Software Wallet for Spartan Gold

Tanmay Singal and Angela Yang

San José State University, San José, CA

May 2023

Abstract

Spartan Gold is a cryptocurrency developed as an academic exercise to emulate the proof-of-work protocol of Bitcoin. While it has achieved feature parity with real-life currencies, its lack of user-friendliness limits its mainstream deployment. This project focuses on developing a Hierarchical Deterministic (HD) wallet for Spartan Gold that is compliant with the widely used BIPs 32, 39, and 44. These BIPs provide standards for HD wallets, mnemonic phrases, and account hierarchies, respectively. By implementing these standards, the wallet improves user-friendliness, backup and recovery processes, privacy, and the ability to organize transactions into different accounts. The implementation of a BIP-compliant wallet for Spartan Gold would significantly improve the currency's prospects of being used as a real cryptocurrency, expanding its current utility as an academic exercise. This report provides a comprehensive guide to the development process of the wallet and valuable insights into creating a BIP-compliant cryptocurrency wallet.

Keywords— Bitcoin, Blockchain, Cryptocurrency, Software Wallet, Spartan Gold

1 Introduction

Blockchains and cryptocurrencies have been growing in popularity recently due to their strong decentralization and security features. As novel and exciting technologies are being built on top of cryptocurrencies, academic interest in learning about "Web 3" is at an all-time high. Among the numerous cryptocurrencies available today, Spartan Gold stands out as an academic exercise developed by Prof. T. Austin. Spartan Gold is similar to Bitcoin in features and provides a unique opportunity for students and researchers to understand the underlying technology of cryptocurrencies.

One of the challenges with cryptocurrencies is the complexity of securely managing numerous private keys while always running the risk of losing access to your funds in case the keys are lost or stolen. To address this challenge, we have developed a software wallet that interacts with Spartan Gold. The software wallet is BIPs 32 [2], 39 [3], and 44 [4] compliant, which ensures compatibility with other wallets and provides an

additional layer of security. Furthermore, our implementation is compatible with all other implementations of wallets for Spartan Gold, and it requires no change to the Spartan Gold network to integrate.

With the new software wallet, users only need to remember their mnemonic phrase to access their Spartan Gold. Mnemonic phrases are a series of random words generated by our wallet that represent the private keys derived from the master seed. These phrases eliminate the need for users to remember complex public and private key pairs, which can be a daunting task for many users. Mnemonics also reduce the number of attack vectors for hackers, as the users only need to secure a single string in order to keep their funds secure. By providing a user-friendly and secure way to access Spartan Gold, we hope to increase its adoption and make it more accessible to a wider audience.

The purpose of this project report is to describe the development process of building a software wallet for Spartan Gold. We will discuss the technical details of the project, including its architectural design, implementation, and its integration with the existing Spartan Gold network. Our goal is to make Spartan Gold more accessible to users by providing a user-friendly and secure wallet that is compatible with existing implementations and requires no changes to the Spartan Gold network. This report serves as a comprehensive guide to the development of this wallet and provides valuable insights into the process of creating a BIPs 32 [2], 39 [3], and 44 [4] compliant cryptocurrency wallet.

2 BIP Compliance

To ensure compatibility with other Spartan Gold wallets and provide an additional layer of security, we have implemented BIPs 32 [2], 39 [3], and 44 [4] compliance for our version of the wallet. These BIP standards are widely used in the cryptocurrency industry and provide a set of rules and guidelines for cryptocurrency wallet developers to follow. While we are building a wallet for Spartan Gold and not Bitcoin, by implementing these Bitcoin Improvement Proposal standards, our software wallet ensures compatibility with other Spartan Gold wallets and provides users with a higher level of security. In this report, we will explore each BIP standard in detail and discuss how we have incorporated them into our wallets.

2.1 BIP 32

BIP 32 [2] was proposed by P. Wuille in 2012 and introduced the concept of a Hierarchical Deterministic wallet (HD wallet). These wallets use a master seed to generate an unlimited number of private keys using a deterministic algorithm. Our project implements an HD wallet that is fully BIP 32 compliant and that makes the backup and recovery of the wallet as trivial as persisting the master seed. Apart from the ease of having the user only keep track of a single master seed, a BIP 32-compliant wallet like ours also offers better privacy by generating a new public key for each transaction performed. Since each transaction has a unique public key, linking transactions to each other and ergo back to a particular user becomes more difficult for external parties.

