HARDWARE WALLET FOR CRYPTOCURRENCIES

Final Year Project Report by

Muhammad Fahad Baig	**(174885)**
Haider Tariq	**(177742)**
Safi Majid	**(174203)**

In Partial Fulfilment

Of the Requirements for the degree

Bachelor's in electrical engineering (BEE)

Department of Electrical Engineering

School of Electrical Engineering and Computer Sciences

National University of Sciences and Technology

Islamabad, Pakistan

2020

CERTIFICATE

It is certified that the contents and form of thesis entitled **“Hardware Wallet for Cryptocurrencies”** submitted by **Muhammad Fahad Baig (174885), Haider Tariq (177742)** and **Safi Majid (174203)** have been found satisfactory for the requirement of the degree.

Advisor: ______________________

(Dr. Syed Taha Ali)

Co-Advisor: ______________________

(Dr. Osman Hassan)

DEDICATION

We dedicate this project to our beloved parents, who have been a constant source of motivation and inspiration and their never-ending financial, moral, emotional and spiritual support helped us to accomplish this project.

We also dedicate this to our faculty, especially our advisor, our friends, who gave us advice and encouragement throughout the course of the project.

Lastly, we dedicate this to Almighty Allah. His guidance, help, strength and power is the core reason we have been able to put our skills and life to do this project. Without Him, this wouldn’t have been possible.

ACKNOLWEDGEMENTS

We’re grateful for the support and guidance by our advisor, Dr. Syed Taha Ali and co-advisor, Dr. Osman Hassan, during the course of the project. Their deep insight and experience was of paramount importance to help us achieve our goals and targets set for the project.

We are also thankful to the support provided by Ain-ul-Zia, who was a master student assigned to the project. Her feedback and input to the project was valuable.

Lastly, we’re thankful to authors of the various opensource libraries used in the project, and the development team of Trezor proprietary wallets. Their contribution to the open Bitcoin ecosystem allowed us to gain deeper understanding of the project requirements and the core primitives.

TABLE OF CONTENTS

TABLE OF FIGURES

ABSTRACT

Bitcoin is a decentralized, P2P (peer to peer) electronic cash system that uses cryptographic primitives to allow two entities to make financial transactions between them without involving any third party, such as banks, as in traditional banking systems. The security of Bitcoin is derived from irreversibility of the cryptography features and functions it depends on. User's funds are protected by a unique private key which must be kept secure since losing the key means losing all your funds tied to that key. While the keys can be stored in a cloud wallet or paper-based wallet, the best option is to use hardware wallets since they offer the best balance between versatility and security.

In this work we discuss the fundamentals of Bitcoin technology and formulate a clear architecture and framework on which hardware wallets should be based on. Finally, we propose a semi hardware-software variant of wallet that runs on any ARMv7 CPU running Linux (GNU) kernel (such as Raspberry Pi Zero, 2/3/4), that prototypes wallet's firmware, connector interface tied to Angular based user interface. We also propose a bootloader design that performs device authenticity and integrity checks by cryptographically verifying the firmware against known signatures. This is implemented on a Cortex M3/M4 ARM architecture and doesn't requires any underlying OS to run on.

As further extension, we also developed PCB designs for user (end-product variant) and development board, that greatly ease development and security vulnerability analysis of known proprietary hardware wallets such as the Trezor One.

Chapter 1

INTRODUCTION

Cryptocurrency, being a decentralized digital asset with cryptographic primitives to ensure transactions between entities without the involvement on any intermediator, is lucrative in terms of ownership, privacy and anonymity. The blockchain nature of cryptocurrency makes it decentralized. The irreversibility of cryptographic measures and functions is what makes it secure. Private key and public key being tied down to the user ensure the transactions are verified and signed by the user only. Digital assets are tied to the person's private key. Each transaction is signed by a combination of user's private key and transaction's public key. A person's private hence becomes his biggest asset to prevent any theft and fraudulent attempts.

Securing the private key and accessing your assets is handled by a cryptocurrency wallet. Wallets can be of different nature depending upon the medium of storage. Desktop wallets, that allow the private seed or key to stored locally on your computer, are prone to hacking and phishing attempts. Phone wallets, being similar in nature to desktop wallets, have faced similar threats. Online popular cryptocurrency marketplaces and digital or cloud wallets allow more accessibility and ease but are controlled by third parties. Any cyber-attack on the third party may result in loss of your private key and thus your digital assets.

All above mentioned wallets are software based in nature and prone to numerous vulnerabilities in regard to cyber-theft and fraudulent attempts. The possible solution to such an issue is the use of hardware-based cryptocurrency wallets. Hardware cryptocurrency wallets store the private on a separate device keeping it inaccessible to any attempt of cyber-theft. The transactions are signed on device making it immune to computer viruses, phishing attempts, other cyber-attacks or attempts by hackers for any fraudulent purposes.

The scarcity of research work and publications observed within the domain of hardware cryptocurrency wallets was the key motivation for us to work on it and develop a solution to fasten the progress within this area.

1.1 IMPORTANCE

Hardware wallets, due to securing the private key within separate dedicated hardware deprived of any internet activity, allows the best and the securest current possible solution to safeguard digital cryptocurrency assets from fraudulent attempts of hackers. Current readily available hardware wallets are proprietary in nature and hence lack an open framework. An open framework allows parallel development through modularity. It fast-tracks the overall further development and allow newer opportunities to emerge through its modular approach. It opens new paths to further research and development. A non-empirical study on Hardware cryptocurrency wallets is also compulsory to extend the research possibilities and hence predict the unknown hidden challenges and attacks on hardware wallets in future.

A clear open architecture and framework on hardware cryptocurrency wallets based on the working methodology and principles of hardware wallets is hence formulated within this project. The proposal entails a semi software and hardware approach using ARMv7 cores on a Linux kernel and bootloader on ARM Cortex M3/M4 architecture. Firmware for the wallet and Angular based interface as GUI for interaction were implemented on a single board computer with ARMv7 SOC. The authentication of the firmware, the verification and the cryptographic integrity checks were done on ARM M3 controller without any operating system. PCB design for an end user product was further laid out and a development board was developed to accelerate the growth in research within the domain of cyber security in regard to cryptocurrency wallets and to facilitate the analysis of vulnerabilities on such platform for future attack predictions and to safeguard from the current known threats.

1.2 PROJECT GOAL

The goals of this project include:

- Study and deep comprehension of underlying principles of cryptocurrency and the need of hardware cryptocurrency wallets.
- Understand the need of an open framework to accelerate the research growth in hardware cryptocurrency wallets.

- Develop a hardware wallet solution based on the principles of current available proprietary solutions.
- Allow future research opportunities within this domain through developing a development board to fasten the process.

1.3 REPORT ORGANIZATION

This report is organized into the following six chapters

Chapter 2 *is a literature review that begins by a discussion of cryptocurrency in general and then narrows down to cryptocurrency wallets.*

Chapter 3 *discusses the problem and addresses why the formal study on hardware wallets is necessary moving forward.*

Chapter 4 *gives insight into the functionality and the design and architecture of the implemented wallet*

Chapter 5 *entails the details about the implementation of our proposed cryptocurrency wallet solution*

Chapter 6 *discusses the results that were obtained through our implementation*

Chapter *7 concludes the discussion about the FYP*

Chapter 8 *suggests recommendations for future possibilities related to the project*

Chapter 2

LITERATURE REVIEW

2.1 BRIEF DEFINITION OF CRYPTOCURRENCY

A cryptocurrency is a digital or virtual currency which is secured by cryptography, making it a more secure form of asset. Cryptocurrency is very difficult to forge it due to its secure nature. Cryptocurrencies in essence are based on blockchain technology which are a decentralized network. Blockchain technology is a distributed ledger enforced by a network of computers. The main feature of cryptocurrencies is that these cryptocurrencies are not issued by a central authority which makes them immune to any government or third-party influence. This is one of the most important and valuable aspect of cryptocurrency . It is not subject to any one powerful body which might have its own interest but rather is regulated by a decentralized system which cannot be influenced by people having their own vested interest .

These are some summarized points that helps in explaining the gist of the topic at hand:

- A cryptocurrency is a new form of asset in the form of virtual currency . This new system is based on a network that is based upon the distribution across a large number of computers. This decentralized construction permits them to exist independently and without any government or third-party interference or influence.
- The term cryptocurrency is based on the concept of encryption practices which are used to protect the networks.
- Blockchains are organizational methods for ensuring the reliability and integrity of the overall system, which is of utmost importance for cryptocurrencies.
- There is a threat seen for the more established industries of finance and law from the blockchain and related technology as many experts believe that these newer technologies can be used to bypass the existing finance model and different sanctions can be rendered useless.

- Cryptocurrencies have its critics who criticizes it for different reasons, including their use for illegal activities such as black-market transactions, exchange rate instability, and liabilities of the infrastructure underlying them.
- In spite of the criticism this system is seen as a more portable option .Another favorable aspect of this technology is its independent nature which gives user more confidence. Furthermore, the systems inflation resistance aspect is also a plus point.

Cryptocurrency can be understood as a system that allow for the secure payments online which are denominated in terms of virtual "tokens," which are represented by ledger entries internal to the system. "Crypto" refers to the various encryption algorithms and cryptographic techniques that safeguard these entries, such as elliptical curve encryption, public-private key pairs, and hashing functions. [14]

2.2 BITCOIN HISTORY

The first and most popular and valuable blockchain-based cryptocurrency is Bitcoin. To this day bitcoins remain the most valuable virtual currency but having said that, there are numerous alternate cryptocurrencies developed as well. Some of these new cryptocurrencies are quite valuable and provide an alternative to the more established Bitcoin. Among these newer forms of virtual currency some are based on Bitcoin and are seen as its extension while others are new virtual currencies that were built from scratch and have its own separate technology.

Bitcoin was created in January 2009 following the financial crisis and the crash housing and banking system by an individual or group known by the name of "Satoshi Nakamoto”. The identity of this individual or group remains a mystery and is discussed in more detail in the coming pages of this document. By the last estimates which were taken in November 2019 there existed over 18 million bitcoins with a total market value of around $146 billion. Due to its ever-increasing demand the value of these bitcoins have increase enormously touching a high of $20000. Bitcoins are traded around $10000 at the time of this document creation.

Virtual currencies like Bitcoin promises a lower transaction fees as compared to the traditional online payment mechanisms .This low transaction fee also makes it an attractive proposition for interested people who would want to conduct their transactions through this avenue. The absence of any centralized body or government attracts some shady transactions as well.

Some of the alternatives inspired by the success of Bitcoin are known as "altcoins," include Litecoin, Peercoin, and Namecoin, as well as Ethereum, Cardano, and EOS. Today, the aggregate value of all the cryptocurrencies in existence is around $214 billion—Bitcoin currently represents more than 68% of the total value. [3]

Secrecy of Founder

The identity of bitcoin's founder has always been subject to many conspiracy theories and many questions. The Secrecy surrounding Satoshi Nakamoto has been discussed by many people interested in this technology .There are general agreements on the reason why the full identity of the founder has not been revealed to people. First reason is that the whole concept of cryptocurrency is based on secrecy and privacy so naturally the founder would not want him/herself to be subject to all the scrutiny they will face by different governments and organizations if their name come to limelight.

With the introduction of virtual currency there is possibility that there could be major disruption in the currently established banking and monetary systems If there is a mass adoption of Cryptocurrency than the system can become more valuable than different countries own currency. The current economic model have a few parties who have a lot of influence on the system and can sanctions different parties . This clout will be lost with a decentralized system that the virtual currency promises . Due to this the bitcoins creator can become a target for different entities .

There is another aspect to this secrecy subject. Looking only at the data of year 2009, we see that a total of 32,489 blocks were mined which had the value of 50 BTC per block, the total payout in 2009 was 1,624,500 BTC whose total value was $13.9 billion dollar in Oct 2019. It would be natural to assume that only Satoshi and perhaps a few other people were mining through 2009 and that they possess a majority of that stash of BTC. If the real identity of the Satoshi is revealed than he/she would become a target for criminals and that person would attract unwanted attention. While it's likely the inventor of Bitcoin would take precautions to make any extortion-induced transfers traceable, remaining anonymous is a good way for Satoshi to limit exposure.

Following is a timeline which will show the path followed towards the creation of bitcoin

- *Aug. 18, 2008*: The bitcoin.org domain name is registered [18].
- *Oct. 31, 2008*: Satoshi Nakamoto which is the name of the group or an individual announces on The Cryptography Mailing list at metzdowd.com stating that: "I've been working on a new electronic cash system that's fully peer-to-peer, with no trusted third party. The paper is available at http://www.bitcoin.org/bitcoin.pdf." This link leads to the now-famous whitepaper published on bitcoin.org entitled "Bitcoin: A Peer-to-Peer Electronic Cash System." This Paper became the groundwork and is known as the Magna Carta of Bitcoin operation.
- *Jan. 3, 2009*: The first Bitcoin block, Block 0, is mined. This first block is known as the "genesis block". It contained a statement regarding political commentary
- *Jan. 8, 2009*: Bitcoins Software first version is announced on The Cryptography Mailing List
- *Jan. 9, 2009*: Bitcoin mining starts after the second block is mined.

2.3 PEER BROADCASTING AND MINING

The bitcoin cryptocurrency system of network is a peer-to-peer transaction network that follows a cryptographic protocol. Participants pay and receive bitcoins by broadcasting digitally signed messages to the network using bitcoin cryptocurrency wallet software. Transactions are recorded into a distributed, replicated public database known as the blockchain, with consensus achieved by a proof-of-work system called mining. Satoshi Nakamoto, the designer of bitcoin, claimed that design and coding of bitcoin began in 2007. The project was released in 2009 as open source software.

The network requires minimal structure to share transactions. An ad-hoc decentralized network of volunteers is sufficient. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will. Upon reconnection, a node downloads and verifies new blocks from other nodes to complete its local copy of the blockchain. [9][10]

Different Commentators and experts of the financial system had stated that blockchain technology has a lot of potential for different uses such as online voting and crowdfunding. JPMorgan Chase (JPM) among other major financial institutes views this technology favorably because of lower transaction costs it offers due to its streamlined payment processing [4]. There is a problem which currently faces this overall setup because cryptocurrencies are virtual in nature and has no physical form and are not stored in a unified database , so the entire balance can be lost just by the destruction of a hard drive if the private keys are not stored in a backup storage. This can also be seen as a major benefit as no other party has any control over the resources owned by you.

2.3.1 Blockchain Fundamentals

- All the virtual assets are not copied or transferred but instead distributed.
- Full real time access is given as the assets are decentralized
- The assets are trusted due to the transparent nature of the ledger

Blockchain helps in reducing risks and brings transparency to the system . Furthermore, this technology helps us stamp out fraud and is therefore heralded as a promising and revolutionary technology

Blocks

The Chain is made up of multiple blocks and each and every block is made up of three basic elements:

- The data that the block consists of.

- A whole number called nonce which is 32-bit. The generation of nonce is done randomly when a block is created, after which it generates a block header hash.
- The hash is a 256-bit number wedded to the nonce. It should be extremely small.

The nonce generates a cryptographic hash when the first block of a chai is created. The data contained in the block is considered signed and is tied to the nonce and hash until it is mined. Miners create new blocks on the chain through a process called mining.

In a blockchain there is a unique none and hash for every block, it also references the hash of the previous block in the chain, so therefore the mining process of a block isn't straightforward particularly on long chains

There are special software that miners use to solve the very complex math problem of finding a nonce which generate an accepted hash. As the nonce is 32 bits and the hash is 256 bits ,so therefore there are approximately four billion possible nonce-hash combinations that must be mined before the right one is tracked. When this occurs, miners are said to have found the "golden nonce" and their block is added to the chain. [16]

It is very difficult to manipulate the blockchain technology as making a change to any block earlier in the chain will require mining of all of the blocks that come after. This is why it's extremely difficult to manipulate blockchain technology. When a successful mining is done , the variation is accepted by all of the nodes on the network and the miner is rewarded financially.

Nodes

Decentralization the cornerstone of blockchain technology and no single computer or organization can own a chain on its own .instead there is a distributed ledger through the nodes connected to the particular chain. Any kind of electronic device that have copies of blockchain and keeps the network functioning can operate as a node.

There are several copies of blockchain contained in every node and any new mined block will have to be algorithmically approved for the chain to be updated , trusted and verified. Ledger can be checked after every action as it is transparent . Each User is given an identification number that shows their transactions. The combination of public information with the checks and balance ensures that blockchain keeps its integrity and develops trust among people.

2.3.2 Mining

In Cryptocurrency mining process different transactions for different forms of cryptocurrency are verified and added to the blockchain digital ledger. Also known as crypto-coin mining, altcoin mining, or Bitcoin mining (for the most popular form of cryptocurrency, Bitcoin), As the virtual currency use has increased exponentially in the last few years, so the mining has also increased due to this. Another factor is the increased value of virtual currency which has boosted mining.

Each time a cryptocurrency transaction is made, a cryptocurrency miner is responsible for ensuring the authenticity of information and updating the blockchain with the transaction. The mining process itself involves competing with other crypto miners to solve complicated mathematical problems with cryptographic hash functions that are associated with a block containing the transaction data [17].

The first cryptocurrency miner to crack the code is rewarded by being able to authorize the transaction, and in return for the service provided, crypto miners earn small amounts of cryptocurrency of their own. In order to be competitive with other crypto miners, though, a cryptocurrency miner needs a computer with specialized hardware [17].

Individuals using their own dedicated can generate an income by earning one to two dollars per day in most cases. Electricity bills , internet connection expenses, and computing hardware also effects the net revenue generated .

To start mining ,miners will require dedicated computer hardware with a specialized graphical processing unit (GPU) or application-specific integrated circuit (ASIC),a cooling system for the hardware , a stable internet connection, a cryptocurrency mining software package, and membership in both an online cryptocurrency exchange as well as an online mining pool [17].

Aspiring cryptocurrency miners their research because cryptocurrencies have risen in both use and value, competition has increased exponentially as well and there are now organizations with more extensive resources dedicated to mining which most individuals cannot compete .

2.4 PUBLIC AND PRIVATE KEY GENERATRION PROTOCOL

A pair of keys : Public Key and Private Key are used in Public Key Cryptography, also known as Asymmetric Cryptography. This is the most important aspect of virtual currency concept and protocols and is widely used to make sure that the online currency is only spent by their owners and only the owners can sign of transactions with digital signature, In short, if you send cryptocurrencies to others you sign that transaction using your private key or signature key, which is generated using private key and transaction is verified using your public key. So, if hackers get their hands on your private key, they would be able to send your cryptocurrencies to themselves.

