{S}$ and $k_{mac}$ are kept secret from $\mathcal{A}$. The challenger $C$ selects a random bit $\beta \leftarrow \{0,1\}$. The adversary $\mathcal{A}$ issues queries where each query of the form $(i, cc)$ where $1 \le i \le N$ to the challenger $C$. The challenger $C$ responds with $mm = D(S_i, cc)$ on the basis of the value of $\beta$.

   **Query.** $\mathcal{A}$ outputs a target query of the form $(i, f)$ where $1 \le i \le N$ and $f \in \Gamma$ to the challenger $C$ (Notice that none $f(\boldsymbol{S})$ nor $0^{|f(\boldsymbol{S})|}$ has been given as an answer in Setup phase). If $\beta = 0$, the challenger $C$ responds with $cc_0 = c_0 \| T_0$, where $c_0 = E(S_i, f(\boldsymbol{S}))$ and $T_0 = Tag(k_{mac}, c_0)$; if $\beta = 1$, the challenger $C$ responds with $cc_1 = c_1 \| T_1$, where $c_1 = E(S_i, 0^{|f(\boldsymbol{S})|})$ and $T_1 = Tag(k_{mac}, c_1)$.

   **Challenge.** $\mathcal{A}$ outputs its guess $\beta' \leftarrow \{0,1\}$.

   We define the adversary $\mathcal{A}$'s advantage in this model by $Adv^{KDM-CCA}(\mathcal{A}) = |Pr[\beta = \beta'] - 1/2|$.

3) Trapdoor indistinguishability Model: $\mathcal{A}$ is assumed to be a trapdoor-indistinguishable attacker. This model depicts the trapdoor indistinguishability of our scheme.

   **Setup.** After the system is established, system parameter $prm$ is then generated. $prm$ is given to the adversary $\mathcal{A}$.The challenger $C$ selects $s \in \{0,1\}^\lambda$, $r \in \{0,1\}^\lambda$ uniformly at random and keeps them secret. Then the challenger $C$ builds a keyword dictionary which contains $2^\tau$ index-keyword pairs in the form of $(i, w_i)$, where $i \in [2^\tau]$, $w_i \in \{0,1\}^*$. $\mathcal{A}$ queries a number of arbitrary keywords, each of which is denoted by $w$, to the challenger $C$ and gets the corresponding trapdoor $T_w$.

   **Query.** $\mathcal{A}$ outputs a target keyword pair $(w_0, w_1)$ to the challenger $C$ (Notice that none of $w_0$ nor $w_1$ has been queried in Setup phase). The challenger $C$ selects a random bit $\beta \leftarrow \{0,1\}$ and responds with a target trapdoor $T_{w_\beta} = (p_\beta, f_\beta)$, where $p_\beta = P_s(\mu_\beta)$,$f_\beta = F_r(p_\beta)$ and $\mu_\beta$ is the corresponding index of the keyword $w_\beta$.

   **Challenge.** $\mathcal{A}$ outputs its guess $\beta' \leftarrow \{0,1\}$.

   We define the adversary $\mathcal{A}$'s advantage in this model by $Adv^{Trap-IND}(\mathcal{A}) = |Pr[\beta = \beta'] - 1/2|$.

## 5.3 Security Proofs

We show that the new password-protected wallet scheme proposed in Section 4 is KDM-CCA secure. At the same time, we also prove that our scheme is IND-CCA secure and can provide trapdoor indistinguishability. We have the following theorems.

**Theorem 1.** *Our data processing algorithm is KDM-CCA secure as the LPN problem holds hard.*

*Proof.* We use the KDM-CCA Model to prove this theorem. Without loss of generality, we suppose $\beta = 0$. We consider the following three games: □

**Game0.** The same as the KDM-CCA Model.

**Game1.** The same as the Game0, except in Query phase, the challenger $C$ responds with $cc^{'} = c^{'} \| T^{'}$, where $c^{'} = E(S_i, f(\boldsymbol{S} + \boldsymbol{S}^{'}))$,$T^{'} = Tag(k_{mac}, c^{'})$ and $\boldsymbol{S}^{'} \in \mathbb{U}_2^{\lambda \times N}$.

**Game2.** The same as the Game1, except in Query phase, the challenger $C$ responds with $cc^{''} = c^{''} \| T^{''}$, where $c^{''} = E(S_i, \boldsymbol{R})$,$T^{''} = Tag(k_{mac}, c^{''})$ and $\boldsymbol{R} \in \mathbb{U}_2^{l \times N}$.

