Deterministic Sub-Wallet for Cryptocurrencies

Hossein Rezaeighaleh
Department of Computer Science
University of Central Florida
Orlando, USA
rezaei@knights.ucf.edu

Cliff C. Zou
Department of Computer Science
University of Central Florida
Orlando, USA
czou@cs.ucf.edu

***Abstract*— A big challenge in cryptocurrency is securing a user key from potential hackers because nobody can rollback a transaction made by an attacker with a stolen key once the blockchain network confirms it. One solution to protect users is splitting the money between super-wallet and sub-wallet. The user stores a large amount of money on her super-wallet and keeps it safe; she refills the sub-wallet when she needs while using the sub-wallet for her daily purchases. In this paper, we propose a new scheme to create sub-wallet that we call deterministic sub-wallet. In this scheme, the seed of the sub-wallet keys is derived from the super-wallet master seed, and therefore the super-wallet can build many sub-wallet addresses and refill them in a single blockchain transaction. Compared to existing approaches, our mechanism is cheaper, real-time, more secure against man-in-the-middle attack and easier for backup and recovery. We implement a proof-of-concept on a hardware wallet and evaluate its performance. In addition, we analyze the attacks and defenses of this design to demonstrate that our proposed method has a higher level of security than existing models.**

Keywords—blockchain, cryptocurrency, hardware wallet, smart card, Bitcoin.

I. Introduction

Blockchain technology and cryptocurrencies become increasingly accessible and usable in various areas from purchasing a coffee to transferring vehicles ownership. At the same time, the crypto coins become more attractive and valuable for hackers to steal, as we read the frequent news of hackers stealing a large amount of money from blockchain users. A major security issue in all cryptocurrencies, including Bitcoin and many altcoins, is the safety of users' private key. Cryptocurrencies usually use elliptic-curve asymmetric cryptography to control the ownership of coins or accounts. In other words, to transfer fund from a user to another, the sender signs a transaction with her private key, and the blockchain verifies the signature of the transaction with the sender's public key. If the blockchain network accepts and confirms this transaction, nobody can roll it back, unlike the traditional bank transfer. Thus, if a hacker empties the user account and transfers all her money to his account, she has no way to reverse the transaction and recover her loss. Unfortunately, many people have experienced this disaster.

A user's private key has full control of the user's fund, and she should stand on her own feet and keep her private keys safe by herself, which is one of the most critical challenges in cryptocurrencies [1], [2]. Users usually employ crypto wallets to generate and store their private keys and sign transactions. Crypto wallets have many forms from online wallets to mobile and cold wallets, and the most secure one is hardware wallet which usually is in the form of a USB stick, Bluetooth device or smartcard.

Even though the hardware wallet is a secure option, it is risky that a user puts all of her fund on a device and uses that for day-to-day purchase. A smart and simple solution is proposed in [1] called super-wallet/sub-wallet model. The super-wallet is like a saving account that stores a large amount of money and only refills the same owner's sub-wallet infrequently when needed. The sub-wallet is like a spending account that stores a small amount of fund used by the user for daily expenses. Therefore, if the user's sub-wallet is lost or hacked, she does not lose a significant amount of money.

In the classic model [1], every time a user wants to refill her sub-wallet, she sends fund from the super-wallet address to the sub-wallet address. This process is straightforward but has significant drawbacks. First, each time the user refills the sub-wallet, the super-wallet creates a transaction and publishes to the blockchain network. Thus, she pays a miner fee for each such transaction. Also, she should wait for confirmation, so refilling the sub-wallet takes time. Also, refilling the sub-wallet is risky because a hacker could perform Man-In-The-Middle (MITM) attack to replace the user's sub-wallet address by his address to receive fund from the super-wallet. Furthermore, the user must maintain the backup of both super-wallet and sub-wallet.

