New Secure Approach to Backup Cryptocurrency Wallets

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— **Bitcoin and other cryptocurrencies have become popular and motivate more hackers to steal digital funds. Users protect their private keys using crypto wallets to keep their funds safe from hackers. While the most secure option is hardware wallet, it suffers from lack of a secure and convenient backup and recovery process. Almost all existing wallets use mnemonics to back up the private keys, and a user must write down these words on a piece of paper. This approach is not only inconvenient but also problematic since the paper could be lost or stolen, resulting in a hacker recovering the keys. In this paper, we propose a new digital scheme to securely back up a hardware wallet relying on the side-channel human visual verification enabled by display screen on a hardware wallet. Using this method, we transfer the root of private keys from one hardware wallet to another wallet securely even via an untrusted terminal, such as a smartphone. At the end of this process, the user has two hardware wallets with the same private keys while she may use one of them as the main wallet and another one as a backup wallet.**

Keywords—blockchain, cryptocurrency, hardware wallet, smart card, backup, and recovery.

I. Introduction

As blockchain and cryptocurrencies become increasingly popular and practical in various areas from purchasing a coffee to transferring vehicle ownership, they also become more attractive targets for hackers. Every week, we read the news of stealing money from exchanges, servers, and cryptocurrency owners. A big challenge in Bitcoin and all blockchain cryptocurrencies is securing private keys. Blockchain usually uses elliptic-curve asymmetric cryptography to control the ownership of coins or accounts. For example, to transfer a coin from a user to another, the user signs a transaction with her private key, and the blockchain network verifies the signature of the transaction with her public key. After being verified and accepted by the blockchain network, the transaction, unlike the traditional bank transfer, cannot be rolled back by anyone.

Consequently, the user's private key has full control of her crypto funds, and the most important task of a user is keeping her private keys safe, which is one of the essential challenges in cryptocurrencies [1]. Existing systems require crypto wallets to generate and store private keys and sign transactions. There are several forms of crypto wallets from online wallets to cold wallets, but the most secure one is hardware wallet which usually is a dedicated cryptographic device in the form of a USB stick, Bluetooth device or smart card.

A significant problem in current hardware wallets is the backup and recovery process, while almost all of them use a word list (mnemonics) to back up private keys and restore them when needed. The user must write these words on a piece of paper and keep it safe. This method converts the seed of private keys from digital form to physical form and moves the problem to the outside of the wallet. In this paper, we propose a new digital scheme for backup and recovery using Elliptic-Curve Diffie-Hellman (ECDH) algorithm [2]. This new approach is very convenient for a user because she does not need to write a word list and keep it safe. Also, at the end of the backup process, the user has two identical crypto wallets, and she can use both of them as a functional wallet without any additional recovery step.

In summary, our contributions in this paper are:

- Proposing the first crypto mechanism for secure backup and recovery in cryptocurrency hardware wallets relying on the side-channel human visual verification
- Implementing a prototype using a smart card as the hardware wallet and smartphone to realize the secure and convenient backup operation

In section II, we explain the required technical background of Hierarchical Deterministic wallet and hardware wallet. Section III is an overview of existing backup and recovery processes in crypto wallets. In section IV, we introduce our new approach for backup. Section V is about the prototype implementation of our idea in a smart card, and we evaluate its performance in section VI. Next, we define our assumptions and threat model and do a security analysis of our algorithm and its implementation in section VII. Finally, in section VIII, we finish the paper with a conclusion.

II. Technical Background

A. Hierarchical Deterministic Wallet

Bitcoin and almost all popular cryptocurrencies use elliptic-curve cryptography (ECC) to sign and verify transactions. A user has a pair of a private key and a public key and uses the private key to sign a transaction 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 keys in a specific format called address. Therefore, a user keeps her private key secret and publishes her address to other users.

