VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

BACHELOR OF ENGINEERING THESIS

A HIERARCHICAL DETERMINISTIC WALLET FOR ED25519

COMPUTER SCIENCE COMMITTEE

Supervisors: Dr. Nguyen An Khuong, Ph.D.
Mentors: Mr. Phan Son Tu
Examiner: Dr. Nguyen Tien Thinh, Ph.D.

Students: Nguyen Nguyen Phuong 1712726
Nguyen Dinh Thang 1752503

Ho Chi Minh City, April 2021

Commitment

We commit that the work in this dissertation was carried out following the requirements of the University's Regulations and has not been submitted for any other academic organizations. Except where indicated by specific reference in the text, the works are our own.

Ho Chi Minh City, December 2021

Acknowledgement

The completion of this study could not have been possible without the guidance and expertise of our supervisors. We would like to express our sincere gratitude to professor Nguyen An Khuong for his patience, motivation, and the immense amount of knowledge he led us to. No matter how many times we let him down because of our unprofessional result, he is still calm and continues to support us. We also want to thank the special supports from Mr.Phan Son Tu, who share with us valuable knowledge about the blockchain industry application while letting us work in his office.

Besides that, we would like to show our appreciation to families and friends, who have greatly supported and encouraged us in this thesis and university life.

Preface

On the Internet, most of the financial application apply public key cryptography to verify users' identity, manage their digital asset, authenticate and authorize users, etc. In a blockchain environment, this model is crucial to keep the system works, since there are no intermediaries nor the central database for management. However; the user has to keep track of a lot of key pairs and this is not efficient. To solve this problem, we use HD wallet, which can derive hundreds of child keys from a single master key-pair. Currently, there are already multiple protocols for HD wallet for Secp256k1 while those of ed25519 are neither well known nor well developed. In this thesis, we will try to analyze and build a complete HD wallet for ed25519.

Contents

Acknowledgement v

Preface 1

List of figures 5

List of tables 7

Chapter 1 INTRODUCTION 9
1.1 Overview . . . 10
1.2 Objectives . . . 11
1.3 Scope of the study . . . 11
1.4 Thesis structure . . . 12

Chapter 2 BACKGROUND 13
2.1 Blockchain Technology . . . 14
2.1.1 What is blockchain? . . . 14
2.1.2 How does it works? . . . 14
2.2 HD wallet . . . 15
2.2.1 What is a blockchain wallet? . . . 15
2.2.2 Category . . . 15
2.2.3 What is a HD wallet? . . . 17
2.3 Cryptography . . . 18
2.3.1 Elliptic curve cryptography - ECC . . . 18
2.3.1.1 Finite field & cyclic group . . . 18
2.3.1.2 Weierstrass curve - the general form of an elliptic curve . . . 20
2.3.1.3 Elliptic curve discrete logarithm problem - ECDLP . . . 22
2.3.1.4 Koblitz curve, case study: Secp256k1 . . . 22
2.3.1.5 Montgomery curve, case study: Curve25519 . . . 23
2.3.1.6 Edwards curve and the twisted Edwards curve, case study: Edwards25519 . . . 24
2.3.2 Digital signature with ECC . . . 26
2.3.2.1 The standard ECDSA . . . 26
2.3.2.2 Schnorr's signature scheme and EdDSA . . . 27
2.3.3 Comparison between ECDSA with Secp256k1 and Ed25519 . . . 29
2.3.3.1 Secp256k1 vs. Ed25519 . . . 29
2.3.3.2 ECDSA vs. Ed25519 . . . 30
2.3.4 Cryptographic hash function . . . 31
2.3.4.1 Secure Hash Algorithm with digest 512 bits (SHA-512) . . . 32
2.3.4.2 Hash-based message authentication codes (HMACs) . . . 32
2.3.4.3 Password Based Key Derivation Function (PBKDF2) . . . 33

Chapter 3 PROTOCOLS AND RELATED WORKS 35
3.1 Protocol Proposals . . . 36
3.1.1 Key Derivation . . . 36

3.1.1.1 BIP32 . . . 36
3.1.1.2 SLIP10 . . . 49
3.1.1.3 SLIP23 . . . 54
3.1.2 BIP39 - Random Password Generator . . . 60
3.1.3 BIP44 - Index Path for Multi-coin Wallet . . . 61
3.2 Related Works . . . 63

Chapter 4 SYSTEM DESIGN 65
4.1 The Library . . . 66
4.2 The Hierarchical Deterministic Web Wallet . . . 68
4.2.1 Cryptographic mechanism . . . 68
4.2.2 System Architecture . . . 72
4.2.2.1 User and Web Service . . . 72
4.2.2.2 User and Blockchain . . . 72
4.2.2.3 User and Address Service . . . 73
4.2.3 Information Privacy and Security concern . . . 75
4.2.4 Use Cases . . . 75
4.2.4.1 Use Case diagram . . . 75
4.2.4.2 Use case table . . . 76

Chapter 5 IMPLEMENTATION 85
5.1 The Library . . . 86
5.1.1 Library core . . . 86
5.1.2 Important dependencies . . . 88
5.1.3 Library API . . . 89
5.1.4 Documentation . . . 89
5.2 The Hierarchical Deterministic Web Wallet . . . 90
5.2.1 Ecosystem . . . 90
5.2.2 Address service . . . 91
5.2.2.1 Technologies . . . 91
5.2.2.2 API Provider . . . 91
5.2.3 Web Wallet Service . . . 96

Chapter 6 TESTING 101
6.1 The Library . . . 102
6.1.1 Testcases . . . 102
6.1.2 Security Discussion . . . 104
6.2 The Hierarchical Deterministic Web Wallet . . . 106
6.2.1 Security Discussion . . . 106

Chapter 7 CONCLUSION AND DISCUSSION 109
7.1 Results . . . 110
7.2 Limitations . . . 110
7.3 Future works . . . 111

Bibliography 113

List of Figures

2.1 Montgomery ladder . . . 23
2.2 Hash function . . . 32

3.1 Example of BIP32 key tree with depth 3 . . . 37
3.2 Example of the extended private key . . . 38
3.3 Master Extended Key generation Process . . . 40
3.4 Normal Extended Private Key generation process . . . 42
3.5 Normal Extended Public Key generation process . . . 43
3.6 Hardened Extended Private Key generation process . . . 45
3.7 Relation in BIP32 normal derivation protocol . . . 47
3.8 Master Extended Key generation Process . . . 50
3.9 Hardened Extended Public Key generation process . . . 51
3.10 Multiply process of ed25519 . . . 52
3.11 Master key generation process . . . 55
3.12 Private keypair derivation process . . . 57
3.13 Public keypair derivation process . . . 58
3.14 Random secret seed generation process . . . 61
3.15 Example path of BIP44 . . . 62
3.16 Example tree of wallets from given path . . . 62

4.1 General library structure . . . 66
4.2 Testcase components . . . 67
4.3 Example of Web Wallet flow . . . 69
4.4 HD Web Wallet tree . . . 70
4.5 Between users and Wallet Provider . . . 72
4.6 Between users and Blockchain network . . . 73
4.7 Between users and Address Service . . . 73
4.8 Request Data . . . 74
4.9 General JSON object of a Wallet . . . 74
4.10 Usecase diagram . . . 76

5.1 Typescript library structure . . . 87
5.2 COMPONENTS JSON example . . . 88
5.3 Different implementation in Montgomery Ladder . . . 88
5.4 Overview of entire system . . . 90
5.5 Homepage design . . . 97
5.6 Create mnemonic . . . 97
5.7 Import mnemonic . . . 98
5.8 Main page . . . 99
5.9 List functionality . . . 99

6.1 Testcase results with mocha . . . 103
6.2 Test airdrop with Solana . . . 104
6.3 Test balance with Solana . . . 104

6.4 Test transaction with Solana 104
6.5 Transaction detail on Solana Explorer 105
6.6 Coins transfered details 105

List of Tables

4.1 Use case List 77
4.2 Use case: Create new wallet. 78
4.3 Create new child wallet. 79
4.4 Delete Wallet from the browser. 80
4.5 Search address. 81
4.6 Modify addresses in the database. 82
4.7 Make transaction. 83

1

Introduction

In this chapter, we are going to discuss the overview of the wallet we are creating. Thus, we are going to present the objective, scope, and structure of this thesis.

Contents

1.1 Overview **10**
1.2 Objectives **11**
1.3 Scope of the study **11**
1.4 Thesis structure **12**

1.1 Overview

The Hierarchical Deterministic (HD) secret key derivation and transfer protocol allows creating child keys from parent keys in a hierarchy. Crypto wallets using the HD protocol are called HD wallets. HD wallet is a crypto software/hardware wallet that is used in a blockchain transaction. Some people call it a digital wallet, but that is not entirely correct. A digital wallet (e-wallet) such as Momo and Paypal can hold your digital assets (government-issued currency) and link to your credit/debit card. Your assets in these wallets present how much fiat money you have that is controlled by the central bank. Fiat money gives the authorities greater control over the economy because they can print money out of thin air. One problem of fiat money is that governments can cause hyperinflation if they create too much of it. On the other hand, HD wallets hold the cryptocurrencies (tokens) of the specific blockchain, a decentralized database in which all financial transactions that have ever been carried out are stored. Blockchains demonstrate some characters for the currency, where the number of native tokens is limited and minted to the genesis block. The amount of new coins is automatically reduced by years, so the inflation rate will also decrease.

The money can be quickly transferred between a digital wallet and your bank account or card. The digital wallet secures the connection using public-key cryptography. On the contrary, an HD wallet doesn't hold any of your coins. Your holdings live on the blockchain. It's a collection of private and public keys. You can only be accessed to your assets using a private key. The keys prove the ownership of your digital money and allow you to make transactions. All your balances and transactions are public to the networks, and everyone with the internet can preserve them.

You can directly check your crypto assets without entering your PIN (bank account) or signing in (digital wallet). To use a digital wallet, you have to pay for the bank services, create an account for a digital wallet, then link your account and the bank to be able to use your money across them. However, there is no link between banks/brokerages and your crypto wallet, i.e., no intermediaries. If you want to have crypto coins, you have to either mine them from the blockchain network or trading with your money. This is not a downside, and rather, this is the strong point of the crypto wallet.

You can make a transaction directly, from your crypto wallet to another wallet, without a trusted third party. The transaction will be published to the blockchain network for nodes to confirm. All the process takes in terms of minutes, no matter how far it is from you to the receiver. With some lightning-fast consensus like Proof of History blockchain, it took less than 10 seconds. In the case of digital wallets or bank accounts, you have to wait for days before your transaction is confirmed, and the money is transferred when you decide to transfer through to another country or cross-bank. The bank does have a solution for this problem, but you have to pay a big amount for their service no matter how much you spend. That is why rather than the traditional stock markets, people are more likely to invest in the decentralized system since they have to stay home due to the COVID pandemic. The cryptocurrency still has some more advantages to traditional finance, but for the scope of our thesis, we won't include them here.

1.2 Objectives

We focus on the security and availability of the crypto wallet. In the blockchain system, there are no intermediaries or brokerages, which means all of the responsibility of your crypto asset is on you. The only way to access your crypto assets is your crypto wallet. If you lose your wallet deliberately (no backup, key leakage,..) or not (hacked, program error,...), you take it all. It's too risky and unfair if the crypto wallet is insecure and inefficient. At least, the tradeoff by the CIA triad must be acceptable.

The release of Curve25519 from the safecurves project [1], which has an incredible speed in ECDH and is applicable in the digital signature ed25519 [2], marks a new trend in ECC. The standard ECDSA has had a flaw since 2009 [3] so the need of a higher-security signature is vital. Also, the expiration of Schnorr's signature patent in 2008 allows ed25519, a digital signature based on Schnorr's signature and the twisted Edwards curve, to be publicly used. The advantages of ed25519 are comprised from the RFC 8032 document [4]:

1. High performance in various platforms.
2. Generating a random number for each signature is not needed.
3. Resilience against side-channel attacks.
4. Complete formulas (e.g. addition law): the formula works for all points in the curve.
5. Hash collision resistance.
6. 128-bit security level.
7. Small 32 byte keys and 64 byte signatures

This is the motivation for our thesis, to develop an HD wallet for ed25519. We will explain the benefits of the HD wallet, along with implementing and analyzing the ed25519 of HD wallet's usage.

1.3 Scope of the study

In this thesis, we study elliptic curve cryptography and its digital signature algorithm to point out the difference between ECDSA and ed25519 in a cryptanalytical way. For implementation, we look for open-source protocols and try to make a prototype of a HD wallet for ed25519. We choose Solana Blockchain as our testing environment because Solana uses ed25519 for their signatures.

1.4 Thesis structure

The contents of our thesis are presented in seven chapters, namely:

Chapter 2 introduces and discusses the background knowledge of this thesis, including the necessary knowledge of the HD Wallet and Blockchain network, group and field for Elliptic Curve Cryptography, Schemas of ed25519 and its performance as well as the important Cryptography functions.

Chapter 3 discusses the protocols of the HD Wallet, the related works on implemented library and some open source wallet that used in the industry.

Chapter 4 shows our system design for the HD Wallet for ed25519, how did we solve the challenge on key derivation schema. We also improve the wallet by letting it hold tokens of different Blockchains (expanding with secp256k1 curve).

Chapter 5 describes how we actually implement the HD Wallet, including the discussion on library that we used.

Chapter 6 summarizes the testing results of our application.

Chapter 7 concludes our work. Also, we analyze the drawbacks and how to prevent it with the spaces for improvements to the thesis.

2

Background

In this chapter, we introduce the foundation knowledge of the thesis, including the Blockchain Technology, Cryptography, and Hierarchical Deterministic Wallet (HD wallet)

Contents

2.1 Blockchain Technology **14**
2.1.1 What is blockchain? 14
2.1.2 How does it works? 14
2.2 HD wallet **15**
2.2.1 What is a blockchain wallet? 15
2.2.2 Category 15
2.2.3 What is a HD wallet? 17
2.3 Cryptography **18**
2.3.1 Elliptic curve cryptography - ECC 18
2.3.1.1 Finite field & cyclic group 18
2.3.1.2 Weierstrass curve - the general form of an elliptic curve . . 20
2.3.1.3 Elliptic curve discrete logarithm problem - ECDLP 22
2.3.1.4 Koblitz curve, case study: Secp256k1 22
2.3.1.5 Montgomery curve, case study: Curve25519 23
2.3.1.6 Edwards curve and the twisted Edwards curve, case study: Edwards25519 24
2.3.2 Digital signature with ECC 26
2.3.2.1 The standard ECDSA 26
2.3.2.2 Schnorr's signature scheme and EdDSA 27
2.3.3 Comparison between ECDSA with Secp256k1 and Ed25519 29
2.3.3.1 Secp256k1 vs. Ed25519 29
2.3.3.2 ECDSA vs. Ed25519 30
2.3.4 Cryptographic hash function 31
2.3.4.1 Secure Hash Algorithm with digest 512 bits (SHA-512) . . 32
2.3.4.2 Hash-based message authentication codes (HMACs) 32
2.3.4.3 Password Based Key Derivation Function (PBKDF2) . . . 33

2.1 Blockchain Technology

2.1.1 What is blockchain?

Blockchains are immutable digital ledger systems implemented in a distributed fashion (i.e., without a central repository) and usually without a central authority. The definition of blockchain was introduced to the world by a person (or a group of people) under the name Satoshi Nakamoto on October 31, 2008. It was applied to enable the emergence of a "purely peer-to-peer (no financial institution or third party) electronic cash" named Bitcoin where transactions take place in a distributed system. Satoshi did not invent the blockchain, and the Bitcoin blockchain is not the first chain that was ever created.

The word "blockchain" or "block" and "chain" wasn't used back then. Only when it was known in Satoshi Nakamoto's Bitcoin paper did the term "chain" of "blocks" become a representation for this technology. Later on, the community used that word for Nakamoto's invention. Bound to the emergence of Bitcoin and cryptocurrency, a concise description of blockchain technology is provided by NIST: Blockchains are distributed digital ledgers of cryptographically signed transactions that are grouped into blocks. Each block is cryptographically linked to the previous one (making it tamper evident) after validation and undergoing a consensus decision. As new blocks are added, older blocks become more difficult to modify (creating tamper resistance). New blocks are replicated across copies of the ledger within the network, and any conflicts are resolved automatically using established rules.

2.1.2 How does it works?

To make a blockchain work, it needs a blockchain network formed by many master nodes, a consensus protocol, and the distributed and immutable blockchain ledger. Every master node has a copy of the ledger, and most of the master nodes contribute to "validate and create" the next block to be added to the current chain to create a new ledger. By validation, that means the master node acts as a validator and confirm if a transaction is valid: is the owner of the transaction authorized, is this transaction not a double-spend one, or does this transaction follow the rules, etc. By creation, the master node each validate and put multiples valid transactions into a block, then thanks to the consensus protocols to choose whose block will be added to the chain to form a new ledger. The consensus protocol is an agreement protocol, that helps many master nodes to make a general approval on which blockchain ledger is used for the whole network and dispose of the malicious one.

Suppose that Alice wants to send 10 BTC to Bob, she uses her blockchain wallet to make a transaction, sign it, and then send the signed transaction to the blockchain network. The transaction then waits to be validated and put in a block by the master node. Now inside the blockchain network, the transaction is broadcasted to most of the master nodes to queue. Then each node can choose a number of transactions that are going to be validated and put into a block, which may contains Alice's transaction. After that, the blockchain network works on agreement which is the appropriate block to be chained with the current

blockchain using consensus protocol. Bitcoin's consensus is Proof-Of-Work, in which the master node competes against each other by solving a cryptographic hash puzzle to gain the right to add their node to the chain. The puzzle must be too hard to break but fast to verify, so that the network can verify one's work and approve the new block. The first node to find out the nonce will 'mine' the block, put the validated transactions into a block with an extra transaction called coin-base transaction. The coin-base transaction is the reward for the master node, and works as incentives to encourage the node to continue to contribute to mine new blocks faithfully. That's why the term miner is born and being more common to end-users rather than master node. Once a block is mined, the node will broadcast the blockchain appended with the new block to the blockchain network. If that block contains Alice's transaction, her transaction is finished and Bob now has 10 BTC in his blockchain wallet.

2.2 HD wallet

2.2.1 What is a blockchain wallet?

A blockchain wallet, sometimes referred to as a cryptocurrency wallet or crypto wallet, is a program that allows you to "store", send and receive digital currencies. Since cryptocurrency doesn't exist in any physical form, your wallet doesn't actually hold any of your coins. Instead, it tracks the transactions you made, which are stored in blockchain, and then infers your balance. Thereby, blockchain is an essential component of a blockchain wallet.

Instead of holding physical coins, a wallet has a public key and a private key. Public key is a long sequence of letters and numbers that can form the wallet address. With this, other people can send money to your wallets. It's similar to a bank account number in which it can only be used to send money to an account. Private key is used to access the funds stored in the wallet. With this, you can control the funds tied to your wallet's address. It works like your PIN number, you should keep it 100% secret and secure. However, it's worth noting that not all wallets give you sole ownership of your private key, which essentially means that you don't have full control over your coins. As well as storing your public and private keys, crypto wallets interface with the blockchains of various currencies so that you can check your balance and send and receive funds.

2.2.2 Category

Now that you know what it is, let's take a closer look at the five different types of wallets available, each with its own advantages and disadvantages in terms of security, ease of use, convenience and a range of other factors. The most common type of wallet out there, desktop wallets are downloaded and installed on your computer. Easy to set up and maintain, most are available for Windows, Linux and Mac, although some may be limited to a particular operating system. Many cryptocurrencies offer a desktop wallet specifically designed for their coin.

Desktop wallets provide a relatively high level of security since they're only accessible from the machine on which they're installed. The biggest disadvantage is that they also rely on you to keep your computer secure and free of malware, so antivirus and anti-malware software, a strong firewall and a common-sense approach to security are required to keep your coins safe. Most desktop wallets will provide you with a long string of words upon installation. These words are known as your recovery seed or sentence and map with your private key, so it's important to store them somewhere safe in case your computer dies or you need to format the operating system and re-install your desktop wallet. Some popular desktop wallets: Electrum, Exodus, Copay.

Mobile wallets are fairly similar to desktop wallets, with the obvious difference being that they run as an app on your smartphone. Mobile wallets feature many of the same advantages and disadvantages as desktop wallets, with your private key stored on your device. Smartphone wallets are often easier to use compared to their desktop counterparts and include the ability to scan other wallet addresses for faster transactions. They also make it simpler to access your coins on the go and use cryptocurrency as part of everyday life. You will need to be extra careful about losing your smartphone, though, because there's a risk that anyone who has access to your device might also have access to your funds. Choosing an app that allows you to back up your wallet with a 12- or 24-word passphrase is a good idea. Popular mobile wallets: Jaxx, Coinomi, Edge.

Online wallets (most often provided by exchanges but sometimes offered by third parties) are connected to the Internet and are generally the easiest to set up and use. Most only require an email address and a password to create an account, and web wallets are usually designed to provide a simple and straightforward user experience. The biggest advantages to online wallets are that they can't be lost and that they're accessible from any computer with an Internet connection. However, being online is unfortunately also their biggest disadvantage. Because some platforms maintain the wallets of thousands of users, they can become hot targets for hackers. It's also important to check whether the wallet you choose lets you retain complete control of your private keys or whether they're owned by the wallet provider. Popular web wallets: blockchain.info, MyEtherWallet, Coinbase.

Hardware wallets add another layer of security by keeping your private key on a USB stick or specially designed piece of hardware. They allow the user to plug the USB stick into any computer, log in, transact and unplug – so while transactions are carried out online, your private key is stored offline and protected against the risk of hacking. As a result, hardware wallets are widely considered to offer the most secure storage option. The biggest disadvantage of hardware wallets is that they'll cost you. Prices vary depending on the model you choose, but they generally cost upwards of $150. You also need to keep the device safe, but if you do lose your hardware wallet, the device itself is PIN-protected and there are usually other protective measures in place to help you recover your funds. Popular hardware wallets: Ledger Nano X, Ledger Nano S, TREZOR, KeepKey.

Paper wallets take the concept of entirely offline keys used for hardware wallets to the next logical step: simply print out your public and private keys and use that piece of paper as your wallet. As secure as they are, paper wallets are also complex and quite confusing for beginners. They're typically only used by advanced users who want a high level of security. To transfer money to a paper wallet, you use a software wallet (any

of the above mentioned) to send money to the public key printed on the sheet of paper. Most often, this is printed as a QR code for easy scanning. To transfer money from the paper wallet to someone else, you would first need to transfer money to a software wallet (by manually entering the private key into the software), and then transfer money from the software wallet to the recipient as usual. Popular paper wallets: Bitaddress.org, WalletGenerator.net.

As you're researching and comparing a range of wallets, you'll probably come across the terms "hot wallet" and "cold wallet", or perhaps the concept of "cold storage". So, what does temperature have to do with crypto storage? A wallet is hot when it's connected to the Internet. Nothing on the Internet is 100% secure, so funds kept in a hot wallet are always at a slight risk of theft or loss from software bugs or hackers. A wallet is cold when it's safely offline and can't be deliberately or accidentally compromised over the Internet.

From an economic perspective, the wallet can be custodial (requires KYC) or non-custodial (doesn't require KYC). Custody means that your wallet is controlled by a third party, and they need your personal information to authenticate you whenever you want to use your wallet. An example is the Binance "wallet", or actually, the application provided by Binance. Your private keys are managed by them, which resembles a bank managing your account. Also, you have to provide your ID number, phone number, and email so that every time you want to make a transaction, the application can authenticate the right owner of the wallet. With that, you put faith in a third party to secure your funds and return them if you want to trade or send them somewhere else. While a custodial wallet lessens your responsibility, it requires you to trust in the custodian that holds your funds, which is usually a cryptocurrency exchange. On the other hand, non-custodial crypto wallets give you complete control of your keys and therefore your crypto assets. While some people store large amounts of crypto on exchange accounts, many feel more comfortable with a non-custodial wallet, which eliminates a third-party between you and your crypto. Apart from that, based on an engineering prespective, we cared about how the key is managed. We re-categorize the wallet into three main groups: hardware, software and hybrid. Hardware wallets are only connected to the internet when you need to make a transaction and you have no information about the secret keys, whereas software wallets are connected to the internet and store your secret keys. Hybrid is the combination of both hardware and software wallet, that your secret keys are stored in a secured space and are managed by the software. We will develop on a hybrid wallet as we need some information about the secret keys and need to manage the keys.

2.2.3 What is a HD wallet?

A hierarchical deterministic wallet allows you to generate and manage multiple child wallets. The HD wallet architecture consists of a pair of master private key and master public key and the management software. The software has 3 functionalities: manage the keys, check assets and transfer/deposit assets. For the keys management, the software can store and derive the child key using the CKD function to create a corresponding sub wallet for the native asset of a blockchain. By that way, we can hold multiple types of cryptocurrencies and perform cross-chain actions.

Why an HD wallet? Essentially, the HD wallet helps the user to manage multiple key

pairs efficiently. As mentioned above, it is not practical for the wallet to keep track of all generated key pairs, for which each of them is used to sign a transaction. The more transactions the users make, the more key pairs they have to keep track. Instead of backing up every generated key pair, now you only need to backup the master key pair in case of HD wallet. If one of the key pairs is leaked, your wallet and crypto assets are gone. Secondly, HD wallet offers stronger privacy than non-HD wallet because key pairs are derived automatically for every transaction. Because Bitcoin and other blockchains are public ledgers, address balances are public knowledge. However, as you have multiple addresses, others may not know which addresses are linked to you and which aren't - provided that you haven't shared your extended public key. Your extended public key can show all of your balances, and therefore should never be shared. Beyond privacy, HD wallet offers enhanced security because every transaction received is on a different address. Someone would need multiple private keys to access your wallet's multiple crypto balances. So as long as you haven't shared your extended private key, your funds should be secure. And most HD wallets make it difficult, if not impossible, to share confidential information such as your master key pair.

2.3 Cryptography

2.3.1 Elliptic curve cryptography - ECC

Elliptic curve cryptography is a branch of public key cryptography, a.k.a asymmetric-key cryptography. An elliptic curve is defined by a curve equation built on a finite field F for coordinates. In order to understand elliptic curves, knowledge about group theory and number theory are required. In the next section, we will introduce the finite field and an interesting characteristic of the group that the elliptic curve used. Then, we explain the elliptic curve cryptography, the ECDLP and some case studies that are necessary for our thesis.

2.3.1.1 Finite field & cyclic group

A finite field, sometimes called Galois field, is a set with a finite number of elements. Roughly speaking, a Galois field is a finite set of elements in which we can add, subtract, multiply and invert. Before we introduce the definition of a field, we first need the concept of a simpler algebraic structure, a group

Definition: A group is a set of elements G together with an operation $\circ$ which combines two elements of G. A group has the following properties

- The group operation $\circ$ is closed. That is, for all $a, b \in G$, it holds that $a \circ b = c \in G$.
- The group operation is associative. That is, $a \circ (b \circ c) = (a \circ b) \circ c$ for all $a, b, c \in G$.
- There is an element $1 \in G$, called the neutral element (or the identity element), such that $a \circ 1 = 1 \circ a = a$ for all $a \in G$.

- For each $a \in G$ there exists an element $a^{-1} \in G$, called the inverse of a, such that $a \circ a^{-1} = a^{-1} \circ a = 1$ for all $a \in G$.
- A group G is an Abelian group if, furthermore, $a \circ b = b \circ a$ for all $a, b \in G$.

So, a group is set with one operation and the corresponding inverse operation. If the operation is addition, the inverse operation is subtraction; and if the operation is multiplication, the inverse operation is division or multiplication with the inverse element. What we call a field is a group with all four basic arithmetic operations. A field is defined as:

Definition: A field F is a set of elements with the following properties.

- All elements of F form an additive group with the group operation "+" and the neutral element 0.
- All elements of F except 0 form a multiplicative group with the group operation $\times$ and the neutral element 1.
- When the two group operations are mixed, the distributivity law holds, i.e., for all $a, b, c \in F : a \times (b + c) = (a \times b) + (a \times c)$.

In the elliptic curve, we are interested in fields with a finite number of elements, the finite field. The number of elements in the field F_p is called the order (cardinality) of the field, denoted by $|F_p|$. A field with order m only exists if m is a prime power, i.e., $m = p^n$, for some positive integer n and a prime integer p. p is called the characteristic of the finite field, denoted as $char(p)$. A finite field is denoted as $GF(p^n)$ or F_{p^n}. The most intuitive examples of finite fields are fields of prime order with $n = 1$. Elements of the prime field F_p can be represented by integers $0, 1, \ldots, p-1$. The two operations of the field are modular integer addition and integer multiplication modulo p. With $n > 1$, the field is called extension field, the elements are polynomials $a_{n-1}X^{n-1}+a_{n-2}X^{n-2}+\ldots+a_1X+a_0$, where the coefficient a is in F_p. The arithmetic operations in an extension field are polynomial addition modulo p with the coefficients and polynomial multiplication modulo $P(x)$. $P(x)$ is called the irreducible polynomials, which acts like 'a prime' of the polynomials.