To resolve these challenges in the super-wallet/sub-wallet model, we propose a new scheme that we call deterministic sub-wallet. In this model, the sub-wallet seed is derived from the super-wallet seed, and this process being executed inside the super-wallet. The super-wallet derives the sub-wallet addresses and transfer fund to them in only one blockchain transaction. To refill, the user transports a seed from the super-wallet to the sub-wallet instead of creating a blockchain transaction. Consequently, this model can refill multiple sub-wallet addresses with only one mining fee and one-time waiting for confirmation. It is secure because the super-wallet does not need to get the sub-wallet addresses from the outside of the wallet and it prevents a MITM attack. Also, there is no need to back up the sub-wallet, because it can be derived from the super-wallet. For proof-of-concept, we implement a prototype of our proposed deterministic sub-wallet in a hardware wallet and evaluate its performance. In summary, our contributions in this paper are:

- Designing a new super-wallet/sub-wallet model which reduces refilling cost and time, enhances the security, and removes the necessity for the sub-wallet backup
- Implementing a proof-of-concept in a hardware wallet

978-1-7281-4693-5/19/$31.00 ©2019 IEEE
DOI 10.1109/Blockchain.2019.00064

In section II, we overview related works including Hierarchical Deterministic wallet and classic super-wallet/sub-wallet model. In section III we explain our new proposed deterministic sub-wallet model and Section IV is about our prototype implementation in a hardware wallet, and we evaluate its performance in section V. Next, we define our security assumptions and threat model and do a security analysis of the algorithm and its implementation in section VI. Finally, in section VII, we finish the paper with a conclusion.

II. Related Works

A. Hierarchical Deterministic Wallet

Bitcoin, Ethereum, Litecoin, and almost all popular cryptocurrencies use elliptic-curve cryptography (ECC) to sign and verify transactions. They usually use secp256k1 domain parameters with ECC 256-bit [4]. Therefore, the user has a key pair and uses the private key to sign transactions and transfer fund to another user's public key. The sender must know the receiver's public key to perform a transaction, and all users publish their public key in a specific format called address. Therefore, a user keeps her private key secret and publishes her address to other users in the network that causes privacy concerns because everyone that has access to the Internet can discover the user's addresses and track her transactions.

Thus, anonymity is a challenge in most cryptocurrencies because all transaction history is on the blockchain network. To tackle this problem, the user should use a new address in each transaction to receive fund from others or return the remining value of spending transaction called 'change address'. It means that she generates a new key pair for each transaction. Thus, nobody can track her just by watching her transaction history, and this is a best-practice in Bitcoin and many cryptocurrencies [5]. However, generating a random private key for each transaction requires maintaining a lot of private keys which is hard to manage. Deterministic wallets are invented to solve this problem and use a predictable algorithm to generate new private keys, and because it can be hierarchical, they are called Hierarchical Deterministic (HD) wallets [6]. In HD wallet, the user has a tree of private keys which any node can be derived from its parent using Child Key Derivation (CKD) algorithm. The root of this tree is a private key which is called 'master private key' and derived from an random value called 'master seed'. In other words, anyone who has the master seed can derive all subordinate private keys and addresses. Consequently, the user only needs to keep one seed value safe and generates a lot of pseudo-random addresses which provide anonymity.

HD wallet uses a path to address each key in the key tree that is a sequence of a letter and a few numbers. The first element in the path is letter 'm' that denotes master seed and subsequent numbers are the input indexes for CKD algorithm in the corresponding round [6]. In addition to HD wallet base algorithms, the cryptocurrency community proposed a complementary standard to define a universal path format for all coins (Bitcoin, Ethereum, Litecoin, and other coins) [7]. The format of this addressing is as follows:

$$\text{path} = \text{m/purpose}'\text{/coin}'\text{/account}'\text{/change/address\_index} \quad (1)$$

There is also another proposal [8] which defines a conversion algorithm to convert a list of memorable words (mnemonics) to a seed for HD wallets. The user must write down these words (12 to 24 words) on a piece of paper and keep that safe. She can recover whole her key tree on a new wallet using these words. Crypto wallets usually use this conversion to back up the master seed.

Finally, there is a large universal tree derived from a word list that covers all keys of all coins for a user wallet and each key in the tree has a unique path. However, these mechanisms are silent about the super-wallet/sub-wallet model, and there is not any link between two wallet keys. In our proposed scheme, we use the existing HD wallet structure and add a link between the master seed of the super-wallet and the master seed of the sub-wallet that we called sub-seed.

B. Classic Super-Wallet/Sub-Wallet Model

The idea of super-wallet and sub-wallet is proposed in [1]. It is separating the main account that conveys a large amount of money from spending account that is used for the daily transactions. It mimics personal saving account and spending account in traditional banking. A user uses her spending account on a sub-wallet for day-to-day expenses such as a purchase from online stores, pay bills or buy a coffee. On the other hand, she uses her saving account on a super-wallet just for receiving like a deposit of salary and refill her spending account on the sub-wallet. Therefore, she uses her super-wallet rarely, e.g., one or two times per month, and her sub-wallet several times per day.