There exists number of algorithms to generate public and private keys. For example, Bitcoin protocol uses Elliptic Curve Digital Signature Algorithm This algorithm is named after its founders, Ron Rivest, Adi Shamir, and Leonard Adleman and it has become the algorithm of choice within the public key cryptography enthusiasts. [15]

Elliptic Curve Digital Signature Algorithm or ECDSA/RSA makes wide-ranging use of arithmetic operations like mod *n* (modulo n) arithmetic. For example, *5 mod 2* simply means the remainder of 5 when divided by 2 which is equal to 1 in this example . Another example is shown here *13 mod 5 = 3*. In general, there are three major parts of ECDSA stated below :

- Generation of both the public key and the private key
- Encryption of data using generated public key
- Decryption of data using generated private key

Generation of the public key and the private key

For public and private keys Generation, A or/and B performs the following steps:

1. Select two large prime numbers, p and q. The larger the values, the more difficult it is to break RSA, but on the other hand the encoding and decoding takes longer to complete.
2. Calculate $n = pq$ and $z = (p — 1)(q — 1)$.
3. Choose a number e, less than n, that has no common factors, other than 1, with z or their greatest common divisor (gcd) equals 1, $gcd(e, z) = 1$. In this case, e and z are said to be relatively prime. e will be used in encryption.
4. Find and select a number d, such that $ed — 1$ is exactly divisible by z. In another way $ed \bmod z = 1$. d will be used in decryption.
5. The public key that B or A makes available to the world is the pair of numbers (n, e) and the private, which must be secret, is the pair of numbers (n, d).

Encryption of data using generated public key

Let's Suppose A wants to send B a message, which is represented by bit pattern m (plaintext message), where $m < n$. The encrypted value c (ciphertext) of plaintext message m is $c = m^e (mod\ n)$. Ciphertext c will be sent to B. Note that A encrypts message using B's public key [15].

Decryption of data using generated private key

To decrypt the received ciphertext c B computes $m = c$^$d \bmod n$ which requires use of his private key (n, d).

The security of ECDSA/RSA depends on the fact that there are no known algorithms for quickly factoring (prime factorization) a number. In this case the public value n into p and q.

2.5 TRANSACTION SIGNING PROCEDURE

Inputs and outputs are what basically makes a transaction. Inputs are bitcoins that have been assigned to a wallet. Inputs are essentially the bitcoins you earn, and outputs are the bitcoins you spend.

It is important to understand that inputs are no different than the various denominations of your local currency in your physical wallet. You may hold a $1 coin, a $2 coin, and maybe a $5 bill in your wallet which brings the total amount you hold to $8. These three separate values can be considered as the inputs of your wallet.

Before you can spend bitcoins, you are required to prove your ownership of those funds. This is done by the use of locking and unlocking scripts. Every input in your wallet was an output from another wallet. Before the other wallet transferred those funds to you, it placed a requirement using the hash of your public key (also known as your bitcoin address), that is, only the public key that can produce the aforementioned hash can have access to the funds. In other words, since the hash can only be produced by you, the funds are locked to you.

Locking and unlocking scripts

There are two types of scripts involved in this process. The locking script and the unlocking script. The locking script is provided by the wallet whose outputs are the inputs to your wallet. The steps run by the locking script are:

```
OP_DUP OP_HASH160 <PublicKeyHash> OP_EQUALVERIFY
                  OP_CHECKSIG
```

There are two types of steps in the script. Each step is either an operation or something that puts data in for the next operation to act on. To elaborate, here is what the steps of the locking script do:

`OP_DUP:` An operation that creates a duplicate of a value.

- `OP_HASH160:` An operation that creates a hash of a value.
- *`<PublicKeyHash>:`* Data that represents the hash of your public key.
- *`OP_EQUALVERIFY:`* An operation that aborts the script if it fails.

- *`OP_CHECKSIG`*: An operation that attempts to verify a digital signature and generates either true or false.

The requirement placed by the wallet that sent you bitcoins was actually for you to complete the locking script in a way that the end result of the script is the value "true". In other words, what data would have to be given to the locking script, so the end result is the value "true"? This is where the unlocking script comes in.

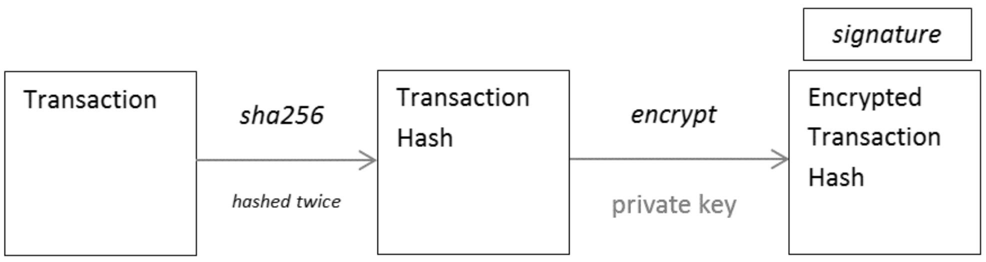
```
Unlocking script + OP_DUP OP_HASH160 <PublicKeyHash>
          OP_EQUALVERIFY OP_CHECKSIG
```

The unlocking script is provided by your wallet which is trying to unlock the inputs for use. This script provides the data that, when put together with the locking script, will result in the value "true". The steps run by the unlocking script are:

```
<signature> <PublicKey>
```

- *`<signature>`*: Data that represents the digital signature generated using your private key.
- *`<PublicKey>`*: Data that represents your public key.

Figure 1 Generating the Digital Signature

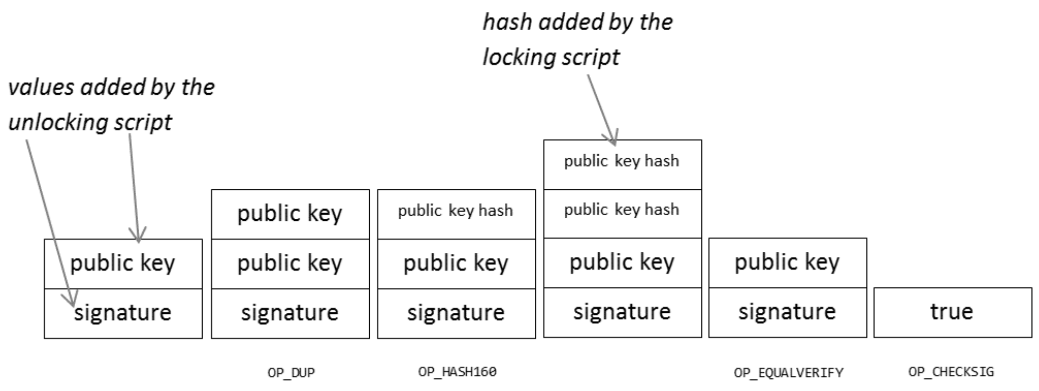


When put together, the complete script is:

```
<signature> <PublicKey> OP_DUP OP_HASH160
<PublicKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

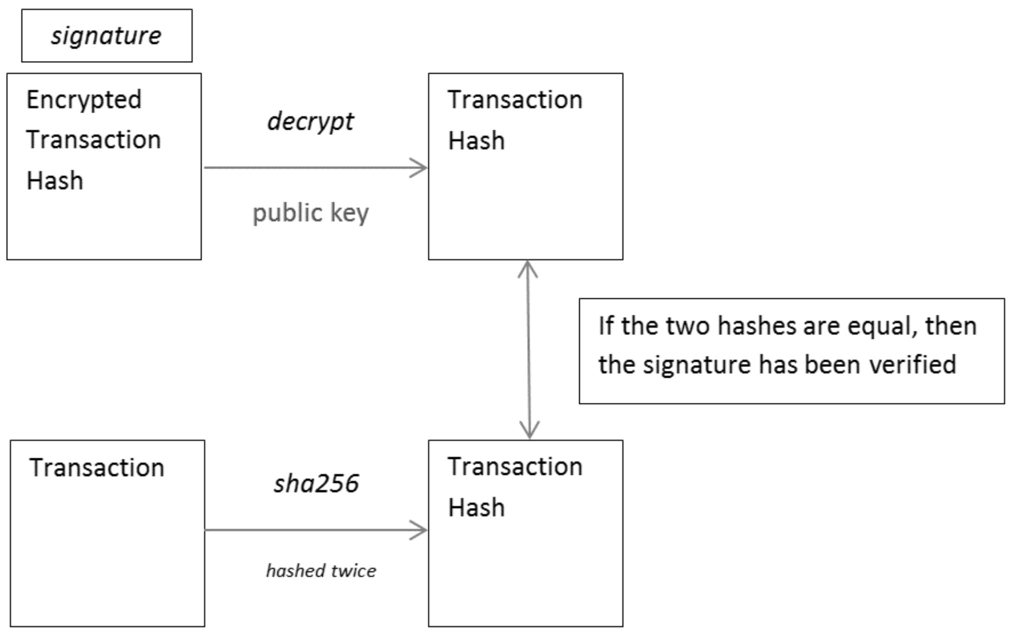
The script runs as a stack-based execution, meaning, results of each step are added to the top of a stack and any operations run by the scripts use the data from the top of the stack as parameters. Processing starts with two pieces of data: the digital signature (created using your private key) and your public key. These two values are added to the stack. The digital signature is created by hashing the contents of the transaction twice and then encrypting the resulting hash using your private key.

Figure 2 Stack Based Script Execution



The first operation executed by this script is `OP_DUP`. This creates a duplicate of the value at the top of the stack, that is, the public key. The next operation is `OP_HASH160` which takes the top value, which is the public key, and hashes it. Next, the public key hash included in the locking script is added to the stack. Then, `OP_EQUALVERIFY` is executed which compares the two public key hashes. If the two hashes are not equal, then execution of the script fails and access to the inputs is denied. Comparison of the hashes can fail in the case where anyone other than the intended recipient of the funds tries to access them. If the hash comparison passes, then we are left with only two values: the public key and the signature. Finally, the last operation to run is `OP_CHECKSIG`. This operation will attempt to confirm the digital signature using the public key provided.

Figure 3 Digital Signature Verification


signature
Encrypted Transaction Hash
decrypt
public key
Transaction Hash
If the two hashes are equal, then the signature has been verified
Transaction
sha256
hashed twice
Transaction Hash

Since the digital signature was created by your private key, it can only be decrypted using your public key. This public key was provided through your unlocking script. Decryption will result in a hash. If this hash matches the hash of the hash of the transaction (yes, hashed twice as mentioned earlier), then the result of the locking script is true, and you are granted access to the inputs to be spent.

2.6 Wallet Concepts and Options

There are several options available in the market for securing and accessing your digital assets. Software , Hardware and Paper wallets are three most popular options which we will have a look at.

Software Wallets

Software wallets can be computer , mobile or cloud based .

- Desktop based wallets work such that wallets are downloaded and installed on your personal PC. You can only access your digital assets by that one PC or laptop where the wallet is installed .It is a secure way to operate but if your

PC or laptop is lost or hacked than there is a risk that you might lose all your digital assets.

- Cloud /Online :The wallets are operated on cloud computing making it accessible from any computer throughout the world . This provides flexibility to the user but essentially a third party stores your private keys, and this can lead to theft and hacking attacks . The third party can also use its influence to restrict your access to your digital assets.
- Mobile: These are wallets run on an app on your personal mobile . This way you can your digital assets anywhere and the device is such that you can carry it anywhere. These are simpler due to the limited space available on mobile devices.

Hardware Wallets

The main difference between a hardware and a software wallet is that your private keys are stored on a piece of hardware like an USB . Transactions are made online, but they are stored in an offline device which makes it more secure . A hardware wallet can support different currencies and thus provides the user with more options. The making of a transaction is simple . Owner can plug their device to any computer with an internet connection and enter a pin and confirm their transaction .Hardware wallets makes it easy to perform transactions and furthermore keeps the assets secure from hacking with its offline feature

Paper wallets

Paper wallets are simple to use and are very secure. The Concept and use of paper wallets is very straight forward. Paper wallet is a fundamentally a software which generates a pair of public and private key which are then printed . Any transactions are done by transferring of funds from your software wallets to the public address shown on your paper wallet while if user wants to withdraw or spend their assets, they can simply transfer their earning from the paper wallet to their software wallet [15]. This process , also known as sweeping , can be accomplished by entering your private keys or by scanning the QR code on paper wallet.

Among all the options available Hardware wallet is an option which provides greatest flexibility and security

A hardware wallet is a type of wallet where the user can store their private keys in an offline device. It is a good option for people who are not well versed in coding and technical details. It allows the secure storage of al cryptocurrency of a user and provides greater flexibility

Chapter 3

PROBLEM DEFINITION

Cryptocurrency wallets exist to safeguard the underlying threat of theft and illegal procurement of cryptocurrency, but the safety measures of various wallets are questionable due the rise in evidently observed cyber thefts of such solutions [23]. The possible securest solution is the use of hardware cryptocurrency wallets, but the lack of an open framework and a defined formal model has hampered the research advancement in prediction of future unknown threats on the platform.

3.1 RISE IN CRYPTOCURRENCY THEFT

The operation without a central figure is one of the major propositions that attracts the masses. The use of hash functions and digital signatures is the most salient characteristic of cryptocurrency. These primitives can easily be computed but cannot reversed with ease. In fact, the reversibility is so low that it is stated to be practically impossible [19]. Transactions use asymmetric cryptography. To send cryptocurrencies to others transaction is signed using a private key and then is verified using public key. Hackers would be able to send your cryptocurrencies to themselves if somehow, they obtain your private key. Securing the private key hence becomes integral to avoid unauthorized transactions that may or may not to be a result of phishing, hacking or a supply chain attack.

Cryptocurrency wallets allow to keep the private seed or key used to digital sign the tractions safely. Various types of cryptocurrency wallets exist within the current world scenario and they are mainly categorized based on the medium of information storage whether on software, physical (paper) or dedicated hardware. Software depending upon the nature of device and storage access can be either online, desktop and mobile.

The action of storing your private seed or key on your local device via Desktop wallet makes it exploitable by a virus or malware and vulnerable to hacking. With global detections topping 50 million in numbers (as reported by

Malwarebytes) [20] and with the increase in phishing attempts by 640% last year (as reported by Webroot) [21], the possibility of being targeted specifically to access your private key rises. Targeting selective individual victims via spear phishing has increased the successfulness of the hacker in obtaining personal information. In this day and age of social computing in the form of Facebook, twitter and Instagram, recognizing patterns and habits enables targeted phishing attack much easily. A recent example is hijacking of email reply chains where upon access to user's email or social account, the hacker infects the contacts or friends of the victim by replying with links to genuine conversations threads within email or the social network's message thread [21]. Being the part of contact list the conversation remains most likely unfiltered and hence increases the likeliness of the recipient to open it as the conversation is made deliberately believable and thus infecting the system via trojan or keylogger in attempt to receive personal data and classified information in form of bank passwords, credit card numbers and possibly the private key stored within the desktop cryptocurrency wallet of the person for illegal and morally wrong gains. Another recent prime example is the bargain Trojan malware called *MasterMana Botnet*. It mass-mailed cryptocurrency investors with malicious code to create backdoors on their computer to access their wallets [22].

Mobile Wallets where the private seed may be stored locally are prone to similar threats. The rapid increase in its growth in comparison to other platforms has made it the ideal platform for targeting as it is accessible to billions of peoples which may or may not have the necessary know-how to protect themselves and their information from the malicious attempts by these hackers. For a hacker, the ability to access text messages, calls, voice messages, email, camera, location via GPS and all the keystrokes a user makes is very striking and thus making it more sought out platform in contrast. The major issue with mobile devices as a whole is the non-uniformity of the platform version where security updates are not provided to older systems still running on millions of devices due to underpowered hardware and other unspecified reasons due to which the manufacturer stops updating the firmware. The reason to which security support is dropped is debatable but the fact that all those devices remain vulnerable to security exploits remain true and this led to the possibility of accessing the device for personal information. Around 40% of android market share is still running version less than android v9 and this makes the conversation extremely relevant [21].

Online wallets run on cloud give the ease and accessibility to use it from a multiple array of devices but being controlled by third party makes them more prone to hacking attacks and thefts evident by the multiple cryptocurrency heists by various hacking groups observed throughout the past few years since the emergence of cryptocurrency. One of the most famous cryptocurrency hacks is Mt. Gox. The Japan based cryptocurrency exchange was hacked twice, first in 2011 when the auditor's credentials where accessed by the hackers and 2609 BTC were transferred to an unknown account and second in 2014 when they held 70% BTC transactions more than 750,000 BTC went missing [24]. 2019 alone, $292,665,886 worth of cryptocurrency and 510,000 user logins were stolen from online cryptocurrency exchanges and wallets [23].

3.2 POSSIBLE SOLUTION – HARDWARE WALLETS:

Hardware wallets instead of storing on either cloud or your local device whether your computer or mobile device, store private keys on a dedicated hardware device such as a USB stick. The compatibility exists across multiple platforms enabling widespread use. The transactions are made online, but verification is done on device and the private keys remain unexposed.

The security benefits of hardware wallets are worth the relatively high cost it is usually associated in contrast to other cryptocurrency wallets. In a hardware wallet, the private seed or key remains on device throughout the transaction keeping the key unexposed to the computer and the internet and thus making it resilient to copying and hacking. This provision of immunity to computer viruses make these wallets extremely viable option in comparison to other alternatives in this day and age where security threats have become prevalent and widespread even with exponential advancement observed in the fields of cyber security. In Trezor, a hardware proprietary wallet, a write protected, and non-modifiable bootloader checks the integrity of bootloader [24]. This provides an automatic defense mechanism against any attack or code injection that can reconfigure the program. The bootloader checks the integrity of the firmware which operates the device and executes its main functionality and keeps its operations secure [24]. The firmware is updatable but with physical confirmation by the user. The memory is erased during the process and only recovered upon signature verification.

A user provided PIN or password provides an added security layer to prevent unauthorized physical access or forceful activation of the device in case of theft and with added self-destruction upon a specified number of wrong attempts makes the keys deleted from the device. Passphrase protection within Trezor builds a completely new hidden wallet over your private seed making it more secure through this addition of entropy to the loaded seed. The passphrase being not within device makes it a robust untraceable and unbreakable solution [24]. Recovery seed serves as an ultimate backup to recover a wallet for long term safety purposes.

These practices make the hardware wallets most secure solution for cryptocurrency transactions currently and provides protection to the user's private key.

3.3 DEFINED FORMAL MODEL AND OPEN FRAMEWORKS FOR HARDWARE WALLETS

3.3.1 Open Frameworks

Open frameworks govern on the principle of modularity and replicability of those modules for further development of the application or to develop newer applications [25]. The accessibility of the modules and the openness of the code is what makes it an open framework. This approach towards design allows parallel development allowing the task division in small chunks with common interface for interconnectivity. Reusability of the modules for different applications is possible via this approach. Hence it accelerates overall growth in software development when the developer is focused on the problem in hand in contrast to developing everything in house inclusive of underlying interfaces integral for the application but irrelevant to main problem. This not only reduces cost but is more effective and efficient. Open frameworks thus have provided us a significant leap in application development [26].

3.3.2 Proprietary Hardware Wallets and Lack of Open Framework

Proprietary in accordance with Merriam-Webster dictionary is defined as “used, made, or marketed by one having the exclusive legal right” [27]. Any hardware cryptocurrency wallet having any part of its interface whether software or hardware controlled and distributed by its owner under law and protection rights in the form of patents to protect intellectual property falls under the category of proprietary hardware cryptocurrency wallet. Although many components of proprietary hardware cryptocurrency wallets are openly available under various open source licenses, but no hardware wallet has a fully open framework to enable full transparency and thus hampering the development of the hardware wallets. Development and documentation of a fully open sourced framework of the various component of a hardware wallet has become hence compulsory to accelerate the potential research growth in the area of cyber security with reference to the domain of cryptocurrency as it has become one of the most significant assets moving forward. This will enable newer research ideas to cater unforeseeable threats when designing new secure platforms at the hardware level. Foundation for research has to be therefore laid as this can be very welcome start to new research horizons within the field of cryptocurrency hardware wallets. SmartOTPs represents the how potent the research in this domain can be [29].

3.3.3 Formal Methods and our Approach

Formal methods are “system design techniques that use rigorously specified mathematical models to build software and hardware systems” [28]. Complex systems are modeled after mathematical entities for verification of system properties other than empirical testing methodology.

Non-empirical study of cryptocurrency wallets is scarce with only few recent publications [30][31]. We aim to revitalize the existing publications to provide a foundation for further development of non-empirical study on wallets.

Chapter 4

DETAILED DESIGN AND ARCHITECTURE

PART A:

RASPBERRY PI BASED BITCOIN HARDWARE WALLET

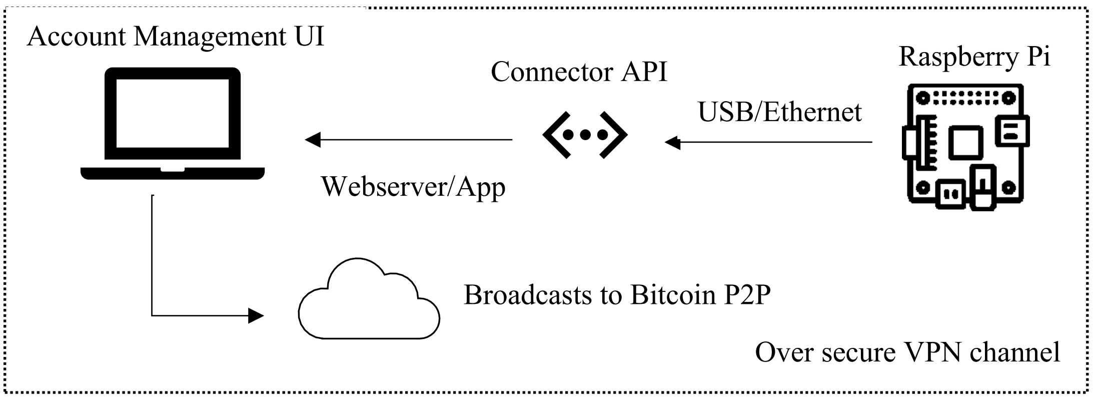
4.A.1 SYSTEM ARCHITECTURE

The hardware wallet is a complex project, requiring deep understanding of different software frameworks and hardware components. A functional prototype was developed using Raspberry Pi with the ability to perform basic wallet functionalities such as PIN protection, key generation, transaction generation and signing. However, the generated transaction was not broadcasted to the Bitcoin P2P network and no such implementation was thus added to comply with State Bank's strict policy against cryptocurrency usage and transactions. However, while this prototype doesn't check for secure communication and attesting for genuineness of the device, it does however encrypt all generated keys using AES ciphers to prevent an adversary from stealing the keys.

4.A.1.1 Architecture Design

Figure 4-1 below shows a general architecture of the prototype. All components are explained in detail in the subsequent sections.

Figure 4 General Architecture for Prototype A



A basic user flow is as follows:

1. **Acquire Raspberry Pi:** Needless to say, a Raspberry Pi (or any ARMv7 base running Linux GNU) is required. This can be bought easily at any retailer. An OS must be preloaded to the device before continuing.
2. **Install Software:** User will contact us or visit a website to download and install the connector and account management software on their device.
3. **Connect to PC:** User will connect the device to either Wi-Fi or through a USB to the PC. However, the device won't work unless the IP is whitelisted in the VPN network controlled by us.
4. **Access Management UI:** User will access the web management portal to generate a new PIN and sign into new account.
5. **Generate Private Key:** User will generate either a new private key or add a previous key for their preferred cryptocurrency.
6. **Create Transaction:** Using previous transaction history addresses and IDs, user can create new sending transactions by defining amount and destination address. The signed transaction however has to be manually propagated to the Bitcoin network
7. **Receiving Transactions:** Compressed and Uncompressed Public Keys are generated using the private key, allowing the user to make receiving transactions.

4.A.1.2 Subsystem Architecture

A brief explanation of the architecture components is explained below:

User

Needless to say, the user is responsible for all human interactions as mentioned in the basic flow above. In addition to that:

- The PIN should be backed up Keys cannot be recovered if either the PIN is lost.
- It's the responsibility of the user to keep the hardware wallet safe and secure from any physical attacks.

Hardware Wallet

The hardware wallet performs the following high-level operations:

1. **Initialize**
    a. Prompt user to generate a new PIN.
    b. Verify PIN is correct before decrypting and loading the wallet details.
2. **Address Management**
    a. Generate new cryptocurrency address or ask user to import an existing one.
    b. Manage multiple cryptocurrency accounts.
    c. Generate fresh public address for each transaction.
3. **Transactions Signing**
    a. Sign all transactions using the stored private key, encoded in WIF (Wallet Interchangeable Format).
    b. Prompt user to enter account and amount details prior to each transaction.
4. **Interaction**
    a. Dump all wallet details on PIN validation.
    b. Destroy wallet details (permanent and unrecoverable).
    c. Verify if wallet exists and validates against provided PIN combination.

Connector

The connector emulates an Ethernet connection over USB to allow the installed webserver on hardware wallet to be accessible to user's PC. The connector can be also be activated by allowing the wallet internet connectivity access through a secure VPN channel.

The connector handles the following endpoints (shown in Table 4-1):

Figure 5 Connector Endpoints

Endpoint	**Parameters**	**Response**	**Explanation**
/unlock	PIN/Password	nil (OK)	Takes PIN from user and decrypts the stored wallet
/generate	Wallet Type, Optional Private Key	nil (OK)	Creates new WIF based on the input parameters
/dashboard	-	-	Frontend management (includes other endpoints not mentioned here)
/transaction	Amount, transaction hash, address	Binary (serialized transaction)	Returns a signed transaction to be broadcasted
/dump	PIN/Password	WIF	Returns the whole decrypted wallet details
/destroy	PIN/Password	-	Destroys the current stored wallet

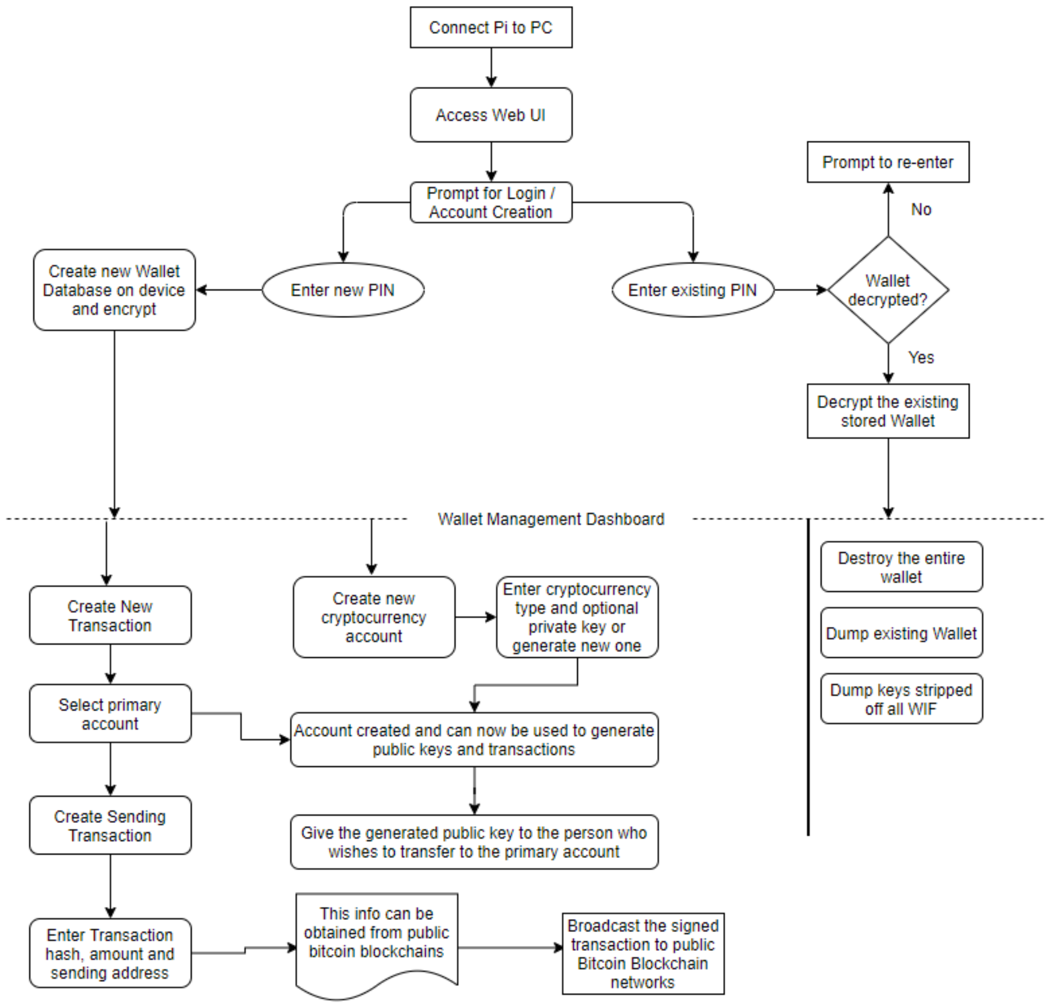
Wallet Web UI Interface

The Web UI is hosted directly on the hardware wallet and is accessible on PC through any modern web browser. A private IP is assigned to the wallet webserver which can be used to directly access the interface. Do note that for a domain name mapping to the interface, user's hosts files have to be changed to allow browser to properly service the domain name redirects to the proper IP address.

The interface is built on modern web-based technologies as a single application package (SAP). Further details on the implementation are listed in the implementation section of the report.

Figure 4-2 below shows a general flow chart of the capabilities and usage of the web dashboard:

Figure 6 Flowchart of Web Administration Portal


Connect Pi to PC
Access Web UI
Prompt for Login / Account Creation
Enter new PIN
Enter existing PIN
Create new Wallet Database on device and encrypt
Prompt to re-enter
No
Wallet decrypted?
Yes
Decrypt the existing stored Wallet
Wallet Management Dashboard
Create New Transaction
Select primary account
Create Sending Transaction
Enter Transaction hash, amount and sending address
Create new cryptocurrency account
Enter cryptocurrency type and optional private key or generate new one
Account created and can now be used to generate public keys and transactions
Give the generated public key to the person who wishes to transfer to the primary account
This info can be obtained from public bitcoin blockchains
Broadcast the signed transaction to public Bitcoin Blockchain networks
Destroy the entire wallet
Dump existing Wallet
Dump keys stripped off all WIF

Bitcoin P2P Network

We purposively didn't include the ability to broadcast the transaction to bitcoin blockchain as this requires downloading the entire 20 GB blockchain on the user's end. Instead, any public P2P network API can be used to broadcast the transaction to the global network.

The resulting signed transaction can be uploaded to a blockchain explorer such as: https://insight.bitpay.com/tx/send. However, do note that a valid Transaction hash has to be used for the transaction to be accepted into the blockchain network.

4.A.2 DETAILED SYSTEM DESIGN

The detailed system design outlines the generalized routines, functions and classes that build up the main system. Implementation details are left in the subsequent sections.

4.A.2.1 Private and Public Keys

The core wallet functionality to generate new coins (i.e. private keys and public keys) and import existing ones are wrapped up inside a package.

Classification

This is main package comprising of coins and currency specific network structures and the necessary helper functions.

Responsibilities

The following cryptocurrencies are supported:

• Litecoin (LTC)	Public: 0x30 Private: 0xb0
• Dash (DASH)	Public: 0x4c Private: 0xCC
• Bitcoin (BTC)	Public: 0x00 Private: 0x80

The public and private hex codes are the prefixes for the public and private keys for each cryptocurrency respectively. These are part of the core specification and are extracted from the currency repositories.

A general structure thus looks like:

```
var network = map[string]Network{
    "btc":  {name: "bitcoin", symbol: "btc", xpubkey: 0x00, xpriva
tekey: 0x80, magic: 0xf9beb4d9},
    "ltc":  {name: "litecoin", symbol: "ltc", xpubkey: 0x30, xpriv
atekey: 0xb0, magic: 0xfbc0b6db},
    "dash": {name: "dash", symbol: "dash", xpubkey: 0x4c, xprivate
key: 0xcc, magic: 0xd9b4bef9},}
```

The magic attribute is used in generating the private keys. While they can be arbitrarily chosen to some degree, these values are ‘secure’ to use in cryptography routines.

Interface/Exports

Following routines are handled by the coins package:

- **GetNetworkParameters**
  Returns a *Network Structure* that contains the private and public keys, and the currency’s network magic value used in cryptography routines for signing transactions.
- **GenerateNewCoin**
  Generates a *New Cryptocurrency Coin* structure. If a private key is provided, it is used to generate the accompanying public key and stored in the wallet.
- **ImportExistingCoin**
  Imports an already existing *Coin Structure* if the user provides the private key. This key is then used to generate other coin parameters and stored in the wallet.

4.A.2.2 Wallet Encryption

The wallet encryption package adds an AES encrypted wrapper over the wallet data protected using a PIN/Password, thus ensuring that in events of device theft or loss, the stored keys can never be extracted.

Classification

A cryptographic wrapper that serializes coins data and encrypts them, and decrypts and deserializes them as per the requirement. Uses open-source AES libraries to accomplish encryption. Used libraries are mentioned in implementation section.

Interface/Exports

- **CreateHash**
  Generates *MD5 Hash of User Password/Pin* that is used to encrypt the wallet. RAW password is never stored or transmitted elsewhere in the device beyond this point and is immediately destroyed.
- **EncryptWallet**
  Encrypts the coin structures and serializes them. Uses PIN/Password to encrypt the data using AES and stores it in EMMC storage of the wallet in serialized form.
- **DecryptWallet**
  Performs opposite function to the encryption function

4.A.2.3 Wallet Management

A helper package that adds helper functions to make interacting with wallet and coins manageable and easier.

Classification

The wallet package encloses the coin and encryption routines, acting as a high-level package that hides those implementations from the user endpoint perspective.

Interface/Exports

- **CreateWallet**
  Creates a new Wallet. Takes a PIN/Password as input. The created wallet is then stored on the disk
- **DestroyWallet**
  Deletes the encrypted wallet on disk. This process is non-recoverable and deletes all private keys and coins data.
- **ImportData**
  Adds coins data to the encrypted wallet. Connects with frontend to receive coins data or generates a new one.

- **Dump**

  Returns all private keys of all stored coins to the user. Should be only done if user wishes to migrate to another device/platform. These must be destroyed after migration or whatever the intended purpose for their extraction was.

- **Authenticate**

  Checks PIN/Password to see if it is valid for the stored wallet. This does not decrypt the stored wallet. Only checks if PIN is valid.

- **GetAddress**

  Returns coins public keys stripped off WIF information stored in the wallet data.

4.A.2.4 Creating and Signing Transactions

This is the core part of wallet functionalities:

1. Extract WIF coin data and get the private key.
2. Generate public key script for the transaction.
3. Creates origin transaction (or provide your own UTXO data)
4. Creates redeem transaction.
5. Sign the redeem transaction so funds can be transferred.
6. Validate the transaction before it can be broadcasted.

Structure

A generalized transaction structure is shown below:

```go
type Transaction struct {
	TxId               string `json:"txid"`
	SourceAddress      string `json:"source_address"`
	DestinationAddress string `json:"destination_address"`
	Amount             int64  `json:"amount"`
	UnsignedTx         string `json:"unsignedtx"`
	SignedTx           string `json:"signedtx"`
}
```

Things to particularly note here are as follow:

- We are not connected to the global bitcoin blockchain. Thus, we're creating our own UTX transaction which serves as input amount to our new transaction.
- This UTX transaction is hashed to create origin transaction.
- Using the origin transaction, sender's address and receiver's public address, bitcoin scripts are created.
- These scripts are added to a new redeeming transaction.
- A redeeming transaction can be only verified by receiver's private key.

To handle receiving transactions:

- A public key is generated. This key can be given to the sender to create a transaction.
- We don't have to do anything to receive the amount. Once the transaction is created and added to blockchain at sender's side, it is automatically owned by us.
- This receiving transaction can serve as UTXO for new sending transaction.

4.A.2.5 Frontend

The frontend details are explained in the implementation section.

4.A.2.6 Limitations

While we term the Raspberry Pi implementation as hardware wallet, it still relies on OS and many third-party libraries to accomplish its task. Furthermore, due to intricacies of the cryptocurrency language itself, many transaction details, implementation and components of an attractive and full-fledge UI were left out. They are addressed in the next section.

PART B:
MICROCONTROLLER IMPLEMENTATION

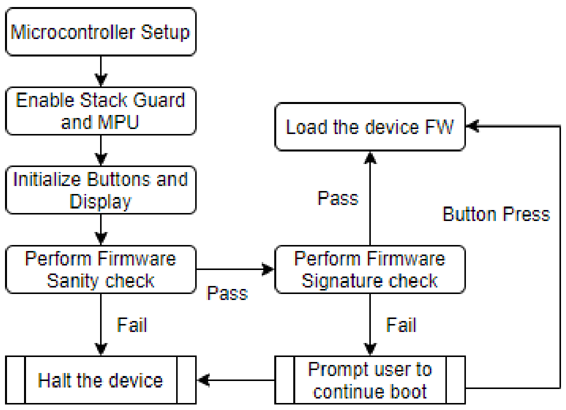
4.B.1 SYSTEM ARCHITECTURE

While the Raspberry Pi implementation does fulfill the basic wallet functionalities, it is cryptographically insecure and doesn't performs any kind of device genuineness attestation. Furthermore, it relies on Linux OS as base to function. To remove any OS dependency, reduce hardware footprint, and make device fundamentally more secure, we moved to a microcontroller implementation of the wallet. Details of the controller and hardware specifications are mentioned in the implementation section. This section aims to generally discuss the framework and flow of our wallet design.

4.B.1.2 The Bootloader Architecture

The bootloader is a vendor-proprietary code that runs on the device before offloading to an OS or a firmware. It is responsible for initializing the hardware and setting up the device for user space to run. In addition to this, a bootloader also offers device verification and recovery tools as desired. We'll try explaining the architecture without going into too much of the microcontroller specific details. The flowchart of bootloader is shown below:

Figure 7 Bootloader Program Basic Flowchart


Microcontroller Setup
Enable Stack Guard and MPU
Initialize Buttons and Display
Perform Firmware Sanity check
Pass
Fail
Halt the device
Perform Firmware Signature check
Pass
Fail
Load the device FW
Button Press
Prompt user to continue boot

This section won't do justice to explain the above components. We'll continue the explanation in the detailed design section.

4.B.2 Detailed System Design

We now move onto explaining the system architecture in great detail. This section doesn't offer explanation of core cryptographic routines and embedded architecture concepts. Please feel free to revisit them to better understand the following.

4.B.2.1 Bootloader In-Depth

The bootloader consists of a number of classes and platform specific implementations. It's written in pure bare-metal programming i.e. it doesn't rely on any underlying OS to run. We will now explain all components listed in the basic flowchart in Section 4.B.1.

Microcontroller Setup

The choice of microcontroller used is explained in the Implementation section. For this section, it is sufficient to note that our solution is based on **ARM Cortex M3/M4** architecture and **ARMv4** platform. To ease initializing basic device setup and layout implementations (pin mapping, ADCs, timers, GPIOs, etc.) we're using the open-source *OpenCM3* libraries. The basic setup involves:

1. Enable all GPIO pins and 120MHz clock
2. Setup GPIO A and B for OLED display and GPIO C for buttons (includes setting up the pins for input/output, pull-down and pull-up configurations respectively).
3. Enable serial interface.
4. Enable on-board Random Number Generation or RNG controller
5. Enable OTG File System support to accept USB connection.

Enable Stack Guard and Memory Protection Unit (MPU)

Cortex M3 series devices suffer from a basic stack attack vulnerability which requires hardware modifications to be eliminated. The attack involves spoiling the stack memory with malicious code until it overflows causing the device to error out in an undefined state. To mitigate this, a random 32-bit variable is stored inside the stack. If the function fails to stack the variable, since we are at initial device launch and we don't expect the stack to be filled, an error is thrown, and the device is immediately halted. Demonstrating stack smashing attack requires complex hardware and thus is not easy to be reproduced.

The Cortex M platform provides *an Options Register* that can be used to configure device. The Options Register is a 4-bytes register that allows to enable read and write protection to the flash memory. As per the platform documentation, writing the following bytes to the register enables MPU:

1. 0xFFFC to the upper 2 bytes to enable Write Protection for the bootloader
2. 0xCCFF to the lower 2 bytes to enable Read Protection Level 2.

F	F	F	C	C	C	F	F

When RDP level 2 is activated, all protections provided in Level 1 are active and the chip is fully protected. The RDP option byte and all other option bytes are frozen and can no longer be modified. The JTAG, SWV (single-wire viewer), ETM, and boundary scan are disabled. When booting from Flash memory, the memory content is accessible to user code. However, booting from SRAM or from system memory bootloader is no more possible. This protection is irreversible (JTAG fuse), so it's impossible to go back to protection levels 1 or 0. [32]

Both permissions disallow an adversary to read the firmware and bootloader code and make any unauthorized modifications to it. However, we've disabled this in our development builds to allow us in making modifications to the device. Note that enabling MPU, permanently locks device.

Initializing OLED Display

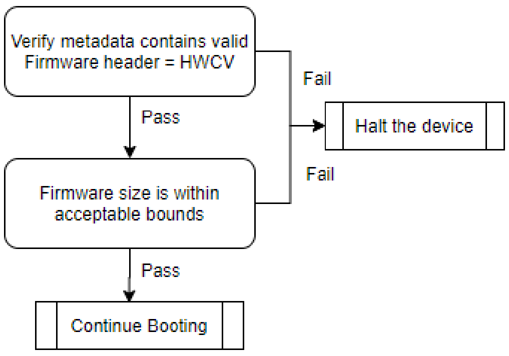
The OLED display libraries are taken from Open Source repositories owned by *Satoshi Labs – Pavol Rusnak*. They include setting up the display, initializing it and also contain code for displaying prompts with simple sliding animations. These libraries are a core part of Trezor project. The only modifications made to the library were GPIO pins layout to match our design.

Perform Firmware Sanity Check

This is first-stage firmware verification check. It verifies basic integrity of the firmware before performing the actual signature verification. This is helpful if the firmware running on the device is too old or too new and the bootloader doesn't support it or if the firmware was modified. The sanity checks however are not an absolute way of measuring firmware authenticity. They exist to save CPU cycles by skipping signature verification if a bogus firmware can be identified at a very early stage.

The firmware is split into two parts, metadata and the actual wallet implementation. The following checks are performed on the firmware memory region:

Figure 8 Firmware Sanity Checks


Verify metadata contains valid Firmware header = HWCV
Fail
Pass
Halt the device
Fail
Firmware size is within acceptable bounds
Pass
Continue Booting

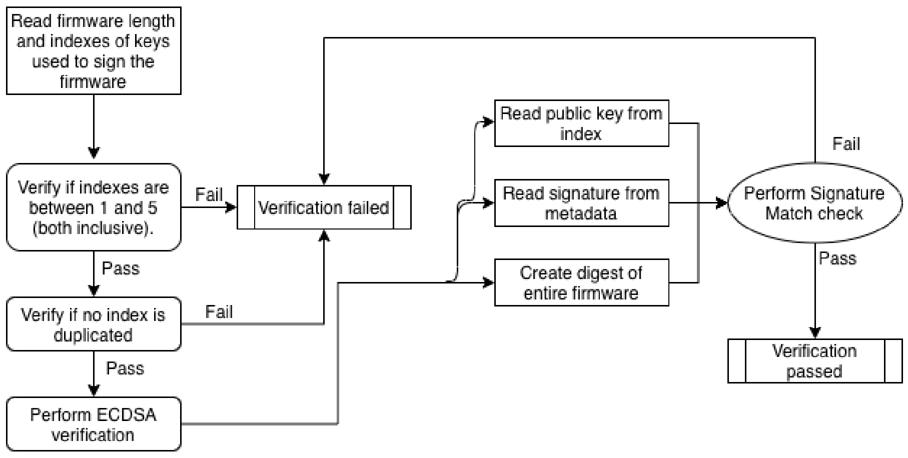
1. The first check simply verifies if the firmware header (also called magic) is HWCV.
2. The second check ensures that the firmware installed on the device is not less than 4 KiB and not greater than the total flash memory allocated to it. These calculations are based on memory maps which is outlined in implementation section.
3. At least one of these checks should fail to halt the device. All tests have to pass to continue the device to boot.

Perform Firmware Signature Check

The firmware is cryptographically signed using ECDSA algorithm. The details of the algorithm itself and how the firmware is actually signed is not necessary at this stage.

The firmware is signed using three private keys which are owned by us (randomly chosen out of five keys). The public keys required to verify are hardcoded inside the bootloader. The metadata inside the firmware holds indexes of the three private keys that were used to sign the firmware. They are used to index the respective public keys. The following flow diagram explains the verification process:

Figure 9 Firmware Signature Verification



Read firmware length and indexes of keys used to sign the firmware
Verify if indexes are between 1 and 5 (both inclusive).
Fail
Verification failed
Pass
Verify if no index is duplicated
Fail
Pass
Perform ECDSA verification
Read public key from index
Read signature from metadata
Create digest of entire firmware
Perform Signature Match check
Fail
Pass
Verification passed

Load the device firmware

If all previous checks have completed, the device continues to load the firmware. On Cortex M architectures, this involves setting the *VTOR* pointer to point at the start of memory space corresponding to the firmware. The actual de-loading of bootloader and loading of firmware is handled by the microcontroller itself.

4.B.2.2 Firmware

The general architecture of the firmware resembles our Raspberry Pi implementation as explained in Part A of this chapter. As noted previously, re-implementing the bitcoin protocol, cryptographic and transaction creation and signing routine was not the main focus of the project. The end goal was to create a more secure repository for our private keys tied to cryptocurrencies. Furthermore, coding one from scratch especially for a microcontroller is a painless process. To ease our development, we used *Trezor One Firmware* as the base and made adjustments/improvements on top of it.

Chapter 5

IMPLEMENTATION DETAILS

5.1 HARDWARE WALLET BASED ON RASPBERRY PI

5.1.1 Hardware Requirements

The original hardware planned was Raspberry Pi Zero. However, due to unavailability in lockdown, we switched over to Pi 3. The implementation details are unchanged and don't depend on Pi model used.

Raspberry Pi 3 Model B+ (1GB) was used for the firmware implementation. It comes preloaded with the following hardware specifications: [33]

- SoC: Broadcom BCM2837B0 Quad-core A53 (ARMv8) 64-bit @1.4GHz
- GPU: Broadcom Videocore-IV
- RAM: 1GB LPDDR2 SDRAM
- Networking: Gigabit Ethernet (USB channel), 2.4GHz and 5GHz 802.11 b/g/n Wi-Fi
- Storage: Micro-SD (8 GB)

The inbuilt networking capability allows the wallet to behave as cloud wallet if accessed over the internet and as a hardware wallet if the ethernet via USB channel is used.

The Operating System used was *Arch Linux (bleeding edge)* distribution. It's a Linux GNU based OS. This OS was our preferred choice due to its extremely small footprint, measured at approximately 800 MB installation size and 80 MB memory consumption, leaving sufficient resources for our wallet and hosting service to function properly.

Since, we rely on OS for this wallet version, the underlying hardware dependency isn't mandatory. The wallet can function properly on any Raspberry Pi (even the cheap Zero variant), any laptop or desktop computer. While the compiled binaries can only run on ARMv8 architecture, the source code can be compiled for any supported architecture.

5.1.2 Firmware Language and Model

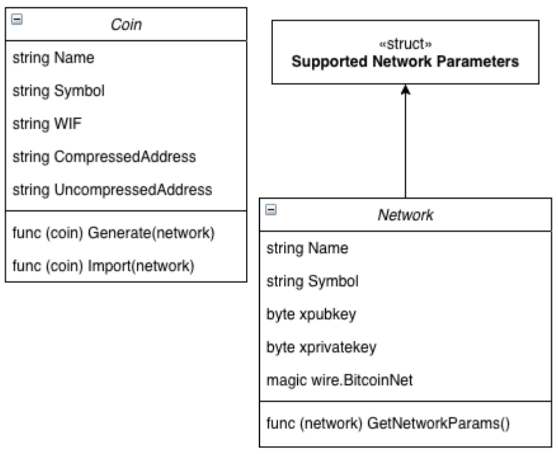
The entire firmware was written in Golang (or Go Language). Go is a platform independent language and can be easily cross compiled to multiple architectures. All of the main wallet functionality i.e. code that performs the actual stuff is written in Go. The following open-source libraries were used to make implementation easier:

- *go.rice [github.com/GeertJohan/go.rice]* Allows bundling the firmware and frontend together in one single executable package.
- *handlers [github.com/gorilla/handlers]* Handles cross origin resource sharing. This is extremely useful when browser actively rejects connections from resources not whitelisted in the remote origin resource file.
- *mux [github.com/gorilla/mux]* Acts as the connector API for serving requests from server to the wallet package.
- *btcutil and btcd [github.com/btcsuite]* Contains various cryptographic routines implementations that are core to the bitcoin specification.

Coins and Network

The following class diagrams show how information about wallets is represented in our design:

Figure 10 Main Wallet Classes

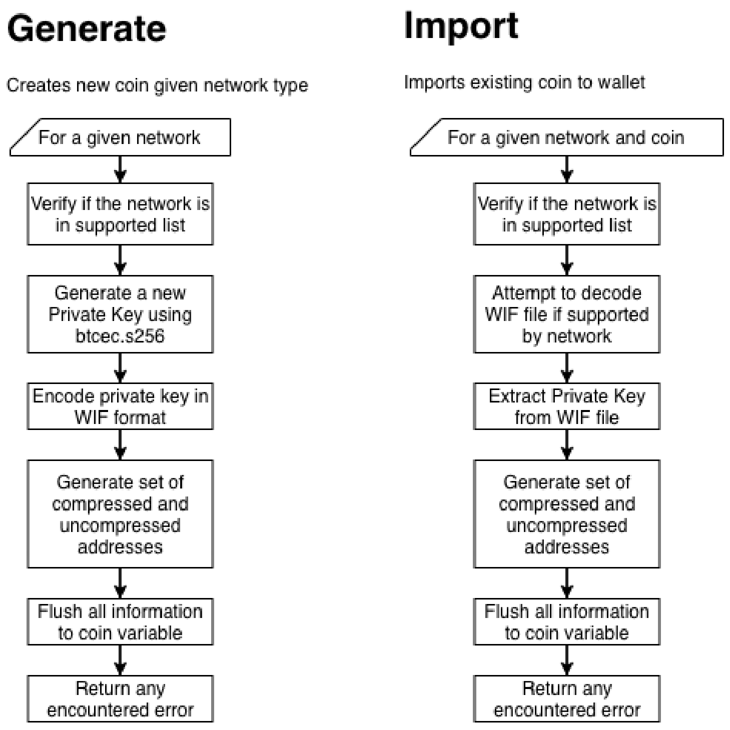


For a given coin, we require the name, symbol and private key encoded in WIF format. The compressed and uncompressed addresses are public keys generated using the private key and the network parameters stored in the structure. Both can be used as addresses for receiving transactions.

Most cryptocurrency networks append a special prefix to the private and public addresses. They are stored as bytes in the network structure. The magic parameter contains implementation specific information about the specific cryptocurrency network such as the curve points used to calculate addresses in the ECDSA scheme, signing methodology, etc. The supported networks are Bitcoin, Litecoin and Dash. Their parameters are listed in Design section.

Algorithm for *Generate* and *Import* functions is shown below:

Figure 11 Generate and Import Algorithms

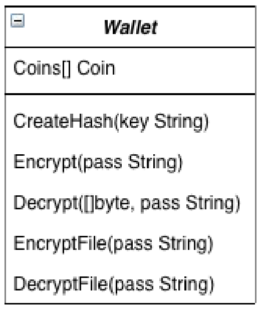

Generate
Creates new coin given network type
For a given network
Verify if the network is in supported list
Generate a new Private Key using btcec.s256
Encode private key in WIF format
Generate set of compressed and uncompressed addresses
Flush all information to coin variable
Return any encountered error
Import
Imports existing coin to wallet
For a given network and coin
Verify if the network is in supported list
Attempt to decode WIF file if supported by network
Extract Private Key from WIF file
Generate set of compressed and uncompressed addresses
Flush all information to coin variable
Return any encountered error

The *GetNetworkParameters* is simply a getter function that flushes network details to the variable.

Wallet Data Encryption and Decryption

A wallet structure stores all coin's information. However, to ensure that the wallet data can't be decoded to extract private keys if leaked, the whole node is encrypted using AES ciphers.

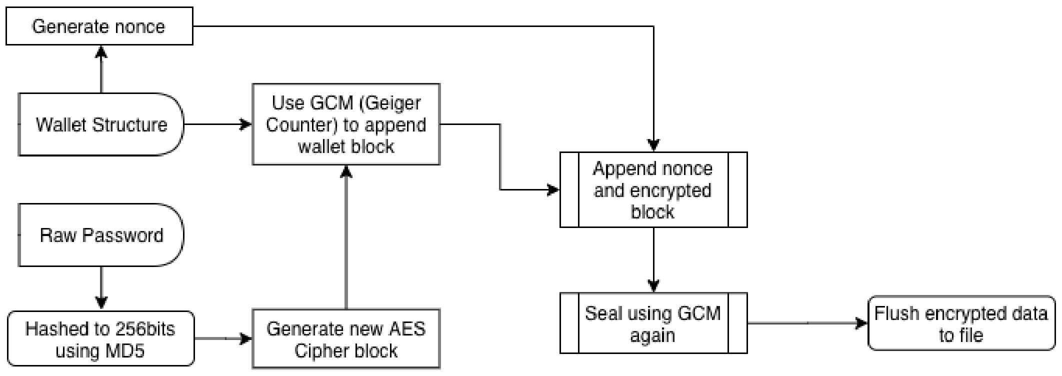
Figure 12 Wallet Structure



The Encrypt and Decrypt functions operate on the *Wallet* structure flushed in the memory while the *File* helper functions read and write wallet data from disk.

In AES encryption, the passphrase to encrypt must be of specific size (256-bits). However, to avoid imposing any password size restriction on user's end, the provided key is hashed to 256-bits using MD5 hashing algorithm. This hashed passphrase cannot be converted back to original string that was used to encrypt and decrypt thus adding more security to the overall implementation. The encryption routine algorithm is shown below. The decryption routine is simply the reverse of it.

Figure 13 Encryption Routine


Generate nonce
Wallet Structure
Use GCM (Geiger Counter) to append wallet block
Append nonce and encrypted block
Raw Password
Hashed to 256bits using MD5
Generate new AES Cipher block
Seal using GCM again
Flush encrypted data to file

The flushed data is serialized using *JSON Marshal* before writing to disk to ensure that the wallet structure is platform independent and can be recovered easily.

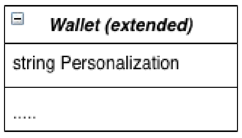
Wallet Administrative Functions

To further extend functionality, attractive helper functions are implemented that operate directly on the wallet structure to add new coins, create new wallet, dump wallet, etc. Their details are trivial and are not mentioned here.

Evil Maid Protection

A trivial example of attack is hardware spoofing. To avoid the risk of entering passphrase on a device from malicious agent, personalization is embedded inside the wallet structure. The default wallet structure is extended to add one extra variable:

Figure 14 Extended Wallet Structure for Evil Maid



This personalization message is shown every time after a successful user login. Furthermore, a secondary PIN is also added to device. The second PIN acts as ‘deceiving PIN’. Entering this PIN instead of the main wallet master PIN completely destroys all wallet data. The implementation and structure are similar to the wallet structure, except that the structure contains a static string that holds no value instead of storing coins data.

We can also exploit this functionality to add support for multiple wallets within a single hardware wallet, but this is not implemented.

Signing Transactions

While most of the bitcoin specification is included in the implementation, the following functionality is missing:

- Calculation of transaction fees. This wasn’t added since we were not broadcasting the transaction anywhere
- User has to manually add unspent transaction hash since we are not connecting to any Bitcoin public API.
- The final signed transaction is returned to the user to be manually broadcasted to Bitcoin P2P network.

Also note that as of the writing of this project, buying and selling of cryptocurrencies is prohibited by law in Pakistan by State Bank. Thus, most popular and credible public exchanges (like Coinbase) don’t offer cryptocurrency wallet services in Pakistan due to legal restrictions.

Anyways, our transaction structure and algorithm are shown below:

Figure 15 Transaction Structure

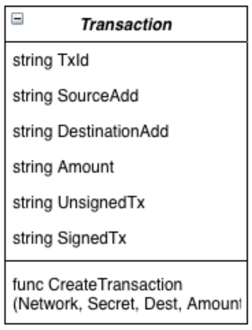
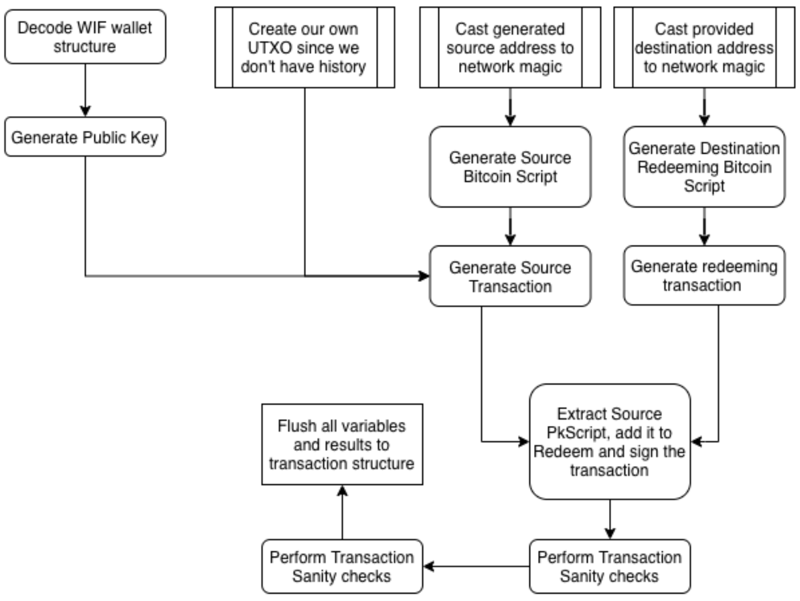


Figure 16 Transaction Generation and Signing



Decode WIF wallet structure
Generate Public Key
Create our own UTXO since we don't have history
Cast generated source address to network magic
Cast provided destination address to network magic
Generate Source Bitcoin Script
Generate Destination Redeeming Bitcoin Script
Generate Source Transaction
Generate redeeming transaction
Extract Source PkScript, add it to Redeem and sign the transaction
Flush all variables and results to transaction structure
Perform Transaction Sanity checks
Perform Transaction Sanity checks

5.1.3 API Endpoints with Gorilla Mux

The API builds up easily accessible RESTful endpoints to allow communication between the management portal and wallet interface served with the prefix */api*.

/api/wallet

This endpoint serves the following functionalities:

- *CreateWallet:Method(Post)* A key must be provided in the request query. This key is used to create a new wallet. This is encoded in json and sent as response in the HTTP request.
- *DestroyWallet:Method(Delete)* Destroys the entire wallet. This step is non-reversible and permanently deletes all keys. This is added only for debugging purposes and should be removed in production builds.
- *DumpWallet:Methoid(Get)* Dumps the entire encoded wallet (including coins data) as a downloadable file in plain-text. This file should be either immediately deleted or stored in a private vault since it contains highly sensitive information. This is not recommended in production builds.

/api/create-coin

This endpoint serves a single *CreateCoin:Method(Post)* request that takes coin symbol and the key to create a new coin and add it to the wallet structure. Nothing too fancy.

/api/import-coin

Similar to create-coin endpoint, if the user already has a private key, that's used to create the new coin structure and is included in the wallet file. This is served through *ImportCoin:Method(Post)* request.

/api/authenticate

Checks for authenticity of the key provided by verifying if it encrypts the wallet file stored on disk. Further functionality can be added here to prevent brute-force attacks by adding rate limiting features i.e. the endpoint doesn't accept keys for a definite amount of time. This functionality is served by *AuthenticateWallet:Method(Post)* request.

/api/addresses

Serves a single *GetAddresses:Method(Get)* request that returns all coin information stripped off private WIF after a successful authenticate request.

/api/export

Similar to dump wallet request, this also returns the entire wallet structure but in encoded (encrypted) form. This way even if an adversary is able to extract your wallet, it is still encrypted rendering his attempts useless. This functionality is served by *ExportWallet:Method(Get)* request.

5.1.4 Angular Management Interface

While we can interact with RESTful API to fulfil all of the wallet functionalities, a nice frontend and user-friendly interface makes the final solution more accessible and attractive. The technologies used to develop the end user interface are:

- HTML/CSS for document layout and customization
- JavaScript/Angular for user aware and responsive interface.
- Bootstrap for a barebone skeleton to rapidly deploy our app. It eliminates coding the website from scratch by providing a highly usable template with decent documentation.

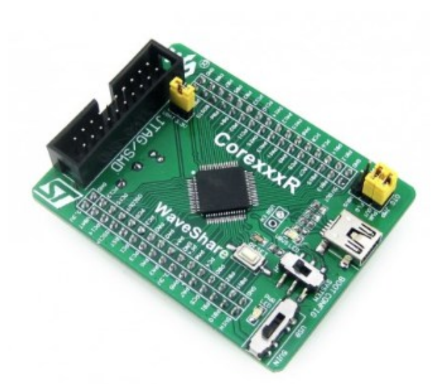
Screenshots and mock diagrams of interface are shown in the results section of the report.

5.2 Microcontroller Implementation

5.2.1 Development Board

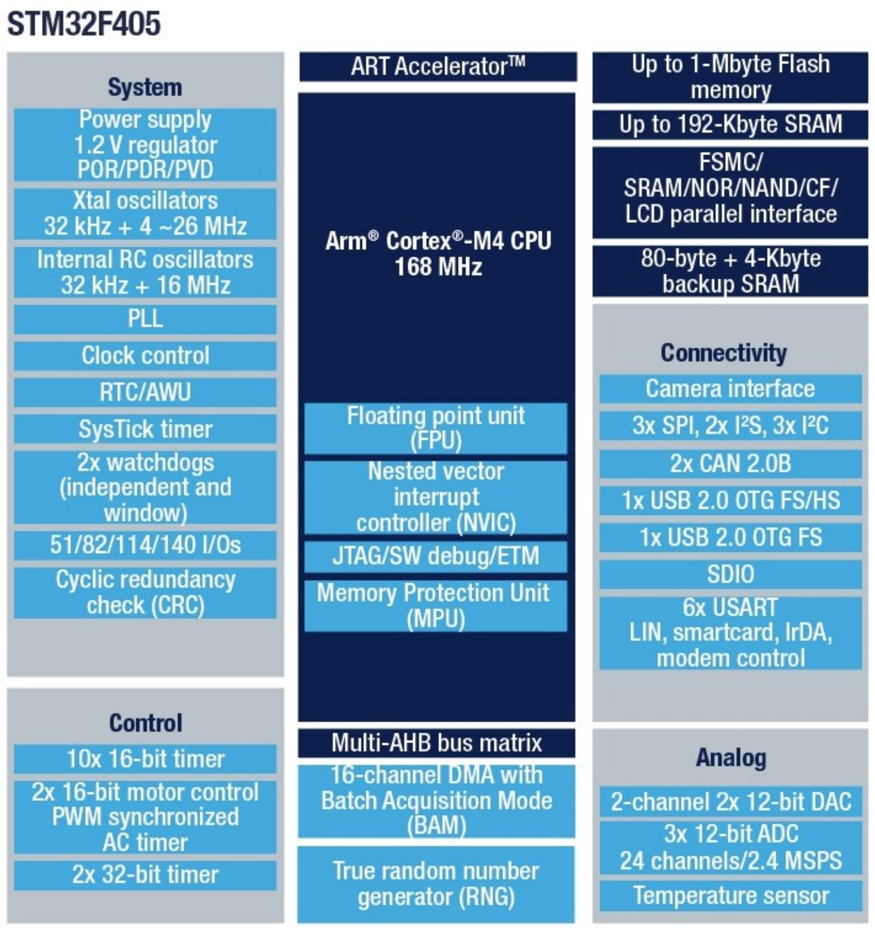
For rapid prototyping and development, the initial bootloader solution was built on *Waveshare Core405R Board*. It's MCU board designed for STM32F405RGT6 (Cortex M4), and comes with pre-integrated clock circuit, USB control and connector, JTAG/SWD debugging interfaces and a header switch to switch between different boot modes.

Figure 17 Core405R Development Board



STM32F405RGT6 is ARM 32-bit Cortex-M4 CPU clocked at 168 MHz and comes with 1 Mbyte of Flash memory. It also comes equipped with LCD interface, USB and Ethernet support, a number of IO and communication ports and a hardware RNG that is used in most cryptographic primitives. It has a basic MPU support to read and write restrictions on any portion of the flash memory. [34]

Figure 18 STM32F405 Circuit Diagram


STM32F405
System
Power supply 1.2 V regulator POR/PDR/PVD
Xtal oscillators 32 kHz + 4 ~26 MHz
Internal RC oscillators 32 kHz + 16 MHz
PLL
Clock control
RTC/AWU
SysTick timer
2x watchdogs (independent and window)
51/82/114/140 I/Os
Cyclic redundancy check (CRC)
Control
10x 16-bit timer
2x 16-bit motor control PWM synchronized AC timer
2x 32-bit timer
ART Accelerator™
Arm® Cortex®-M4 CPU 168 MHz
Floating point unit (FPU)
Nested vector interrupt controller (NVIC)
JTAG/SW debug/ETM
Memory Protection Unit (MPU)
Multi-AHB bus matrix
16-channel DMA with Batch Acquisition Mode (BAM)
True random number generator (RNG)
Up to 1-Mbyte Flash memory
Up to 192-Kbyte SRAM
FSMC/ SRAM/NOR/NAND/CF/ LCD parallel interface
80-byte + 4-Kbyte backup SRAM
Connectivity
Camera interface
3x SPI, 2x I²S, 3x I²C
2x CAN 2.0B
1x USB 2.0 OTG FS/HS
1x USB 2.0 OTG FS
SDIO
6x USART LIN, smartcard, IrDA, modem control
Analog
2-channel 2x 12-bit DAC
3x 12-bit ADC 24 channels/2.4 MSPS
Temperature sensor

5.2.2 Extra Components

The development kit consists of the following components enabling user to write and develop on to the development board and interact with the wallet:

- 1x ST-Link V2 STM32 Debug Adapter
- 1x SSD1306 OLED Display Module
- 2x Momentary Switches
- 1x Breadboard
- 1x USB Cable Type A to Type Mini-B (included with kit)
- Female to Male jumper cables
- Extra copper wires

5.2.3 Printed Wiring Board

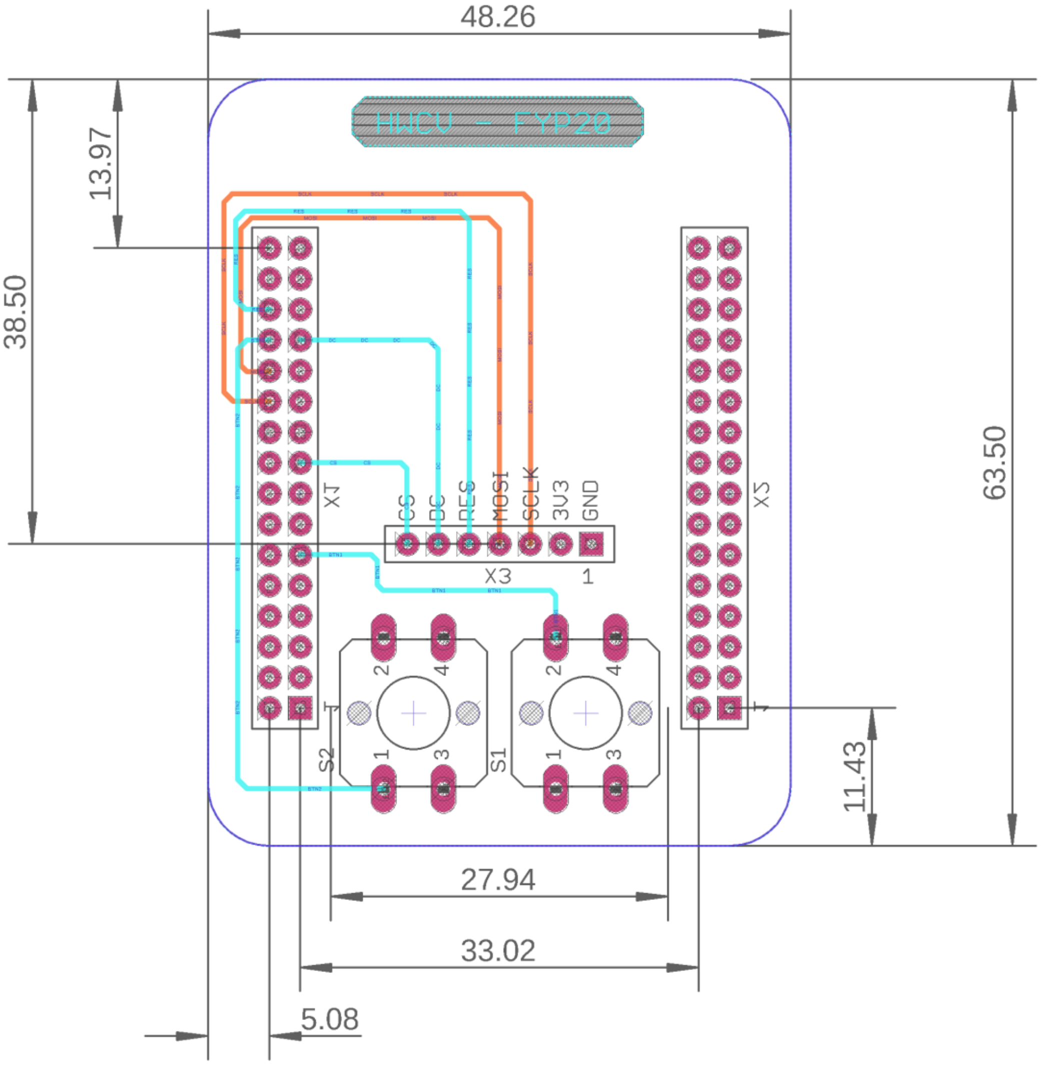
For more easier accessibility to the device and remove breadboard and wires from the design, a PWB version of the development kit was also designed. While it needs the same components as mentioned above, additional Female to Male sockets are required to plug in the LCD kit to IO pins exposed on the Core405R board.

The PWB was designed on Eagle, based on a single layer PCB with copper engravings on top and bottom for informational markings.

However, due to the COVID outbreak, our PWB design was never realized. Still, we believe that the design adds more versatility to the development kit and allows developers to easily test our bootloader model for security and vulnerability analysis without going into the hassle of wiring the components on a breadboard.

The front and bottom designs are shown in the figure below:

Figure 19 Development Kit PWB Design



- Middle Ports: for 7-pin OLED Display Module
- Side Ports: Core405R mounts
- Bottom Ports: Connectors for 2x Momentary Switches

The following table summarizes the wiring information (for both the PWB and breadboard).

Figure 20 Wiring Layout Information

Core 405R Pin	**Breadboard Pin**	**Name / Comment**
PC5	Button 2	
PA4	CS	Chip Select
PB0	DC	Data Command

PB1	RES	Reset
PA7	SDA	MOSI
PA5	SCK	SCLK
3.3V	VDD	3V3
GND	GND	Ground
PC2	Button 1	

5.2.4 Bootloader

The bootloader is completely written in Embedded C, resembling bare-metal programming style since no platform or operating system is used as a framework. The following opensource libraries were used to aid in development:

- *libopencm3 http://libopencm3.org:* Firmware Library for various Cortex-M microcontrollers.
- *micropython* *http://www.micropython.org:* Python 3.x implementation for microcontrollers and small embedded systems
- *NanoPB* *https://jpa.kapsi.fi/nanopb/:* Protocol Buffers implementation for microcontrollers. This is used to serialize header magic structures and store them in the flash memory.
- *libsecp256k1:* The core of the project. This is an optimized C library for EC operations on curve secp256k1 useful for ECDSA signing/verification, key generation, serialization and better randomization routines. It should be noted here that randomization itself is a highly researched topic since getting better randomization guarantees security of most cryptographic primitives.

Information on how to obtain the source code is in Appendix I section of the report.

5.2.5 Trezor Firmware

The firmware running on the microcontroller is based on Trezor One firmware (1.0.0). There were several reasons to choose this firmware:

- It includes all of the bitcoin transaction signing, keys creation routine.

- Firmware is based on Cortex M3, allowing it to be easily ported over to Cortex M4 architecture.
- It also includes a handy recovery mode to flash signed firmware on-the-go without touching the debugging interface.

The following modifications were made to the source code:

- Removed redundant cryptocurrencies support. Only bitcoin-based currencies are supported.
- Removed firmware flashing logic since we're using our own bootloader instead of proprietary one shipped with Trezor hardware.
- Remove all signing information and re-sign it using our private keys to pass bootloader authenticity checks.
- Removed flash locks and memory protection routines.
- Enable debugging links at various points in the firmware.

The following enhancements were made to the firmware:

- Add support for our bootloader authenticity checks by hardcoding the magic keywords and signatures to the metadata firmware region.
- Enhanced protection against evil made attacks by adding a non-reversible device personalization message embedded directly in the memory region. It is preserved during firmware flashes.
- Add a secondary, device specific 'destroy' PIN that completely wipes the entire storage region. This is useful if an adversary threatens the user and demands PIN for the wallet.
- Our open deployment model and development kit allows developers and hardware enthusiasts to build and test Trezor One implementations without actually buying one of their proprietary devices.
- Encrypt the entire storage region using AES ciphers. Reworked Trezor implementation to decrypt all nodes before extracting private key information for signing and verifying transactions.

5.2.6 Memory Map

The Flash memory region is distributed as follows:

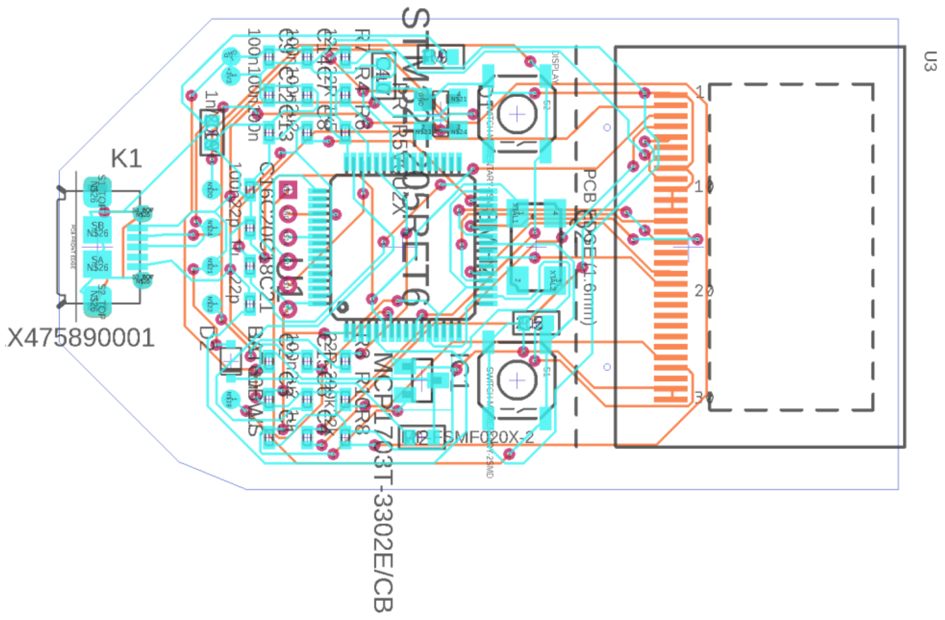
```
   name    |          range           |  size   |     function
-----------+-------------------------+---------+------------------
 Sector  0 | 0x08000000 - 0x08003FFF |  16 KiB | bootloader
 Sector  1 | 0x08004000 - 0x08007FFF |  16 KiB | bootloader
-----------+-------------------------+---------+------------------
 Sector  2 | 0x08008000 - 0x0800BFFF |  16 KiB | metadata area
 Sector  3 | 0x0800C000 - 0x0800FFFF |  16 KiB | metadata area
-----------+-------------------------+---------+------------------
 Sector  4 | 0x08010000 - 0x0801FFFF |  64 KiB | firmware
 Sector  5 | 0x08020000 - 0x0803FFFF | 128 KiB | firmware
 Sector  6 | 0x08040000 - 0x0805FFFF | 128 KiB | firmware
 Sector  7 | 0x08060000 - 0x0807FFFF | 128 KiB | firmware
===========+=========================+============================
 Sector  8 | 0x08080000 - 0x0809FFFF | 128 KiB | N/A
 Sector  9 | 0x080A0000 - 0x080BFFFF | 128 KiB | N/A
 Sector 10 | 0x080C0000 - 0x080DFFFF | 128 KiB | N/A
 Sector 11 | 0x080E0000 - 0x080FFFFF | 128 KiB | N/A

 metadata area:

 offset | type/length |  description
--------+-------------+-------------------------------
 0x0000 |  4 bytes    |  magic = 'HWCV'
 0x0004 |  uint32     |  length of the code
 0x0008 |  uint8      |  signature index #1
 0x0009 |  uint8      |  signature index #2
 0x000A |  uint8      |  signature index #3
 0x000B |  uint8      |  flags
 0x000C |  52 bytes   |  reserved
 0x0040 |  64 bytes   |  signature #1
 0x0080 |  64 bytes   |  signature #2
 0x00C0 |  64 bytes   |  signature #3
 0x0100 |  32K-256 B  |  persistent storage
```

5.3 END PRODUCT SCHEMATICS

The end goal of the project was to realize our solution in a compact user-friendly device. However, due to the outbreak and lockdown, we only limited the design to software 3D prints and circuit schematics (scope reduction).

The PCB schematic of the end device is shown below. It's based on the same MPU and OLED module as the development kit, just compressed into a smaller form factor.

Figure 21 Device PCB Schematic



5.3.2 Components

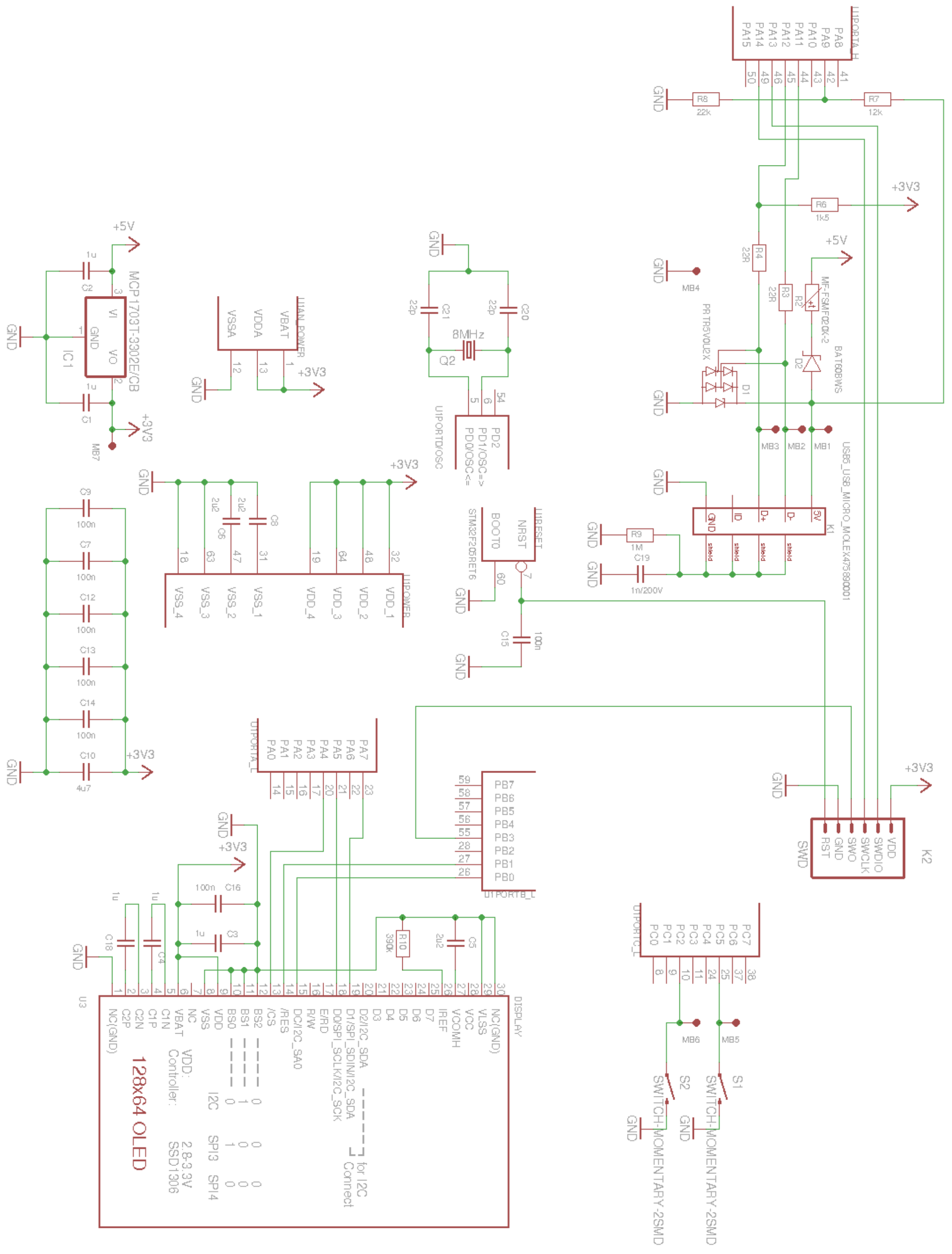
**included in a text file with the PCB schematic*

- 5x SMD Capacitor 1uF
- 7x SMD Capacitor 0.1uF
- 3x SMD Capacitor 2.2uF
- 1x SMD Capacitor 4.7uF
- 2x SMD Capacitor 22pF (all 0603)
- 2x Micro-buttons
- 1x Micro USB Type B Receptacle
- 1x OLED Display 128x64

- 1x STM32F405RE
- 1x MCP1703 Low Dropout Voltage Regulator
- 1x 8MHz external Crystal

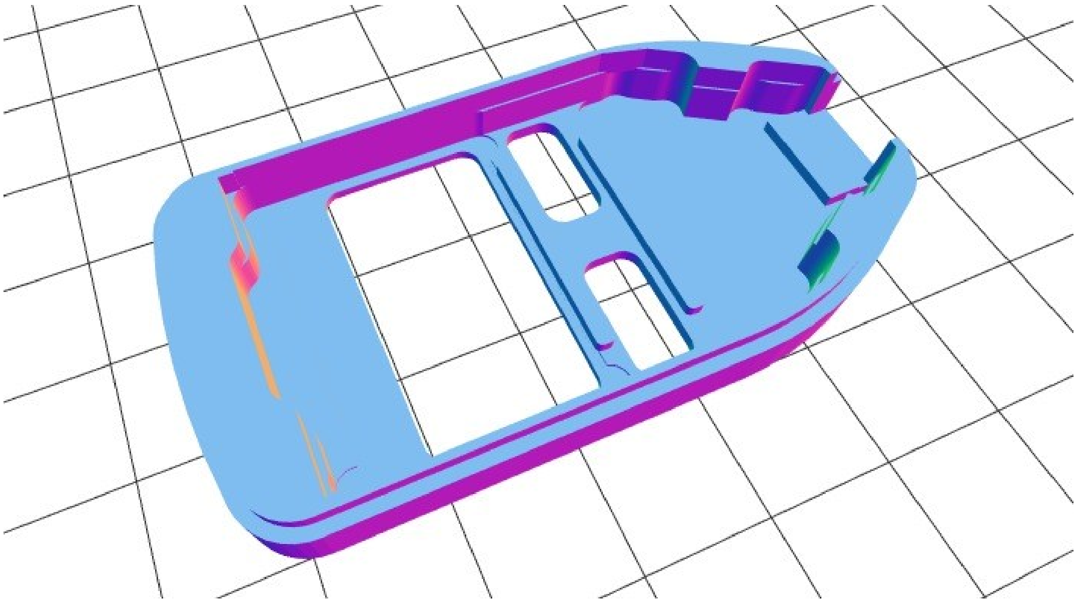
5.3.3 Circuit Diagram

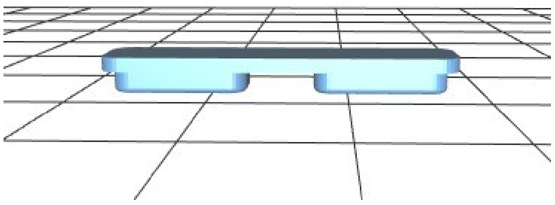
Figure 22 Device Circuit Diagram

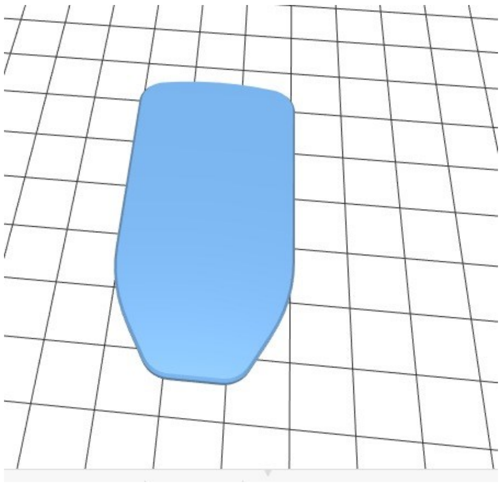


5.3.4 3D Case Design

Figure 23 : Bottom case (Top), Buttons (Left), Shield (Right) [adapted from Trezor]







Chapter 6

RESULTS DISCUSSION

This section includes the storage structures and end-user software interface screenshots. Technical aspects related to these results are already explained in the previous chapters.

6.1 FRONTEND INTERFACE GUIDE

The wallet service is automatically started on Raspberry Pi using *Linux's Systemd Service*. The user only needs to plug in the device to the computer using the USB port or connect it to Wi-Fi, the latter is not recommended since the device is supposed to work in offline mode for greater security. Wi-Fi interface is discussed in later sections.

Emulating Ethernet over USB

Raspbian OS was modified to enable Ethernet connectivity over USB port, which is also used for charging. To do this, add the following line in `config.txt` in root partition of OS:

```
dtoverlay=dwc2
```

Also include this line in `cmdline.txt` to change boot parameters to load emulation drivers:

```
modules-load=dwc2,g_ether
```

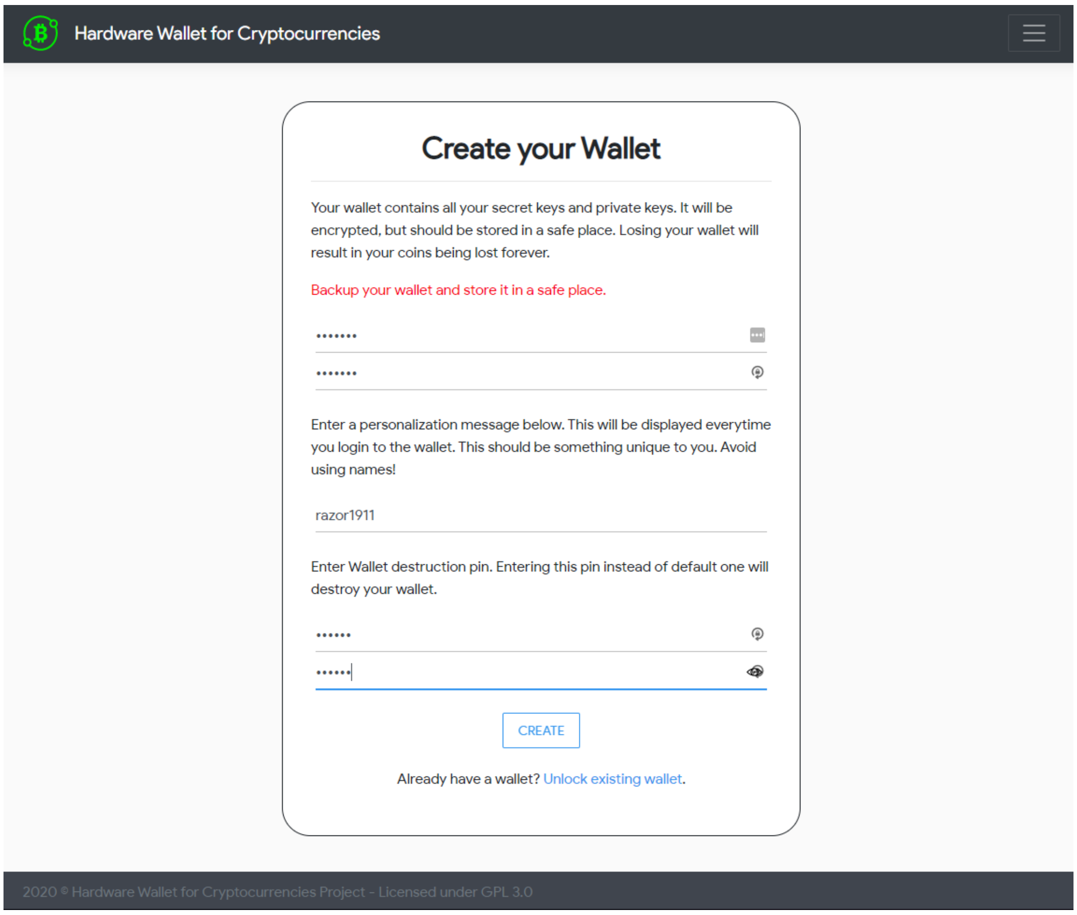
Once connected, wait for the device to boot. By default, the hostname is `raspberry.local`. You can enter this as domain in any web browser followed by `:6969 (port number)` to access the web interface.

Creating a new Wallet

To create new wallet user needs to enter a wallet PIN that is hashed to encrypt the wallet structure using AES256-GCM.

A personalization message is also provided to make the device *specific to a user* and prevent device spoof or Evil Maid attacks. Lastly a destruction PIN is also required that destroys the wallet in case the user needs to setup the device again or needs to destroy it in case of emergency purposes. There's no other interface to wipe the device other than entering destruction PIN to ensure that user funds can't be lost.

Figure 24 Wallet Creation Screen



Hardware Wallet for Cryptocurrencies

Create your Wallet

Your wallet contains all your secret keys and private keys. It will be encrypted, but should be stored in a safe place. Losing your wallet will result in your coins being lost forever.

Backup your wallet and store it in a safe place.

•••••••

•••••••

Enter a personalization message below. This will be displayed everytime you login to the wallet. This should be something unique to you. Avoid using names!

razor1911

Enter Wallet destruction pin. Entering this pin instead of default one will destroy your wallet.

••••••

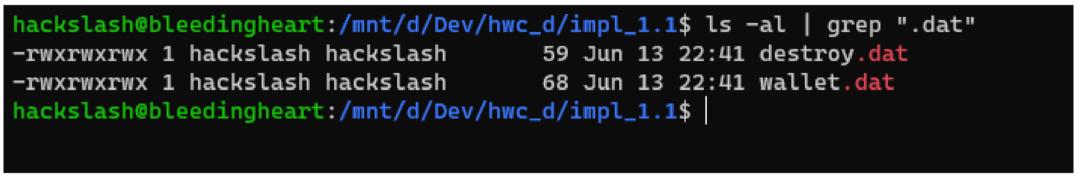
••••••

Already have a wallet? Unlock existing wallet.

2020 © Hardware Wallet for Cryptocurrencies Project - Licensed under GPL 3.0

Once the wallet is created, the following changes are made to the Pi's filesystem:

Figure 25 Filesystem after Wallet Creation

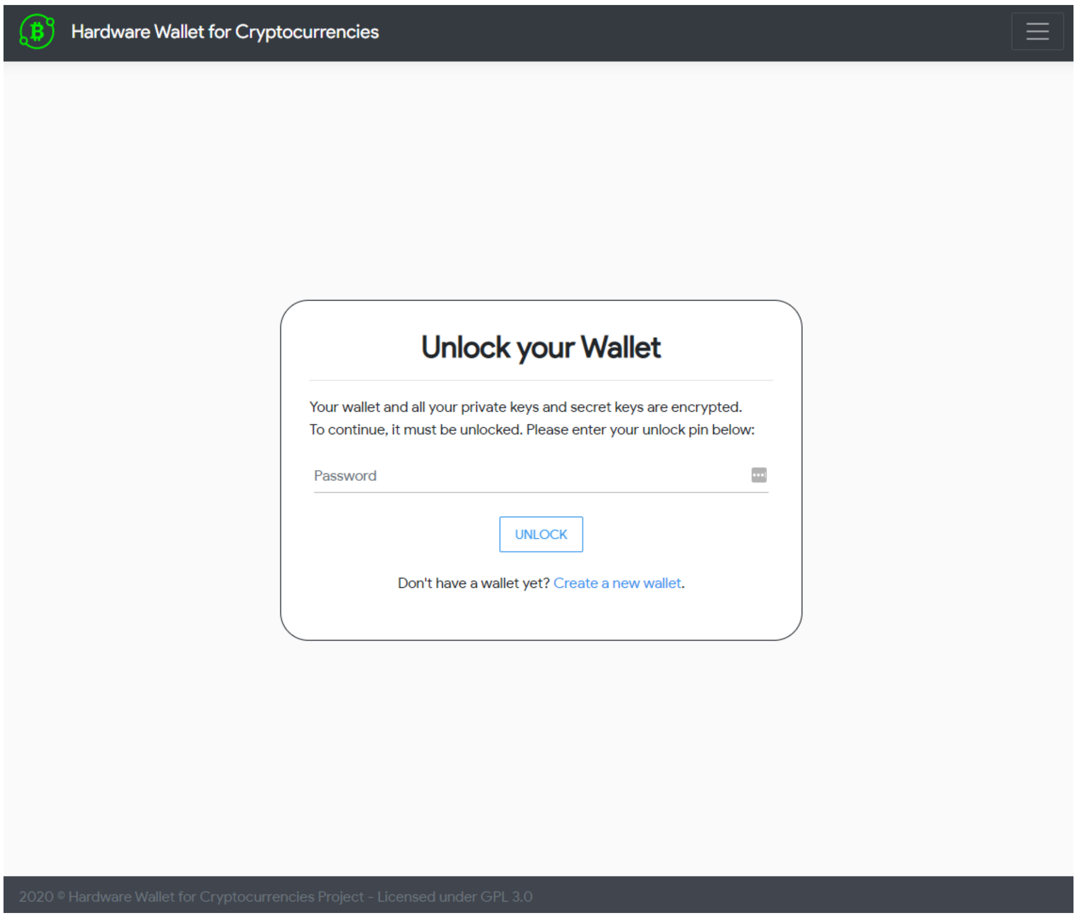


```
hackslash@bleedingheart:/mnt/d/Dev/hwc_d/impl_1.1$ ls -al | grep ".dat"
-rwxrwxrwx 1 hackslash hackslash        59 Jun 13 22:41 destroy.dat
-rwxrwxrwx 1 hackslash hackslash        68 Jun 13 22:41 wallet.dat
hackslash@bleedingheart:/mnt/d/Dev/hwc_d/impl_1.1$
```

Unlocking the Wallet

Entering the correct PIN unlocks the device. A wrong PIN logs the error in *Chromium Browser → Developer Mode → Console Logs*.

Figure 26 Wallet Unlock Screen



Hardware Wallet for Cryptocurrencies

Unlock your Wallet

Your wallet and all your private keys and secret keys are encrypted.
To continue, it must be unlocked. Please enter your unlock pin below:

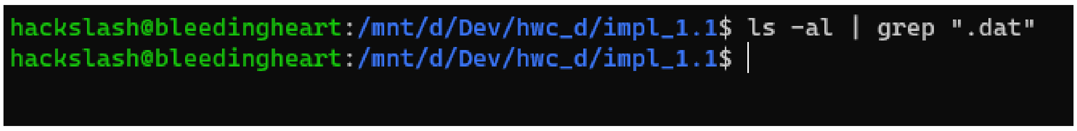
Don't have a wallet yet? Create a new wallet.

2020 © Hardware Wallet for Cryptocurrencies Project - Licensed under GPL 3.0

Destroying the Wallet

Entering the *Destroy PIN* instead of the unlock PIN wipes all wallet data stored on Pi.

Figure 27 Filesystem after Wallet Creation

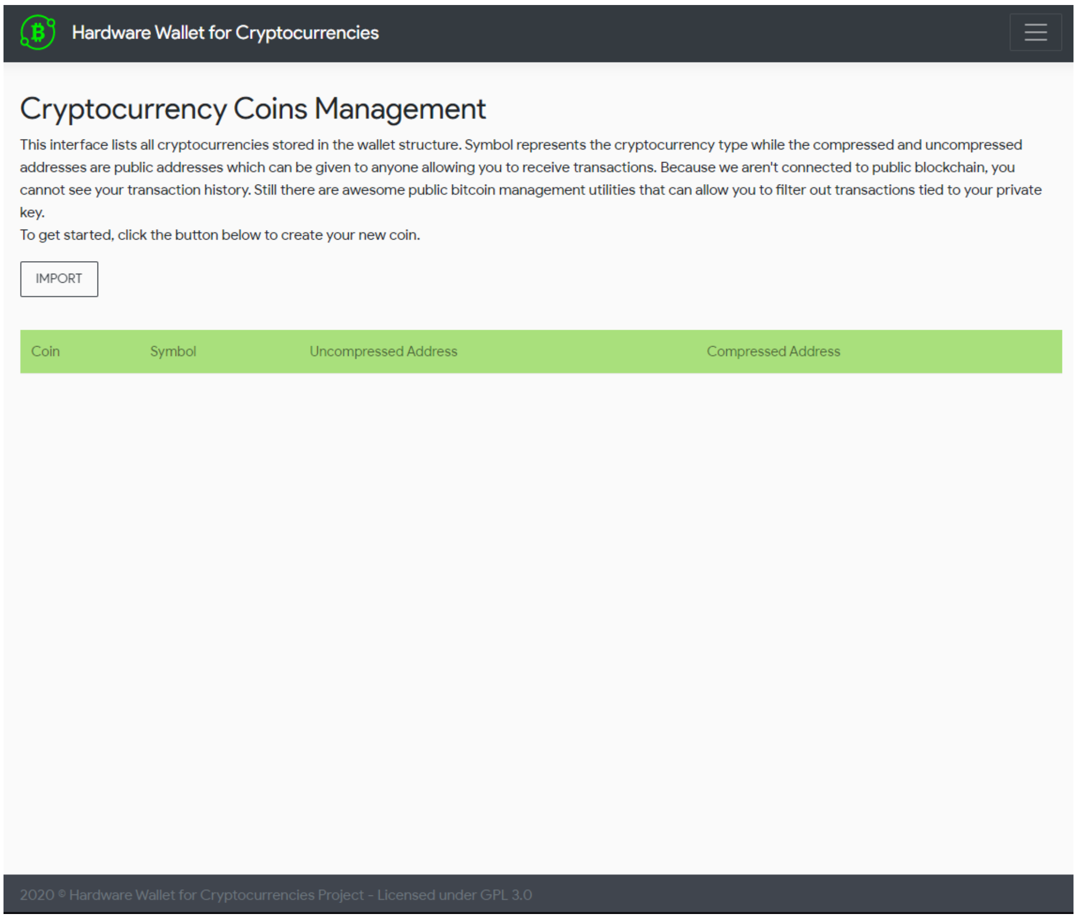


```
hackslash@bleedingheart:/mnt/d/Dev/hwc_d/impl_1.1$ ls -al | grep ".dat"
hackslash@bleedingheart:/mnt/d/Dev/hwc_d/impl_1.1$
```

Main Dashboard

After successful unlock, the main user dashboard is displayed. User can add new coins and create new transactions. Special *Dev Tools* are also provided which allows dumping the entire wallet in encrypted and decrypted form. It also shows the *Personalization Message* stored inside the wallet structure to verify device ownership.

Figure 28 Wallet Dashboard Screen



Hardware Wallet for Cryptocurrencies

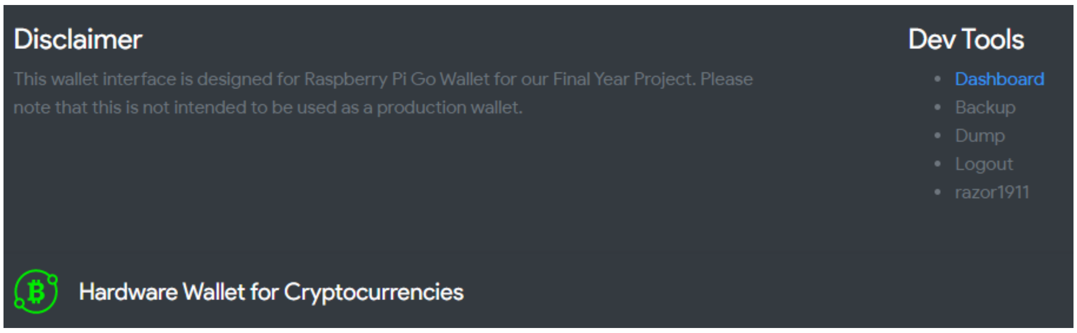
Cryptocurrency Coins Management

This interface lists all cryptocurrencies stored in the wallet structure. Symbol represents the cryptocurrency type while the compressed and uncompressed addresses are public addresses which can be given to anyone allowing you to receive transactions. Because we aren't connected to public blockchain, you cannot see your transaction history. Still there are awesome public bitcoin management utilities that can allow you to filter out transactions tied to your private key.
To get started, click the button below to create your new coin.

Coin	Symbol	Uncompressed Address	Compressed Address

2020 © Hardware Wallet for Cryptocurrencies Project - Licensed under GPL 3.0

Figure 29 Dev Tools



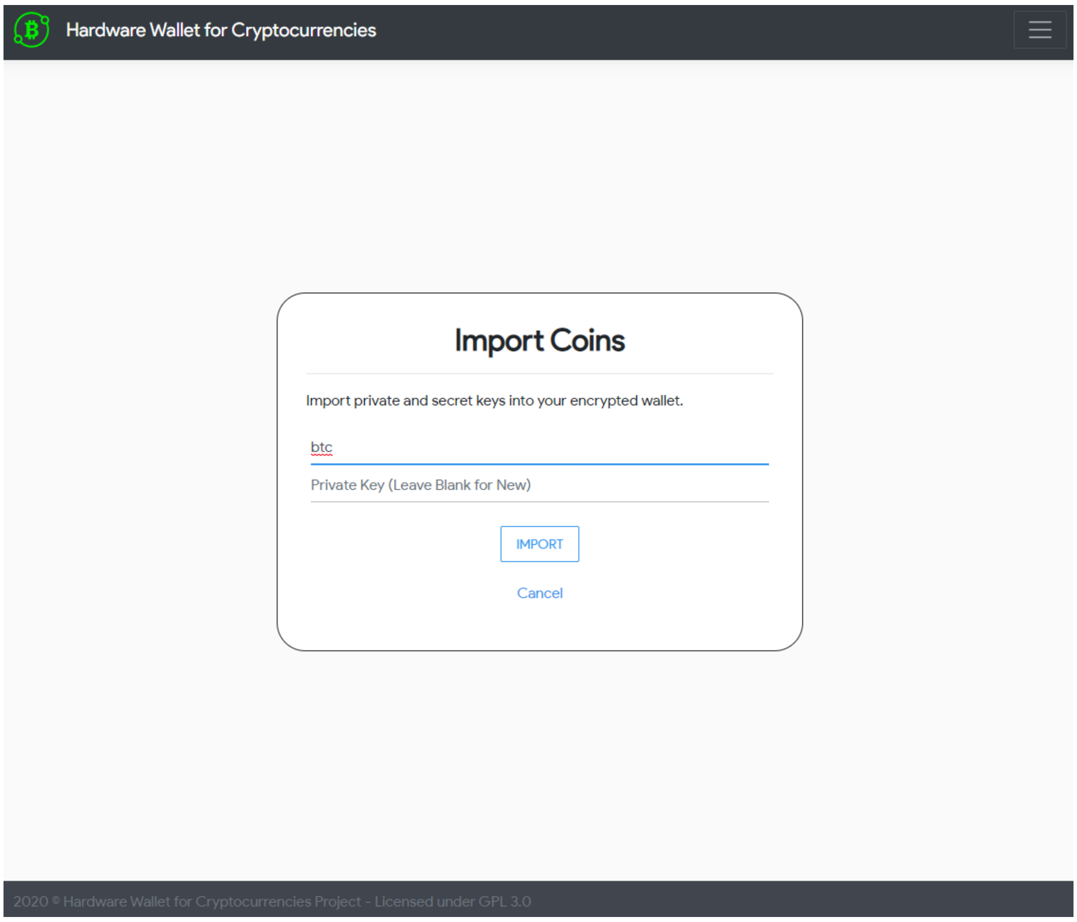
Disclaimer

This wallet interface is designed for Raspberry Pi Go Wallet for our Final Year Project. Please note that this is not intended to be used as a production wallet.

Adding New Cryptocurrency Coins

Clicking the *Import Button* allows the user to add new coin. User can also import a previous coin if the private key (WIF format) is already known. This field is optional and if left blank, a private key is randomly generated and stored inside the wallet structure.

Figure 30 Importing Coins



Hardware Wallet for Cryptocurrencies

Import Coins

Import private and secret keys into your encrypted wallet.

btc

Private Key (Leave Blank for New)

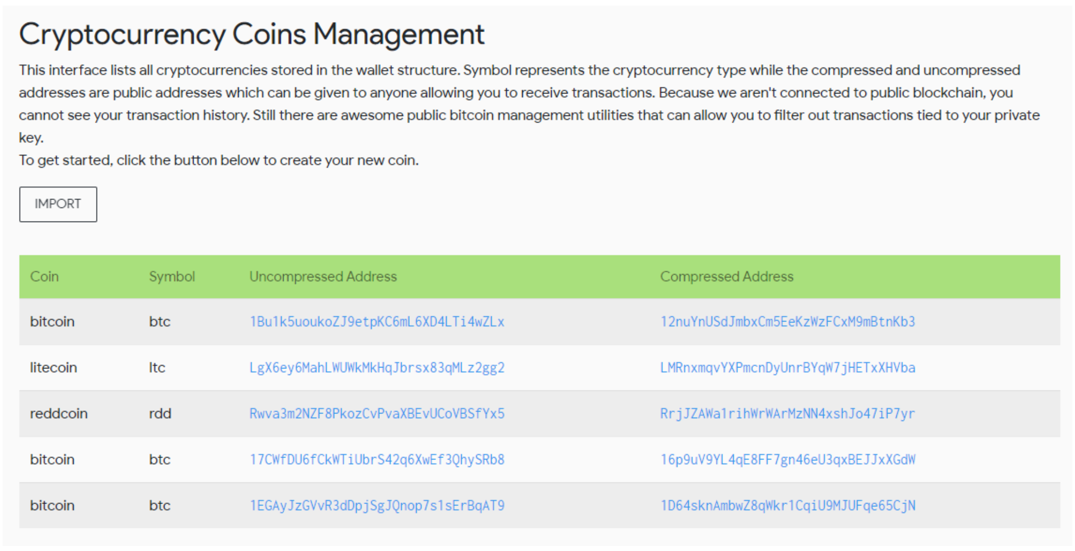
IMPORT

Cancel

2020 © Hardware Wallet for Cryptocurrencies Project - Licensed under GPL 3.0

After creating 'several' coins the main dashboard is populated automatically with all the generated coins. The private key information is hidden and only the public key is shown to the end-user.

Figure 31 Dashboard Populated with imported coins



Cryptocurrency Coins Management

This interface lists all cryptocurrencies stored in the wallet structure. Symbol represents the cryptocurrency type while the compressed and uncompressed addresses are public addresses which can be given to anyone allowing you to receive transactions. Because we aren't connected to public blockchain, you cannot see your transaction history. Still there are awesome public bitcoin management utilities that can allow you to filter out transactions tied to your private key.
To get started, click the button below to create your new coin.

Coin	Symbol	Uncompressed Address	Compressed Address
bitcoin	btc	1Bu1k5uoukoZJ9etpKC6mL6XD4LTi4wZLx	12nuYnUSdJmbxCm5EeKzWzFCxM9mBtnKb3
litecoin	ltc	LgX6ey6MahLWUWkMkHqJbrsx83qMLz2gg2	LMRnxmqvYXPmcnDyUnrBYqW7jHETxXHVba
reddcoin	rdd	Rwva3m2NZF8PkozCvPvaXBEvUCoVBSfYx5	RrjJZAWa1rihWrWArMzNN4xshJo47iP7yr
bitcoin	btc	17CWfDU6fCkWTiUbrS42q6XwEf3QhySRb8	16p9uV9YL4qE8FF7gn46eU3qxBEJJxXGdW
bitcoin	btc	1EGAyJzGVvR3dDpjSgJQnop7s1sErBqAT9	1D64sknAmbwZ8qWkr1CqiU9MJUFqe65CjN

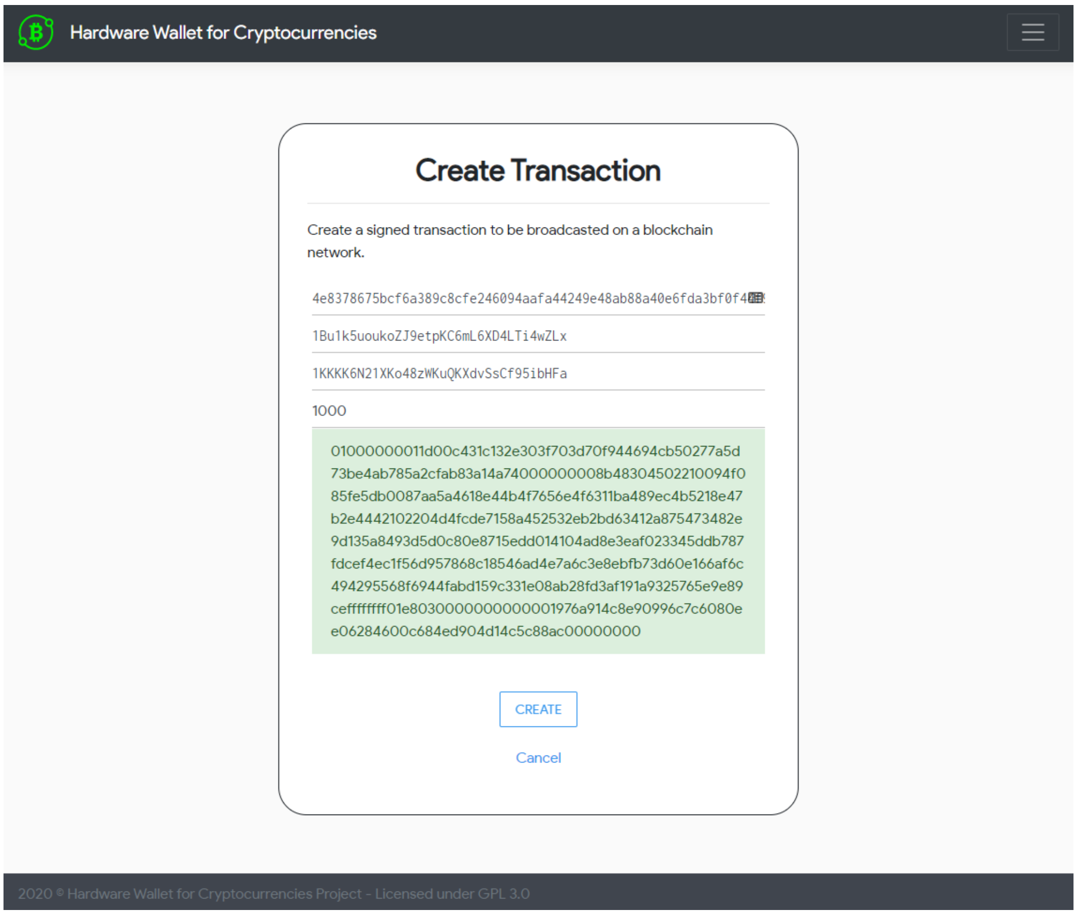
Creating and Signing Transactions

To create a new transaction, simply click on either of the compressed or uncompressed address and provide valid details in the required text boxes. If the provided transaction ID and sending address is valid, signed transaction will be displayed on screen which can be then broadcasted to any blockchain. The whole transaction structure is also dumped to filesystem for backup or validation purposes.

You can provide the following details to create a transaction:

```
{"txid":"744aa183abcfa285b74abe735d7a2750cb9446940fd703f703e332c131c4
001d",
    "source_address":"1Bu1k5uoukoZJ9etpKC6mL6XD4LTi4wZLx",
    "destination_address":"1KKKK6N21XKo48zWKuQKXdvSsCf95ibHFa",
    "amount":1000,
    "unsignedtx":"01000000016a914ff4f03bda6f0ea488ab489e2444faaa94602
4fe8c9c386acf5b6778834e0000000000ffffffff01e8030000000000001976a91477
8782087626f4d286d2dc32c10a135dedb675e888ac00000000",
    "signedtx":"01000000011d00c431c132e303f703d70f944694cb50277a5d73b
e4ab785a2cfab83a14a74000000008b48304502210094f085fe5db0087aa5a4618e44
b4f7656e4f6311ba489ec4b5218e47b2e4442102204d4fcde7158a452532eb2bd6341
2a875473482e9d135a8493d5d0c80e8715edd014104ad8e3eaf023345ddb787fdcef4
ec1f56d957868c18546ad4e7a6c3e8ebfb73d60e166af6c494295568f6944fabd159c
331e08ab28fd3af191a9325765e9e89ceffffffff01e8030000000000001976a914c8
e90996c7c6080ee06284600c684ed904d14c5c88ac00000000"
}
```

Figure 32 Creating new Transaction



Hardware Wallet for Cryptocurrencies

Create Transaction

Create a signed transaction to be broadcasted on a blockchain network.

```
4e8378675bcf6a389c8cfe246094aafa44249e48ab88a40e6fda3bf0f4
1Bu1k5uoukoZJ9etpKC6mL6XD4LTi4wZLx
1KKKK6N21XKo48zWKuQKXdvSsCf95ibHFa
1000
```

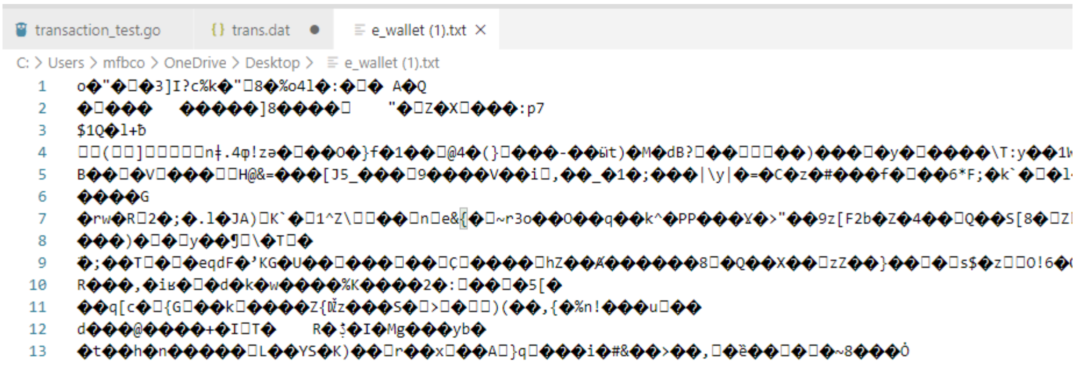
```
01000000011d00c431c132e303f703d70f944694cb50277a5d
73be4ab785a2cfab83a14a74000000008b48304502210094f0
85fe5db0087aa5a4618e44b4f7656e4f6311ba489ec4b5218e47
b2e4442102204d4fcde7158a452532eb2bd63412a875473482e
9d135a8493d5d0c80e8715edd014104ad8e3eaf023345ddb787
fdcef4ec1f56d957868c18546ad4e7a6c3e8ebfb73d60e166af6c
494295568f6944fabd159c331e08ab28fd3af191a9325765e9e89
ceffffffff01e8030000000000001976a914c8e90996c7c6080e
e06284600c684ed904d14c5c88ac00000000
```

2020 © Hardware Wallet for Cryptocurrencies Project - Licensed under GPL 3.0

Verifying encryption

To check if your wallet is encrypted, navigate to *Dev Tools* and click on Backup. Download the file to your PC and open it up in any text editor. The text is pretty much unreadable and garbage indicating that the entire wallet structure is encrypted.

Figure 33 Unreadable Wallet Backup



```
o�"�□�3]I?c%k�"□8�%o4l�:�□� A�Q
�□���   �����]8����□   "�□Z�X□���:p7
$1Q�l+ƀ
□□(□□]□□□□□n‡.4φ!zə�□���O�}f�1��□@4�(}□���-��ӹt)�M�dB?□��□□□��)���□�y�□����\T:y��1
B��□�V□���□□H@&=���[J5_���□9����V��i□,��_�1�;���|\y|�=�C�z�#���f�□���6*F;�k`�□�l
����G
�rw�R□2�;�.l�JA)□K`�□1^Z\□□��□n□e&{�□~r3o��O��q��k^�PP���Ӽ�>"��9z[F2b�Z�4��□Q��S[8�□Z
���)�□�□y��¶□\�T□�
�;��T□�□�eqdF�’KG�U��□���□��□Ç□����□hZ��Ӓ������8□�Q��X��□zZ��}��□�□s$�z□□O!6�
R���,�iв�□�d�k�w����%K����2�:□��□�5[�
��q[c�□{G□��k□����Z{Ŭz���S�□>□�□□)(��,{�%n!���u□��
d���@����+�I□T�    R�ȿ�I�Mg���yb�
�t��h�n�����□L��YS�K)��□r��x□��A□}q□���i�#&��>��,□�ȅ��□�□�~8���Ȯ
```

Dumping decrypted wallet

Selecting the *Dump* option downloads the decrypted wallet with all private keys or WIF details. Users are suggested to either delete the file after checking or back it up in a secure vault. If this file is leaked, all stored coins are compromised.

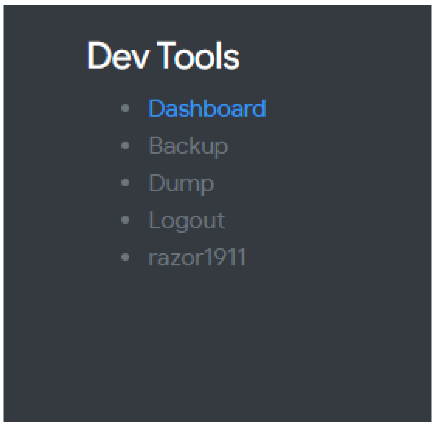
DO NOT USE THESE WIF KEYS FOR YOUR PERSONAL WALLET! THEY ARE FOR DEMONSTRATION PURPOSES ONLY

```
{"coins":[{"name":"bitcoin","symbol":"btc","wif":"5J3mVaTEmb9zUEGBHnq
esXEUiDxQxxy3b2E5GtzWKybMazKq9F8","uncompressed_address":"1Bu1k5uouko
ZJ9etpKC6mL6XD4LTi4wZLx","compressed_address":"12nuYnUSdJmbxCm5EeKzWz
FCxM9mBtnKb3"},{"name":"litecoin","symbol":"ltc","wif":"6usCkXqFSkbYo
23zxZ4NwBYCbn3Fsm71JAFP7KqPQNnxVugWasw","uncompressed_address":"LgX6e
y6MahLWUWkMkHqJbrsx83qMLz2gg2","compressed_address":"LMRnxmqvYXPmcnDy
UnrBYqW7jHETxXHVba"},{"name":"reddcoin","symbol":"rdd","wif":"7MrryJA
teMJvj7yLdTcshgMcWBLvsoF3JLmcxyV222DcceYwJjC","uncompressed_address":
"Rwva3m2NZF8PkozCvPvaXBEvUCoVBSfYx5","compressed_address":"RrjJZAWa1r
ihWrWArMzNN4xshJo47iP7yr"},{"name":"bitcoin","symbol":"btc","wif":"5J
vSepeA9xw8UeVfRcE75uA2yVjaqvMSBv5wkZLpQLjut1REqGY","uncompressed_addr
ess":"17CWfDU6fCkWTiUbrS42q6XwEf3QhySRb8","compressed_address":"16p9u
V9YL4qE8FF7gn46eU3qxBEJJxXGdW"},{"name":"bitcoin","symbol":"btc","wif
":"5JFeAgzLyHM2nzNvUR5vW5jMTtzu2vSNB8VjnCqYz98rndSUr4k","uncompressed
_address":"1EGAyJzGVvR3dDpjSgJQnop7s1sErBqAT9","compressed_address":"
1D64sknAmbwZ8qWkr1CqiU9MJUFqe65CjN"}],"personalize":"razor1911"}
```

Personalization Message

Within the Dev Tools the Personalization Message added during Wallet Setup is also included to verify device ownership.

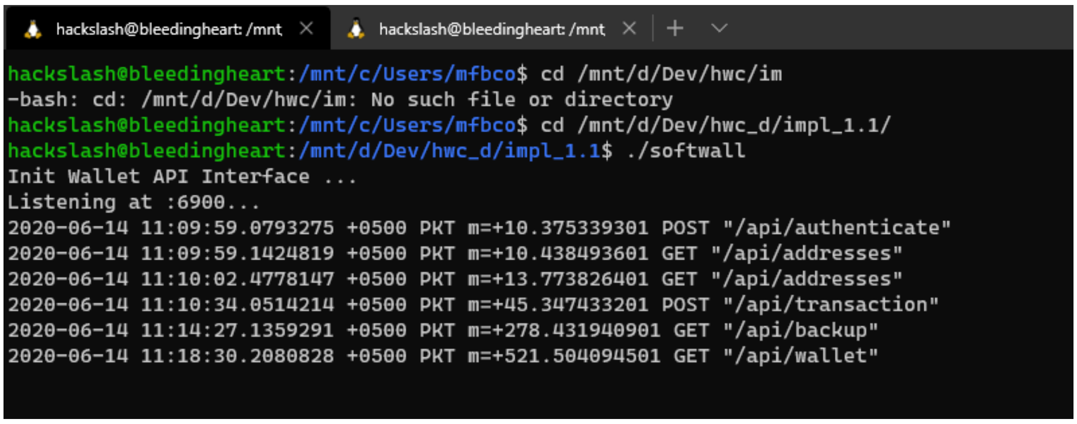
Figure 34 Personalization Message



Hit Endpoints

If a console (SSH) is monitored while accessing the wallet, all API endpoints over USB are logged and displayed:

Figure 35 API Endpoints Log



```
hackslash@bleedingheart: /mnt    hackslash@bleedingheart: /mnt
hackslash@bleedingheart:/mnt/c/Users/mfbco$ cd /mnt/d/Dev/hwc/im
-bash: cd: /mnt/d/Dev/hwc/im: No such file or directory
hackslash@bleedingheart:/mnt/c/Users/mfbco$ cd /mnt/d/Dev/hwc_d/impl_1.1/
hackslash@bleedingheart:/mnt/d/Dev/hwc_d/impl_1.1$ ./softwall
Init Wallet API Interface ...
Listening at :6900...
2020-06-14 11:09:59.0793275 +0500 PKT m=+10.375339301 POST "/api/authenticate"
2020-06-14 11:09:59.1424819 +0500 PKT m=+10.438493601 GET "/api/addresses"
2020-06-14 11:10:02.4778147 +0500 PKT m=+13.773826401 GET "/api/addresses"
2020-06-14 11:10:34.0514214 +0500 PKT m=+45.347433201 POST "/api/transaction"
2020-06-14 11:14:27.1359291 +0500 PKT m=+278.431940901 GET "/api/backup"
2020-06-14 11:18:30.2080828 +0500 PKT m=+521.504094501 GET "/api/wallet"
```

6.2 BOOTLOADER AND FIRMWARE

Unfortunately, due to unavailability of hardware in lockdown, we were unable to add results for bootloader and verification. However, the bootloader was validated on actual hardware before the lockdown and thus we guarantee its working and proper functionality. See the Appendix I section to request a copy of the source code and accompanying build instructions to compile and test.

6.3 ZEROTIER CONFIGURATION

If the Raspberry Pi wallet is used over Wi-Fi instead of USB emulation, please execute the following command using SSH:

```
curl -s
'https://raw.githubusercontent.com/zerotier/ZeroTierOne/master/doc/contac
t%40zerotier.com.gpg' | gpg --import && \
if z=$(curl -s 'https://install.zerotier.com/' | gpg); then echo "$z" |
sudo bash; fi
```

Then follow the Getting Stared Guide:

https://zerotier.atlassian.net/wiki/spaces/SD/pages/8454145/Getting+Started+with+ZeroTier to setup your first network and attach a static private IP to Raspberry Pi. After your main PC device and Pi is connected to same ZeroTier Private VPN network, you can simply enter that static IP in browser to access wallet interface and manage it.

Chapter 7

CONCLUSION

In this work, we dived into Bitcoin currency fundamentals and explored different cryptographic primitives that make up the whole Bitcoin ecosystem. We examined different wallet solutions available in the market, from cloud wallets to paper based and finally hardware wallets. We examined the complications and restraints of hardware wallets and realized the need of open and robust framework and architecture on which future wallets can be based on.

We also developed a prototype wallet based on our design for both Raspberry Pi and Cortex M4 MPU. We verified primitive wallet functionality and backed up our design with a friendly and intuitive user interface. The prototype we developed is limited in contrast to other proprietary wallets, in terms of both security and functionality. However, we do believe that the prototypes can be easily extended to accommodate more feature enhancements leading to a viable open alternative to other solutions.

Chapter 8

FUTURE RECOMMENDATIONS

Over the tenure of this project, we realized that hardware wallet as a project encompasses many different fields, ranging from cryptography to hardware security, protocols, encryption, RNG, etc. All of these different fields can be exploited as separate projects to improve the overall wallet design.

We believe there's a rich set of possibilities as future recommendations to further improve on this project:

- Extend the Pi wallet to add integrated display and buttons to allow user to confirm transactions on hardware instead of the wallet interface.
- Extend Pi wallet by adding a secure element (on GPIO pins) to separate storage element.
- Set up attack and vulnerability analysis vectors to improve security aspects of the wallet.
- Add blockchain communication to the wallet and add split transactions support.
- Research on microprocessor chaining protocols to separate wallet algorithms and private key storage such as ARM Secure Core microcontrollers.
- Research on TEE (Trusted Execution Environments) on both Intel and ARM architecture to further safeguard transactions.
- Research on a pure stateless wallet design that generates all private keys on-the-go, removing the need of any secure storage element.
- Extend the user interface to handle and show receiving transactions and display account balances, etc.

APPENDIX I

Obtaining Source Code

The project's source code is licensed under General Public License 3.0 (GPL 3). You can read the terms here: https://www.gnu.org/licenses/gpl-3.0.en.html

In short, if use any portion of our code, you've to opensource your entire project under the same license.

The code can be requested by sending a mail to mbaig.bee16seecs@seecs.edu.pk or fadi@fahadbaig.com.

REFERENCES

[1] https://news.bitcoin.com/satoshi-nakamotos-brilliant-white-paper-turns-9-years-old/ (June 2020)

[2] https://www.coinbase.com/price/bitcoin (June 2020)

[3] https://coinmarketcap.com/charts/ (June 2020)

[4] https://www.jpmorganchase.com/corporate/news/stories/could-blockchain-have-great-impact-as-internet.htm (June 2020)

[5] https://news.bitcoin.com/hackers-have-looted-more-bitcoin-than-satoshis-entire-stash/ (June 2020)

[6] https://nakamoto.com/public-key-cryptography/ (June 2020)

[7] DeVries, Peter. (2016). An Analysis of Cryptocurrency, Bitcoin, and the Future. International Journal of Business Management and Commerce. Vol. 1. Pages 1-9.

[8] How Bitcoin Transactions Work, by Aalim Khan

[9] "Charts". Blockchain.info. Archived from the original on 3 November 2014. Retrieved 2 November 2014.

[10] Nakamoto, Satoshi (24 May 2009). "Bitcoin: A Peer-to-Peer Electronic Cash System" (PDF). Retrieved 20 December 2012.

[11] https://www.webopedia.com/TERM/C/cryptocurrency-mining.html (June 2020)

[12] https://en.bitcoin.it/wiki/Hardware_wallet (June 2020)

[13] https://www.ubuntupit.com/best-cryptocurrency-wallets/ (June 2020)

[14] https://www.investopedia.com/terms/b/blockchain.asp (June 2020)

[15] https://www.investopedia.com/terms/b/bitcoin-mining.asp (June 2020)

[16] Cryptocurrency Wallet Guide: A Step-By-Step Tutorial by Ameer Rasic

[17] https://blockgeeks.com/guides/best-hardware-wallets-comparative-list-blockgeeks/ (June 2020)

[18] https://builtin.com/blockchain (June 2020)

[19] Bernstein, D.J. and Lange T., (2017), Post-quantum cryptography, Nature 549, 188–194

[20] 2020 State of Malware Report, malwarebytes.com, Malwarebytes LABS, (2020), https://resources.malwarebytes.com/files/2020/02/2020_State-of-Malware-Report.pdf

[21] Webroot Threat Report 2020. webroot.com, Webroot Inc., OpenText, (2020), https://mypage.webroot.com/rs/557-FSI-195/images/2020%20Webroot%20Threat%20Report_US_FINAL.pdf

[22] Pan, D. (2019). Hackers Launch Widespread Botnet Attack on Crypto Wallets Using Cheap Russian Malware. Retrieved from Coindesk: https://www.coindesk.com/hackers-launch-widespread-botnet-attack-on-crypto-wallets-using-cheap-russian-malware

[23] Thompson, P. (2020). Most Significant Hacks of 2019 — New Record of Twelve in One Year. Retrieved from Cointelegraph https://cointelegraph.com/news/most-significant-hacks-of-2019-new-record-of-twelve-in-one-year

[24] Trezor Software Security Documentation https://wiki.trezor.io/Security:Software

[25] Open Framework. BusinessDictionary.com. WebFinance, Inc. , http://www.businessdictionary.com/definition/open-framework.html

[26] Haapaviita, N., Jokelainen, J., Reijonen, P. and Haikonen, J., Open Source Software Project Success Factors / Modularity as a Success Factor. University FYP

[27] "Proprietary." Merriam-Webster.com Dictionary, Merriam-Webster, https://www.merriam-webster.com/dictionary/proprietary

[28] Collins, M., (1998), Formal Methods, Carnegie Mellon University, 18-849b Dependable Embedded System

[29] Ivan Homoliak, Dominik Breitenbacher, Ondrej Hujnak, Pieter Hartel, Alexander Binder, Pawel Szalachowski, (2020), SmartOTPs: An Air-Gapped 2-Factor Authentication for Smart-Contract Wallets

[30] Das, Poulami., Faust, Sebastian., and Loss, Julian., (2019), A Formal Treatment of Deterministic Wallets. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), Association for Computing Machinery, New York, NY, USA, 651–668. DOI: https://doi.org/10.1145/3319535.3354236

[31] Arapinis, Myrto., Gkaniatsou, Andriana., Karakostas, Dimitris., and Kiayias, Aggelos., (2019), A formal treatment of hardware wallets. Cryptology ePrint Archive, Report 2019/034, https://eprint.iacr.org/2019/034

[32] Page 07 https://www.st.com/resource/en/application_note/dm00186528-proprietary-code-readout-protection-on-microcontrollers-of-the-stm32f4-series-stmicroelectronics.pdf

[33] https://magpi.raspberrypi.org/articles/raspberry-pi-3bplus-specs-benchmarks (June 2020)

[34] https://www.st.com/en/microcontrollers-microprocessors/stm32f405rg.html (June 2020)