

Electronic Theses and Dissertations, 2020-

2020

Improving Security of Crypto Wallets in Blockchain Technologies

Hossein Rezaeighaleh
University of Central Florida

 Part of the [Computer Sciences Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd2020>
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2020- by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Rezaeighaleh, Hossein, "Improving Security of Crypto Wallets in Blockchain Technologies" (2020).
Electronic Theses and Dissertations, 2020-. 403.
<https://stars.library.ucf.edu/etd2020/403>

IMPROVING SECURITY OF CRYPTO WALLETS
IN BLOCKCHAIN TECHNOLOGIES

by

HOSSEIN REZAEIGHALEH
M.S. University of Central Florida, 2018

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2020

Major Professor: Cliff C. Zou

© 2020 Hossein Rezaeighaleh

ABSTRACT

A big challenge in blockchain and cryptocurrency is securing the private key from potential hackers. Nobody can rollback a transaction made with a stolen key once the network confirms it. The technical solution to protect private keys is the cryptocurrency wallet, software, hardware, or a combination to manage the keys. In this dissertation, we try to investigate the significant challenges in existing cryptocurrency wallets and propose innovative solutions. Firstly, almost all cryptocurrency wallets suffer from the lack of a secure and convenient backup and recovery process. We offer a new cryptographic scheme to securely back up a hardware wallet relying on the side-channel human visual verification on the hardware wallet. Another practical mechanism to protect the funds is splitting the money between two wallets with small and large amounts. We propose a new scheme to create hierarchical wallets that we call deterministic sub-wallet to achieve this goal. The user can send funds from the wallet with a large amount to a smaller one in a secure way. We propose a multilayered architecture for cryptocurrency wallets based on a Defense-in-Depth strategy to protect private keys with a balance between convenience and security. The user protects the private keys in three restricted layers with different protection mechanisms. Finally, we try to solve another challenge in cryptocurrencies, which is losing access to private keys by its user, resulting in inaccessible coins. We propose a new mechanism called lean recovery transaction to tackle this problem. We make a change in wallet key management to generate a recovery transaction when needed. We implement a proof-of-concept for all of our proposals on a resource-constraint hardware wallet with a secure element, an embedded display, and one physical button. Furthermore, we evaluate the performance of our implementation and analyze the security of our proposed mechanisms.

ACKNOWLEDGMENTS

I want to extend special thanks to advisor Dr. Cliff Zou, who supported me from the early stage of the Ph.D. degree and helped with consulting in the study program, guided to publish high-quality papers, and provided the required equipment and devices. I would like to thank the committee members Dr. David Mohaisen, Dr. Xinwen Fu, and Dr. Fan Yao, who made helpful comments to improve this research.

TABLE OF CONTENTS

LIST OF FIGURES	xi
LIST OF TABLES	xiii
CHAPTER 1: INTRODUCTION	1
1.1 Problem and Motivation	1
1.2 Crypto Wallet Security	2
1.3 Crypto Wallet Backup Problem	2
1.4 Super-Wallet/Sub-Wallet Model	3
1.5 Defense-in-Depth Architecture	4
1.6 Avoiding Inaccessible Wallet	5
1.7 Document Structure	6
CHAPTER 2: TECHNICAL BACKGROUND	7
2.1 Cryptography Primitives	7
2.1.1 Hash Function	7
2.1.2 Hash-based Message Authentication Code	7
2.1.3 Symmetric Cryptography	8
2.1.4 Asymmetric Cryptography	8
2.1.5 Digital Signature	9
2.1.6 Elliptic-Curve Cryptography	9
2.1.6.1 Elliptic-Curve Domain Parameters for secp256k1	10

2.1.6.2	Elliptic-Curve Adding.....	11
2.1.6.3	Elliptic-Curve Doubling.....	12
2.1.6.4	Elliptic-Curve Multiplying.....	13
2.1.6.5	Elliptic-Curve Key Generation	14
2.1.6.6	Elliptic-Curve Digital Signature Generation	14
2.1.6.7	Elliptic-Curve Digital Signature Verification	15
2.2	Blockchain Technology	15
2.2.1	History.....	15
2.2.2	Blockchain Mechanics	16
2.2.3	UTXO-based versus Account-based Blockchain.....	18
2.2.4	Smart Contract	19
2.2.5	Consensus Mechanisms	19
2.2.5.1	Proof-of-Work.....	20
2.2.5.2	Proof-of-Stake.....	21
2.2.5.3	Delegated Proof-of-Stake.....	21
2.2.6	Blockchain Networks.....	22
2.2.6.1	Public Blockchain	22
2.2.6.2	Private Blockchain	22
2.3	Crypto Wallet.....	23
2.3.1	Wallet Types	23
2.3.1.1	Brain Wallet	23

2.3.1.2	Paper Wallet	24
2.3.1.3	Hot Wallet	24
2.3.1.4	Cold Wallet	25
2.3.1.5	Desktop Wallet	25
2.3.1.6	Mobile Wallet	25
2.3.1.7	Hardware Wallet	26
2.3.2	Hierarchical Deterministic Wallet	27
2.3.2.1	BIP-32: Hierarchical Deterministic Wallets	28
2.3.2.2	BIP-39: Mnemonic code for generating deterministic keys	30
2.3.2.3	BIP-44: Multi-Account Hierarchy for Deterministic Wallets	31
2.4	Smart Card	33
2.4.1	IC Card Components	33
2.4.2	Java Card Technology	35
2.4.3	Global Platform	37
2.4.4	Smart Card Programming	37
2.4.5	Smart Card Simulation	38
CHAPTER 3: FINDINGS		40
3.1	Smart Card Security	40
3.1.1	Threat Model	40
3.1.2	Fundamental Vulnerabilities	41
3.1.2.1	Capturing the Smart Card PIN	41

3.1.2.2	Altering the Digital Signature	42
3.1.3	Implementation of Smart Card Attacks	42
3.1.4	New Smart Card Capabilities.....	44
3.2	Crypto Wallet Backup	45
3.2.1	Existing Solutions	45
3.2.1.1	Paper Backup	45
3.2.1.2	Secret Sharing	46
3.2.1.3	Multi-Signature Wallet	46
3.2.1.4	Backup on the Cloud.....	47
3.2.2	Proposed Crypto Wallet Cloning Mechanism	47
3.2.2.1	Elliptic-Curve Diffie-Hellman Key Agreement.....	48
3.2.2.2	Proposed Algorithm	49
3.2.3	Prototype Implementation on Smart Card	51
3.2.4	Performance Evaluation	54
3.2.5	Security Analysis	56
3.2.5.1	Assumptions and Threat Model	56
3.2.5.2	Theft of Backup Attack.....	57
3.2.5.3	Vulnerability to Brute Force Attack	57
3.2.5.4	Capturing the Master Seed.....	58
3.2.5.5	MITM Attack: Replacing the Backup Public Key.....	58
3.3	Super-Wallet/Sub-Wallet.....	59

3.3.1	Classic Super-Wallet/Sub-Wallet Model	59
3.3.2	Proposed Deterministic Sub-Wallet.....	60
3.3.3	Classic versus Proposed Super-Wallet/Sub-Wallet Model.....	61
3.3.4	Proposed Deterministic Sub-Wallet Details	62
3.3.4.1	Sub-Wallet Seed Derivation	62
3.3.4.2	Sub-Wallet Refilling	63
3.3.4.3	Sub-Wallet Seed Transporting	65
3.3.5	Prototype Implementation on Smart Card	66
3.3.6	Performance Evaluation.....	68
3.3.7	Security Analysis	71
3.3.7.1	Assumptions and Threat Model.....	71
3.3.7.2	Less Super-Wallet Signings.....	72
3.3.7.3	Capturing Sub-Wallet Seed.....	72
3.3.7.4	MITM: Replacing Sub-Wallet Address	72
3.3.7.5	MITM: Replacing Sub-Wallet Transport Public Key.....	73
3.4	Multilayered Defense-in-Depth Architecture	73
3.4.1	Proposed Multi-Layer Wallet	74
3.4.2	Proof-Of-Concept	78
3.4.3	Security Analysis	79
3.4.3.1	Security Advantages	79
3.4.3.2	Adversary Models.....	80

3.5	Off-Chain Transaction to Avoid Inaccessible Wallet.....	84
3.5.1	Recovery Transaction	85
3.5.2	Hardware Wallet Architecture	86
3.5.3	Evaluating Recovery Transaction for Hardware Wallets	88
3.5.4	Proposed Lean Recovery Transaction	90
3.5.5	Evaluation	94
CHAPTER 4: CONCLUSION		96
REFERENCES		97

LIST OF FIGURES

Figure 2-1: Simplified Example of Transaction Signing in Bitcoin	9
Figure 2-2: Elliptic-Curve Domain Parameters for secp256k1	10
Figure 2-3: Elliptic-Curve Graph of $y^2 = x^3 + 7$	11
Figure 2-4: Addition of Two Points in ECC with Geometry Approach	11
Figure 2-5: Addition of Two Points in ECC with Algebraic Approach	12
Figure 2-6: Doubling of a Point in ECC with Geometry Approach	12
Figure 2-7: Doubling of a Point in ECC with Algebraic Approach	13
Figure 2-8: Multiplying of Two Points in ECC	13
Figure 2-9: Digital Signature Generation in ECC.....	14
Figure 2-10: Digital Signature Verification in ECC	15
Figure 2-11: Chain of blocks in the blockchain.....	17
Figure 2-12: Sample Bitcoin Brain Wallet	24
Figure 2-13: BIP-32 master key generation function	28
Figure 2-14: BIP-32 private parent key to private child key function	29
Figure 2-15: BIP-32 public parent key to public child key function	29
Figure 2-16: Smart card vs. magnetic-stripe card.....	34
Figure 2-17: Sample smart card chip layout	35
Figure 2-18: Java Card Runtime Environment (JCRE) Architecture	36
Figure 2-19: Java card application compiling and loading process	37
Figure 2-20: Command and response APDU structure	38
Figure 3-1: Windows smart card software stack vs. hacked software stack.....	43
Figure 3-2: Smart card with an e-paper display, physical buttons, and an IC chip	45
Figure 3-3: Elliptic-Curve Diffie-Hellman (ECDH) key agreement	48

Figure 3-4: Proposed secure backup mechanism to transfer master seed.....	50
Figure 3-5: The proposed secure backup procedure from the user perspective	54
Figure 3-6: Performance of ECC 256-bit and RSA 2048-bit on a smart card.....	55
Figure 3-7: Performance results of the secure backup procedure on a smart card	56
Figure 3-8: Capture the master seed by injecting a key by a hacker	58
Figure 3-9: Sub-wallet refilling pseudo-code	64
Figure 3-10: Simplified example of proposed sub-wallet refilling mechanism	65
Figure 3-11: Sub-wallet refilling and sub-seed transporting from the user’s perspective	68
Figure 3-12: Smart card execution time to refill multiple sub-wallets simultaneously	69
Figure 3-13: Fee to refill one sub-wallet multiple times.....	70
Figure 3-14: Time to refill one sub-wallet multiple times	71
Figure 3-15: The proposed multi-layer defense-in-depth architecture for cryptocurrency wallets	75
Figure 3-16: General hardware wallet components	87
Figure 3-17: Performance of generating recovery transaction on a Trezor One hardware wallet	89
Figure 3-18: Performance of generating recovery transaction on a Ledger Nano S hardware wallet.....	90
Figure 3-19: Sample key tree to illustrate the coverages of Recovery Transaction and our proposed Lean Recovery Transaction.....	91
Figure 3-20: Example for comparing lean recovery transaction with recovery transaction	93
Figure 3-21: Comparison of micropayment transactions in recovery transaction proposed in [57] and our proposed lean recovery transaction schemas	95

LIST OF TABLES

Table 2-1: Relation between Entropy and Mnemonic Sentence.....	30
Table 2-2: Account Discovery Process in a Sample HD Wallet	33
Table 3-1: Acronyms of proposed secure backup mechanism	50
Table 3-2: Sub-wallet Refilling pseudo-code Acronyms.....	64
Table 3-3: Bitcoin Network Metrics	70
Table 3-4: Adversary Model I: Malicious App with Dangerous Permission	81
Table 3-5: Adversary Model II: Physical Access	83

CHAPTER 1: INTRODUCTION

1.1 Problem and Motivation

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 almost all blockchains is securing the private keys. Blockchain usually uses elliptic-curve asymmetric cryptography to control the ownership of coins or accounts. For example, a user signs a transaction with her private key to transfer coins to another one, and the blockchain network verifies the signature of the transaction with her public key. After being confirmed by the blockchain network, the transaction, unlike the traditional bank transfer, cannot be rolled back by anyone.

Consequently, the private key has full control of the crypto funds, and the most crucial task of the user is keeping her private keys safe. It is one of the essential challenges in cryptocurrencies [1]. Existing systems require a particular software or hardware called crypto wallet to store the private keys and sign the transactions. Crypto wallets have a spectrum from online wallet to cold wallet while many experts believe the most secure one is the hardware wallet. It usually is a dedicated cryptographic device in the form of a USB stick, Bluetooth device, or smart card. Even though the hardware wallet is secure in many aspects, some essential issues should be addressed. In this work, we consider these issues and propose innovative schemes to solve them.

1.2 Crypto Wallet Security

At first, we consider the security of the existing hardware wallets and search to find an appropriate hardware device to implement that. So, we find a secure hardware wallet must have direct input and output like a display and few buttons to communicate with the user directly without trusting to the terminal such as a computer and smartphone. Furthermore, a hardware wallet must have a Secure Element to store secrets and keys and perform cryptographic operations. As we believe, the most promising device to build a secure hardware wallet is the smart card (IC card). However, the traditional smart cards do not have any display and button and have fundamental vulnerabilities. Thus, in the first research we focus on smart card security and implement such attacks to one the most pervasive smart cards. Then, we propose using a server-based solution to solve the digital signature security issue in the traditional smart card. We published the paper of the result of our research and attack implementation in the 2018 International Conference on Computing, Networking and Communications (ICNC-2018) as “Secure Smart Card Signing with Time-based Digital Signature” [2].

1.3 Crypto Wallet Backup Problem

The first significant problem in current hardware wallets is the backup and recovery process. 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 work, we propose a new digital scheme for backup and recovery using Elliptic-Curve Diffie-Hellman (ECDH) algorithm [3]. This new approach is very convenient for a user because she does not need to write a word list and keep it safe. At the end of the backup process, the user has two same crypto wallets, and she can use both of

them as a functional wallet without any additional recovery step. We did this research and development and published our paper in the 2019 IEEE Global Communications Conference (GLOBECOM-2019) as “New Secure Approach to Backup Cryptocurrency Wallets” [4].

1.4 Super-Wallet/Sub-Wallet Model

The second issue in crypto wallets is separating funds between wallets. Even though the hardware wallet is a secure option, it is risky that the user puts all of her funds on one 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 saves 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, every time a user wants to refill her sub-wallet, she sends the 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 that 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 original 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. The super-wallet calculates the sub-wallet addresses and transfer the fund to them in only one blockchain transaction. To refill, the user transports a sub-wallet 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 the 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. We did the research and development of this project and published our paper in the 2019 IEEE International Conference on Blockchain (Blockchain-2019) as “Deterministic Sub-Wallet for Cryptocurrencies” [5].

1.5 Defense-in-Depth Architecture

In this project, we propose a multilayered architecture for cryptocurrency wallets based on a Defense-in-Depth strategy to protect private keys with a balance between convenience and security. Defense-in-Depth (DiD) is an approach in IT security that usually conveys multiple layers with various security mechanisms to protect a system from attacks in several steps. The user protects the private keys in three restricted layers with different protection mechanisms. So, a single breach cannot threaten the entire fund, and it saves time for the user to respond. We implement a proof-of-concept of our proposed architecture on both a smart card hardware wallet and an Android smartphone wallet with no performance penalty. Furthermore, we analyze the security of our proposed architecture with two adversary models. We published our paper in the IEEE 6th International Conference on Computer and

Communications (ICCC2020) as “Multilayered Defense-in-Depth Architecture for Cryptocurrency Wallet” [6].

1.6 Avoiding Inaccessible Wallet

Blockchain user locks her private keys with a password and stores them on a piece of software or a hardware wallet to protect them. A challenge in cryptocurrencies is losing access to private keys by its user, resulting in inaccessible coins. These coins are assigned to addresses which access to their private keys is impossible. Today, about 20 percent of all possible bitcoins are inaccessible and lost forever. A promising solution is the off-chain recovery transaction that aggregates all available coins to send them to an address when the private key is not accessible. Unfortunately, this recovery transaction must be regenerated after all sends and receives, and it is time-consuming to generate on hardware wallets. In this project, we propose a new mechanism called *lean* recovery transaction to tackle this problem. We make a change in wallet key management to generate the recovery transaction as less frequently as possible. In our design, the wallet generates the lean recovery transaction only when needed and provides better performance especially for micropayment. We evaluate the regular recovery transaction on two real hardware wallets and implement our proposed mechanism on a hardware wallet. We achieve a %40 percentage of less processing time for generating payment transactions with few numbers of inputs. The performance difference becomes even bigger with larger number of inputs. We published our paper in the Third International Workshop on Blockchain Systems and Applications (BlockchainSys2020) in conjunction with IEEE TrustCom 2020 as “Efficient Off-Chain Transaction to Avoid Inaccessible Coins in Cryptocurrencies” [7].

1.7 Document Structure

In this document, we first overview the technical background of crypto wallets. We review the required cryptography primitives like hash function, digital signature, and elliptic-curve cryptography. Then, we introduce the blockchain technology, including blockchain mechanics, consensus mechanisms, smart contract, and various types of blockchain. Next, we consider crypto wallets, which includes explaining different wallet types and hierarchical deterministic wallet. We also describe smart card technology as a secure option to implement the hardware crypto wallet. In the next chapter, we start by discussing smart card security issues and explain our implementation to evaluate them. We continue in the next chapter with argue about existing crypto wallet backup mechanisms and their drawbacks. Then, we propose a new cryptographic backup mechanism based on elliptic-curve Diffie-Helman. Following sections present the prototype implementation, performance evaluation, and security analysis of this new mechanism. Then, we explain the details of our proposed solution for super-wallet/sub-wallet model. We describe our prototype implementation, performance evaluation, and security analysis of this model. Next, we propose our multi-layer architecture for cryptocurrency wallets that provide a defense-in-depth architecture. This model uses our previous proposed mechanism for wallet cloning and derivation. We present our proof-of-concept implementation and depict two adversary model for security analysis. Finally, we propose a key management schema to avoid inaccessible wallet using an efficient off-chain transaction. In this section, we examine the current solution on two hardware wallets to illustrate that it is not applicable to resource-constraint wallets and then evaluate our proposed efficient mechanism on a hardware wallet. In the end, we finish with the conclusion.

CHAPTER 2: TECHNICAL BACKGROUND

2.1 Cryptography Primitives

In this section, we provide an overview of the essential concepts of cryptography required for the subject of this dissertation.

2.1.1 Hash Function

The hash function is a one-way procedure to create a unique digest for a message or transaction. One-way means everybody can compute the digest from the message, while the reverse of this function; recovering the message from its hash, is not possible or is very hard to do. Hash functions generate a fixed-length hash value, and the message length can be larger than the hash length. Since the hash functions map a larger set to a smaller set, they have collisions. The stronger hash function has a lower collision probability.