Privacy, or in other words, anonymity, is a challenge in Bitcoin and several altcoins because whole transaction history is on the blockchain, resulting hackers may know all of a user's purchase and transfer activities. To tackle this problem, the user may change her address in each transaction and generate a new private key each time. In this way, nobody can track her just by watching the transaction history. It is a best practice in blockchain [4]. However, generating a random private key for each transaction causes difficulty for the wallet because it should maintain many keys. Deterministic wallet solves this problem using a deterministic algorithm to generate new private keys, and because it can be hierarchical, it is called Hierarchical Deterministic (HD) wallet [5]. An HD wallet has a tree of private keys while each node (private key) is derived from its parent. The root of this tree is a private key called 'master private key' derived from a random number called 'master seed'. In other words, anyone who has the master seed can build entire key tree and derive all private keys. Consequently, the user needs to keep only one seed value safe to protect all private keys.

B. *Hardware Wallet*

As argued in [6] one of the top ten obstacles of cryptocurrencies adoption is key management – the task of crypto wallets, which includes generating, storing and backing up keys. There are various crypto wallets with different security levels. If we ignore some simple forms like the brain wallet and the paper wallet, the most popular one is cloud wallet or hot wallet (e.g., Coinbase wallet [7]) where the user stores her keys on an online server like exchanges protected with a password or two-factor authentication. It is convenient, but if hackers exploit a cloud server all users' keys will be compromised. It occurs many times in the real world because these servers are a honeypot for hackers.

Another option is the software wallet that stores the keys on a computer or smartphone, but it is vulnerable to virus and hack [9] although there are new mobile wallets that use security features of smartphones such as ARM TrustZone for protection [10], [11]. There is another secure alternative, offline wallet or cold wallet which is a software wallet installed on an offline air-gapped device to avoid any online hack and virus. This device has no Internet connection and transfers keys and transactions with a USB stick. This type of wallet is still vulnerable to advanced attacks. For example, the author of [12] transfers the keys via ultrasound from an offline wallet to an adjacent online computer.

The most secure existing wallet is the hardware wallet which is a dedicated cryptography device to generate and store private keys and sign transactions, and authors of [13] introduced the early functional version of it. This type of wallet is a USB stick, Bluetooth device or smart card with special embedded software to do cryptography functions. There are various forms of hardware wallets, but a secure hardware wallet must have a screen and buttons to interact with the user directly. Otherwise, if a hardware wallet uses a terminal like a computer or a smartphone to communicate with the user, it is vulnerable to Man-In-The-Middle attack [14].

The smart card is a mature technology to build a hardware wallet. It has a tamper-resistant chip and usually has passed hardware security evaluations in a cryptographic module lab [18]. This chip is a secure element that has limited resources in terms of memory amount and processing power and unfortunately is hard to program. Even though all wallets can use our new scheme for backup and recovery, in this paper, we implement that on a smart card as a proof-of-concept to prove that a hardware wallet with limited resources could use this scheme.

III. Related Works

Hardware wallet as protected storage and trusted source of random numbers is responsible for generating and storing the master seed and other keys. Maybe the master seed is secure in a hardware wallet, but a wallet can be lost or broken and needs backup. Existing hardware wallets use a mnemonic word list to convert the master seed from digital form to physical form as a backup [7]. This list is a limited number (from 12 to 24) of words while more words provide higher security. This algorithm converts a seed value to several groups of bits (from 4 to 8 bits), and each group maps to an index of a word in a pre-defined 2048-word list. It makes a "sentence" that is a unique order of words. The user may either save this "sentence" (words) in a computer file that is not secure at all or writes them down on a piece of paper.

It is critical for the user to keep this paper in a safe place because whoever gets access to that can build the entire key tree. For better protection, the user may use a passphrase in the converting process and remember that for the recovery process. However, it brings two problems:

1. If a hacker finds the word list, he can make a brute force attack to the passphrase without any limitation. So, the user should choose a complex long passphrase.
2. If the user chooses a complex long passphrase and later forgets the passphrase, she cannot recover the keys and loses all funds.

One alternative solution is secret sharing [15]. Secret sharing mechanism splits the master seed to multiple parts (shares) that must be stored and protected separately. To recover the master seed, a threshold (e.g., two of three) of shares must be present. It has the following disadvantages:

1. Secret sharing would downgrade usability in crypto wallets because a user has to keep the multiple secrets safe to protect her fund.
2. Secret sharing requires a trusted terminal to create shares and recover them.

