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 the user's keys from potential hackers because if the blockchain network confirms a transaction, nobody can rollback that. One solution to protect users is splitting the money between super-wallet and sub-wallet. The user stores a large amount of money on the super-wallet and refills the sub-wallet when she needs while she uses the sub-wallet for her daily purchases. In this paper, we propose a new mechanism to create sub-wallet that we call deterministic sub-wallet. In this mechanism, the seed of sub-wallet keys is derived from super-wallet seed, and therefore super-wallet can build many sub-wallet addresses and refill them in a single blockchain transaction. Compared to existing approaches, our mechanism is less expensive, real-time, more secure against MITM attack and easier for backup and recovery. We implement a proof-of-concept on a hardware wallet and evaluate its performance. Also, we analyze the attacks and defenses in our mechanism to demonstrate that our proposed method has a higher level of security than the classic super-wallet sub-wallet model.**

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 news of hackers stealing a large amount of money from blockchain users. A major security issue in all cryptocurrencies, including Bitcoin and Litecoin, is the safety of a user's private key. Blockchains usually use elliptic-curve asymmetric cryptography to control the ownership of coins or accounts. In other words, to transfer a coin 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 network accepts and confirms this, nobody can roll back the transaction (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(s) has full control of the user's money, and because there is no central authority, she should stand on her own feet and keep her private key(s) safe by herself, which is one of the most critical challenges in cryptocurrencies [1], [7]. Users usually employ crypto wallets to generate and store their private key(s) and sign transactions. Crypto wallets have many forms from online wallets to mobile and cold wallets, but the most secure one is hardware wallet equipped with a specific secure element in the form of a USB stick, Bluetooth device or smartcard.

Even though the hardware wallet is secure, it is risky that a user puts all of her money on a device and uses it day-to-day to purchase. A smart and simple solution is proposed in [1] that 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 sub-wallet infrequently when needed. The sub-wallet is like a checking account role that stores a small amount of money 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 needs to send coins from her super-wallet address to her sub-wallet address. This mechanism is very straightforward but has significant drawbacks. First, each time that the user refills her sub-wallet, her super-wallet creates a transaction and publishes to the blockchain network. Thus, for each such transaction, she must pay the miner fee. Also, she should wait for confirmation, and refilling sub-wallet takes time. Also, refilling the sub-wallet is risky because a hacker could perform Man-In-The-Middle attack to replace the user's sub-wallet address by his poison address to receive coins 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 solution, in this paper, we propose a new model that we call deterministic sub-wallet. In this model, the sub-wallet seed is derived from super-wallet seed, and this process executes inside super-wallet. Therefore, super-wallet derives sub-wallet addresses and transfer coins to many of them in only one blockchain transaction. After that, the user refills her sub-wallet by transporting a seed from super-wallet to sub-wallet instead of creating a blockchain transaction. Consequently, this model can refill multiple sub-wallet addresses with just one mining fee and one-time waiting for confirmation. It is secure because super-wallet does not need to get sub-wallet addresses from the external environment and it prevents a MITM attack. Also, there is no need to back up sub-wallet. For proof-of-concept, we implement a prototype of deterministic sub-wallet in a hardware wallet with security element and evaluate its performance. In summary, our contributions in this paper are:

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

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 [3]. Therefore, the user has a pair of private key and public key and uses her private key to sign the transaction and transfer coins 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 cryptocurrencies because everyone can watch the address of everyone, and all transaction history is on the blockchain network. So, a hacker may know all of a user's purchases and transfers, and the user does not have privacy. To tackle this problem, the user should change her address in each transaction. This address is called 'change address', which means that she generates a new private key and public key each time to receive coins from others or receive remaining coins of her spending transaction. Thus, nobody can track her just by watching her transaction history, and this is the best practice in blockchain networks now [4]. However, If the user generates a random private key in each transaction, she should maintain a lot of private keys that are 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 [5]. 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 entropy called 'master seed'. In other words, anyone who has the master seed can derive all subordinate private keys. Consequently, the user 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 a path is letter 'm' that denotes master seed and subsequent numbers are the input indexes for child key derivation algorithm in the corresponding round [5]. 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) in various HD wallets [6]. 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 seed for HD wallets. The user must write these generated words (12 to 24 words) on a piece of paper and keep it safe. She can recover whole her key tree on a new wallet using these words. Crypto wallet usually uses this process to back up the master seed.