For Game0 and Game1, as noise matrix $\boldsymbol{E} \in Ber_{\varepsilon}^{m \times N}$ is randomly selected, the adversary cannot distinguish the ciphertext of $f(\boldsymbol{S} + \boldsymbol{S}^{'})$ with the ciphertext of $f(\boldsymbol{S})$. Thus the Game0 and Game1 are computationally indistinguishable.

For Game1 and Game2, as solving the LPN problem is difficult, the adversary cannot distinguish the ciphertext of $f(\boldsymbol{S} + \boldsymbol{S}^{'})$ with the ciphertext of $\boldsymbol{R}$. Thus the Game1 and Game2 are computationally indistinguishable.

In summary, Game0 and Game2 are computationally indistinguishable. The adversary $\mathcal{A}$'s advantage in this model is negligible in $\lambda$. Therefore, our data processing phase is KDM-CCA secure.

**Theorem 2.** *Our data processing algorithm is IND-CCA secure.*

*Proof.* According to [10], KDM-CCA security implies IND-CCA security. From **Theorem 1**, our data processing algorithm is proved to be KDM-CCA secure. So the advantage of the adversary $\mathcal{A}$ to distinguish the two ciphertext in IND-CCA Model is also negligible, which means that the encryption algorithm in the proposed scheme is IND-CCA secure. □

**Theorem 3.** *The proposed scheme satisfies the property of trapdoor indistinguishability, if $s, r$, are kept secret.*

*Proof.* When the adversary $\mathcal{A}$ submits a target keyword pair $(w_0, w_1)$ to the challenger ? in the Trapdoor indistinguishability Model, the challenger $C$ will find the corresponding index $\mu_\beta$ of the keyword $w_\beta$ from the dictionary and respond with the corresponding trapdoor $T_{w_\beta}$. The calculation process is as follows:

$$T_{w_\beta} = (p_\beta, f_\beta) = (P_s(\mu_\beta), F_r(P_s(\mu_\beta))). \tag{9}$$

If $s \in \{0,1\}^*$, $r \in \{0,1\}^*$ are kept secret, according to the one-way property of pseudo-random permutations $P_K(x)$ and pseudo-random functions $F_K(x)$, the advantage of the adversary $\mathcal{A}$ to distinguish the two trapdoors is negligible. Therefore, the proposed scheme is trapdoor-indistinguishable. □

**Theorem 4.** *The cloud service provider(CSP) cannot get any information of the backup files or the search credentials.*

*Proof.* Though the third-party cloud service provider is curious, we can prove that in our scheme CSP cannot get any information during the whole process. From **Theorem 2**, our data processing algorithm is proved to be IND-CCA secure. So CSP cannot distinguish between any two ciphertexts which means that CSP cannot obtain any valuable data during the data processing phase. From **Theorem 3**, it is impossible for the CSP to recover the original keywords from the search credential submitted by the user as $s, r$ holds secret. Therefore our scheme keeps the contents of wallet files private from the CSP. □

### 5.4 Security Evaluation

In this subsection, we use the evaluation framework proposed by Eskandari *et al.* in citeeskandari2015first to analyze the usability and security of our scheme. This evaluation framework considers the attacks that occur in practice, such as malware attack, physical theft, physical observation, password loss and so on. In addition, it considers the usability, such as accessibility and cross-device portability. According to these evaluation criteria, we make a comparison with several related schemes. The security evaluation results are shown in Table 1.

As can be seen from Table 1, we compare our scheme with the scheme proposed in [5], the scheme proposed in [24] and the scheme proposed in [19]. In Table 1, the black dot(·) means that the scheme can satisfy this property. The black circle(○) means that the scheme can partially satisfy this property. Empty means that the scheme cannot satisfy this property.

From the evaluation results, our scheme satisfies many properties. First of all, users can unlock the wallet files only by entering a password token, so we claim that our scheme is immediate access to funds. Second, since our encryption algorithm is KDM-CCA secure, our scheme can resist key leakage when a malicious attack or physical theft occurs. Third, the backups of the wallet files are stored in CSP, so even if malware attacks such as ransomware attacks occur, users do not have to worry about the security of the wallet files. Furthermore, wallet files can also be obtained on other devices by interacting with CSP, which achieving cross-device portability. Finally, the key generation is executed by CA, and CA distributes the keys through a secure channel, which avoiding physical observation attack. Beyond the evaluation results, our scheme can recover the private keys via backup files stored in CSP, which is very important for preventing the loss of assets.

## 6 Efficiency Analysis

In this section, we present the efficiency analyses of the whole scheme.