The classic solution to build super-wallet and sub-wallet proposed in [1] is straightforward. The user should have two regular wallets. She designates one wallet as super-wallet and stores all of her fund on that. Then, each time she wants to refill the sub-wallet (second wallet), she retrieves a receiving address from the sub-wallet and sends fund from the super-wallet to this address. In this mechanism, the user creates a transaction in the super-wallet each time she wants to refill the sub-wallet. This process requires paying miner fee and waiting a period for confirmations. Because usually, the terminal (e.g., laptop or smartphone) is vulnerable to malware attacks, it is possible that a hacker replaces the sub-wallet address by his own address to steal funds from the super-wallet. Furthermore, the user should back up both super-wallet and sub-wallet similar to all regular wallets. In the next section, we address these issues with our proposed model.

III. Proposed Deterministic Sub-Wallet

In contrast to classic super-wallet/sub-wallet model with unlinked key trees, in our new scheme, deterministic sub-wallet, we derive the sub-wallet seeds from the super-wallet master seed. Therefore, the super-wallet can build all sub-wallet key trees. So, the super-wallet refills several sub-wallet addresses with one blockchain transaction, and refills the sub-wallet with transporting one sub-seed.

Compared to the classic super-wallet/sub-wallet model, the advantages of our proposed deterministic sub-wallet are:

- Deterministic sub-wallet is cheaper in terms of the miner fee because it can refill multiple sub-wallet addresses with one blockchain transaction, while classic model requires a blockchain transaction in each refill.

- Refilling sub-wallet is real-time in the deterministic sub-wallet because it is an offline sub-seed transporting from the super-wallet to the sub-wallet without any transaction with blockchain network.
- The classic model is vulnerable to Man-In-The-Middle attack for key injection similar to other regular wallets, but deterministic sub-wallet is not because the sub-wallet addresses are generated inside the super-wallet.
- The user must back up both the super-wallet and the sub-wallet seeds in the classic model, but in the deterministic sub-wallet, there is no need to back up the sub-wallet seed because it is derivable from the super-wallet seed. So, it is enough to back up the super-wallet seed.

The abstract process of deterministic sub-wallet refilling is as follows. The super-wallet generates a pool of sub-wallet addresses and constructs a large transaction which transfer funds from one (or more) super-wallet addresses to the generated sub-wallet addresses. Then, the super-wallet signs and publishes the transaction. After that, each time the user wants to refill the sub-wallet, she exports a sub-wallet seed from the super-wallet and imports that to the sub-wallet securely. In our previous paper [9], we proposed a secure cryptographic mechanism to transport a seed between wallets using Elliptic-Curve Diffie-Hellman. We explain the details of the process in the following sections.

A. *Sub-Wallet Seed Derivation*

Both super-wallet and sub-wallet should be HD wallet to support the anonymity and privacy of the user. In our model, one sub-wallet can have only one seed at a time, but the super-wallet derives a new seed each time to generate a new sub-wallet address. So, to implement a deterministic sub-wallet, we propose a simple function to derive multiple sub-wallet seeds (subSeed) from a super-wallet master seed (masterSeed). This function is as follows.

$$\text{subSeed} = \text{HMAC-SHA512}(\text{key}=\text{"Sub-wallet xxxx"}, \text{data}=\text{masterSeed}) \qquad (2)$$

In this function, we use a procedure similar to the master key generation function in [6] with some modifications. The core function is an HMAC-SHA512 with a master seed as input data and "Sub-wallet xxxx" string as input key. The “xxxx” is the index of sub-wallet starting from 0 which is a four-digit hexadecimal number. For example, the input key for sub-wallet number 1 will be "Sub-wallet 0001". The output of this function is a 512-bit deterministic pseudo-random value which can be used as a regular seed to construct an HD wallet key tree on the sub-wallet.