Therefore, there is a large universal tree that covers all keys of all coins for a user wallet and each key in the tree has a unique path, but ehese mechanisms are silent about the super-wallet sub-wallet model, and there is not any link between two wallet seeds. In our proposed mechanism, we use the HD wallet structure and add a link between the master seed of super-wallet and the master seed of sub-wallet that we called sub-seed.

B. *Classic Super-Wallet and Sub-Wallet Model*

The idea of super-wallet and sub-wallet is proposed in [1] that is separating the main account that conveys a large amount of money from spending account that is used for the daily transactions. The main account corresponds to super-wallet while spending account corresponds to sub-wallet. It mimics personal saving account and checking 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, for example, one or two times per month, and uses 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 money on that. Then, each time that she wants to refill her sub-wallet (second wallet), she retrieves a receiving address from the sub-wallet and sends coins 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 her sub-wallet. This process requires the user to pay miner fee and wait a period for confirmations. Because usually, her terminal (laptop or smartphone) is vulnerable to malware attacks, it is possible that a hacker replaces her sub-wallet address by his poison address to receive coins from the super-wallet. Also, 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 independent key trees, in our new mechanism, deterministic sub-wallet, we derive sub-wallet seeds from super-wallet master seed. Therefore, we can construct all sub-wallet key trees in super-wallet. Also, we use one super-wallet blockchain transaction to refill several sub-wallet addresses, and when the user wants to refill her sub-wallet, she needs to import one seed to her sub-wallet.

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

- Deterministic sub-wallet is cheaper because it refills multiple sub-wallet addresses with only one blockchain transaction, while classic model requires a blockchain transaction in each refill.
- Refilling sub-wallet is real-time in deterministic sub-wallet because it is only transporting a seed from super-wallet to sub-wallet without any transaction with blockchain network.
- The classic model is vulnerable to Man-In-The-Middle attack for poison key injection similar to other regular wallets, but deterministic sub-wallet is not because the sub-wallet addresses are generated inside super-wallet with no need to outside of the wallet.
- The user must back up both super-wallet and sub-wallet seeds in classic model, but in deterministic sub-wallet, there is no need to back up sub-wallet seed because it is derived from super-wallet seed and only one back up of super-wallet seed is enough.

The abstract process of deterministic sub-wallet is as follows. The super-wallet generates a pool of sub-wallet addresses and constructs a large transaction which transfer coins from one (or many) super-wallet addresses to generated sub-wallet addresses. Then, the super-wallet signs and publishes the transaction. After that, each time that the user wants to refill her sub-wallet, she exports a sub-wallet seed from the super-wallet and imports that to the sub-wallet securely. In our previous paper which is in submission, we proposed a secure cryptographic mechanism to transport (export and import) 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 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 master key generation function in [5] with some modifications. The core function is an HMAC-SHA512 with 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*

To refill one or many addresses (many sub-seeds) of the sub-wallet, we use a specific blockchain transaction created and signed by the super-wallet. The refilling algorithm 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 will be used to transfer coin from super-wallet to sub-wallet that is 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. We divide 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

Acronym	Meaning
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$_{address1}$) 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 network, the super-wallet address has 28 Bitcoin and each sub-wallet address (Sub-wallet$_{address1}$ to Sub-wallet$_{address5}$) has 0.4 Bitcoin.

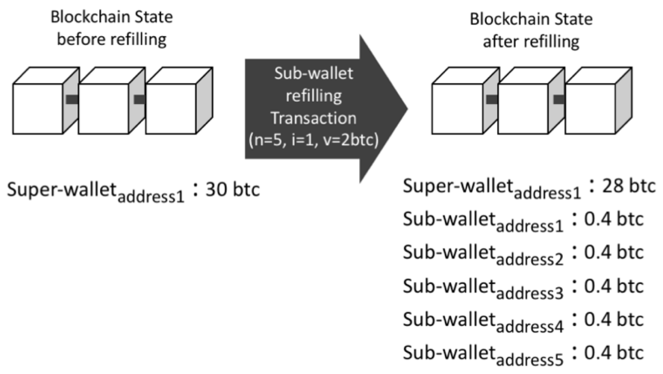
Blockchain State before refilling
Sub-wallet refilling Transaction (n=5, i=1, v=2btc)
Blockchain State after refilling
Super-wallet$_{address1}$: 30 btc
Super-wallet$_{address1}$: 28 btc
Sub-wallet$_{address1}$: 0.4 btc
Sub-wallet$_{address2}$: 0.4 btc
Sub-wallet$_{address3}$: 0.4 btc
Sub-wallet$_{address4}$: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

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 super-wallet to receive remaining coins in the left side is different from input super-wallet address in the right side. Furthermore, the sum of the fund before publishing and after publishing the refilling transaction are not equal because of the mining fee. Also, the first super-wallet address could be replaced by multiple super-wallet addresses to provide enough fund to refill the sub-wallet addresses.

C. *Sub-Wallet Seed Transporting*

We need an algorithm to transport a sub-wallet seed (sub-seed) from super-wallet to sub-wallet securely. To do that, we employ a modified version of the seed transport algorithm that we proposed in our previous paper which is in submission. This seed transport algorithm is based on Elliptic-Curve Diffie-Hellman key (ECDH) agreement [2]. To guarantee secure transfer, the wallet, which is preferred to be a hardware wallet, should have a screen to display a verification code (key check value) to the user, and a physical button to get her confirmation.

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 [2]. In our algorithm, we use the computed secret as an AES 256-bit encryption key to encrypt the sub-seed. Fig. 3 illustrates the steps of the mechanism and describes its acronyms. The goal of this algorithm is transporting a secure copy of a sub-seed from the super-wallet to the sub-wallet. For security, we assume both wallets are general hardware crypto wallets and have a screen, (at least) one physical button and is protected with a passcode. Also, we assume the transport channel is an untrusted terminal like a computer, laptop or smartphone that may be compromised by a hacker.

TABLE II. SUB-WALLET SEED TRANSPORT ALGORITHM ACRONYMS

Acronym	Meaning
$tkPri_x$	Elliptic-Curve Transport Private key of entity X
$txPub_x$	Elliptic-Curve Transport Public key of entity X
b58	Base-58
kcv_X	Key Check Value for Wallet X
ECDH	Elliptic-Curve Diffie-Hellman key agreement algorithm
tk	Transport Key (Symmentric)
encSubSeed	Encrypted Sub-seed

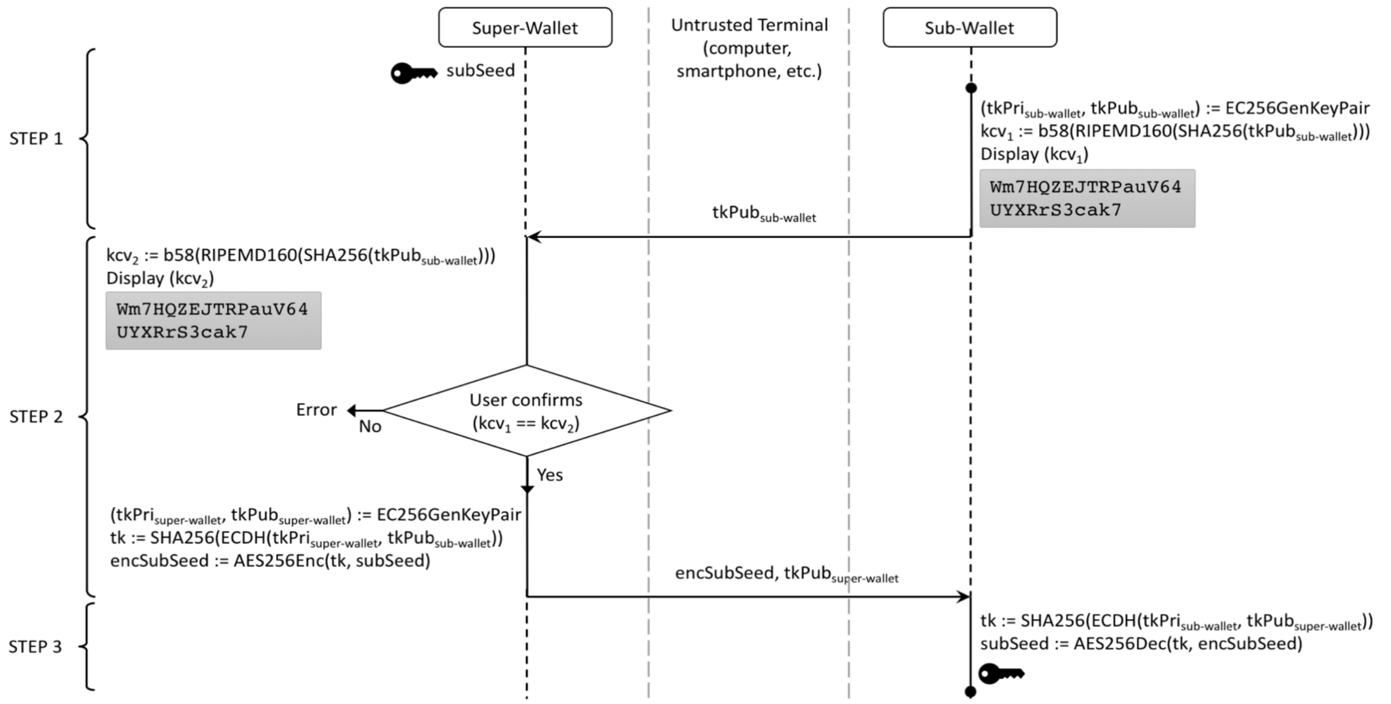
Super-Wallet
Untrusted Terminal (computer, smartphone, etc.)
Sub-Wallet
subSeed
STEP 1
(tkPri_sub-wallet, tkPub_sub-wallet) := EC256GenKeyPair
kcv1 := b58(RIPEMD160(SHA256(tkPub_sub-wallet)))
Display (kcv1)
Wm7HQZEJTRPauV64 UYXRrS3cak7
tkPub_sub-wallet
STEP 2
kcv2 := b58(RIPEMD160(SHA256(tkPub_sub-wallet)))
Display (kcv2)
Wm7HQZEJTRPauV64 UYXRrS3cak7
Error
No
User confirms (kcv1 == kcv2)
Yes
(tkPri_super-wallet, tkPub_super-wallet) := EC256GenKeyPair
tk := SHA256(ECDH(tkPri_super-wallet, tkPub_sub-wallet))
encSubSeed := AES256Enc(tk, subSeed)
encSubSeed, tkPub_super-wallet
STEP 3
tk := SHA256(ECDH(tkPri_sub-wallet, tkPub_super-wallet))
subSeed := AES256Dec(tk, encSubSeed)

Fig. 3. Secure sub-seed transport algorithm to transport a sub-seed from super-wallet to sub-wallet through an untrusted terminal. (The gray boxes illustrate information that is displayed on hardware wallets' screens for user verification. The values shown on the two wallets should be identical.)

IV. Prototype Implementation

The most secure crypto wallet is a hardware wallet equipped with a secure element, screen, and at least one physical button. Although all wallets can use our new mechanism to create super-wallet and sub-wallet, we choose hardware wallet for prototype implementation since it is more secure and realistic. We use a smart card that has a screen and button. As [9] and [12] argued, a traditional smart card is not secure for digital signature (applicable to crypto wallet) because it uses a terminal (e.g., computer, smartphone) for interaction with the user and a hacker may install a malware on the terminal and modify data before signing by the smart card.

Fortunately, now there are some smart cards in the market that use e-paper technology as an on-card display. This technology enables the smart card to directly show information to the user without relying on the external terminal. Also, capacitive and mechanical buttons are available in modern smart cards too. Thus, we use a smart card with a display and buttons to implement our mechanism. This smart card also supports near-field communication (NFC). Fig. 4 demonstrates the image of such a smart card.

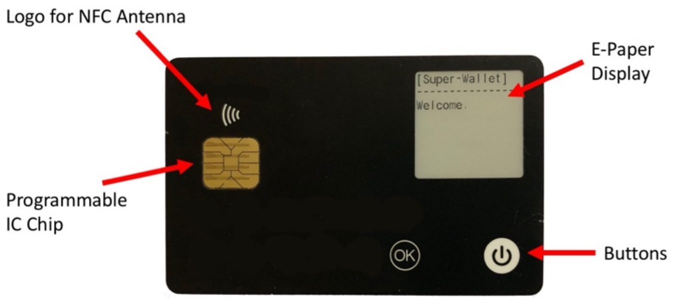

Fig. 4. New smart card with an e-paper display, physical buttons, and a programmable IC chip

To develop a card application to run on a smart card, we employ Java Card technology [13] 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 any JC compatible smart card, and only the display API is card-specific. JC 3.0.1 supports ECC 256-bit key generation and signing, SHA-256 digest algorithm, AES 256-bit encryption/decryption, and Elliptic-Curve Diffie-Hellman (ECDH) key agreement but does not include secp256k1 domain parameters that we need in cryptocurrency.

Furthermore, to calculate kcv, we use the SHA-256 hash algorithm to digest the public key, RIPEMD-160 hash algorithm to shorten the digest length and base-58 encoding to make it more readable for users. These algorithms are all supported and available on existing hardware wallets, but smart cards usually do not provide them. To resolve this issue, we utilize some codes in the Ledger Unplugged Java Card wallet GitHub repository [11] with a few minor changes to add these required algorithms. We have published our source code on GitHub as well [14].

As we mentioned, 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 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, we limit it to 16 sub-wallet addresses which are enough in significant cases.

In our prototype, we developed our code for Bitcoin, but our proposed mechanism applies to other similar altcoins too such as Litecoin. Fig. 5 demonstrates the whole process from the user's perspective.

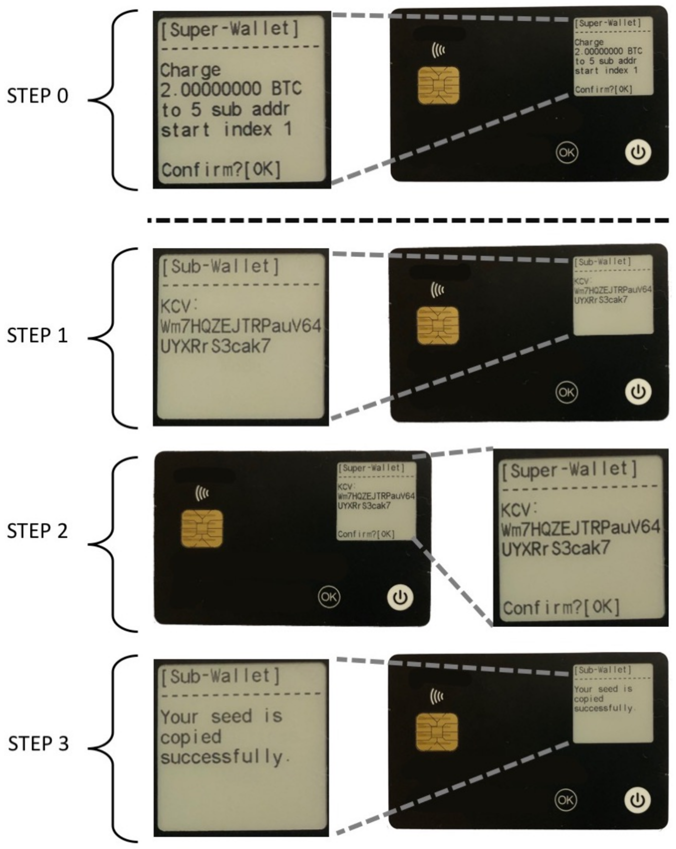

Fig. 5. The whole process of sub-wallet refilling and sub-seed transporting from the user's perspective. Step 0 is for refilling sub-wallet addresses and Step 1 to step 3 are for secure sub-seed transport from super-wallet to sub-wallet.

V. Performance Evaluation

In our performance test, we use a contactless (NFC) smart card reader connected to a laptop with a USB cord. We run each test case 10 times and use our evaluation program [15] 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. 6.

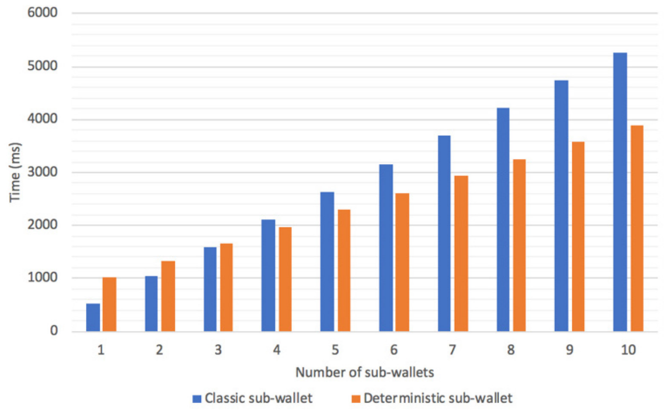

Fig. 6. Smart card execution time to refill multiple sub-wallets simultaneously

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. 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.

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 [16]. For the time of writing this paper, TABLE III. 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.

TABLE III. 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 sub-wallet. Fig. 7 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.

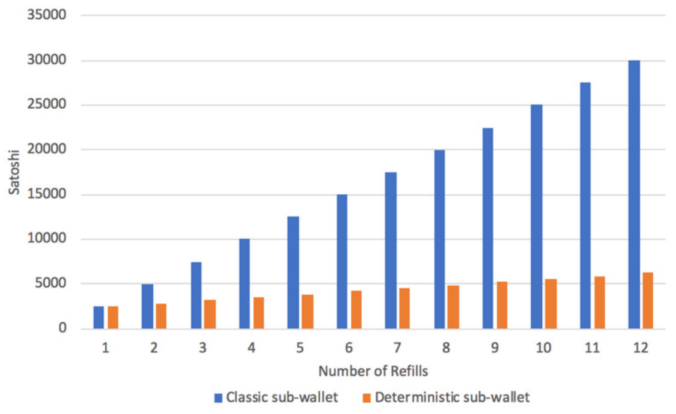

Fig. 7. Fee to refill one sub-wallet multiple times

The results for the time are similar. Fig. 8 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.

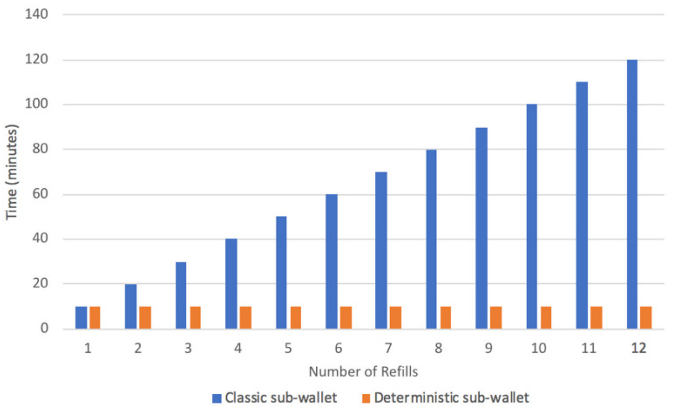

Fig. 8. Time to refill one sub-wallet multiple times

VI. Security Analysis

A. *Assumptions and Threat Model*

The goals for our mechanism are secure refilling sub-wallet addresses and secure transporting a sub-seed from super-wallet to sub-wallet. The threat model is as follows. 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 wallet has a secure element, a display and at least one physical button similar to Ledger Nano S and CoolWallet [10], as illustrated in Fig. 3.
- The hardware wallet is protected by a passcode (PIN-Code) to access private keys to prevent unauthorized access to the wallet.
- The master seed is generated securely on the main wallet, and nobody has a copy of that.
- The user follows the instructions and checks kcv on both wallets' displays during the 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 required permission signing and provides better security than the classic model. In other words, the user's big fund is less accessible to potential hackers.

C. *Capturing Sub-Wallet Seed*

A hacker may sniff the communication and steal the sub-wallet seed in two situations. First, the sniffing attack could happen when the user creates the sub-wallet refilling transaction on super-wallet. To defend against this sniffing 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 coins. The classic model is vulnerable to this attack because the sub-wallet key tree is independent, and the super-wallet needs to get the sub-wallet address from the input. In contrast, our proposed mechanism avoids this attack by deriving the sub-wallet seeds from the super-wallet master seed and generating the sub-wallet private keys and 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 Change Address*

The hacker may modify change address and replace it by his address. As we discussed, in HD wallet, change address is an address that remaining money of a transaction will return to that. In our case, when the user refills his sub-wallet addresses and pays the fee, the blockchain network returns remaining coins to the change address provided in the refilling transaction. If the hacker could replace this address, he steals all remaining fund from the super-wallet. To avoid that, in our implementation, we create the change address inside the super-wallet based on the master seed, and the terminal passes the key path of change address instead of change address itself. The key path is not subject to MITM because if the hacker modifies the key path, he cannot insert his address.

F. *MITM: Replacing 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 his poison public key to convince the super-wallet to encrypt the sub-seed using the poison 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 key check value (kcv) in the sub-wallet seed transport algorithm. Both wallets compute their kcv 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 poison public key to the super-wallet, the user will be able to detect such an attack by comparing the two displayed wallets' kcv values and hence reject this MITM attack.

VII. CONCLUSION

In this paper, we proposed a new mechanism to create super-wallet and sub-wallet. It derives sub-wallet seed from 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 and sub-wallet creation mechanism. 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, and the grant from US Army PEO STRI.

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

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

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

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

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

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

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

[9] 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.

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

[11] Ledger Unplugged [Online]. Available: https://github.com/LedgerHQ/ledger-javacard

[12] 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.

[13] "Java Card Runtime Environment (JCRE) Specification," 3rd Edition, 2011.

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

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

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