In order to create a [2]-compliant software wallet for Spartan Gold, we went on a

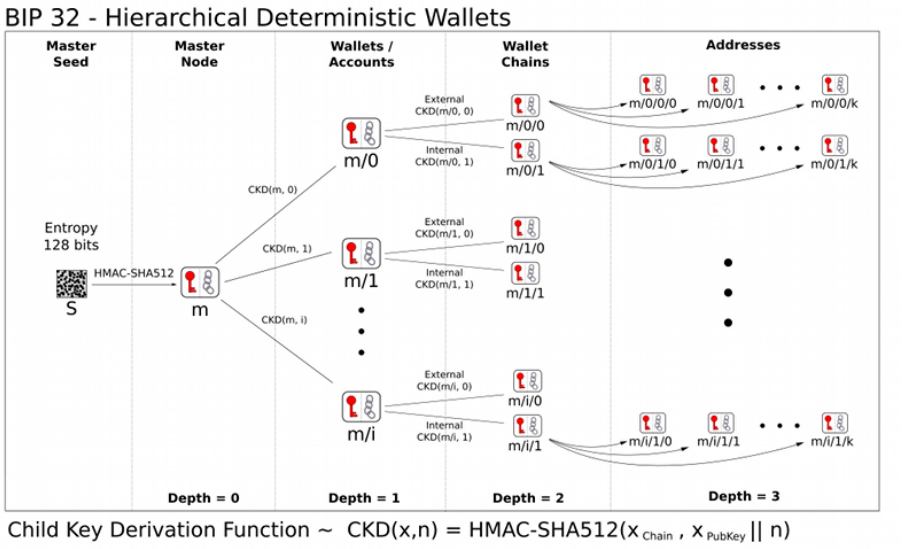
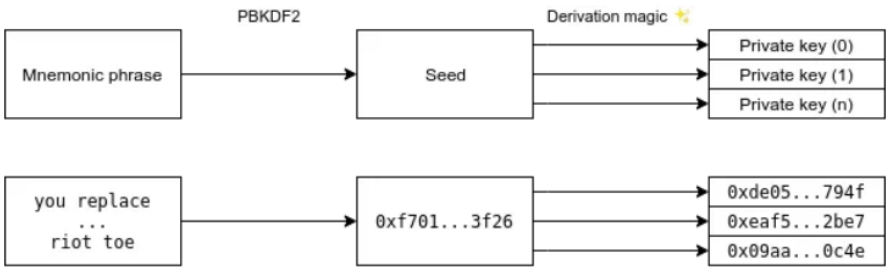


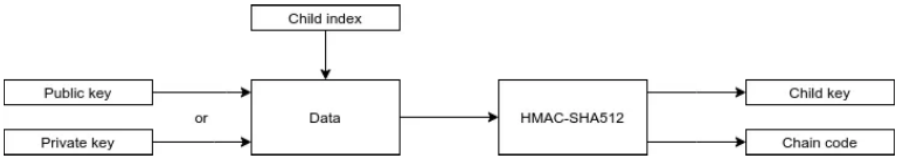
Figure 1: Various Depths of Key Derivation in an HD Wallet

journey to understand random seed generation, key derivation, and mnemonic phrases. We explored various concepts and processes involved in transforming a mnemonic phrase into cryptographic addresses. The mnemonic phrase is the human-readable version of a cryptographic seed represented as a set of words. These cryptographic seeds represented by mnemonic phrases are fundamental in generating the hierarchical deterministic wallets in this project.



The first step in creating an HD Wallet is generating a binary seed based on some source of entropy. This source of entropy can be a random coin toss, the movement inside a lava lamp, or in our case, a cryptographically secure random function-generating computer program. To generate the binary seed, we first created a mnemonic phrase based on this source of entropy and then converted it into a seed based on the PBKDF2 function (Password-Based Key Derivation Function 2). The master key generated from this seed will then go on to become the starting point of our key derivation process.

The master key derived from the binary seed is a cryptographic key obtained by applying a key derivation function to the binary seed. This involves hashing the seed multiple times using a function such as HMAC-SHA512. Once the master key has been derived, it can be used to generate the various public and private keys that will be used by our Spartan Gold wallet.



The process of deriving a child key.