B. *Sub-Wallet Refilling*

Refilling many addresses of the sub-wallet in one transaction requires a multi-output transaction. This type of transaction can have more than one output to send coins to multiple addresses. Cryptocurrencies like Bitcoin and other altcoins that uses UTXO (Unspent Transaction Output) model support the multi-output transaction, while some account-based cryptocurrencies like Ethereum does not. This paper focuses on first group of cryptocurrencies, but this design is applicable on Ethereum with an additional Smart Contract like [10].

To refill the sub-wallet, the super-wallet creates and signs a multi-output transaction. The refilling function gets inputs *n*, *i* and *v* that described in TABLE I. This algorithm runs on the super-wallet and generates *n* sub-seeds starting from index *i* using sub-wallet seed generation function. Next, it derives the sub-wallet private keys and their addresses with a predefined fixed path illustrated in Fig. 1. This path is fixed for all sub-seeds and we use only the first address of each sub-seed. In this path, ‘change’ is 1 because the result address is used to transfer funds from the super-wallet to the sub-wallet as an internal use.

The super-wallet generates *n* addresses from *n* sub-seeds and creates a transaction that transfers *v/n* coin to each address. It divides the input fund for all addresses equally. Fig. 1 shows the pseudo-code of the sub-wallet refilling algorithm and TABLE I. describes the acronyms of the pseudo-code.

```
refillSubWallet (n, i, v){
  for j=i to i+n {
    $s_j$ = deriveSubSeed(masterSeed, j)
    $k_j$ = deriveKey(seed=$s_j$,
              path="m/44'/coin'/0'/1/0")
    $a_j$ = privateKeyToAddress($k_j$)
  }
  tx = signTX(v/n => $a_j$ : j=i to i+n)
  sendTransaction(tx)
}
```

Fig. 1. Sub-wallet refilling pseudo-code

TABLE I. SUB-WALLET REFILLING PSEUDO-CODE ACRONYMS

Acronym	Meaning
n	number of sub-wallet addresses
i	index of the first sub-wallet address
v	sum of funds to refill
s_j	Sub-seed of sub-wallet index j
k_j	Private key of sub-wallet index j
a_j	Address of sub-wallet index j
tx	Blockchain transaction

To clarify this algorithm, we discuss a simplified example of the sub-wallet refilling procedure illustrated in Fig. 2. Assume that the super-wallet address (Super-wallet$_{\text{address1}}$) has 30 Bitcoin at first. The sub-wallet refilling algorithm creates a transaction with 5 sub-wallet addresses (*n*=5) starting from sub-wallet index 1 (*i*=1), and the total fund is 2 Bitcoin (*v*=2). After confirmation by blockchain, the super-wallet address has 28 Bitcoin and each sub-wallet address (Sub-wallet$_{\text{address1}}$ to Sub-wallet$_{\text{address5}}$) has 0.4 Bitcoin.

In the real world and also our prototype implementation some details are different. For example, to provide anonymity, a change address is used that means the address of the super-wallet to receive remaining fund in the left side is different from the input super-wallet address in the right side. Furthermore, the sum of the fund before and after publishing the refilling transaction are not equal because of the mining fee. Also, the input super-wallet address could be replaced by multiple super-wallet addresses to provide enough fund to refill the sub-wallet addresses.

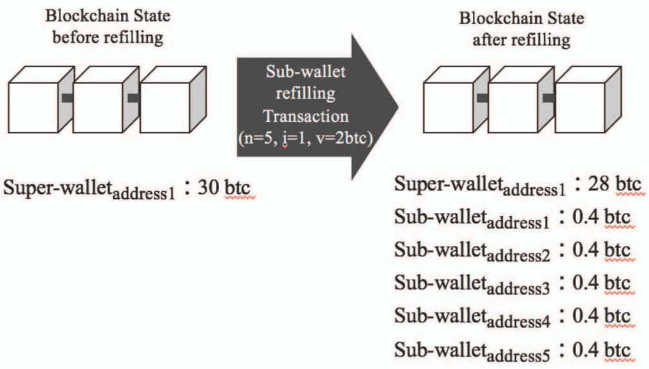
Blockchain State before refilling
Sub-wallet refilling Transaction (n=5, i=1, v=2btc)
Blockchain State after refilling
Super-wallet address1 : 30 btc
Super-wallet address1 : 28 btc
Sub-wallet address1 : 0.4 btc
Sub-wallet address2 : 0.4 btc
Sub-wallet address3 : 0.4 btc
Sub-wallet address4 : 0.4 btc
Sub-wallet address5 : 0.4 btc

Fig. 2. The simplified example of sub-wallet refilling in the blockchain. The left side demonstrates the blockchain state before publishing the sub-wallet refilling transaction, and the right side shows the state after that.