Another solution is multi-signature [16] where a user uses multiple private keys with a threshold (e.g., two of three) to sign a transaction; if she loses one (or more) of her keys, she still can protect her funds. Some literature like [4] and [6] advise the users to use multi-signature; however, it has these drawbacks:

1. Multi-signature has a similar challenge to secret sharing where the user must protect multiple secrets separately.
2. Multi-signature requires multiple wallets to sign a transaction, which would cause downgrade of usability in crypto wallets.

In our proposed scheme, we try to tackle these problems. In contrast to the paper-based backup, our scheme uses ECC to back up and restore the keys on another wallet. So, the user does not need to either write a list of words or remember a complex long passphrase. Additionally, our scheme does not require the user to protect multiple secrets similar to secret sharing and multi-signature which downgrades the usability of wallets. Furthermore, it does not enforce using a trusted terminal for backup and recovery and does not need multiple wallets to sign a transaction.

IV. Proposed Secure Backup Scheme

Our new scheme uses elliptic-curve cryptography to back up the keys. It employs Elliptic-Curve Diffie-Hellman (ECDH) key agreement protocol [2] for backup and recovery. In ECDH, each party has its key pair, and both parties compute a shared secret with its private key and the other party's public key. Fig. 1 illustrates a general view of ECDH. In ECC the private key value is a random scaler, and the public key is calculated with multiplying ('*') private key with the fixed-point G. G is the Generator point in secp256k1 domain parameters, and '*' is ECC multiplication [2], [3].

In Fig. 1, A and B have their private keys, and they exchange their public keys with each other. Then, A multiplies its private key with B's public key; the result is denoted as S_A, and B does the same calculation with its private key to calculate S_B. By replacing public keys with its corresponding calculation, it is clear that S calculation is $ecPri_A * ecPri_B * G$ in both parties and S_A is equal to S_B. In this way, both parties create a shared secret with only exchanging their public keys. Also, an additional SHA-256 computation of ECDH result value is recommended [2].

$$\begin{array}{lcr}
\mathbf{A} : ecPri_A & & \mathbf{B} : ecPri_B \\
ecPub_A = ecPri_A * G & & ecPub_B = ecPri_B * G \\
 & \xrightarrow{ecPub_A} & \\
 & \xleftarrow[ecPub_B]{} & \\
S_A = ecPri_A * ecPub_B & & S_B = ecPri_B * ecPub_A \\
\quad = ecPri_A * (ecPri_B * G) & & \quad = ecPri_B * (ecPri_A * G) \\
 & S_A == S_B &
\end{array}$$

Fig. 1. Elliptic-Curve Diffie-Hellman (ECDH) key agreement. As a result, both parties A and B have the same secret share S. However, it suffers from the Man-In-The-Middle attack.

The problem of ECDH is the Man-In-The-Middle attack where a hacker replaces the public key of B by a fake public key, and A cannot distinguish the original public keys from the fake one. To solve this problem, we employ side-channel user visual verification, where, the wallet displays a code aka vcode in its trusted screen and the user confirms that. Existing hardware wallets use a similar method by displaying the transaction information like receiver address, amount and fee on their screen before signing [15].

Fig. 2 illustrates our proposed backup scheme and TABLE I. describes the meanings of used acronyms. In the backup process, there are two wallets: the main wallet and the backup wallet. Before start, the main wallet has generated and stored the master seed, and the goal of our proposed backup process is to transfer a secure copy of the master seed from the main wallet to the backup wallet. We assume both wallets have a screen and (at least) one physical button. Also, we assume the backup channel is an untrusted terminal like a smartphone that may be compromised by a hacker.