Blockchain technology employs the hash function in several situations. For example, the Bitcoin transaction ID is the hash of the whole transaction body, and the key derivation and address generation procedures use hash functions. The most popular hash algorithm in Bitcoin is SHA256, SHA512, and RIPEMD160.

2.1.2 Hash-based Message Authentication Code

Hash-based message authentication code (HMAC) is a specific type of message authentication code that includes a hash function and a secret key. Therefore, HMAC provides both integrity and authentication of data. The most used HMAC algorithms in crypto wallets are HMAC-SHA256 and HMAC-SHA512.

2.1.3 Symmetric Cryptography

Symmetric encryption refers to a cryptography scheme where the encryption key and the decryption key are same. It means that everyone who has the secret key can both encrypt and decrypt all messages. The most popular symmetric algorithms are Triple-DES and AES.

The main advantage of symmetric cryptography is the performance where encryption and decryption functions are fast and make it suitable for large data encryption. On the other hand, the significant disadvantage of symmetric cryptography is the key distribution which requires a pre-existing key or out-of-band channel.

2.1.4 Asymmetric Cryptography

In contrast to symmetric algorithms, asymmetric cryptography uses different keys for encryption and decryption. In the encryption scheme, the sender uses the receiver “public key” to encrypt a message, and the receiver uses her own “private key” aka “secret key” to decrypt the ciphertext. Therefore, the receiver publishes her public key to the desired senders. Everyone who has the receiver public key can encrypt a message for her. However, only the receiver can decrypt the cipher because only she has the corresponding private key. The most popular asymmetric algorithms are RSA and ECC.

In comparison to the symmetric cryptography, the main advantage of asymmetric cryptography is the easier key distribution. On the other hand, the encryption and decryption functions are usually slower. Thus, for encryption scheme, the hybrid solution is more realistic while a symmetric key encrypts the data and an asymmetric public key encrypts the symmetric key. So, only who has the asymmetric private key can decrypt the encrypted aka “wrapped” symmetric key and consequently decrypt the data.

2.1.5 Digital Signature

The digital signature is a process in asymmetric cryptography when the signer uses her private key to sign the hash of a message and generates a signature value. The verifier verifies the signature using the signer public key. The digital signature provides data integrity and authenticity in addition to non-repudiation. The last one means the signer cannot deny her action to sign a signed message.

The digital signature is one of the fundamental elements of blockchain technology. In the blockchain, each user has a private key aka secret key and digitally signs a transaction to transfer funds to others or initiate an on-chain function. Figure 2-1 demonstrates a simplified transaction signing in Bitcoin blockchain when Alice wants to transfer one bitcoin to Bob.

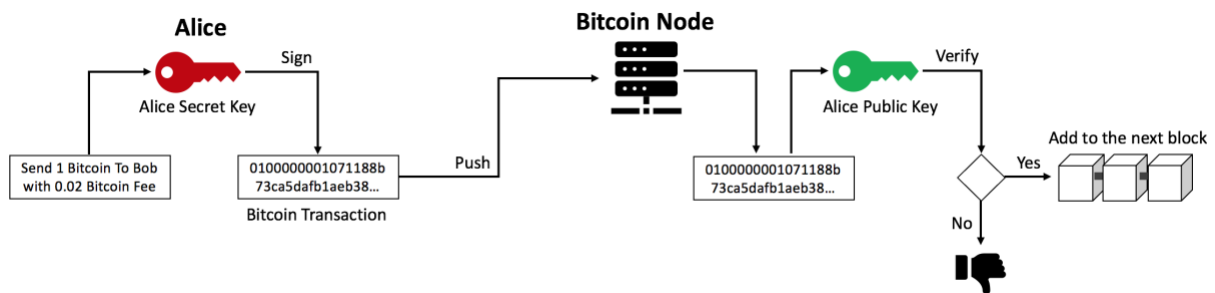


Figure 2-1: Simplified Example of Transaction Signing in Bitcoin

2.1.6 Elliptic-Curve Cryptography

Elliptic-curve cryptography (ECC) is a type of asymmetric cryptography based on the elliptic curve over finite field mathematics. The detail explanation of ECC is out of the scope of this document. However, we overview the basic concepts of ECC in this section.

2.1.6.1 Elliptic-Curve Domain Parameters for secp256k1

The elliptic curve equation is shown in Equation (1). It is a cubic operation in a finite field which means every integer result of multiplication, addition, subtraction, and division will get modulo number p where p is a prime number.

$$y^2 = x^3 + ax + b \pmod{p} \tag{1}$$

There are various ECC equations with different a and b parameters, but one of the popular ones used in many blockchains is secp256k1 defined in Standards for Efficient Cryptography (SEC) [6]. Figure 2-2 lists these parameters.

```
p = 2256 - 232 - 29 - 28 - 27 - 26 - 24 - 1
  = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC2F
a = 0
b = 7
G: xG = 0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798,
    yG = 0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8
n = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141
```

Figure 2-2: Elliptic-Curve Domain Parameters for secp256k1

As Figure 2-3 demonstrates the simplified graph of secp256k1, this is symmetric over x -coordinate and goes to infinity for positive and negative y -coordinate. For simplicity, this graph is drawn for the real number, while, the secp256k1 is defined for an integer finite field which is different and like a set of randomly scattered dots on a page.

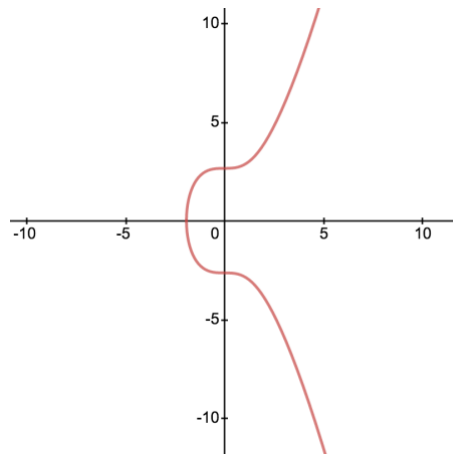


Figure 2-3: Elliptic-Curve Graph of $y^2 = x^3 + 7$

2.1.6.2 Elliptic-Curve Adding

Elliptic-Curve Cryptography defines an adding operation for EC-Points. To add two points $P(x_p, y_p)$ and $G(x_g, y_g)$ on the elliptic-curve graph in ECC, we should draw a line between the points. This line intersects with the elliptic-curve graph in the third point ($-R$). Then, we find the reflected point $R(x_r, y_r)$ of the third point over x-coordinate on the elliptic-curve graph. This point is the result of the addition as displayed in Figure 2-4.

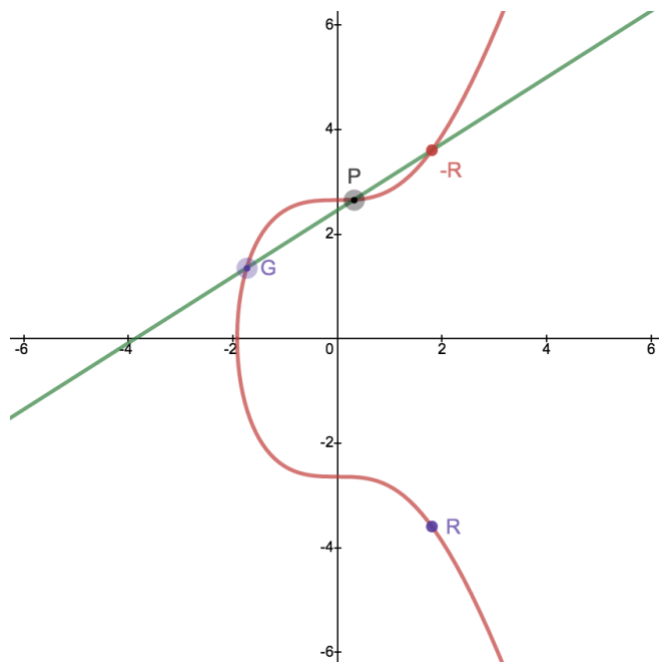


Figure 2-4: Addition of Two Points in ECC with Geometry Approach

Figure 2-5 lists the algebraic approach to add two points in ECC on finite field Z_p where Z_p is a set of integer numbers between 1 and $p-1$ that is achieved by modulo p . The term “modinv” means modular inverse defined by the Extended Euclidean algorithm [9].

```

s = (yg - yp) modinv (xg - xp) mod p
xr = (s2 - xp - xg) mod p
yr = (s * (xp - xr) - yp) mod p

```

Figure 2-5: Addition of Two Points in ECC with Algebraic Approach

2.1.6.3 Elliptic-Curve Doubling

Elliptic-Curve Cryptography also defines another primitive operation for EC-Points called doubling means put P equal to G in the addition equation. To find the double operation in geometry approach, we must draw a tangent line to the elliptic-curve graph at point P . Then, this line intersects with the elliptic-curve graph in the second point ($-R$). When we find the reflected point $R(x_r, y_r)$ of the second point over x -coordinate on the elliptic-curve graph, it is the result of doubling. Figure 2-6 demonstrates the double operation on Elliptic-Curve Cryptography.

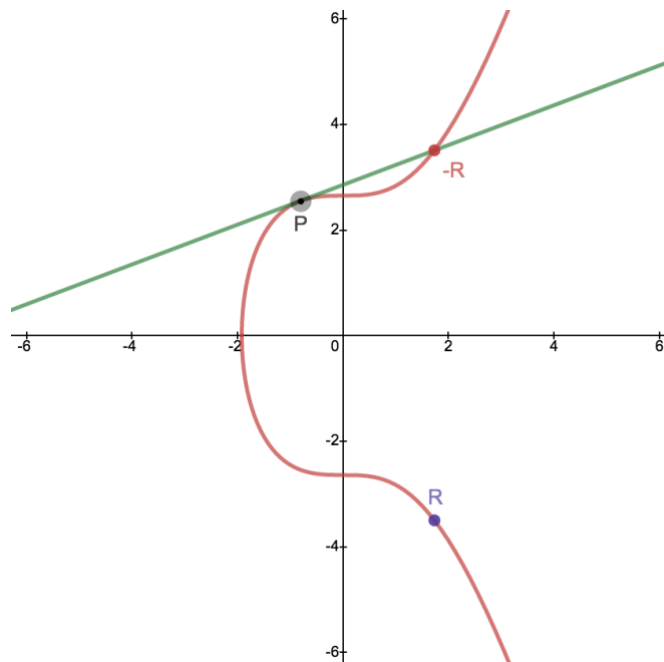


Figure 2-6: Doubling of a Point in ECC with Geometry Approach

Similar to the addition operation, Figure 2-7 shows the algebraic approach to compute the double of an EC-Point.

$$s = (3x_p^2) \text{ modinv } 2y_p \text{ mod } p$$

$$x_r = s^2 - 2x_p \text{ mod } p$$

$$y_r = (s * (x_p - x_r) - y_p) \text{ mod } p$$

Figure 2-7: Doubling of a Point in ECC with Algebraic Approach

2.1.6.4 Elliptic-Curve Multiplying

Elliptic-Curve Cryptography defines the point multiplication based on the point addition and the point doubling. This operation multiplies a point to a scalar value and finds the resulting point when we double the input point P in scalar times. However, there is a more efficient way to multiply called double-and-add, which is similar to multiply-and-square in modular exponentiation [9]. In this method, we scan the scalar “s” binary value from left to right and double the point P for each bit and add the point P to the result if the bit is 1. In the end, we have the result point R. Figure 2-8 lists the pseudocode of this algorithm.

```
ECMultiply (P, s) {
    if not 0 < s < n then error
    R := P
    for i=0 to len(s) {
        R := ECDouble(R)
        if s[i] == 1
            R := ECAdd(R, P)
    }
    return R
}
```

Figure 2-8: Multiplying of Two Points in ECC

In Figure 2-8 the “len(s)” is the length of scalar s in bits. The first line of the function checks that the scalar is in the correct range where can be from 1 to n defined in secp256k1 unless there will be a cycle in the infinite field.

2.1.6.5 Elliptic-Curve Key Generation

In the Elliptic-Curve Cryptography, the private key is a random scalar which is less than predefined fix value n . n is the upper-bound of possible private keys defined in secp256k1 and listed in Figure 2-2. The public key is a point calculated with Equation (2) from predefined Generation point G listed in Figure 2-2.

$$\text{publicKey} = \text{privateKey} * G \quad (2)$$

In the above equation, ‘*’ denotes ECC multiplication of point G and scalar privateKey . The result is an EC-Point with x and y value used as the public key. Because the discrete logarithm is a hard problem in computer science [9], calculating the private key from the public key is not practical while extracting the public key from the private key is trivial.

2.1.6.6 Elliptic-Curve Digital Signature Generation

Elliptic-Curve Cryptography has a special algorithm to generate the digital signature. Figure 2-9 displays the algorithm where the inputs are the hash of the message or transaction and the private key. The output is the generated signature. In this algorithm, G is Generation point, and n is the modulo according to secp251k1 defined in Figure 2-2. rand is a 256-bit random value, and messageHash is the calculated hash of the message or transaction. The resulting signature is the concatenation of r and s .

```
P(xp, yp) = rand * G
r = xp mod n
s = ((messageHash + r * privateKey) * rand modinv n) mod n
signature = (r, s)
```

Figure 2-9: Digital Signature Generation in ECC

Temporary point P is calculated by performing ECC-Multiplication of scalar rand and Generation point. Then, x coordinate of P is used to compute r and s value.

2.1.6.7 Elliptic-Curve Digital Signature Verification

The verification algorithm demonstrated in Figure 2-10 is used to verify a digital signature in Elliptic-Curve Cryptography. The inputs are the hash of the message or transaction, the generated signature, and the public key, and the output is true or false. In this algorithm, G is Generation point, and n is the modulo according to secp251k1 defined in Figure 2-2, and $messageHash$ is the calculated hash of the message or transaction. Temporary point Q is calculated by adding two results: the resulting point of multiplication of scalar u_1 and Generation point, and the resulting point of multiplication of scalar u_2 and the public key point. The final result is the comparison between the calculated value t and the extracted value r from the signature. If these two values are equal, the signature is valid.

```
w = s modinv n
u1 = (messageHash * w) mod n
u2 = (r * w) mod n
Q(xq, yq) = (u1 * G) + (u2 * publicKey)
t = xq mod n
if (r == t) signature is valid
```

Figure 2-10: Digital Signature Verification in ECC

2.2 Blockchain Technology

2.2.1 History

Blockchain idea is started from 1991 [10] by Stuart Haber and W. Scott Stornetta work to record timestamped digital documents and make it tamper-resistant by chaining them in a set of blocks. The authors of this project continued their work, and in 1992, they added Merkle Tree to group multiple documents' digest in each block [11]. No practical project used these studies. In 2004, Hal Finney introduced a hashcash-based token which uses Proof-of-Work and generates RSA-signed token and controls the ownership of them in a central trusted server [12].

The real blockchain emerged with the Bitcoin project since 2008 [13]. A person or a group known as Satoshi Nakamoto designed a distributed cryptocurrency that is not controlled by any central trusted server, government agency, or private company. Blockchain is an open distributed ledger that records all transactions increasingly in autonomous network nodes. In other words, the classic blockchain is a full-distributed database that its only operation is append and there is no update or delete operation.

In 2013, Vitalik Buterin crafted Ethereum [14] that upgrades the classic blockchain and creates a distributed virtual machine that records state-transitions as append-only records in the blockchain. Finally, the most advanced consensus algorithm for the blockchain to solve its classical problems was introduced by Daniel Larimer in 2014 [15] and implemented in EOS blockchain. It is a digital democracy to operate and govern the whole blockchain system.

2.2.2 Blockchain Mechanics

Blockchain is a chain of blocks which means that each block has the hash of the previous block. It makes a chain of blocks, and if someone wants to change one of them, he must change all next blocks too. A block usually contains a random number to make the change operation more difficult. This random number, aka nonce causes to generate a particular hash value that is less than a predefined number. In other words, the hash value must have some zeros at the beginning. The blockchain network updates this predefined number periodically to adjust the difficulty of the system and balance the required computing power and the hacking risk [16].

Each block contains the hashes of the transactions. A transaction is a message digitally signed by the sender and includes some information like transferring fund from someone to others or calling a function in a blockchain virtual machine. Figure 2-11 demonstrates a sample

chain of blocks. A particular hash tree called Merkle Tree calculates the hashes of transactions to insert in a block. Using Merkle Tree, there is no need to record all hash value of transactions in a block which saves storage, bandwidth, and computation power. Figure 2-11 displays a Merkel Tree in the blockchain.

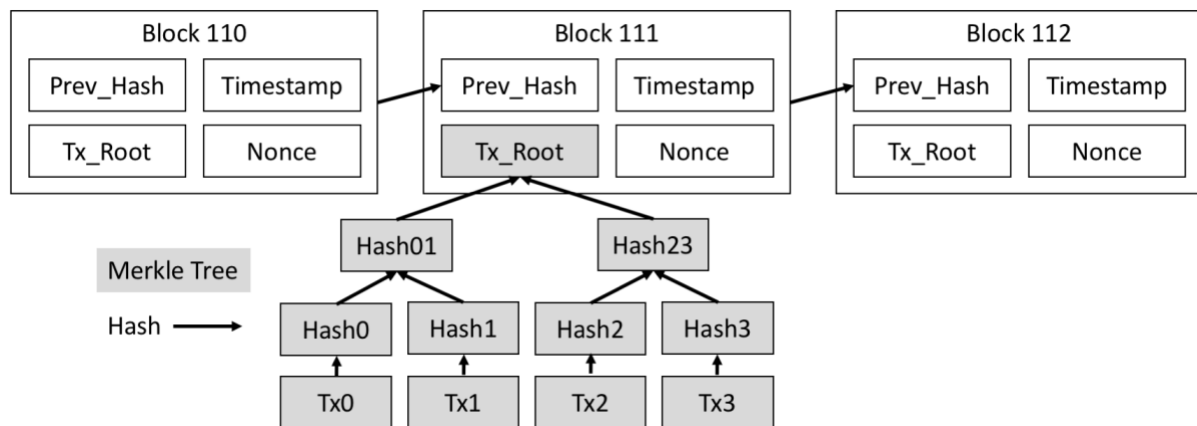


Figure 2-11: Chain of blocks in the blockchain

In Figure 2-11, Prev_Hash is the hash of the previous block, Tx_Root is the root of the Merkle Tree, and Tx indicates the transaction. Each block also has a timestamp, the time that the block is mined. The first block of a chain called genesis block where its previous hash value is set to 0.

There are a few types of entities which are a little bit different in various blockchains. The first entity is the user that has funds and make a transaction to transfer her fund to other users or entities. The second entity is the full node or verifier that checks the transaction and records that in the ledger. These nodes are connected in a peer-to-peer network. The third entity is miner that aggregate transactions, find the nonce, and generate blocks to append that to the blockchain. Miners compete to find the matched nonce and produce the next block and get rewarded for that with the blockchain cryptocurrency. It is the point that generates a new fund called coin in a blockchain. A miner also charges the sender of a transaction a small amount of money called transaction fee to put the transaction in its next block. If the sender pays more

fee, her transaction will be inserted in the next block sooner because the miners would like to add the transactions that pay more transaction fee.