C. Sub-Wallet Seed Transporting

We need an algorithm to transport a sub-wallet seed (sub-seed) from the super-wallet to the sub-wallet securely. To do that, we employ a modified version of the seed transport algorithm that proposed in [9]. This algorithm is based on Elliptic-Curve Diffie-Hellman key (ECDH) agreement [3].

In ECDH, each party has its key pair, but both parties compute a shared secret with its private key and the other party's public key. Also, an additional SHA-256 computation of EDCH result value is recommended [3]. In our algorithm, we use the computed secret as an AES 256-bit encryption key to encrypt the sub-seed and transfer that from the super-wallet to the sub-wallet. The problem of ECDH is the Man-In-The-Middle attack where a hacker replaces the sub-wallet public key by hacker's public key, and the super-wallet cannot distinguish the sub-wallet public key from the hacker's one. To tackle this problem, we employ side-channel user visual confirmation called verification code aka vcode. Vcode is a cryptographic digest (hash value) computed from the sub-wallet public key. Each wallet computes the vcode independently and displays that on the its screen. The user visually compares the equity of two vcodes and ensures that no hacker replaces the sub-wallet public during the transport process. Then, she confirms that by pressing a physical button on the super-wallet (receiver). Visual confirmation is a regular method in existing hardware wallets to confirm transaction information like receiver address, amount and fee before signing [12].

IV. Prototype Implementation

One of the most secure crypto wallet is hardware wallet equipped with a screen and at least one physical button, else as [11] and [13] argued a crypto hardware is not secure when it uses a terminal (e.g., computer and smartphone) for interaction with the user, because a hacker may install malware on the terminal and make a Man-In-The-Middle attack. Traditional smart cards are not secure enough to use as a crypto wallet because of no direct input/output with the user. Fortunately, now there are new smart cards in the market that use e-paper technology as an on-card screen. This technology enables the smart card to display information to the user with no intermediate terminal. Also, buttons are available in these new smart cards. Thus, we use a smart card with a screen and a button as a hardware crypto wallet to implement our mechanism and Fig. 3 shows the photo of such a smart card.

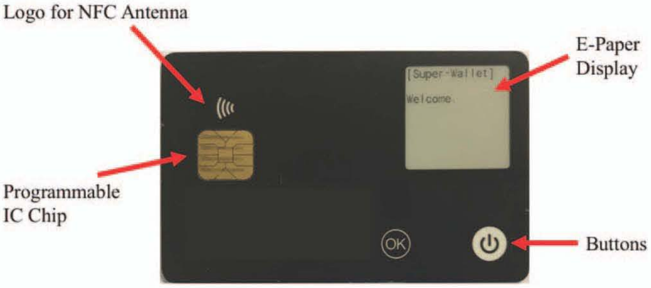

Fig. 3. Smart card with an e-paper display, physical buttons, and a programmable IC chip

To develop a card application for the smart card, we employ Java Card technology [14] which is a limited version of Java Runtime Environment with fewer features. We write and compile our program in Java, convert it to a Card Application (CAP) and load it to the programmable IC chip on the smart card. We implement our code with Java Card (JC) 3.0.1 API, and it can run on all JC compatible smart cards, but the screen API is vendor-specific.