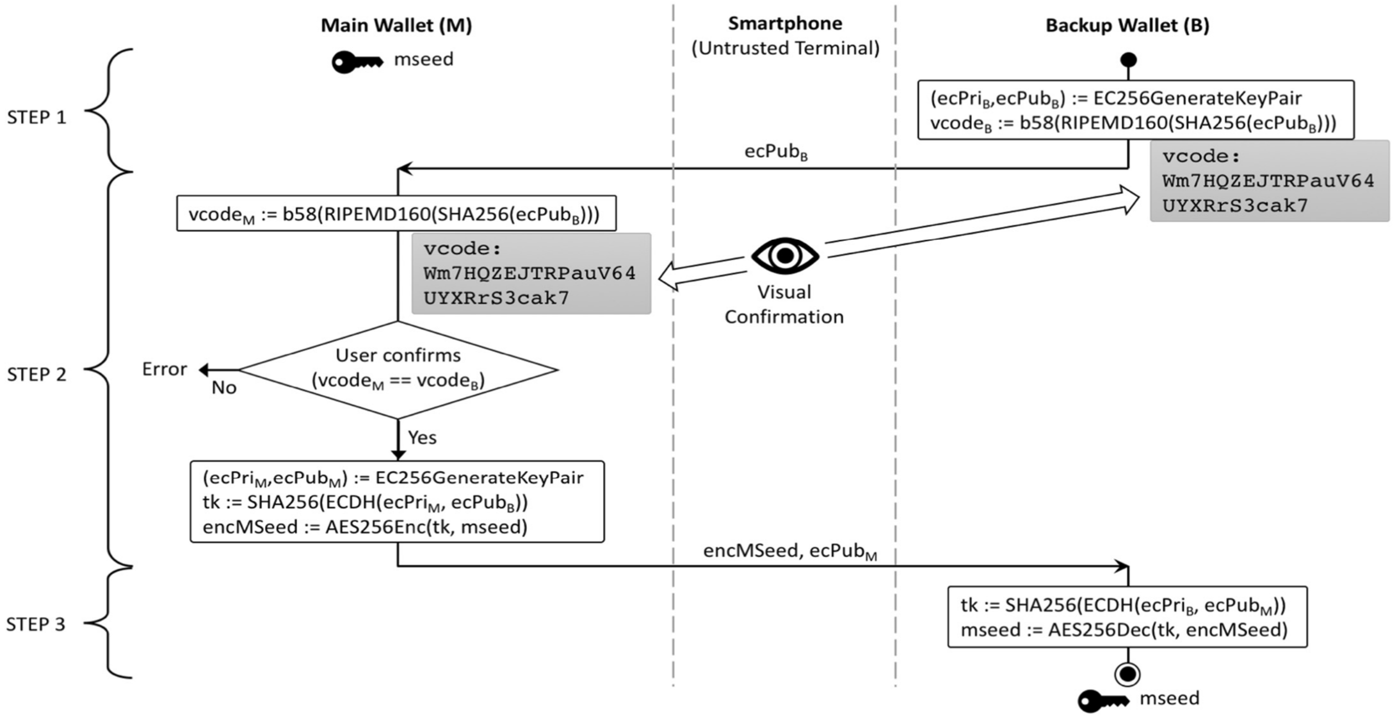
Main Wallet (M)
Smartphone (Untrusted Terminal)
Backup Wallet (B)
mseed
STEP 1
(ecPri_B,ecPub_B) := EC256GenerateKeyPair
vcode_B := b58(RIPEMD160(SHA256(ecPub_B)))
ecPub_B
vcode: Wm7HQZEJTRPauV64 UYXRrS3cak7
vcode_M := b58(RIPEMD160(SHA256(ecPub_B)))
vcode: Wm7HQZEJTRPauV64 UYXRrS3cak7
Visual Confirmation
STEP 2
Error
No
User confirms (vcode_M == vcode_B)
Yes
(ecPri_M,ecPub_M) := EC256GenerateKeyPair
tk := SHA256(ECDH(ecPri_M, ecPub_B))
encMSeed := AES256Enc(tk, mseed)
encMSeed, ecPub_M
STEP 3
tk := SHA256(ECDH(ecPri_B, ecPub_M))
mseed := AES256Dec(tk, encMSeed)
mseed

Fig. 2. Proposed secure backup mechanism to transfer master seed (mseed) from the main wallet (M) to the backup wallet (B) through an untrusted terminal. (The gray boxes illustrate the vcode that displayed on hardware wallets' screen for user verification. The values shown on the two wallets should be identical.)