This section is divided into two parts. We start with the data volume analysis which includes local storage, data upload phase and data download phase. In the second part, we present the performance evaluation.

Table 1: Security evaluation

| Schemes | Malware Resistant | Keys Kept Offline | No Trusted Third Party | Resistant to Physical Theft | Resistant to Physical Observation | Resilient to Password Loss | Resilient to Key Churn | Immediate Access to Funds | No New User Software | Cross -device Portability |
|---|---|---|---|---|---|---|---|---|---|---|
| Our scheme | • | ○ | | • | • | | • | • | | • |
| Scheme in [5] | ○ | • | • | | | | | | | • |
| Scheme in [24] | | | • | | • | • | • | • | | |
| Scheme in [19] | ○ | ○ | | • | • | • | • | • | | • |

## 6.1 Data Volume Analysis

In our scheme, the amount of data volume is divided into three parts, respectively derived from the KDM-CPA secure symmetric encryption algorithm, the HMAC algorithm and the symmetric keyword-based searchable encryption algorithm.We will successively analyze the data volume in local storage, in data upload phase and in data download phase.

When applying the KDM-CPA secure symmetric encryption algorithm, the user stores the symmetric key which occupies $|\boldsymbol{S}|$ bits of storage locally. When applying the HMAC algorithm, the user stores the HMAC key which occupies $|k_{mac}|$ bits of storage locally. And when applying the searchable encryption algorithm, the user stores $s$, $r$, which occupies $2\lambda$ bits, plus an index dictionary locally. To make the discussion more convenient, we use $\Phi$ to denote the index dictionary. In summary, the user stores $|\boldsymbol{S}| + |k_{mac}| + 2\lambda + |\Phi|$ bits locally.

Assume that the user uploads $\rho$ files per time. The shared HMAC key occupies $|k_{mac}|$ bits. The ciphertexts obtained by applying the KDM-CPA secure symmetric encryption algorithm occupies $|\Sigma_{j\in\rho}(\psi_j)|$ bits in total. In the practical application, the output length of HMAC-MD5 algorithm is fixed to 128bits. So the total authentication tags and index strings occupies $\rho \cdot (2^{\tau} + 128)$ bits. In summary, the user needs to send total $|\Sigma_{j\in\rho}(\psi_j)| + \rho \cdot (2^{\tau} + 128) + |k_{mac}|$ bits to the CSP.

Assume that only one trapdoor is allowed to submit at a time. The user needs to send the trapdoor, which occupies $(\tau + \lambda)$ bits, to the CSP.

The whole data volume analysis is shown in Table 2.

As can be seen from the contents of the Table 2, our scheme has a low total data volume. The user stores $|\boldsymbol{S}| + |k_{mac}| + 2\lambda + |\Phi|$ bits locally and sends $|\Sigma_{j\in\rho}(\psi_j)| + \rho \cdot (2^{\tau} + 128) + |k_{mac}|$ bits to the CSP in data upload phase and sends $(\tau+\lambda)$ bits to the CSP in data download phase.

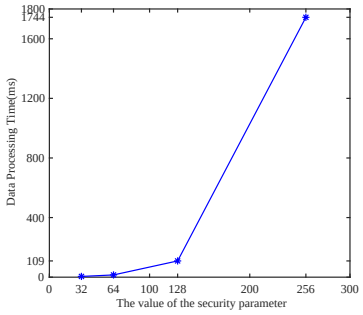


Figure 4: Performance evaluation of data processing phase

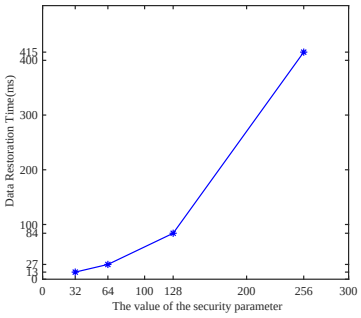


Figure 5: Performance evaluation of data recovery phase

## 6.2 Performance Evaluation

In this section, we use the Java security APIs to implement all cryptographic operations in our scheme. All algorithms are implemented by using Java language. The simulation is performed on a laptop computer with a Core i3-2310M, 2.10 GHz processor. The simulation results are shown in Figure 4 and Figure 5.

The application scenario of our scheme is for single-user to store and manage personal wallet files. As we mentioned earlier, our scheme has two stages, respectively the data processing phase and data recovery phase. During the simulation, we ignore the time cost of communications between users and the CSP so that the results below will not be good enough than theoretical results.