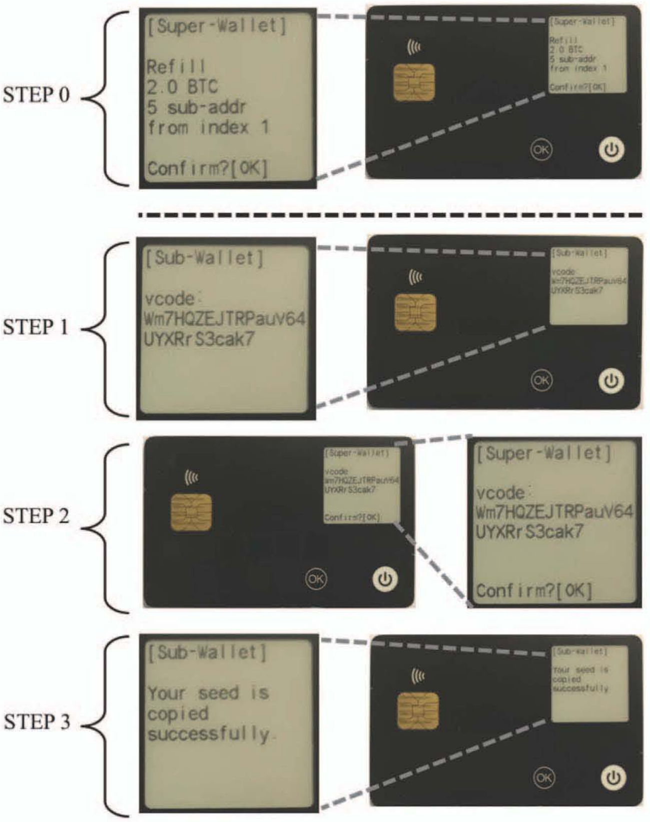

Fig. 4. The whole process of the sub-wallet refilling and the sub-seed transporting from the the user's perspective. Step 0 is for refilling the sub-wallet addresses and Step 1 to step 3 are for the secure sub-seed transport from the super-wallet to the sub-wallet.

The smart card has limited resources, and our test card has only 2.5-kilobyte memory. Thus, we have implemented our code efficiently to use minimum memory. A well-known technique that we used is sharing the memory. We define just two big arrays to allocate all available memory in one place and then pass them to all functions that require them. Also, we avoid very nested function callings and any recursive function because calling function requires stack allocation which consumes memory. In this type of programming inside a secure element (IC card) you should be very stingy and use each byte carefully. Because the refilling transaction is large for a smart card, we have to limit the number of sub-wallet addresses that the wallet can refill in one transaction. In our implementation for Bitcoin, we limit it to 16 sub-wallet addresses which are enough in significant cases. We have published our source code on GitHub [15]. Fig. 4 demonstrates the whole process from the user's perspective.

V. Performance Evaluation

In our performance test, we use a smart card reader connected to a laptop with a USB cord. We run each test case 10 times and use our evaluation program [16] to measure the period of sending and receiving packets.

We compare classic sub-wallet and deterministic sub-wallet in two scenarios. First, we assume that the user has several sub-wallets and wants to refill some of them simultaneously. In this scenario, the classic model creates one transaction per sub-wallet, but deterministic model creates one transaction for multiple sub-wallets. The performance result to execute this process on the test smart card (sample hardware wallet) is illustrated in Fig. 5. For one, two and three sub-wallets the classic model is a little bit better because it is similar to regular wallets and get all input addresses from outside of the hardware wallet with no internal process. On the other hand, the super-wallet on deterministic model derives sub-wallet seeds and addresses internally that takes more time, but for four sub-wallets and more it has better performance because of fixed overhead time to sign a transaction in the classic model.

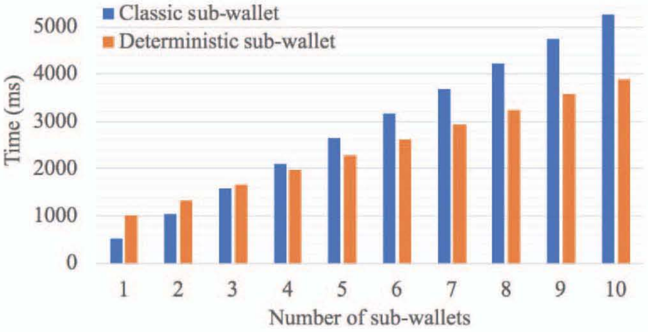

Fig. 5. Smart card execution time to refill multiple sub-wallets simultaneously

In the second scenario, we assume that the user has only one sub-wallet and wants to refill it repeatedly. For example, she refills her sub-wallet one time per month in a year. In this scenario, she may refill her sub-wallet for 1, 2, 3 to 12 months. In the classic model, she should create a blockchain transaction each time, but on the deterministic model, she can refill her sub-wallet for multiple months in one blockchain transaction. To compare the classic and the deterministic model in this scenario, we use the current metrics of the Bitcoin network [17]. For the time of writing this paper, TABLE II. shows the Bitcoin network metrics. In these calculations, we assume that the average transaction size is 250 bytes. Also, our mechanism to make deterministic sub-wallet adds 34 bytes per sub-wallet address except first one and it uses legacy addresses.

