A Key Backup Scheme Based on Bitcoin

Zhongxiang Zheng[1], Chunhuan Zhao[2], Haining Fan[1], Xiaoyun Wang[2,3]

[1] Department of Computer Science and Technology,Tsinghua University, Beijing 100084, China,
[2] Institute for Advanced Study,Tsinghua University, Beijing 100084, China,
[3] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong Universtiy, Jinan 250100, China

Abstract. Since first introduced by Satoshi Nakamoto in 2008, Bitcoin has become the biggest and most well-known decentralized digital currency. Its anonymity allows users all over the world to make transactions with each other and keep their identities hidden. However, protecting private key becomes a very important issue because it is the only access to a unique account and can hardly be recovered if missing. Storing an encrypted backup of private key and its corresponding advanced key is a traditional but effective way, and many other techniques help to make the backup harder to obtain by attackers. While in this paper, we introduce a new backup scheme that can provide protection when an attacker manages to obtain the backup. It is based on Bitcoin system and ECDSA signature scheme. The biggest difference is the generation and recovery of the backup processes are both related with some specific transactions on blockchain, thus it creates a gap for legal users and attackers who manages to obtain backup to recover key. The gap is decided by the number of accounts and transactions on the blockchain which increases rapidly with the growth of Bitcoin's popularity and therefore strengthens the security of our scheme at the same time. What's more, our technique can also be combined with former ones to achieve better security.

Key words: Bitcoin, backup, ECDSA, key recovery, blockchain

1 Introduction

Since first introduced by Satoshi Nakamoto in 2008 [15], Bitcoin has attracted billions of dollar economy and also provides a new method in studying decentralized digital currency. Different from traditional electronic cash schemes, Bitcoin uses a proof-of-work technique instead of a trusted center and realizes a transaction protocol on a distributed network. Based on Bitcoin's architecture, many other proof-of-work schemes or Bitcoin-like distributed consensus schemes have been proposed. However, Bitcoin is still the biggest and the most well-known system among these schemes, the research on it is of representative significance.

Bitcoin is welcomed all over the world because of its anonymity and decentralization, however, these features also bring a new challenge about protecting private key. A private key is the only available access to the corresponding account which is randomly generated by the user. Once the private key is missing because of hardware errors, oblivion or malicious attacks, the corresponding account would be unavailable because recovering private key from the public key is computationally infeasible. For a traditional centralized digital currency, an authority center may help by verifying the user's identity and retrieving the loss, but that is not the case for a decentralized system such as Bitcoin, thus protecting private key becomes one of the important issues in Bitcoin research.

Creating a backup copy of the encrypted private key and either delegating it to a trusted third party or storing it in an offline device is a traditional but useful way. Besides, some other techniques can be used to play a subsidiary role, such as *Mnemonic Code Technique* *Secret Sharing Technique* and *DSS Attack Technique of Bellare et al.*. *Mnemonic Code Technique* is first introduced in BIP-0039 [13], the idea of *Mnemonic Code Technique* is to establish a deterministic bijection between a wallet seed and a group of words that are easy to be remembered. By remembering these understandable words instead of dealing with a serial of binary or hexidecimal representations, users are likely to remember their secrets more easily. And these understandable words become a natural backup of the corresponding private key. While a (t, n)-threshold *Secret Sharing Scheme* is to share a secret among n participants where no less than t participants can reconstruct the secret and any t or more participants can work together to recover the full secret. With *Secret Sharing Technique*, users can split their private key into several pieces and store them in different third parties. If the majority of these third parties are believable, the users' private keys are well-protected and can be recovered when missing. This technique was first introduced independently by Shamir and Blakley, Shamir's scheme [1] was based on Lagranges interpolation and Blakley's scheme [10] was based on hyperplane geometry. Since then, *Secret Sharing Technique* has been extended to Bitcoin such as threshold signature [7]. *DSS Attack Technique of Bellare et al.* is introduced in 1997 [4], which provides an attack method to recover private key through a few signatures when the random integer, denoted as nonce k, is generated by Knuths linear congruential generator. Though it is used as an attack method, it reveals a relationship among signature, nonce and private key.