Table 2: Data volume analysis

| Phase | Enc | HMAC | Index | Trapdoor | Total |
|---|---|---|---|---|---|
| **Local storage** | $\|\boldsymbol{S}\|$ | $\|k_{mac}\|$ | $\|\Phi\|$ | $2\lambda$ | $\|\boldsymbol{S}\| + \|k_{mac}\| + 2\lambda + \|\Phi\|$ |
| **Data upload** | $\|\Sigma_{j\in\rho}(\psi_j)\|$ | $\rho \cdot 128 + \|k_{mac}\|$ | $2^{\tau} \cdot \rho$ | / | $\|\Sigma_{j\in\rho}(\psi_j)\| + \rho \cdot (2^{\tau} + 128) + \|k_{mac}\|$ |
| **Data download** | / | / | / | $(\tau + \lambda)$ | $(\tau + \lambda)$ |

In data processing phase, the user needs to perform one encryption operation, one HMAC operation and several binary bitwise operations. Since binary bitwise operation is fast, we ignore the computation time of it. As shown in Figure 4, the time spent in the data processing phase mainly includes encryption and hash. In data recovery phase, the user submits the search credential to the CSP to obtain the backup file. The user needs to compute pseudo-random permutation and pseudo-random function respectively one time firstly. And then call the decryption algorithm to recover the plaintext. As mentioned above, we still ignore the computation time of binary bitwise operations. So the time spent in the data recovery phase mainly includes decryption as shown in Figure 5.

Because the size of plaintext has a polynomial relationship with the security parameter, as the security parameter increase, the size of plaintext increases. So as the security parameter increase, the time spent is also increasing. As can be seen from both the two figures, when the security parameter $\lambda$ exceeds 128, the time consumed increases significantly. Without loss of generality, when the security parameter $\lambda$ is 80, it is efficient to use our scheme to store and manage the wallet files.

# 7 Conclusions

Aiming at enhancing the security of password-protected wallet in Bitcoin, we put forward a new password-protected wallet scheme utilizing backups. Specifically, the encryption algorithm is able to resist the active KDM attack. So the user can rest assured that the backup files are securely encrypted, without fear of key information disclosure. And the encrypted backup files will be uploaded to the CSP to prevent local data loss. Although we introduce a semi-trusted third-party cloud server, we prove that the cloud server cannot get any detail about the backup files or the search credentials.

We also give a detailed security analysis and efficiency analysis of the proposed scheme. The analyses show that the proposed scheme is secure and efficient. In the future, the more transactions are initiated, the more key pairs will be stored in personal wallet. The encryption of key-dependent messages is inevitable. So the proposed scheme will play an important role in improving the security of password-protected wallet, providing privacy protection and promoting the development of Bitcoin economy. Our scheme is only applicable to single user scenario currently, and the scheme for multi-users is worthy of further study.

# Acknowledgments

This work was supported in part by the National Key Research and Development Program of China (No. 2016YFB0800601), the Natural Science Foundation of China (No. 61303217,61502372,61671360), the Natural Science Foundation of Shaanxi province (No. 2013JQ8002,2014JQ8313). The authors gratefully acknowledge the anonymous reviewers for their valuable comments.

# References

[1] J. Alperin-Sheriff and C. Peikert, "Circular and kdm security for identity-based encryption," in *Public Key Cryptography (PKC'12)*, pp. 334–352, 2012.

[2] B. Applebaum, "Key-dependent message security: Generic amplification and completeness," in *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, pp. 527–546, 2011.

[3] B. Applebaum, D. Cash, C. Peikert, and A. Sahai, "Fast cryptographic primitives and circular-secure encryption based on hard learning problems," in *Advances in Cryptology (CRYPTO'09)*, pp. 595–618, 2009.

[4] M. Backes, B. Pfitzmann, and A. Scedrov, "Key-dependent message security under active attacks–brsim/uc-soundness of dolev–yao-style encryption with key cycles," *Journal of Computer Security*, vol. 16, no. 5, pp. 497–530, 2008.

[5] T. Bamert, C. Decker, R. Wattenhofer, and S. Welten, "Bluewallet: The secure bitcoin wallet," in *International Workshop on Security and Trust Management*, pp. 65–80, 2014.

[6] J. Black, P. Rogaway, and T. Shrimpton, "Encryption-scheme security in the presence of key-dependent messages," in *Selected Areas in Cryptography*, vol. 2595, pp. 62–75, 2002.

[7] Y. C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *ACNS*, vol. 5, pp. 442–455, 2005.