Since the hash function is one-way, there is no way to find the nonce from the desired hash value. So, a miner has to generate many random nonces and compute the hash of the entire block to check the final result. Thus, each time a miner is lucky and finds the nonce, it propagates the result to the blockchain nodes and gets the reward. Besides, two miners may find a nonce to solve this puzzle at the same time and publish their result to the nodes. Two different sets of nodes could accept both messages because the blockchain network is peer-to-peer, and there is not any central node to manage the system. Therefore, they could accept the new block and append it to their ledger. After some time, there will be two different chains of blocks called temporary fork. To tackle this problem, when a node receives two different forks, it accepts the longest fork and rejects the shorter one. Therefore, in the mean-time, there are two chains of the block, but after some time, there will be only one chain in the ledger.

2.2.3 UTXO-based versus Account-based Blockchain

There are two transaction models in the blockchains. For example, Bitcoin uses the Unspent Transaction Output (UTXO) model, and Ethereum uses the Account model. In the UTXO model, each transaction has some inputs and outputs. Each input indicates one previous transaction output and the amount of that. The new transaction spends all amount of the previous transactions input and put them in its outputs. The next transaction does the same. If an output of a transaction is used as an input with another transaction, it is called "spent output". While, if an output of a transaction is never used by another transaction, it is called "unspent output". The blockchain nodes only accept a transaction that its inputs are not spent, if so, it is double-spending that is not permitted in the blockchain. Hence, this model is called the Unspent

Transaction Output (UTXO) model. It is very similar to cash transferring in the real world where the coins move from one person to another.

On the other hand, there is another model called Account-based. In this model used in Ethereum, each account has a balance, and each transaction indicates fund transfer from one account to another. It is similar to the bank account in the real world.

2.2.4 Smart Contract

The new generation of blockchain technology employs the idea of the chain of the blocks and extends that to building a distributed virtual machine. The most popular instance of this type of blockchain is Ethereum. In this scheme, each node has a Turing-complete virtual machine that gets a piece of code and executes that until obtaining the result or reaching to a limit. This piece of code is called Smart Contract. All nodes of the blockchain run the code by receiving a trigger message, and compute the same result, then record that in the blockchain. A user or another smart contract can call a function of a smart contract by sending a transaction. The nodes receive a transaction fee (e.g., gas in Ethereum) to add a smart contract result to the blockchain. So, each block includes the state of all smart contracts and appends that to the ledger. The smart contract is a special part of code should run on a trusted computer while does not have any user interface and called as a library in other programs.

2.2.5 Consensus Mechanisms

Blockchain includes an algorithm called consensus mechanism to achieve the shared data or state between all nodes. This mechanism is distributed along with autonomous nodes which run the consensus protocol independently. At the end of this process, all nodes must agree on the same data value or state with no central authority to control them. There are

various types of consensus mechanisms, and we overview the most popular ones here. Consensus mechanism defends against hackers or malfunction nodes and miners would like to cheat, for example, by making a fake fork for double-spending. With this algorithm, nobody can attack the network until having 51 percent of the nodes. It is because if a hacker controls 51 percent of the nodes, he can make a new fork of the blockchain and force the entire network to accept that by creating the longest chain. It is called 51 percent attack in the blockchain terminology.

2.2.5.1 Proof-of-Work

Proof-of-Work (PoW) is a widespread consensus mechanism used by Bitcoin, Ethereum, and many other blockchains. In PoW, a miner solves a puzzle that requires significant computational resource. For example, the miner should find a random number called nonce included in a block that creates a particular hash value of the block, which is less than a predefined number. To find the nonce, the miner has to generate many random numbers and calculate the hash of the whole block. This process requires a lot of computation or “work”. So, if a miner presents a correct nonce, this is proof that it has performed significant computation work. The main disadvantage of the PoW is high energy consumption to find the nonce for the next block. All attended miners consume a lot of energy simultaneously, and only one of them wins and gets the reward while others wasted their power. Also, PoW is computation, and it is possible to make the Application-Specific Integrated Circuit (ASIC) board speed up the process. So, somebody with a big server farm and a lot of ASIC boards can aggregate a massive hash power to make 51 percent attack [16].

2.2.5.2 *Proof-of-Stake*

The Proof-of-Stake (PoS) is proposed to solve the problems of proof-of-work. In PoS, there is no need to compute a magic nonce and consume a lot of energy for computation. In contrast, the validators replace the miners. Each validator has a deposit of cryptocurrency as a stake to participate in the block creation. Validators with bigger stakes have more chance to select for creation of the next block. The blockchain network uses a randomized protocol to choose the next validator. It prevents selecting the validator with a massive capital for all blocks creation and gets the control of the whole system. To perform the 51 percent attack in PoS blockchain, the attacker has to own 51 percent of the entire cryptocurrency.

The main issue of the PoS is nothing-at-stake problem that occurs when two forks are created. Because the validator has no cost to create a block, it validates both forks to choose one of them in the future, which has more rewards. Since the validator does not lose anything in both cases, it could bet on them. A validator has enough motivation to make this attack because when it chooses one fork and spends time to validate its blocks if a longer fork emerges and invalidates this fork, the validator losses its time and rewards.

2.2.5.3 *Delegated Proof-of-Stake*

One of the recent new consensus mechanisms is the Delegated Proof-of-Stake (DPoS). In DPoS, the owners of the stakes are all users who have coins of the cryptocurrency, and they vote to elect a limited number of delegates from a set of candidates to validate next blocks. The election repeats in a period, and the users vote again to change the delegates. If a delegate cheats, in the first step, other delegates could vote against him, and in the second step, the users remove him from delegates by do not vote for him. Usually, each delegate deposits a number of coins to escrow in case of malicious behavior. DPoS is a form of digital democracy that all

users participate in network decisions. New blockchains like EOS and TRON employ DPoS consensus mechanism.

DPoS is very faster than PoW because there are no computational or other types of puzzle to solve. It does not waste much energy too. This algorithm is considered as the most trusted consensus mechanism. The disadvantages of DPoS is the classic challenges of the real-life election. For example, if the number of delegates is small, the possibility of attack increases. Also, the users with fewer coins may do not participate in the election because they think their vote in comparison to who has a high volume of coins has no impact.

2.2.6 Blockchain Networks

Another taxonomy of blockchain network is according to their accessibility a is explained in this section.

2.2.6.1 *Public Blockchain*

In public blockchain, everyone with an Internet connection can join the network to own coins or tokens, send transaction and become a node, miner, validator, etc. In other words, there is no restriction to use a public blockchain. The two most prominent public blockchains as market cap are Bitcoin and Ethereum. The public blockchain is reliable and trustable because its community is extensive and includes entities with various interest, and no one can control the entire system strategy or technology and can be fully distributed.

2.2.6.2 *Private Blockchain*

A private blockchain is restricted from the public community. To join a private blockchain as a user, miner, node, etc. a privilege from the administer is required. A common

use-case of the private blockchain is a group of companies that would like to create a blockchain to use in their products and services, but they do not want to open it to public users to engage. An example of the private blockchain is Hyperledger Fabric. The private blockchain has lower trustworthiness because one or a group of companies with common interests control the system, and it is not fully distributed.

2.3 Crypto Wallet

Crypto wallet is a term that is used for cryptographic applications that manage secret keys, addresses and seed for a user in blockchain and cryptocurrency. This program can be on an online server, laptop, smartphone, and even particular hardware. It would securely generate and store the keys and provide some mechanism to back up and restore them.

2.3.1 Wallet Types

2.3.1.1 *Brain Wallet*

The brain wallet is the simplest one. The user chooses a passphrase, and all secret keys and addresses are derived from that. So, the user does not need to maintain a paper, software, or device as a wallet. Each time he needs to make a transaction, he enters the passphrase into a wallet program and constructs the secret keys. After signing a transaction, the wallet program removes the passphrase and all constructed keys from memory. The brain wallet has significant drawbacks. Firstly, if the user forgets the passphrase, he loses all funds. Secondly, malware in the wallet program can sniff the passphrase and steal his funds.

2.3.1.2 Paper Wallet

Another popular, still simple wallet is the paper wallet. A paper wallet usually is one-page paper where the secret key and public key or address are printed in the QR-Code format. There are some online and offline web-pages [17] that generate secret keys and prepare paper wallet to print. Figure 2-12 shows a sample paper wallet. The user is not required to remember a passphrase, although she can add a passphrase to encrypt the secret key for better security. Each time the user needs to make a transaction, she uses the QR-Code on the printed paper wallet to recover the secret key and sign the transaction. A paper wallet is easy to backup just by coping a paper.



Figure 2-12: Sample Bitcoin Brain Wallet

The paper wallet has the same issue with the brain wallet. Because it needs a third-party wallet program to make a transaction, malware can steal the secret key, even if the user adds a passphrase to that. Also, if a hacker finds a paper wallet, he can recover the secret key unless the user uses a complex long passphrase, and it increases the chance to forget.

2.3.1.3 Hot Wallet

One of the most popular wallets is the hot wallet (e.g., Coinbase wallet [18]) where the user stores the keys on an online cloud server like exchanges protected with a password or two-

factor authentication. It is convenient and accessible everywhere on the desktop, laptop, and smartphone, but if hackers exploit a cloud server, all users' keys will be compromised. It occurs many times in the real world [19][20] because these servers are a honeypot for hackers.

2.3.1.4 Cold Wallet

There is another secure alternative; cold wallet also called offline wallet which is a wallet program installed on an offline air-gapped device (laptop, smartphone, and even Raspberry Pi) 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 [21] transfers the secret keys via ultrasound from an offline wallet to an adjacent online computer.

2.3.1.5 Desktop Wallet

Another option is the desktop wallet that stores the secret keys on a desktop or laptop computer. Desktop wallets usually require a passphrase from the user and encrypt the keys on the computer storage. The first implementation of bitcoin client also called Satoshi Client [58], and Bitcoin reference client is an instance on the desktop wallet.

This type of wallet is popular too, but it is vulnerable to virus and hacks [22]. Because desktop and laptop are a general-purpose computer, a hacker can install malware like key-logger and trojan-horse to capture the user passphrase or copy the secret key.

2.3.1.6 Mobile Wallet

The mobile wallet is a mobile application installed on a smartphone. There are many mobile wallets in Apple AppStore for iPhone and Google Play for Android. These wallets

usually store the secret key on the smartphone and not on an online server. The wallet app is protected by smartphone lock mechanisms such as password, fingerprint, and facial-recognition. Besides, there are new mobile wallets that use security features of smartphones such as ARM TrustZone [23][24]. TrustZone is a hardware-based Trusted Execution Environment (TEE) and is available in many smartphones. The advanced mobile wallets use TEE to store the secret key securely and execute critical operations such as transaction signing in a trusted environment.

The mobile wallet has its disadvantages. Even though it is suitable for daily and online purchases, it is not appropriate to store a large amount of money when it can be lost or stolen. Also, the existing mobile wallets do not have a secure backup solution, and most of them use online solutions or paper backup.

2.3.1.7 Hardware Wallet

The most secure existing wallet is the hardware wallet, which is a dedicated cryptography device to generate and store secret keys and sign transactions, and authors of [25] introduced the early functional version of that. This type of wallet usually is a USB stick, Bluetooth device, or smart card with special embedded software to do cryptography functions. Because a hardware wallet is not a general-purpose computer like desktop, laptop, and smartphone, a hacker cannot install a malware program easily. Also, most of the hardware wallets have a special chip called Secure Element module (SE) as a cryptographic co-processor to perform cryptography operations like key generation and transaction signing fast and secure.

There are various forms of hardware wallets from a USB dongle [26] to a full tablet [27]. 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 [2]. In this situation, the user enters the password on a general-purpose computer (terminal) and gets the wallet responses on the computer screen. Therefore, the malware program can steal the password or displays mislead information on the screen.

In addition, a secure hardware wallet must have a Secure Element module to protect the secrets from electrical and physical attacks such as side-channel attacks and power-analysis. The authors of [28] demonstrate a series of successful attacks to the most popular hardware wallets that do not have Secure Element or are not well-designed.

2.3.2 Hierarchical Deterministic Wallet

Bitcoin, Ethereum, Litecoin, and almost all popular cryptocurrencies use elliptic-curve cryptography (ECC) to sign and verify transactions. Therefore, the user has a key pair and uses the private (secret) 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 and the blockchain network 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 remaining 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 [16]. 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 [29]. In HD wallet, the user has a tree of private keys which any node is derived from its parent using a deterministic algorithm. The root of this tree is a private key which is called ‘master private key’ and derived from a 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.

2.3.2.1 BIP-32: Hierarchical Deterministic Wallets

BIP-32 is a Bitcoin Improvement Proposal that defines Hierarchical Deterministic Wallet (HD Wallet) [29]. This document explains different algorithms to derive each node from its parent in the key tree. The core of this document is one master key generation, and two Child Key Derivation (CKD) functions. The master key generation function is as follows where S is 128 to a 512-bit random value called master seed, H is a 512-bit hash value, and H_L is left 256 bit, and H_R is right 256 bit of H . k_m as master private key and c_m as master chain code are 256-bit outputs.

```
MKG(S) => (km, cm):  
H := HMAC-SHA512(Key = "Bitcoin seed", Data = S)  
km := HL  
cm := HR
```

Figure 2-13: BIP-32 master key generation function

The first derivation function, CKD_{priv} converts the private parent key to private child key. These keys are extended, which means each key has an additional 256-bit random number called the chain code to prevent solely depending on parent key. This function gets k_{par} as

private parent key, c_{par} as parent chain code and i as index and computes k_i as private child key and c_i as child chain code. k and c are 256-bit, and i is 32-bit value. H is a 512-bit. The CKD_{priv} conversion function is as follows. The HD wallet constructs all private keys from the master private key using CKD_{priv} function,

```

CKDpriv((kpar, cpar), i) => (ki, ci):
if i ≥ 232 then
    H := HMAC-SHA512(Key = cpar, Data = 0x00 || kpar || i)
else
    H := HMAC-SHA512(Key = cpar, Data = point(kpar) || i)
ki := HL + kpar mod n
ci := HR

```

Figure 2-14: BIP-32 private parent key to private child key function

Another derivation function is CKD_{pub} that converts public parent key to public child key. This function gets K_{par} as parent public key, c_{par} as parent chain code and i as an index and computes K_i as public child key and c_i as child chain code. The HD wallet using CKD_{pub} constructs all public keys from the master extended public key.

```

CKDpub((Kpar, cpar), i) => (Ki, ci):
if i ≥ 232 then
    error
else
    H := HMAC-SHA512(Key = cpar, Data = Kpar || i)
Ki := point(HL) + Kpar
ci := HR

```

Figure 2-15: BIP-32 public parent key to public child key function

Because if the index (i) be more than 2^{32} , it is impossible to compute the public child key from the public parent key, the private keys for index bigger than 2^{32} called hardened keys and are more secure than normal keys for index lesser than 2^{32} .

Consequently, the HD wallet computes the master key from the master seed and derive child keys from the master key. 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_{priv} algorithm in the corresponding round. Then hardened key index is indicated by ' character. The format of this path is as follows:

$$\text{path} = m/i/\dots$$

For example, the path “m/1/3” means running $CKD_{priv}((k_m, c_m), 1)$ to compute $(k_{m/1}, c_{m/1})$ and $CKD_{priv}((k_{m/1}, c_{m/1}), 3)$ to compute $(k_{m/1/3}, c_{m/1/3})$.

2.3.2.2 BIP-39: Mnemonic code for generating deterministic keys

The BIP-39 document describes a method to build a set of rememberable words to generate the master seed [30]. These words are readable and easy to remember for humans. The algorithm is as follows. At first, a random number is generated as entropy that is multiple of 32 bit and is from 128 to 256-bit where longer entropy means more security and more words.

Then, the checksum of the entropy is calculated with first entropy length divided by 32 bits of the SHA256 hash of entropy. This checksum is appended to the end of the entropy. Then, the result splits into 11-bit groups which each of them is a number between 0 and 2047 ($2^{11}-1$). Each number is used as an index in a predefined fixed word list, and all numbers make a 12 to 24 set of words called a “mnemonic sentence”. The following table demonstrates different word list lengths.

Table 2-1: Relation between Entropy and Mnemonic Sentence

Entropy Length (bits)	Checksum Length = Entropy Length / 32 (bits)	Entropy + Checksum Length (bits)	Mnemonic Sentence Length = (Entropy + Checksum Length) / 11 (words)
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

To convert a mnemonic sentence to the master seed, BIP-39 uses PBKDF2 function [52]. This function gets pseudorandom function, password, salt, number of iterations, and desired length of the derived key to generate a key from a password. BIP-39 feeds the PBKDF2 function by HMAC-SHA256 function as pseudorandom function, the mnemonic sentence as the password, string “mnemonic” plus passphrase as salt, 2048 as the number of iterations and 512 bits as output derived key length. Therefore, the output of this function always is 512-bit pseudorandom master seed. In this function, the passphrase is an additional password chosen by the user to protect the master seed. If the user uses a passphrase, a hacker cannot recover her master seed only by knowing the mnemonic sentence.

2.3.2.3 BIP-44: Multi-Account Hierarchy for Deterministic Wallets

In addition to HD wallet base algorithms, the cryptocurrency community proposed a complementary standard BIP-44 [31] to define a universal path format for all coins (Bitcoin, Ethereum, Litecoin, and other coins), because BIP-32 only defines the derivation function and path. The BIP-44 path addresses all coins in an HD wallet with only one single master seed. It makes key management and backup process easy. A path in BIP-44 format has following levels in BIP-32:

$$\text{path} = \text{m}/\text{purpose}'/\text{coin}'/\text{account}'/\text{change}/\text{address_index}$$

In the above path, m is the master seed, the purpose is the fixed number 44 for BIP-44, the coin is a predefined value for registered coins, for example, 0 for Bitcoin and 60 for Ether. The account is a group of funds and helps the user to manage her money, for example, to create a separate set for the spending account and the saving account. Change is 0 for external address and 1 for internal address. The external address is a regular address that published to others to receive funds while an internal address is “change address” that is for receiving remained funds

from spending transaction and never published to others. Address_index is a sequential number from 0 to generate multiple unique addresses.

BIP-44 document defines a process called Account Discovery to explore all used addresses in an HD wallet and finds the user funds in the blockchain. The HD wallet traverses all nodes of the key tree from the root that is the master seed and searches for a transaction with the derived address as input or output in the blockchain. The exploring process is infinite because the BIP-32 key tree definition is unbound. To limit the exploring process, BIP-44 define an algorithm for discovery as follows:

1. Start from m/44'/coin' key and run this algorithm for all supported coins.
2. Set account to 0.
3. Set change to 0.
4. Derive the external chain node.
5. Search for external addresses. To do that, set address_index to 0 while increment it one-by-one, derive addresses and search that them in the blockchain. If a transaction found, set change to 1 and search for the internal chain node too. If there is no used external address for 20 continues indexes, stop the search.
6. Increment the account value and repeat from step 3 until the account node does not have any used address.