In order to derive child keys from the generated master key, we made use of the Elliptic Curve Digital Signature Algorithm (ECDSA). It is a widely used algorithm in cryptocurrencies and allowed us to derive child keys and their corresponding public keys from the master key.

By following the key derivation process outlined in [2] and leveraging the *bip32* and *crypto* JavaScript libraries, we ensured that our Spartan Gold wallet implementation generated a hierarchical deterministic structure of private and public keys. Through our extensive research and dedication to adhering to standards outlined in [2], we were able to implement a software wallet for Spartan Gold that provides users with a secure and trusted solution for managing their Spartan Gold cryptocurrency.

2.2 BIP 39

BIP 39 [3] proposed in 2013 by M. Palatinus improves upon [2] by standardizing the list of words used by HD wallets. This standardized list leads to the generation of the same public key/private key pair when words from it are used. These words were specifically selected to be easy to read and type, while being distinct enough from each other alphabetically and phonetically to not be confused with each other.

The Spartan Gold wallet developed as a part of this project uses the same list of words standardized in BIP 39 and creates a standardized mnemonic phrase for users based on the randomly generated seed. Additionally, a passphrase (created by the user) is used to secure the wallet further, providing an extra layer of security.

[3] defines a list of 2048 words in its various word lists that can be used to generate a mnemonic to present to the user. The generated mnemonic is passed through a key stretching function to add an extra layer of security by preventing brute-force attacks.

After deriving the master seed, we generated the hierarchical tree of child keys at various paths in the BIP 32 key derivation tree. These keys were generated in a way that they would be compatible with all implementations of a wallet for Spartan Gold. This enhanced the usability and interoperability of the wallet and made it compatible

with any implementation of a wallet for this cryptocurrency that complied with BIP 32 and BIP 39.

2.3 BIP 44

Spartan Gold in its current implementation only maintains a global account for each user and does not allow users to have multiple accounts within the network. This limitation prevents users from being able to organize their transactions into different accounts, similar to how a traditional banking system allows for separate Savings and Checking accounts. This project implements the standardized account hierarchy scheme proposed in BIP 44 [4] such that the accounts created by all implementations of an application or wallet for Spartan Gold are able to interact with each other.

In our implementation of a [4]-compliant wallet for Spartan Gold, we faced several challenges when attempting to support multiple accounts. Initially, we extended the Client class in Spartan Gold to enable us to specify custom "from" addresses to perform transactions from different accounts. However, we found that our balances would get overridden as soon as another Spartan Gold miner came online. This was because of our blockchain and its genesis block's inability to compete with the miner who had found several proof-of-work solutions on top of their version of the blockchain.

To solve this problem, we had to switch the custom TcpClient class of our wallet to also be a Spartan Gold miner. This allowed us to enforce our account hierarchy on the blockchain by finding proof-of-work targets as well over our genesis block. The money mined in the process goes to the (m/0) root account of the wallet, making our users richer just for using our wallet.

By making these significant changes to the TCP communication interface of Spartan Gold in our wallet, we were able to support multiple accounts using the hierarchical deterministic scheme specified in [4] while still being compatible with the existing network. During wallet initialization, the root account's public and private key pair is generated from the mnemonic and user-specified passphrase, and then from the root account, we derive the child accounts. By following these standards, we ensure that all accounts have unique addresses and can be backed up and recovered easily using the mnemonic phrase.

3 Problems Encountered

During the course of our project, we encountered several challenges related to the derivation and conversion of our private keys. In our implementation, we utilized the bip32 functions, which rely on the ECDSA private key algorithm. However, these private keys are encoded in a hexadecimal format. SpartanGold requires the private key to be in a PEM format in order to sign a transaction.

A PEM format, or a Privacy-Enhanced Mail format is an encoding method for cryptographic objects It is easily recognized by the "——BEGIN" and "——END" delimiters.

One significant problem we faced was the lack of available resources explaining how to convert private keys from hexadecimal format to PEM format. After thorough

research, we identified a potential solution to this problem. The conversion process involves several steps. We need to convert the privateKey into DER format, which is a binary representation of the data structure and then we can transform the DER format into the desired PEM format. We learned that nodeJS does not support converting raw keys into DER format, so we utilized a third-party library called "eckey-utils" that would be able to do it for us.

The private key we get is still incorrect type of PEM format- sec1 type instead of pkcs8, so we used the crypto module to change the types.