In this paper, we propose a new Bitcoin key backup strategy which is based on the structure of Elliptic Curve Digital Signature Algorithm(ECDSA) and Bitcoin system. In this backup scheme, a user can generate a backup by his private key, and the backup is restricted with a transaction signed by the user himself on the blockchain. When the private key is missing, he can recover his private key within t trials at the worst case where t denotes the number of

transactions signed with the corresponding private key on the blockchain. On the other hand, however, when an attacker manages to know the backup pair, he should perform T trials at the worst case before he gets the correct private key where T denotes the number of *all* transactions on the blockchain. In this way, our backup strategy can provide a better security than former backup strategies when the backup is obtained by an attacker, furthermore, this backup strategy can also be extended with some techniques, such as *Mnemonic Code Technique* and *DSS Attack Technique of Bellare et al.*, to fulfill higher security requirements.

Former researches mainly concentrate on how to make it harder for attackers to obtain the secret (once obtained the private key is revealed), such as hardware wallet, paper wallet (write down the secret on paper and store it in a safe), brain wallet (memorizing a passphrase in head which can be transformed into the secret) and threshold signatures [7], while the idea behind our scheme is to generate a gap between the time for a legal user and the time for an attacker to recover the private key. The gap which enlarges with the growth of the number of transaction on Bitcoin is the guarantee of our scheme's secutiry. Besides, our scheme can be easily combined with former techniques to make both the obtain of the backup and the recovery of the secret hard for an attacker, thus provides a better security.

Our contribution mainly contains three parts. We first propose a new Bitcoin key backup scheme that can provide better security under the condition that the backup is obtained by an attacker. Secondly, we extend the scheme with several existing techniques and make it more secure and simpler to store. Thirdly, we also make an analysis about these schemes to show their security and effectiveness.

The rest of the paper is organized as follows. In section 2, we introduce the background about Bitcoin and ECDSA. Then we provide the Bitcoin key backup scheme in section 3 followed with its analysis. And we provide some extensions with better security and simpler storage combined with analysis in section 4. Finally, a conclusion is given in section 5.

2 Preliminaries

2.1 Bitcoin

First introduced by Satoshi Nakamoto in 2008 [15], Bitcoin provides a decentralized electronic cash system that solves double-spending problem by using a peer-to-peer network instead of relying on a trusted third party. Transactions in Bitcoin system are recorded as a chain of digital signatures known as blockchain. Each transaction is made up of the fixed structure where an owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin (Bitcoin uses double SHA-256 as the hash function for signature). To generate a block

containing several transactions, a proof-of-work system is used which demands participants to solve a special problem decided by these transactions, and only the longest chain will be recognized as proof of the sequence of events witnessed, thus the system is secure if a majority of CPU power is controlled by honest nodes. The proof-of-work system works better to prevent Sybil attacks basing on the assumption that it is much harder to control the majority of the computation power in the system than to control the majority of identities. Besides, transactions are all public and arbitrary number of key pairs can be generated by users, the gap between identities and arbitrarily generated accounts is the key to protect privacy.

These features make Bitcoin become the biggest and the most well-known decentralized electronic cash system which attracts billions of dollar economy and attentions of many researchers, including decentralization [2], double spending attacks [11, 12] and prevention [3, 8].

2.2 ECDSA

Elliptic Curve Digital Signature Algorithm is the digital signature system used in Bitcoin. It is first introduced in 2001 [9]. An ECDSA contains following parameters: a suitably chosen elliptic curve E defined over a finite field $\mathbb{F}_q$ of characteristic p, an indication field representation of the representation used for the elements of $\mathbb{F}_q$ denoted as FR, a base point $G \in E(\mathbb{F}_q)$, two field elements a and b in $\mathbb{F}_q$ which decide the equation of the elliptic curve E over $\mathbb{F}_q$ (by an equation of the form $y^2 = x^3 + ax + b$), the order n of the point G and the cofactor $h = \#E(\mathbb{F}_q)/n$. And in Bitcoin system, these parameters are fixed as $secp256k1$. It also should be noted that in Bitcoin system, the public key of a receiver of a transaction is not included directly in a transaction message but its transformation form instead (the public key first goes through SHA-256, then RIPEMD-160, appended by a version number and checksum, denoted as address). However, if the receiver starts a new transaction later, he should provide his public key to prove his ownership. This design is used to obfuscate and shorten public key and it does not affect the process of signature, so we will not distinguish between public key and address in this paper to facilitate the narrative.

Signature Generation. To sign a message m given an ECDSA key pairs (d, Q) with domain parameters $D = (q, FR, a, b, G, n, h)$, following steps are needed.