[8] K. Christidis and D. Michael, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[9] P. Dikshit and K. Singh, "Efficient weighted threshold ecdsa for securing bitcoin wallet," in *Asia Security and Privacy (ISEASP'17)*, pp. 1–9, 2017.

[10] Y. Dodis, S. Goldwasser, Y. T. Kalai, C. Peikert, and V. Vaikuntanathan, "Public-key encryption schemes with auxiliary inputs," in *TCC*, vol. 5978, pp. 361–381, 2010.

[11] S. Eskandari, J. Clark, D. Barrera, and E. Stobert, "A first look at the usability of bitcoin key management," *Cryptography and Security*, 2015. DOI: 10.14722/usec.2015.23015

[12] E. Gaetani, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "Blockchain-based database to ensure data integrity in cloud computing environments," in *ITASEC*, pp. 146–155, 2017.

[13] P. Gauravaram, S. Hirose, and S. Annadurai, "An update on the analysis and design of nmac and hmac functions," *International Journal of Network Security*, vol. 7, no. 1, pp. 49–60, 2008.

[14] 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.

[15] S. Goldfeder, R. Gennaro, H. Kalodner, J. Bonneau, J. A. Kroll, E. W. Felten, and A. Narayanan, "Securing bitcoin wallets via a new DSA/ECDSA threshold signature scheme," *Sematic Schlar*, 2015. (`https://www.semanticscholar.org/paper/Securing-Bitcoin-wallets-via-a-new-DSA-\%2F-ECDSA-Goldfeder-Narayanan/8b9b7e1fb101a899b0309ec508ac5912787cc12d`)

[16] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[17] L. Guo, X. Li, and J. Gao, "Multi-party fair exchange protocol with smart contract on bitcoin," *International Journal of Network Security*, vol. 20, 2018.

[18] G. Gutoski and D. Stebila, "Hierarchical deterministic bitcoin wallets that tolerate key leakage," in *International Conference on Financial Cryptography and Data Security*, pp. 497–504, 2015.

[19] S. He, Q. Wu, X. Luo, Z. Liang, D. Li, H. Feng, H. Zheng, and Y. Li, "A social-network-based cryptocurrency wallet-management scheme," *IEEE Access*, vol. 6, pp. 7654–7663, 2018.

[20] M. H. Ibrahim, "Securecoin: A robust secure and efficient protocol for anonymous bitcoin ecosystem," *International Journal of Network Security*, vol. 19, no. 2, pp. 295–312, 2017.

[21] S. M. Khan and K. W. Hamlen, "Penny: Secure, decentralized data management," *International Journal of Network Security*, vol. 16, no. 5, pp. 340–354, 2014.

[22] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE Symposium on Security and Privacy (SP'16)*, pp. 839–858, 2016.

[23] I. C. Lin and T. C. Liao, "A survey of blockchain security issues and challenges," *International Journal of Network Security*, vol. 19, no. 5, pp. 653–659, 2017.

[24] Y. Liu, R. Li, X. Liu, J. Wang, L. Zhang, C. Tang, and H. Kang, "An efficient method to enhance bitcoin wallet security," in *11th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID'17)*, pp. 26–29, 2017.

[25] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. (`https://bitcoin.org/bitcoin.pdf`)

[26] A. A. Selcuk, E. Uzun, and M. R. Pariente, "A reputation-based trust management system for p2p networks," *International Journal of Network Security*, vol. 6, no. 2, pp. 227–237, 2008.

[27] F. Tschorsch and S. Björn, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[28] Y. Zhu and Y. Hu, "Surepath: An approach to resilient anonymous routing," *International Journal of Network Security*, vol. 6, no. 2, pp. 201–210, 2008.

## Biography

**Liyan Wang** received the B.S. degree in College of Information Science and Engineering from Shandong Agricultural University in 2015. She is currently studying for the M.S. degree in School of Telecommunications Engineering from Xidian University. Her research interests include information security, Bitcoin, blockchain and key-dependent message security.

**Juntao Gao** received his PhD degree of Cryptography in December 2006 at Xidian University. He is currently an associate professor at School of Telecommunications Engineering, Xidian University. He is also a member of Chinese Association for Cryptologic Research. His research interests include information security, stream cipher and cryptographic functions, pseudorandom sequences and blockchain.

**Xuelian Li** received her PhD degree of Cryptography in December 2010 at Xidian University. Currently, she is an associate professor at School of Mathematics and Statistics, Xidian University. Her research interests include information security, cryptographic functions and stream ciphers.