International Journal of Information Security (2020) 19:245–255
https://doi.org/10.1007/s10207-019-00476-5

SPECIAL ISSUE PAPER

# Secure hierarchical Bitcoin wallet scheme against privilege escalation attacks

Chun-I Fan[1] · Yi-Fan Tseng[2] · Hui-Po Su[1] · Ruei-Hau Hsu[1] · Hiroaki Kikuchi[3]

Published online: 8 November 2019


**Abstract**
As the rising popularity of Bitcoin, people tend to use Bitcoin wallets to manage the keys for spending or receiving funds. Instead of generating randomly pairs of keys, which may need higher space complexity for key management, hierarchical deterministic (HD) wallets derive all the keys from a single seed, which is sufficient to recover all the keys, to reduce the complexity of key management. In an HD wallet, it allows users to generate child public keys from the parent public keys without knowing any of the corresponding private keys. This feature allows a permitted auditor to derive all the public keys for auditing. However, this feature makes HD wallets suffered from so-called privilege escalation attacks, where the leakage of any child private key along with its parent public key will expose the other child private keys. To confront with this security flaw, we propose a novel HD wallet scheme that gives out a signature with trapdoor hash functions instead of directly giving private keys for signing. Since it conceals private keys from any child nodes, it can prevent from privilege escalation attacks. Nevertheless, the proposed scheme also provides unlinkability between two public keys to achieve anonymity of user identities and high scalability to the derivations of huge amount of keys. Thus, the proposed scheme achieves user anonymity, public key derivation, and high scalability.

**Keywords** Bitcoin · HD wallets · BIP032 · Privilege escalation attacks · Schnorr signature · Trapdoor hash function

## 1 Introduction

As Bitcoin was first introduced in 2008 [16], more and more people intend to use this state-of-the-art decentralized currency since its benefits are significant [19]. A Bitcoin system can simply be viewed as an implementation of a public key signature system combined with blockchain technology, where no trusted third party is necessary. In such a system, each of transaction is kept and shared simultaneously across plenty of nodes in a distributed ledger which is publicly maintained by the whole group of Bitcoin network users, called "miners". Compared to e-Cash [21], Bitcoin is totally decentralized, since the miners benefit from an incentive mechanism which encourages them to support the network without any central authority. The advantage of decentralization in Bitcoin is that all transactions are transparent while the users' personal identities are hidden [3,8]. Furthermore, the users do not require to trust any third party [15] to exchange funds among them, thus a single point of failure [13] can be avoided [2].

In a Bitcoin system, numerous pairs of keys will be randomly generated by users to pay or receive funds. In order to achieve anonymity, each key is used only once to make the transactions unlinkable. However, this mechanism makes it more complex for key managements, especially when a user may own many pairs of keys. To confront with the key management issue, several different programs called "Bitcoin wallets" have been created. Bitcoin wallets can be separated into different ways according to protection mechanisms for

✉ Chun-I Fan
cifan@mail.cse.nsysu.edu.tw

Yi-Fan Tseng
yftseng1989@gmail.com

Ruei-Hau Hsu
rhhsu@mail.cse.nsysu.edu.tw

Hiroaki Kikuchi
kikn@meiji.ac.jp

[1] Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan

[2] Department of Computer Science, National Chengchi University, Taipei 11605, Taiwan

[3] Department of Frontier Media Science, Meiji University, Tokyo 164-8525, Japan

securing private keys [9]. One simple method supported by the original Bitcoin client is to store private keys in a local wallet file. Nevertheless, a malware may lead to the loss of funds in an infected computer or local storage device since all of the key would be leaked once the computer/device has been hacked. Because of the malicious intent of illegal users, password-protected wallets is much more preferable, which allow users to encrypt their wallet files using a key generated by their passwords. Another shortcoming is that a new key needs to be generated for each new transaction since key reuse would cause the linkability. Therefore, each key must be backed up frequently to ensure that it can still be accessed once the wallet file is compromised.

In order to simplify the backup procedures, wallets with password-derived keys, known as Hierarchical Deterministic (HD) Wallets, are proposed in BIP32 standard [23]. In an HD wallet mechanism, a master key will first be derived from a random seed, and the extend it to all the child keys. A seed can be easily stored in any offline key storage (e.g., paper wallet) for backup, thus frequent backups are avoided. Hierarchical deterministic wallets allow the parent keys to extend child keys and the child keys extend grandchild keys, and so on. That makes an HD wallet being able to be presented a tree-like structure. A more impressive feature of HD wallets is that a child's public key can be derived from a parent public key without any private information. This feature allows one to issue a chain of receive-only addresses in an untrusted environment. Those addresses will be used to receive payments in an insecure server, but it would not be allowed to access the merchant's account to spend funds. This mechanism seems convenient for key management; however, it suffers from one kind of attack, called *privilege escalation attack* [6]. In such attack, once a derived child private key along with a parent extended public key at any level is leaked, it will lead to the leakage of all the child private keys. What's more, the master private key might also be recovered to the adversary due to the reversible relationship between child keys and parent keys. Privilege escalation attacks are widely discussed in several Bitcoin forums [5] and research community [11] and are mentioned in the specification of BIP32 [23] as well. We refer the reader to [5,6,11] for more details about privilege escalation attacks.

In this paper, we survey the related works [7,10,12,23] that attempted to fix the vulnerability cause by privilege escalation attacks. In BIP32 standard [23], the risk of parent key leakage has been resolved by adding hardened key derivation functions in the recent changes. The hardened key derivation functions combine parent private extended keys with an index to derive the child key rather than using the parent public key. However, this kind of derivation breaks the relationship between parent public and child extended keys. Therefore, it is unable to generate the child public key without knowing the parent private key, which shall be an important feature of HD wallets. In 2015, Gutoski et al. proposed a new HD wallet scheme [12] which tolerates key leakage. Instead of deriving all keys from one single master key, in their scheme, $m$ master keys are used to derive child keys so that it is allowed to tolerate the leakage of up to $m$ private keys. Nevertheless, they sacrifice the flexibility of generating child keys since the number of $m$ is determined once a wallet is created. Furthermore, it would encounter an exponential growth of public key size. In the same year, Goldfeder et al. [10] presented a threshold signature scheme that realizes deterministic wallets for shared addresses. In their scheme, the master private key is secretly shared among $t$ participants. Thus, participants can generate their new public and private keys via the derivation of their shares. Nevertheless, once more than $t$ participants collude with each other, they are able to reconstruct the master private key. In 2017, Courtois et al. [7] proposed a key management technique combined with two distinct elliptic curve cryptography (ECC) arithmetic properties (i.e., addition and multiplication) to derive keys. However, their scheme is still suffered from privilege escalation attacks, since the construction is similar to the original one.

To deal with the aforementioned issues, we proposed a hierarchical deterministic wallet based on Schnorr signature [20] and trapdoor hash functions. In our wallet scheme, when a user wants to withdraw a fund, she/he needs to prove the ownership of the public key associated to the fund. Therefore, instead of giving a private key to the child node, i.e., the child key holder in the tree-like structure, we give the child node a signature associated to the corresponding public key of the child node. This is how we achieve security against privilege escalation attacks, since no private keys are given to the child nodes. Moreover, the unforgeability of Schnorr signature implies that the private keys cannot be recovered from signatures. Furthermore, our scheme also owns the feature of public derivability, i.e., deriving child public keys from parent public key. Thus our scheme can be deployed in untrusted surroundings, and even remain secure against an untrusted third-party auditor who may intend to view every transaction related to the wallet in detail.

## 2 Preliminaries

We present some preliminaries in this section, including Schonorr signature scheme [20], trapdoor hash function [22], hash-based message authentication code [4] and hierarchical key derivation function.

### 2.1 Schnorr signature

A Schnorr signature is said to be a secure signature scheme if discrete logs is secure and $H$ is modeled as random oracle [17].

**Definition 1** Let $\mathbb{G}$ be a group of large prime order $q$, with generator $g$, i.e., $g^q = 1 \mod p$. $q$ is chosen as a subgroup of $\mathbb{Z}_p^\times$, a multiplicative cyclic groups of integers modulo prime $p$, such that $q \mid p-1$. An one-way hash function $H : \{0,1\}^* \to \mathbb{Z}_q$.

- **KeyGen:** Choose a secret signing key $x \in \mathbb{Z}_q^\times$ and compute corresponding public key $y = g^x \in \mathbb{G}$ for publicly verification. A public-private key pair is denoted as $(g^x, x)$.
- **Signing:** To sign a message $M \in \{0,1\}^*$, choose a random $k \in \mathbb{Z}_q^\times$, $r = g^k \mod p$. Let the signature signed on $M$ denoted as the pair $(s, e)$, where $e = H(r \parallel M) \mod p$ (note that $r$ is represented as $\{0,1\}^*$) and $s = k - xe \mod q$.
- **Verifying:** Let $r_v = g^s y^e \mod p$ and $e_v = H(r_v \parallel M)$. Check if $e_v = e$ then the signature pair $(s, e)$ is verified, else $\perp$.
- **Correctness:** The proof of correctness can be computed to see if the signed message $e_v$ equals the verified message $e$ as follows:

$$\begin{aligned} r_v &= g^s y^e \\ &= g^{k-xe} g^{xe} \\ &= g^k \\ &= r. \end{aligned}$$

### 2.2 Trapdoor hash function

A trapdoor hash function is a provably secure one-way hash function which is collision resistant and non-reversible (i.e., it is hard to find its inverse value) without any special information or a secret key. While for a trapdoor holder, he can take any message as input and find any collision with the same hash value. A trapdoor hash function scheme consists of three generating algorithms (*KeyGen*, *HashGen*, *ColGen*) and has two properties. A trapdoor hash function can be defined as follows:

**Definition 2** Let $HK$ be a publicly known hash key and $TK$ be a private trapdoor key of the trapdoor hash function. Anyone can compute a trapdoor hash function with the public hash key $HK$, denoted by $TH_{HK}()$. There are three generating algorithms:

- **KeyGen:** The probabilistic polynomial-time (PPT) algorithm takes a security parameter $\lambda$ as input. It outputs a pair of trapdoor hash keys ($HK$, $TK$).
- **HashGen:** The PPT algorithm takes a message $m$, a random number $r$, and public key $HK$ as input. It outputs the hash value $TH_{HK}(m, r)$.
- **ColGen:** The PPT algorithm takes the message $m$, the random number $r$ and an message $m'$, that $m' \neq m$ and private trapdoor key $TK$. It outputs a collision value $c$ which can be taken as input and compute the same hash value with the public key $HK$, such that

$$TH_{HK}(m, r) = TH_{HK}(m', c).$$

**Definition 3** A trapdoor hash function has the following properties.

- **Collision resistant:** There does not exist a PPT algorithm $A$ that inputs a message $m$, a random number $r$ and the public key $HK$ outputs the collision pair $(m', c)$ satisfying

$$TH_{HK}(m, r) = TH_{HK}(m', c).$$

- **Semantic security:** A trapdoor hash function is said to be semantically secure if it does not disclose any information about message $m$, even if a trapdoor hash value $C$ is given. That is, $TH_{HK}(m, r) = C$ and $TH_{HK}(m', c)$ are computationally indistinguishable.

### 2.3 Hash-based message authentication code (HMAC)

A hash-based message authentication code (HMAC) is a specific type of message authentication code (MAC) scheme based on cryptographic hash function. Different from public hash functions in common use which any person can compute without any secret key involved in the calculation, HMAC can utilize any cryptographic hash function like SHA-2 or MD5 to combine with MAC mechanism which is used to authenticate a message using a secret key. The HMAC scheme enjoys several significant advantages of security properties such as collision resistance or protection against some kinds of attacks like birthday attacks, extension attacks, and key recovery attacks.

**Definition 4** HMAC has the following properties.

- **One-way property:** It is computationally infeasible to find $M$ from $H(M)$.
- **Weak collision resistance:** For any given $M$, it is computationally infeasible to determine another different message $M'$ such that $H(M') = H(M)$.
- **Strong collision resistance:**It is computationally infeasible to determine a message pair $M$, $M'$ such that $H(M) = H(M')$.

### 2.4 Hierarchical deterministic key derivation

In order to avoid the requirement of backups frequently, deterministic wallets permit that one can generate a master key to derive all the other keys from a single seed. A seed is generated from a pseudo-random number generator and stored in an offline key storage secretly. In deterministic wallets, only a single backup for the seed at creation time is sufficient to recover all the derived keys. However, deterministic wallets only perform a single chain of keys that are not proper to some use cases since deterministic wallets does not allow partial reveal of keys. Hence, in BIP32 standards, hierarchical deterministic wallets (HD wallets) are proposed to derive keys in a tree structure so that HD wallets perform many chains of keys for the additional business purposes. BIP32 implements hierarchical key derivations using secp256k1's elliptic curve to calculate the derivation of keys. In the following, we will show how to extend child keys in the defined key derivation (CKD) functions.

**Definition 5** There is a 32-byte nonce called chain code used to extend keys, denoted as $c$. Given an extended private key represented as $(k, c)$ and index $i$, with $k$ the normal private key, and c the chain code, it is possible to compute corresponding private child extended key. In the same way, given extended public key $(K, c)$ and an index $i$, corresponding public child extended key can be computed. $Point(p)$ is defined as the coordinate pair resulting from EC point multiplication of the secp256k1 base point with the integer $p$.

- **Master key generation:** A pair of master key $(m, M)$ is generated from a seed of chosen length between 128 and 512 bits.
  1. Generate a seed $r$ from a pseudo-random number generator (PRNG) of length between 128 and 512 bits (256 bits is advised).
  2. Let $I = \text{HMAC-SHA512}\ (s, r)$, where $s =$ "Bitcoin seed" is a byte string.
  3. Let $I_L$, $I_R$ be left sequence and right sequence of $I$, respectively, after splits $I$ into two 32-byte sequences.
  4. $I_L$ is the master secret key $m$.
  5. $I_R$ is the master chain code.
  6. Master public key $M = Point(m)$.
- $((K_{par}, c_{par}), i) \rightarrow (K_i, c_i)$: The function computes a child extended public key from the parent extended public key.
  1. Let $I = \text{HMAC-SHA512}\ (c_{par}, K_{par} \parallel i)$.
  2. Let $I_L$, $I_R$ be left sequence and right sequence of $I$, respectively, after splits $I$ into two 32-byte sequences. If $I_L$ is invalid in case of $I_L \geq n$, where $n$ is the order of $\mathbb{G}$ of secp256k1, one should choose another value of $i$.
  3. Returned child key $K_i = Point(I_L) + K_{par}$. Note that the next value for $i$ should be proceeded if $K_i$ is the point of $\infty$.
  4. Returned chain code $c_i = I_R$.
- $((k_{par}, c_{par}), i) \rightarrow (k_i, c_i)$: The function computes a child extended private key from the parent extended private key.
  1. Let $I = \text{HMAC-SHA512}\ (c_{par}, Point(k_{par}) \parallel i)$.
  2. Let $I_L$, $I_R$ be left sequence and right sequence of $I$, respectively, after splits $I$ into two 32-byte sequences. If $I_L$ is invalid in case of $I_L \geq n$, where $n$ is the order of $\mathbb{G}$ of secp256k1, one should choose another value of $i$.
  3. Returned child key $k_i = I_L + k_{par}$. Note that the next value for $i$ should be proceeded if $k_i$ equals to zero.
  4. Returned chain code $c_i = I_R$.

### 2.5 Privilege escalation attack

A privilege escalation attack in Bitcoin HD wallets is a severe attack that a collusion between a child key owner and the auditor may cause all keys in the wallet compromised. In the following, we will present a Bitcoin HD wallet in BIP32 suffered from a privilege escalation attack.

**Definition 6** For a given child private key $k_i$, parent public extended key $(K_{par}, c)$ and the corresponding index $i$, the adversary can recover parent private key $k_{par}$ by computing

$$k_{par} = k_i - I_L,$$

where $\text{HMAC-SHA512}(c_{par}, K_{par} \parallel i) = (I_L \parallel I_R)$.

## 3 The proposed algorithm

In a hierarchical deterministic wallet, all of the child keys are derived from a single master secret key. However, the master secret key might be disclosed to adversary if any single derived key and master public key are leaked. We propose a new hierarchical wallet signature algorithm secure against a collusion between the auditor and any delegated child key owner. In the proposed algorithm, we adopt Schnorr signature rather than the original elliptic curve digital signature (ECDSA) and choose the same parameters as Schnorr signature. We distribute the signature value containing a trapdoor hash value which is changeable for a trapdoor key holder in substitute for giving out any child key to a specific child. The notations used in our algorithm are defined in Table 1. We will describe our algorithm in details as follows.

In the proposed algorithm, any user will have a position in the hierarchy defined by the index of users, denoted as

**Table 1** The notations

| Notation | Meaning |
|---|---|
| $E_{\text{index}}$ | A user with a position in the hierarchy tree defined by the index |
| *param* | The public parameters $(\mathbb{G}, p, q, g)$ |
| *msk* | The master secret key |
| *mpk* | The master public key |
| $TH_{HK}()$ | A secure trapdoor hash function |
| *THK* | A pair of trapdoor hash key |
| $i$ | The index in hierarchy tree |
| $ssk_i$ | A child private extended key |
| $spk_i$ | A child public extended key |
| $x_i$ | A child private signing key |
| $y_i$ | A child public verification key |
| $m$ | An arbitrary message |
| $c, d, k$ | The random numbers |
| $\tilde{m}$ | A message containing information of receiver's address and payment. |
| $\sigma$ | A signature for Bitcoin payment or used to delegation |

$E_{\text{index}}$. We assume that $(E_0, E_t, E_n, T)$ represents a root KGC, lower-level KGCs, regular users, and a auditor, respectively. The root KGC is the highest-level ancestor of all users in the hierarchy tree who has capability to derive all his descendants. The lower-level KGCs are delegated to generate the signature of his child while he must receive his own signature value and some parameters from the root KGC. However, any lower-level KGC is not permitted to generate the signatures of his parents and siblings. A regular user is only allowed to generate his own signature. An auditor is permitted to derive public keys of all users for auditing without knowing any private key.

### 3.1 Initialization($1^\lambda$) → (*param*)

This algorithm takes security parameter $1^\lambda$ as input and compute $\mathbb{G}$ to be a group of large prime order $q \in \mathbb{Z}_p^*$, where $q \mid p-1, |p| = K$, with a generator $g$ of $\mathbb{G}$. System parameters $(\mathbb{G}, p, q, g)$ are publicly distributed, denoted as $param$.

### 3.2 Setup(*param*) → (*msk*, *mpk*, $THK_0$, $n$, $e_0$)

The root KGC (i.e., an HD wallet owner) performs the following steps:

1. Choose a 64 bytes root seed $s \in \mathbb{Z}_p$ from a secure pseudo-random generator.
2. Select master private key $x = s \mod q$ and compute master public key $y = g^x \mod p$.
3. Randomly pick $k \in \mathbb{Z}_q^*, c, d \in \{0,1\}^{256}$.
4. Compute $\kappa = g^k$
5. Generate $THK_0 = (HK_0, TK_0)$, where $HK_0$ is publicly distributed. Let $TH_{HK_0}() : \{0,1\}^* \to \mathbb{Z}_q$ be a secure trapdoor hash function.
6. Select $r \in \mathbb{Z}_q^*$, $M \in \{0,1\}^*$ and compute $e_0 = TH_{HK_0}(r, M)$.
7. Set $n = (c, d, \kappa)$.

### 3.3 HMAC($k$, $m$) → ($h_L \parallel h_R$)

1. Compute $h = \text{HMAC-SHA512}\ (k, m)$.
2. Split $h$ into two 256-bits sequences $h_L$ and $h_R$.
3. Return $(h_L \parallel h_R)$.

### 3.4 Set-Child-Private-Key($ssk_{par}$, $i$) → $ssk_i$

Let the $i$th child be denoted as $E_i$ and $E_i$'s parent be denoted as $E_{par}$. The root KGC performs the following steps:

1. If $E_p$ is the root KGC, set $ssk_{par} = (x, c)$, if not, set $ssk_{par} = (x_{par}, c_{par})$.
2. Compute $(I_L \parallel I_R) = HMAC\ (c_{par}, i)$.
3. Compute $x_i = x_{par} + I_L \mod q$.
4. Set child chain code $c_i = I_R$.
5. Return $ssk_i = (x_i, c_i)$.

### 3.5 Set-Child-Public-Key($spk_{par}$, $i$) → $spk_i$

Let the $i$th child be denoted as $E_i$ and $E_i$'s parent be denoted as $E_{par}$. One can compute the corresponding public key of $E_i$ in the following:

1. If $E_{par}$ is the root KGC, set $spk_{par} = (y, c)$, if not, set $spk_{par} = (y_{par}, c_{par})$
2. Compute $(I_L \parallel I_R) = HMAC\ (c_{par}, i)$.
3. Compute

$$y_i = y_{par} \times g^{I_L} = g^{x_{par}} \times g^{I_L} = g^{x_{par}+I_L} \mod p.$$

4. Return $spk_i = (y_i, c_i)$.

### 3.6 Root-KGC-SigGen-For-Child($m$, $THK_0$, $x$, $n$, $e_0$, $i$) → ($\sigma_i$, $THK_i$)

The root KGC $E_0$ can generate a signature $\sigma_i$ on an arbitrary message $m \in \{0,1\}^*$ using private signing key $x_i$ of $E_i$ derived from master secret key $msk$ in the following steps:

1. Let $E_i$'s parent be denoted as $E_{par}$. If $E_{par}$ is the root KGC, set $(x_{par}, n_{par}) = (x, n)$.
2. Compute $(D_L \parallel D_R) = HMAC\ (d_{par}, i)$ and set $d_i = D_R$.

3. Compute $k_i = k_{par} + D_L \mod q$ and then compute $\kappa_i = g^{k_i}$.
4. Generate $i$th child trapdoor hash key $THK_i = (HK_i, TK_i)$ and trapdoor hash function using public hash key $HK_i$ is denoted as $TH_{HK_i}()$.
5. Select $\gamma_i \in \mathbb{Z}_q^*$ and compute $M_i = TH_{HK_i}(\gamma_i, (\kappa_i \parallel m))$.
6. Use $TK_0$ to find the collision $r_i \in \mathbb{Z}_q^*$

$$e_i = TH_{HK_0}(r_i, M_i) = TH_{HK_0}(r, M) = e_0.$$

7. Compute the signature $\sigma_i = (e_i, s_i, r_i, \gamma_i, m)$, such that

$$e_i = TH_{HK_0}(r_i, M_i),$$
$$s_i = k_i - x_i \cdot e_i \; mod \; q.$$

8. If $E_i$ is a regular user, output $(TK_i, \kappa_i, \sigma_i)$.
9. If $E_i$ is a designated lower-KGC, output $(TK_0, TK_i, n_i, \sigma_i)$.

### 3.7 Lower-level-KGC-SigGen-For-Child($m_j$, $THK_0$, $n_t$, $j$) → ($\sigma_j$, $THK_j$)

The lower-level KGC $E_t$ can use its own signature value $\sigma_t = (e_t, s_t, r_t, \gamma_t, m_t)$ with $n_t$ to generate the signature $\sigma_j$ of its child $E_j$ on an arbitrary message $m_j \in \{0, 1\}^*$ in the following steps:

1. Compute $(I_{jL} \parallel I_{jR}) = HMAC\ (c_t, j)$, $(D_{jL} \parallel D_{jR}) = HMAC\ (d_t, j)$.
2. Compute $c_j = c_t + I_{jL} \mod q$, and set $c_j = I_{jR}$.
3. Compute $\kappa_j = \kappa_t \cdot g^{D_{jL}} \mod q$, and set $d_j = D_{jR}$.
4. Generate the $j$th child trapdoor hash key $THK_j = (HK_j, TK_j)$ and trapdoor hash function using public hash key $HK_j$ is denoted as $TH_{HK_j}()$.
5. Select $\gamma_j \in \mathbb{Z}_q^*$ and compute $M_j = TH_{HK_j}(\gamma_j, (\kappa_j \parallel m_j))$.
6. Use $TK_0$ to find the collision $r_j \in \mathbb{Z}_q^*$

$$e_j = TH_{HK_0}(r_j, M_j) = e_t$$

7. Compute the signature $\sigma_j = (e_j, s_j, r_j, \gamma_j, m_j)$

$$\begin{aligned} e_j &= TH_{HK_0}(r_j, M_j), \\ s_j &= s_t + D_{jL} - I_{jL} \cdot e_j \mod q \\ &= k_t - x_t \cdot e_j + D_{jL} - I_{jL} \cdot e_j \mod q \\ &= k_t + D_{jL} - (x_t + I_{jL}) \cdot e_j \mod q \\ &= k_j - x_j \cdot e_j \mod q. \end{aligned}$$

8. If $E_j$ is a regular user, output $(TK_j, \kappa_j, \sigma_j)$.
9. If $E_j$ is a designated lower-KGC, output $(TK_0, TK_j, n_j, \sigma_j)$.

### 3.8 UserSigGen($\tilde{m}$, $\sigma_i$, $\kappa_i$, $THK_i$) → $\tilde{\sigma}$

A user can generate his own signature $\tilde{\sigma}$ on a message $\tilde{m} \in \{0, 1\}^*$ for Bitcoin payment using the given signature value $\sigma_i = (e_i, s_i, r_i, \gamma_i, m_i)$, $\kappa_i$ and $TK_i$ in the following steps:

1. Find the collision $\tilde{\gamma} \in \mathbb{Z}_q^*$ using $TK_i$.
2. Compute the signature $\tilde{\sigma} = (\tilde{e}, \tilde{s}, r_i, \tilde{\gamma}, \tilde{m})$, where

$$\begin{aligned} \tilde{e} &= TH_{HK_0}(r_i, TH_{HK_i}(\tilde{\gamma}, (\kappa_i \parallel \tilde{m}))) \\ &= TH_{HK_0}(r_i, TH_{HK_i}(\gamma_i, (\kappa_i \parallel m_i))) = e_i, \\ \tilde{s} &= s_i. \end{aligned}$$

### 3.9 Verifying($pk$, $\sigma$) = 1/0

To check if the signature $\sigma = (e, s, r, \gamma, m)$ is valid, a verifier using public verification key $pk = y$, and public hash key $(HK_0, HK_i)$ for verification:

1. Let $r_v = g^s y^e$.
2. Let $e_v = TH_{HK_0}(r, TH_{HK_i}(\gamma, (r_v \parallel m)))$.
3. The verifier outputs 1 if $e_v = e$, else 0.

### 3.10 Correctness

The correctness computation is shown below:

$$\begin{aligned} r_v &= g^s y^e \\ &= g^{k - xe} g^{xe} \\ &= g^k. \end{aligned}$$

Thus,

$$\begin{aligned} e_v &= TH_{HK_0}(r, TH_{HK_i}(\gamma, (r_v \parallel m))) \\ &= TH_{HK_0}(r, TH_{HK_i}(\gamma, (g^k \parallel m))) = e. \end{aligned}$$

## 4 The proposed HD wallet system

The construction of our hierarchical deterministic Bitcoin wallet (HD wallet) scheme based on Schnorr Signature consists is presented in this section. Our scheme consists of nine algorithms: **Initialization, Setup, HMAC, Set-Child-Private-Signing-Key, Set-Child-Public-Key, Root-KGC-SigGen-For-Child, Lower-level-KGC-SigGen-For-Child, UserSigGen, Verify**. We consider a scenario of our scheme that a HD wallet owner (i.e., the root KGC) wants to delegate his ability to use his child signing key to generate a signature (i.e., the ability of paying his certain part of money) and a auditor would like to view every detailed transaction of the wallet. Rather than giving out the child signing key to the delegated user, the owner computes the signature of his child and send it along with some secret value to the delegated one

**Table 2** The comparisons between others and our scheme

| | Privilege escalation attack resistant | Without $n$-of-$t$ threshold | Public key derivation | High scalability | Master public key size |
|---|---|---|---|---|---|
| [23] | Yes | Yes | No | No | $O(1)$ |
| [12] | Yes | No | Yes | No | $O(t)$ |
| [10] | Yes | No | Yes | No | $O(1)$ |
| [7] | No | Yes | Yes | Yes | $O(1)$ |
| Ours | Yes | Yes | Yes | Yes | $O(1)$ |

$t$: the amount of shares of the secret key
$n$: the amount of participants sharing the secret key

so that the delegated one can recompute the child signature on a new message. The wallet owner also needs to reveal his master public key and the secret value to the auditor in order that the auditor can compute the corresponding public key of every transaction of the wallet. A delegated user can be a lower-level KGC or a regular user. A lower-level KGC is able to compute the signature of his own or his child on a message; besides, it can delegate his ability of using his child signing key to sign to another lower-level KGC below him or a regular user. A regular user only has the capability of generating the signature of his own on a message. After the scenario of our scheme is presented, the construction of our scheme is also illustrated in Fig. 1. We describe the construction of our scheme in details as follows.

– ***Setup***

(a) The root KGC executes **Initialization**$(1^{\lambda})$ to generate system parameters $param$ and publishes $param$ to all users.
(b) The root KGC executes **Setup**$(param)$ to generate master key pair $(msk, mpk) = (x, y)$, trapdoor hash key $(HK_0, TK_0)$, secret random values $n = (c, d, \kappa)$. The root KGC sends extend public key $spk = (y, c)$ to the auditor while keeps extend private key $ssk = (x, c)$ and $(d, \kappa)$ as secret. Then the root KGC publishes $HK_0$ to all users. In the following, all $HK$ would be public known for everyone.

– ***KeyGen***

(a) With the index $i$, the root KGC extends $i$th child private signing keys $x_i$, child chain code $c_i$ by executing **Set-Child-Private-Signing-Key** $(ssk_{par}, i)$, where $ssk_{par}$ is parent extended private key and keeps $x_i$ as secret. Note that the $i$th child chain code $c_i$ can also be computed by the auditor executing **HMAC**$(c_{par}, i)$.
(b) With the index $i$, users extend $i$th child public signing key $y_i$ by executing **Set-Child-Public-Key**$(spk_{par}, i)$, where $y_{par}$, $spk_{par}$ is parent extended public key. All users can take hash of $y_i$ as Bitcoin address to publish for receiving funds.

– ***Delegation***

(a) The root KGC can run **Root-KGC-SigGen-For-Child**$(m, THK_i, x, n, e_0, i)$ to output a signature $\sigma_i$ of $i$th child $E_i$ on a message $m$ and parameters $d_i, \kappa_i$. Let the lower-level KGC be $E_t$ where the index is $t$ in the hierarchy tree. When the root KGC wants to delegate his ability to generate child signature of $E_t$'s child, $E_t$'s grandchild and so on. He sends $(TK_0, TK_t, n_t, \sigma_t)$ to a lower-level KGC $E_t$. If the root KGC just intend to give out his ability to generate a signature of $i$th child on a message $m$, he sends $(TK_i, \kappa_i, \sigma_i)$ to the regular user $E_i$.
(b) With the given ability to generate the child signature, a lower-level KGC $E_t$ can give his child the ability to sign on a new message. Let $E_j$ be $E_t$'s child with index $j$. For given $(TK_0, TK_t, n_t, \sigma_t)$, the delegated lower-level KGC $E_t$ can generate his child signature $\sigma_j$ on a new message $m_j$ by running **Lower-level-KGC-SigGen-for-child** $(m_j, THK_0, THK_t, n_t, \sigma_t, j)$ to get $(THK_j, \kappa_j, \sigma_j)$ and sends it to $E_j$.
(c) Suppose $E_{t'}$ is another lower-level KGC with index $t'$. If $E_{t'}$ is one of $E_t$'s child where $t < t'$, for given $(TK_0, TK_t, n_t, \sigma_t)$, $E_t$ can delegate the ability to generate the signature of $E_{t'}$'s child on a new message $m_j$ by executing **Lower-level-KGC-SigGen-for-Child**$(TK_0, TK_t, n_t, \sigma_t, t')$ to get $(TK_0, TK_{t'}, n_{t'}, \sigma_{t'})$ and sends it to $E_{t'}$.

– ***Paying Bitcoin***

(a) Let a message $\tilde{m}$ consisting of an address of the receiver and some conditions which have to be fulfilled to include those unspent Bitcoin. While anyone of HD wallet user publishes his signature on $\tilde{m}$, it is regarded as paying Bitcoin to the receiver contained in the signature if it is verified valid.
(b) For the root KGC, he can execute **Root-KGC-SigGen-For-Child** algorithm to generate any signature of his child on $\tilde{m}$.
(c) For a lower-level KGC, he can execute **UserSigGen** algorithm to generate his own signature on message $\tilde{m}$ or he can execute **Lower-level-KGC-SigGen-for-**

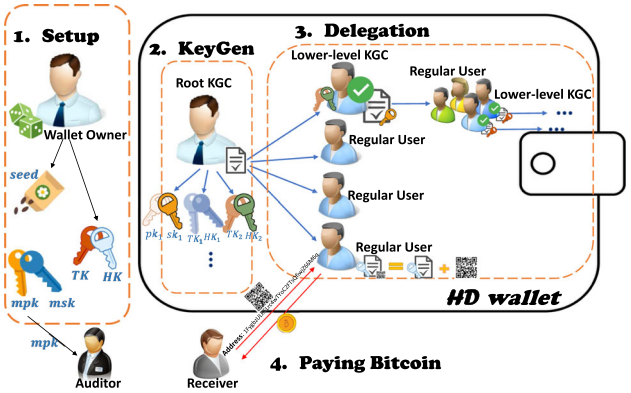


**Fig. 1** The system architecture

**Child** algorithm to generate the child signature on message $\tilde{m}$.

(d) For a regular user who is not permitted to generate any of child signatures, he runs **UserSigGen** algorithm to generate the signature of his own on message $\tilde{m}$.

– ***Publicly verifying***

(a) For a given signature $\sigma$, any of other users in Bitcoin network can execute **Verifying**$(pk, \sigma)$ to verify the signature provided by its owner. The output is 1 if valid, else 0.

## 5 Threat models, security proofs and analyses

In this section, we prove that the proposed scheme is secure against adversaries defined in the following threat model. We focus on an adversary (forger) $\mathcal{F}$ who tries to gain as much information as possible in order that he could forge a signature which not belongs to him. We assume that such adversary is from within the users we defined in Sect. 3 and from outside. Therefore, an adversary in our scheme can be denoted as one of as follows: malicious lower-level KGCs, a malicious regular user colluded with a trustless auditor or other external users. Considering a malicious lower-level KGC may obtain more information than malicious regular users, we turn to model a malicious lower-level KGC to prove the security of our scheme. With respect to the target for the adversaries trying to forge, we also take the following security analysis to model the different types of adversaries. In the threat models, an adversary $\mathcal{F}$, who is a polynomial-time attacker, can interact with a simulator $\mathcal{S}$.

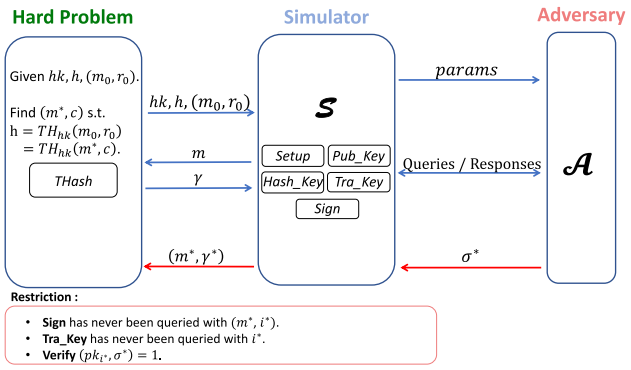


**Fig. 2** The unforgeability game I

### 5.1 Security against $\mathcal{F}_1$ adversary

#### 5.1.1 Threat model

Let the security game involving an polynomial-time adversary (forger) $\mathcal{F}_1$ proceed as follows.

**Definition 7** (*game I*) $\mathcal{F}_1$ interacts with a simulator $\mathcal{S}$ in the following game.

– **Setup**. $\mathcal{S}$ generates the parameters $param$, trapdoor hash key $(HK_0, TK_0)$, secret nonce $n$ and sends $params = (param, HK_0, n)$ to $\mathcal{F}_1$.
– **Query**. In this phase, $\mathcal{F}_1$ has adaptively query to *Public-Key*, *Hash-Key*, *Trapdoor-Key*, and *Sign*.
– **Forgery**. $\mathcal{F}_1$ outputs a forgery $(\sigma^*, m^*)$ and wins the game if the following conditions hold.
  – $\mathcal{F}_1$ is restricted to query **Sign**$(m^*, i^*)$ and **Sign**$(m^*, i')$.
  – $\mathcal{F}_1$ is restricted to query **TraKey**$(i^*)$ and **TraKey**$(i')$.
  – The forgery is valid, i.e., **Verify**$(pk_{i^*}, \sigma^*) = 1$.

In this model, we assume $\mathcal{F}_1$ be a malicious lower-level KGC $LK_1$ trying to forge a descendant user's signature $\sigma^*$ in condition of giving no corresponding trapdoor hash function key which is actually generated by another lower-level KGC $LK'$. Aiming for the forgery, the $LK_1$ can collude among lower-level KGCs and regular users to ask for their signatures and secret nonces except his target user and the target user's ancestor node $LK'$. Besides, $LK_1$ has parameters, public hash keys and public verification keys which are publicly known for all internal users. Our scheme is said to be secure in this model if there exists no adversary $\mathcal{F}_1$ who can forge the target signature with non-negligible probability (Fig. 2).

#### 5.1.2 Security proof

**Theorem 1** *Our scheme is secure against* $\mathcal{F}_1$ *adversary assuming that the trapdoor hash function is collision resistant.*

***Proof*** Assume that there exists a polynomial-time $\mathcal{F}_1$ who can forge a signature with non-negligible probability. Then, we construct a probabilistic polynomial-time algorithm $\mathcal{S}$ that has non-negligible probability to win the collision-resistant-trapdoor-hash-function-game (CRTHFG). At first, $\mathcal{F}_1$ sets the index of target to $i^*$ and sends $i^*$ to $\mathcal{S}$. $(hk, h, m_0, r_0)$ is given to $\mathcal{S}$. $\mathcal{S}$ is allowed to make queries $THash(m) = \gamma$ to the CRTHFG, such that $TH_{hk}(m_0, r_0) = TH_{hk}(m, \gamma)$. In this game, $\mathcal{S}$ simulates the oracles to response all queries for $\mathcal{F}_1$ as the following.

- **Setup**. $\mathcal{S}$ executes the *Initialization* algorithm and *Setup* to generate the parameters $param = (\mathbb{G}, p, q, g)$ and $(msk, mpk, THK_0, n)$. Then, $\mathcal{S}$ sends $params = (param, mpk, HK_0, n)$ to $\mathcal{F}_1$.
- **Query**. In this phase, $\mathcal{F}_1$ makes queries as follows and $\mathcal{S}$ records the result of queries corresponding to the index in a local maintained table $\mathcal{TB}$. If $\mathcal{F}_1$ makes a query with same index asked before, $\mathcal{S}$ looks up its $\mathcal{TB}$ to find the corresponding entry and returns it to $\mathcal{F}_1$.
  - *Public-Key Oracle*. Upon receiving a *Public-Key* query with index $i$, $\mathcal{S}$ executes **Set-Child-Private-Key** to generate a secret key $x_i$ and returns public key $pk_i = g^{x_i}$ to $\mathcal{F}_1$.
  - *Hash-Key Oracle*. Upon receiving a *Hash-Key* query with index $j$, $\mathcal{S}$ checks if $j = i^*$, then $\mathcal{S}$ sets $HK_j = hk$ and returns $HK_j$ to $\mathcal{F}_1$. Else, $\mathcal{S}$ generates trapdoor hash key $THK_j = (HK_j, TK_j)$ and returns $HK_j$ to $\mathcal{F}_1$.
  - *Trapdoor-Key Oracle*. Upon receiving a *Trapdoor-Key* query with index $i$, where $i \neq i^*$, $\mathcal{S}$ returns $TK_i$ to $\mathcal{F}_1$.
  - *Sign Oracle*. $\mathcal{F}_1$ makes a *Sign* query with message $m$, index $i$, where $i \neq i^*$. $\mathcal{S}$ executes **UserSigGen** and returns $\sigma_i$ to $\mathcal{F}_1$.
- **Forgery**. In this phase, $\mathcal{F}_1$ tries to make a forgery. If $\mathcal{F}_1$ can output a valid forgery $\sigma^* = (e^*, s^*, r^*, \gamma^*, \tilde{m})$, such that

$$\begin{aligned} e^* &= TH_{HK_0}(r^*, h) \\ &= TH_{HK_0}(r^*, TH_{hk}(\gamma^*,\ g^{s^*} \cdot pk_{i^*}{}^{e^*} \parallel \tilde{m})). \end{aligned}$$

After receiving the valid $\sigma^* = (e^*, s^*, r^*, \gamma^*, \tilde{m})$ from $\mathcal{F}_1$, $\mathcal{S}$ can find the collision $(\gamma^*, m^*)$, such that

$$\begin{aligned} &\gamma^* = \gamma^*, \\ &m^* = g^{s^*} \cdot pk_{i^*}{}^{e^*} \parallel \tilde{m}. \end{aligned} \tag{1}$$

Thus, we can win the CRTHFG with the non-negligible advantage as follows:

$$\begin{aligned} &Adv_{\mathcal{S}} = \Pr[(m^*, \gamma^*) \leftarrow \mathcal{S}(hk, h, m_0, r_0) : \\ &h = TH_{hk}(m_0, r_0) = TH_{hk}(m^*, \gamma^*)]. \end{aligned}$$

## 5.2 Security analysis

In this section, we present an analysis of the security properties of our scheme in the following.

### 5.2.1 Security against privilege escalation attack

Our scheme is considered secure against privilege escalation attack compared to the BIP32 HD wallet. In the BIP32 HD wallet, users need to delegate their derived child private key to children, thus children can pay the funds corresponding to those keys. However, combination of a derived child key along with the higher level public key always makes the recovery of parent keys. Instead of giving any private key to the child, we give out a signature to the child so that he can prove the ownership of funds. In our scheme, the private key used to sign may only be known to the HD wallet owner, hence it prevents such attack.

### 5.2.2 The unforgeability of another lower-level KGC's signature

Considering a malicious lower-level KGC $LK_1$ trying to forge a signature of another lower-level KGC $LK'$ who is actually absent in the hierarchy tree, $LK_1$ might collude among lower-level KGCs and regular users to ask for their signatures, secret hash keys and secret nonces except his target $LK'$ and $LK'$'s ancestor node. Besides, $LK_1$ generates a pair of trapdoor hash key for the forgery and $LK_1$ has all public parameters which are publicly known for all internal users, including group parameters, public hash keys and public verification keys. However, a Schnorr signature has been proved secure. If $LK_1$ makes a valid forgery, it means that a Schnorr signature can also be forged in condition of no knowledge of the corresponding private key.

### 5.2.3 The unforgeability of a trustless auditor

A wallet owner may partially disclose his public keys and chain code to a auditor who is delegated the ability to derive certain child public keys from given parent public keys and chain codes. Considering a trustless auditor $T$ tries to derive a child public key as his target from a parent public key in which he has no corresponding parent chain code. In condition of given no chain code of target's ancestors, $T$ cannot find the target more efficiently than a $2^{256}$ brute force of HMAC-SHA512.

### 5.2.4 The unlinkability of public keys for external users

A Bitcoin wallet can be said to have anonymity if it would not be possible to link addresses to the identity of their owners. For external users (this means the users not included in the

hierarchy tree of the wallet), it is not feasible to distinguish one derived public key from another in our scheme if there is no information (e.g., chain code) leaked between the parent public keys and the derived child private keys. For instance, given public key $pk$ and index $i, i'$, the probability of deriving child public key $pk'$ is $Pr_1 = Pr[pk' = pk \cdot g^{I_L}]$ or $Pr_2 = Pr[pk' = pk \cdot g^{I'_L}]$, where $I_L = \text{HMAC}(c, i)$ and $I'_L = \text{HMAC}(c, i')$. Since the chain code involved in HMAC function is hidden from external users, the hash value is indistinguishable after taking different index as input, and thus the probabilities of $Pr_1$ and $Pr_2$ should be same in condition of no knowledge of the chain code $c$. Consequently, our scheme provides the unlinkability of public keys for external users.

## 6 Comparisons

This section discusses the properties of HD wallet schemes and compares the properties between the proposed scheme and the other related schemes in Table 2. The properties of HD wallet schemes are discussed in the following.

- Privilege escalation attack resistant: This kind of attack is considered a severe vulnerability as mentioned in Sect. 1. In our scheme, it conceals the corresponding private keys from child nodes to prevent from privilege escalation attacks.
- Without $n$-of-$t$ threshold: This property means that no use of $t$-out-of-$n$ threshold can prevent a collusion between $n$ participants with a threshold of $t$ to recover the master private key. Gutoski et al. [12] adopt $m$-of-$m$ policy to tolerate the leakage of up to $m - 1$ private keys. Goldfeder et al. [10] split the master secret key into $n$ shares and distribute them to child nodes which is permitted to reconstruct the master private key when at least $t$ nodes participate. In our scheme, the master private key or child private keys would never been revealed to any child nodes.
- Public key derivation: The scheme with this property allows an auditor to have the capability to derive public keys without knowing any private key, thus he can view every transaction related to those keys. Wuille et al. adopt a hardened key derivation function to prevent privilege escalation attacks; however, it breaks the relationship between parent public key and child public key. In our scheme, for given $(spk_{par}, i)$, one can compute the $i$th child's public key by executing Sect. 3.5.
- High scalability: The key owner can derive his public/private keys and delegate them to child nodes at any time. One scheme is deemed to be high scalability if it has the properties of *without n-of-t threshold* and *public key derivation* simultaneously.

**Table 3** Computation overhead of our scheme

| Algorithm | Time complexity | Overhead |
|---|---|---|
| Root-KGC-SigGen | $2T_{exp} + 1.1T_m$ | $\approx 1.85$ ms[a] |
| Lower-KGC-SigGen | $1T_{exp} + 1.2T_m$ | $\approx 0.93$ ms |
| UserSigGen | $1T_{exp} + 0.1T_m$ | $\approx 0.92$ ms |

We assume that the trapdoor hash function we adopted is construction 3 [22] and the bit length of modulus is 1024 bits

**Table 4** Computation cost of child key generation in [7,10,12,23]

| | Overhead |
|---|---|
| [7] | $\approx 2T_h + 1T_m \approx 8.46\ \mu$s[a] |
| [10] | $\approx 1T_h + 1T_m \approx 7.23\ \mu$s |
| [12] | $\approx 1T_h \approx 1.23\ \mu$s |
| [23] | $\approx 1T_h \approx 1.23\ \mu$s |

The computation cost for a SHA-512 hash operation is from [14]

- Master public key size: It is the size of the master public key when the depth of the hierarchy tree does not exceeds one. Gutoski et al. would encounter an exponential growth of master public key size if the depth of the hierarchy tree more than one. However, the master public key size is still constant in our scheme if the depth more than one.

We also list the theoretical computation cost for Root-KGC-SigGen, Lower-KGC-SigGen, UserSigGen algorithms. It is not straightforward that how to compare the performance of the proposed scheme with Wuille's HD wallet [23] or other existing works [7,10,12]. The reason is that a signature is given to a child node instead of a derived key. Therefore, we present computation overhead of our scheme in Table 3 and show the cost of child key generation algorithm for [7,10,12,23] in Table 4. The implementation of modular multiplication and exponentiation are performed on [1,18]. We denote the cost of a modular exponentiation and a modular multiplication as $T_{exp}$ and $T_{T_m}$, respectively. Besides, $T_h$ denotes the cost of a SHA-512 hash operation. Though our scheme may not take advantage over other existing works on efficiency, our scheme focuses on solving the problem of achieving security against privilege escalation attacks with high scalability and public key derivation property.

## 7 Conclusion

This work proposes a new HD wallet scheme to prevent from privilege escalation attacks, which is caused by the insecure child key derivation in certain HD wallet schemes, in Bitcoin. In addition, the proposed scheme provides unforgeability by adopting trapdoor hash functions in the design of the scheme.

Moreover, the proposed scheme offers an impressive feature that an auditor is allowed to test the validity of child public keys from the master public keys without knowing the corresponding secrets. Nonetheless, this work shows that the proposed scheme enjoys more features compared with the other related schemes. Eventually, this work demonstrates the security of the proposed scheme by formal security proofs.

**Acknowledgements** This work was supported by Taiwan Information Security Center at National Sun Yat-sen University (TWISC@NSYSU) and the Intelligent Electronic Commerce Research Center from the Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan.

**Funding** This study was funded by the Ministry of Science and Technology of Taiwan (MOST 105-2923-E-110-001-MY3, MOST 107-2218-E-110-014).

## Compliance with ethical standards

**Conflict of interest** Chun-I Fan, Yi-Fan Tseng, Hui-Po Su, Ruei-Hau Hsu and Hiroaki Kikuchi declare that they have no conflict of interest.

**Ethical approval** This article does not contain any studies with human participants or animals performed by any of the authors.

## References

1. Arruda, T.V., Venturini, Y.R., Sakata, T.C.: Performance analysis of parallel modular multiplication algorithms for ECC in mobile devices. Revista de Sistemas de Informação da FSMA **13**, 57–67 (2014)
2. Axon, L: Privacy-awareness in blockchain-based PKI. CDT Technical Paper Series 21/15 (2015)
3. Barcelo, J.: User privacy in the public bitcoin blockchain (2014). http://www.dtic.upf.edu/~jbarcelo/papers/20140704_User_Privacy_in_the_Public_Bitcoin_Blockchain/paper.pdf. Accessed 9 May 2016
4. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Annual International Cryptology Conference, pp. 1–15. Springer (1996)
5. Buterin, V.: Deterministic wallets, their advantages and their understated flaws. Bitcoin Magazine (2013). https://bitcoinmagazine.com/articles/deterministic-wallets-advantages-flaw-1385450276
6. Courtois, N.T., Emirdag, P., Valsorda, F.: Private key recovery combination attacks: On extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor RNG events. Cryptology ePrint Archive, Report 2014/848 (2014). https://eprint.iacr.org/2014/848
7. Courtois, N., Mercer, R.: Stealth address and key management techniques in blockchain systems. In: Proceedings of the 3rd International Conference on Information Systems Security and Privacy, vol. 1, pp. 559–566. ICISSP (2017). ISBN 978-989-758-209-7. https://doi.org/10.5220/0006270005590566
8. Dagher, G.G., Bünz, B., Bonneau, J., Clark, J., Boneh, D.: Provisions: privacy-preserving proofs of solvency for bitcoin exchanges. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 720–731 (2015)
9. Eskandari, S., Clark, J., Barrera, D., Stobert, E.: A first look at the usability of bitcoin key management. arXiv preprint arXiv:1802.04351 (2018)
10. Goldfeder, S., Gennaro, R., Kalodner, H., Bonneau, J., Kroll, J.A., Felten, E.W., Narayanan, A.: Securing bitcoin wallets via a new DSA/ECDSA threshold signature scheme. http://stevengoldfeder.com/papers/threshold_sigs.pdf (2015)
11. Gutoski, G., Stebila, D.: Hierarchical deterministic bitcoin wallets that tolerate key leakage. In: Böhme, R., Okamoto, T. (eds.) Financial Cryptography and Data Security, pp. 497–504. Springer, Berlin (2015)
12. Gutoski, G., Stebila, D.: Hierarchical deterministic bitcoin wallets that tolerate key leakage. In: International Conference on Financial Cryptography and Data Security, pp. 497–504. Springer (2015)
13. Kent, S.: Evaluating certification authority security. In: IEEE Aerospace Conference, pp. 319–327 (1998)
14. Latinov, L.: MD5, SHA-1, SHA-256 and SHA-512 speed performance. https://automationrhapsody.com/md5-sha-1-sha-256-sha-512-speed-performance/ (2018)
15. Levi, A., Caglayan, M.U.: The problem of trusted third party in authentication and digital signature protocols. In: Proceedings of the 12th International Symposium on Computer and Information Sciences (1997)
16. Nakamoto S.: Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf (2008)
17. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: International Conference on the Theory and Applications of Cryptographic Techniques, pp. 387–398. Springer (1996)
18. Rezai, A., Keshavarzi, P.: High-throughput modular multiplication and exponentiation algorithms using multibit-scan–multibit-shift technique. IEEE Trans. Very Larg. Scale (VLSI) Integr. Syst. **23**(9), 1710–1719 (2015)
19. Schmidt, R., Möhring, M., Glück, D., Haerting, R., Keller, B., Reichstein, C.: Benefits from using bitcoin: empirical evidence from a European country. Int. J. Serv. Sci. Manag. Eng. Technol. (IJSSMET) **7**(4), 48–62 (2016)
20. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)
21. Schoenmakers, B.: Security aspects of the ecash$^{TM}$ payment system. In: State of the Art in Applied Cryptography. Lecture Notes in Computer Science, vol. 1528. Springer, Berlin, Heidelberg (1998)
22. Shamir, A., Tauman, Y.: Improved online/offline signature schemes. In: Annual International Cryptology Conference, pp. 355–367. Springer (2001)
23. Wuille, P.: Hierarchical deterministic wallets. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki (2012)