TABLE I. ACRONYMS

Acronym	Meaning
mseed	Master Seed
$ecPri_x$	Elliptic-Curve Private key of wallet X
$ecPub_x$	Elliptic-Curve Public key of wallet X
b58	Base-58 encoding algorithm
$vcode_X$	Verification code of wallet X (displayed on the hardware wallet screen)
ECDH	Elliptic-Curve Diffie-Hellman algorithm
tk	Transport Key
encMSeed	Encrypted Master Seed

Our proposed mechanism has three steps:

1. The backup wallet generates a key pair and computes the verification code (vcode) of its public key to display on the backup wallet screen. Then it exports the backup wallet's public key ($ecPub_B$).
2. On the other side, the main wallet receives the backup wallet public key ($ecPub_B$) and calculates the same vcode to display on its screen. Then, the user visually compares these two vcodes in wallets' screens and confirms their match by pressing a button on the main wallet. Next, the main wallet generates its key pair and computes Transport Key (tk) using ECDH algorithm. Then, it encrypts the master seed (mseed) under transport key (tk) with AES 256-bit. Finally, it exports its public key ($ecPub_M$) and encrypted master seed (encMSeed).
3. The backup wallet imports $ecPub_M$ and encMSeed, computes Transport Key using ECDH algorithm and decrypts encMSeed to retrieve the master seed. Consequently, the backup wallet has the master seed to build the entire key tree.

V. PROTOTYPE IMPLEMENTATION

As we discussed, the most secure crypto wallet is a hardware wallet equipped with a screen and at least one physical button. However, as [14] and [20] argued, a traditional smart card is not secure for the digital signature because it uses a terminal (e.g., computer and smartphone) for interaction with the user, and a hacker may install malware on the terminal and make a Man-In-The-Middle attack. 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 to implement our mechanism and Fig. 3 shows the photo of such a smart card.

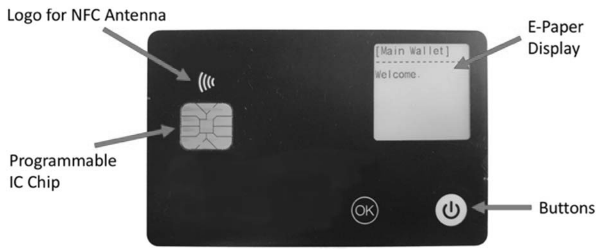

Fig. 3. Smart card with an e-paper display, physical buttons, and a programmable IC chip

To develop a card application for the smart card, we employ Java Card technology [21] 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 card-specific.

Java card (at least JC 3.0.1 API) supports ECC 256-bit key generation and signing/verification, 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, for vcode calculation, 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 supported and available on existing hardware wallets for address generation, but smart cards usually do not provide them. To resolve these issues, we utilize some codes of the Ledger Java Card wallet GitHub repository [19] with a few minor changes to add these algorithms. We have published our source code on GitHub as well [22].

Another challenge was the public key derivation. Existing hardware wallets back up the master seed and compute entire private key tree using the master seed; but what about the public keys? In ECC, as we explained in previous sections, the public key is derived from the private key. Therefore, the hardware wallet requires only the private key and calculates the corresponding public key with multiplying private key with the Generator point (G). In our prototype, the ECC multiplication is not easy due to the limited resources of the smart card. Therefore, we use ECDH function in a tricky way where we use the private key as the input private key and the Generator point (G) as the input public key. Since ECDH multiplies the input private key to the input public key, in this case, it multiplies the private key to the Generator point, and the result is the public key instead of a shared secret.

Additionally, in actual implementation, we split the second step of our mechanism (shown on Fig. 2) to two sub-steps to get confirmation from the user. Step 2.a includes importing the backup card public key, computing its vcode and getting confirmation from the user. Step 2.b includes encrypting the master seed and exporting the encrypted seed with the main card public key.