The HD wallet should not let the user create a new account when the previous one does not have any transaction in the blockchain; otherwise, the above algorithm does not work. Searching for external chain nodes is enough because an internal chain node receives fund if and only if the corresponding external node is used. Table 2-2 demonstrates a step-by-step sample running of the account discovery process in an HD wallet that supports Bitcoin and Litecoin. This sample wallet has only two used Bitcoin addresses in the first account and no used Litecoin address.

Table 2-2: Account Discovery Process in a Sample HD Wallet

coin	account	chain	address_index	Path
Bitcoin	first	external	first	m/44'/0'/0'/0/0
Bitcoin	first	internal	first	m/44'/0'/0'/1/0
Bitcoin	first	external	second	m/44'/0'/0'/0/1
Bitcoin	first	internal	second	m/44'/0'/0'/1/1
Bitcoin	first	external	no transaction for 20 addresses	m/44'/0'/0'/0/2-21
Bitcoin	second	external	no transaction for 20 addresses	m/44'/0'/1'/0/0-19
Litecoin	first	external	no transaction for 20 addresses	m/44'/2'/0'/0/0-19

2.4 Smart Card

As discussed before, in a blockchain application, each user has a unique asymmetric key pair (public and private keys) to digitally sign a transaction. Usually, the user must have a security device such as a personal smart card which stores her private key, and she can access it by entering her PIN. In a simple scenario, when the user approves the information of a transaction, her terminal (such as a desktop computer, laptop or smartphone) computes the hash of the transaction and sends it to her smart card. Then the user enters her PIN, and the smart card generates a secure digital signature using her private key. The smart card is a tamper-resistant cryptography device that stores private keys and performs cryptographic operations like signing. Usually, the smart card is used as a term for a plastic card where has an IC chip to execute a cryptography program in a secure execution environment [32].

2.4.1 IC Card Components

The smart card is an IC card that has computing capabilities and is different from the magnetic-stripe card and memory card. These two different cards are shown in Figure 2-16.

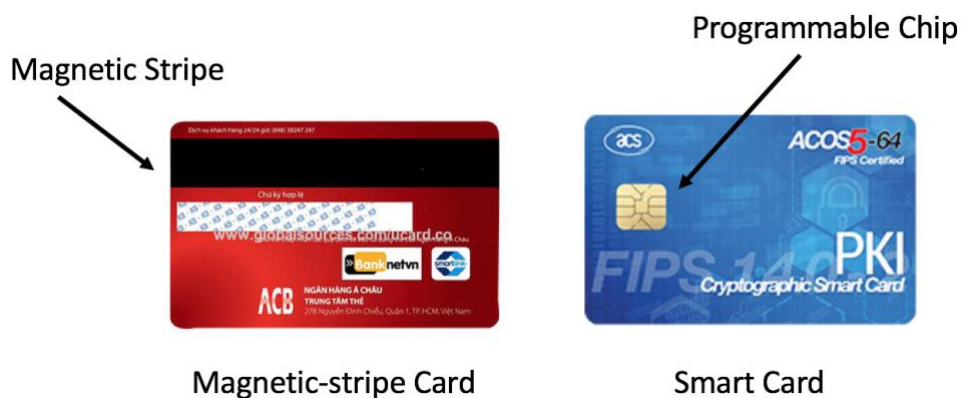


Figure 2-16: Smart card vs. magnetic-stripe card

The magnetic-stripe card like many credit cards stores a small amount of data, for example, cardholder name, credit card number, etc. in a magnetic stripe memory. They do not have any computing capabilities, and they are easy to clone. In contrast, smart cards have computing features and usually are programmable. The card issuer like banks and government can store card holder's private data on the secure memory of a smart card and perform cryptographic operations like encryption and digital signature in a secure way on smart card chip. Because of that, new credit cards are based on smart cards for better security, and several national ID cards are issued on smart cards. The components of a classic smart card are explained in the next paragraphs.

A regular smart card has a programmable chip (IC), which is a tiny computer to execute limited programs in a secure execution environment. This chip has a small processor (CPU), a memory about 1 to 3 kilobytes (RAM), storage between 32 to 256 kilobytes (e.g., EEPROM) and a byte-stream input/output, but usually does not have internal clock and battery. Figure 2-17 demonstrates a sample layout of a smart card chip [32].

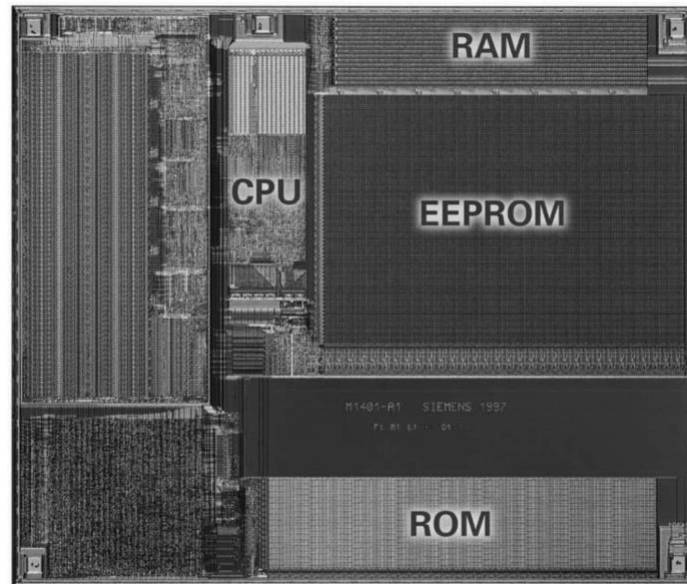


Figure 2-17: Sample smart card chip layout

Smart cards have two popular interfaces, including contact and contactless. A terminal must have, for example, a USB smart card reader to connect to the contact interface compatible with ISO/IEC 7816 part 1 to 3. Smart cards also support contactless interface according to ISO-IEC 14443 and recently Near Field Communication (NFC). Consequently, smartphones which have NFC antenna do not need any additional card reader and connect to the smart card with NFC interface. Some new smart cards also support Bluetooth Low Energy (BLE) interface but requires a battery.

2.4.2 Java Card Technology

There are few solutions to develop a program which is call Card Application or Card Applet for a smart card. The most popular one is the Java Card Technology [33]. Smart cards that support Java Card Technology have a limited version of Java Virtual Machine called Java Card Runtime Environment (JCRE). Figure 2-18 shows the architecture of JCRE in the smart card [34].

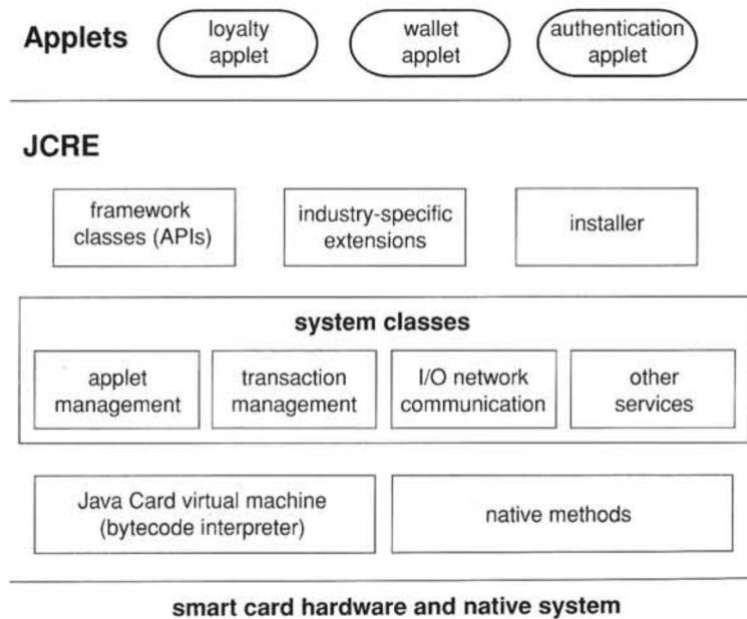


Figure 2-18: Java Card Runtime Environment (JCRE) Architecture

JCRE in a smart card is similar to an Operating System in computers. It runs multiple programs (Applets), manages memory allocation, provides system functions as APIs, etc. However, there are fundamental differences between computer and smart card. For example, a smart card cannot perform multi-tasking and runs only one program at a time. A card manufacturer could add extra packages and functions to his JCRE for additional features, for example as GSM library for the mobile network. Furthermore, JCRE has a pre-load specific applet to manage the loading and removing other applets called applet manager.

Today, Oracle is the owner of Java Card Technology and publishes the Java Card Platform Specification to define Java Card API and features. The last version of Java Card Platform Specification is 3.0.5, and popular versions are 2.2.2, 3.0.1 and 3.0.4. In our projects, we use an open-source command tool called “ant-javacard”. This tool is an Ant task and uses Oracle Java Development Kit (JDK) to compile a Java code to bytecode (class file) and convert that to card application file (CAP file).

2.4.3 Global Platform

Global Platform (GP) is a consortium of several smart card companies which defines a series of standards to manage the application on smart cards [35]. For instance, they define administration commands, key management, and applet life cycle for all smart cards, including Java Cards. The most popular version of Global Platform specification is 2.1.1. In our work, we use GlobalPlatformPro [36], an open-source command-line program to load and delete applet to/from a smart card.

2.4.4 Smart Card Programming

To program a smart card, a developer writes his program in a limited version of Java language and compiles that to Java bytecode with Java Card compiler. Then, he uses Java card tools to convert the compiled code to a Card Application Package (CAP) and load it to a real smart card chip. Finally, he can send his defined commands in the form of byte-arrays to the card and receive their responses in byte-arrays too. Figure 2-19 demonstrates the whole process of compiling and loading of card application [34].

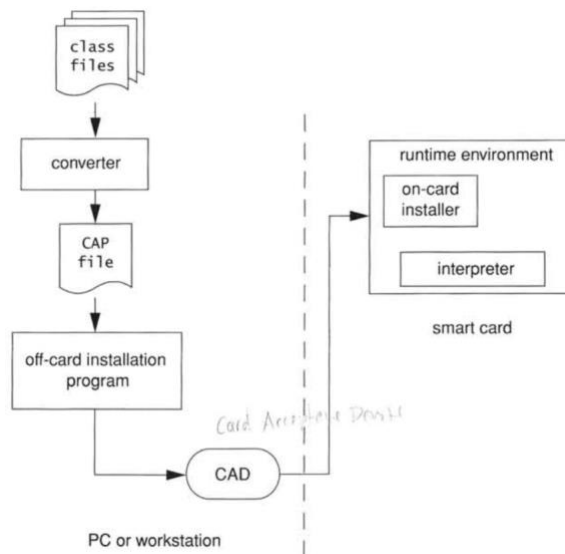


Figure 2-19: Java card application compiling and loading process

The input and output of card applications are in byte-stream form. These byte-streams are in a particular format defined in ISO/IEC 7816-4 called Application Protocol Data Unit (APDU). There are two types of APDUs; command APDU or C-APDU and response APDU or R-APDU. Command and response APDUs have the following structures [34].

Command APDU structure

Mandatory header				Optional body		
CLA	INS	P1	P2	Lc	Data field	Le

Response APDU structure

Optional body	Mandatory Trailer	
Data field	SW1	SW2

Figure 2-20: Command and response APDU structure

The detail description of each APDU fields are explained in ISO/IEC 7816-4 [37], and we discuss only the related parts here. The CLA field is one byte to indicate class field which is 0x00 usually. The INS field is a one-byte instruction code determined by the card application. P1 and P2 are two bytes parameters for the instruction. Lc is one to three bytes to indicate the length of the data field. Data field conveys byte-stream data with flexible length, and Le is the length of expected response byte-stream.

In response APDU, Data field is response data returned by the card application. SW1 and SW2 are two bytes status words which indicate the error, warning, or successful result return by the card application.

2.4.5 Smart Card Simulation

A big challenge in smart card programming is the simulation. To develop a program, a programmer needs a set of tools to write the code and test that with tracing and debugging line-by-line. Unfortunately, the smart card does not have these features and cannot run a card

application in debug mode. The programmer needs a simulator to run the code on a computer before loading that to a real smart card. There are few tools to simulate a Java card, including Java Card Reference Implementation by Oracle [33]; however, this tool has significant limitations. So, in our work, we chose an open-source tool called jCardSim [38]. It is a regular Java package that includes command definitions and API of Java card application. The programmer writes his Java card code in Java and debugs it in regular Java Virtual Machine using jCardSim packages. If everything works, he compiles and loads his code to a real smart card.

CHAPTER 3: FINDINGS

3.1 Smart Card Security

The smart card is a mature technology to build a hardware crypto wallet. It has a tamper-resistant chip and usually has passed hardware security evaluations in a cryptographic module lab [40][41]. 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 proposed schemes, we implement them on the smart card as a proof-of-concept to prove that a hardware wallet with limited resources could use our designs. So, at first, we consider the security of the smart card as a platform for the hardware wallet.

3.1.1 Threat Model

The authors of [39] proposed a reference threat model for smart card. They identify various parties in a smart card system including cardholder, data owner, terminal, card manufacturer and software manufacturer. In hardware crypto wallet, we can map these parties as follows. Cardholder is the owner of the private keys that signs the transaction and owns the coins. Data owner is same as cardholder because there is no additional data like personal information and photo on a crypto wallet. Terminal is usually a desktop computer, laptop, smartphone and etc. Card manufacturer is the company that produce the physical card including the programmable chip, NFC antenna and etc. Finally, software manufacturer is the company that provides the card application for crypto wallet.

In our work [2], we focus on the user or cardholder point of view and consider the security threats from this angle. So, we assume that the cardholder is the trusted party. We also assume the card manufacturer is trusted because the smart card has passed the hard security

checks in an evaluation lab [40][41]. In addition, we assume that the software company that provides the hardware wallet is trusted, while it can be a subject for considering in another research. So, we focus on terminal party that can be compromised by a hacker, because the least secure part of the system is the terminal. In the rest of this document, we assume that the terminal is not secure at all.

3.1.2 Fundamental Vulnerabilities

Authors of [39] claim that the existing smart cards have fundamental vulnerabilities because they do not have direct interface with the user and use the terminal input/output devices like display and keyboard to show the messages and get the commands. With respect to this threat model, there are two distinguished attacks that we inspected in our work [2] and are discussed next. These attacks are Man-In-The-Middle attack and change the terminal software parts to misguide the digital signature process.

3.1.2.1 *Capturing the Smart Card PIN*

A smart card receives its password aka Personal Identification Number (PIN) to gain access to the keys on the card. The main security challenge is that a smart card doesn't have a direct input device and must use the terminal's keyboard, mouse etc. to get the PIN from the user. In this situation, an attacker can compromise the terminal and install a key logger or another malware to capture the PIN. Then, the attacker can use this PIN to authenticate himself to the card without the user's authorization. So, for example, he can sign a transaction and transfer fund from the crypto wallet without the user interaction.

3.1.2.2 *Altering the Digital Signature*

The major usage of the smart card is digital signature, for example, for signing a transaction in the blockchain. The regular digital signature mechanism is as follows: a user sees the information of a transaction like the amount, the receiver address and etc. in an online website, computer or smartphone application and if she approves it, she signs it using her hardware crypto wallet, in this case, her smart card. In this process, a cryptographic library, as part of the application, computes the hash of the transaction and sends this hash value to the smart card for signing. The challenge is that a malware can change the hash value just before transmitting it to the smart card, resulting in the user signing an unwanted transaction with her private key. For instance, a hacker can change the receiver address and transfer the coins to his address.

3.1.3 Implementation of Smart Card Attacks

To measure the possibility of these attacks in practice, we designed an attack scenario and implement these attacks on a pervasive smart card. We implemented the mentioned attacks on Windows, but they are applicable on other operating systems, too. The attack code is called MinidriverSpy. We used Personal Identity Verification (PIV) card [42] in our attacks. PIV is a smart card standard which is supported with built-in drivers from Windows 7 SP1, from OpenSC 0.11.1 (in Linux), and from Mac OS Sierra 10.12. So, it is a mature technology that all operating system use that as Two-Factor Authentication. In addition, the PIV cards are used in many government organizations.

Microsoft Windows uses a software stack to communicate with smart card and conduct cryptography operations, and its important module is minidriver [43]. Windows has a built-in minidriver for PIV smart card which is MSCLMD.DLL. We implemented a spyware

“MinidriverSpy” as a hooking DLL and replaced MSCLMD.DLL. Right side diagram in Figure 3-1 shows our change on attack. The only permission we need to do this action is file copy permission.

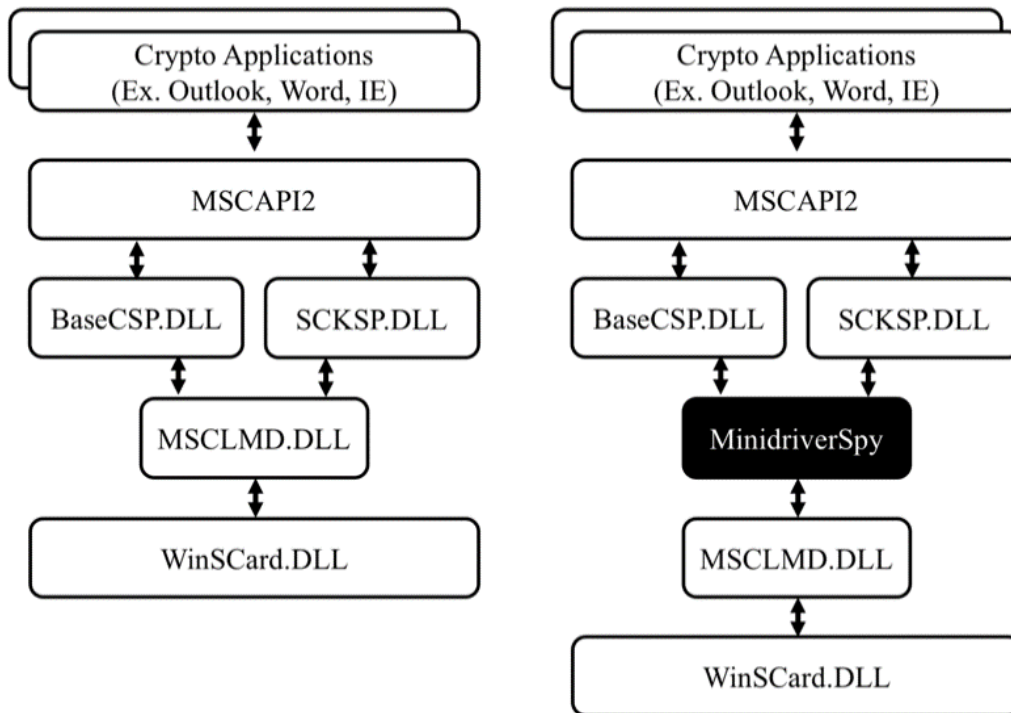


Figure 3-1: Windows smart card software stack vs. hacked software stack

Original minidriver (MSCLMD.DLL) has only one entry point “CardAcquireContext”. This function returns a set of function pointers of smart card minidriver. We added “CardAcquireContext” in our MinidriverSpy and pass these pointers from original minidriver to the caller, with some changes to implement our attacks. To sniff the smart card PIN, MinidriverSpy alters pointer of “CardAuthenticateEx” function and copy this PIN value before sending it to the original minidriver. To alter digital signature, MinidriverSpy modifies pointer of “CardSignData” function to change hash value just before sending it to the smart card, and with this attacking tool, a user will be tricked to sign a fake data using her private key on the smart card.

To test our attack, we used two types of smart card including embedded smart card in USB token and traditional ID-1 sized smart card (credit card size) with USB card reader. We tested our MinidriverSpy successfully on YubiKey 4, PIVKey T600 USB Tokens and PIVKey C910 PKI Smart Card on Windows 7 Service Pack 1 64-bit and Windows 10 64-bit. We published the essential parts of MinidriverSpy as open-source program at GitHub [44].

Therefore, we proof that the threat model is true, and the hacks are applicable to one of the most common smart cards that are used for traditional security challenges like login to a computer or a website. So, the classic smart card is not secure to use as the crypto hardware wallet, unless it has direct input and output for the user such as a screen and some buttons.

3.1.4 New Smart Card Capabilities

To summarize, the most secure crypto wallet is the hardware wallet equipped with a screen and at least one physical button. However, as we argued, the 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 Figure 3-2 shows the photo of such a smart card.

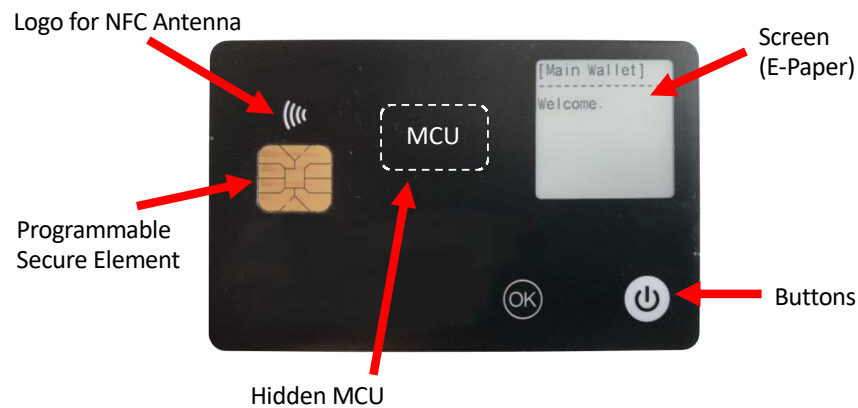


Figure 3-2: Smart card with an e-paper display, physical buttons, and an IC chip

3.2 Crypto Wallet Backup

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. A convenient and secure backup and restore process is a challenge in all crypto wallets including hardware crypto wallets.

3.2.1 Existing Solutions

3.2.1.1 *Paper Backup*

Existing hardware wallets (and many other wallets) use a mnemonic word list to convert the master seed from digital form to physical form as a backup [30]. As we discussed in previous sections, 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.

3.2.1.2 Secret Sharing

One traditional alternative solution is secret sharing [47]. 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.

3.2.1.3 Multi-Signature Wallet

Another solution is multi-signature [48] 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 [16] and [49] 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.

3.2.1.4 Backup on the Cloud

One recent crafted solution is backing up the master seed on the cloud. In this solution, the user chooses a passphrase to generate an encryption key. Then, this key encrypts the master seed and the wallet store it on a user provided cloud storage like iCloud, Google Drive or Drop Box account. So, the user does not worry about keeping a paper in a safe place and the backup never be lost. However, the disadvantage of this solution is that, the cloud server is a honeypot for hackers, and it is similar to the problem of storing the keys on exchange servers and hot wallets. Similar to paper backup, choosing a complex long passphrase is risky when the user may forget that and choosing a simple passphrase is vulnerable to hack, because if a hacker finds the backup on the cloud, he can make brute force attack with no limitation.

3.2.2 Proposed Crypto Wallet Cloning Mechanism

In this research, we propose a cryptographic scheme to tackle crypto wallet backup problem [5]. 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. Furthermore, our scheme does not require the user to protect multiple secrets similar to secret sharing and multi-signature which downgrades the usability of wallets. In addition, it does not enforce using a trusted terminal for backup and recovery and does not need multiple wallets to sign a transaction. Finally, because our proposed mechanism

backs up the wallet on another wallet, there is no any soft file of encrypted the master seed out of the wallet to store on the cloud which has potential hack opportunity.

3.2.2.1 Elliptic-Curve Diffie-Hellman Key Agreement

Our new scheme uses elliptic-curve cryptography to back up the keys. It employs a crafted version of Elliptic-Curve Diffie-Hellman (ECDH) key agreement protocol [3] 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. Figure 3-3 illustrates a general view of ECDH. As we discussed, in ECC the private key value is a random scaler, and the public key is calculated with multiplying ('*') private key with the Generation point (G) defined in secp256k1 domain parameters, and '*' is ECC multiplication [6].

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

Figure 3-3: Elliptic-Curve Diffie-Hellman (ECDH) key agreement

In Figure 3-3 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, the S calculation is as shown in Equation (3) in both parties and S_A is equal to S_B.

$$S_A = S_B = ecPri_A * ecPri_B * G \tag{3}$$

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 [3].

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 key from the fake one. To solve this problem, we employ side-channel user visual confirmation (verification code aka vcode), which will be explained in the next section. Existing hardware wallets use a similar method to confirm transaction information like receiver address, amount and fee before signing [26][27].

3.2.2.2 Proposed Algorithm

In summary, our contributions in this work 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

Figure 3-4 illustrates our proposed backup scheme and Table 3-1 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.

Table 3-1: Acronyms of proposed secure backup mechanism

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
vcodex	Verification code of wallet X (displayed on the hardware wallet screen)
ECDH	Elliptic-Curve Diffie-Hellman algorithm
tk	Transport Key
encMSeed	Encrypted Master Seed

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. The gray boxes in Figure 3-4 illustrate the vcode that displayed on hardware wallets' screen for user verification. The values shown on the two wallets should be identical.

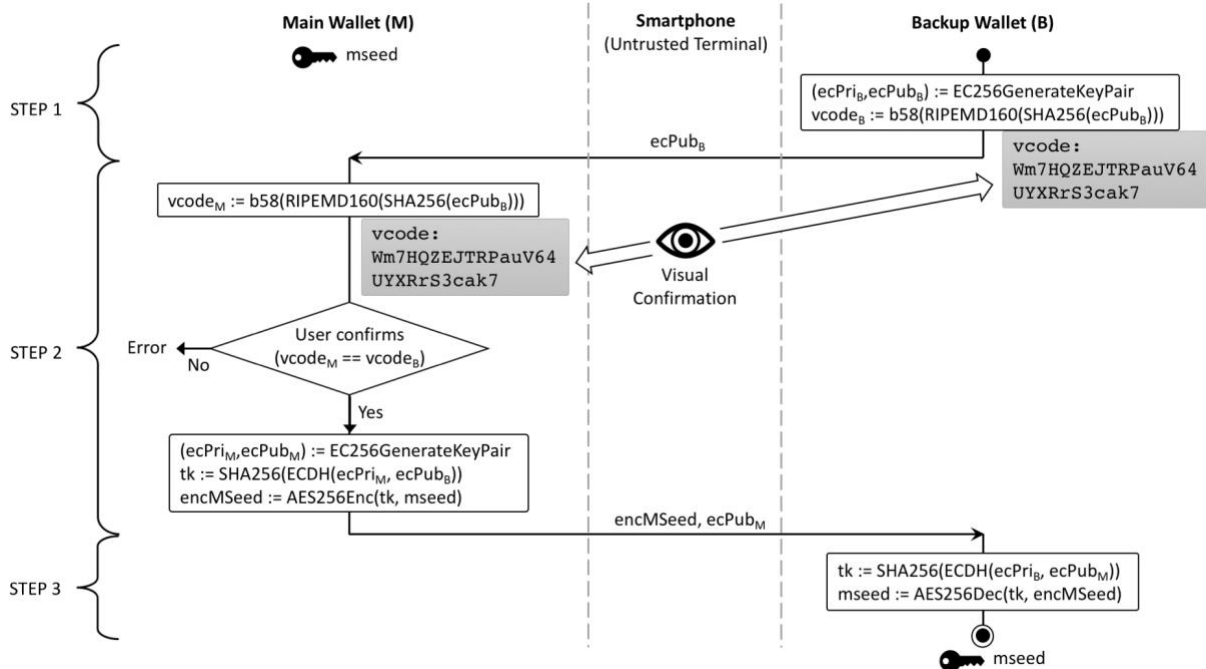


Figure 3-4: Proposed secure backup mechanism to transfer master seed

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.

3.2.3 Prototype Implementation on Smart Card

To build a prototype as a proof-of-concept for our proposed backup mechanism, we implement our code on a secure but resource-constraint hardware that is the smart card. As displayed in Figure 3-2, this smart card has a programmable IC chip, NFC interface, e-paper display and physical buttons. So, it has direct trusted input and output with the user without requiring relying on an untrusted terminal.

As we explained earlier, to develop a card application for the smart card, we employ Java Card technology [33] 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, however, 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 [50] with a few minor changes to add these algorithms.

Another challenge was the public key derivation. Existing Hierarchical Deterministic (HD) 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 crypto 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. In this solution, we use the ECDH key agreement function with the private key as the input key and the Generator point (G) as the input data. Thus, the result of ECDH will be the public key instead of a shared secret, because as we discussed earlier, ECDH actually multiplies the private key to the input Generator point.

Additionally, in actual implementation, we split the second step of our mechanism (shown on Figure 3-4) 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.

Figure 3-5 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.

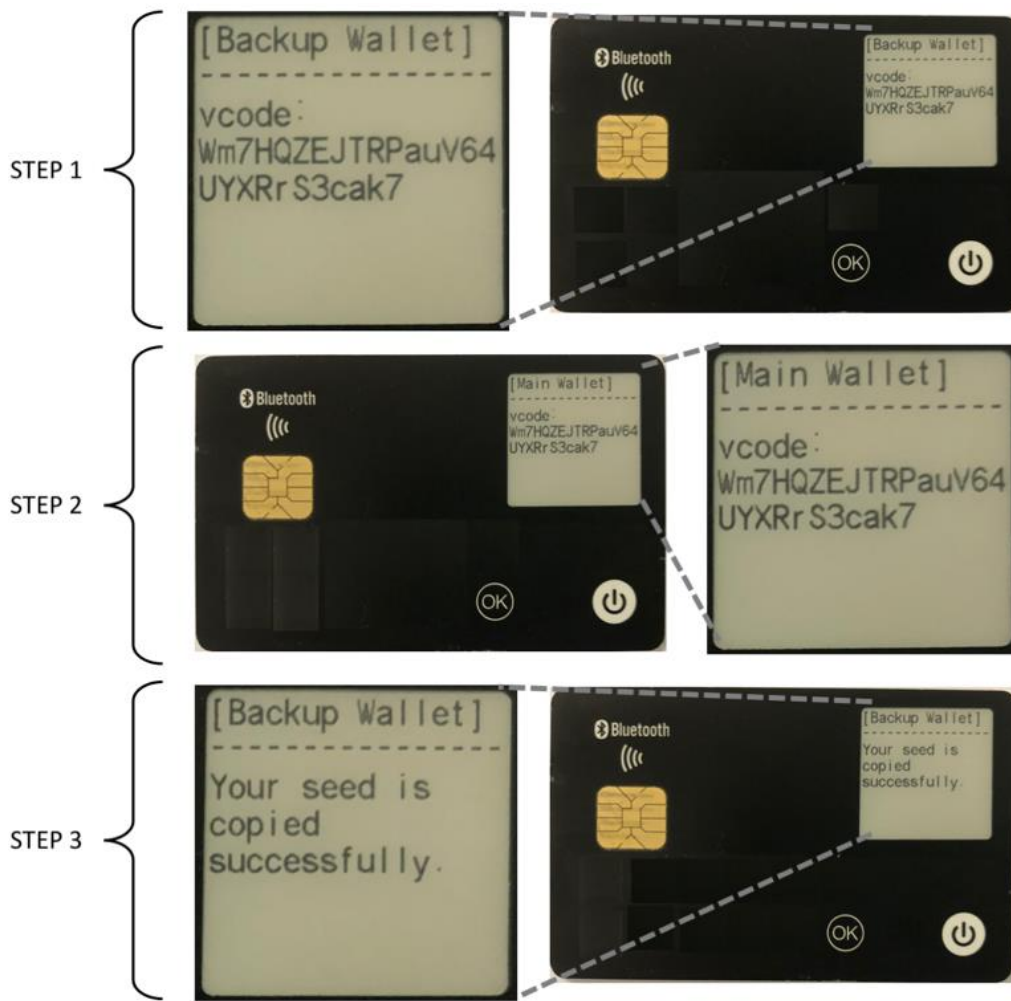


Figure 3-5: The proposed secure backup procedure from the user perspective

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.

3.2.4 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 Figure 3-6. 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 n [6].

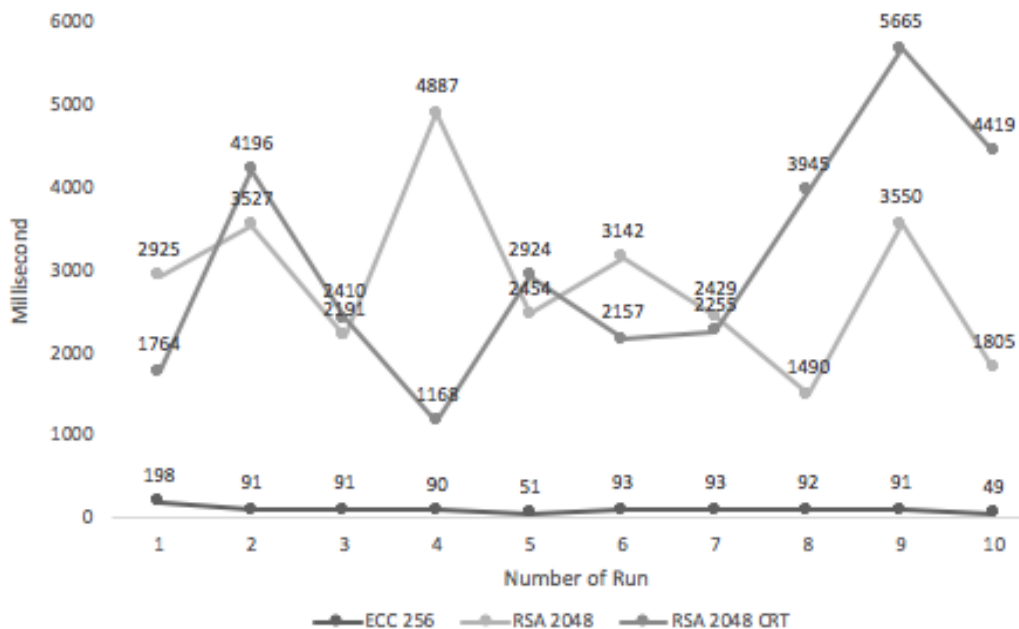


Figure 3-6: Performance of ECC 256-bit and RSA 2048-bit on a smart card

To measure the execution time of ECC and RSA, we implemented a benchmark card application and run that on the smart card. In our test, we run the key generation of ECC 256-bit, RSA 2048-bit and RSA 2048-bit CRT while the last one is the faster version of RSA algorithm. We executed the evaluation for 10 times and the result is displayed on Figure 3-6.

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 10 times and use our test app to measure the time between sending and receiving packets to/from the smart card.

The complete evaluation results are illustrated in Figure 3-7 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.



Figure 3-7: Performance results of the secure backup procedure on a smart card

3.2.5 Security Analysis

3.2.5.1 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 Figure 3-2 similar to existing hardware wallets [26][27].

- 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.

3.2.5.2 Theft of Backup Attack

In the existing backup solution on a piece of paper [30], 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 finds the backup wallet or the main wallet, he needs to know the password of the wallet to unlock it.

3.2.5.3 Vulnerability to Brute Force Attack

To improve the backup security, the existing algorithm [30] 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. It is a best practice in smart card systems. Therefore, if a hacker finds the backup wallet, he can only try a limited number of guessed passwords and could not make a brute force attack.

3.2.5.4 *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 ($encMSeed$) under an AES 256-bit key. Therefore, the attacker cannot extract any private data ($ecPri_M$, $ecPri_B$ and $mseed$).

3.2.5.5 *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 displayed in Figure 3-8.

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

Figure 3-8: Capture the master seed by injecting a key by a hacker

To defend against this attack, we have used a side-channel verification code ($vcode$) in our mechanism. Both wallets compute their $vcode$ s of the backup wallet public key ($ecPub_B$) and display their $vcode$ s on their screens. The user visually inspects and confirms the equality of these two $vcode$ s by pressing a physical button on the main wallet. Existing hardware wallets

use a similar method to confirm transaction information like receiver address, amount and fee before signing them. So, if a hacker injects his fake public key ($ecPr_{IH}$) 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.

3.3 Super-Wallet/Sub-Wallet

Even though the hardware crypto 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.

3.3.1 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.

3.3.2 Proposed Deterministic Sub-Wallet

To resolve the mentioned challenges in the super-wallet/sub-wallet model, we propose a new scheme that we call Deterministic Sub-wallet [5]. 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 sub-wallet, 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 work 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

3.3.3 Classic versus Proposed Super-Wallet/Sub-Wallet Model

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.

3.3.4 Proposed Deterministic Sub-Wallet Details

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 another work [4], 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.

3.3.4.1 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). Equation (4 displays this function.

$$\text{subSeed} = \text{HMAC-SHA512}(\text{key}=\text{"Sub-wallet xxxx"}, \text{data}=\text{masterSeed}) \quad (4)$$

In this equation, we use a procedure similar to the master key generation function in [29] 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.

3.3.4.2 *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 work focuses on first group of cryptocurrencies, but this design is applicable on Ethereum with an additional Smart Contract like [45].

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 3-2. 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 Figure 3-9. 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. Figure 3-9 shows the pseudo-code of the sub-wallet refilling algorithm and Table 3-2 describes the acronyms of the pseudo-code.

```

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

```

Figure 3-9: Sub-wallet refilling pseudo-code

Table 3-2: 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	Number of sub-wallet addresses

To clarify this algorithm, we discuss a simplified example of the sub-wallet refilling procedure illustrated in Figure 3-10. 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, the super-wallet address has 28 Bitcoin and each sub-wallet address (Sub-wallet_{address1} to Sub-wallet_{address5}) has 0.4 Bitcoin. In Figure 3-10 the left side demonstrates the blockchain state before publishing the sub-wallet refilling transaction, and the right side shows the state after that.

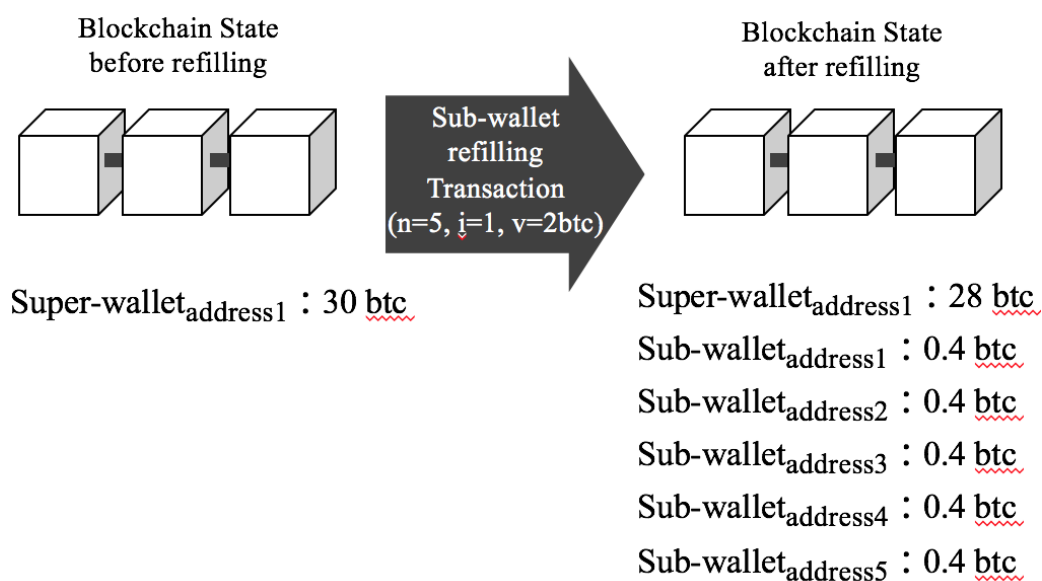


Figure 3-10: Simplified example of proposed sub-wallet refilling mechanism

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.

3.3.4.3 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 we proposed in another work [4]. This algorithm is based on Elliptic-Curve Diffie-Hellman key (ECDH) agreement.

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 [26][27].

3.3.5 Prototype Implementation on Smart Card

As we discussed, one of the most secure crypto wallet is hardware wallet equipped with a screen and at least one physical button, else as [2] and [39] [2]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. As we illustrated earlier, fortunately, now there are new smart cards in the market that use e-paper technology as an on-card screen. Thus, we use a smart card with a screen and a button as a hardware crypto wallet to implement our mechanism.

To develop a card application for the smart card, we employ Java Card technology [33] which as we introduced earlier, it 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.

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. Figure 3-11 demonstrates the whole process from 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.

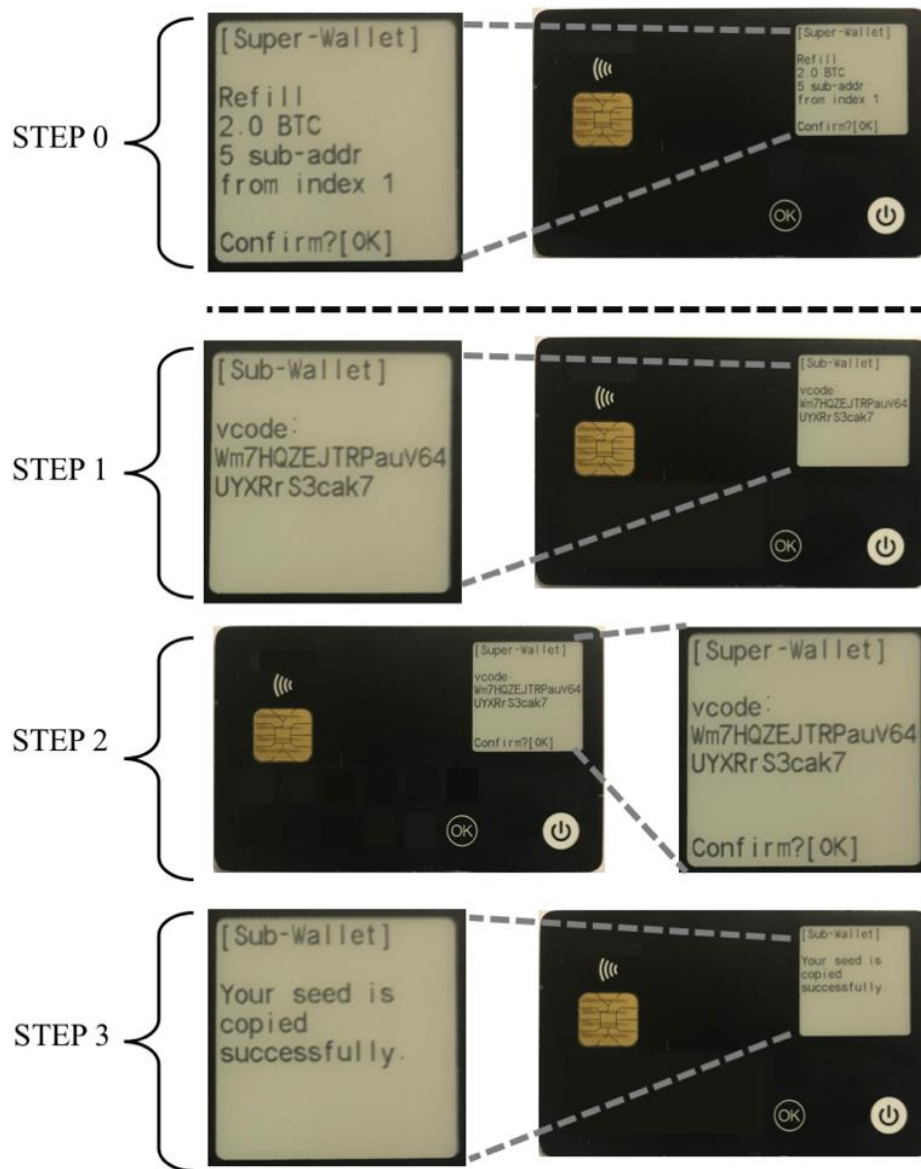


Figure 3-11: Sub-wallet refilling and sub-seed transporting from the user's perspective

3.3.6 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 [51] 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 Figure 3-12. 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.

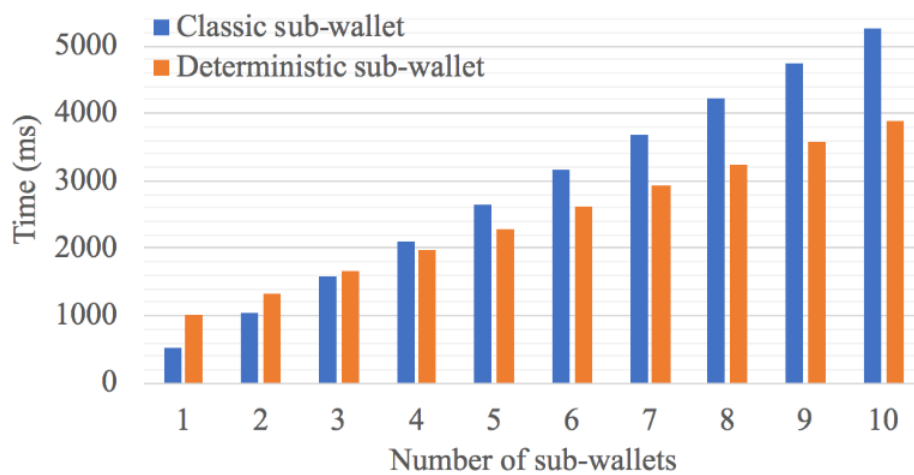


Figure 3-12: Smart card execution time to refill multiple sub-wallets simultaneously

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 [46]. For the time of writing this document, Table 3-3 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 3-3: 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. Figure 3-13 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.

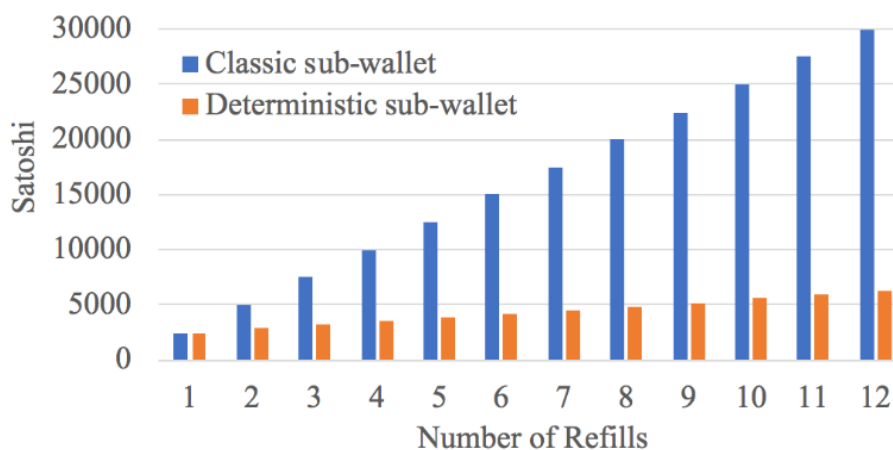


Figure 3-13: Fee to refill one sub-wallet multiple times

The results for the time are similar and Figure 3-14 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.

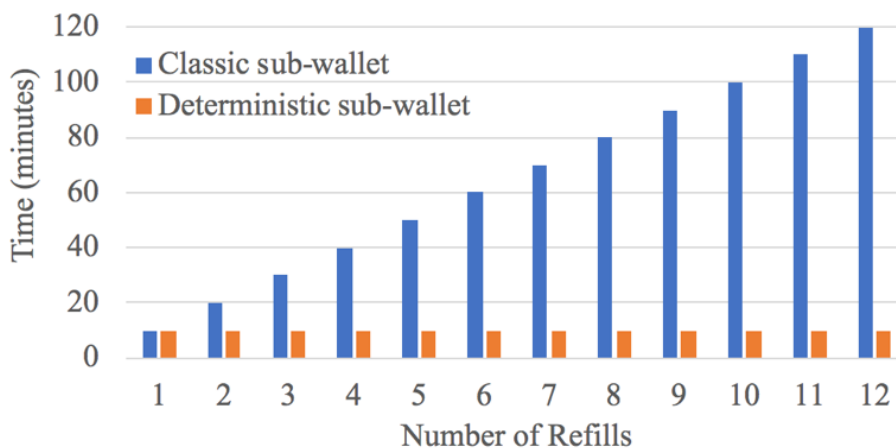


Figure 3-14: Time to refill one sub-wallet multiple times

3.3.7 Security Analysis

3.3.7.1 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 Figure 3-2 similar to existing hardware wallets [26][27].
- The user follows the instructions and checks vcode on both wallets' displays during the sub-seed transfer procedure.

3.3.7.2 *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.

3.3.7.3 *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.

3.3.7.4 *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.

3.3.7.5 MITM: Replacing Sub-Wallet Transport Public Key

Another possible MITM attack is that the attacker relays the messages between the super-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.

3.4 Multilayered Defense-in-Depth Architecture

The hardware wallets are good but not enough because they are hard to use in comparison to hot wallets (i.e., software wallets) and smartphone wallets. We need an appropriate setup when using hardware wallets to achieve a balance between convenience and security. Defense-in-Depth (DiD) is an approach in IT security that usually conveys multiple layers with various security mechanisms to protect a system from attacks in several steps. DiD applies to all IT systems and is a standard solution for network security. In this section, we propose a multi-layer architecture that provides a Defense-in-Depth design for cryptocurrency

wallets. We propose a layered deployment of wallets that delivers a balance between convenience with security for cryptocurrencies. The user protects the private keys in three restricted layers with different protection mechanisms. So, a single breach cannot threaten the entire fund, and it saves time for the user to respond.

3.4.1 Proposed Multi-Layer Wallet

To protect the private keys from attackers, we introduce a defense-in-depth architecture for cryptocurrency wallets. Our proposed architecture has three layers with different usage and protection mechanisms, which makes a balance between usability and security. Figure 3-15 demonstrates this architecture. It has three layers, including *offline layer*, *protected layer*, and *online layer*.

The protected layer consists of a *superior wallet*. This wallet conveys the master seed, which generates the entire key tree and all addresses. The offline layer has at least one *backup wallet* where it is a clone of the superior wallet. We use our previously proposed method in [4] for encrypted wallet-to-wallet cloning. The online layer can have multiple *spending wallets* for regular daily purchases. A spending wallet has a subordinate seed from the superior wallet with a limited fund. We use our previously proposed mechanism in [5] for key derivation to generating subordinate seeds and seed transferring from superior wallet to a spending wallet.

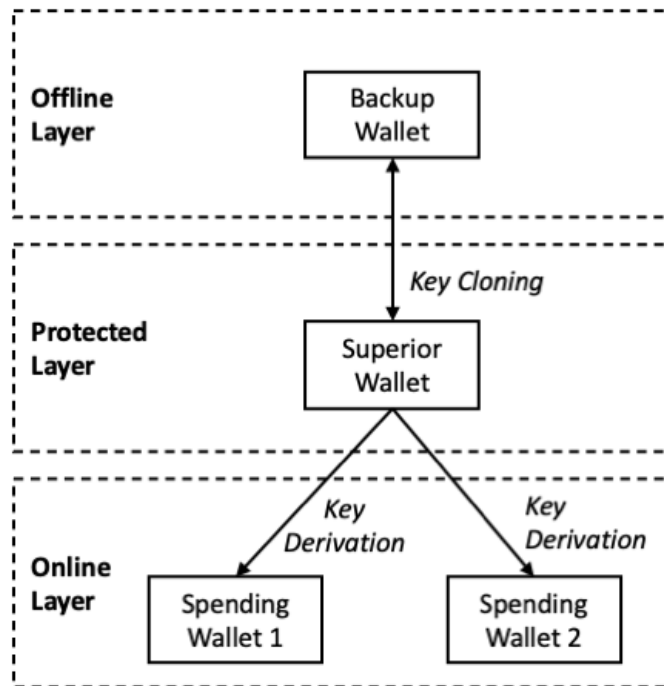


Figure 3-15: The proposed multi-layer defense-in-depth architecture for cryptocurrency wallets

We have revised our previous algorithm [5] to support our new proposed architecture. Firstly, we modify the derivation function as follows where *swSeed* stands for spending wallet seed, *mSeed* stands for master seed, and *xxxx* indicates the spending wallet index starts from zero in 4-digit hex number format (0000).

$$swSeed = \text{HMAC-SHA512}(\text{key}=" swSeed xxxx", \text{data}= mSeed) \quad (5)$$

The superior wallet uses the derivation function only when it creates a new seed for a spending wallet.

Secondly, we also modify the refilling address selection policy. On the original work [5], the wallet only refills the first address index of each derived seed. However, in our new proposed architecture, the superior wallet uses multiple addresses of a spending wallet seed. For each refilling, it searches the blockchain to find the first unused address to send the fund.

The offline layer is designed to be offline and does not need any connection to the blockchain network. It gets online if and only if an incident occurs for the superior wallet and

needs an emergency response. If the superior wallet is compromised by an attack or is lost, the backup wallet generates a brand-new master seed. It creates a blockchain transaction to transfer all available funds of the last master seed to an address under the new master seed. It avoids any unintended transfer from the superior wallet as soon as possible. We recommend a secure hardware wallet with a secure element, a trusted display, and an embedded button for the backup wallet.

The protected layer has only one superior wallet. This wallet only refills the spending wallets. It calculates the spending wallet addresses internally, so it does not send any fund to other addresses that are vulnerable to MITM attack for receiving address injection. Similar to the backup wallet, we recommend a secure hardware wallet for the superior wallet too.

Finally, the online layer can have multiple spending wallets. These wallets can be software wallets like smartphone wallets or hot wallets (third-party hosted wallets). Spending wallets do not need a backup because the superior wallet can recreate their seeds [5].

These three layers provide a balance between security and usability. While the user stores her large fund on the superior wallet and creates a clone of it on the backup wallet, she enjoys the convenience of a smartphone wallet or hot wallet to purchase online and pay her expenses.

Receiving funds does not need private keys, so there are two possible options. If the user context does not have privacy concerns, she can generate an address under master seed on the superior wallet to share with others. If the context is sensitive to privacy, the superior wallet creates an extended public key to generate hierarchical deterministic addresses outside of the superior wallet without exposing the master seed or any private keys [29].

For better understanding, we explain an example setup. Alice has 10 Bitcoin (BTC) equals to \$100k (we assume the bitcoin price is \$10,000 for simplicity). She stores her fund into the superior wallet, which is a secure hardware wallet and keeps it safe at her home. She

creates a backup wallet, which is a secure hardware wallet too, and put it in a safe deposit box in a bank that is physically secure. Then, she installs a wallet app on her smartphone and makes it a spending wallet under the superior wallet and refills 0.5 BTC (\$5K) into it. To receive her salary, she gets a receiving address from the superior wallet and shares it with her employer. She gets paid bi-weekly with bitcoin without requiring using the superior wallet. Alice uses the smartphone to buy a coffee, pay the bills, and purchase from online stores. When the spending wallet has a low balance, she refills it using the superior wallet.

For convenience, Alice uses a type of hardware wallet for superior wallets and backup wallets that support Bluetooth or NFC, and she can do backup and refilling operations using a smartphone. However, she may use an offline laptop or another offline smartphone for better protection to do the backup and refilling.

Now, we consider two possible security incidents and how the defense-in-depth architecture mitigates them. First, assume an incident in the online layer, for example, Alice loses her smartphone or recognizes a malware program on her smartphone. In this scenario, only the spending wallet is at risk with a maximum of 0.5 BTC amount. To respond to this incident, she uses superior wallet to transfer the fund of the suspected spending wallet to an address under the master seed. Then, she can reset her smartphone or get a new one, and the superior wallet generates a brand-new spending wallet seed and transfers the seed to the smartphone.

Secondly, an incident can occur in the protected layer. For example, Alice may lose the superior wallet because of the physical robbery in her home. Since she uses a secure hardware wallet for the superior wallet, it is password protected and, if an attacker tries password guessing more than the retry counter (i.e., five times), the wallet will be blocked permanently. On the other hand, for responding to this incident, Alice uses the backup wallet to generate a brand-new master seed and create a blockchain transaction to transfer all funds from the

previous master seed to an address under the new master seed. She should do that as soon as possible before any breach of the suspected superior wallet. She also must create a new backup and regenerate the subordinate spending wallets.

3.4.2 Proof-Of-Concept

To evaluate our proposed architecture on bitcoin, we use the implementation of the backup wallet and the superior wallet on a hardware wallet device from our previous works that supports fundamental functionalities of hierarchical deterministic wallets, according to BIP-32 [29] and BIP-44 [31]. We use a secure element for key operations such as key generation and digital signature.

We choose a smart card that has essential parts of a secure hardware wallet. It has a secure element for cryptography operations and key storage, a screen to display sensitive information to the user, and a button to get confirmation from the user. Figure 3-2 demonstrates a picture of our test device. This device is in credit card size and has NFC and contact interfaces to communicate. Our test smart card has the following specification; Java Card 3.0.5, Global Platform 2.2.1, e-paper display 256x256 pixel, 2.5 KB memory, 170 KB storage, contact and NFC interfaces, support for SHA256, SHA512, HMAC, AES256, ECC256, and ECDH algorithm.

Since the secure element is a resource-constraint device with limited memory and processing ability, our code must use the minimum amount of memory. We use the sharing memory technique and allocate the entire memory to only two arrays. We pass these arrays with the maintained indexes to the functions that require arrays, and it minimizes the heap consumption. Furthermore, we do not use a very nested function and any recursive call, and it minimizes stack memory usage. We use the Java Card framework [33] to program the secure

element. It is a limited version of Java Virtual Machine with fewer features to run on microcontrollers and secure elements. We compile the code with the Java Development Kit, convert it to a Card Application (CAP), and load it into the secure element.

For the spending wallet, we develop a mobile app to test our prototype with a smartphone. We use a Google Pixel smartphone with an NFC antenna and the following specifications: Google Pixel G-2PW4100 smartphone, quad-core Qualcomm Snapdragon 821 processor with two 2.15 GHz cores and two 1.6 GHz cores, 4 GB memory, 32 GB storage, and Android 8.1.0.

According to our evaluation, the total execution time for creating a backup on the test smart card takes less than one second to complete based on our prototype [4]. The derivation mechanism and refilling a spending wallet also can complete around one second [5].

3.4.3 Security Analysis

In this section, we analyze the security aspect of our proposed architecture and the implemented proof-of-concept on hardware wallets and smartphones. Firstly, we argue about the security advantages of our proposed architecture in comparison to the existing solutions. Next, we provide appropriate adversary models to investigate the possible major attacks and countermeasures.

3.4.3.1 Security Advantages

No Paper Backup: Spending wallets do not need any backup, and the superior wallet has one or more identical backup on other hardware wallets. Therefore, all backups are in digital format, and there is no physical backup on a paper that is vulnerable to traditional attacks.

Less Vulnerable to Lose Large Amount: In our architecture, we split the fund between two layers. The protected layer stores a large amount and is used rarely, while the online layer stores a small amount and is used frequently. Therefore, a spending wallet is more exposed to the network and accessible for attacks; however, it has a small fund at risk. On the other hand, the superior wallet is less accessible on the network, and hence, more secure to possible attacks.

Control of spending wallets: The superior wallet can regenerate the spending wallet seed and all corresponding keys. Therefore, if a spending wallet is lost or stolen, the user can use the superior wallet to recover all spending wallet keys and transfer their funds to a brand-new address and empties the spending wallet.

3.4.3.2 Adversary Models

Authors of [53] survey security analyses on several papers and propose a comprehensive adversary model to employ in future security researches. This model defines three aspects of an adversary, including Assumptions, Goals, and Capabilities. The assumptions describe the environment, resources, and equipment of the adversary. The goals identify the intentions of the adversary and explain why he targets the system. The capabilities are the abilities and actions that the adversary performs to achieve his goals.

The authors of [53] discuss various adversary models for diverse environments like personal computers, networks, and cryptography parties. We use the models of the smartphone environment to measure the security of our final prototype on an Android smartphone.

1) Malicious App Adversary Model

The adversary model has different properties in various fields of study, and the authors of [53] provide several adversary models for smartphone applications. Their proposed Malicious App Adversary Model is appropriate for our conditions. This model includes three

sub-models based on the app permissions: Zero Permission Adversary only has access to the list of installed apps and files stored on external storage. Normal Permission Adversary adds Internet access, Bluetooth, and NFC interfaces. Finally, Dangerous Permission Adversary has access to all resources such as camera, microphone, contact, and SMS. In this section, we use the Dangerous Permission Adversary model to assume maximum power for the attacker that is defined in Table 3-4.

Table 3-4: Adversary Model I: Malicious App with Dangerous Permission

Assumptions	Goals	Capabilities
<ul style="list-style-type: none"> • Android 8.1.0 • Internet access • NFC access • Knowledge of the low-level wallet protocol (APDUs) 	<ul style="list-style-type: none"> • Capture the master seed or sub seed • Inject the adversary address to receive the fund 	<ul style="list-style-type: none"> • Record the screen or log the pressed buttons to capture the password • Sniff the low-level packets to capture the master seed or spending seed • Inject the adversary address into spending wallet refill transaction to receive the fund (MITM) • Replace the backup or spending wallet original transport public key with the adversary public key to extract the master seed or spending seed (MITM)

According to Table 3-4, the adversary could capture the user's password by recording the screen or log the pressed buttons. Even though some solutions exist for this attack like Trezor [54] that uses a blind visual matrix to avoid entering a plain password on the host, we use a physical button on the hardware wallet for confirmation.

Also, the adversary may sniff the transmitted messages between hardware wallets and the smartphone app to eavesdrop the master seed or spending seed. Our mechanism is secure against this attack because the smartphone only transmits public information, including the superior wallet, the backup wallet and spending wallet public keys, and encrypted master seed or encrypted seed under an AES 256-bit key. Therefore, the attacker does not have access to any private data.

Another capability of the adversary is making an MITM attack to replace the receiver address by his injected address in the transaction. The classic super-wallet/sub-wallet model [1] is vulnerable to this attack because the super-wallet needs to get the sub-wallet address from the host like a smartphone. However, in our architecture, we use the deterministic sub-wallet that prevents this attack since the spending wallet seeds are derived from the superior wallet master seed, and the superior wallet generates the receiving addresses internally. Therefore, there is no need to get the receiving addresses from the external source, and the hacker has no chance to replace them.

Last but not least, the adversary may make an MITM attack to intercept the messages between the superior wallet and the backup wallet or the superior wallet and the spending wallet. Then, he replaces the backup wallet public key or the spending wallet public key by the adversary public key in ECDH key agreement, and he can recover the transferred seed.

To defend against this attack, we have used a side-channel verification code (vcode) in our mechanism. Both wallets compute their vcodes of the public key and display the vcode on their screens (see the hardware wallet shown in Figure 3-2). The user visually inspects and confirms the equality of these two vcodes by pressing a physical button on the superior wallet. Existing hardware wallets use a similar method to confirm transaction information like receiver address, amount, and fee before signing them. Therefore, during the wallet transfer operation, if a hacker injects his public key to the superior wallet, the user will be able to detect such an attack due to the mismatch of the two vcodes shown on two wallets' screen and reject this MITM attack.

2) Physical Access Adversary Model

Another possible adversary model for our proposed architecture is an adversary with physical access to the superior wallet (or backup wallet). In this case, the adversary can do

anything directly with the hardware wallet without the need to install a malicious app on the remote user's smartphone. Table 3-5 demonstrates the Physical Access Adversary Model.

Table 3-5: Adversary Model II: Physical Access

Assumptions	Goals	Capabilities
<ul style="list-style-type: none"> • Access to the hardware wallet device • Knowledge of the low-level wallet protocol (APDUs) 	<ul style="list-style-type: none"> • Sign a transaction and send the fund to the adversary address 	<ul style="list-style-type: none"> • Make a brute-force attack to guess the password and sign a transaction to transfer the fund

In this adversary model, the adversary can make a brute-force attack to obtain the hardware wallet password (PIN code) and sign his desired transaction. Our proposed architecture recommends a hardware wallet with a secure password for the superior wallet that has a fixed password retry counter, usually between 3 and 15. After that, the secure element locks permanently. It is a standard mechanism for secure elements. Therefore, if a hacker finds the superior wallet, he can only try a limited number of guessed passwords and could not make a brute-force attack. For instance, if the PIN code length is four digits and the retry counter is 10, the chance to find the PIN code is 0.001 (tries / possible PINs = 10/10⁴ = 0.001). On the other hand, the user has time to use her backup wallet to transfer all funds to a brand-new seed as soon as possible.

We must mention that the attacks to the security element or other hardware parts and their countermeasures are out of the scope of this project and apply to entire hardware wallets not specific for our proposed schemes.

3.5 Off-Chain Transaction to Avoid Inaccessible Wallet

Today, a user can perform various electronic commerce transactions like paying a bill, booking a hotel or flight, purchasing online products, and paying taxes with cryptocurrency. While cryptocurrencies become more usable for average users, the inaccessible coins issue arises as a challenging problem in cryptocurrencies. Since, as a design paradigm, only the user's private key can send the coins from its associated address, if the user cannot access her private key, she loses her coins. The user may forget her password, or in a worst-case, she may die, and her coins will be lost forever. It happens in the cryptocurrency many times, and as several reports like [55] shows, about 21 percent of all possible bitcoins are out of circulation and maybe are lost forever.

Furthermore, there are several cases where the owner of the key dies or pretends to die to steal the coins from others. These persons have control of other users' coins like investors in the position of an online cryptocurrency exchange president or something like that [56]. Since there is no clear technical solution to recover the lost coins, the investors lost their money.

There are limited choices for users to avoid inaccessible coins, such as creating a backup for another person or using a multi-signature wallet. These solutions not only are inconvenient but also put the user at risk. Recently, authors of [57] suggested generating an off-chain recovery transaction and publishing such a transaction when the coins are inaccessible. The wallet must frequently regenerate this transaction because any change in inputs by a sending transaction invalidates the previously generated recovery transaction. Also, any receiving transaction conveys new coins that should be added to the recovery transaction.

In this section, we investigate this off-chain recovery transaction and evaluate its performance in real conditions with actual hardware wallets as a secure option for cryptocurrency users. We demonstrate that generating such a recovery transaction consumes a

significant amount of time on hardware wallets or other resource-constraint wallets. Hence, it is not a practical solution in real life. To resolve this performance challenge, we propose a new key management schema to separate frequent micropayments from other transactions and keep the recovery transaction updated with regenerating as less frequent as possible. Our proposed schema prevents inaccessible coins in most cases and provides better performance compared to the previous method.

3.5.1 Recovery Transaction

Authors of [57] explain a mechanism to recover inaccessible wallets using an off-chain transaction. Each time that the wallet sends or receives a coin, the wallet creates a recovery transaction to gather all available coins in Unspent Transaction Outputs (UTXO) and saves it on a file. When the user forgets her wallet password or the password became inaccessible because of any reason like death, the wallet *retirement* mechanism activates with a policy like no login for more than six months. The wallet publishes the last recovery transaction and transfers all coins to a reserved address. Since the recovery transaction is signed in-advance, there is no need for private keys.

On the other hand, all received coins in the last six months aka retirement period, are lost because these new coins are not included in the recovery transaction. The user can set the retirement period. It is not a timer on the wallet; it is a value embedded into the recovery transaction itself. So, if the wallet or other entity publishes the recovery transaction on the blockchain, it will not be effective until the pre-defined time. This mechanism works for UTXO-based cryptocurrencies like bitcoin, and the lock time in bitcoin transactions support this feature.

The recovery transaction that authors of [57] explain is designed for old-fashion software wallets like Satoshi Client [58] that runs on a powerful enough personal computer. However, regenerating a recovery transaction has a significant performance problem in modern wallets like mobile wallets that use Trusted Execution Environment [23][24] and hardware wallets that run on a microcontroller and secure element with limited resources. In this section, we propose a practical off-chain recovery transaction that avoids inaccessible coins in hardware wallets with a minimum performance penalty. We call it *lean* recovery transaction.

3.5.2 Hardware Wallet Architecture

As we discussed earlier, a hardware wallet is a dedicated cryptographic device to generate and store the secret keys and sign the transactions. Since a hardware wallet is not a general-purpose computer, a hacker cannot install a malware program easily. Furthermore, some secure hardware wallets have a *secure element*. It is a tamper-resistant module to protect the secrets from electrical and physical attacks such as side-channel attacks and power-analysis.

Hardware wallets usually have a screen and a few buttons to interact with the user directly; otherwise, they are vulnerable to Man-In-The-Middle attack [2]. Figure 3-16 depicts the general components of hardware wallets. They usually have a main control unit (MCU) that connects all components and communicates with the host application via USB, Bluetooth, or NFC.

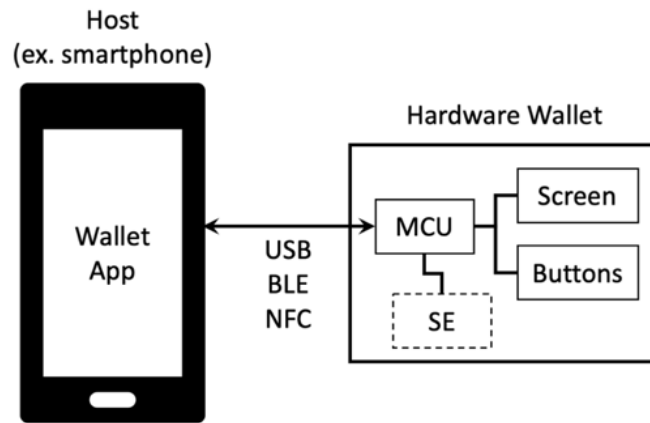


Figure 3-16: General hardware wallet components

Since a hardware wallet does not have internet access, it uses an app on the host like a personal computer or a smartphone to connect to the blockchain network. However, critical tasks like storing the keys and signing a transaction will be done on the hardware wallet. The overall procedure of signing a transaction on a hardware wallet is as follows.

Transaction Signing Process on a Hardware Wallet:

1. Host App: Gather information from blockchain nodes and prepare inputs and outputs.
2. Hardware Wallet: Receive data and display the receiving addresses, amount, and fee of the transaction on the embedded screen and get the user confirmation by pressing an embedded button.
3. Hardware Wallet: Derive required keys and sign the transaction for each input, and return the result to the host app.
4. Host App: Publish the signed transaction to the blockchain nodes.

While the network connection is good, and the host has enough resource, the time-consuming steps are step 2 and step 3 that run on the hardware wallet. A transaction with more input UTXOs takes more time on the hardware wallet for key derivation and digital signature.

3.5.3 Evaluating Recovery Transaction for Hardware Wallets

In this section, we conduct some experiments to evaluate the recovery transaction suggested in [57] with real hardware wallets. We illustrate that the recovery transaction is a heavy-loaded transaction to generate. We show that creating a brand-new recovery transaction for all sendings and receivings has a significant performance penalty, which makes it impractical in resource-constraint cryptocurrency wallets like hardware wallets.

First of all, in contrast to payment transactions, recovery transaction has several inputs and only one output. It aggregates entire available UTXOs to transfer all coins to the reserved address. Multiple inputs make the recovery transaction larger than a typical payment transaction. A recovery transaction needs several key derivations to calculate required private keys for all UTXOs and several ECC signings to generate outputs. Even though a recovery transaction is not very different from a payment transaction for traditional software wallets like Satoshi Client [58] that run on a computer, it has a significant performance penalty on a resource-constraint device like Hardware Wallets.

Since bitcoin is the gold standard in UTXO-based cryptocurrencies and many other coins copy the entire or parts of its codebase, we choose bitcoin to do our measurement. We also choose Segregated Witness protocol aka SegWit to perform our tests. It is a new version of the bitcoin protocol [59] with better performance for multiple inputs. To employ SegWit protocol, we use the following path for key derivation:

$$\text{path} = \text{m}/49'/1'/0'/\text{change}/\text{address_index} \quad (6)$$

The number 49 refers to BIP-49 [60] that defines the derivation scheme for SegWit addresses. Next number 1 is the defined constant for bitcoin testnet. To compare recovery transactions with typical payment transactions, we use the typical payment transaction format with one input and two outputs. The recovery transaction has one to ten inputs for available

UTXOs and one output for the reserved address. It may have more than ten inputs in real life, but we assume this number just for demonstration. We use WireShark to monitor USB packets and measure the timing [61]. Our tests executed on a MacBook Pro with Intel Core i7 2.2 GHz processor and 16 GB memory, and we use the same USB port for all tests.

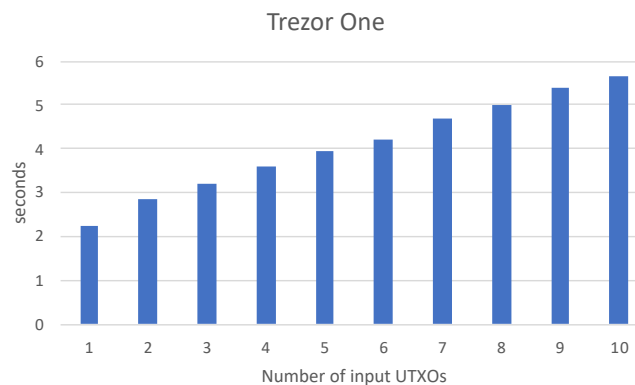


Figure 3-17: Performance of generating recovery transaction on a Trezor One hardware wallet

To evaluate generating a recovery transaction on hardware wallets, we only measure step 2 and step 3 of the *Transaction Signing Process on the Hardware Wallet*, because step 1 and step 4 execute on the host application and network. Figure 3-17 and Figure 3-18 demonstrate the results for both hardware wallets. Increasing the number of input UTXOs takes more time on the wallet to generate a recovery transaction. In comparison, Ledger Nano S has lower performance because it uses a secure element [62]. In the worst-case scenario, generating a recovery transaction on a secure hardware wallet like Ledger Nano S takes around 40 seconds, with only ten input UTXOs.

Authors of [57] discussed that the wallet must create a recovery transaction after all sending transactions because one or more input UTXOs is spent. Spending invalidates the previous recovery transaction because at least one of its input UTXOs is not available. In other words, the wallet has to generate a brand-new recovery transaction after even a micropayment transaction like buying a coffee, purchasing a ticket, or paying a bill.

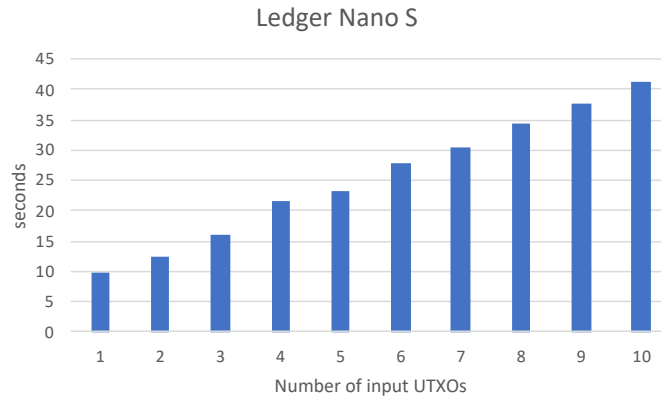


Figure 3-18: Performance of generating recovery transaction on a Ledger Nano S hardware wallet

3.5.4 Proposed Lean Recovery Transaction

As we explained, inaccessible coins are a big challenge in cryptocurrencies. The recovery transaction proposed in [57] to generate two transactions for each payment and save one of them off-chain for disaster recovery has a significant performance penalty in reality.

In this section, we propose a more efficient solution called *lean* recovery transaction. In this solution, the wallet generates the recovery transaction less frequently, and only when needed. To do that, we make a change in wallet key management and divide the key tree into two sections. One section is assigned to a spending account, and the other section includes other accounts. The path is as follows when the account is 0 for spending account and non-zero for non-spending accounts.

$$\text{path}=\text{m}/\text{purpose}'/\text{coin}'/\text{account}(0|\text{n})'/\text{change}/\text{addr_index} \quad (7)$$

Figure 3-19 illustrates a sample key tree. The wallet uses only the spending section for all spendings (micropayments). It creates the off-chain recovery transaction only for non-spending section, which means all addresses except addresses under the spending account. We call it *lean* recovery transaction because it does not include the heavy part of a recovery transaction, which includes many but small amount UTXOs.

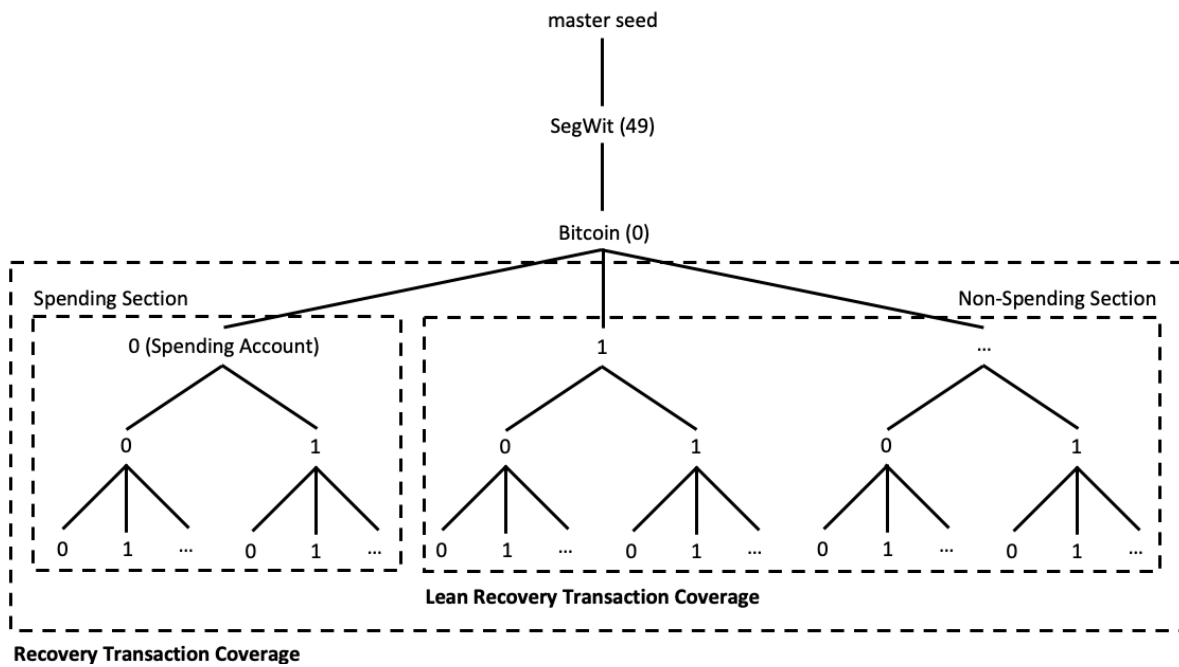


Figure 3-19: Sample key tree to illustrate the coverages of Recovery Transaction and our proposed Lean Recovery Transaction

Only sending and receiving for addresses out of the spending account requires regenerating a recovery transaction, like buying new bitcoins or getting paid for salary with cryptocurrency. So, micropayments do not change the inputs of the existing lean recovery transaction, and only large payments need a new one.

In another scenario, for receiving transactions, a new received UTXO must be added to the recovery transaction to avoid potential inaccessibility of it. To prevent from regenerating a recovery transaction for all receives even small transactions, we define a threshold that can be changed by the user. If the sum of receiving coins reaches the threshold, the wallet generates a new recovery transaction to add the new UTXOs.

The spending account does not receive any coins from outside. So, we define a new *transfer* function, where the user transfers coins from other accounts to the spending account or, in other words, from the non-spending section to the spending section. After creating a

transfer transaction, the wallet generates a new recovery transaction because its inputs have been changed.

Our proposed schema has the following advantages in comparison to the recovery transaction proposed in [57]:

- Generating a lean recovery transaction takes considerably less time on the wallet because it has fewer input UTXOs, which is crucial on hardware wallets.
- The wallet generates the lean recovery transaction less frequently because spending from the spending account does not change the input UTXOs of the existing lean recovery transaction and does not invalidate it.
- Common payment transactions for micropayments are faster in our proposed mechanism because they do not need to generate a recovery transaction anymore.
- The wallet adds new receiving UTXOs into a lean recovery transaction only when their total funds reaches a defined threshold, and it makes generating recovery transactions less frequent.

To help a reader understanding our proposed lean recovery transaction mechanism, we use an example to illustrate and compare the recovery transaction in [57] and our proposed method. Assume that a user has a bitcoin wallet with \$7000 value that conveys three UTXOs with \$500, \$2500, and \$4000 equivalent bitcoin.

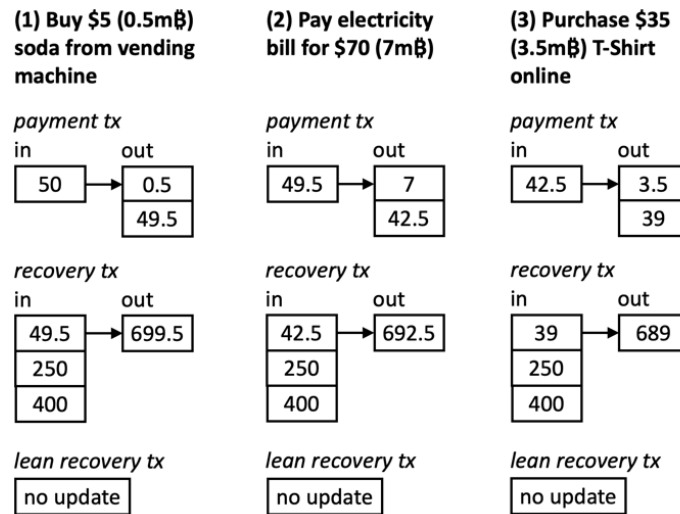


Figure 3-20: Example for comparing lean recovery transaction with recovery transaction

Suppose the user makes three usual payments today to buy a \$5 soda, pay an electricity bill for \$70, and purchase a \$35 T-Shirt from an online store. She uses her wallet to make these payments by bitcoin. Figure 3-20 illustrates sample bitcoin transactions that the wallet generates. We ignore purchase taxes and fees, bitcoin exchange fee, and bitcoin network fee to simplify the example. We assume the bitcoin price is \$10,000 and we use milli-bitcoin (mBTC or m฿) in our example.

We assume that before beginning, the wallet has generated a valid recovery transaction. In the first scenario, the wallet uses recovery transaction described in [57] which includes all three UTXOs with 50, 250 and 400 mBTC. In the second scenario, the wallet uses our proposed lean recovery transaction including only two UTXOs with 250 and 400 mBTC and assigns one UTXO with 50 mBTC to the spending account.

This example demonstrates that each payment in the first scenario includes generating a payment transaction and a recovery transaction, while in the second scenario it includes generating only a payment transaction without any recovery transaction.

In our test setup that we described earlier, Trezor One [63] hardware wallet takes 2.2 seconds for a payment transaction and 3.2 seconds for a recovery transaction with three

UTXOs. Ledger Nano S takes 9.7 seconds and 15.8 seconds respectively. So, the payment process takes 5.4 seconds for Trezor One and 25.5 seconds for Ledger Nano S in scenario one, while it uses 2.2 seconds for Trezor One and 9.7 seconds for Ledger Nano S in scenario two when using the lean recovery transaction. Therefore, the lean recovery transaction has significant advantage, at least %40 percentage of less processing time for generating payment transactions with three input UTXOs. The performance difference becomes even bigger with larger number of UTXOs in the wallet.

3.5.5 Evaluation

To evaluate lean recovery transaction model, we modify our previous implementation on a hardware wallet that supports fundamental functionalities of hierarchical deterministic wallets, according to BIP-32 [29] and BIP-44 [64]. We use a secure element for key operations such as key generation and digital signature.

As discussed, our proposed lean recovery transaction has several advantages compared to the recovery transaction explained in [57] because it generates lighter recovery transactions with less input UTXOs. It reduces the number of generating recovery transactions by assigning a section in the key tree to spending and defining a threshold for adding receiving funds to the recovery transaction. In this section, we measure the performance of our proposed lean recovery transaction on our implemented proof-of-concept wallet with a secure element.

We test our implementation with two payment scenarios. In the first scenario, the wallet uses the recovery transaction proposed in [57] and generates a recovery transaction just after the payment transaction. In the second scenario, the wallet uses the lean recovery transaction mechanism, and it does not generate a recovery transaction for payment transactions. Figure

3-21 illustrates the result of our tests in both scenarios for various recovery transaction sizes with one to ten input UTXOs.

Since the lean recovery transaction schema does not require regenerating a recovery transaction after each micropayment, the payment transaction performance does not have any change in Figure 3-21. On the other hand, the regular recovery transaction makes double the payment transaction time on the wallet, and more input UTXOs increases its generating time.

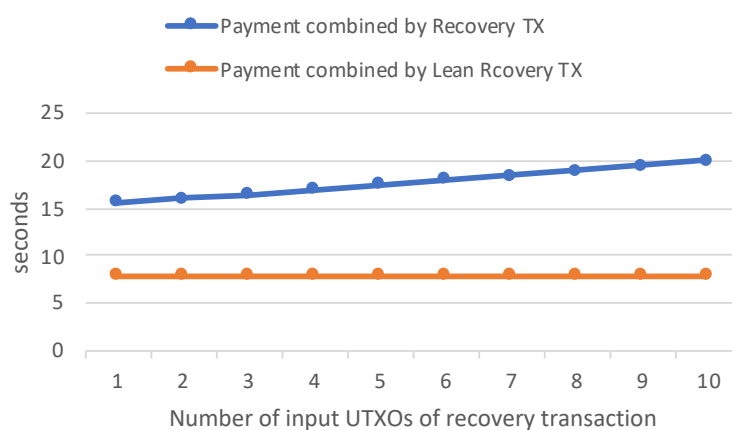


Figure 3-21: Comparison of micropayment transactions in recovery transaction proposed in [57] and our proposed lean recovery transaction schemas

Our tests have executed on a MacBook Pro with Intel Core i7 2.2 GHz processor and 16 GB memory, and we use the same USB port for all tests which is similar to our tests for evaluating the previous recovery transaction on two hardware wallets.

CHAPTER 4: CONCLUSION

In this thesis, we considered the significant issues in crypto wallets for blockchain technology. Even though the most secure choice is hardware wallets, we argued that there are critical issues that should be addressed. After providing the required technical background about cryptography primitives, blockchain technology, crypto wallets, and smart card, we proposed a secure and convenient backup mechanism and super-wallet/sub-wallet model with deterministic sub-wallet. We used our crafted Elliptic-Curve Diffie-Helman for key transfer using out-of-band visual confirmation by the user. We also proposed a multi-layer architecture based on our previous works for cryptocurrency wallets. Finally, we offered an efficient off-chain recovery transaction to avoid inaccessible wallets. For our proposed mechanisms, we implemented a prototype as a proof-of-concept on a smart card, which is a secure but resource-constrained option to build a hardware wallet. We also provided performance evaluation and security analysis for these mechanisms.

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] 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, IEEE, 2018.
- [3] “SEC 1: Elliptic Curve Cryptography”, Version 2.0, Standard for efficient cryptography group, 2009.
- [4] H. Rezaeighaleh, C. C. Zou, “New Secure Approach to Backup Cryptocurrency Wallets”, in Proceedings of the 2019 Global Communications Conference (GLOBECOM), IEEE, 2019.
- [5] H. Rezaeighaleh, C. C. Zou, “Deterministic Sub-Wallet for Cryptocurrencies”, in Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain-2019), IEEE, 2019.
- [6] H. Rezaeighaleh, C. C. Zou, “Multilayered Defense-in-Depth Architecture for Cryptocurrency Wallet”, in Proceedings of the IEEE 6th International Conference on Computer and Communications (ICCC2020), IEEE, 2020.
- [7] H. Rezaeighaleh, C. C. Zou, “Efficient Off-Chain Transaction to Avoid Inaccessible Coins in Cryptocurrencies”, in Proceedings of the 3rd International Workshop on Blockchain Systems and Applications (BlockchainSys2020), IEEE, 2020.
- [8] “SEC 2: Recommended Elliptic Curve Domain Parameters”, Version 2.0, Standard for efficient cryptography group, 2010.
- [9] D. Hankerson, S. Vanstone, A. Menezes, “Guide to Elliptic Curve Cryptography”, Springer Professional Computing, 2004.

- [10] S. Haber, W. S. Stornetta, “How to time-stamp a digital document”, *Journal of Cryptology*, Volume 3, Issue 2, pp 99–111, 1991.
- [11] D. Bayer, S. Haber, W. S. Stornetta, “Improving the Efficiency and Reliability of Digital Time-Stamping”, *Sequences II: Methods in Communication, Security and Computer Science*, pp 329-334, 1992.
- [12] H. Finney, “Reusable Proofs of Work”, Online:
<https://nakamotoinstitute.org/finney/rpow/index.html>, 2004.
- [13] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System”, Online:
<https://bitcoin.org/bitcoin.pdf>, 2008.
- [14] V. Buterin, “A Next-Generation Smart Contract and Decentralized Application Platform”, Online: <https://github.com/ethereum/wiki/wiki/White-Paper>, 2013.
- [15] D. Larimer, “Delegated Proof-of-Stake (DPOS)”, Online:
https://github.com/bitshares/bitshares.github.io/blob/master/_drafts/v1/2014-04-03-delegated-proof-of-stake.md, 2014.
- [16] A. Narayanan, J. Bonneau, E. W. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*, Princeton University Press, 2016.
- [17] Open Source JavaScript Client-Side Bitcoin Wallet Generator, Online:
<https://www.bitaddress.org>
- [18] Coinbase home page, <https://www.coinbase.com/>
- [19] P. Rizzo, “Mt. Gox allegedly loses \$350 million in Bitcoin (744,400 BTC), rumoured to be insolvent”, *CoinDesk*, Feb. 25, 2014 [Online]. Available:
<https://www.coindesk.com/mt-gox-loses-340-million-bitcoin-rumoured-insolvent>
[Accessed Oct. 8, 2018].

- [20] Y. Nakamura, A. Tan, and Y. Hagiwara, “Coincheck Says It Lost Crypto Coins Valued at About \$400 Million”, Bloomberg, Jan. 26, 2018 [Online]. Available: <https://www.bloomberg.com/news/articles/2018-01-26/cryptocurrencies-drop-after-japanese-exchange-halts-withdrawals> [Accessed Oct. 8, 2018].
- [21] 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].
- [22] 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].
- [23] 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.
- [24] 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.
- [25] 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.
- [26] Ledger Nano X, Online: <https://shop.ledger.com/pages/ledger-nano-x>
- [27] Ledger Blue, Online: <https://shop.ledger.com/products/ledger-blue>
- [28] D. Nedospasov, J. Datko, T. Roth, “35C3 - wallet.fail”, Online: <https://wallet.fail/>
- [29] “Hierarchical Deterministic Wallets”, Bitcoin Improvement Proposal 32 (BIP-0032), 2012.

- [30] “Mnemonic code for generating deterministic keys”, Bitcoin Improvement Proposal 39 (BIP-0039), 2013.
- [31] “Multi-Account Hierarchy for Deterministic Wallets”, Bitcoin Improvement Proposal 44 (BIP-0044), 2014.
- [32] W. Rankl, W. Effing, “Smart Card Handbook”, Third Edition, Wiley, 2002.
- [33] “Java Card Runtime Environment Specification,” 3rd Edition, 2011.
- [34] Z. Chen, “Java Card™ Technology for Smart Cards, Architecture and Programmer’s Guide”, Addison-Wesley Professional, 2000.
- [35] “GlobalPlatform Card Specification”, Version 2.1.1, 2003.
- [36] GlobalPlatformPro, Online: <https://github.com/martinpaljak/GlobalPlatformPro>
- [37] “ISO/IEC 7816 Part 4: Organization, security and commands for interchange”, Second Edition, 2005.
- [38] “jCardSim - Java Card Runtime Environment Simulator”, Online:
<https://jcardsim.org/>
- [39] B. Schneier, 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.
- [40] “FIPS PUB 140-2: Security requirements for cryptographic modules”, National Institute of Standards and Technology, December 2002
- [41] “Common Criteria for Information Technology Security Evaluation (CC)”, Online:
<https://www.commoncriteriaportal.org/>
- [42] “SP 800-73-3. Interfaces for Personal Identity Verification”, National Institute of Standards and Technology, February 2010
- [43] “Windows smart card minidriver specification”, Version 7.07, Microsoft corporation, February 2016

- [44] “MinidriverSpy”, <https://github.com/hosseinpro/MinidriverSpy>
- [45] MultiSend Smart Contract [Online]. Available:
<https://github.com/Alonski/MultiSendEthereum>
- [46] Bitcoin Fees [Online]. Available: <https://bitcoinfees.info>
- [47] A. Shamir, “How to share a secret”, *Communication of the ACM*, Vol. 22, Issue 11, pp 612-613, 1979.
- [48] “Multisignature”, Bitcoin wiki, 2019 [Online]. Available:
<https://en.bitcoin.it/wiki/Multisignature> [Accessed May. 7, 2019].
- [49] S. Meiklejohn, “Top Ten Obstacles along Distributed Ledgers Path to Adoption”, *IEEE Security & Privacy*, vol. 16, issu. 4, pp. 13-19, 2018.
- [50] Ledger Unplugged [Online]. Available: <https://github.com/LedgerHQ/ledger-javacard>
- [51] smartcardPage [Online]. Available: <https://github.com/hosseinpro/smartcardPage>
- [52] “PKCS #5: Password-Based Cryptography Specification”, Version 2.1, Online:
<https://tools.ietf.org/html/rfc8018>
- [53] D. Quang, B. Martini and C. K.-K. Raymond, "The role of the adversary model in applied security research," *Computers & Security*, vol. 81, pp. 156-181, March 2019.
- [54] Trezor, "User manual: Entering PIN," [Online]. Available:
https://wiki.trezor.io/User_manual:Entering_PIN.
- [55] B. Brown, "21% of Bitcoin Hasn't Moved for Five Years, Stroking Monumental Supply Shock," 24 July 2019. [Online]. Available: <https://www.ccn.com/21-of-bitcoin-hasnt-moved-for-five-years-stroking-monumental-supply-shock/>.
- [56] N. De, "Troubled Canadian crypto exchange QuadrigaCX owes its customers \$190 million and cannot access most of the funds, according to a court filing obtained by CoinDesk," 1 Feb 2019. [Online]. Available: <https://www.coindesk.com/quadriga-creditor-protection-filing>.

- [57] P. Rakdej, N. Janpitak, M. Warasart and W. Lilakiatsakun, "Coin Recovery from Inaccessible Cryptocurrency Wallet Using Unspent Transaction Output," in *2019 4th International Conference on Information Technology (InCIT)*, Bangkok, Thailand, 2019.
- [58] "Original Bitcoin client," [Online]. Available: https://en.bitcoin.it/wiki/Original_Bitcoin_client.
- [59] "Segregated Witness," [Online]. Available: https://en.bitcoin.it/wiki/Segregated_Witness.
- [60] D. Weigl, "Derivation scheme for P2WPKH-nested-in-P2SH based accounts," 2016. [Online]. Available: <https://github.com/bitcoin/bips/blob/master/bip-0049.mediawiki>.
- [61] G. Harris, "USB capture setup," 2019. [Online]. Available: <https://wiki.wireshark.org/CaptureSetup/USB>.
- [62] "Frequently asked questions - Ledger Support," [Online]. Available: <https://support.ledger.com/hc/en-us/articles/360015216913-Frequently-asked-questions>.
- [63] "Trezor One," Trezor, [Online]. Available: <https://shop.trezor.io/product/trezor-one-white>.
- [64] M. Palatinus and P. Rusnak, "Multi-Account Hierarchy for Deterministic Wallets," 2014. [Online]. Available: https://en.bitcoin.it/wiki/BIP_0044.