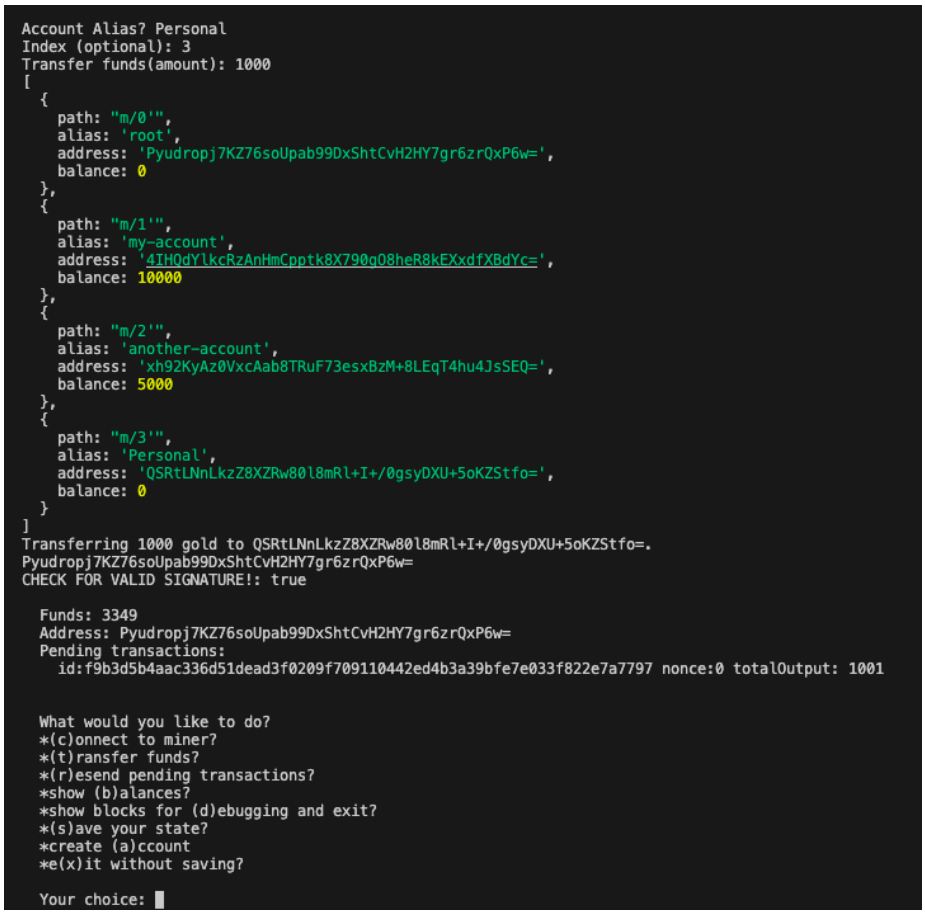
4 Screenshots of Output

Users must supply a wallet configuration file, which should include the name of the wallet and the connection settings for the wallet application. The connection should

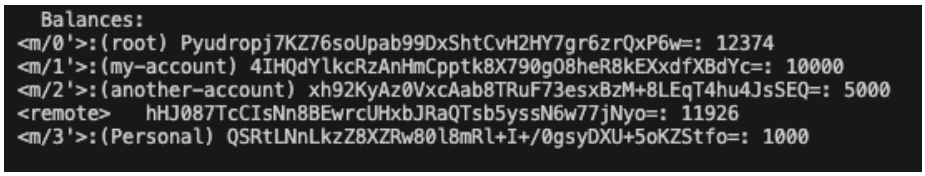
include the host name, which should be set to "localhost" for now and a port number for which the wallet application is listening for incoming connections. User can also additionally specify a mnemonic and a passphrase, if they wish to restore the wallet and access the user's funds.

Here, we ran with the configuration wallet-config.json. This file specifies a mnemonic and passphrase. In the screenshot above, we print out this mnemonic.

If the users does not specify a mnemonic, the wallet will create a BIP 39 mnemonic phrase for the user.