Fig. 4 demonstrates the 3-step process from the user perspective. At first, the user taps the backup card to the smartphone to generate a backup card key pair and export its public key. The backup card screen displays the calculated vcode, and the user sees the vcode on the smartphone to compare. Then, at the second step, she removes the backup card and taps the main card to the smartphone to import the backup card public key and export encrypted master seed with main card public key. During this step, the user must compare the vcodes displayed on the main card screen and smartphone and confirm their equality by pressing a physical button (OK button) on the main card. At the third step, she taps the backup card again to import and extract the master seed finally. The backup card screen displays a message to acknowledge the backup procedure completion.

In summary, our mechanism requires neither trusted terminal nor mutual authentication and session encryption between wallets. As a result, it can be deployed using only one regular smartphone with no additional device and no paper and is very convenient for average users.

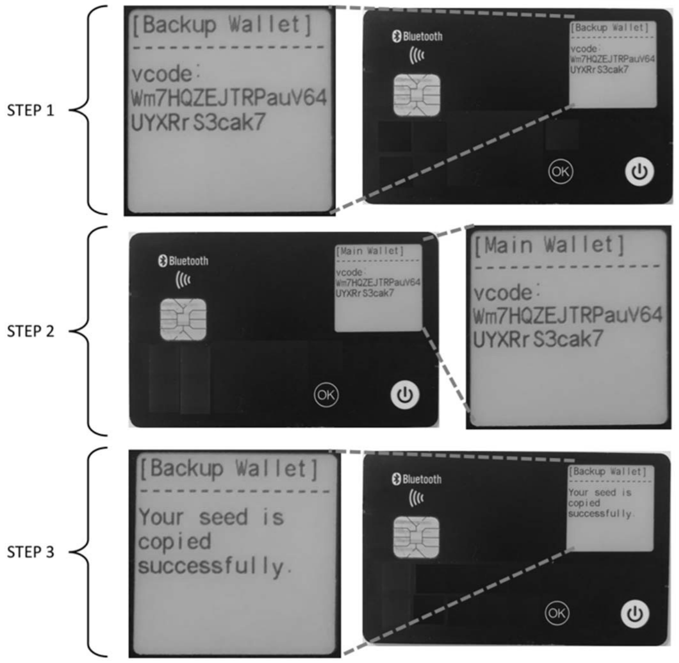

Fig. 4. The secure backup procedure from the user's perspective

VI. Performance Evaluation

To implement Diffie-Hellman key agreement in our backup algorithm, we choose ECC rather than RSA because of two reasons. First, existing hardware wallets support it, and second, it is faster than RSA. We express our RSA and ECC performance evaluation in Fig. 5. ECC execution time is not only faster but also more predictable and stable because RSA private key is a large random prime number (e.g., 2048-bit) that requires more time to compute on the IC chip. On the other hand, EC private key is a short random number (e.g., 256-bit) that is smaller than a fixed upper bound value [2].

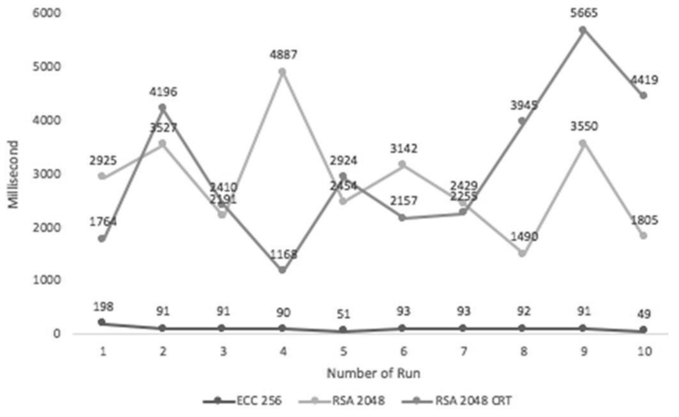

Fig. 5. Performance evaluation of ECC 256-bit, RSA 2048-bit and RSA 2048-bit CRT key pair generation on a smart card in 10 test runs

In the integration test, we evaluate the performance of the whole secure backup mechanism in our prototype wallet using a Google Pixel smartphone with NFC feature. We run our test case for ten times and use our test app to measure the time between sending and receiving packets to/from the smart card.