1. Randomly choose an integer, known as nonce k, where $1 \leq k \leq n - 1$.
2. Compute $kG = (x_1, y_1)$, and convert x_1 to an integer $\bar{x}_1$.
3. Compute $r = \bar{x}_1 \bmod n$, if $r = 0$ then go to step 1.
4. Compute $k^{-1} \bmod n$.

5. Compute $Hash(m)$ and convert this bit string to an integer e.

6. Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.

7. The signature for m is (r, s).

Signature Verification. To verify the signature (r, s) on m, one should obtain the domain parameters $D = (q, FR, a, b, G, n, h)$ as well as the associated public key Q, and takes following steps.

1. Verify that r and s are integers in the interval $[1, n - 1]$.

2. Compute $Hash(m)$ and convert this bit string to an integer e.

3. Compute $w = s^{-1} \bmod n$.

4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.

5. Compute $X = (x_1, y_1) = u_1G + u_2Q$.

6. If $X = O$, then reject the signature. Otherwise, convert x_1 into an integer $\bar{x}_1$, and compute $v = \bar{x}_1 \bmod n$.

7. Accept the signature if and only if $v = r$.

Signature Vulnerability. A well-known signature vulnerability exists in ECDSA signature when a user signs different messages with the same private key d and the same nonce k. For example, a user signs m_1 as (r_1, s_1) and m_2 as (r_2, s_2) with key pairs (d, Q), nonce k and $D = (q, FR, a, b, G, n, h)$. An attacker can compute the private key d by:

1. Compute $k = (Hash(m_2) - Hash(m_1))/(s_2 - s_1) \bmod n$, because $ks_1 = Hash(m_1) + rd$ and $ks_2 = Hash(m_2) + rd$ according to the ECDSA signature generation process.

2. Compute $d = (s_1k - Hash(m_1))/r \bmod n$.

This vulnerability exists in non-elliptic curve signature schemes such as ElGamal and DSA, and also applies to ECDSA which is the elliptic curve analogue version of DSA. It has been used to perform attacks in Bitcoin system [6, 16], besides, further studies show that even using different k will also lead to the leakage of private key, for example, Bellare et al. [4] showed that one can recover private key if nonce is produced by Knuths linear congruential generator with known parameters.

Bellare et al.'s work described a method to compute private key through a few signatures whose nonce k is generated by Knuths linear congruential generator with known parameters. For example, give following equations:

$$s_1k_1 = Hash(m_1) + r_1d \bmod n$$

$$s_2k_2 = Hash(m_2) + r_2d \bmod n$$

$$-a_mk_1 + k_2 = b_m \bmod m$$

The problem is to recover the private key d without knowing k_1, k_2. If $m = n$, the problem is easy to solve by linear algebra. And for $m \neq n$, Bellare et al. solved them by the following steps:

1. To search for $x' \in [1, n-1]$, set $x' = n/2, k_1' = k_2' = m/2, \gamma_x = min\{x', n - x'\}, \gamma_{k_1} = min\{k_1', m - k_1'\}, \gamma_{k_2} = min\{k_2', m - k_2'\}$.

2. Generate a matrix $\mathbf{B}$:

$$\mathbf{B} = \begin{pmatrix} -r_1 & s_1 & 0 & n & 0 & 0 \\ -r_2 & 0 & s_2 & 0 & n & 0 \\ 0 & -a_m & 1 & 0 & 0 & m \\ \gamma_x^{-1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \gamma_{k_1}^{-1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \gamma_{k_2}^{-1} & 0 & 0 & 0 \end{pmatrix}$$

3. Apply Babai's nearest lattice vector algorithm on $\mathcal{L}(\mathbf{B})$ with the target vectors of the form $\mathbf{Y} = (Hash(m_1), Hash(m_2), b_m, x'/\gamma_x, k_1'/\gamma_{k_1}, k_2'/\gamma_{k_2})^T$ where the first three elements of $\mathbf{Y}$, $(Hash(m_1), Hash(m_2), b_m)$, are fixed and the last three, $(x'/\gamma_x, k_1'/\gamma_{k_1}, k_2'/\gamma_{k_2})$, will be enumerated by choosing different (x', k_1', k_2') to find a vector on $\mathcal{L}(\mathbf{B})$ which is close to these $\mathbf{Y}$.

4. Obtain the result vector of the form $\mathbf{X} = (Hash(m_1), Hash(m_2), b_m, x/\gamma_x, k_1/\gamma_{k_1}, k_2/\gamma_{k_2})^T$, accept it if the x is the desired solution for d, otherwise, redefine the range of x' as $[1, x-1]$ and $[x, n-1]$, then goto 1.

The expected number of false solutions obtained by the method is proved to be less than mn^{-1}. When m and n have the same size (i.e., $1/2 < m/n < 2$), one can get the right private key in about two rounds with high probability.

3 Bitcoin Key Backup Scheme

In this section, we introduce our key backup scheme which provides a new solution for recovering Bitcoin missing keys which are in ECDSA form, the scheme is based on the signature vulnerability described in Section 2.

3.1 Basic Scheme

Backup Generation. In our basic scheme, Alice with a Bitcoin key pair (PK_A, SK_A) can set up a key backup through following steps:

1. Randomly choose a nonce k where $1 \leq k \leq n-1$.
2. Randomly choose an advanced private key $\bar{SK}_A$.
3. Compute $kG = (x_1, y_1)$, and convert x_1 to an integer $\bar{x}_1$.
4. Compute $r = \bar{x}_1 \bmod n$, if $r = 0$ then go to step 1.
5. Compute $k^{-1} \bmod n$.

6. Compute $Hash(\bar{SK}_A)$ and convert this bit string to an integer e.
7. Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
8. Use k as the nonce when signing a transaction on the blockchain.
9. Record the pair $(\bar{SK}_A, s)$ as the key backup.

Recover Key. By recording the pair $(\bar{SK}_A, s)$, Alice can recover SK_A corresponding to PK_A when the private key is missing with following steps:

1. Collect transactions' signature where the sponsor's public key is PK_A from the blockchain, denoted as $(r_1, s_1), ..., (r_t, s_t)$ and their corresponding message $(m_1, ..., m_t)$.
2. For each $1 \leq i \leq t$ do
3. Compute $k_i = (Hash(m_i) - Hash(\bar{SK}_A))/(s_i - s) \bmod n$.
4. Compute $k_i G = (x_i, y_i)$, and convert x_i to an integer $\bar{x}_i$,
5. If $r_i = \bar{x}_i \bmod n$ then
6. Compute $SK_A = (sk_i - Hash(\bar{SK}_A))/r_i \bmod n$.
6. End if.
7. End for.

3.2 Correctness and Analysis

According to the scheme above, there exists a transaction's signature (r_j, s_j) $(1 \leq j \leq t)$ which uses the same nonce $k_j = k$ and leads to the same the first half of signatures $r_j = r$, so we have:

$$s = k^{-1}(Hash(\bar{SK}_A) + SK_A r) \bmod n$$

$$s_j = k_j^{-1}(Hash(m_j) + SK_A r_j) \bmod n$$

So k can be computed by

$$k = k_j = (Hash(m_j) - Hash(\bar{SK}_A))/(s_j - s) \bmod n$$

And SK_A can be recovered by

$$SK_A = (sk_j - Hash(\bar{SK}_A))/r_j \bmod n$$

While for other signatures (r_i, s_i) $(1 \leq i \leq t$ and $i \neq j)$, k_i is generated randomly which ensures the probability $P(k_i \neq k) = 1 - \frac{1}{n-1} \approx 1 - 2^{-256}$. When $k_i \neq k$, we have:

$$k_i' = (Hash(m_i) - Hash(\bar{SK}_A))/(s_i - s) \bmod n = (s_i k_i - sk)/(s_i - s) \bmod n \neq k$$

The wrong k_i' will not lead to a right $\bar{x}_i$ which satisfies $\bar{x}_i = r_i$ thus will not return a SK_A successfully.

As a result, each round of the recover key process requires 2 $Hash$ processes and 1 elliptic multiply process regardless of basic number operations, to successfully recover a private key, t rounds of the process shall be needed at the worst case and the recovery process needs to run $2t + 1$ $Hash$ processes and t elliptic multiply processes (an extra $Hash$ process is caused by computing the private key and t denotes the number of transactions sponsored by this account). While if an attacker Bob manages to obtain the key pair $(\bar{SK}_A, s)$ and wants to recover the private key, he has:

$$s = k^{-1}(Hash(\bar{SK}_A) + SK_A r) \bmod n$$

For any nonce $k' \in [1, n-1]$, a corresponding r' and SK_A' can be computed, thus Bob can not get any information about Alice's account through the key backup pair. And it is needed for Bob to collect every transaction's signature on the blockchain and run Key Recover process for each signature at the worst case which costs $2T + 1$ $Hash$ processes and T elliptic multiply processes (T denotes the number of transactions on the whole blockchain).

Data obtained from blockchain website(www.blockchain.info) shows that up to July 5th, 2017, the total number of transactions on the blockchain is 2.37×10^8 (in which 1.42×10^8 are performed by the 100 most popular addresses) and the total number of unique addresses used on the blockchain is 2.16×10^8. If we exclude the 100 most popular and inactive addresses (we regard 98.95% of the total addresses as inactive addresses who own less than 0.7% of the whole BTC according to Bitcoin Distribution [5]).

As a result, the average number of transactions made by an account is about $(2.37 \times 10^8 - 1.42 \times 10^8)/(2.16 \times 10^8 \times (1 - 98.95\%) - 100) \approx 41.89$. Then we can draw a conclusion that the time for an attacker to recover key from our backup scheme is $(2.37 \times 10^8 - 1.42 \times 10^8)/41.89 \approx 2.27 \times 10^6 \approx 2^{21}$ times more than a legal user takes to do so.

The biggest difference between the proposed scheme and traditional encrypted backup scheme is that the recover process in our scheme is combined with a unique transaction on the blockchain. If the account to recover is decided, only limited number of trials shall be run and this is often the case for the legal owner because he has a specific target. However, it is hard to recover the key for an attacker who manages to obtain the backup pair because the anonymity of Bitcoin covers the relationship between users and accounts, and the backup pair does not leak information about accounts. With the growing number of users and transactions, the gap between a legal owner and an attacker to recover key grows observably. While for a traditional encrypted backup scheme, it is equally convenient for either legal owner or attacker to recover key from backup pair thus the attacker can get the private key nearly as soon as he obtains the backup pair.

4 Improved Backup Scheme

In this section, we introduce several improvements of the above Bitcoin backup scheme which can provide better security and simpler storage.

4.1 Remove Time Relationship

In the backup generation process of the basic scheme, the generation time of the backup pair $(\bar{SK}_A, s)$ is always earlier than the corresponding transaction on the blockchain, this feature may be used by attacker to recover key by collecting all transactions with timestamp later than the backup pair's generation time instead of all transactions. To overcome this situation, an alternative Backup Generation process is provided.

Backup Generation Alt.

1. Record a nonce k which was used in signing a transaction on the blockchain.
2. Choose an advanced private key $\bar{SK}_A$.
3. Compute $kG = (x_1, y_1)$, and convert x_1 to an integer $\bar{x}_1$.
4. Compute $r = \bar{x}_1 \bmod n$, if $r = 0$ then go to step 1.
5. Compute $k^{-1} \bmod n$.
6. Compute $Hash(\bar{SK}_A)$ and convert this bit string to an integer e.
7. Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
8. Delete the record of integer k.
9. Record the pair $(\bar{SK}_A, s)$ as the key backup.

The difference between this generation process and the former one in the basic scheme is that the nonce k in this process is chosen as the one once used in signing a transaction on the blockchain, as a result, the backup is connected with a transaction signature which is signed earlier than the backup pair generating. Randomly choosing between these two Backup Generation processes to set up key backup can remove time relationship between the backup generation and the corresponding transaction, and this technique can make the scheme more secure.

4.2 Simplify Storage with Existing Techniques

As described above, to successfully recover a private key from its backup, one needs to obtain the pair $(\bar{SK}_A, s)$ as well as its corresponding transaction at the same time. Transactions can be gained from the blockchain, while the pair $(\bar{SK}_A, s)$ needs to be stored. Offline storage is the most secure way, while storing with the help of third parties is less secure but more convenient.

In our basic scheme, an advanced private key $\bar{SK}_A$ is used to generate backup and to recover the private key. The choice of this advanced private key is with

less limit because it is used with its hash form. However, a weak advanced private key may increase the risk of being attacked while a complex one, i.e. randomly generated, is hard to remember. Mnemonic Code Technique can be useful to help generate a pseudorandom advanced private key as well as a mnemonic sentence. The mnemonic sentence plays the same role as the advanced private key and is much easier to remember.

If users have further requirement about simplifying storage, half part of the backup, says s, can also be deposited to a trusted third party. However, it is necessary for the user to deposit s with a pseudonym but not with an identification that is related to his account in order to protect his privacy. The third party has a duty to provide deposited messages to its owner when needed, and this will leads to a successful key recover. If accident happens, such as an attack or a theft, which causes the leakage of s. The security of private key is still guaranteed, because for an attacker who obtains s and tries to recover the secret without $\bar{SK}_A$ and account's information has to collect every transaction and enumerate for $Hash(\bar{SK}_A)$ at the same time:

1. Collect all transactions' signatures from the blockchain, denoted as $(r_1, s_1), ..., (r_T, s_T)$ and their corresponding message $(m_1, ..., m_T)$.
2. For each $1 \leq i \leq T$ do
3. For each $j \in$ output region of $Hash$ function do
4. Compute $k_i = (Hash(m_i) - j)/(s_i - s) \bmod n$.
5. Compute $k_i G = (x_i, y_i)$, and convert x_i to an integer $\bar{x}_i$,
6. If $r_i = \bar{x}_i \bmod n$ then
7. Compute $SK_A = (sk_i - j)/r_i \bmod n$.
8. End if.
9. End for.
10. End for.

As a result, the process of recovering key will cost $2^{256}(T+1)$ $Hash$ processes and $2^{256}T$ elliptic multiply processes at the worst case (T denotes the number of transactions on the blockchain and the output of $Hash$ is considered to be uniformly distributed among 2^{256}). According to the former computation, the time for an attacker to recover key from the backup scheme is $2.27 \times 10^6 \times 2^{256} = 2.62 \times 10^{83} \approx 2^{276}$ times more than a legal user takes to do so.

4.3 Generate Backup with Different Nonce

We now made use of the signature vulnerability of ECDSA when signing two messages using the same nonce and establish our backup scheme, is it possible to use different nonce to generate a backup with better security? This doubt leads to a further improvement.

4.3.1 Generate Backup with Two Different Nonce

Bellare et al. [4] showed that one can recover private key if k is produced by Knuths linear congruential generator with known parameters. With the help of this technique, we can further improve our backup scheme by using two different nonce in backup generation process.

Backup Generation. In our improved scheme, Alice with a Bitcoin key pair (PK_A, SK_A) can set up a key backup through the following steps:

1. Select a 256-bit prime m such that $m > n$, set $b_m = 1$, randomly choose two nonce k_1, k_2 in $[1, n-1]$ (for $m > n$, we have $k_1 (\text{mod } n) = k_1 (\text{mod } m)$,$k_2 (\text{mod } n) = k_2 (\text{mod } m)$) .

2. Compute $a_m = k_1^{-1}(k_2 - b_m) \bmod m$.

3. Respectively use k_1 and k_2 as the nonce when signing two different transactions on the blockchain.

4. Record a_m as the key backup.

Recover Key. By recording a_m, Alice can recover SK_A corresponding to PK_A when the private key is missing with following steps:

1. Collect transactions' signature where the sponsor's public key is PK_A from the blockchain, denoted as $(r_1, s_1), ..., (r_t, s_t)$ and their corresponding message $(m_1, ..., m_t)$.

2. For each $1 \leq i, j \leq t$ where $i \neq j$ do

3. Set $x' = n/2, k_1' = k_2' = m/2, \gamma_x = min\{x', n-x'\}, \gamma_{k_1} = min\{k_1', m-k_1'\}, \gamma_{k_2} = min\{k_2', m - k_2'\}$.

4. Generate a matrix $\mathbf{B}$:

$$\mathbf{B} = \begin{pmatrix} -r_i & s_i & 0 & n & 0 & 0 \\ -r_j & 0 & s_j & 0 & n & 0 \\ 0 & -a_m & 1 & 0 & 0 & m \\ \gamma_x^{-1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \gamma_{k_1}^{-1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \gamma_{k_2}^{-1} & 0 & 0 & 0 \end{pmatrix}$$

5. Apply Babai's nearest lattice vector algorithm on $\mathcal{L}(\mathbf{B})$ with the target vectors of the form $\mathbf{Y} = (Hash(m_i), Hash(m_j), 1, x'/\gamma_x, k_1'/\gamma_{k_1}, k_2'/\gamma_{k_2})^T$ and check if a result vector has the form $\mathbf{X} = (Hash(m_i), Hash(m_j), 1, x/\gamma_x, k_1/\gamma_{k_1}, k_2/\gamma_{k_2})^T$, if not, goto 2.

6. Compute $Q' = xG$, accept it if Q' is the account's public key. Otherwise, if x' is searched through $[1, n-1]$, then divide x' searching spaces into $[1, x-1]$,$[x, n-1]$ and goto 3.

7. End for.

The scheme neither changes the signature on transactions nor uses linear congruential generator to generate nonce. The idea is to compute the relationship between two transactions and transform the problem into the one analysed by Bellare et al. To recover private key, one should compute about $t(t-1)$ $Hash$ processes, $t(t-1)$ elliptic multiply processes and $t(t-1)$ nearest lattice vector processes (to decide which two of the transactions are related, and the complexity is about $t(t-1)log(n)log^2(m)$). But for an attacker who manages to obtain a_m, should compute about $\sum_{account\ i} t_i(t_i-1)$ $Hash$ processes, $\sum_{account\ i} t_i(t_i-1)$ elliptic multiply processes and $\sum_{account\ i} t_i(t_i-1)$ nearest lattice vector processes before he gets the right one (t_i denotes the number of transactions on the blockchain for each account i). According to the former computation, the time for an attacker to recover key from the backup scheme is about $(2.27\times10^6)^2 = 5.15\times10^{12} \approx 2^{42}$ times more than a legal user takes to do so.

Compared with the basic scheme, the improved scheme needs more computation to recover key from backup but it increases the gap for attacker to calculate at the same time. Besides, the storage is simplified for the latter one which only needs to store 256 bits.

4.3.2 Generate Backup with Arbitrary Number of Different Nonce

Moreover, we can extend the updated scheme by increasing the number of transactions which are used to generate backup from two to arbitrary integer c to further improve its security. To achieve this improvement, a few changes shall be made:

Backup Generation. In our improved scheme, Alice with a Bitcoin key pair (PK_A, SK_A) can set up a key backup through the following steps:

1. Select a 256-bit prime m such that $m > n$, randomly choose c nonce $k_1, ..., k_c$ in $[1, n-1]$ (for $m > n$, we have $k_i(\text{mod } n) = k_i(\text{mod } m)$ for $1 \le i \le c$) .

2. Compute $a_m = k_1^{-1}(k_c - \sum_{j=1,...,c-1} k_j - 1) \bmod m$, thus we have $k_c = a_m k_1 + \sum_{j=1,...,c-1} k_j + 1 \bmod m$.

3. Respectively use $k_1, ..., k_c$ as the nonce when signing c different transactions on the blockchain.

4. Record a_m as the key backup.

Recover Key. By recording a_m, Alice can recover SK_A corresponding to PK_A when the private key is missing with following steps:

1. Collect transactions' signature where the sponsor's public key is PK_A from the blockchain, denoted as $(r_1, s_1), ..., (r_t, s_t)$ and their corresponding message $(m_1, ..., m_t)$.

2. For each $1 \leq i_1, ..., i_c \leq t$ where they are all different do

3. Set $x' = n/2, k'_1 = ... = k'_c = m/2, \gamma_x = min\{x', n - x'\}, \gamma_{k_1} = min\{k'_1, m - k'_1\}, ..., \gamma_{k_c} = min\{k'_c, m - k'_c\}$.

4. Generate a matrix $\mathbf{B}$:

$$\mathbf{B} = \begin{pmatrix} -r_{i_1} & s_{i_1} & 0 & \dots & 0 & n & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \dots & 0 & \vdots & \vdots & \vdots & \vdots \\ -r_{i_c} & 0 & 0 & \dots & s_{i_c} & 0 & \dots & n & 0 \\ 0 & -a_m & -1 & \dots & 1 & 0 & 0 & 0 & m \\ \gamma_x^{-1} & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ 0 & \gamma_{k_1}^{-1} & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \dots & 0 & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \gamma_{k_c}^{-1} & 0 & 0 & 0 & 0 \end{pmatrix}$$

5. Apply Babai's nearest lattice vector algorithm on $\mathcal{L}(\mathbf{B})$ with the target vectors of the form
$\mathbf{Y} = (Hash(m_{i_1}), ..., Hash(m_{i_c}), 1, x'/\gamma_x, k'_1/\gamma_{k_1}, ..., k'_c/\gamma_{k_c})^T$ and check if a result vector has the form $\mathbf{X} = (Hash(m_{i_1}), ..., Hash(m_{i_c}), 1, x/\gamma_x, k_1/\gamma_{k_1}, ..., k_c/\gamma_{k_c})^T$, if not, goto 2.

6. Compute $Q' = xG$, accept it if Q' is the account's public key. Otherwise, if x' is searched through $[1, n-1]$, then divide x' searching spaces into $[1, x-1]$,$[x, n-1]$ and goto 3.

7. End for.

According to the probability analysis of [4], the expected number of false solutions obtained by the method is proved to be less than $n^{c-1}n^{1-c} \approx 1$, which indicates that we can obtain the desire private key with about 2 rounds of computation with high probability. As described before, a legal user has to determine which c out of t transactions shall be used to recover the backup. While an attacker who obtains the backup but does not know which account is the target must search c transactions through all blockchain. In this way, the gap for them to recover key is about $(2.27 \times 10^6)^c \approx 2^{21c}$ which indicates that a proper chosen c will make the computation time for a legal user acceptable while for an attacker computationally infeasible.

5 Conclusion

In this paper, we introduced a key backup scheme for Bitcoin based on ECDSA signature. The biggest difference between our scheme and the former ones is that

the generation and recovery of backups are bounded with several unique transactions on the blockchain. The storage of the backup leaks no information about the accounts. Besides, due to the privacy protection mechanism of Bitcoin, even if an attacker somehow manages to obtain the backup files, it takes much more time for him to recover the key than for a legal user who has a specific aim. The gap between users and attackers to recover key is decided by the number of accounts and transactions on the blockchain, which means the gap naturally enlarges with the growth of Bitcoin popularity. Besides, we also provide three improvements to further enhance its security and simplify its storage which indicates that the combination of our scheme and former techniques can make it harder for attackers to obtain backups and to recover private keys at the same time.

References

1. Shamir A. How to Share a Secret. Communications of the ACM, Vol. 22, Issue 11, 1979. 612-613
2. Gervais A., Karame G., Capkun S., *et al.* "Is bitcoin a decentralized currency?". in Proc. 35th IEEE Symp. Secur. Privacy (SP14), 2014. 54-60
3. Bamert T., Decker C., Elsen L., *et al.* Have a snack, pay with Bitcoins[C]//Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on. IEEE, 2013. 1-5
4. Bellare M., Goldwasser S., Micciancio D. Pseudo-random number generation within cryptographic algorithms: The DDS case[J]. Advances in CryptologyCRYPTO'97, 1997. 277-291
5. SANTOS M. "Who Owns All the Bitcoins?", 2013.
   Available: https://99bitcoins.com/who-owns-all-the-bitcoins-infographic/
6. Bos J. W., Halderman J. A., Heninger N., *et al.* Elliptic curve cryptography in practice. in Proc.18th Int. Conf. Financial Cryptogr. Data Secur. (FC14), 2014. 157-175
7. Goldfeder S., Gennaro R., Kalodner H., *et al.* Securing bitcoin wallets via a new DSA/ECDSA threshold signature scheme. Tech. Rep., 2015.
8. Finney H. "Best Practice for Fast Transaction AcceptanceHow High is the Risk?", 2011.
9. Johnson D., Menezes A., Vanstone S. The elliptic curve digital signature algorithm (ECDSA)[J]. International Journal of Information Security, 1(1), 2001. 36-63
10. Blakley G. R. Safeguarding cryptographic keys, in Proceedings of the International Workshop on Managing Requirements Knowledge, New York, 1979. 313-317
11. Karame G. O., Androulaki E., Capkun S., Double-spending fast payments in bitcoin, in Proc. 19th ACM Conf. Comput. Commun. Secur.(CCS12), 2012. 906-917
12. Karame G. O., Androulaki E., Roeschlin M., *et al.* Misbehavior in bitcoin: A study of double-spending and accountability, ACM Trans. Inf. Syst. Secur., vol.18, no. 1, 2015. 2:1-2:32

13. Mnemonic code for generating deterministic keys.
Available: https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki
14. Möser M., Böhme R., Breuker D., Towards risk scoring of bitcoin transactions, in Proc. 1st Workshop Bitcoin Res. (BITCOIN14), 2014.
15. Nakamoto S., Bitcoin: A peer-to-peer electronic cash system[J]. 2008.
16. Schneider N., Recovering Bitcoin private keys using weak signatures from the blockchain[J]. 2013.