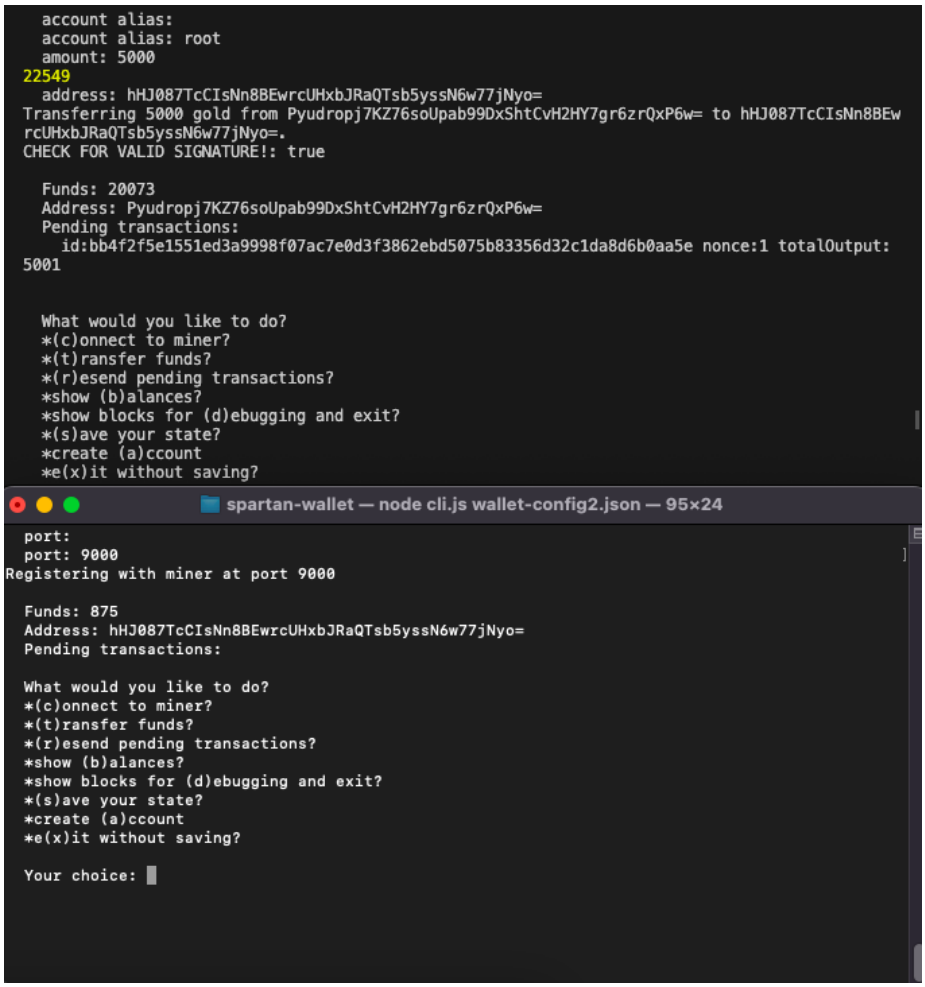


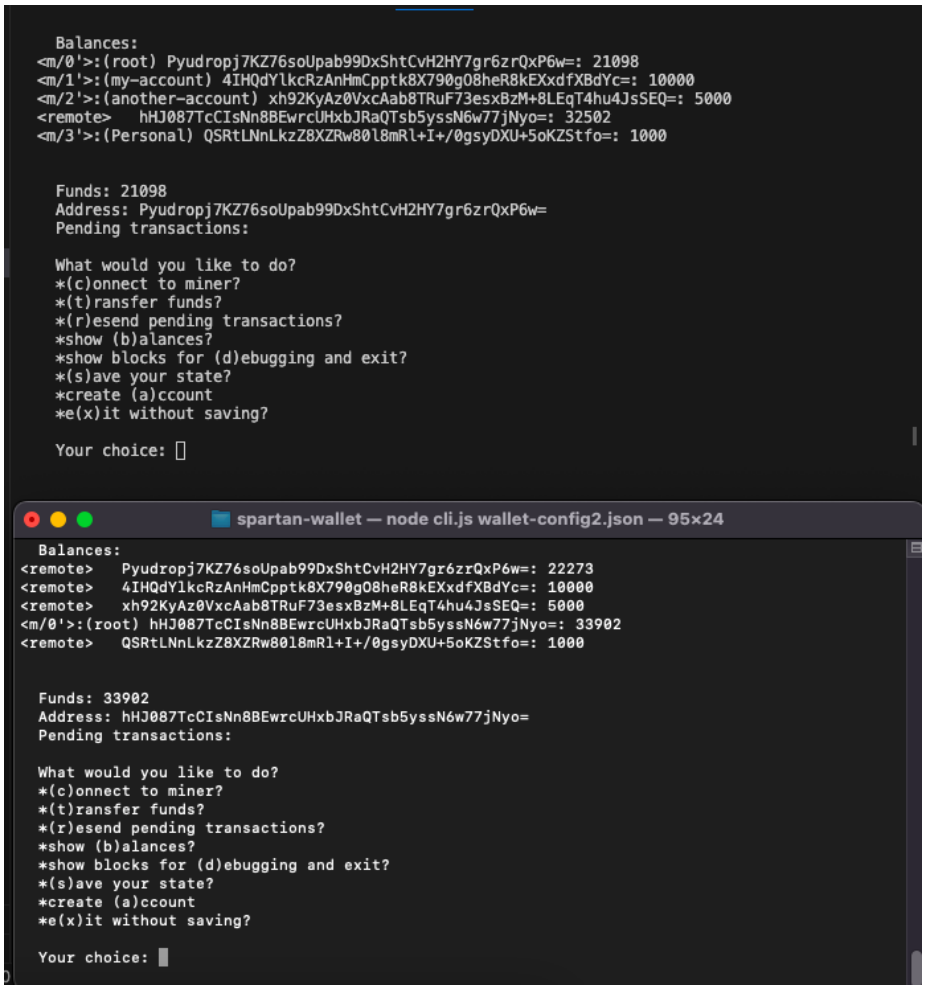
In the screenshot above, we created a new account for the wallet. The wallet comes with 3 accounts manually added already and we added a fourth account. It has the alias "Personal", the path derivation "m/3'" and a balance of 1000. We transferred the funds from the root account to the Personal account.



We called the Show Balances option, which will display all balances associated with the wallet application. In the function, we iterate over the balances of the last confirmed block and retrieves additional information about each account and logs the account's address, balance, and wallet context string to the console.

If the account's address is not inside the user's wallet, then we know they must be from a remote user. Instead of displaying the account's alias, the code now displays "remote" in its place. This adjustment helps differentiate between accounts owned by the user and those associated with remote users.


account alias:
account alias: root
amount: 5000
22549
address: hHJ087TcCIsNn8BEwrcUHxbJRaQTsb5yssN6w77jNyo=
Transferring 5000 gold from Pyudropj7KZ76soUpab99DxShtCvH2HY7gr6zrQxP6w= to hHJ087TcCIsNn8BEwrcUHxbJRaQTsb5yssN6w77jNyo=.
CHECK FOR VALID SIGNATURE!: true
Funds: 20073
Address: Pyudropj7KZ76soUpab99DxShtCvH2HY7gr6zrQxP6w=
Pending transactions:
id:bb4f2f5e1551ed3a9998f07ac7e0d3f3862ebd5075b83356d32c1da8d6b0aa5e nonce:1 totalOutput:
5001