Elliptic curve usually works in the prime field, to gain a finite number of curve points. The points on an elliptic curve over F_p also belong to a group, called cyclic group. A cyclic group is a multiplicative group $F_p^* = gcd(x, p) = 1 | x \in 1, 2, ..., p-1$, where p is a prime and contains at least one special element α. The element α is called primitive element or generator, such that $ord(\alpha) = |(F_p)^*|$, where the order of α ($ord(\alpha)$) is the smallest positive integer k satisfies $a^k = a \circ a \circ \ldots \circ a$ (k times) $= 1$. The multiplies of α generates all the elements in the F_p^* , and they repeat cyclically similar to a modulo congruence. E.g.: For a F_{11}^* with order of 10, the generator α can be 2 since.

$\alpha^1 \mod 11 = 2$ $\alpha^2 \mod 11 = 4$
$\alpha^3 \mod 11 = 8$ $\alpha^4 \mod 11 = 5$
$\alpha^5 \mod 11 = 10$ $\alpha^6 \mod 11 = 9$
$\alpha^7 \mod 11 = 7$ $\alpha^8 \mod 11 = 3$
$\alpha^9 \mod 11 = 6$ $\alpha^{10} \mod 11 = 1$
$\alpha^{11} \mod 11 = 2$ $\alpha^{12} \mod 11 = 4$ $\alpha^{13} \mod 11 = 8$ And so on...

Note that there can be more than one generator for a cyclic group, but a group only needs one element α to be cyclic. It is similar to a finite field, with an additional condition, that is the element α. This behavior will also work for an elliptic curve introduced in the next section, where the scalar multiples of a point on an elliptic curve are repeating cyclically.

2.3.1.2 Weierstrass curve - the general form of an elliptic curve

About elliptic curves, there are curve types and equation types. An elliptic curve E defined over a field F_p is given by the equation

$$y^2 + a_1xy + a_2y \equiv x^3 + a_3x^2 + a_3x^2 + a_4x + a_5 \quad (mod \quad p)$$

where the coefficients a_i are elements in the field F_p. The set of points in $E(F_p)$ satisfied the equation and the point of infinity Θ. The elliptic curves that satisfy the equation above are called Weierstrass curves, i.e., a curve type. Consider polynomials over F_p, where a characteristic $char(p) \neq 2, 3$, the equation above can be reduced into a short Weierstrass form

$$y^2 \equiv x^3 + a_1x + a_2 \quad (mod \quad p), \text{ where } 4a_1^3 + 27a_2^2 \not\equiv 0 \mod p$$

The set of all pairs $(x, y) \in F_p$ together with an imaginary point at infinity Θ satisfying the equation is the set of points on the elliptic curve, with the condition to exclude singular curve. The scalar multiplication, addition and doubling laws allow arithmetic operations in the elliptic curve group laws. The group laws for an elliptic curve

- The elements of the group are the points of an elliptic curve.
- The neutral element is the point at infinity ∞, denoted Θ.
- The inverse of a point P is the one symmetric about the x-axis.
- Addition is given by the rule: given three aligned, non-zero points P, Q, and R, their sum is $P + Q + R = \Theta$.
- The inverse or negative point of a point $P(x, y)$ is defined as $-P(x, y) = P(x, -y)$ such that $P + (-P) = \Theta$.

The set of points on the elliptic curve with the point Θ has cyclic subgroups. Under certain conditions all points on an elliptic curve form a cyclic group. Each point on an

elliptic curve is a generator for its cyclic subgroup of point on E. As mentioned above, an element α now is a point on the curve, it may not generate all point on a curve but its generated group is a cyclic subgroup, i.e., part of all points on the curve. The ECC works around this special case in ECDLP.

The point arithmetic operations defined in the group law are in geometric representation. To work with F_p, point addition laws need to be calculated in an algebraic way. The addition laws for a general curve, given $P(x_1, y_1), Q(x_2, y_2), R(x_3, y_3)$, are

- $P + \Theta = \Theta + P = P$
- $\Theta + \Theta = \Theta$
- $P + Q = \Theta$ if $Q = -P$ or $P = -Q$
- If $P \neq Q$, $P + Q = R$, the coordinates of R can be computed as
  - Compute $s = (y_2 - y_1)(x_2 - x_1)^{-1} \mod p$
  - Compute $x_3 = s^2 - x_1 - x_2 \mod p$, and $y_3 = s(x_1 - x_3) - y_1 \mod p$
- If $P = Q$, $P + Q = P + P = 2P = R$, the coordinates of R can be computed as
  - Compute $s = (3x_1^2 + a_1(2y_1)^{-1} \mod p$
  - Compute $x_3 = s^2 - 2x_1 \mod p$, and $y_3 = s(x_1 - x_3) - y_1 \mod p$

Here, the addition law is incomplete, since we can't add the same point twice, we have to use the point doubling formula. The Weierstrass form is the general form of an elliptic curve, any curve can be expressed in a Weierstrass form. There are also other types of curves such as the Koblitz curve, Montgomery curve and (twisted) Edwards curve, which will be presented in the following sections. Before that, most of the elliptic curves need a domain parameter as follow.

- p: The prime that specifies the size of the finite field
- a, b: The coefficients of the elliptic curve equation in (short) Weierstrass form
- N: the order of the elliptic curves ($\#E$)
- P: a point on the elliptic curve
- G: The base point that generates our subgroup
- n: The order of the subgroup (which is the order of the basepoint G). n must be prime
- h: The cofactor of the subgroup such that with $n(hP) = 0$, $G = hP$. In other words, $h = N/n$

Also, in practice, a point $P(x, y)$ can be compressed to an integer (y's LSB in the first byte $\parallel$ x). The first byte of y can be $0x02$ for even y, or $0x03$ for odd y(negative y) with the given x. For instance, a point $P(6, 8)$ can be compressed to the integer $0x026$ in hexadecimal format. The actual number is almost 256-bit long, this is just a simple example.

2.3.1.3 Elliptic curve discrete logarithm problem - ECDLP

Given an elliptic curve E, we consider a primitive element P and other element T, the discrete log problem is finding an integer d, where

$d \in [1, \#E]$ such that $P+P+\ldots+P$ ("+" d times) $= dP = T$. The ECDLP is considered to be infeasible if the elliptic curve parameters are carefully chosen [Selecting elliptic curve for cryptography: an efficiency and security analysis]. In ECC, we interpret the integer d as the private key, T as the public key, and the exact number of point points E of a curve. The number of points on the curve can be counted by trying all the possible values for x in F_p, but this is not efficient if the p is a large prime. Haskel's theorem only gives an upper and lower bound for number of point on an elliptic curve parameters are carefully chosen [Selecting elliptic curve for cryptography: an efficiency and security analysis]. In ECC, we interpret the integer d as the private key, T as the public key, and the exact number of point points $\#E$ of a curve. The number of points on the curve can be counted by trying all the possible values for x in F_p, but this is not efficient if the p is a large prime. Haskel's theorem only gives an upper and lower bound for number of point on an elliptic curve

$$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}$$

In order word, find the exact numbers of points on a curve is computationally difficult. Luckily, there's a faster algorithm for computing the order: Schoof's algorithm. We won't enter the details of the algorithm - what matters is that it runs in polynomial time, and this is what we need.

2.3.1.4 Koblitz curve, case study: Secp256k1

The elliptic curve Secp256k1 is a Koblitz curve, which has the equation

$$y^2 \equiv x^3 + 7 \pmod{p}$$

with the param standards Secp256k1, where $p = 2^{256} - 232 - 29 - 28 - 27 - 26 - 24 - 1, h = 1$. Its finite field is defined over the finite field F_2, with the neutral element as the point at infinity Θ. The addition law for Secp256k1 are Secp256k1 documentaton and improved in Koblitz curve cryptosystem .The addition law for the Koblitz curve is "incomplete".

2.3.1.5 Montgomery curve, case study: Curve25519

Daniel J. Bernstein originally discovered the Curve25519, in the form of the Montgomery curve, has the equation

$$By^2 \equiv x^3 + A^2 + x \pmod{p}, \text{ where } B(A^2 - 4) \neq 0, B \neq 0, A \neq \pm 2$$

with $A = 486662x, B = 1$, over the prime field defined by the prime number $p = 2^{255} - 19$, the base point has x-coordinate $= 9$, the cofactor $h = 8$ and the subgroup order is $n = 2^{252} + 27742317777372353535851937790883648493$. The neutral element is the point at infinity Θ, and the negation is $-P(x, y) = P(x, -y)$. The Montgomery curve has a special algorithm to compute scalar multiples of points in constant-time, called Montgomery ladder. Let the scalar integer is a in binary form and the point is P, the algorithm works as follow

Montgomery ladder

```
def cswap(bit, R, S):  # constant time conditional swap
        dummy = bit * (R - S)   # 0 or R - S
        R = R - dummy    # R or R - (R - S) = S
        S = S + dummy    # S or S + (R - S) = R
        return (R, S)

a = 44444   # our super secret scalar. No, not that one.
l = max    # some maximum bit length, matching order(P)
A = a.digits(2,padto = l) # fill with 0 to lenght l
P0 = 0     # so initial doublings don't matter, 0=0P
P1 = P    # difference P1 - P0 = P
for i in range(l-1,-1,-1): # fixed-length loop
  (P0, P1) = cswap(A[i], P0, P1)  # see above
  P1 = P0 + P1  # addition with fixed difference
  P0 = 2P0  # double point for which bit is set
  (P0, P1) = cswap(A[i], P0, P1) # swap back, can merge
print(P0)
```

This uses one doubling and one addition per bit. No dummy additions.

Tanja Lange Elliptic-curve cryptography VIII 3

Figure 2.1: Montgomery ladder

- Assign initial value for $P_0 = 0$ and $P_1 = P$ so initial doublings don't matter ($P_0 = 0P$, $2 * 0P = 0P$) and P_1 is the difference $P_1 - P_0 = P$
- Assign initial value for $P_0 = 0$ and $P_1 = P$ so initial doublings don't matter ($P_0 = 0P, 2 * 0P = 0P$) and P_1 is the difference $P_1 - P_0 = P$
    - If the bit is 0 compute $P_1 = P_0 + P1$ and $P_0 = 2P_0$

  - Else the bit is 1 $P_0 = P_0 + P_1$ and $P_1 = 2P_1$
- The final result of $[a]P$ is P_0

Montgomery ladder always performs doubling and adding in the algorithm, allowing the time cost to be constant. It works since the Montgomery curve always has the point (0, 0) of order 2 (assignable to P_0). Also, the addition laws for Montgomery, given $P(x_1, y_1), Q(x_2, y_2), R(x_3, y_3)$, are

- $P + \Theta = \Theta + P = P$
- $\Theta + \Theta = \Theta$
- $P + Q = \Theta$ if $Q = -P$ or $P = -Q$
- If $P = Q$ and $x_1 = x_2 \neq 0$, the coordinates of R can be computed as
  - Compute $s = (3x_1^2 + 2Ax_1 + 1)(2By_1)^{-1} \mod p$
  - Compute $x_3 = Bs^2 - A - 2x_1 \mod p$, and $y_3 = s(x_1 - x_3) - y_1 \mod p$
- If $P = Q$, $P + Q = P + P = 2P = R$, the coordinates of R can be computed as
  - Compute $s = (y_1 - y_2)(x_1 - x_2)^{-1} \mod p$
  - Compute $x_3 = Bs^2 - A - x_1 - x_2 \mod p$, and $y_3 = s(x_1 - x_3) - y_1 \mod p$

2.3.1.6 Edwards curve and the twisted Edwards curve, case study: Edwards25519

The Curve25519 can be transformed to twisted Edwards form (from here, we refer to Curve25519 in twisted Edwards form as Edwards25519). The curve Edwards25519 has the equation

$$ax^2 + y^2 \equiv 1 + dx^2y^2 \quad (mod \quad p)$$

with the same prime field p and the cofactor h. Edwards25519 has some additional parameters:

- $b = 256$, the bit length of the coordinate satisfies $2^{b-1} > p$
- A non-square d : $-121665/121666$
- A non-zero a: if $a = 1$, it's a normal Edwards curve, else it's a twisted Edward curve ($a = -1$ for Edwards25519)

- An odd prime L (same as n in a general elliptic curve), where $h \times L = \#E$: $2^{252} + 27742317777372353535851937790883648493$ ($\tilde{2}53$ bits)
- The basepoint B: $B(x, 4/5) \neq (0, 1)$ with positive x

The neutral element is $\Theta = (0, 1)$, and the negation is $-P(x, y) = P(-x, y)$. Note that Curve25519 and Edwards25519 are birational equivalence (proved in High-speed high-security signatures). Therefore; Edwards25519 also supports Montgomery ladders. The Edwards25519 uses a point encoding scheme rather than a standard point compression. A point $P(x, y)$ is presented as a 32-octet string, interpreted as a little-endian integer. To encode a point $P(x, y)$

- Encode the y-coordinate as a little-endian string of 32 octets. The MSB of the last octet is always zero.
- Copy the LSB of the x-coordinate to the MSB of the last octet string. Resulting in an encoded point with 256 bits length, with the 256-bit to be the LSB of x, called x_0. To decode a point, it takes more works than encoding
- Clearing the x_0 - the MSB of the octet string to obtain y. If $y \geq p$, decoding fails
- x can be retrieved by computing $x^2 = (y^2 - 1)/(dy^2 + 1) \mod p$
- Denote $u = (y^2 - 1), v = (dy^2 + 1)$, w is the candidate root. Calculate $w = uv^3(uv^7)^{(p-5)/8} \mod p$
- There are three cases
  - If $vw^2 = u \mod p, x = w$
  - If $vw^2 = -u \mod p, x = w \times 2^{(p-1)/4}$
  - Otherwise, no square root exists. Decoding fails
- If $x = 0, x_0 = 1$, decoding fails. Else, if $x_0 = x \mod 2$ then $x = x$, if not then $x = p - x$

The main reason for point encoding is for security (same as point compression) performance (will be explained in Ed25519). The addition laws for twisted Edwards curve are always, given $P(x_1, y_1), Q(x_2, y_2)$:

$$(x_1, y_1) + (x_2, y_2) \equiv \left(\frac{x_1 y_2 + x_2 y_1}{1 + d x_1 x_2 y_1 y_2}, \frac{y_1 y_2 - a x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \right) \quad (mod \quad p)$$

which means that the addition laws for twisted Edwards are complete (no exception cases, as long as the parameters a and d follow their condition above).

2.3.2 Digital signature with ECC

2.3.2.1 The standard ECDSA

ECDSA is a standard Digital Signature Algorithm for elliptic curve cryptography. The necessary parameters are defined above, with a hash function H and the message M needs to be signed. Denote that k is the randomly generated private key in the range $[1, n-1]$, and K is the public key by a scalar multiplication $K = [k]G$. To sign a message, do the following steps:

- Choose a random integer a from $[1, n-1]$ (called nonce)
- Calculate $P = [a]G = (x, y)$, then $r = x \mod n$. If $r = 0$, goes back to step 1
- If not, calculate $s = (m + rk)a^{-1} \mod n$,with $m = H(M)$ as the hash value of the message. If $s = 0$, goes back to step 1
- If not, the pair (r, s) is the signature.

The process requires choosing an appropriate random integer a to mask the private key through the modular multiplicative inverse of a and the sum of m with r times k. To verify the signature, we re-calculate point P' and compare the x-coordinate:

- Calculate $u = s^{-1}m \mod n, v = s^{-1}r \mod n$
- Calculate $P' = [u]G + [v]K = (x', y')$
- Compare r and $(x' \mod n)$; if they are congruent, the signature is valid

The algorithm's proof of correctness

- $P' = [u]G + [v]K = [u]G + [vk]G = [u + vk]G = [s^{-1}m \mod n + s^{-1}rk \mod n]G = [s^{-1}(m + rk) \mod n]G$ (1)
- Since $s = a^{-1}(m + rk) \mod n \Rightarrow a = s^{-1}(m + rk) \mod n$ (2)
- Substitute (2) into (1), $P' = [s^{-1}(m + rk) \mod n]G = [a]G = P$

The verification is simple to understand, but note that we have to calculate modular multiplicative inverse in both processes, signing and verifying. The modular inverse is an expensive ECC computation. In modular arithmetic, to find a multiplicative inverse of an integer a with a prime p, we first compute if $gcd(a, p) = 1$ to know If a has a modular inverse. ECC requires one more condition compared to modular arithmetic: the point needs to be on the curve, which means the coordinates have to satisfy the curve equation and follow the curve's addition law. That's why signing a message takes a lot of trial and error to find a suitable nonce. Moreover, to ensure the private key won't be leaked, the nonce has to be randomly generated for every signature. Or else, the private key can be retrieved as follow

- Assume that the signature s_1 and s_2 use the same nonce, which means $r_1 = r_2 = r$. The hash values of the two messages are $H(M_1) = m_1 and H(M_2) = m_2$.
- $s_1 - s_2 = a^{-1}(m_1 + rk) - a^{-1}(m2 + rk) \mod n$
- $s_1 - s_2 = a^{-1}(m_1 + rk - m_2 - rk) \mod n$
- $s_1 - s_2 = a^{-1}(m_1 - m_2) \mod n$
- $a = (m_1 - m_2)(s_1 - s_2)^{-1} \mod n$, the attacker now retrieves the nonce
- Since $s = a^{-1}(m + rk) \mod n \Rightarrow k = (sa - m)r^{-1} \mod n$, simply substitute nonce a into the equation, the attacker can access your private key.

This error has been exploited in the real-world, et al Console Hacking 2010 - PS3 Epic Fail Archived December 15, 2014, at the Wayback Machine, page 123-128. Because Sony uses a static nonce for every signature, the private key was extracted, thereby gaining the public key required to run any software on the machine.

2.3.2.2 Schnorr's signature scheme and EdDSA

EdDSA is a digital signature scheme using a variant of Schnorr's signature based on twisted Edwards curves. Originally, the standard Schnorr's signature algorithm for an elliptic curve includes these steps

- Choose a private key k in range $[1, n-1]$ and calculate $P = [k]G$
- Hash the message M to obtain $m = H(M)$
- Generate a random number z in range $[1, n-1]$, then calculate $R = [z]G$
- Calculate $s = z + H(r \| P \| m) \times k$, where r is the x-coordinate of the point R, $\|$ denote string concatenation
- The signature is the pair (r, s)

Once you obtain the signature (r, s) and the public key (the point P), you can verify it by

- Calculate $R(r, y)$ with r
- Verify if $[s]G = R + [H(r \| P \| m)]P$. If true, the signature is valid

Proof of correctness

- LHS: $[s]G = [z + H(r \| P \| m) \times k]G$

- RHS: $R+[H(r\|P\|m)]P = [z]G+[H(r\|P\|m)]\times[k]G = [z]G+[H(r\|P\|m)\times k]G = [z+H(r\|P\|m)\times k]G = LHS$

Schnorr's signature algorithm, therefore, guarantees that if either r, the original message m, or the public key P is invalid, the signature is also invalid. Note that a point is compressed when hashing, with a corresponding compression scheme for the curve. Then, in 2012, Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang adjusted Schnorr's signature for twisted Edwards curve and developed an EdDSA scheme, called Ed25519. Since Ed25519 use a point little-endian encoding scheme, the key generation is a bit different

- Private key: a b-bit randomly generated integer k
- Public key: also a b-bit integer. To generate a public key from the private key
  - Hash the private key: $H(k) = (h_0, h_1, \ldots, h_{2b-1})$
  - Use the lower 32 bytes $(h_0, h_1, ..., h_{b-1})$ for buffer. Pruning the buffer by setting $h_0 = h_1 = h_2 = h_{b-1} = 0$, and $h_{b-2} = 1$
  - Interpret the buffer as a little-endian integer, denoted s
  - Computer the point $A' = [s]B$
  - Encode A' to obtain A. A is the public key

To sign a message M, you need the private key k and the public key A. The steps are

- Hash the private key: $H(k) = (h_0, h_1, \ldots, h_{2b-1})$
- Use the higher 32 bytes $(h_b, h_{b+1}, ..., h_{2b-1})$ for buffer
- Compute $r = H(\text{ buffer } \|M) \mod L$
- Compute $R' = [r]B$ and encode R' to gain R
- Compute s (the same as public key generation). Then compute and encode to obtain the public key A
- Compute $e = H(R\|A\|M) \mod L$
- Compute $S' = (r + e \times s) \mod L$, encode S' to obtain S
- The signatures is $(R\|S)$

Note that S is the little-endian encoding of S', which has the three MSB of the final octets are always zero. This causes the last 3 bits of the signature $(R\|S)$ are also zeros. For verification, any decoding failure means that the signatures are invalid. To verify the signature $(R\|S)$

- Decode the first half (the first 256 bits) of the signature for R' and the other half for S'
- Decode A to obtain A'
- Compute $e' = H(R\|A\|M)$ as a 64-byte little-endian integer
- If $[S']B = R' + [e']A'$, then the signature is valid

Take a note that e' (512-bit) is congruent to e(253-bit) mod L, which means the scalar multiplication will result in the same point. Proof-of-correctness

- LHS $= [S]B = [(r + e \times s) \mod L]B = [r \mod L]B + [e \times s \mod L]B = [r]B + [e \times s]B = R' + [e]A'$
- As noted, $[e']P = [e]P \Rightarrow [e']A' = [e]A'$ (1)
- Substitute (1) into LHS $= R' + [e]A' = R' + [e']A' =$ RHS

In EdDSA, we can see that there is no random nonce used. Instead, EdDSA uses a deterministic nonce from the message digest and a part of the private key. Through all the process, the private key is not used directly to sign, ensure that there are no information about the private key can be leaked.

2.3.3 Comparison between ECDSA with Secp256k1 and Ed25519

2.3.3.1 Secp256k1 vs. Ed25519

For ECC, we consider a curve is secure based on these criteria (according to SafeCurve): the curve's resistance to the attacks on the elliptic curve discrete logarithm (ECDLP security) and its resistance to attack in the real world (ECC security). SafeCurve did list some of the criteria for ECDLP, but M. C. Semmouni, A. Nitaj and M. Belkasmi in Bitcoin security with a twisted Edwards curve had conducted more thorough research on the cryptanalytical study and the efficiency analysis to point out that Edwards25519 is more secure than the curve Secp256k1. First of all, they reviewed the ECDLP for each curve in terms of complex-multiplication field discriminants, Pohlig - Hellman attack, Pollard's rho attack (with j-invariant for speedup), anomalous attack, Frey-Ruck attack, and MOV supersingular attack. Their work proved that Edwards25519 is more resistant than the curve Secp256k1 to two attacks, which are complex-multiplication field discriminants and Pollard's rho. Secondly, to analyze the efficiency of Edwards25519 and the curve secp256k1, they compared the curves' arithmetical operations over projective coordinates. To evaluate the computational cost, let denote multiplication as M, squaring as S, addition as Add, and doubling as D. All the operations are performed in modular arithmetic, assuming that 1S = 1M (multiply by itself), 1D = 2Add. They pointed out that the computational cost on the curve secp256k1 takes 12M + 4S + 1D + 6Add (= 16M + 8Add) for point addition and 3M + 6S + 8D + 5Add (= 9M + 21Add) for point doubling. Whereas, the computational cost on Edwards25519 takes up to 7M + 2D + 7Add (= 7M

+ 11Add) for point addition and up to 3M + 4S + 1D + 6Add (= 7M + 8Add) for point doubling. It's clear that the operations on Edwards25519 are faster than those on the curve Secp256k1. They concluded that "it is more convenient to use the curve Edwards25519 for industrial applications such as in a Bitcoin system".

On the other hand, the requirements for ECC security are ladders , completeness , and indistinguishability . Ladders require a curve's single-scalar multiplication to be time-constant. Single-scalar multiplication is the most necessary computation in ECC (computing the curve point $[n]P$ given an integer n and a curve point P, which is the bottleneck in key generation, signing, and key exchanges. If the multiplication doesn't take constant time, the attackers can perform a timing attack to find out the secret keys. In 2014, Naomi Benger and Joop van de Pol and Nigel P. Smart and Yuval Yarom published a paper explaining how to recover secret keys from OpenSSL's implementation of ECDSA-secp256k1 using timing information from "as little as 200 signatures". They were able to make use of the side-channel information from almost all of the observed executions and succeeded in recovering the private key. To avoid the leakage of side-channel timing information, a safe curve has to support ladders. As mentioned above, any curve that is birationally equivalent to Montgomery form is ladder supported. The Koblitz curve Secp256k1 uses a different finite field, which lacks the point (0, 0) and its order is not divisible by 4, causing it can be isomorphic to neither the standard Weierstrass form nor Montgomery form. On the contrary, Edwards25519 is birationally equivalent to the Montgomery form of Curve25519, which allows it to have the ladder by default. Secondly, completeness means that a curve's addition law must have no exceptions, which means that for every rational point P on E and every rational point Q on E, the addition law for inputs P and Q produces exactly outputs $P + Q$. Secp256k1 has a special case for adding the same point, adding with negation and adding with the infinity point (i.e., they have cases for adding the point at infinity, adding the point twice,...), while Edwards25519's addition law is proven to be complete. Finally, indistinguishability is short for "indistinguishability from uniform random strings". Standard representations of elliptic-curve points are easily distinguishable from uniform random strings. This poses a problem for many cryptographic protocols using elliptic curves: censorship-circumvention protocols, for example, and password-authenticated key-exchange protocols. 2013 Bernstein–Hamburg–Krasnova–Lange introduced the following solution to the underlying problem. Construct an efficient constant-time bijective map between a large set of b-bit strings (large enough to be indistinguishable from all b-bit strings; i.e., very close to 2^b possibilities) and a large set of rational points on an elliptic curve (e.g., about half of all points). Use uniform random points in this set, and represent them by the corresponding strings under this bijective map. These strings are indistinguishable from uniform random b-bit strings. There are various requirements for a curve to be able to construct a map, such as a subgroup order has to be congruent to 3 mod 4 or multiple of 4, but Secp256k1 fails these requirements. Overall, Edwards25519 does pass the core requirements of a curve to be "safe" enough to be applied in the real-world ECC in comparison to Secp256k1.

2.3.3.2 ECDSA vs. Ed25519

To evaluate these two digital signature schemes, we will analyze their algorithm and benchmark the time required for key generation, signing, and verifying. ECDSA is the

standard DSA for elliptic curve cryptography based on the ElGamal. The signature digital scheme Ed25519 is a variant of Schnorr's signature algorithm based on twisted Edwards curves. The hash functions used in each signature scheme are different: SHA-256 for ECDSA and SHA-512 for Ed25519. They differ only in the input bit lengths and produce outputs of lengths of 256 bits and 512 bits respectively. Nevertheless, SHA-256 and SHA-512 don't have the same security level. SHA-512 is more secure than SHA-256 and is recommended by various cryptographic standards such as NIST [NIST: Policy on Hash Functions, ENISA [ENISA: Algorithms, key size and parameters report (2014)] and BlueKrypt [Bluekrypt. cryptographic key length recommendation] for use for more sensitive data and for longest terms. This is an advantage for the digital signature Ed25519 over the digital signature ECDSA for long terms. Also, in the case of Ed25519, it chooses the nonce deterministically as the hash of a part of the private key and the message rather than a randomly generated nonce for each signature. Therefore; as long as the generated private key is secure, Ed25519 has no further need for a random number generator in order to make signatures, and there is no danger that a broken random number generator used to make a signature will reveal the private key.

For benchmarks, we found some awesome repo on github: justmoon, the Rust implementation of Ed25519 dalek, and the Rust implementation of Secp256k1 (tarcieri). In justmoon repo, he ran the original implementation of both algorithms, which produced a result of Ed25519 having a slightly faster average time by 76626.2 mirco-second and average performance by almost 1.1 times. emCrypt also performs a real benchmark on the hardwares.

2.3.4 Cryptographic hash function

Cryptography hash function is a one-way hash function that operates on a preimage of arbitrary size (often called "message") then returns a fixed-length hash value (or "message digest"). A one-way hash function means that it is easy to compute the hash value from an input but infeasible to find the information given the hash value. The formula is

$$h = H(M)$$

Where h is hash value, M is preimage and H is cryptographic hash function.