TABLE II. Bitcoin Network Metrics

Inserted block	Time for confirmation	Fee per byte	Fee per transaction
Next block	10 min	23 satoshi/byte	5750 satoshi
3 blocks	30 min	22 satoshi/byte	5500 satoshi
6 blocks	60 min	10 satoshi/byte	2500 satoshi

We compare the classic model with the deterministic model with these metrics for time and fee. To simplify the comparison, we only consider the worse cases. At first, to compare fee, we use the best fee that is 2500 satoshi per transaction with 60 min to confirm. In this situation, the classic model consumes less fee to refill the sub-wallet. Fig. 6 demonstrates the consuming fee for both models. For the classic model, the cost is the number of sub-wallet times transaction fee, but on the deterministic model, the cost is not very different for 1 to 12 refills and increase a small amount for additional 34 bytes per sub-wallet address.

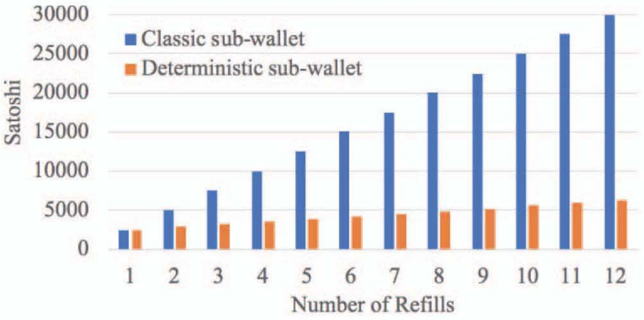

Fig. 6. Fee to refill one sub-wallet multiple times

The results for the time are similar and Fig. 7 shows the time results. In this comparison, we use the best network confirmation time (10 min) which cost more, but it is the best option for the classic model. Because the user should wait for network confirmation for each refill, it takes much time. On the other hand, because the deterministic wallet does all of that in one transaction, the time is not related to the number of refills.

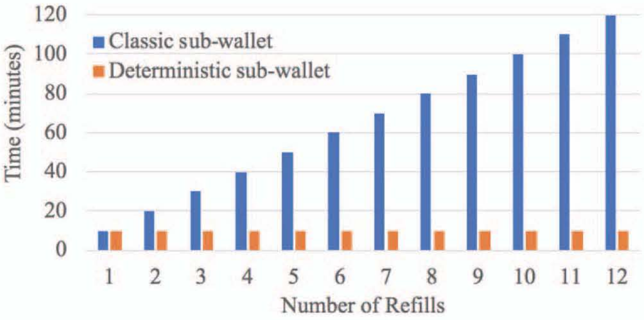

Fig. 7. Time to refill one sub-wallet multiple times

VI. Security Analysis

A. Assumptions and Threat Model

The goals for our scheme are secure refilling the sub-wallet addresses and secure transporting a sub-seed from the super-wallet to the sub-wallet. In our threat model, we have the following assumptions on hardware wallet, terminal, and user:

- The terminal, such as a computer, laptop or smartphone is untrusted and could be compromised by a hacker, e.g., by installing malware.
- The hardware wallets have a display and at least one physical button as illustrated in Fig. 3 similar to existing hardware wallets [12].
- The user follows the instructions and checks vcode on both wallets' displays during the sub-seed transfer procedure.

B. Less Super-Wallet Signings

Our proposed mechanism only needs one super-wallet transaction signing to refill multiple sub-wallet addresses. It decreases the permission required signing and provides better security than the classic model. In other words, the user's big fund is less accessible to the potential hackers.

C. Capturing Sub-Wallet Seed

A hacker may sniff the communication to steal the sub-wallet seed in two situations. First, it could happen when the user creates the sub-wallet refilling transaction on the super-wallet. To defend against this attack, we implement the entire procedures of sub-seed creation, private key derivation and address conversion on the super-wallet (e.g., via the onboard IC chip on a smart card). Thus, the terminal passes the sub-wallet index to the super-wallet, and there is no secret information to sniff. Second, the hacker may try to sniff the terminal when the user transports a sub-wallet seed from the super-wallet to the sub-wallet. The sub-seed is encrypted with AES-256 bit to avoid this attack, and there is no plaintext secret to steal.