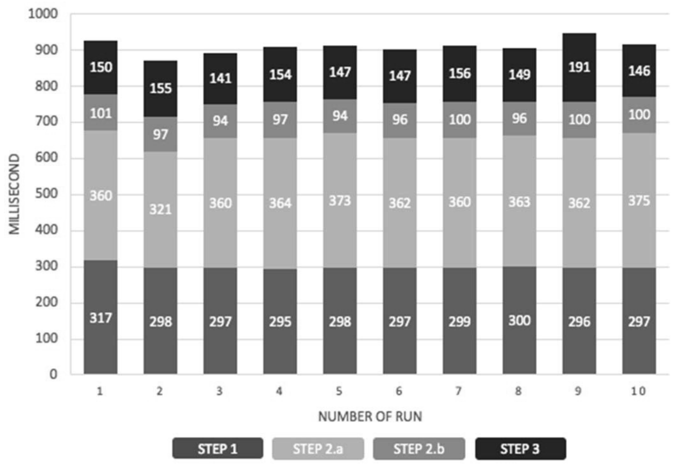

Fig. 6. Execution time performance results of each step in the secure backup procedure on a smart card in ten test runs

The complete evaluation results are illustrated in Fig. 6 for each step. This figure shows that the consuming time for each step is stable and predictable. According to our evaluation, the average total execution time for Step 1 on the smart card is 299.4 ms, for Step 2 is 457.5 ms and for Step 3 is 153.6 ms. Thus, the whole secure backup process takes no more than one second to complete based on our prototype evaluation.

VII. Security Analysis

A. *Assumptions and Threat Model*

The goal for our backup mechanism is securely transferring the master seed from the main wallet to the backup wallet. We have the following assumptions on the hardware wallet, terminal, and user:

- The terminal is a smartphone, which is untrusted and could be compromised by a hacker, e.g., by installing malware.
- The hardware wallet has a screen and at least one physical button as illustrated in Fig. 3 similar to existing hardware wallets [15].
- The master seed is generated securely on the main wallet, and nobody has a copy of the seed.
- The user follows the instructions and checks vcode on both wallets' screen during the backup procedure.

B. *Theft of Backup*

In existing backup solution on a piece of paper [7], if a hacker finds the backup paper with no passphrase, he can recover the master seed quickly and steal all funds. This attack happens regularly in the real-world robbery. On the other hand, in our proposed scheme, there is no plain text of the master seed to steal, and the backup is stored on another hardware wallet. If the hacker steals the backup wallet or the main wallet, he needs to know the password of the wallet to unlock it.

C. Vulnerability to Brute Force Attack

To improve backup security, the existing algorithm [7] supports an optional passphrase. Thus, the generated mnemonics require the passphrase to recover the master seed. Though, if a hacker finds the backup paper, he can make a brute force attack and try to guess the passphrase without any limitation. The hacker only needs one of the user's public addresses to perform this attack. For each guessed passphrase, he generates a new master seed and creates a set of addresses to match with the user's address. Therefore, the difficulty of this attack is similar to hacking a password. If the passphrase is simple, then the hacker can guess it quickly, and if it is complex, then the user may forget it and lose her funds.

On the other hand, our new scheme keeps the backup in another hardware wallet with a protected password. For example, in a smart card, there is a fixed password retry counter usually between 3 and 15, and after that, the smart card chip is locked automatically. So, hackers can only try a limited number of guessed passwords and could not make a brute force attack.

D. Capturing The Master Seed

In our proposed mechanism, an attacker may sniff the transmitted messages between wallets and the smartphone to eavesdrop the master seed. He can either capture NFC wireless communication or install a sniffing malware on the smartphone since we have assumed that the terminal is untrusted. Our mechanism is secure against this attack because the terminal observes only public information includes the main wallet and the backup wallet public key ($ecPub_M$ and $ecPub_B$), and encrypted master seed (encMSeed). Therefore, the attacker cannot extract any private data ($ecPri_M$, $ecPri_B$ and mseed).

E. MITM Attack: Replacing the Backup Public Key

Another possible attack is Man-In-The-Middle (MITM) where the attacker relays the messages between the main wallet and the backup wallet, trying to replace the backup wallet public key ($ecPub_B$) by his fake public key ($ecPri_H$) in ECDH key agreement. Then, the attacker can recover the master seed as:

```
tk' := SHA256(ECDH(ecPri_H, ecPub_M))
mseed := AES256Dec(tk', encMSeed)
```

To defend against this attack, we have used a side-channel verification code (vcode) in our mechanism. Both wallets compute their vcodes of the backup wallet public key ($ecPub_B$) and display their vcodes on their screens. The user visually inspects and confirms the equality of these two vcodes by pressing a physical button on the main wallet. So, if a hacker injects his fake public key ($ecPri_H$) to the main wallet, the user will be able to detect such an attack due to the mismatch of the two wallets' vcodes shown on two wallets' screen and reject this MITM attack.

VIII. Conclusion

A major security challenge in the blockchain is keeping the private keys safe from hackers. A practical and secure solution is cryptocurrency hardware wallet, but existing wallet practice uses an old-fashion non-digital mechanism and backs up the master seed (the root of the key tree) on a piece of paper. In this paper, we introduced a new digital cryptographic scheme based on Elliptic-Curve Diffie-Hellman key agreement and relying on the side-channel human visual verification of the trusted screen on the hardware wallet for secure backup and recovery. Our algorithm is fast and secure and can be used by all existing wallets even cryptocurrency hardware wallets with limited resources. Furthermore, we have developed a prototype of our proposed mechanism on a real smart card as a proof-of-concept.

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

[7] Coinbase home page, https://www.coinbase.com/

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

[9] W. Amir, "Critical Vulnerability in Electrum Bitcoin Wallets Finally Addressed", HackRead, Jan. 9, 2018 [Online]. Available: https://www.hackread.com/electrum-bitcoin-wallets-vulnerability-addressed [Accessed Oct. 8, 2018].

[10] M. Gentilal, P. Martins, and L. Sousa, "TrustZone-backed bitcoin wallet", in Proceedings of the Fourth Workshop on Cryptography and Security in Computing Systems, pp. 25-28, ACM, 2017.

[11] W. Dai, J. Deng, Q. Wang, C. Cui, D. Zou, and H. Jin, "SBLWT: A secure blockchain lightweight wallet based on Trustzone", IEEE Access, vol. 6, pp. 40638-40648, 2018.

[12] M. Guri, "BeatCoin: Leaking Private Keys from Air-Gapped Cryptocurrency Wallets", arXiv.org, 2018 [Online]. Available: https://arxiv.org/pdf/1804.08714.pdf [Accessed Oct. 8, 2018].

[13] T. Bamert, C. Decker, R. Wattenhofer, and S. Welten, "BlueWallet: The secure Bitcoin wallet", in Proceedings of the 10th International Workshop on Security and Trust Management, pp. 65-80, Springer, 2014.

[14] 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, ACM, pp. 182-187, 2018.

[15] A. Shamir, "How to share a secret", Communication of the ACM, Vol. 22, Issue 11, pp 612-613, 1979.

[16] "Multisignature", Bitcoin wiki, 2019 [Online]. Available: https://en.bitcoin.it/wiki/Multisignature [Accessed May. 7, 2019].

[17] "Hardware wallet", Bitcoin wiki, 2019 [Online]. Available: https://en.bitcoin.it/wiki/Hardware_wallet [Accessed May. 7, 2019].

[18] "FIPS PUB 140-2: Security requirements for cryptographic modules", National Institute of Standards and Technology, 2002.

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

[20] 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, pp. 175-185,1999.

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

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