According to NIST, a cryptographic hash mush satisfied following three properties:

- Collision resistance: It is computationally infeasible to find two inputs that have the same hash value. The most efficient method to find must be brute-forcing of random inputs. The birthday "paradox" places an upper bound on collision resistance. That is, the complexity (or collision-resistance strength) of seeking two different inputs M and M' matched two hash $H(x) = H(x')$ equal to $L/2$ bit-operations for a L-bit hash function. For example, SHA-512 produces a collision resistance of 256 bits.

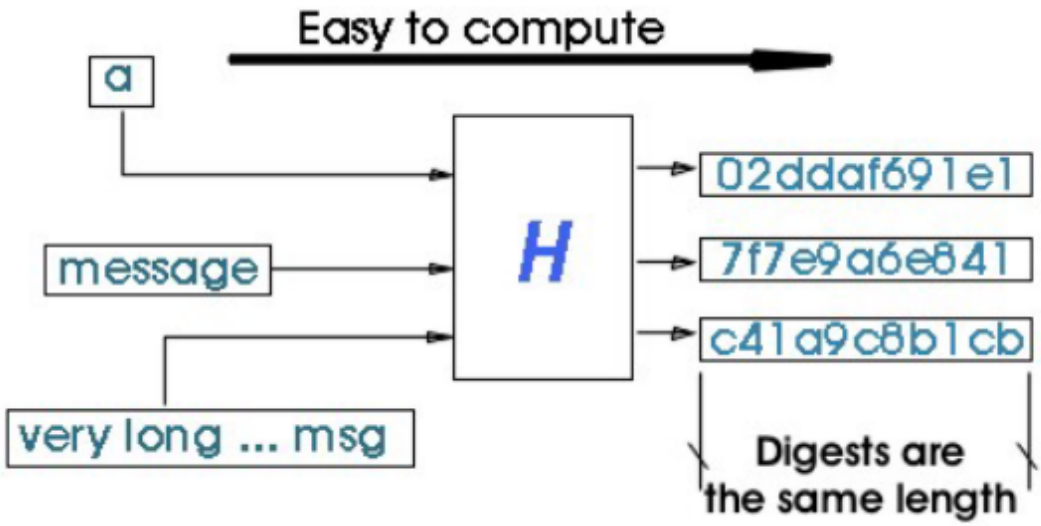

Figure 2.2: Hash function

- Preimage resistance: Given a hash value h, it is computationally infeasible to find an M that $H(M) = h$. Preimage resistance strength is equal to the amount of work to brute-force L-bit of operations for an L-bit hash function. For example, SHA-512 provides a preimage resistance of 512 bits.
- Second preimage resistance: Finding a second input that has the same hash value as any other specified input would be computationally infeasible. Second preimage resistance is calculated by the number of operations equal to L bits for an L-bit hash function. For example, SHA-512 produces a (full-length) hash value of 512 bits. However, in some cases of the hash functions, the second preimage resistance also depends on the length of the message resulting from the hash function.

2.3.4.1 Secure Hash Algorithm with digest 512 bits (SHA-512)

The SHA-512 may be used to hash a message M having a length of l bits, where $l \leq 0 \leq 2^{128}$. It produces a result of 512-bit message digest. The SHA-512 is part of a group of hashing algorithms that are very similar in how they work, called SHA-2. Algorithms such as SHA-256 are a part of this group alongside SHA-512. SHA-256 is used in blockchain as the default hash function. In our thesis, SHA-512 and SHA-256 will be used in the key derivation algorithm.

2.3.4.2 Hash-based message authentication codes (HMACs)

HMACs are a tool for calculating message authentication codes using a cryptographic hash function coupled with a secret key. HMAC can provide message authentication using a shared secret instead of using digital signatures with asymmetric cryptography. NIST mentioned in that HMAC uses a key, K, of appropriate security strength, compute with a text using the operation as follows:

$$HMAC(K, text) = H((K_0 \oplus opad) \| H((K_0 \oplus ipad) \| text))$$

Where:

- H is a cryptographic hash function (our thesis use SHA-512 and SHA-256)
- $text$ is the data
- K is the secret key
- K_0 is a key derived from the secret key by padding to the right with 0s up to the block size, or by hashing down to less than or equal to the block size first and then padding to the right with zeros
- $\oplus$ denotes bitwise exclusive or (XOR)
- $opad$ is a block size repeated bytes $0x5c$
- $ipad$ is a block size repeated bytes $0x36$

The final result of HMAC is equivalent to that of the underlying hash function (for example, 256 and 512 bits respectively in the case of SHA-256 and SHA-512), although it can be truncated if desired.

2.3.4.3 Password Based Key Derivation Function (PBKDF2)

PBKDF2 is a simple cryptographic key derivation function approved by NIST, which is resistant to dictionary attacks and rainbow table attacks (use precomputed hashes). It applies a pseudorandom function (our thesis uses HMAC) to the input password or passphrase along with a salt value and repeats the process many times to produce a derived key, which can then be used as a cryptographic key in subsequent operations. PBKDF2 takes several input parameters and produces the derived key as output:

$$key = pbkdf2(password, salt, iteration, h, k)$$

Technically, the input data for PBKDF2 consists of:

- A *password* - array of bytes / string, e.g. “stronger mnemonic”
- A *salt* - array of bytes / string generated by approved Random Bit Generator, e.g. ”stronger password” (minimum 64 bits, NIST recommended 128 bits is optimal)
- The *iteration*, e.g. 100000 iterations
- h: a hash function for calculating HMAC, e.g. SHA256
- k: key-length size, e.g. 32 bytes (256 bits)

3

Protocols and Related Works

This chapter describes the attempts on Cryptography protocols behind the existing HD wallets as well as the attempt of adapting these mechanisms to curve ed25519. We analyze the famous wallet that the community are using.

Contents

3.1 Protocol Proposals **36**
3.1.1 Key Derivation 36
3.1.1.1 BIP32 36
3.1.1.2 SLIP10 49
3.1.1.3 SLIP23 54
3.1.2 BIP39 - Random Password Generator 60
3.1.3 BIP44 - Index Path for Multi-coin Wallet 61
3.2 Related Works **63**

3.1 Protocol Proposals

3.1.1 Key Derivation

The word “Hierarchical Deterministic” in crypto wallet were first introduced by Pieter Wuille in Bitcoin Improvement Proposal number 32 (BIP32)[5]. The initial idea was to build a wallet architect like a tree. From the root seed, we can derive child private keys and child public keys that construct many child wallets for different purposes. However, BIP32 only supports the elliptic curve used in Bitcoin, defined by the name "secp256k1" [6]. SatoshiLabs Improvement Proposal number 10 (SLIP10) is an attempt to generalize the BIP32's key derivation schema for different curves, e.g., NIST P-256 and ed25519. For some security reasons, SLIP10 forbade normal child key derivation on curve ed25519. But SLIP23 adopted them both and proposed a new schema on curve Edwards25519 called "BIP32-ed25519" [7]. In this section, we will explain how these schemes work and analyze the difficulties in adapting BIP32 to the Edwards25519 Curve.

3.1.1.1 BIP32

The most significant motivation of BIP32 is how one can calculate the public key of another without revealing the private keys or the sender has to generate it from private keys. Elliptic curve mathematics permits these kinds of schemes. For example, a webshop business lets its web server generate fresh addresses (public key hashes) for each order or for each customer, without giving the webserver access to the corresponding private keys (which are required for spending the received funds).

BIP32 also makes wallets more recoverable since users only have to maintain the root seed and a list of wallet indexes. In the example of a webshop, the web owner doesn't have to keep all the private keys safe, and this becomes more redundant if the owner has an increase in a business requiring more wallets. Instead, he only has to create the seed one and save the indexes each time he makes a wallet. The next part will cover the indexes of children' wallets and how we can recover the entire wallet structure using the initial seed and list of indexes.

Derivation

BIP 32 key derivation is a mechanism used to generate a specific ECDSA key pair within a tree of indexes. Given a parent extended key and an array of indexes, we will be able to deterministically regenerate keys at that specified index in the tree. Starting from a master seed to the creation of one master extended key, we can create a tree with infinite depth. Each node can also have a maximum of 4,294,967,296 child nodes (2^{32}/32-bit unsigned integer) resulting in an incredible ability to generate a limitless number of keys. Figure 3.1 is an example of a BIP32 key tree with depth 3.

Extended keys

An extended key is defined as follow:

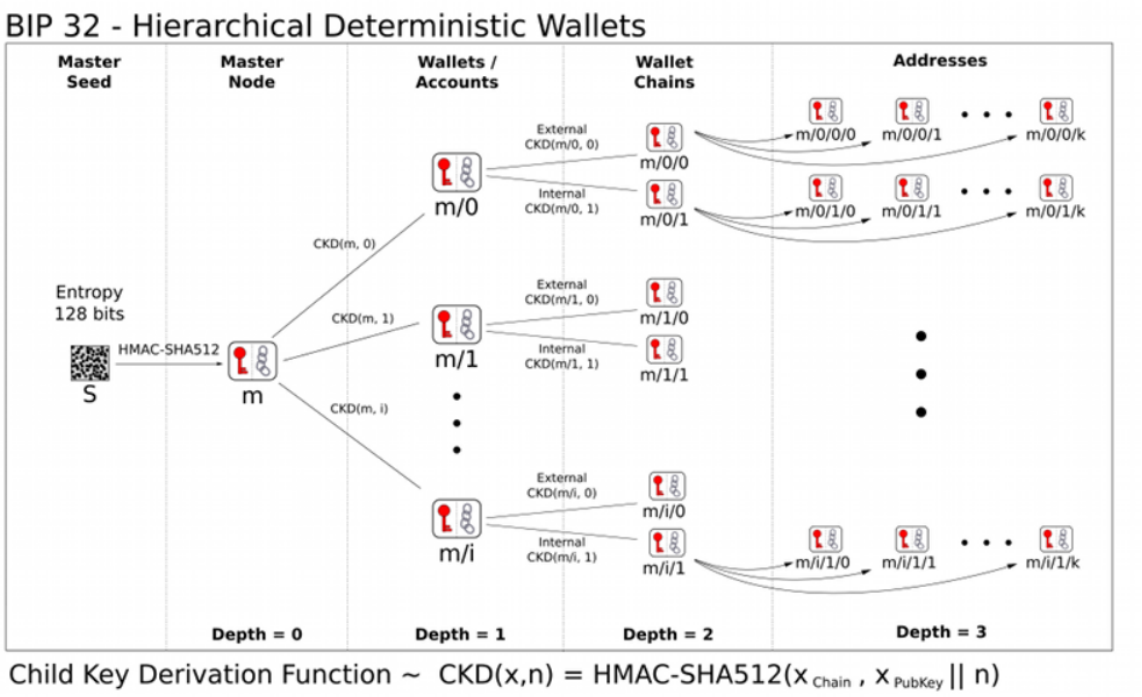

Figure 3.1: Example of BIP32 key tree with depth 3

> An extended key consists of a private or public key and chain code. An extended key can create children, generating its own branch in the tree structure. Sharing an extended key gives access to the entire branch.

The term "extended key" could also be thought of as "extensible key" because such a key can be used to derive children. BIP32 extends both private and public keys first with an extra chain code. This extension is added in order to prevent depending solely on the key itself; it also is identical for corresponding private and public keys.

In total, an extended key is represented simply as the concatenation of the 32 bytes key and 32 bytes chain code into a 64 bytes sequence (512-bits). So the total number of possible extended key pairs is almost 2512, but the produced keys are only 256 bits long and offer about half of that in terms of security. But the designed key size already satisfies the requirement of the NIST recommendation. In specifically, to maintain security against classical attacks, NIST has recommended transitions from key sizes and algorithms that provide 80 bits of security to key sizes and algorithms that provide 112 or 128 bits of security in 2016 [8].

Driven by advances in both classical and quantum computing technologies, in May 2020, NIST recommended improving key length to 256 bits [9]. It is unclear when scalable quantum computers will be available, and we will keep updating our system with the latest news on NIST publication.

Child key on the BIP32 key tree can only be derived from an extended key. From a security perspective, this is an excellent idea on how we can protect the child keys if the attackers get a hold of a stolen parent key. Because the extended key is collectively made up of a combination of key and chain code, attackers won't have a better way to derive

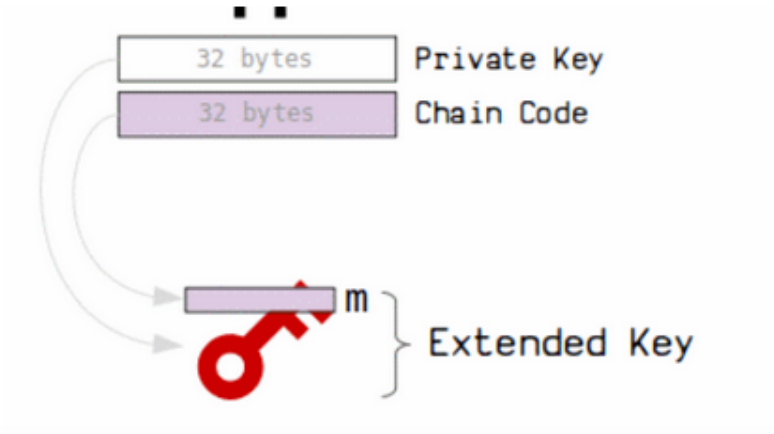

Figure 3.2: Example of the extended private key

the children keys other than brute-forcing 32 bytes of extended entropy.

There are two types of extended keys. An extended private key is the combination of a private key and chain code and can be used to derive child private keys (and from them, child public keys) (see Figure 3.2). An extended public key is a public key and chain code, which can be used to create child public keys (public only).

The index of the child key is a 32-bit integer, meaning that each extended key has 2^{32} child keys. This index number is divided into two ranges to easily determine between keys derived through the normal derivation function versus keys derived from hardened derivation. Index numbers between 0 and 2^{31}–1 (0x0 to 0x7FFFFFFF) are used for normal derivation and the index numbers between 2^{31} and 2^{32}–1 (0x80000000 to 0xFFFFFFFF) are used only for hardened derivation. Therefore, the normal child will have an index of less than 2^{31}, whereas the hardened child index equals or is above 2^{31}.

We couldn't find any reason behind why the author of BIP32 designed the parent extended keys to have 2^{32} total child keys each. Poulami Das et al.[10] investigated BIP32 with security analysis and experimented with a similar key derivation schema (BIP32-m) with only 2^{20} child keys, pointing out 2^{32} child keys won't create any vulnerability on the system.

In the next part, we will represent an extended private key as (k, c), with k the normal private key, and c the chain code. An extended public key is represented as (K, c), with K = point(k) and c the chain code. The index is denoted to i; to ease, i_H is the index of a hardened key representing a number of i+2^{31}.

Convention

In the rest of this thesis, we will assume the public key cryptography used in Blockchain, namely elliptic curve cryptography using the field and curve parameters. Variables below are either:

- Integers modulo the order of the curve (referred to as n).

- Coordinates of points on the curve.
- Byte sequences.

The addition (+) of two coordinate pairs is defined as the application of the EC group operation. Concatenation (||) is the operation of appending one-byte sequence onto another.

As standard conversion functions, we assume:

- $point(p)$: returns the coordinate pair resulting from EC point multiplication (repeated application of the EC group operation) of base point with the integer p.
- $ser_{32}(i)$: serialize a 32-bit unsigned integer i as a 4-byte sequence, most significant byte first.
- $ser_{256}(p)$: serializes the integer p as a 32-byte sequence, most significant byte first.
- $ser_P(P)$: serializes the coordinate pair $P = (x, y)$ as a byte sequence using the compressed form: $(0x02$ or $0x03) \parallel ser_{256}(x)$, where the header byte depends on the parity of the omitted y coordinate. 0x02 if y is positive and 0x03 if y is negative.
- $parse_{256}(p)$: interprets a 32-byte sequence as a 256-bit number, most significant byte first.

Master Extended Key generation function

Master extended key is the first key to be generated from the root seed. A master key also possesses a size of 2^{512} where 256 first bits is the master private key and the last 256 bits is the chain code. From the master private key, we can calculate the master public key.

Pseudo-code of generation function is as below:

- Generate a seed byte sequence S of a chosen length (between 128 and 512 bits; 256 bits is advised) from a (pseudo) random number generator (PRNG).

  > The PRNG, as mentioned in Section 2, is recommended to produce a root seed. However, BIP39 [11] invented a method to create a seed from mnemonic code or mnemonic sentence entropy. We will discuss BIP39 in Section 3.1.1.2 . Ideally, the creation of BIP39 helps users to maintain their wallets more efficiently since secure random sentences (12 to 24 words) are more comfortable than protecting a random byte sequence.

- Calculate I = HMAC-SHA512(Key = "Bitcoin seed", Data = S)

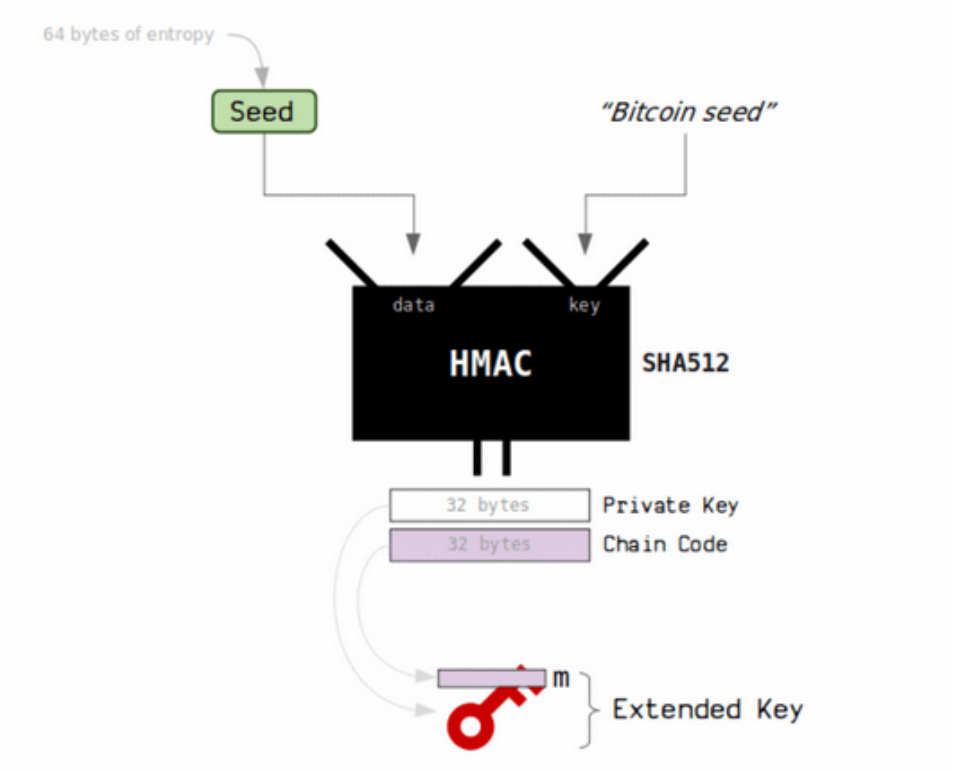

Figure 3.3: Master Extended Key generation Process

The HMAC function returns 512 bits of data (which is totally unpredictable) from a key and data component. BIP32 added the key "Bitcoin seed" to the function, although it could be arbitrary when deriving the master node from the seed. This is considered necessary to ensure proper domain separation between different elliptic curves or different types of key hierarchy generation schemas.

- Split I into two 32-byte sequences, I_L and I_R.

  The result of the HMAC function with the size 512 bits (64 bytes) will be split into 32 bits left I_L and 32 bits right I_R.

- Use $parse_{256}(I_L)$ as the master secret key, and I_R as master chain code.

  Return 32 bits left I_L as the secret key and 32 bits right I_R as the extended chain code.

In case I_L is 0 or n (private key order n), the master key is invalid.

Discussing the function HMAC-SHA512

The HMAC-SHA512 is specified in [12]. It is used in all key derivation functions of the BIP32 schema. This raises the question: isn't it just

SHA512 with the input as Key ‖ data is enough to generate an unexpected output of the root seed? We believe the answer is to avoid the flaws that can be caused by simple concatenation, for example length extention attacks. In cryptography and computer security, a length extension attack is a type of attack where an attacker can use Hash(message1) and the length of message1 to calculate Hash(message1 ‖ message2) for an attacker-controlled message2, without needing to know the content of message1. Also it provide a better integrity of information for the secret keys.

Child key derivation (CKD) functions

There are 3 methods for deriving child keys:

1. Private parent key → normal private child key
2. Public parent key → normal public child key
3. Private parent key → hardened private child key

And Public parent key → private child key is not possible. Derived child extended keys (and parent keys) are independent of each other. In other words, you wouldn't know that two public keys in an extended tree are connected in any way.

Normal extended child private key derivation function The function CKDpriv((k_{par}, c_{par}), i) → (k_i, c_i) computes a normal child extended private key from the parent extended key. The Figure 3.4 describes the process.

Pseudo-code:

- Check whether $i \geq 2^{31}$ (whether the child is a hardened key).
  - If so (hardened child): return false
  - If not (normal child):

    Let I = HMAC-SHA512(Key = c_{par}, Data = $ser_P(point(k_{par}))$ ‖ ser_{32}(i)). Key input c_{par} is the chain code of the extended parent key. Data input $ser_P(point(k_{par}))$ ‖ ser_{32}(i) contains 2 parts that are concatenated together. The $point(k_{par})$ is the result of working out the coordinate pair of the public key from k_{par}, as shown in Figure 4, the public parent is a part of the Data. The function ser_P will return the compressed representation of $point(k_{par})$ with the size of 33 bytes [6]. And the value ser_{32}(i) is just the representation of the normal index—two of these elements concatenated together to form the data of HMAC-SHA512 function.

- Split I into two 32-byte sequences, I_L and I_R.

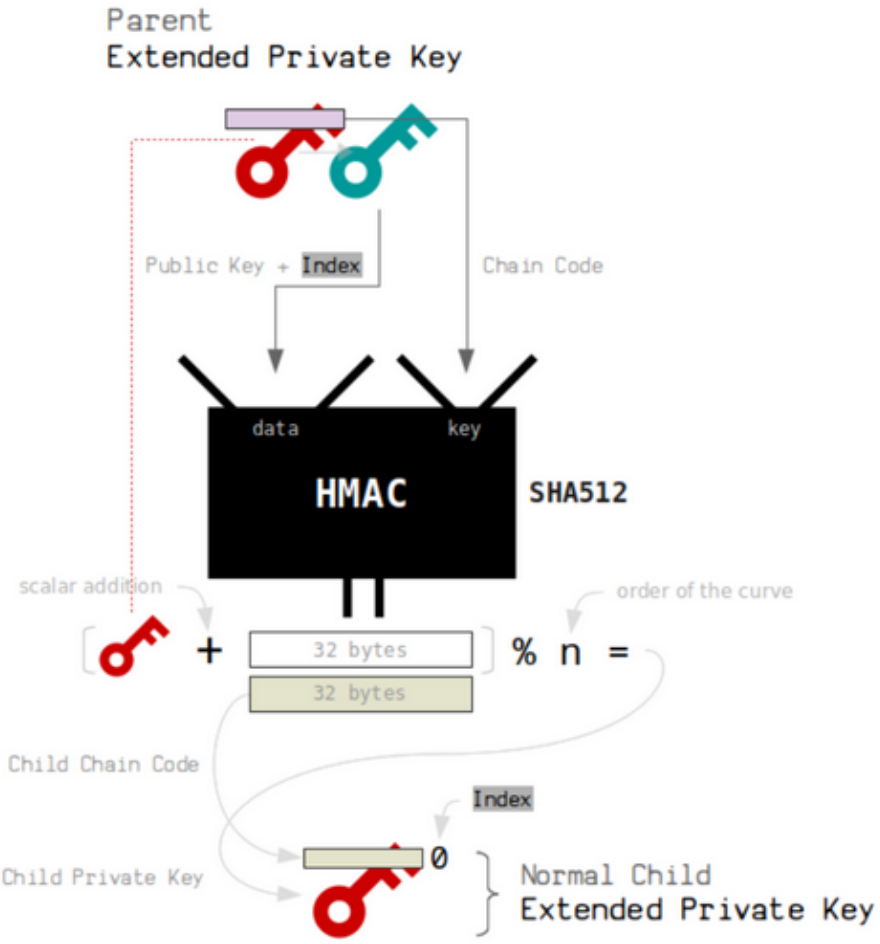

Figure 3.4: Normal Extended Private Key generation process

- The returned child key ki is $parse_{256}(I_L) + k_{par}$ (mod n).

  The child ki is an increment of the private parent key k_{par} with the amount of $parse_{256}(I_L)$. We take the module of the new private key by the order of the curve (n) to keep the new private key belonging to the valid range of numbers for the secp256k1 curve. As mentioned previously, a very useful characteristic of BIP32 wallets is the ability to derive public child keys from public parent keys, without having the private keys. This returned key plays an important part in that ability.

- The returned chain code ci is I_R.

- In case $parse_{256}(I_L) \geq$ n or ki = 0, the resulting key is invalid, and one should proceed with the next value for i. (Note: this has probability lower than 1 in 2^{127}.)

To summarize, we make use of the data contained inside the parent extended private key: the private key, the chain code, and the index of the desired child. Putting it through the HMAC function will generate some unique random bytes. We use these new random bytes to construct the following private key from the old one.

Normal extended child private key derivation function

The function CKDpub((K_{par}, c_{par}), i) $\rightarrow$ (K_i, c_i) computes a child extended public key from the parent extended key. It is only defined for non-hardened child keys. The Figure 3.5 describes the process.

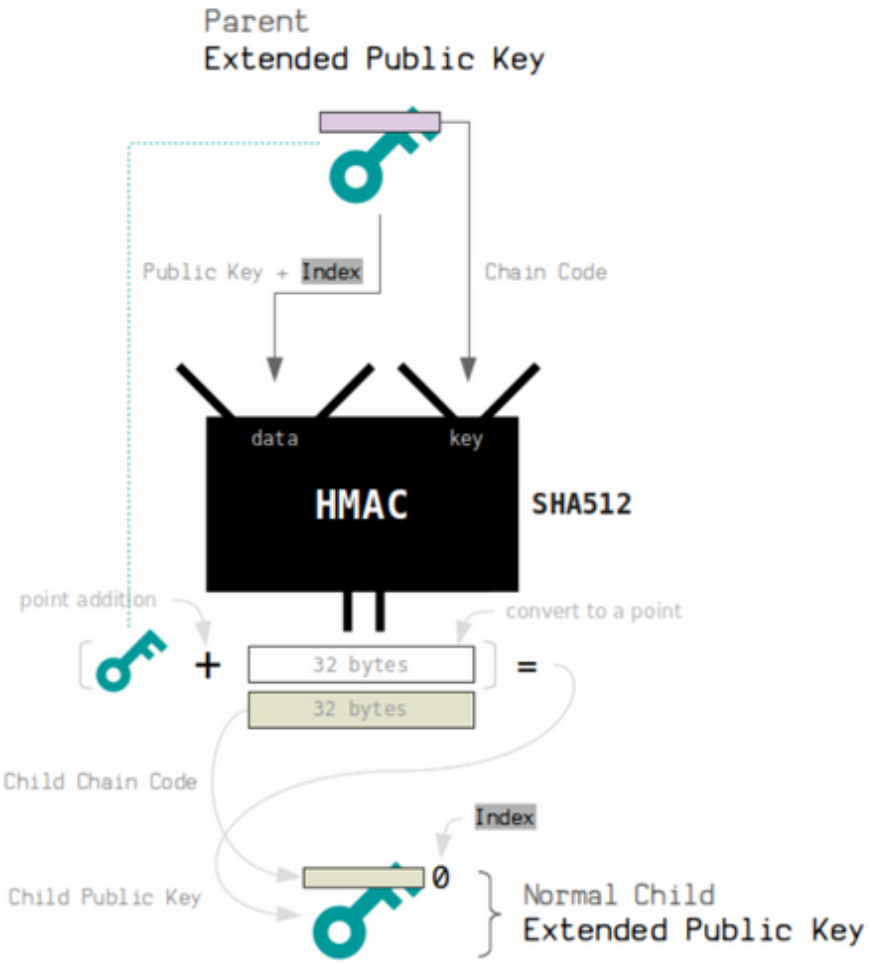

Figure 3.5: Normal Extended Public Key generation process

Pseudo-code:

- Check whether $i \geq 2^{31}$ (whether the child is a hardened key).
  - If so (hardened child): return false
  - If not (normal child):

let I = HMAC-SHA512(Key = c_{par}, Data = $ser_P(K_{par}) \parallel ser_{32}$(i)). The function $ser_P(K_{par})$ returns the compressed representation of the parent public key with the size of 33 bytes. And the value ser_{32}(i) is just the representation of the normal index.

- Split I into two 32-byte sequences, I_L and I_R.
- The returned child key K_i is $point(parse_{256}(I_L)) + K_{par}$.

  The split left half I_L will be converted into a point on curve secp256k1 by function $point()$. Then $point(parse_{256}(I_L)) + K_{par}$ is the point addition on curve secp256k1 and returns the child public key.

- In case $parse_{256}(I_L) \geq$ n or K_i is the point at infinity, the resulting key is invalid, and one should proceed with the next value for i.

In summary, public key derivation only allows normal child derivation. Putting the same key and Data through the HMAC function as we did in normal private, we can produce the child public key from the left 32 bytes via elliptic curve point addition. The schema normal key derivation make the motivation of BIP32 come true. We will explain the relation between normal keys derivation. It is very convenient to derive a branch of public keys from an extended parent key, but it comes with a potential key leakage. We will discuss the potential risk of normal child key derivation schema compared to its value in the very last of this Section and analyze if it is suitable for our thesis.

Hardened extended child private key derivation function

The function CKDpriv((k_{par}, c_{par}), i) $\rightarrow$ (k_i, c_i) computes a child extended private key from the parent extended key. The Figure 3.6 describes the process.

Pseudo-code:

- Check whether $i \geq 2^{31}$ (whether the child is a hardened key).
  - If so (hardened child):

    let I = HMAC-SHA512(Key = c_{par}, Data = 0x00 $\parallel$ $ser_{256}(k_{par})$ $\parallel$ ser_{32}(i)). (Note: The 0x00 pads the private key to make it 33 bytes long.)
    The difference in hardened key derivation is that we don't recover the public key from the parent extended key. Instead, we padded 0 to the left of k_{par} to make it 33 bytes long. Adding the leading zeros will not change the value. However, it is needed, so the number takes up the entire 33 bytes. (Since the compressed representation of a public key is 33 bytes).
  - If not (normal child): return false
- Split I into two 32-byte sequences, I_L and I_R.

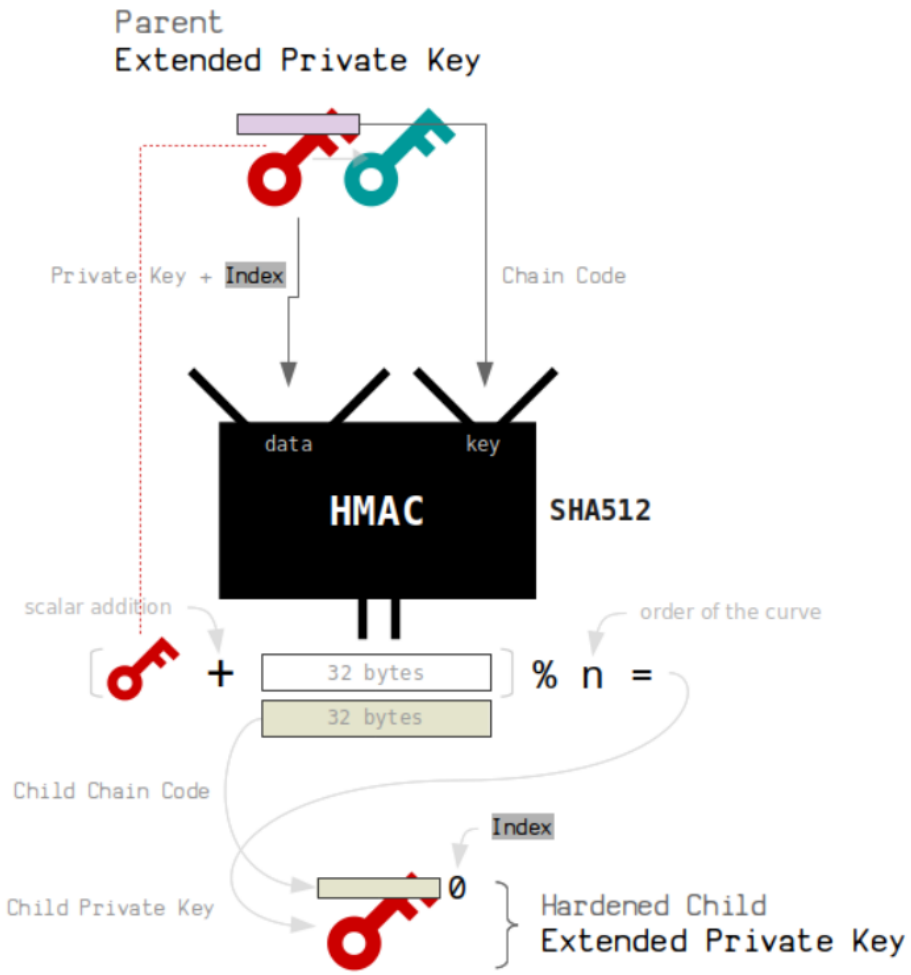

Figure 3.6: Hardened Extended Private Key generation process

- The returned child key ki is $parse_{256}(I_L) + k_{par}$ (mod n).
- The returned chain code ci is I_R.
- In case $parse_{256}(I_L) \geq$ n or ki = 0, the resulting key is invalid, and one should proceed with the next value for i. (Note: this has probability lower than 1 in 2^{127}.)

The hardened derivation is therefore used to create a "gap" in the tree above the level where extended public keys are used. Therefore, The hardened key derivation schema is needed to preserve security in HD Wallet.

The relation between normal keys derivation

From here to the rest of BIP32 section, we assume a scenario where:

> Alice is a shop owner, and she wants to create an HD wallet. Each wallet can receive and transfer coins for different purposes. She decided to use a wallet with the normal key derivation schema to transfer money between her wallet and others without any private key access whatsoever.
>
> Bob wants to transfer 1 Bitcoin to Alice. He generates an address from Alice's master public key and chain code using normal child key derivation schema with an index i. Bob sends 1 Bitcoin to that address with the message containing the index i.
>
> Alice now can generate the normal child private key from the master secret key with an index i and possess a child wallet with an index i.

What Bob just did is derive a public key from the master extended public key that corresponds to the Alice private key derived from Alice's master extended private key, without knowing her extended private key. This scenario comes to reality thanks to the point addition law of the elliptic curve.

Denote child private key is k_{child}, child public key is K_{child}. From normal extended child private key derivation function, we have:

- I = HMAC-SHA512(Key = c_{par}, Data = $ser_P(point(k_{par}))$ $\|$ ser_{32}(i))
- Child private key $k_{child} = parse_{256}(I[:32]) + k_{par}$ (mod n).

and normal extended child public key derivation function, we have:

- I = HMAC-SHA512(Key = c_{par}, Data = $ser_P(K_{par})$ $\|$ ser_{32}(i)).
- Child public key $K_{child} = point(parse_{256}(I[:32])) + K_{par}$.

For both child extended keys, we are inserting the *identical inputs* to the HMAC function. The Key is the parent chain code. The Data is the parent public key. Through the HMAC function, we generate the same data as a result. Using the left 32 bytes of this result, we calculate:

- Increase the parent private key k_{par} an amount of the same 32 bytes (which is a number) to get the k_{child}.
- Apply point addition on the parent public key K_{par} by the same 32 bytes to create the child public key K_{child}.

On account of the mechanism elliptic curve mathematics works, the derived child private key will correspond to the derived child public key:4

$$(k_{par} + I[:32]).G = k_{child}.G = K_{child}. \tag{3.1}$$

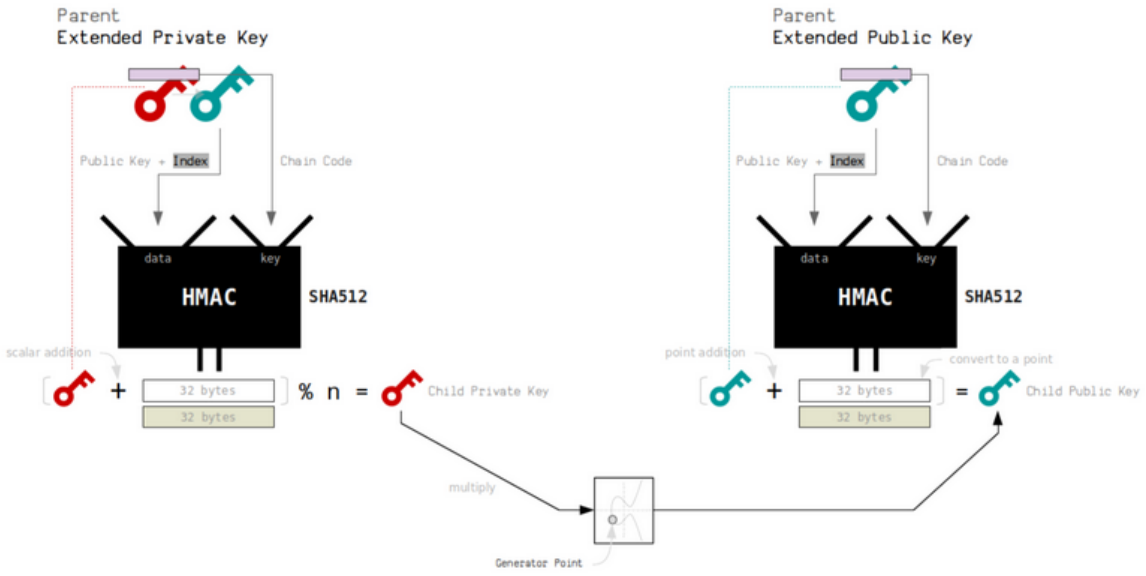

Figure 3.7: Relation in BIP32 normal derivation protocol

Figure 3.7 presents the relation of normal derived child keys with their parent keys.

Discuss the potential risk of normal child key derivation schema

First, the value

This shortcut can be used to create a very secure public key–only deployments where a server or application has a copy of an extended public key and no private keys whatsoever. That kind of deployment can produce an infinite number of public keys and Bitcoin addresses, but cannot spend any of the money sent to those addresses. Meanwhile, on another, more secure server, the extended private key can derive all the corresponding private keys to sign transactions and spend the money.

One common application of this solution is to install an extended public key on a web server that serves an eCommerce application. The web server can use the public key derivation function to create a new Bitcoin address for every transaction (e.g., for a customer shopping cart). The web server will not have any private keys that would be vulnerable to theft. Without HD wallets, the only way to do this is to generate thousands of Bitcoin addresses on a separate secure server and then preload them on the eCommerce server. That approach is cumbersome and requires constant maintenance to ensure that the eCommerce server doesn't "run out" of addresses.

Another common application of this solution is for cold storage or hardware wallets. In that scenario, the extended private key can be stored on a paper wallet or hardware device (such as a Trezor hardware wallet), while the extended public key can be kept online. The user can create "receive" addresses at will, while the private keys are safely stored offline. To spend the funds, the user can use the extended private key on an offline signing Bitcoin client or sign transactions on the hardware

wallet device (e.g., Trezor).

The main use case for which this feature is advertised is in hierarchical organizations: the treasurer of a company might have control over the root private key of a BIP32 wallet and then hand off a "child" seed to each of the company's departments who will then use that seed to operate their own wallet. The treasurer will have the master key to everything, but each department will only have the key to their own part of the funds.

Second, the vulnerability

Gus Gutoski et al.[13] investigated the BIP32 schema and considered the normal derivation vulnerable to master key recovery attacks. This attack vector is well known in the Bitcoin community and is considered to be "cannot be avoided" for BIP32 HD Wallet. The whole wallet hierarchy can be exposed to the hackers if one of the child private key is leaked.

Following Vitalik Buterin's article [14], the hackers can generate the parent key of the leaked wallet if they can get their hands on the private child key ki. The formula to calculate k_{par} is as follow:

$$k_i = k_{par} + hash(K_{par}, c_{par}, i) \tag{3.2}$$

$$k_{par} = k_i - hash(K_{par}, c_{par}, i) \tag{3.3}$$

If the attackers already have users private keys, they are more likely to possess the index i as well, even if they are not, they can still brute force the index (range 0 to 2^{31}) until satisfied the K(i) = k(i) * G. So that they can easily produce the parent private key. Not only that, together with a parent chain code, if they were continuing to perform the formula, they can reveal the master secret key. Worse, the master secret key with it's chain code will generate the entire private keys of all the wallets.

Avoiding the weakness

The ability to derive a branch of public keys from an extended public key is advantageous, but it comes with the above potential risk. The more keys derived for each party lead to the more fragile their wallet becomes. In practice, we see a lot of HD wallets still use normal key derivation like (Electrum lightweight Bitcoin Wallet). We evaluate these kinds of wallets in Section 3.2. Right now, we consider those applications that use normal key derivation that their security limitations are well worth the benefit.

To counter this risk, BIP32 later came up with hardened key derivation. The method is basically the same but does not evolve the parent public key to the input of HMAC function, which breaks the relation between parent public key K_{par} and the child chain code (last 32 bytes). The hardened derivation function uses the parent private key to derive the child chain code instead of the parent public key. This creates a

"firewall" in the parent/child sequence, with a chain code that cannot be used to compromise a parent or sibling private key. When the hardened private derivation function is used, the resulting child private key and chain code are entirely different from what would result from the normal derivation function. The resulting keys can be used to produce extended public keys that are not vulnerable because they cannot exploit the chain code to reveal any private keys. Therefore, the hardened derivation is used to create a "gap" in the tree above the level where extended public keys are used.

In simple terms, if we want to avoid the risks, we can use a hardened derivation schema rather than a normal derivation. But the value of BIP32 is too great if we can apply it to our Edwards25519 curve. In Section 3.1.1.3, we will introduce the BIP32-Edwards25519 schema where normal derivation key leakage can totally be prevented but it also has flaws. For the next section, we present a schema of Edwards25519 key derivation, which is a part of an attempt to apply BIP32 globally.

3.1.1.2 SLIP10

Due to the emergence of Blockchain technology and the more powerful full quantum attack, the industry applies some other more powerful curves and digital signature algorithms. The SatoshiLabs Improvement Proposals number 10 (SLIP10) was created for this reason. It defines a new hierarchical key derivation schema which is an extended superset of the BIP32 derivation algorithm to work on other curves. With secp256k1, SLIP10 is basically compatible. For the sake of our thesis, we will only examine their adaptation for the curve Edwards25519.

Master key generation function

Adapting the master key generation from BIP32, SLIP10 also has a master key that possesses a size of 512 bits. From the master private key, we can calculate the master public key. Figure 3.8 describes the process.

Pseudo-code:

- Calculate I = HMAC-SHA512(Key = "ed25519 seed", Data = S)

  Random seed S is a seed byte sequence of 128 to 512 bits in length (same as BIP32). The value of S should be the binary seed obtained from a PRNG function, or it should be the master secret obtained from a set of BIPs39 mnemonic codes and arbitrary passphrases. BIP39 is also a part of the HD wallet, and we will introduce it in Section 3.1.2. For the Key input, we use "ed25519 seed" to avoid generating the same private key for different elliptic curves with different orders.

- Split I into two 32-byte sequences, I_L and I_R.

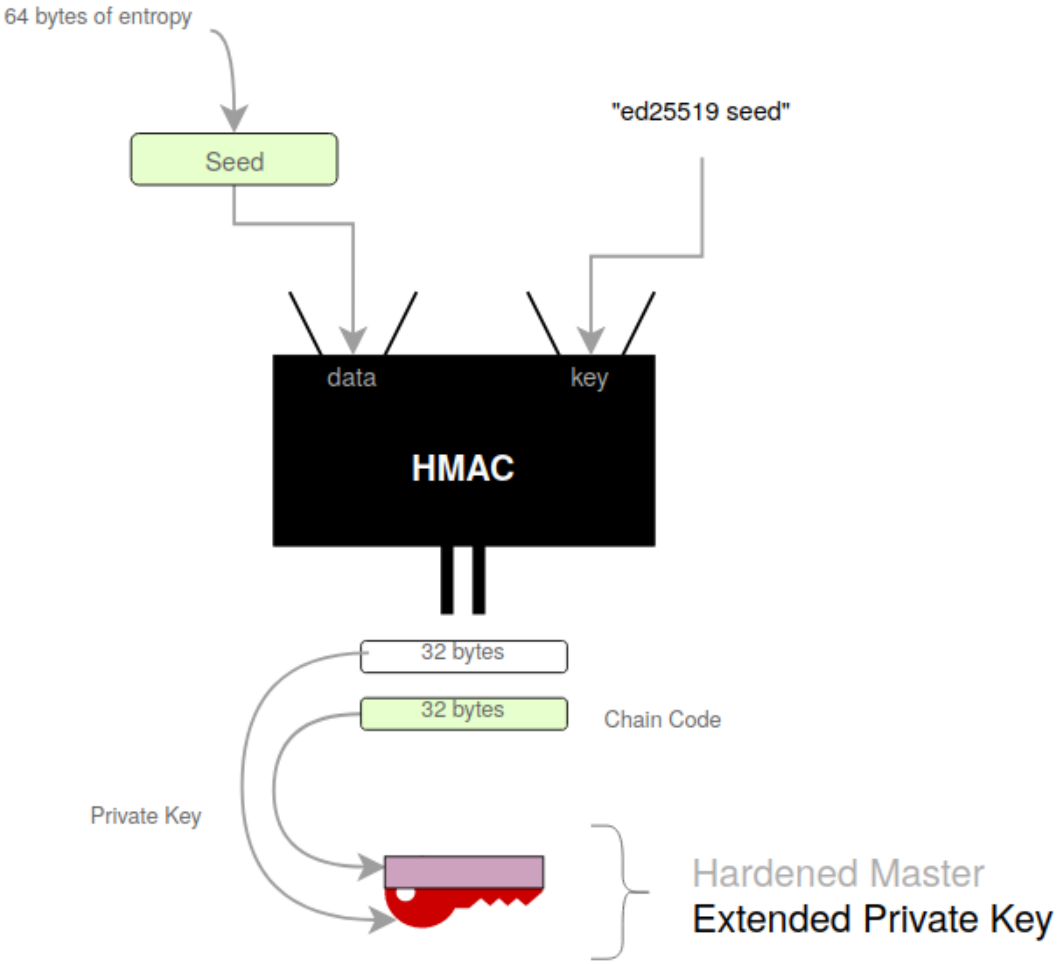

Figure 3.8: Master Extended Key generation Process

- Use $parse_{256}(I_L)$ as the master secret key, and I_R as master chain code.

  Because every 256-bit number (even 0) is a valid private key (see 2) on curve ed25519, we don't have to validate the $parse_{256}(I_L)$ and return it as a secret key.

Hardened child key derivation function

SLIP10 only supports hardened key generation for the Edwards25519 curve, meaning that we can not derive public child keys from any extended parent keys. The information on the reasoning for this will be shown in **Difficulties in adapting BIP32 to ed25519 curve**.

The function CKDpriv((k_{par}, c_{par}), i) $\rightarrow (k_i, c_i)$ computes a child extended private key from the parent extended private key. Figure 3.9 describes the process.

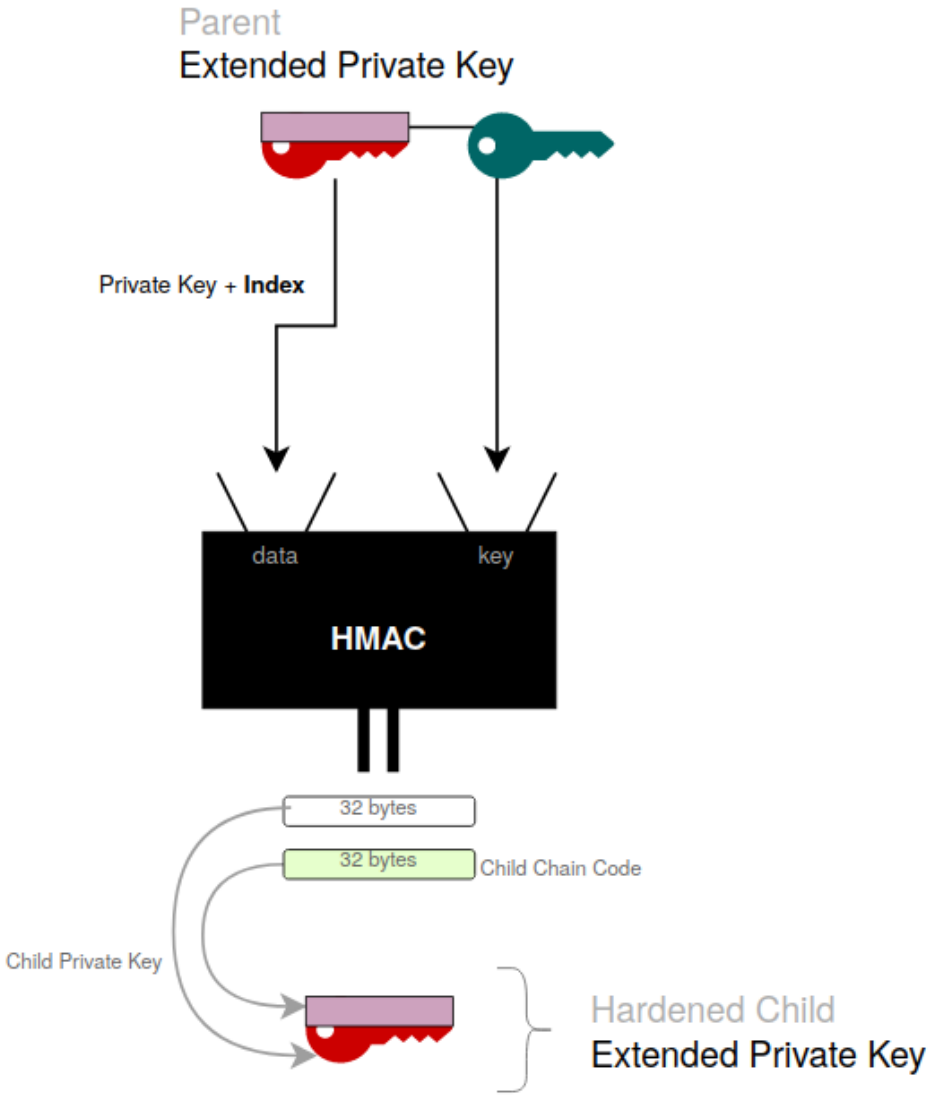

Figure 3.9: Hardened Extended Public Key generation process

Pseudo-code:

- Check whether $i \geq 2^{31}$ (whether the child is a hardened key).
  - If so (hardened child):

    Let I = HMAC-SHA512(Key = c_{par}, Data = 0x00 $\|$ $ser_{256}(k_{par})$ $\|$ ser_{32}(i)). (Note: The 0x00 pads the private key to make it 33 bytes long.)
    The return I in SLIP10 is compatible with BIP32.

– If not (normal child): return false

- Split I into two 32-byte sequences, I_L and I_R.

- The returned chain code ci is I_R and returned child key ki is $parse_{256}(I_L)$.

  Same as the Master key generation function, SLIP10 returns the hashes directly as extended private child keys.

The HMAC-SHA512 function is specified in the Background.

Difficulties in adapting BIP32 to ed25519 curve

BIP32 was designed for curve secp256k1 only. Applying different curve schema is very challenging due to the nature of each curve. Ed25519 specifically, researchers around the world meet two main problems as below.

Ed25519's private key is not the multipliers for the group generator

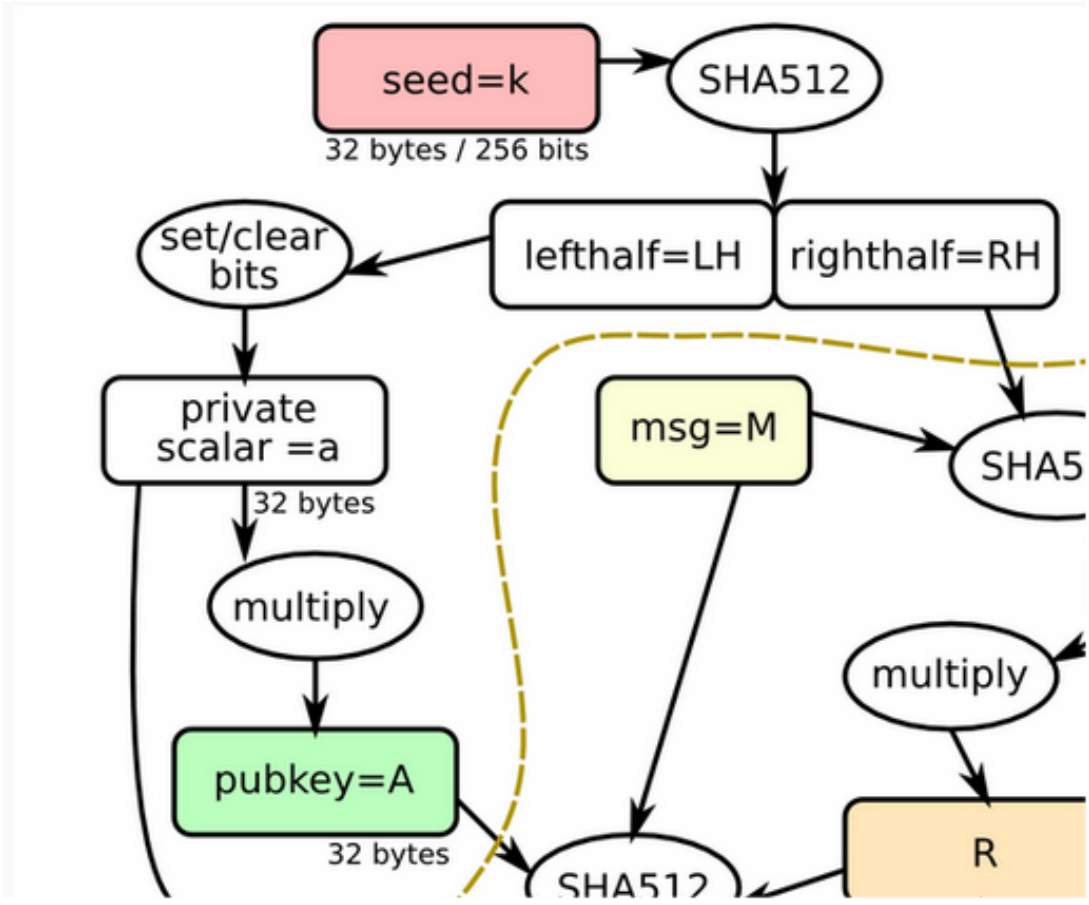

Figure 3.10: Multiply process of ed25519

Ed25519 key pair generation is different from secp256k1. Figure 3.10 shows how public key A was generated from private key k. As we present in Section 2.1, the public key A is the encoding of the point caused by a fixed-base scalar multiplication [s]B (B is the generator) where s is a "bit-clamping" (or pruning) value produced from the left half of the HMAC function result. The linearity in the BIP32 schema is lost entirely when applied to the Ed25519 keypair. The private keys are now no longer multipliers for the group generator.

So the normal private keys, when derived, won't correspond to the normal public key. The modified hash of the private key is now the multiplier. This cannot be changed since it is the core of speeding up in the ed25519 curve. Researchers will have to find out another way to work around key derivation for ed25519 because of this problem.

Ed25519's "bit clamping" problem

For performance reasons, ed25519 doesn't provide a prime-order group and provides a group of order h * l instead, where h is a small cofactor of 8 and l is a 252-bit prime (Section 2.3). In short, a cofactor is one of the parameters that make up an elliptic curve. The number of points on the elliptic curve can be described as n = r * h where r is the prime order of a subgroup and h is the cofactor.

The method of using a cofactor to boost a better performance [15] has proven to be a source of issues and vulnerabilities in higher-layer protocol. If the cofactor is 1, like in many of the standardized curves, it's not something we need to worry about. If it is not one, careful consideration needs to be made for its implications in cryptographic schemes to avoid a few types of attacks, notably small-subgroup attacks [16] and more announced malleability attacks (For example in Monero Blockchain [17]).

To prevent this, ed25519 performs a "bit clamping" on step *set/clear bits* in Figure 3.10 as follow (bytes represented in little-endian form):

- input[0] &= 248

  Setting the lowest 3 bits to 0 makes the private key the multiple of 8. This is only useful to prevent small-subgroup attacks in Diffie Hellman key exchange [18], but the creator of ed25519 makes it a global schema. In our thesis, we don't use DH in any of our processes, so we won't discuss this problem further. Clearing these 3 bits does not roughly reduce the keyspace since it still leaves 252 higher bits, but if we continue to clamp through the process, this will result in big security issues (see Section 3.1.1.3).

- The highest bit of the last byte is cleared, and the second-highest bit of the last byte is set to 1.

  Gregory Maxwell (Bitcoin core dev, CEO of Blockstream) said it would prevent the scalar from exceeding the order of the group [19]. Furthermore, it's also a performance hack so that the exponentiation ladder does not need to correctly handle the point at infinity.

> Later on, Daniel J. Bernstein (djb), the creator of ed25519, explained in a mailing list post from 2014 that some implementations implement it in variable-time based on the position of the highest bit. To avoid implementations having to care about this, he decided to make this part of the standard so that if the implementation is variable time in that way, it will run in constant time because of this clamping [20]. This prevents side-channel attack vector via timing leakage on weak scalar multiplication implementations [21].

In conclusion, pruning the scalar generated from the private key changes the linearity of the system completely. SLIP10 avoids these issues because it doesn't try to support non-hardened parent public key to child public key derivation and only supports hardened private parent key to private child key derivation when used with the ed25519 curve. Also, SLIP10 doesn't use any hardened derivation containing "bit clamping" in the process. The randomly generated child keys act as a seed in the next generation. In the signature generation, they don't have to prune the secret key the second time. SLIP10 remains the most implemented key derivation structure that the blockchain community uses and is recommended to use.

3.1.1.3 SLIP23

SatoshiLabs Improvement Proposals number 23 (SLIP23) is about Cardano HD master node derivation from a master seed, but also documented child key derivation function, which is based on Dmitry's paper BIP32-Ed25519 [7]. We use SLIP23 instead of CIP03 (Cardano Improvement Proposal) because CIP is merely to document the existing standards and not to provide rationales for the various methods used in key derivation. For a long time, Cardano hasn't updated documentation on this topic. Documentation about BIP32-ED25519 on the Cardano website is also removed. This section describes a hierarchical deterministic key derivation approach for curve ed25519 that overcomes the security issues mentioned in BIP32.

In practice, there is much confusion in implementing the original paper to their library (or application). In every next part, we will present the most general ideas of the original paper.

Master Extended Key generation function (root key in the paper)

This scheme adopts the master node derivation used in BIP32 and SLIP10 but defines a new Key input "ed25519 cardano seed" for the Cardano deterministic key hierarchy. They implement ed25519's "bit clamping" to the derivation schema. Figure 3.11 describes the process.

Pseudo-code:

- Let S be a seed byte sequence such as the master secret from BIP39.

  The length of the seed is from 128 bits to 512 bits.

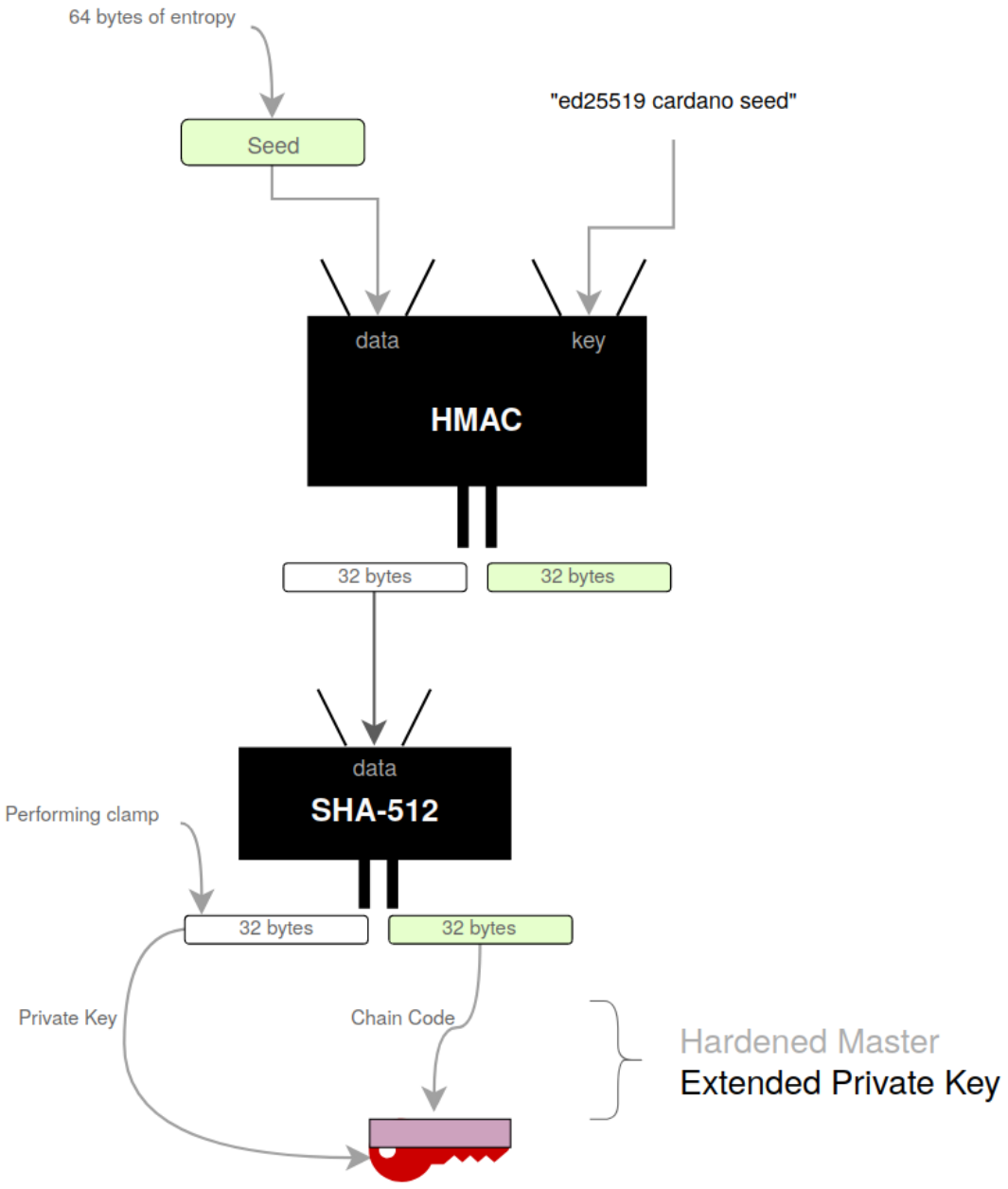

Figure 3.11: Master key generation process

- Calculate I := HMAC-SHA512(Key = "ed25519 cardano seed", Data = S).
- Split I into two 32-byte sequences, I_L as 32 bits left and I_R as 32 bits right.
- Let k := SHA-512(I_L).
- Split k into two 32-byte sequences, k_L as 32 bits left and k_R as 32 bits right.
- If the third highest bit of k_L (which is $k_L[29]$) is not 0, modify k by assigning:

  - $k_L[0]$:= k[0] & 0xf8
  - $k_L[31]$:= ($k_L[31]$ & 0x1f) — 0x40.

    This step is compatible with "bit clamping" in ed25519.

- Let k_R := k[32:64] and use (k_L, k_R) as the root extended private key and c := SHA-256(I_R) as the root chain code.

Child key derivation function

Surprisingly, BIP32-ED25519 allows both normal and hardened derivation. We can even generate hardened derivations on public keys since the linearity is ignored. With B is the generator, the next parts will generalize the derivation processes.

Private key derivation

Let $k = (k_L, k_R)$ be the extended private key and K the public key. Extended private child key $k_i = (k_L, k_R)$ for child i is produced as Figure 3.12.

Pseudo-code:

- Check whether $i \geq 2^{31}$ (whether the child is a hardened key).
  - If so (hardened child):

    let Z = HMAC-SHA512(Key = c_{par}, Data = 0x00 $\|$ k_{par} $\|$ ser_{32}(i)).
  - If not (normal child):

    let Z = HMAC-SHA512(Key = c_{par}, Data = 0x02 $\|$ $ser_P(K_{par})$ $\|$ ser_{32}(i)).
    The value $ser_P(K_{par})$ is represented as a little-endian string of 32 octets defined in ed25519 signature [22].

- Split Z into two parts: the left 28-byte sequences Z_L and right side 32-byte sequences Z_R.
  - let $k_L = 8Z_L + k_{parL}$. If k_L mod n (base order) = 0: return false.
  - let $k_R = Z_R + k_{parR}$ (mod 2**256)

- The returned child private key (k_L, k_R).

- Check whether i $\geq 2^{31}$ (whether the child is a hardened key).
  - If so (hardened child):

    Return ci = HMAC-SHA512(Key = c_{par}, Data = 0x01 $\|$ k_{par} $\|$ ser_{32}(i)).
  - If not (normal child):

    Return ci = HMAC-SHA512(Key = c_{par}, Data = 0x03 $\|$ $ser_P(K_{par})$ $\|$ ser_{32}(i)).

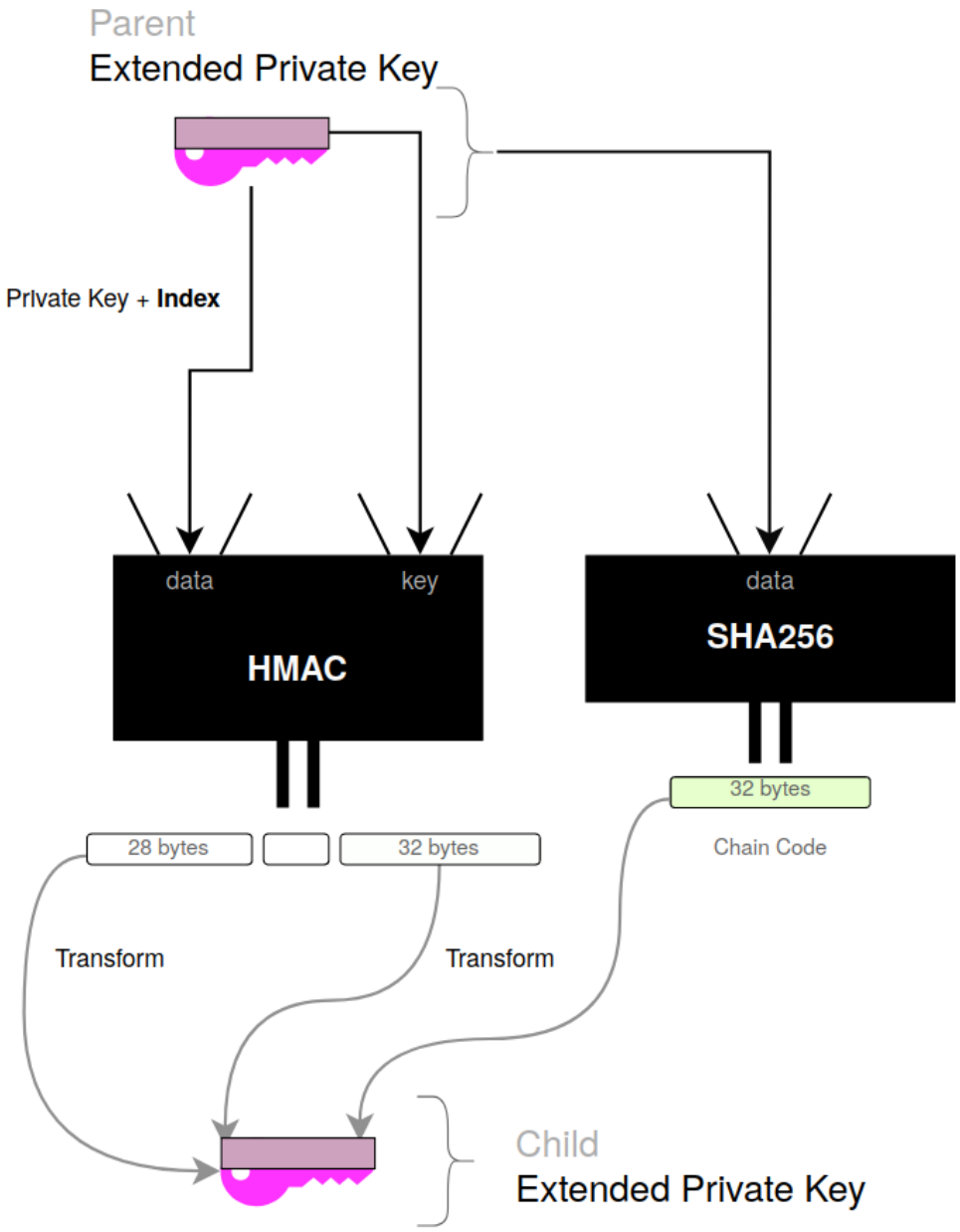

Figure 3.12: Private keypair derivation process

- The child public key K_i can be derived as $K_i = k_L\mathrm{B}$.

Public key derivation

Same as BIP32, the SLIP32 schema allows us to generate hardened and normal child keys from extended parent keys. Figure 3.13 presents the process of SLIP23 public key

derivation.

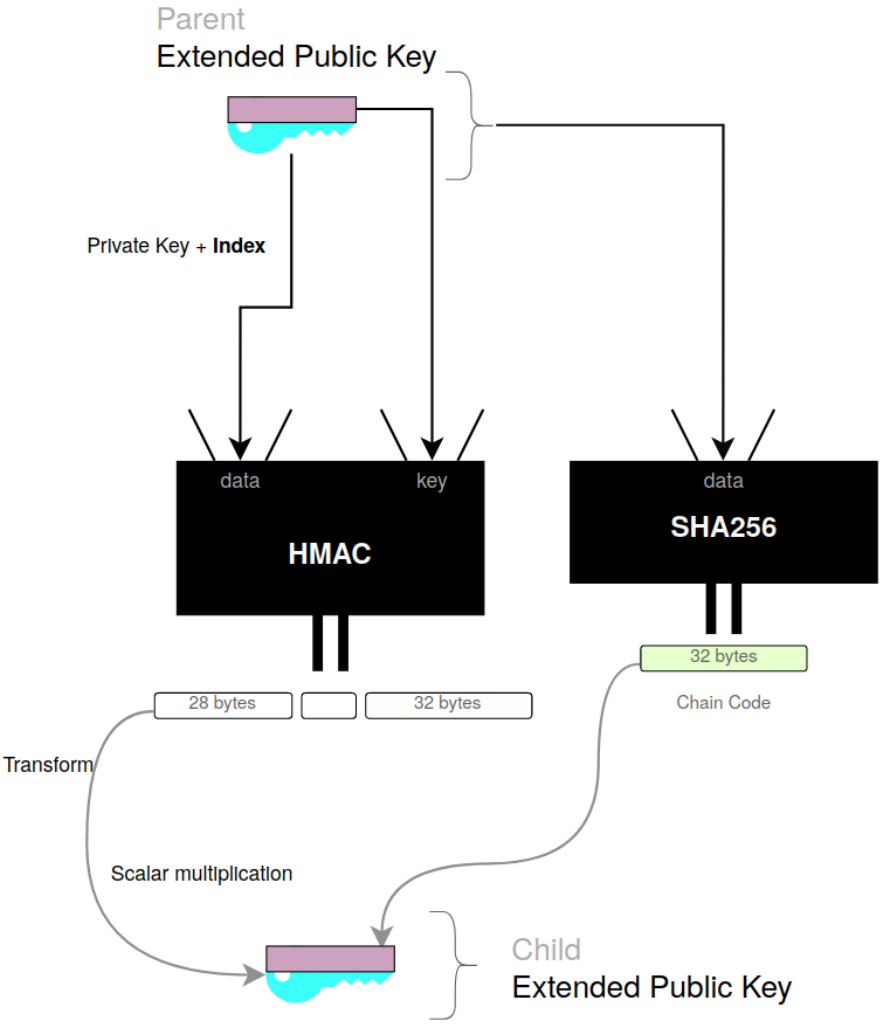

Figure 3.13: Public keypair derivation process

Pseudo-code:

- Check whether $i \geq 2^{31}$ (whether the child is a hardened key).
  - If so (hardened child):
    let Z = HMAC-SHA512(Key = c_{par}, Data = 0x00 $\|$ k_{par} $\|$ ser_{32}(i)).
  - If not (normal child):
    let Z = HMAC-SHA512(Key = c_{par}, Data = 0x02 $\|$ $ser_P(K_{par})$ $\|$ ser_{32}(i)).

- Split Z into two parts: the left 28-byte sequences Z_L and right side 32-byte sequences Z_R. (No usage of Z_R)

- Calculate the child public key K_i:

  K_i = Kp + $8Z_L$B. If K_i equals to the identity point (0,1) return false.

- Check whether i $\geq 2^{31}$ (whether the child is a hardened key).
  - If so (hardened child):

    Return ci = HMAC-SHA512(Key = c_{par}, Data = 0x01 $\|$ k_{par} $\|$ ser_{32}(i)).
  - If not (normal child):

    Return c_i = HMAC-SHA512(Key = c_{par}, Data = 0x03 $\|$ $ser_P(K_{par})$ $\|$ ser_{32}(i)).

In summary, the above methods allow the derived child private keys to correspond to the child public key. The advantages described in Section 3.1.1.1 are enabled. The authors gives us an working derivation schema but they didn't explained why did they choose spec$_i$fic metrics for the schema. For example they shortened to 28-byte of Z_L for the child keys. The Z_R of public key derivation is completely unused.

Security concern

The BIP32-ED25519 key leakage problem

Same as BIP32, the problem is if the attackers have one of the private child keys, can they recover all of the other wallets? The answer is no thanks to the hash function HMAC-SHA512 between the secret parent key and its children. Therefore, the extended key can be guessed by brute-force 2**256 different master secrets. But the number of recovered wallets is only limited in this case. They can retrieve only the parent keys they derived from and then generate all branches of child keys from that node. This may lead to a fundamental leakage problem if the implementers fail to check the depth of child wallets. The implementers also have difficulties reviewing all the cases, even harder to keep track of the depth when more child wallets are produced. This case raises a problem in our thesis where we allow users (who have no ideas of the security issues) to choose their index and depth of the tree. Hardened derivation of BIP32-ED25519 also is vulnerable to this attack vector. Furthermore, using the public key as a part of the function to derive a child private key won't serve the standard properties that the SLIP10 hardened key provided.

The BIP32-ED25519 bit clamping before signing problem

Jeff Burdges (Web3 Foundation, Polkadot Blockchain), who is famous for finding the vulnerabilities in the BIP32-ED25519 proposal, warned us about a potential attack

vector occurring under the many time the derivation function clamping. Ed25519's bit clamping is only designed for preventing attacks in the signing method and key exchange. According to Burdges, clamping the bits doesn't make the key more secure; instead, it makes the key weaker. BIP32-Ed25519 also uses only 224-bit scalar derivation (28-byte), so each level of derivation weakens the adequate strength of the computed scalar value. Burdges also mentioned a straightforward full key recovery attack if one permits long key derivation paths along with either clamping by an Ed25519 library [23].

In conclusion, there are three most notable key derivation methods that we mentioned. There is some more approach to HD key tree derivation like the Tor's approach [24]. It was released four years before the BIP32-ED25519. However, they haven't published any implementations as well as the test vector on the internet, making it very hard for the open-source researcher and the community to validate. We consider any project with this behavior as a fraud proposal and continue with our thesis.

3.1.2 BIP39 - Random Password Generator

BIP39 is one of the many design ideas that was approved by an economic majority of the blockchain community and became a standard for many popular wallets. It describes the implementation of a mnemonic sentence ("mnemonic code", "seed phrase", "seed words") – a group of easy to remember words – for the generation of deterministic wallets, making it more convenient for humans to store (compared to random raw binary or hexadecimal representations of a wallet seed). A mnemonic sentence is a way of representing a large randomly-generated number as a sequence of words. These words are then used to create a seed S, which can replace the root seed generated from PRNG in the BIP32 or SLIP10 section. The process of creating a random secret key is present in Figure 3.14.

Generate Mnemonic

The first of all steps is to generate entropy, which is our source of randomness. The entropy (ENT) we produce must be a multiple of 32 bits, allowing us to split the entropy up into even chunks and convert to words later on. The proper size of ENT is 128-256 bits, that's enough to make it impossible for two people to generate the same entropy. Entropy values must be sourced from a strong source of randomness. This means flipping a fair coin, rolling a fair dice, noise measurements etc.

Next, create a bit sequence of the SHA256 hash of ENT called checksum where n = ENT.length / 4. This value is appended to the end of ENT. Next, these converted bits are split into groups of 11 bits, each encoding a number from 0-2047, present as an index. Finally, we convert these numbers into words from the BIP39 wordlist and use these words as a mnemonic sentence.

Mnemonic to Seed

To create a root seed from the mnemonic, BIP39 uses the PBKDF2 function with the generated mnemonic sentence (in UTF-8 NFKD) used as the password and the string

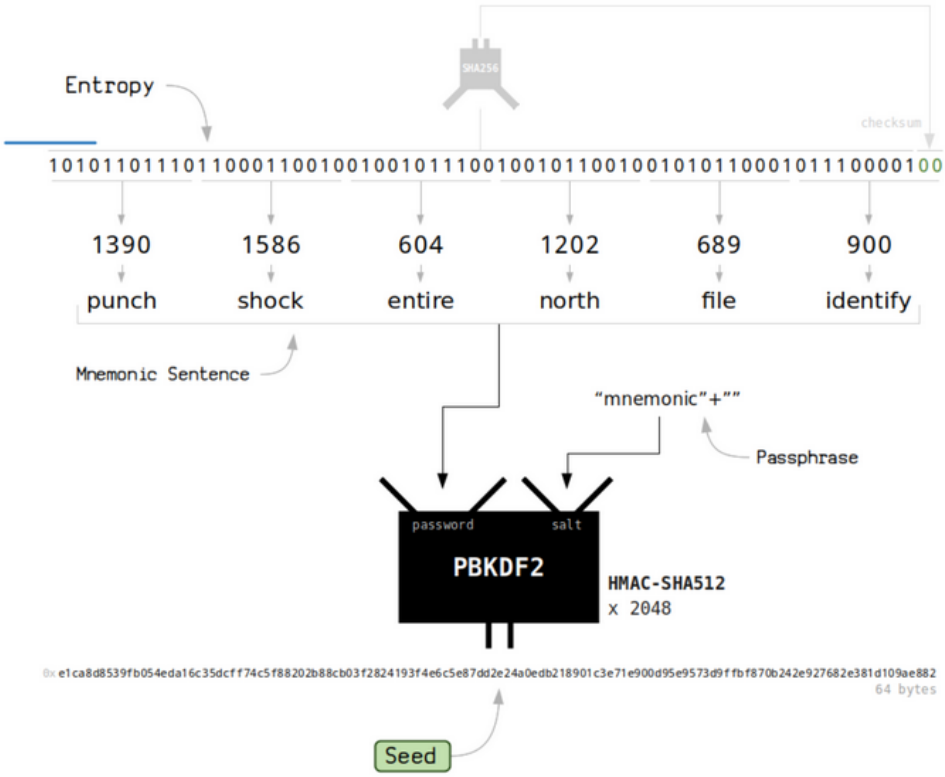

Figure 3.14: Random secret seed generation process

"mnemonic" + optional passphrase (UTF-8 NFKD) as the salt. The iteration count is set to 2048. The HMAC-SHA512 is used as the pseudo-random function, providing the length of the derived key is 512 bits (64 bytes).

3.1.3 BIP44 - Index Path for Multi-coin Wallet

BIP-44 defines an index path for key derivation in BIP32, allowing a wallet to handle multiple coins, multiple accounts, external and internal chains per account, and millions of addresses per chain. In BIP44, Bitcoin registers their coin type as 0 and their testnet as 1. Later on SLIP44 [25], other blockchains start to register their type too. Right now there are more than 2000 indexes (each index represents a blockchain) on SLIP44 documentation.

Wallet Structure

Each node of structure has a different meaning but we will use BIP44 as a standard path for key derivation. Figure 3.15 presents the initial idea of how BIP 44 path look like.

BIP44 defines a logical hierarchy for deterministic wallets. Each level of the hierarchy has a meaning, describe as follow (see also Figure 3.16):

- **m (or M)**: The master private key generated from root seed.

```
m / purpose' / coin_type' / account' / change / address_index
```

Figure 3.15: Example path of BIP44

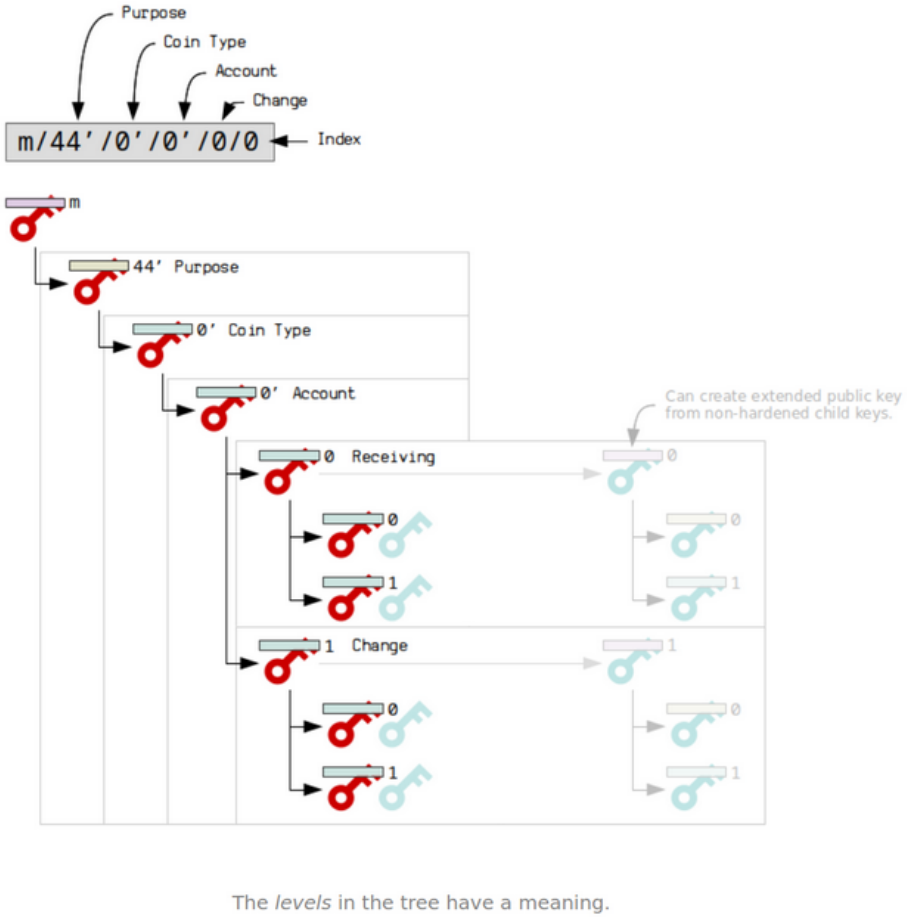

Figure 3.16: Example tree of wallets from given path

- **purpose**: Purpose is set to 44' constant (or 0x8000002C). It meaning that the child keys of this node is used according to BIP44 specification.
- **coin type**: One master secret seed can be used for unlimited number of independent cryptocurrency such as Bitcoin, Solana or Ethereum. However, sharing the same space for various blockchain may has some disadvantages. Creating this level makes sure a separate subtree for every blockchain, avoiding reusing the same addresses across cryptocoins and improving privacy character. Coin type is a constant, set for each blockchain. Blockchain developers will ask for registering an unused number for their blockchain (see [25]).
- **account**: An parameter that separate the path for funds. This node splits the key range into independent user identities, so the wallet avoids mixing the cryptocurrency across different accounts. Users can use these wallets to organize the funds in the same fashion as bank; for donation purposes, for saving purposes, for common expenses etc.

- **change**: Constant 0 is used for external chain and constant 1 for internal chain (also known as change addresses). External chain is used for addresses that are meant to be visible outside of the wallet (e.g. for receiving payments). Internal chain is used for addresses which are not meant to be visible outside of the wallet and is used for return transaction change.
- **index**: Addresses are numbered from index 0 in sequentially increasing manner. This number is used as child index in BIP32 derivation.

3.2 Related Works

The HD wallets on curve ed25519 were quite a challenge for the community. There has been quite an effortless number of implemented open-source libraries and evaluated by experts in cryptography. We do not try to find and analyze all applicable works. Instead, we focus on the notable library and the application that is generally used. We divide into two categories as the core library and real-world implementation.

Library implementation

The SLIP10 approach is widely used because of its easy understanding process and strict security. *christsim*'s Typescript library [26] followed all the instructions of SLIP10 and provided all derivation features of the three curves in SLIP10. However, it lacks test vectors on ed25519 keys. The code seems to be redundant since using class objects but never making use of OOP structure.

alepop / ed25519-hd-key (github) [27] offers a simpler implementation of SLIP10 using functional programming. It is only built for ed25519 key generation. The library provides valid result keys or input path checking and also gives the user permission to change the value of the offset (range of hardened keys).

chatch / stellar-hd-wallet is an HD wallet library designed for the Stellar (blockchain) application. It implements full hardened derivation of the SLIP10 version and supports multiple languages for mnemonic code. The library lacks a Stellar address extraction from the public key.

On the other hand, due to the missing references and test vectors, BIP32-ED25519 struggles with many incompatible implementations.

vbmirthr's OCaml library [28] and *islishude*'s Go implementation [29] follow BIP32-ED25519 original master key derivation specification and use SHA512 and SHA256 for deriving the private key k and chain code c (respectively) from the master secret S.

LedgerHQ's Python implementation [30] uses HMAC-SHA512 and HMAC-SHA256 instead. Furthermore, vbmirthr's instructions to discard the seed (i.e. master secret) if the master key's third-highest bit of the last octet of k_L is not zero. Meanwhile, islishude's Go library just follows BIP32-Ed25519 papers and clears the master key's third-highest bit. LedgerHQ's implementations repeatedly set the seed and feed the input to the master key generation and restart the process until a master key meets the desired properties.

Real-world application

Interestingly, the IOTA ledger combines Ed25519 as a second signature scheme with the currently used Winternitz-OTS [Ref] but uses SLIP10 as their key derivation schema. They aim to address all the points above with the drawback of being less quantum robust. The digital signature is called a bundle, and they prove it is sufficiently immune against a powerful quantum computer running Shor's algorithm [31] when addresses are not reused.

Sollet [32] is a famous open-source non-custodial and HD wallet for users to manage assets on the Solana blockchain safely. Users can access the wallet by their browser. The wallet provided multiple types of token (only on chain) management. But the user only has access to one wallet at a time. If they want to use another, they will have to log out and choose a different index. The wallet is mainly used by developers for testing.

The Polkadot community recommends their wallet developers implement SLIP10 since they haven't prioritized investigating into Bip32-Ed25519. Jeff Burdges research on Polkadot wallet security refers to BIP32-ED25519 as a "zombie" [33] and suggests their cryptosystem should not use this as HD wallet core.

Oasis [34] on the other hand don't use use BIP32-Ed25519's private and public key directly but use the obtained k_L (first 32 bytes) of the 64 byte BIP32-Ed25519 derived private key as Ed25519's seed (i.e. non-extended private key). But in Mar 2021 they decided to change from BIP32-ED25519 to SLIP10 schema, leaving it "not applicable in future".

The BIP32-ED25519 has been very poorly adapted in practice ever since Jeff Burdges warning [23]. But it is still very useful for cold wallets. Hardware wallets Ledger and Trezor implement both SLIP10 and BIP32-ED25519 to their key generation schema. The advantages of public-key derivation impactfully remain. Attacks aim to steal private keys from hardware wallets but are still shallow. Users can publish their master public key to the networks, while child keys can be kept safe in users' pockets. If ever the transaction is needed, users just need to derive child keys from their hardware and sign the transaction. Plus, Ledger and Trezor are better than other wallets since they are custodial type wallets and can hold multiple types of native tokens.

4

System design

This chapter describes the process of designing our implementation. We divide our work into creating an open-source library that can help developers handle the primary function of a HD wallet (key generation, transaction integration, balance checking, etc.) and a Web Wallet for cryptocurrency users.

Contents

4.1 The Library **66**

4.2 The Hierarchical Deterministic Web Wallet **68**

4.2.1 Cryptographic mechanism 68

4.2.2 System Architecture 72

4.2.2.1 User and Web Service 72

4.2.2.2 User and Blockchain 72

4.2.2.3 User and Address Service 73

4.2.3 Information Privacy and Security concern 75

4.2.4 Use Cases 75

4.2.4.1 Use Case diagram 75

4.2.4.2 Use case table 76

4.1 The Library

Our work does not only share an identical problem with the problem statement. We realize that we could do better. With the support of SatoshiLabs proposal number 44 (BIP44), we can create wallets that hold multiple native tokens of different blockchains. We decided our HD wallet will support the blockchain using ed25519 as well as secp259k1 (two famous curves of the blockchain community). They are Bitcoin, Ethereum, and Solana. As we researched, there are almost no non-custodial hot wallets that support different kinds of coin holders. Even the existing libraries limit their support to a specific blockchain.

Also, we believe anyone should have access to any of the non-custodial wallets so they can be bug-free. Hence, we came up with an open-source library for the other developers. They can use it, contribute to it (merge more blockchains, etc.) and audit our works. Furthermore, we will use this library as a core functionality that supports the cryptography mechanism in our web wallet.

Library structure

The library is supposed to be easy to maintain, read and contribute to. We organize as Figure 4.1:

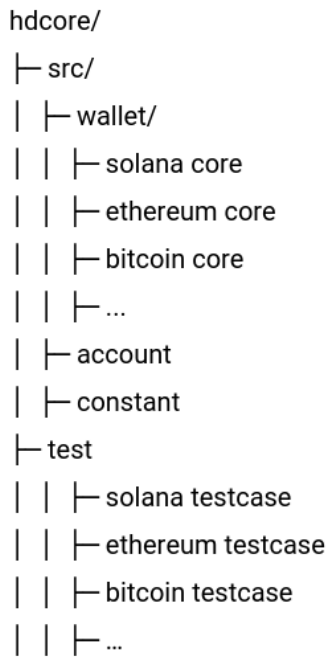

Figure 4.1: General library structure

We name our library "hdcore" meaning the cores of HD wallets. The library is divided into two parts.

- The /src folder comprises the /wallet folder which is the implementation of API for each chosen blockchain. There is also a constant file including all general information about the blockchain (index, url, etc.). And an account file where we generalize the wallet folder, users can choose their blockchain wallet through this file.
- The /test folder contains test cases for every function.

API

The library will supports following features of a hd wallet:

- Mnemonic code generation
- Key pair generation
- Key tree derivation
- Address generation (for each blockchain)
- Verification of keys and addresses
- Balance checker
- Transaction builder (signing and broadcasting the transaction)

Unit testing

For testing our library, we use test vectors. A test vector offers a compact way of defining test input/output. It is a set of test cases where each test case is defined as a set of actions and expectations to verify our library's particular feature or functionality.

A test case contains steps as in Figure 4.2. The test case includes specific variables or conditions, using which a user can compare expected and actual results to determine whether the library behaves appropriately as per their requirements.

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass/Fail

Figure 4.2: Testcase components

The test vector should show the functionalities of our library (declared in API).

Security

The library will acknowledge the security flaws, and we will look into this seriously. We focus on the coding concern, discussing how we can replace poisoned libraries, and prevent the addition of new vulnerable libraries to our code.

We discuss our properties from different perspectives:

- Information leaking: Part of the API supports the connection to the blockchain (for broadcasting the transaction); therefore, we have to make sure the library would not send any sensitive or redundant information.
- Library implementation: We will carefully analyze the library dependencies if we choose to use them, even benchmarking them if necessary.
- Programming language: We will discuss the safety of the language we choose to implement.

Our library will be published to the related package manager, so open-source users and developers can audit and give us their thoughts.

Documentation

The goal of our documentation is an overview and usage exploration of our library. Thus, the documentation comprises all the expected results of the main functions, overlook of test vectors, and how to apply to some specific blockchains.

4.2 The Hierarchical Deterministic Web Wallet

We decided to create a web wallet for our thesis so we don't have to deal with version control and different kinds of operating systems (Android, iOS, etc.). Everybody can access our website with their browsers.

Our web wallet is a non-custodial hot wallet where users can sign in, generate accounts in different blockchains, and send and receive native tokens (see Figure 4.3). They have absolute controls over private keys for every blockchain address, and the wallet provider has no access to them. We will use the localhost environment as a server for the wallet provider.

4.2.1 Cryptographic mechanism

The security of cryptography defines the strength of the blockchain wallet. Everything is protected by the laws of mathematics. We discuss which technology from Section [Ref technology] will be implemented for our wallet protocols.

Key pair and Address

In a blockchain system, the private key determines the ownership of the address property generated by the corresponding public key. Our generated key pair must be corresponding to each other. The private key is required to be functioned appropriately when the owner accesses and sends cryptocurrency. The address is an encoded or produced by cryptographic algorithm from the public key depending on the Blockchain cryptographic system. Our wallet addresses can be looked up on network explorers.

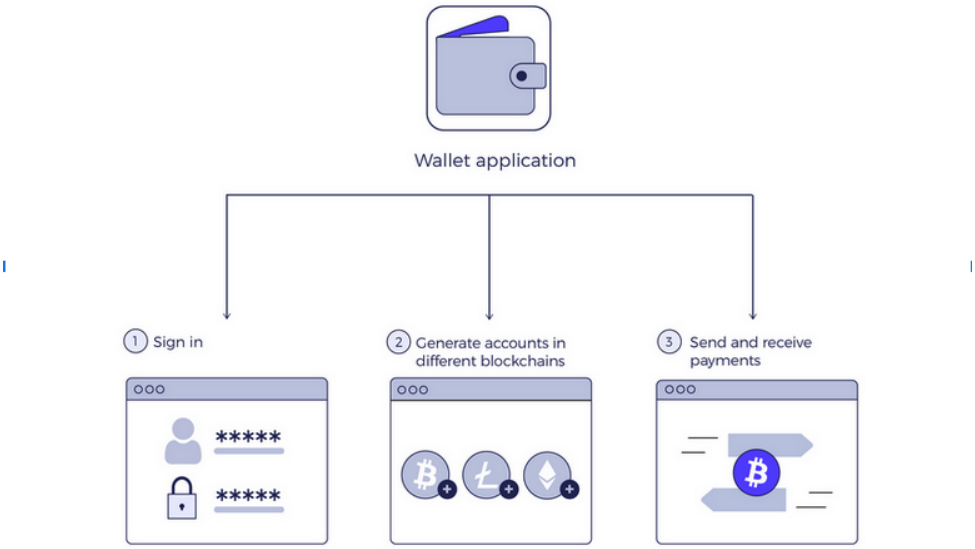

Figure 4.3: Example of Web Wallet flow

We will support Solana and Ethereum blockchain which utilises secp256k1 and ed25519.

Password generation

A password or a mnemonic code is the secret that users have to back up and keep away from the internet. From the code, users can recover the whole wallet tree from a single seed due to all the keys of a wallet being derivable from a single plaintext string. With this property, even key transfer and management are very convenient. We will implement BIP39's mnemonic sentence process for our wallet since it is the industrial standard for seed wallets. The architecture of the password generation is described in subsection 3.1.2.

Digital signature generation

We use both secp256k1 and ed25519 digital signature algorithms to sign our wallet's transactions.

Key derivation

Derivation function is the most important part of our wallets. We presented three of the most famous schemas in Chapter 3. In order to match the standard of cryptography technology to achieve a securely appropriate application, we will adapt hardened key derivation of both BIP32 (for secp256k1) and SLIP10 (for ed25519) for our wallet. Our key derivation tree will look like Figure 4.4

The BIP32-ED25519 and normal key derivation of BIP32 have serious security flaws, as we explained in Section 3.1.1.1. Furthermore, the schema of public key derivation is only applicable to cold wallets where your private keys can be kept in a disjoint hardware while the master public keys can be broadcast to the internet.

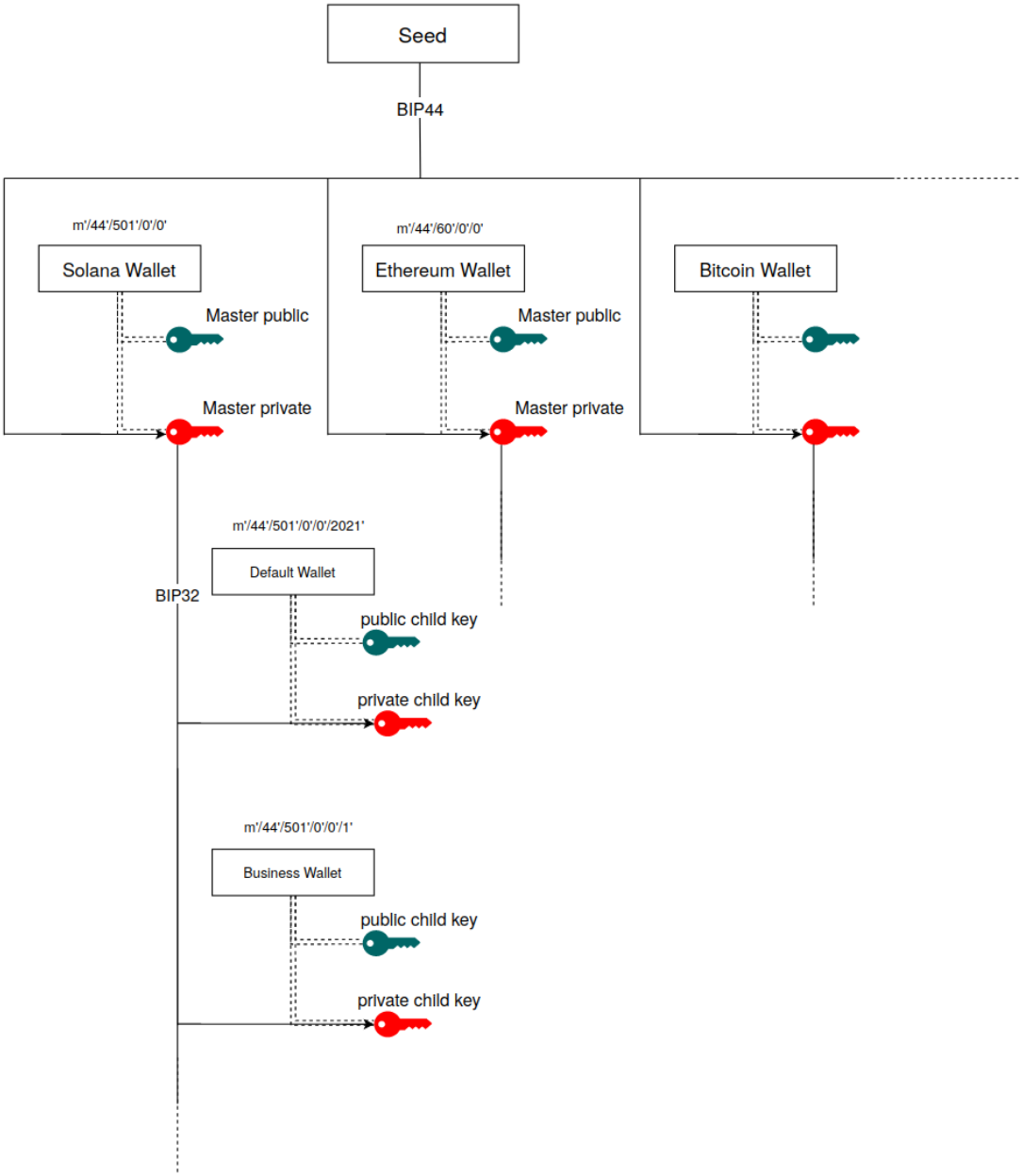

Figure 4.4: HD Web Wallet tree

Solving the normal key derivation problem

By applying the hardened key version, we will lose the magical ability of public key derivation. With the effort of optimizing the wallet, we came up with an idea that can replace this advantage. First, we discuss the disadvantages of BIP32 normal key derivation in our multicoin wallet:

- It still has serious flaws, as we mentioned (Section 3.1.1.1). All of the public key derivation systems will suffer this problem. Users cannot afford to give or lose any child keys since the attackers can recover the entire purse. Therefore cannot use this schema for any business operations.

- Users have to generate every master public key for every supported curve. For example, we support ed25519 and secp256k1, and our wallets will produce two different master public keys (*mpub*) for two curves. Somebody who wants to send coins to us will have to save our two *mpub*, look for the *mpub* of a specific currency, generate the child address, and then ship the coins plus the index to that child. This restricts the scalability of our wallets if we're going to support more blockchain. It also affects users' experience if they have more connections (more *mpub* to save).Our idea is explained as follow: We create a server to retrieve all the allowed addresses from our user wallets and store it as a tree structure. We call it "Address service" (details in Section [Ref]). The purpose of this is to recreate the process of public key generation but in a more convincing way. Senders can perform an address look-up on the server to get the receiver's currency address.

Our idea is explained as follow: We create a server to retrieve all the allowed addresses from our user wallets and store it as a tree structure. We call it "Address service" (details in Section 4.2.2). The purpose of this is to recreate the process of public key generation but in a more convincing way. Senders can perform an address look-up on the server to get the receiver's currency address.

- The server only holds customer addresses. The tree structure helps to optimize address searching.
- The server has no access to user transactions and assets. Users broadcast transactions straight to the blockchain network. The decentralized nature remains for our wallet.
- Private keys are still in control of user browsers.

The advantages of the server are listed as follow:

- Avoid the flaws of the BIP32 public key derivation completely.
- Senders only need to know one master public key of the receiver for the first time.
- Users have complete control of what wallet they want to show the internet. In BIP32 normal key derivation, if others have your master public key, they can generate the whole wallet public keys.
- Other users can't figure out the index of other wallets since there are only addresses in the server's database.

Also, since the server holds no sensitive information, the attacks aimed to spoof the user's network (MITM attack [35]) are useless.

4.2.2 System Architecture

4.2.2.1 User and Web Service

In practice, our system requires a server to host the RESTful web service to handle the requests from users. Users can access our wallet interface through their browser by entering the domain of our hosting service.

The protocols between users and our server describes as Figure 4.5:

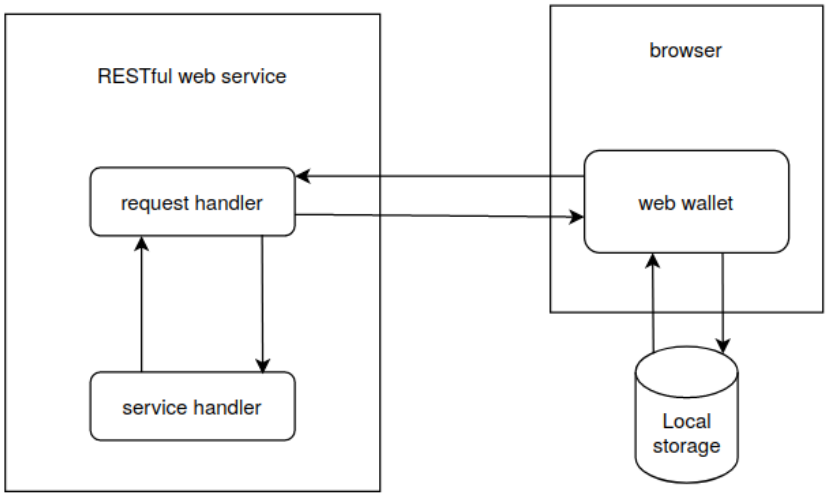

Figure 4.5: Between users and Wallet Provider

The server-side contains the request handler and service handler. The request handler acts like an endpoint where users send in their requests. Then parse the parameter and transfer it to the service handler. The service handler will read the information and return the content of the webpage (HTML, scripts, etc.) to the users.

The client-side (user browser) accesses the wallet through a GET request to the server. All users' critical information will be saved at their local storage, like master private keys or passwords and they should be encrypted.

4.2.2.2 User and Blockchain

If the users want to make a transaction they will connect directly to the blockchain networks. Before building a transaction, users will decrypt the private key from their local storage. The private key will be used to sign the transaction. The transaction will be broadcast directly to decentralized networks without going through the server.

The protocols described as Figure 4.6:

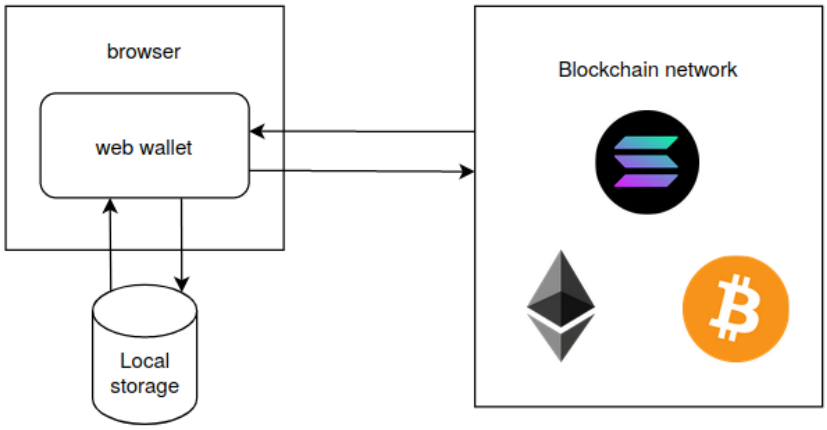

Figure 4.6: Between users and Blockchain network

4.2.2.3 User and Address Service

We created another server to handle "Address service". Note that we can combine this server with the Web Service Server above. But they served different purposes, and we don't want our system to be synthetic. Furthermore, this separated architecture will be easier to update and test.

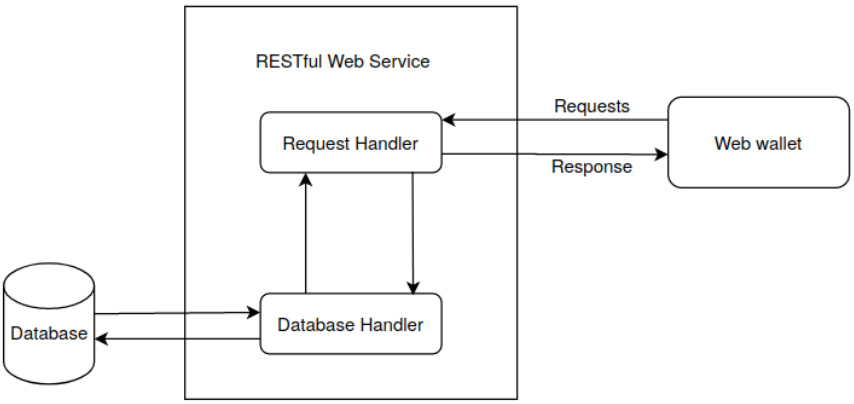

Figure 4.7: Between users and Address Service

The users after creating their wallet will create a POST request to the server with their automatically created public keys in JSON. The JSON data contains the address of master wallet and default child wallets. We created the term "default child" just for init the hierarchical tree. The structure of the JSON request present as Figure 4.8.

The database handler will save the JSON object from the request handler. The reason we choose the JSON object is that it is easier to be updated by the users. Furthermore, they are optimized by the search engine. We recommended using a NoSQL database program for this architecture, especially the one that utilized collections and documentations.

```
{
    "id":       [Hash(master address)],
    "address":  {
                    "[blockchain index 1]" :  {
                                                  "business 1" : [address 1],
                                              },
                    "[blockchain index 2]" : {
                                                  "business 1" : [address 1],
                                              },
                }
}
```

Figure 4.8: Request Data

The database is designed just like the JSON object, but holds more values (see Figure 4.9)

```
{
    "id":       [Hash(master public key)],
    "address":  {
                    "[blockchain index 1]" :  {
                                                  "business 1" : [address 1],
                                                  "business 2" : [address 2],
                                                  ... (more wallet address)
                                              },
                    "[blockchain index 2]" : {
                                                  "business 1" : [address 1],
                                                  ... (more wallet address)
                                              },
                    ... (more blockchain index),
                }
}
```

Figure 4.9: JSON object of a Wallet

4.2.3 Information Privacy and Security concern

Our browser-based crypto wallets give users complete control of their keys and therefore their funds. We guarantee the reliability of the website and the system behind it. The web provider will not hold any sensitive information or require any of the user's identification. On the other hand, the user needs to protect their secret and the hardware (or other place) where they keep them. In extreme cases, for example, the attacker takes over the user's computer (anything they access the wallet with), we can't do anything about it.

To design a web application with a priority on system security, we decided to take OWASP as a recommendation for yielding the foundation. OWASP is a non-profit global association that intends to promote application security across the web. It emerged as a standard awareness documentation for developers and web application security. OWASP strives to improve software security through its participating knowledge, open-source projects, material related, and thousands of community members. A security testing experiment will be done after we based on OWASP's top security risks that can be related to our project in Section . From high to low risk, they are:

1. Broken Access Control.
2. Cryptographic Failures.
3. Injection.
4. Insecure Design.
5. Vulnerable and Outdated Components.

The security tests will be examined with specific aims and different tools. We will try to cover all security threats in our system or at least protect our wallet from common security holes that can directly harm the end users.

4.2.4 Use Cases

4.2.4.1 Use Case diagram

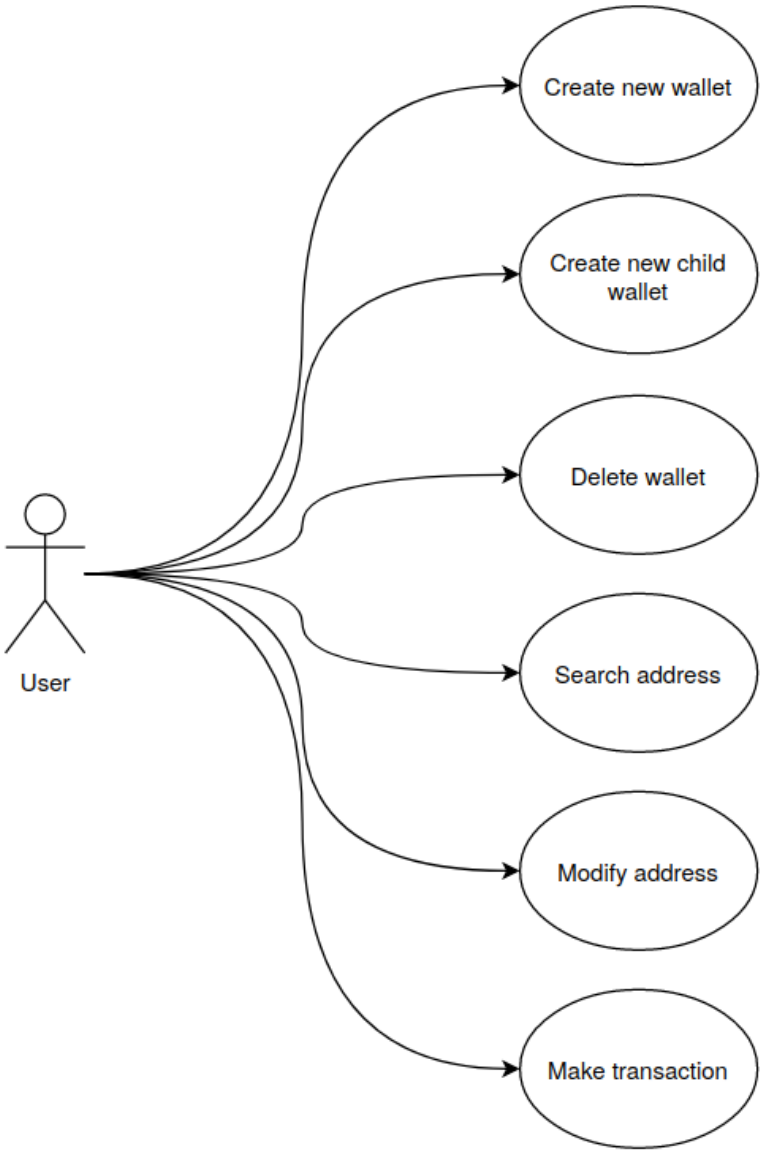

Figure 4.10: Usecase diagram

4.2.4.2 Use case table

Use case detail

Use case name	Table reference
Create new wallet.	Table 4.2
Create new child wallet.	Table 4.3
Delete wallet.	Table 4.4
Search address.	Table 4.5
Modify address.	Table 4.6
Make transaction.	Table 4.7

Table 4.1: Use case List

Use Case Name:	Create new wallet.
Event:	The user accesses the wallet page.
Description:	The user chooses to create or recover their HD wallet and go on to their wallet page.
Preconditions:	• Application has an internet connection.
Post-conditions:	• Default wallets are generated for the user. • The user gets to their wallet page. • The user is able to perform actions with their wallets.
Normal flows:	1. The user accesses the homepage from the browser. 2. Web shows two options: "Create new wallet" and "Recover wallet". 3. Users choose "Create new wallet". 4. Web shows 12 or 24 mnemonic words. 5. The user clicks on the create wallet button or refreshes the mnemonic code until they are satisfied with the code.
Alternative flows:	• 3a. TThe user chooses "Recover wallet". – 3a.1. Web shows an input form. – 3a.2. Users enter their mnemonic code.
Exception:	• 3a. The user enters an invalid mnemonic code. – Web warns the users and tells them to enter another code.

Table 4.2: Use case: Create new wallet.

Use Case Name:	Create new child wallet.
Event:	The user chooses “Derive wallet” button to create a new child wallet.
Description:	The user chooses to derive a new child wallet from the chosen blockchain.
Preconditions:	• Application has an internet connection. • The user already accesses the wallet page.
Post-conditions:	• A child wallet is generated for the user. • The user is able to perform actions with their child wallet.
Normal flows:	1. The user chooses a blockchain on the top bar. 2. Web shows that blockchain wallet page. 3. The user chooses “Derive wallet” button. 4. Web shows index input form. 5. The user enters the index for the child wallet and click ”Create”. 6. Web shows the child wallet.
Alternative flows:	N \ A
Exception:	• 5. The user enters an invalid index. – Web warns the users and tells them to enter another index.

Table 4.3: Create new child wallet.

Use Case Name:	Delete Wallet from the browser.
Event:	The user chooses “Delete this wallet” in the top bar.
Description:	The user no longer uses the wallet and wants to delete the wallet information on their computer.
Preconditions:	• Application has an internet connection. • The user already accesses the wallet page.
Post-conditions:	• Delete all local storage and session from the user browser.
Normal flows:	1. The user clicks on “Delete this wallet”. 2. Web warns the user if they are sure. 3. The user chooses “Yes”. 4. Web shows index input form. 5. Web delete all information from user’s local storage and browser session. 6. Return to homepage.
Alternative flows:	• 3. The user chooses “No”. – Web returns to the current stage.
Exception:	N \ A

Table 4.4: Delete Wallet from the browser.

Use Case Name:	Search address.
Event:	A bar pops up when the user clicks "Send" on the wallet component. To search for an address in the database, the user clicks "Look up".
Description:	The user want to make a transaction and look up the receiver address of a blockchain.
Preconditions:	• Application has an internet connection. • The user already accesses the wallet page.
Post-conditions:	• Return the receiver addresses of the chosen blockchain.
Normal flows:	1. The user clicks on "Send" of a specific wallet. 2. Web pop up a transaction box. 3. The user fills in the master address of the receiver. 4. Web returns a list of addresses of the receiver.
Alternative flows:	N \ A
Exception:	• 4. The address doesn't exist in the database, the web return zero address.

Table 4.5: Search address.

Use Case Name:	Modify addresses in the database (push or pull).
Event:	The user clicks on “Pull” or “Push” from the wallet component.
Description:	The user chooses which wallet to public to others.
Preconditions:	• Application has an internet connection. • The user already accesses the wallet page.
Post-conditions:	• The chosen wallet address is either deleted or added in the database.
Normal flows:	1. The user clicks on “Pull wallet”. 2. The web shows requests succeed and status removed from database.
Alternative flows:	• 1a. The user clicks on “Add”. – The web shows requests succeed and status added.
Exception:	N \ A

Table 4.6: Modify addresses in the database.

Use Case Name:	Make transaction.
Event:	The user clicks on “Send” from the wallet component.
Description:	The user wants to send coins to somebody. .
Preconditions:	• Application has an internet connection. • The user already accesses the wallet page.
Post-conditions:	• The coins is sent and the transaction is public to the blockchain explorer.
Normal flows:	1. The user clicks on “Send”. 2. Web pop up a transaction box. 3. The user enters the address of the receiver (by looking up or entering straight to the forms) and clicks “Confirm”. 4. Web shows transaction status and link to blockchain explorer.
Alternative flows:	N \ A
Exception:	• Web shows something wrong with the transaction and returns to the current stage.

Table 4.7: Make transaction.

5

Implementation

This chapter describes the implementation of the systems we designed and discusses the applied technology.

Contents

5.1 The Library **86**
5.1.1 Library core 86
5.1.2 Important dependencies 88
5.1.3 Library API 89
5.1.4 Documentation 89
5.2 The Hierarchical Deterministic Web Wallet **90**
5.2.1 Ecosystem 90
5.2.2 Address service 91
5.2.2.1 Technologies 91
5.2.2.2 API Provider 91
5.2.3 Web Wallet Service 96

5.1 The Library

Typescript

TypeScript is a strict superset of javascript, designed as a strongly typed programming language by Microsoft. A strongly typed language is more secure because it requires explicit declarations to convert or compare between types in order to convert between languages. This allows for the avoidance of errors and memory exploitations such as the CWE-704 Incorrect Type Conversion or Cast [36]. At the same time, it checks types at compile time to find any potential type errors before they are propagated into the program.

TypeScript uses a transpiler, a source-to-source compiler that takes TypeScript source and converts it into JavaScript. The TypeScript code ends up running as JavaScript. In conclusion, TypeScript does provide a set of constraints regarding typing, which enforces a higher level of security.

Tweetnacl-js

TweetNaCl-js is a translation of TweetNaCl to JavaScript, which is as close as possible to the original C implementation. It provides public key and secret key authenticated encryption, scalar multiplication, digital signatures, hashing, random bytes generation, and constant-time comparison. Tweetnacl was written by many cryptography experts.

Bernstein et al. [37] focuses on the security analysis of the NaCl cryptographic library. More specifically, the security impact of NaCl's design features, such as no data flow from secrets to load addresses, no data flow from secrets to branch conditions; no padding oracles; centralizing randomness; avoiding unnecessary randomness; extremely high speed; and cryptographic primitives chosen conservatively in light of the cryptanalytic literature, were evaluated. Hence, these features result in a high-security cryptographic and bug-free library.

5.1.1 Library core

Our library expands on the implementation of the secp256k1 curve and their blockchain transaction problem. For the sake of our thesis, we will only present the core function of the ed25519 blockchain since others will follow the same schemas and rules (just different dependencies).

Structure

Applying our design with Typescripts, we have the library model as Figure 5.1

We design our work for developers who only use it for one specific blockchain or implement multiple of them. We also acknowledge that every blockchain has different features (especially in transactions). For example, Bitcoin uses the UTXO model (individual coins)

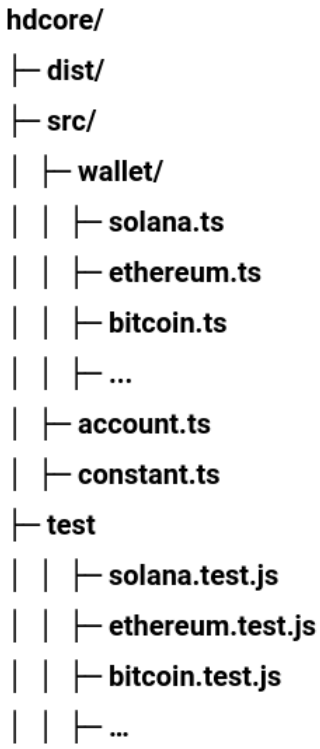

Figure 5.1: Typescript library structure

for wallet assets, while Solana uses the Account/Balance model. This led to different implementations in transaction functions. Aiming for upgradeability, readability, and ease of use, we organized as follow:

- We create a *COMPONENTS* JSON object (see Figure 5.2) with constant information of every supported blockchain in Figure 4.6. This *COMPONENTS* belongs to file */src/constant.ts* (.ts is the standard for Typescript file). We made use of the JSON architecture for legibility. Users can see right away what function a blockchain has and how to access them through a few lines of code. Unlike Tomi Jaga's wallet [38], they use an OOP structure which is very hard to read. Also, it would become more redundant when we created more blockchain dictionaries and added more features for each of every blockchain. Meanwhile, in our JSON object, developers can add more lines with different indexes and add the new function to their blockchain file (solana.ts, etc.).
- For the users who want to implement the functions generally, we create */src/accounts.ts* for this purpose. It includes the function to access these values and supports every element of the key derivation process. We present it in the library API part (see Section 5.1.3).

```
const COMPONENTS: any = {
  "0": {
    index: "0",
    hex: "0x80000000",
    symbol: "BTC",
    name: "Bitcoin",
    link: "https://bitcoin.org/",
  },

  "1": {
    index: "1",
    hex: "0x80000001",
    symbol: "BTCT",
    name: "Testnet (all coins)",
    key_pair_master: create_bitcoin_testnet_pair,
    get_address: get_bitcoin_testnet_address,
    transaction: bitcoin_tn_tx,

  },

  "501": {
    index: "501",
    hex: "0x800001f5",
    symbol: "SOL",
    name: "Solana",
    link: "https://solana.com",
    key_pair_master: create_solana_pair,
    get_address: get_solana_address
```

Figure 5.2: COMPONENTS JSON example

5.1.2 Important dependencies

To handle curve ed25519 arithmetic, we use the tweetnacl-js library as mentioned. The library uses Asymmetric Cryptography in NaCl that applied Bernstein's Curve25519 elliptic-curve Diffie-Hellman key exchange and will use the Ed25519 elliptic-curve signature scheme from Bernstein, Duif, Lange, Schwabe, and Yang as presented in Section 2.3. One thing is different; they didn't use the principle as we have shown for scalar multiplication with the ed25519 curve. They decided not to implement the conversion of points on ed25519 to Montgomery form and back. Instead, they perform a new ladder with a completely new addition and double points in one add function. The pseudo-code describes as in Figure 5.3:

```
set25519(p[0],gf0);
set25519(p[1],gf1);
set25519(p[2],gf1);
set25519(p[3],gf0);
for (i = 255;i >= 0;--i) {
  u8 b = (s[i/8]>>(i&7))&1;
  cswap(p,q,b);
  add(q,p);
  add(p,p);
  cswap(p,q,b);
}
```

Figure 5.3: Different implementation in Montgomery Ladder

5.1.3 Library API

The main API are listed as:

- **hdcore.account.createMnemonic()**

  Return a mnemonic code with 12 or 24 words.

- **hdcore.account.createSeed()**

  Return a master seed with 512 bits.

- **hdcore.account.createMasterAccount()**

  Return a master private key and public key.

- **hdcore.account.getPath()**

  Return BIP44 path (note that we only test with depth = 7). We also provided getFullPath function for who want to create longer path.

- **hdcore.account.createChildAccount()**

  Return child account with a specific path.

- **hdcore.account.getTransaction()**

  Return the **transaction** object, which has some functions below.

- **transaction.airdrop_one()**

  Request airdrop on testnet/devnet.

- **transaction.get_balance()**

  Request get balance of an account (for account-based blockchain).

- **transaction.send()**

  Broadcast transaction to the networks.

A few more function supports every detail to make the wallet work appropriately (for example, address validation, public key validation, transaction building, etc...).

5.1.4 Documentation

Our documentation can be found at *npm* (here).

5.2 The Hierarchical Deterministic Web Wallet

We give an overview of our system technology and flow in Section 5.2.1, presenting the details of Address service and the web wallet in sections 5.2.2 and 5.2.3 as Address service and the Web Wallet, respectively.

5.2.1 Ecosystem

We implement our design as in Figure 5.4.

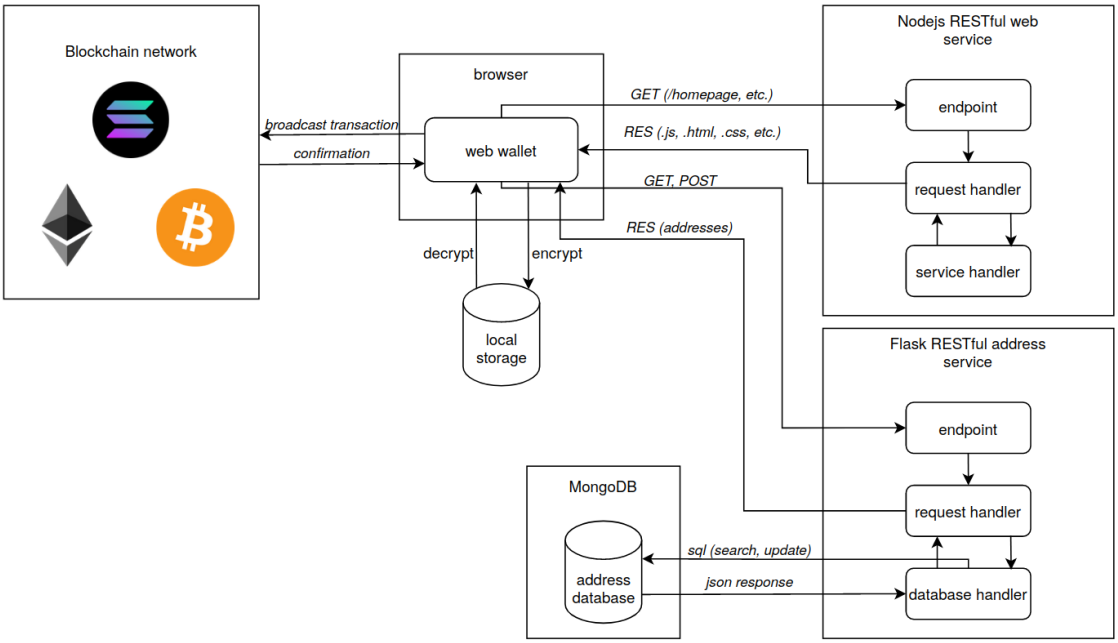

Figure 5.4: Overview of entire system

The user can access our web wallet through their web browser from any platform. Features and detail of every page will be presented in Section 5.2.3. After creating the wallet, the master secret seed will be saved in their local storage with the index paths (for convenience). The master seed should be encrypted with the user's password for safety since there are attacks aimed at browser sensitive information. We will discuss all potential attacks in Chapter 6.

Every time the user derives a new child, the wallet will access the local storage to get (decrypt) the secret seed then generate the expected keys. The exact process happens when the user needs to transfer assets. The secret seed will be accessed to get the private key for the digital signature generation.

5.2.2 Address service

5.2.2.1 Technologies

The address service provides API for public wallet management and searching (in transaction). We apply Flask framework and MongoDB for our module. The system contains the total of four API.

Flask

Flask [39] is a micro web framework written in python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide standard functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies, and several standard framework-related tools. We also use Flask since python is more comfortable dealing with JSON content-type requests.

MongoDB

MongoDB [40] is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas, so it is suitable for our database design.

5.2.2.2 API Provider

For reference only, the base URL of the system is http://localhost:8080.

Get address

Request for the list of receiver address (who also use our HD wallet) on a specific blockchain. The response contains information of purpose of those wallet and the addresses.

Request:

POST http://localhost:8080/getaddress

Field	Description	Format
address	The SHA512 hash of the master address of receiver wallet.	Text
chain	The ID of the blockchain (see in BIP44 [25])	text

Example request:

```
curl - - request POST
- - url http://localhost:8080/getaddress
- - header 'Content-Type: application/json'
- - data '{
    "address": "test",
    "chain": "501"
    }'
```

Example response:

```
{
"list_address": {
   "default": "childaddress",
   "business": "childaddress1"
   },
"message": "Succeed",
"status": "Succeed",
"statusCode": 200
}
```

Create default Wallet

Request to publish the address to the server. The web will automatically call this service when a wallet is created (the one default child's index is 2021). The response contains confirmation of the request.

Request:

```
POST http://localhost:8080/createdefault
```

Field	Description	Format
_id	The value of the SHA512(master address) of the HD wallet. This is also the _id field of MongoDB document schema.	Text
address	The dictionary of default generated addresses	Text

Example request:

```
curl - - request POST
- - url http://localhost:8080/createdefault
- - header 'Content-Type: application/json'
- - data '{
    "_id": "tess1231",
        "address": {
            "1": {
                "default": "wallet1" {
                },
            "60": {
                "default": "wallet2"
                },
            "501": {
                "default": "wallet3"
                }
        }
    }'
```

Example response:

```
{
"message": "Succeed",
"status": "Succeed",
"statusCode": 200
}
```

Push child address

Request to append a child address to existed address list on the server. The response contains confirmation of the request.

Request:

```
POST http://localhost:8080/pushaddress
```

Field	Description	Format
_id	The value of the SHA512(master address) of the HD wallet. This is also the _id field of MongoDB document schema.	Text
chain	The ID of the blockchain	Text
purpose	The purpose of this child wallet	Text
child_address	The address of the child wallet	Text

Example request:

```
curl - - request POST
- - url http://localhost:8080/pushaddress
- - header 'Content-Type: application/json'
- - data '{
        "address": "test",
            "chain": "60",
            "purpose": "business1",
            "child_address": " 0x88d696674a104489f..."
        }
```

Example response:

```
{
"message": "Succeed",
"status": "Succeed",
"statusCode": 200
}
```

Pull child address

Request to delete a child address that existed on the server. The response contains confirmation of the request.

Request:

```
POST http://localhost:8080/deleteaddress
```

Field	Description	Format
_id	The value of the SHA512(master address) of the HD wallet. This is also the _id field of MongoDB document schema.	Text
chain	The ID of the blockchain	Text
purpose	The purpose of this child wallet	Text
child_address	The address of the child wallet	Text

Example request:

```
curl - - request POST
- - url http://localhost:8080/deleteaddress
- - header 'Content-Type: application/json'
- - data '{
        "address": "test",
            "chain": "60",
            "purpose": "business1",
            "child_address": " 0x88d696674a104489f..."
        }
```

Example response:

```
{
"message": "Succeed",
"status": "Succeed",
"statusCode": 200
}
```

There are potential attacks on this kind of server since we haven't provided any authentication at all. We discuss these attacks in the evaluation Section (see Chapter 4.2).

5.2.3 Web Wallet Service

React JS

React was invented by a software engineer at Facebook named Jordan Walke, was first introduced in 2011. Walke finalized the prototype and created React in 2012, soon combined it into Facebook in the same year. In 2013 React became open-source and later became available in Ruby on Rails and Python Applications. This is followed by the release of React Native, an extension of React for mobile development on Android and iOS, in 2015. Since then, React consistently pushed out many releases throughout the years, enhancing and submitting new features for users. React is a versatile tool that can be operated on both desktop and mobile platforms with many different features. One of them is the ability to produce interactively rather than simple static pages that can update and rerender data after each input from the user, thus permitting for a seamless UI without refreshing the whole page every time a component is changed. The reason for this is that React's structure is based on components that allow for the design of complex UIs due to the component logic being written in JavaScript.

There are another framework for JavaScript such as Vue.js, Angular, etc, but we choose React as it the most popular one. Compared to other framework, React is just a library, not a pre-built framework, which is easier to configure and develop. Another difference is that, React uses multiple components together to build up a website rather than a website template and update its inner elements. Each component is an HTML element which has its own properties and state. For this reason, React allows re-rendering specific components of the webpage in comparison to Vue.js or Angular, which re-renders a whole page. The reason why it can work like that is because React does not update the DOM directly, but through a virtual DOM. The virtual DOM is a JavaScript object which stores all the information to create a DOM. Every time a component updates its state, the virtual DOM compares the change and only render the changes onto the DOM. Overall, React helps us to maintain and update code easier through the component structure (similar to Object-oriented programming style), interactive website, and efficient DOM manipulation with virtual DOM.

Node.js

Node.js is required as it provides a JavaScript environment and libraries for developing React. Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

User Interface

We designed the UI with Figma, a free web-based tools. Based on the use cases, the website would have these functionalities: create or import a wallet, create a child wallet, delete wallet (delete the mnemonic from local storage and logout), search for a destination address, modify current address on the server, and finally make a transaction. At the beginning, the website needs your wallet to perform all the other functions so the homepage should require you to create a new wallet or import your own wallet using your mnemonic code.

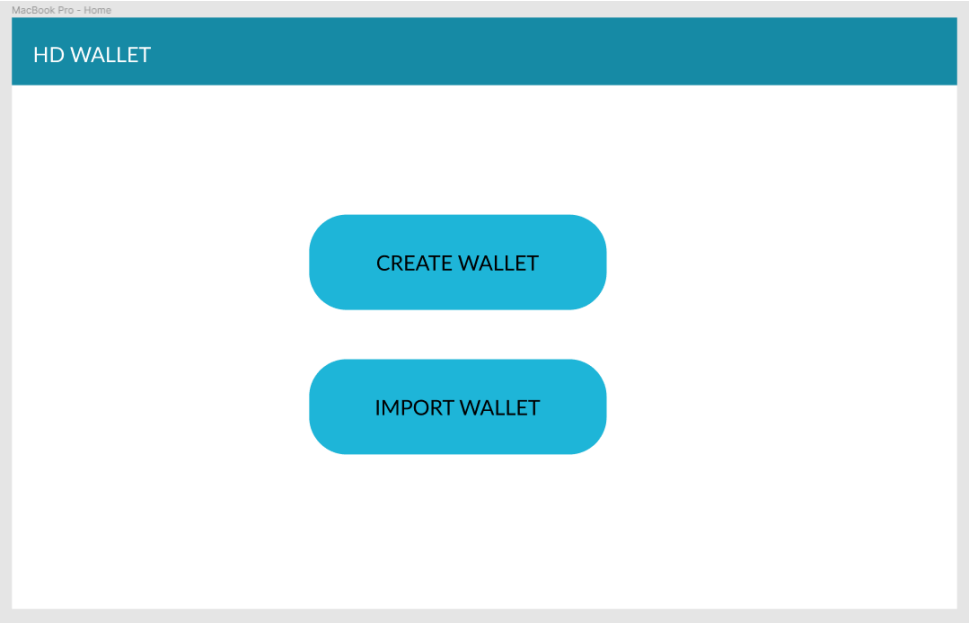

Figure 5.5: Homepage design

Depend on your choice, the website then allows you to choose a mnemonic code and backup if you want to create a new one, or simply paste your mnemonic to restore your own wallet.

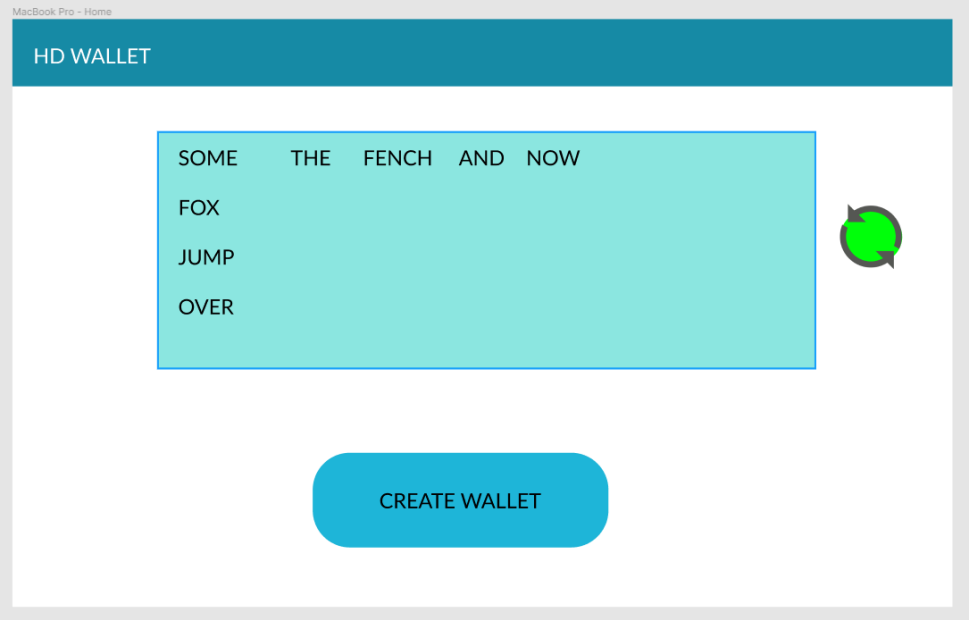

Figure 5.6: Create mnemonic

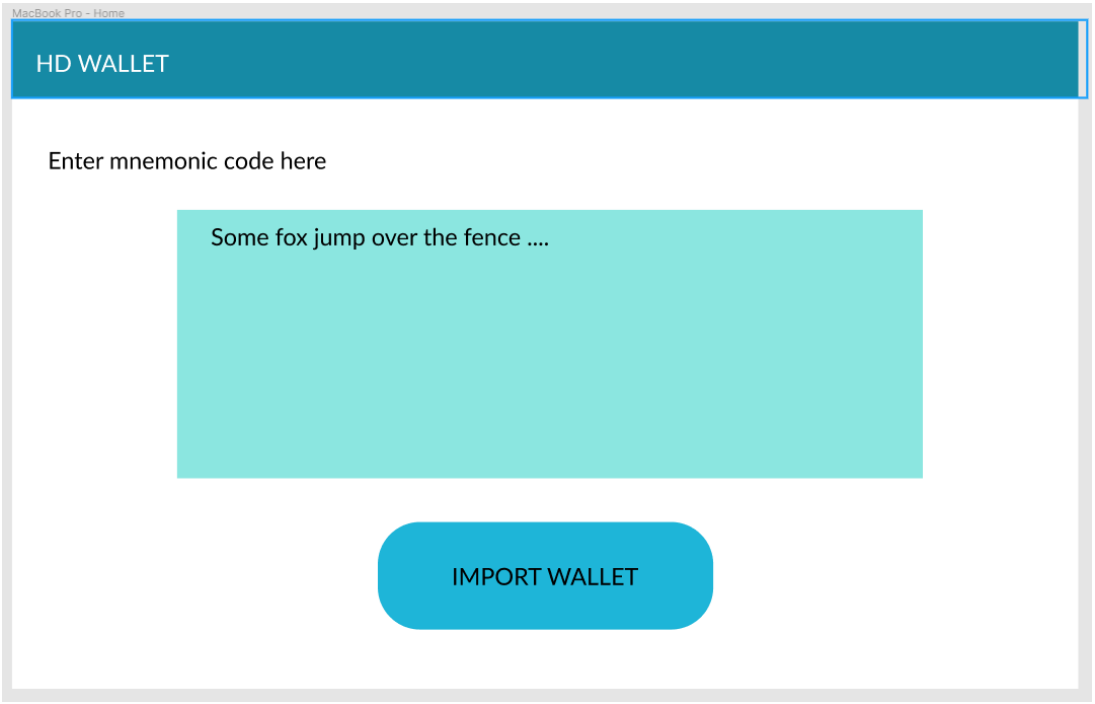

Figure 5.7: Import mnemonic

After that, you are on the main page. The navigation bar will provide you with the option to delete wallet, and change the current blockchain network. The main content is the list of each wallet and all the standard functionalities of a wallet, which looks like a dropdown list.

To implement the UI in React, we need to break it down as smaller components. From the home page, you can see there are two components, the button component and the navigation bar. In the next page, it comes the text area component for you to enter or copy & backup the mnemonic code. The refresh is the button component with an icon, so we don't need to create a new component. In the main page, you can notice that the navigation bar now has some functionalities but we can expand the navigation bar from the home page. The same goes with the add child wallet button, which is the plus icon button at the bottom of the list. The list component is the main component that you will interact with. However, we only reused button component, navigation bar. That means, in total, the whole application has three main components: the navigation bar, the button, and the list. Each component has a function part for even catching, which will be passed in from the API library. We will illustrate how the functions are used in the testing part.

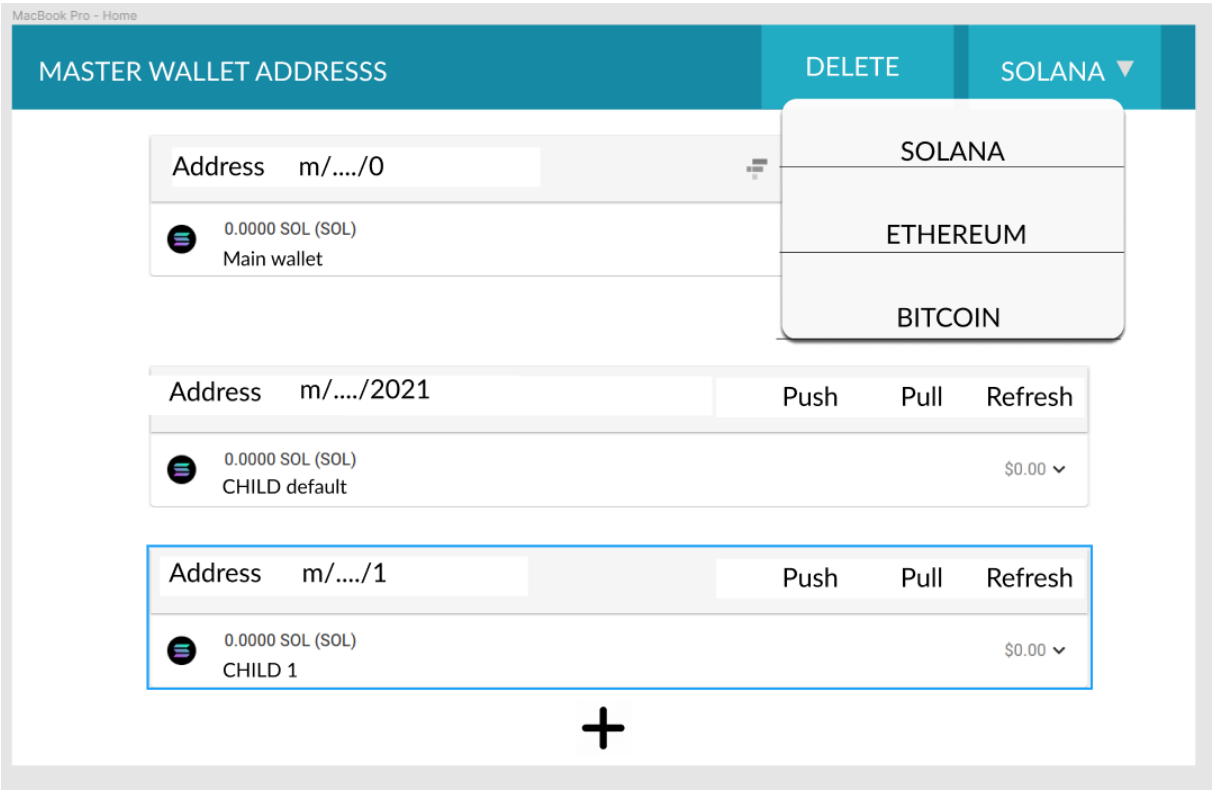

Figure 5.8: Main page

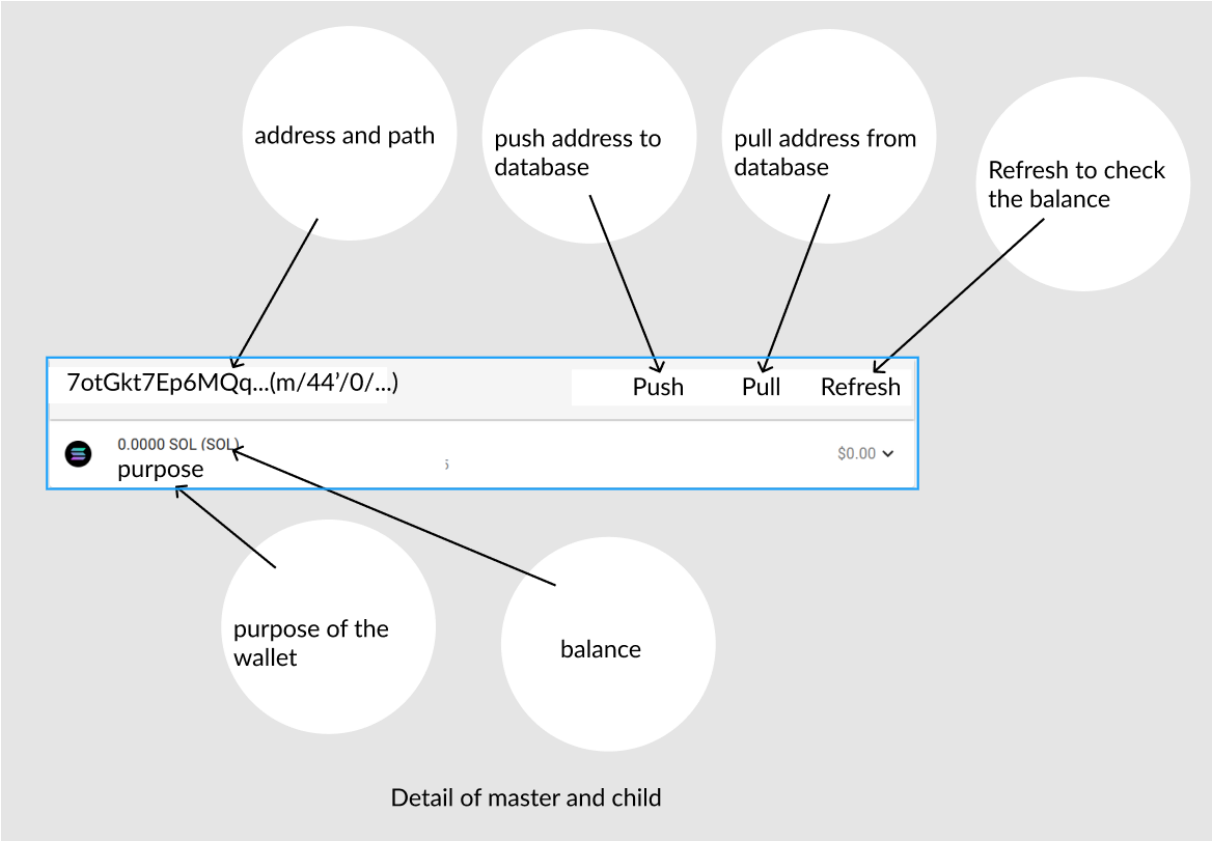

Figure 5.9: List functionality

6

Testing

This chapter presents the results of unit testing and our discussion on security perspectives. Section 6.1 describes our works on the open-source library, and Section 6.2 evaluates the non-custodial HD wallet.

Contents

6.1 The Library **102**
6.1.1 Testcases 102
6.1.2 Security Discussion 104
6.2 The Hierarchical Deterministic Web Wallet **106**
6.2.1 Security Discussion 106

6.1 The Library

6.1.1 Testcases

We run a total of 10 test cases to make sure every API functions normally according to its definition (presenting tests with Solana and ed25519 keys).

Test data

{

"mnemonic" : 'honey opera much plate gloom accuse honey pudding chase beach flight pencil velvet apart series pepper antenna hill amazing season foot goddess harsh theory',

"master_seed" : 'd51741549ced8fdce60e30ff8c0b885d6f156dfc44

b7de53dc66b5f8c3a3d48e06b1fd7c8c8c73748e2e78db3f84a4ba

e855c9304e14f593674870a1db5cdb6d',

"m44'501'0'0'10"" : 'HSPjBD1z

JLgj29hy8i7DD4HmaDPDh3bc1N2J4Dx9qsRn'

}

Test table

ID	Description	Expected Result	Actual Result	Pass/Fail
SOL-01	Generate a valid mnemonic code contains 12 or 24 words.	Generate successfully	As expected	Pass
SOL-02	Generate a valid master secret seed.	Generate successfully	As expected	Pass
SOL-03	Generate a valid master key pair.	Generate successfully	As expected	Pass
SOL-04	Generate a valid path.	Generate successfully	As expected	Pass
SOL-05	Generate a valid child key pair.	Generate successfully	As expected	Pass
SOL-06	Generate a valid public address.	Generate successfully	As expected	Pass
SOL-07	Generate a child generated from given mnemonic.	Generate successfully	As expected	Pass

Test with mocha (Javascript testing framework)

Figure 6.1 presents the results of mocha framework testing for our library.

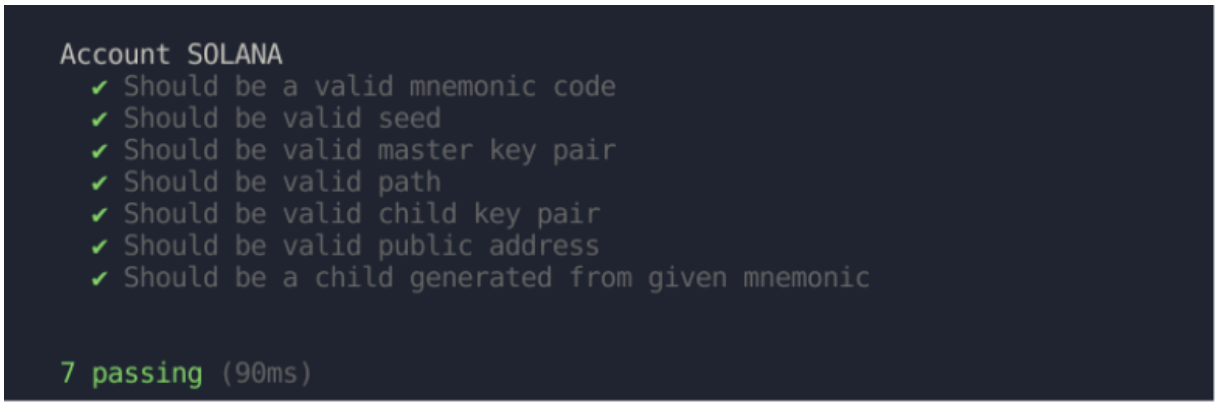

Figure 6.1: Testcase results with mocha

We tested the HD key tree generation with mocha. The transaction between wallets is a bit hard to confirm the result, so we use Jupyter Notebook instead.

Test with mocha (Javascript testing framework)

We import Nodejs kernel to notebook environment and start a session. The grey part is a cell to input code. Below every cell is result of code compilation.

- Test airdrop (see Figure 6.2).
- Checking the address balance (see Figure 6.3).
- Sending 0.05 SOL to another address (see Figure 6.4).
- Checking the return transaction hash on Solana Explorer (see Figure 6.5 and Figure 6.6).Solana Explorer is a website to look up transactions and accounts on the various Solana clusters. In Figure. We can see we transferred 0.05 SOL as the instruction in the notebook Figures.

```
[9]: var address= hdcore.account.getAddress(a.pub, '501')
     address

[9]: '5WWsVcPGznkyPxPbmXQuoMybWKgScu78JEmEeiVVX2mX'

[10]: hdcore.account.getTransaction('501').airdrop_one(a.pub)

[10]: true
```

Figure 6.2: Test airdrop with Solana

```
[11]: hdcore.account.getTransaction('501').get_balance(a.pub)

[11]: 1
```

Figure 6.3: Test balance with Solana

```
hdcore.account.getTransaction('501').send(a.pub, a.prv, 'FydV2tGRR23aiyENhnzZmtux7PPdm4zZF6Xrti8SkjKD', 0.05)

'24GhdWk7XHFQoPYR5xdZekf1rww8WWVTu8FzEY9kUakY4SmK9thKmXT24i2aDWzXPgC4uBjDS5zhGTQ1yaAsqXs1'
```

Figure 6.4: Test transaction with Solana

6.1.2 Security Discussion

We acknowledge that cryptography is a tool to achieve security. Still, if we get anything wrong in the cryptography implementation, that will not deliver the protection it was meant to offer, and it will become ineffective. We focused on researching the security flaws as the warning in the proposed paper to avoid implementing bad crypto.

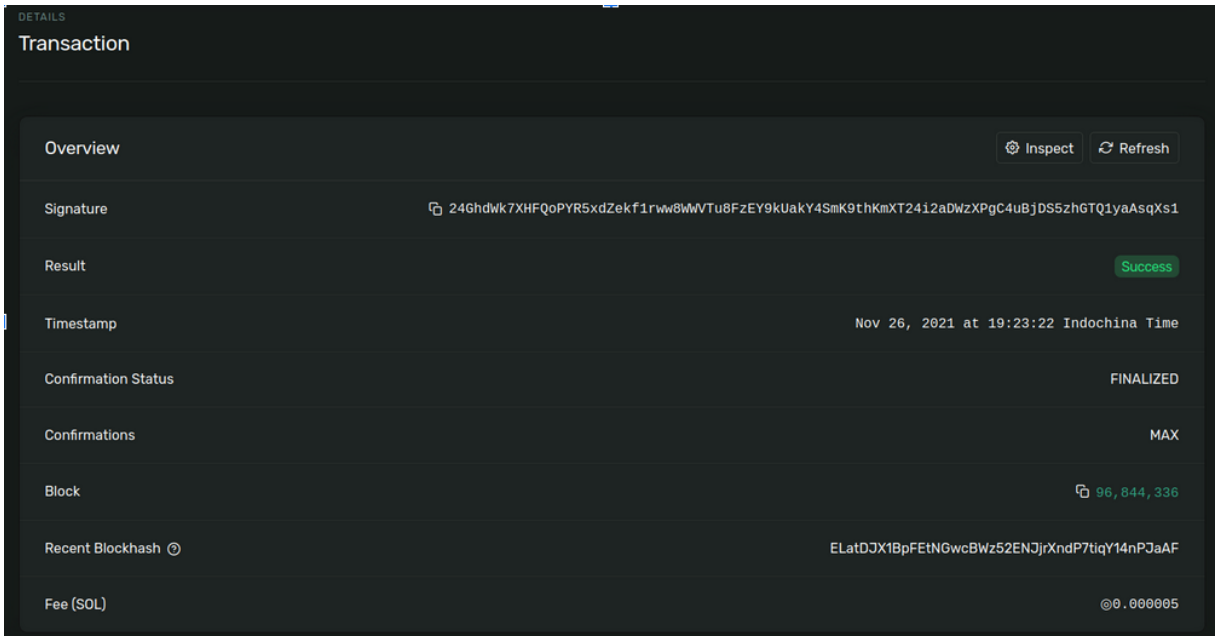

Figure 6.5: Transaction detail on Solana Explorer

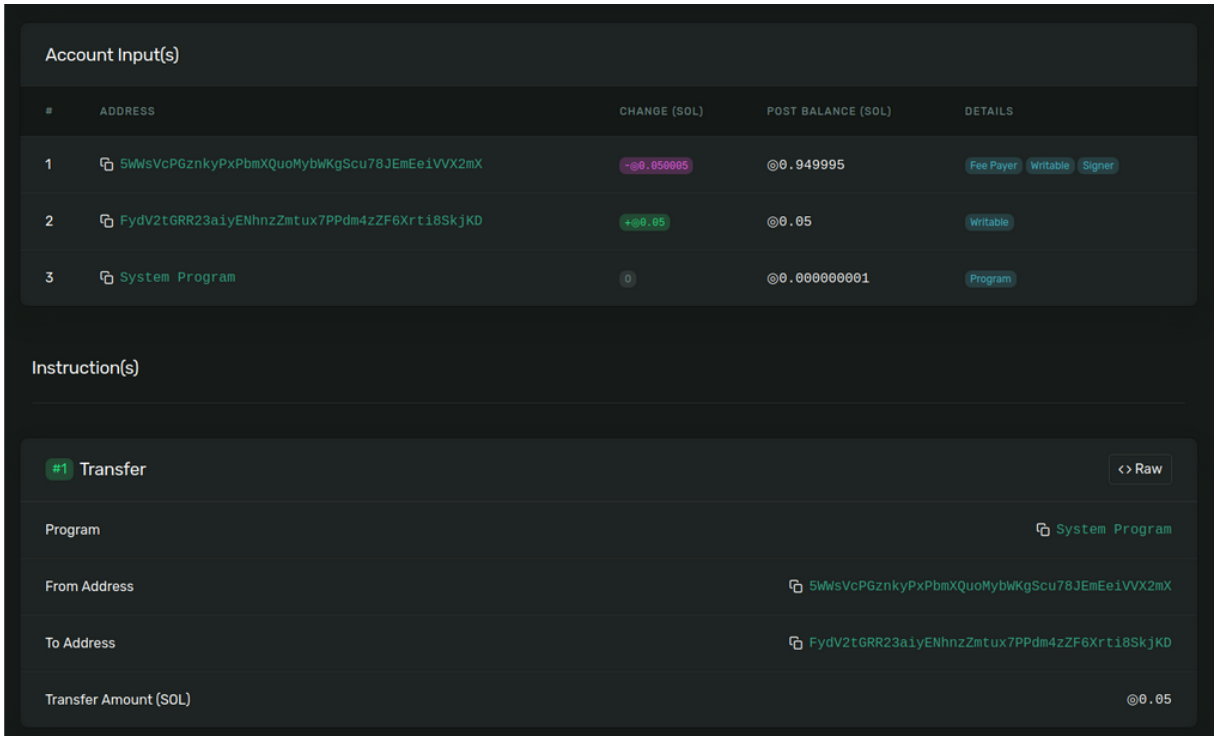

Figure 6.6: Coins transfered details

Digital signature generation libraries like tweetnacl and secp256k1 (for Solana and Ethereum keys) are completely free of timing side-channels since they provide constant time and constant memory access. We look into their publication and implementation to observe their approach compared to the original paper.

We use the hardened key derivation schema of the BIP32 and SLIP10 to avoid the key recovery attack on public key derivation because our wallet is a hot wallet type. We examine the potential vulnerabilities of the BIP32-ED25519 paper for ed25519 curves and

take the researcher's recommendation seriously. To compensate for the inconvenience of hardened key schema, we created a system alongside our library to provide a comfortable assets exchange.

We also make sure the requests and connection of the wallet to the blockchain will not leak any important information about the created wallet. Overall, our library applied a provable cryptography library and followed the safest schema in the industrial community, and improvised the efficiency by current technology.

6.2 The Hierarchical Deterministic Web Wallet

6.2.1 Security Discussion

We think that no system is perfectly safe from security threats. At least, we can prevent common attacks from happening. Observing mentioned OWASP's securities risk, we present as follows:

- **Broken Access Control**.

  The Address service is highly vulnerable to this attack since we design it with no authentication at all. Access control enforces policy such that users cannot act outside of their intended permissions. No authentication lead to an attacker can poison other address databases. For example, they can push their addresses into each other documents in MongoDB and impersonate their child addresses. However, we consider this vulnerability is deemed to be easy to solve. In this thesis, we created a demo product. In practice, we could avoid this attack for our service with any server we deployed on with a regular authentication token for each user.

- **Cryptographic Failures**.

  Our web wallet security depends on heavy cryptography. We created a library that takes parts as a core of our fundamental cryptography to prevent this vulnerability. The provable security of our library is discussed in Section 6.1.

- **Injection**.

  We prevent injection attacks by carefully checking the user's input and rejecting them if they belong to scripts group.

- **Insecure Design**.

  Secure design is a culture and methodology that constantly evaluates threats and ensures that code is robustly designed and tested to prevent known attack methods. We designed a system that leaks no sensitive information of the user. We also researched well-concerned attack vectors on our system and the library and prevented it in our implementation.

- **Vulnerable and Outdated Components**.

  Library and frameworks we used are up-to-date and well researched. We removed all unnecessary features, components, and samples. There are no redundant or risks contained in the library that we applied.

In conclusion, we covered famous attacks on standard web applications and ran test cases for the software to behave appropriately. There are some potential attacks on the client-side that we (the service provider) cant handle. Where users access some malicious software, log in their mnemonic seeds to the wrong website, etc.

7

Conclusion and discussion

In the final chapter, we want to summarize the results for the the thesis, as well drawbacks. Finally, we want to present space of improvements for the thesis.

Contents

7.1 Results **110**
7.2 Limitations **110**
7.3 Future works **111**

7.1 Results

Finishing the thesis, we have more opportunities to learn and earn more knowledge in the field of blockchain technology and its decentralized network application, as well as the security field, especially in the cryptography mechanism. We also study how to create and secure a web application. We have succeeded in building the open-source library, which provides functions for developers and us to implement in multiple blockchains, alongside the non-custodial and hierarchical deterministic web application, which supports every ability of BIP32 while maintaining its avoidance of key leakage.

The "hdcore" library has already met the requirements of key generations, child key derivation, signature generation, transaction creation and broadcasting, with other validation. The background and related work part of this thesis explored a deep understanding of derivation schema and cryptography with the wallet security threat, therefore laying a good foundation for creating the library.

Also, the web wallet we created passed the criteria of industrial standards. Users can have absolute control over their assets without trusted third parties. They can create multiple amounts of child wallets with given purposes. The wallet also supports new features like the limited expose of addresses to other wallet users.

We learned a lot while working on this thesis over the course of 9 months. We also learned how to self-learn, collaborate, contribute to the community and encourage one another throughout adversity. What we have accomplished as a result of this thesis is far superior to anything we could have anticipated. The objective of the thesis was achieved and able to determine the efficiency of the security technologies that are applied by our wallet. At the end of the study, the thesis can be used as a reference to contributing to later research on the constantly-developing HD wallet.

7.2 Limitations

For the library, it only supports three out of a thousand existing blockchains. This will require a lot of time and contribution from the developers and us. The library is still heavy on dependencies where lots of them can be rewritten in more efficient ways. Due to time limitations and the massive amount of background knowledge, in the implementation process, we couldn't research all of exists mechanism. So we take on recommendations from NIST and the community to develop our project without creating it independently.

The web wallet we created is still simple and doesn't reach the level of industrial production with hundreds of people behind it. Our works in this thesis didn't provide a way to authorize and limit the user's access. The microservices architect we use can give availability but is very hard to maintain and update. The client-server model is also lost compared to the decentralization of the blockchain network. There is still plenty of room for development in our wallets.

7.3 Future works

The library implementation can be expanded in multiple ways. We can separate the blockchain using the same curve for the user account and provide a general key derivation for specific kinds. The transaction base and account base blockchain can be determined as well. This way, the library can be less heavy on the dependencies. We can optimize the speed and new configurations for scalar multiplications with the support of different proposed and implementations.

The web wallet can be put more effort into and make it more user-friendly. The centralized client-server can be replaced with the InterPlanetary File System (IFPS), which is a protocol and peer-to-peer network for storing and sharing data in a distributed file system [41]. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices. It also provides a decentralized database so users can participate in maintaining the software. We can also investigate on the new authenticity mechanism for the decentralized network of our. Since password-based login is an insecure approach to online interactions and that multi-factor schemes add friction that reduce user adoption and productivity.

Bibliography

[1] Daniel J. Bernstein, "Curve25519: New diffie-hellman speed records," in *Public Key Cryptography - PKC 2006*, Springer Berlin Heidelberg, 2006, pp. 207–228. DOI: 10.1007/11745853_14. [Online]. Available: https://doi.org/10.1007/11745853_14.

[2] Daniel J. Bernstein and Niels Duif and Tanja Lange and Peter Schwabe and Bo-Yin Yang, "High-speed high-security signatures," in *Cryptographic Hardware and Embedded Systems – CHES 2011*, Springer Berlin Heidelberg, 2011, pp. 124–142. DOI: 10.1007/978-3-642-23951-9_9. [Online]. Available: https://doi.org/10.1007/978-3-642-23951-9_9.

[3] Jörn-Marc Schmidt and Marcel Medwed, "A fault attack on ECDSA," in *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, IEEE, Sep. 2009. DOI: 10.1109/fdtc.2009.38. [Online]. Available: https://doi.org/10.1109/fdtc.2009.38.

[4] S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (EdDSA)," Tech. Rep., Jan. 2017. DOI: 10.17487/rfc8032. [Online]. Available: https://doi.org/10.17487/rfc8032.

[5] Pieter Wuille, "Hierarchical deterministic wallets," 2012. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki.

[6] Daniel R. L. Brown, "Recommended elliptic curve domain parameters," version 2.0, Oct. 2010. [Online]. Available: https://www.secg.org/sec2-v2.pdf.

[7] Dmitry Khovratovich and Jason Law, "BIP32-ed25519: Hierarchical deterministic keys over a non-linear keyspace," in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, Apr. 2017. DOI: 10.1109/eurospw.2017.47. [Online]. Available: https://doi.org/10.1109/eurospw.2017.47.

[8] Elaine Barker and Allen Roginsky, "Transitioning the use of cryptographic algorithms and key lengths," Tech. Rep., Mar. 2019. DOI: 10.6028/nist.sp.800-131ar2. [Online]. Available: https://doi.org/10.6028/nist.sp.800-131ar2.

[9] Elaine Barker, "Recommendation for key management:" Tech. Rep., May 2020. DOI: 10.6028/nist.sp.800-57pt1r5. [Online]. Available: https://doi.org/10.6028/nist.sp.800-57pt1r5.

[10] Poulami Das and Andreas Erwig and Sebastian Faust and Julian Loss and Siavash Riahi, "The exact security of BIP32 wallets," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds., ACM, 2021, pp. 1020–1042. DOI: 10.1145/3460120.3484807. [Online]. Available: https://doi.org/10.1145/3460120.3484807.

[11] Marek Palatinus, Pavol Rusnak, Aaron Voisine and Sean Bowe, "Mnemonic code for generating deterministic keys," 2013. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki.

[12] M. Nystrom, "Identifiers and test vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512," Tech. Rep., Dec. 2005. DOI: 10.17487/rfc4231. [Online]. Available: https://doi.org/10.17487/rfc4231.

[13] Gus Gutoski and Douglas Stebila, "Hierarchical deterministic bitcoin wallets that tolerate key leakage," in *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, R. Böhme and T. Okamoto, Eds., ser. Lecture Notes in Computer Science, vol. 8975, Springer, 2015, pp. 497–504. DOI: 10.1007/978-3-662-47854-7_31. [Online]. Available: https://doi.org/10.1007/978-3-662-47854-7%5C_31.

[14] Vitalik Buterin, *Deterministic wallets, their advantages and their understated flaws*. [Online]. Available: https://bitcoinmagazine.com/technical/deterministic-wallets-advantages-flaw-1385450276.

[15] Daniel J. Bernstein and Tanja Lange, "Montgomery curves and the montgomery ladder," *IACR Cryptol. ePrint Arch.*, p. 293, 2017. [Online]. Available: http://eprint.iacr.org/2017/293.

[16] Robert J. Zuccherato, "Methods for avoiding the "small-subgroup" attacks on the diffie-hellman key agreement method for S/MIME," *RFC*, vol. 2785, pp. 1–11, 2000. DOI: 10.17487/RFC2785. [Online]. Available: https://doi.org/10.17487/RFC2785.

[17] Riccardo Spagni, *Disclosure of a major bug in cryptonote based currencies*. [Online]. Available: https://web.getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html.

[18] Wikipedia contributors, *Diffie-hellman key exchange*. [Online]. Available: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange.

[19] Gregory Maxwell, *Ed25519 "clamping" and its effect on hierarchical key derivation*. [Online]. Available: https://moderncrypto.org/mail-archive/curves/2017/000860.html.

[20] D. J. Bernstein, *Does the curve25519 montgomery ladder always work?* [Online]. Available: https://mailarchive.ietf.org/arch/msg/cfrg/pt2bt3fGQbNF8qdEcorp-rJSJrc/.

[21] Billy Bob Brumley and Nicola Tuveri, "Remote timing attacks are still practical," *IACR Cryptol. ePrint Arch.*, p. 232, 2011. [Online]. Available: http://eprint.iacr.org/2011/232.

[22] Simon Josefsson and Ilari Liusvaara, "Edwards-curve digital signature algorithm (eddsa)," *RFC*, vol. 8032, pp. 1–60, 2017. DOI: 10.17487/RFC8032. [Online]. Available: https://doi.org/10.17487/RFC8032.

[23] Jeff Burdges, *Key recovery attack on bip32-ed25519*. [Online]. Available: https://forum.w3f.community/t/key-recovery-attack-on-bip32-ed25519/44.

[24] David Goulet, George Kadianakis, Nick Mathewson, "Next-generation hidden services in tor," Nov. 2013. [Online]. Available: https://gitweb.torproject.org/torspec.git/tree/proposals/224-rend-spec-ng.txt#n2135.

[25] Pavol Rusnak, Marek Palatinus, *Registered coin types for bip-0044*. [Online]. Available: https://github.com/satoshilabs/slips/blob/master/slip-0044.md.

[26] christsim (github), *Key derivation for ed25519*. [Online]. Available: https://www.npmjs.com/package/hd-key-ed25519.

[27] alepop (github), *Key derivation for ed25519 (ed25519 hd key)*. [Online]. Available: https://github.com/alepop/ed25519-hd-key.

[28] Vincent Bernardoff, *Ocaml implements for ed25519*. [Online]. Available: https://github.com/vbmithr/ocaml-bip32-ed25519.

[29] Shude Li, *Golang implements for bip32-ed25519*. [Online]. Available: https://github.com/islishude/bip32.

[30] LedgerHQ, Trusted hardware for blockchain applications, *Hded25519*. [Online]. Available: https://github.com/LedgerHQ/orakolo/blob/0b2d5e669ec61df9a824df9fa1a363060src/python/orakolo/HDEd25519.py.

[31] Wikipedia contributors, *Shor's algorithm*. [Online]. Available: https://en.wikipedia.org/wiki/Shor%27s_algorithm.

[32] Project Serum, *Spl token wallet*. [Online]. Available: https://github.com/project-serum/spl-token-wallet.

[33] Jeff Burdges, *Issue on ed25519 dalek*. [Online]. Available: https://github.com/dalek-cryptography/ed25519-dalek/issues/137.

[34] Oasis Protocol Foundation, *Adr 0008: Standard account key generation*. [Online]. Available: https://github.com/oasisprotocol/oasis-core/blob/master/docs/adr/0008-standard-account-key-generation.md#difficulties-in-adapting-bip-0032-to-edwards25519-curve.

[35] Wikipedia contributors, *Man-in-the-middle attack*. [Online]. Available: https://en.wikipedia.org/wiki/Man-in-the-middle_attack.

[36] CWE Team, *Incorrect type conversion or cast*. [Online]. Available: https://cwe.mitre.org/data/definitions/704.html.

[37] Daniel J. Bernstein and Tanja Lange and Peter Schwabe, "The security impact of a new cryptographic library," in *Progress in Cryptology – LATINCRYPT 2012*, Springer Berlin Heidelberg, 2012, pp. 159–176. DOI: 10.1007/978-3-642-33481-8_9. [Online]. Available: https://doi.org/10.1007/978-3-642-33481-8_9.

[38] Tomi Jaga, *Thenewboston hd wallet*. [Online]. Available: https://github.com/tomijaga/tnb-hd-wallet.

[39] Armin Ronacher, *Flask*. 2010. [Online]. Available: https://github.com/pallets/flask.

[40] MongoDB Inc., *Mongodb*. [Online]. Available: https://www.mongodb.com/.

[41] Protocol Labs, *Ipfs powers the distributed web*. [Online]. Available: https://ipfs.io/.