D. MITM: Replacing Sub-Wallet Address

The hacker may want to make a Man-In-The-Middle (MITM) attack to modify the receiver address in the transaction before sending the inputs to the wallet. In this way, he can replace the legitimate receiver address by his address to steal the user's fund. The classic model is vulnerable to this attack because the sub-wallet key tree is unlinked, and the super-wallet needs to get the sub-wallet address from the input. In contrast, our proposed scheme avoids this attack by deriving the sub-wallet seeds from the super-wallet master seed and generating the sub-wallet addresses on the super-wallet. Therefore, there is no need to get the sub-wallet addresses from inputs and the hacker has no chance to replace them in the terminal.

E. MITM: Replacing Sub-Wallet Transport Public Key

Another possible MITM attack is that the attacker relays the messages between the supper-wallet and the sub-wallet and tries to replace the sub-wallet public key by the hacker's public key to convince the super-wallet to encrypt the sub-seed using the hacker's key. Then, the attacker computes the transport key using the super-wallet public key and his private key and decrypts the encrypted sub-seed.

To defend against this attack, we have used a verification code (vcode) in the sub-wallet seed transport algorithm. Both wallets compute their vcode of the sub-wallet public key and display that in their screens. The user must confirm the equality of them by pressing a physical button on the super-wallet. If a hacker imports his public key to the super-wallet, the user will be able to detect such an attack by comparing the two displayed wallets' vcode and hence reject this MITM attack.

VII. Conclusion

In this paper, we proposed a new scheme to create super-wallet and sub-wallet. It derives the sub-wallet seed from the super-wallet master seed, and we called it deterministic sub-wallet. We implemented this new mechanism on a hardware wallet as a proof-of-concept, and its performance was better than the classic super-wallet/sub-wallet model. Also, our security analysis illustrates that this mechanism is more secure than the classic one.

Acknowledgement

This work is supported by the National Science Foundation under grant DGE-1723587.

References

[1] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better - how to make Bitcoin a better currency", in Proceedings of The 16th Financial Cryptography and Data Security, 2012.

[2] S. Meiklejohn, "Top Ten Obstacles along Distributed Ledgers Path to Adoption", IEEE Security & Privacy, vol. 16, issu. 4, pp. 13-19, 2018.

[3] "SEC 1: Elliptic Curve Cryptography", Version 2.0, Standard for efficient cryptography group, 2009.

[4] "SEC 2: Recommended Elliptic Curve Domain Parameters", Version 2.0, Standard for efficient cryptography group, 2010.

[5] A. Narayanan, J. Bonneau, E. W. Felten, A. Miller, and S. Goldfeder, Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction, Princeton University Press, 2016.

[6] "Hierarchical Deterministic Wallets", Bitcoin Improvement Proposal 32 (BIP-0032), 2012.

[7] "Multi-Account Hierarchy for Deterministic Wallets", Bitcoin Improvement Proposal 44 (BIP-0044), 2014.

[8] "Mnemonic code for generating deterministic keys", Bitcoin Improvement Proposal 39 (BIP-0039), 2013.

[9] H. Rezaeighaleh, C. C. Zou, "New Secure Approach to Backup Cryptocurrency Wallets", in submission to IEEE Global Communications Conference - Communication & Information Systems Security Symposium, 2019.

[10] MultiSend Smart Contract [Online]. Available: https://github.com/Alonski/MultiSendEthereum

[11] H. Rezaeighaleh, R. Laurens, C. C. Zou, "Secure smart card signing with time-based digital signature", in Proceedings of the 2018 International Conference on Computing, Networking and Communications, pp. 182-187, 2018.

[12] "Hardware wallet", Bitcoin wiki, 2018 [Online]. Available: https://en.bitcoin.it/wiki/Hardware_wallet [Accessed Oct. 8, 2018].

[13] B. Schneier, and A. Shostack, "Breaking up is hard to do: modeling security threats for smart cards", USENIX Workshop on Smart Card Technology, USENIX Press, 1999, pp. 175-185.

[14] "Java Card Runtime Environment Specification," 3rd Edition, 2011.

[15] blackCardApplet [Online]. Available: https://github.com/hosseinpro/blackCardApplet

[16] smartcardPage [Online]. Available: https://github.com/hosseinpro/smartcardPage

[17] Bitcoin Fees [Online]. Available: https://bitcoinfees.info