What would you like to do?
*(c)onnect to miner?
*(t)ransfer funds?
*(r)esend pending transactions?
*show (b)alances?
*show blocks for (d)ebugging and exit?
*(s)ave your state?
*create (a)ccount
*e(x)it without saving?
spartan-wallet — node cli.js wallet-config2.json — 95×24
port:
port: 9000
Registering with miner at port 9000
Funds: 875
Address: hHJ087TcCIsNn8BEwrcUHxbJRaQTsb5yssN6w77jNyo=
Pending transactions:
What would you like to do?
*(c)onnect to miner?
*(t)ransfer funds?
*(r)esend pending transactions?
*show (b)alances?
*show blocks for (d)ebugging and exit?
*(s)ave your state?
*create (a)ccount
*e(x)it without saving?
Your choice:



In the screenshots above, we are transferring funds from the current wallet to another wallet. We need to specify the account that we want to take money from and also the address of the account that we want to transfer to.

References

[1] T. Austin, *A simplified blockchain-based cryptocurrency for experimentation.* Online: https://github.com/taustin/spartan-gold

[2] P. Wuille, *Hierarchical Deterministic Wallets* (2012). Online: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

[3] M. Palatinus, P. Rusnak, A. Voisine, and S. Bowe, *Mnemonic code for generating deterministic keys* (2013). Online: https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

[4] M. Palatinus and P. Rusnak, *Multi-Account Hierarchy for Deterministic Wallets* (2014). Online: https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki