6-15-2018

CryptKi: Mobile Hardware Wallet

Derrick Chan
Santa Clara University, dchan1@scu.edu

Rowan Decker
Santa Clara University, rdecker@scu.edu

William Nguyen
Santa Clara University, wnguyen@scu.edu

Yuya Oguchi
Santa Clara University, yoguchi@scu.edu

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING

Date: June 14, 2018

I HEREBY RECOMMEND THAT THE DESIGN REPORT PREPARED UNDER MY SUPERVISION BY

Derrick Chan
Rowan Decker
William Nguyen
Yuya Oguchi

ENTITLED

CryptKi: Mobile Hardware Wallet

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREES OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING
BACHELOR OF SCIENCE IN ELECTRICAL ENGINEERING

[illegible]
Advisor

[illegible]
Advisor

N. Ling
Department Chair

Shoba Krishnan
Department Chair

CryptKi: Mobile Hardware Wallet

by

Derrick Chan
Rowan Decker
William Nguyen
Yuya Oguchi

Submitted in partial fulfillment of the requirements
for the degrees of
Bachelor of Science in Computer Science and Engineering
Bachelor of Science in Electrical Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 15, 2018

CryptKi: Mobile Hardware Wallet

Derrick Chan
Rowan Decker
William Nguyen
Yuya Oguchi

Department of Computer Engineering
Department of Electrical Engineering
Santa Clara University
June 15, 2018

ABSTRACT

Cryptocurrencies are rapidly receiving mainstream attention and adoption. As a result, new technologies are developing to store and transact cryptocurrencies to keep up with a rapidly developing market. We have developed a way to utilize a mobile phone to communicate with the Ethereum blockchain in order to monitor and initiate cryptocurrency transactions. The project allows users to create new Ethereum addresses and to send arbitrary amounts of Ether to existing Ethereum addresses, while maintaining security for the end user. We conclude that this method of communication with the Ethereum blockchain is viable and is an avenue for exploration in the future. As a result from this project, we anticipate a growth of new cryptocurrency applications on mobile devices. Furthermore, the various security features utilized in this project such as pin randomization, mnemonic recovery, and hardware level encryption can be implemented in other applications to increase security for all users.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Objective

Cryptocurrencies are a type of decentralized currency which are accessible, reliable, and secure. They cannot be counterfeited and can be freely traded without the use of a third party. Currently, there only a few ways people can access their cryptocurrency to accept and enact transactions. We plan to create a more secure and mobile way for people to use cryptocurrencies.

1.2 What are cryptocurrencies?

Cryptocurrencies are a type of decentralized online currency that utilizes blockchain technology. What it means to be decentralized is that there is no single supplier of information. Rather, everybody has the same copy of the same information. This information is known as the blockchain. Having multiple people possess the same information may seem counterintuitive and wasteful but there are many benefits to not having a single repository of information.

Firstly, in order for any new information to be added to the shared data, everyone has to agree to add that information to the record. The implication here is that adding false data is impossible because you would need a majority of people to agree to add false information. Additionally, there are software safeguards in place that validate any attempt to submit data. Furthermore, since everyone has the same copy of the same data, the data cannot be deleted or retroactively changed. The redundant storage of information guarantees that multiple copies will survive data failures and that past data cannot be mutated

Blockchain technology is useful for cryptocurrencies because now, the blockchain acts as a ledger for all transactions done. Since an individual cannot change the blockchain, false transactions cannot be created. Since false transactions cannot be created, counterfeit currency cannot be created. Finally, because the blockchain is decentralized and completely virtual, anyone can exchange money with anyone else at any time. There are no additional fees or restrictions for sending funds to people in other countries. Cryptocurrencies promise on-demand transactions for everyone, which is beneficial for any currency system. [2]

1.3 Why do you need a hardware wallet?

Let us introduce the concept of a hardware wallet with an analogy to debit cards. Just like debit cards, a hardware wallet can be used to buy goods and services. Like a debit card, you can not spend more money then you currently possess with a hardware wallet. However, if you lose your debit card, you are at risk of losing all of your money because all of the information needed to make a transaction is on the card. Whereas a hardware wallet requires additional information that is not on the wallet to initiate a transaction.

The main benefit of using a hardware wallet is maintaining security. One of the fundamental flaws of cryptocurrencies is that you have to maintain your own security. There is no central authority that will provide insurance for lost or stolen funds. A hardware wallet mitigates the risk of having your funds stolen by providing various security measures to protect your private key. In this context, a private key is what enables a transaction to be sent. If your private key is compromised, anyone can send money from your account to any other account. A hardware wallet protects your private key by isolating your private key from the outside world. This protects your private key from malware like keyloggers or trojans.

1.4 Related Products

The two main competitors in this market are the Trezor [3] and the Ledger Nano S [4]. Both are hardware wallets and support multiple cyrptocurrencies like Bitcoin, Litecoin, and Ethereum. Both of these products connect to a desktop computer via a USB interface. This means that both wallets do not have a way to interface with a mobile phone. Additionally, both products have known security flaws and availability issues.

1.5 Solution

Our solution to current cryptocurrency transaction issues is to create a hardware wallet that is easily accessible from the Android app. The wallet will act as a form of multi-factor authentication and provide a separate, secure, area for accounts to reside. The wallet will use explicit physical interaction as a signature and as a confirmation method for transactions.

1.6 Considerations

1.6.1 Why Blockchain Technology Needs Security

Blockchain technology is an emerging technology that promises a way to create decentralized systems and nodes that all agree with each other. This technology creates the platform for decentralization but does not inherently provide a comprehensive security for the end user. Essentially, the user can use the blockchain technology but the technology has no guarantee that the users data will remain secure.

The goal of this project is to provide tools that enable users to more securely use blockchain currencies like Ethereum. The main ethical consideration for this venture is the expectation of privacy. Given recent breaches like Equifax, decentralized systems are growing in popularity in light of big financial institutions losing trust. The main ethical framework that can be employed here is the concept of the common good. People need access to banking institutions. People need security to protect their funds and assets. In this world, people need money. We are providing a framework to build a system that puts trust into individual persons.

The common good framework ensures that ethical decisions are driven by the idea that acts must promote the benefit of all people. The platform we are building is for the common good. The ultimate goal is to provide a method to securely transmit money and to provide a safe way to store the money. These goals relate to the common good because these technologies will benefit the economy because it promotes people to exchange money. A stronger world economy will lead to a higher standard of living for all. A currency system not backed by a single government would allow everyone to use the same currency. People would have the ability to trade goods with people in different countries with no transaction fees and minimal lag time. A single currency enables people to interact directly with each other in an increasingly interconnected world.

1.6.2 Virtues of a Good Engineer

Techno-social sensitivity

Technology has quickly changed the world. It has given people the ability to communicate across the globe instantaneously. Technology is now poised to allow people to transfer funds quickly and securely without the reliance of a central or national institution. This is especially helpful for developing countries where there is not a formal infrastructure to provide these abilities to their people.

Respect for nature

Our project helps reduce the reliance on paper money hopefully leading to reduction in the waste products associated with the production of money. Our project will only have to be produced and purchased once but can then continue to be used without requiring additional physical resources.

Commitment to the public good

Every so often, we experience major economic crisis. This result in damages to individuals, companies and countries all around the world now that the world is all financially interconnected. The vulnerable individuals who are affected by these crises could lose their assets and be left with nothing. With our product, we are promoting use of cryptocurrency, which acts as a secondary independent economy. This prevents the whole economy from crashing down as crisis in one economy is only the half of the whole economy. While economic crisis will still hurt many, our product which promotes the cryptocurrency economy cushions the damage.

Teamwork

Our project has helped us realize the importance of communication. The project contains many separate yet interconnected parts: PCB design, ARM firmware programming, Android development, etc. For instance, the firmware could not have be written until there was a physical device to test on, and the Android app relies upon using a consistent communication protocol with the board. Any changes to the parameters the board looks for will require changes in the app. This required us to document standards so that everyone could stay on the same page and make a working ecosystem.

Courage

Our project requires courage because it is in a very new field that most people do not know about. This makes it tough to advertise and explain our project to the layperson. Furthermore, as just a group of four college students it may be tough for others to trust our implementation with their own financial security. Finally, as the technology is quickly changing we are forced to find information not from formal writeups and documentation but from peering through code which can be tough.

1.6.3 Risks and Safety

Risks

The main risks involved with our project is someone gaining access to our clients cryptocurrency. Attackers can attempt to gain access through both physical attacks and cyber attacks. Physical attacks may include but are not limited to coercing someone to hand over the hardware and tell the attacker security pin number used for transaction authorization. The cyber attacks mainly include finding flaws in software or hardware and using the vulnerability to gain ways to access data in unauthorized ways. Another method of cyber attack is done through random guessing. While it is unlikely for anyone to guess someones pin code or message randomly, it is not 0%. Our clients are often not aware of the exact math and effort behind our hardware and it is often difficult to explain in simple terms to our clients as the process is fairly complex, but it is possible to notify users that there are still potential risks involved with our product even though the whole project revolves around increasing protection.

The Public

The public for our project are people familiar with cryptocurrencies who are looking for a secure way to transfer funds while on the go. The people at risk are those who make transactions from a mobile device that may be tampered with. Our goal is to limit the risk for these people. We believe that any risks introduced by our device will be less likely and less impactful than to those who do not have it.

Informed Consent

Our project will describe the devices operation on a website, allowing people to make an informed decision before purchasing the device. Delivered along with the project will be a piece of paper once again outlining the terms for those who may not have read them online. If a person would not like to use the device they are not forced to. Ultimately while we try to provide a more secure interface, we are unable to guarantee with complete certainty that a person will not lose funds.

Chapter 2

Project and Design

2.1 Project Requirements

These are the requirements necessary to achieve the design goal set forth. This section is broken down into three subsections, the mobile app, the physical hardware wallet, and the Bluetooth services. Each of these sections include the functional and non-functional requirements. In this context, a functional requirement is a design decision that is fundamental to the overall project and is a decision that describes what the system does. A non-functional requirement describes how the system should be.

2.1.1 Specifications and constraints

There are also several specifications and constrains that must be met by the project as a whole:

- The cost must be less than $50 USD. Other current competitors on hardware wallet market marks their prices starting at around $70. marking our product price at below $50 will allow us to become competitive alternative to current existing products.
- The size must be smaller than $50cm^2$ as that is an approximate size of credit cards. The hardware wallet must be smaller than that to fit in wallet along with other cards to be portable.
- The power consumption of hardware wallet must be under 10mA when active. In order to maintain longevity of the device, it needs to use low power in order to let the device work for longer time without changing the battery.

2.1.2 Android App

The Android App is an integral part of the design goal for this project. The project utilizes the smart phone app to communicate with the overall Ethereum block chain. Furthermore, the mobile app connects to the bluetooth hardware wallet and serve as the controller for the wallet. Originally, an iOS app was planned. Instead, we opted to focus only the Android app where a future iOS release will copy the Android app's functionality and aesthetic.

Functional Requirements

- Communicates with an Ethereum full node
- Utilizes Bluetooth Low Energy (BLE) services to communicate with a hardware wallet
- Uses multi-factor authentication
- QR code reader
- App on the PlayStore for Android
- Retrieves data from the Ethereum Blockchain
- Displays address information

Non-Functional Requirements

- Simple and intuitive design
- Easy to interact
- Minimize system resource usage
- Responsive
- Be a secure platform

2.1.3 Physical Hardware Wallet

The hardware wallet is one of the fundamental pieces of this project. The wallet will provide an additional layer of security for this system.

Functional

- Utilizes Openssl and keccak-256 to generate public/private keys
- Generates signed Ethereum transactions
- Implements two-factor authentication
- Uses Bluetooth Low Energy services to send data to an Android phone
- Screen brightness adjustment
- Displays private key as a mnemonic phrase on setup
- Restore account from a mnemonic phrase.

Non-Functional

- Portable size
- Durable housing
- Minimize attack vectors

2.1.4 Bluetooth Services

Bluetooth technology allows communication and data transfer between devices without using physical cables between. Bluetooth is also categorized as low energy. These traits made Bluetooth a good choice for the hardware wallet.

Functional Requirements

- Uses a vendor specific UUID
- Sends transaction information
- Connects to devices within a reachable distance

Non-Functional Requirements

- Low energy consumption
- Maintains a stable connection to the phone
- Private connection
- Small size to fit on hardware wallet

2.2 Use Cases

This section of the report details how the end user interacts with the hardware wallet. For this project, there are two relevant actors: sender and receiver. The main action that a sender will take is to initiate cryptocurrency transactions. The receiver undertakes a more passive role in this exchange because the main action the receiver takes is supplying an address for the sender to send funds to.

2.2.1 Case Diagram

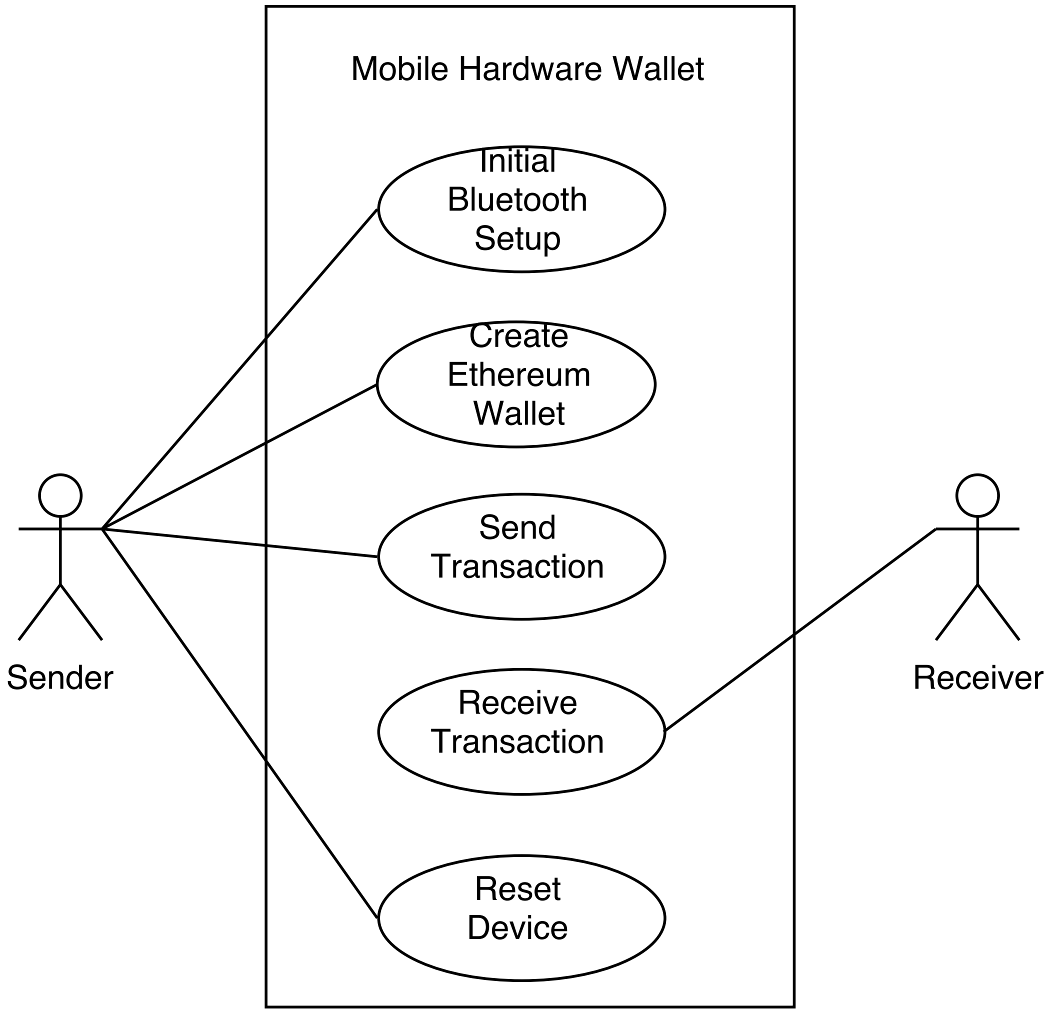


Figure 2.1: Uses Cases

2.2.2 Task Description

To use our product, users will go through several steps to make a transaction. Let's say a user wants to purchase a product from a vendor. If it is the user's first time using the hardware wallet, then they will download our mobile application and connect to the hardware wallet via bluetooth connection through the app. At this point, the user will go through an account creation process to create an Ethereum address. Once the account is connected to an address, the user can send a transaction to the vendor to pay for the product.

If a user wants to check their account, they may do so through the mobile application. If they have already created an account, they can check transaction history on the mobile device.

In case a user wants to completely reset the account, the user will be able to reset their device either from the phone app or the hardware wallet.

This section will describe the individual aspects of each use case. Each case will have its own goal and actors. The goal describes why a particular actor would want to use the project. The precondition for each use case is the set of conditions that must be true before the beginning of a use case.

A) Name: Initial Bluetooth Connection
Goal of use case: Provide the basic setup for the hardware wallet. This includes installing the mobile application and connecting the hardware wallet to the phone.
Actors of use case: The sender
Pre-conditions: A mobile phone running iOS or Android and the hardware wallet.
Steps:

1. Power on the wallet and begin setup
2. Create a security pin
3. Download the app from your respective app store
4. Open the app and navigate to the connection interface
5. Initiate the Bluetooth connection on the mobile app and find the name of hardware wallet from list of bluetooth device list
6. Press the connect button on the hardware wallet to complete the pairing process

Post-Conditions: The sender will have the mobile app installed and the hardware wallet connected to their phone.
Exceptions: Pairing failures
B) Name: Create an Ethereum wallet

Goal of use case: Create an Ethereum wallet that will be stored on the hardware wallet.
Actors of use case: The sender
Pre-Conditions: The mobile app and the hardware wallet
Steps:

1. Initiate the account creation process on the mobile app
2. The hardware wallet will create an Ethereum address
3. The backup codes will be displayed on the hardware wallet
4. Copy the secret words to a safe place

Post-Conditions: An Ethereum address that will be able to send and receive funds and a private key that will be used to recover the account.
Exceptions: None

C) Name: Send Transaction
Goal of use case: Initiate a new transaction that will be sent to an Ethereum address.
Actors of use case: The sender and receiver
Pre-conditions: The sender must have some amount of Ethereum that can be sent. The receiver must make their address public. The mobile app must be connected to the hardware wallet via Bluetooth
Steps:

1. Initiate a transaction from the mobile app
2. Input the transaction amount, destination address, gas limit
3. Confirm the details and send the transaction to the hardware wallet
4. Confirm the transaction on the wallet
5. The app will submit the signed transaction to the Ethereum blockchain.

Post-Conditions: A new transaction is submitted to the Ethereum blockchain.
Exceptions: User cancels the transaction

D) Name: Receiving a Transaction
Goal of use case: Transfer Ethereum to a recipient.
Actors of use case: The sender and receiver
Pre-conditions: The receiver must have a valid Ethereum address.
Steps:

1. Publicize the receiving address
2. Have another user send funds to that address

Post-Conditions: The transaction will be shown in the recipients app
Exceptions: None

E) Name: Factory reset
Goal of use case: Reset the hardware wallet in case the user is no longer the owner of the device whether on purpose such as gifting or on accident such as losing the device or having it stolen.
Actors of use case: The user
Pre-conditions: Must have set up and paired the device with the phone application
Steps:

1. Open phone application.
2. Navigate to the factory reset option.
3. Enter PIN to confirm the reset.

Post-Conditions:
Exceptions: None

2.3 Activity Diagrams

This section of the report describes the actions that a user will take while utilizing this system.

2.3.1 Setup

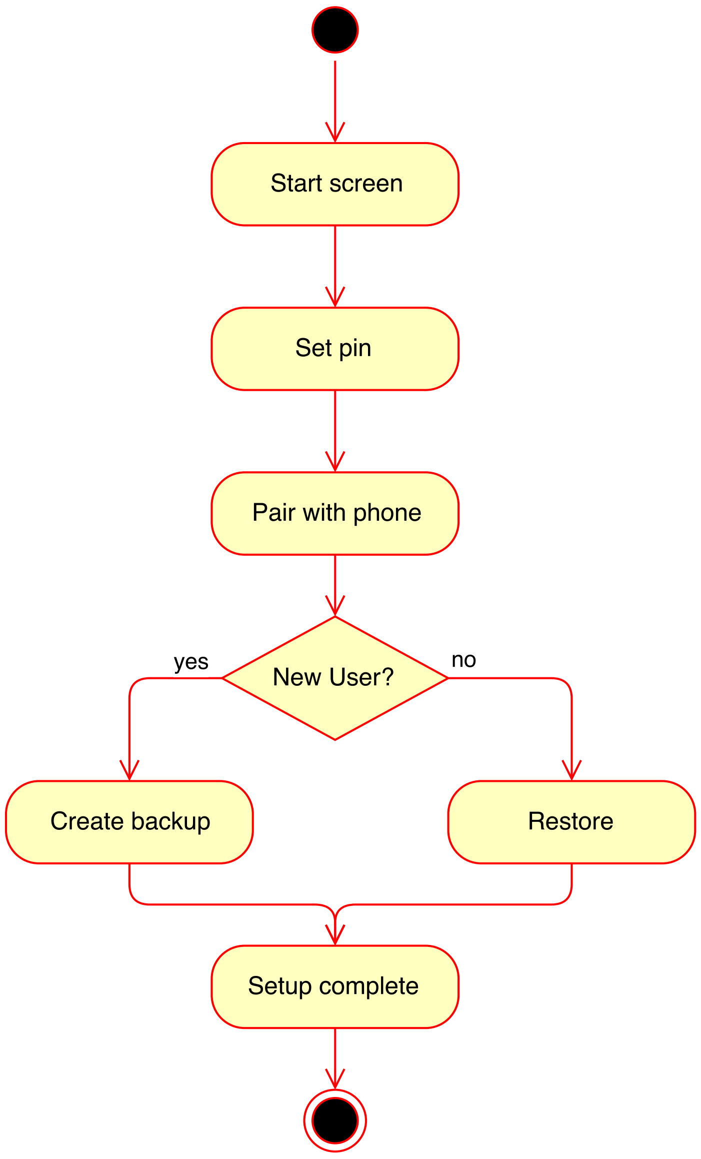


Figure 2.2: Setup Activity

The Figure 2.2 shows list of steps a user will go through to set up the hardware wallet for the first time.

2.3.2 Settings

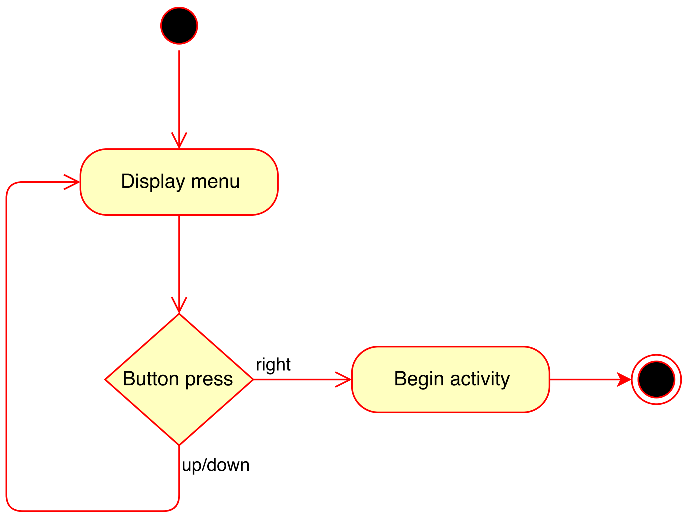


Figure 2.3: Settings Activity

The Figure 2.3 shows menu screen actions. Users have several options on the menu screen to select from. They may select different options with up and down buttons and start the selected action using the right key.

2.3.3 Backup

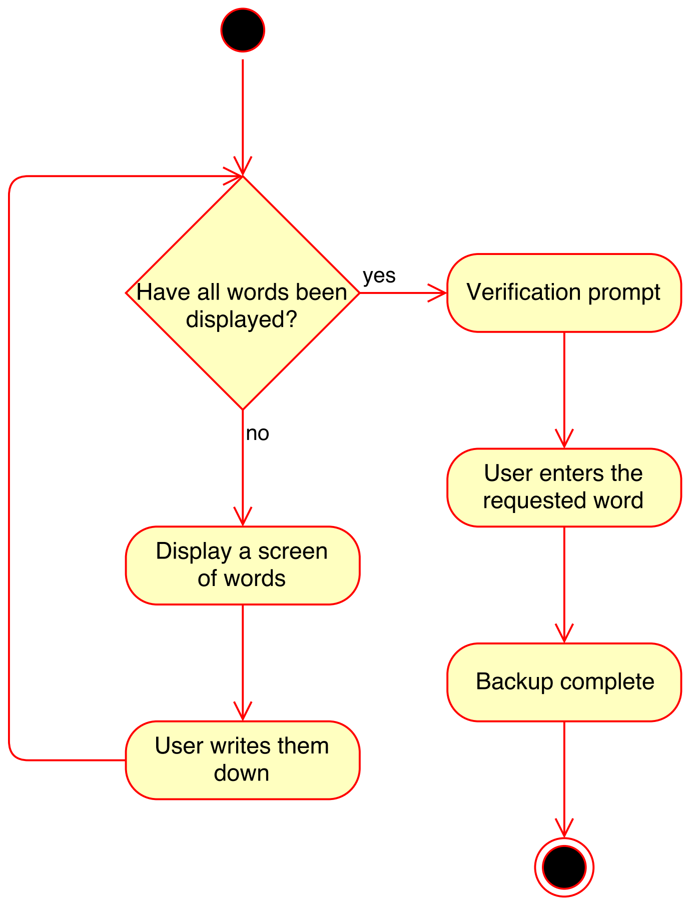

Have all words been displayed?
yes
no
Verification prompt
User enters the requested word
Backup complete
Display a screen of words
User writes them down

Figure 2.4: Backup Activity

The Figure 2.4 shows diagram for creating a backup. The hardware wallet will display list of words that will be used for back up. At this point user will write them down. Once written down, the hardware wallet will verify that user actually wrote them down by testing the user what the keywords were.

2.3.4 Restore

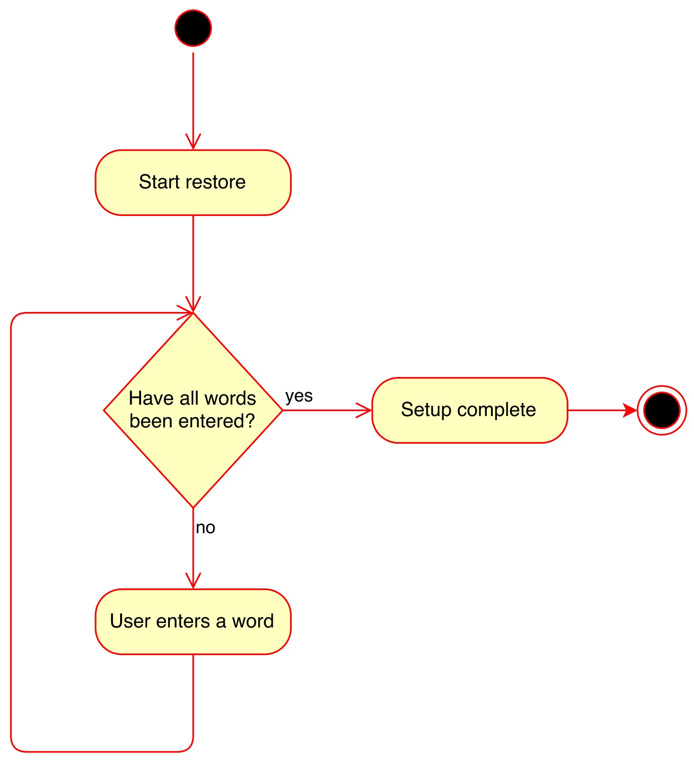


Figure 2.5: Restore Activity

The Figure 2.5 shows diagram for restoring an account on hardware wallet. In order to do so, the user will start restore process and enter all keywords that were initially set up when creating an account. If all the words match, the account is restored.

2.3.5 Send

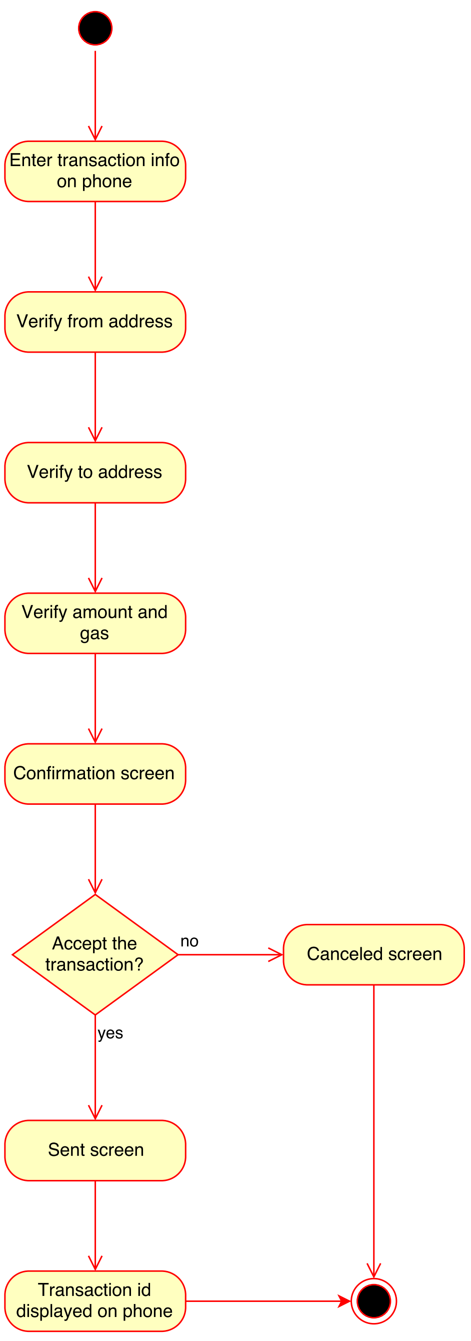


Figure 2.6: Send Activity

The Figure 2.6 shows diagram for sending a transaction. First the user inputs relevant transaction information onto the phone application. Both addresses used in the transaction are checked. Once those are verified, the amount and gas used in the transaction is also verified. Finally, the confirmation screen for the transaction is displayed, at which point a user may decline; which stops the transaction, or accepts to send the transaction and receive a confirmation on the hardware wallet, and transaction id on phone application.

2.4 Usage

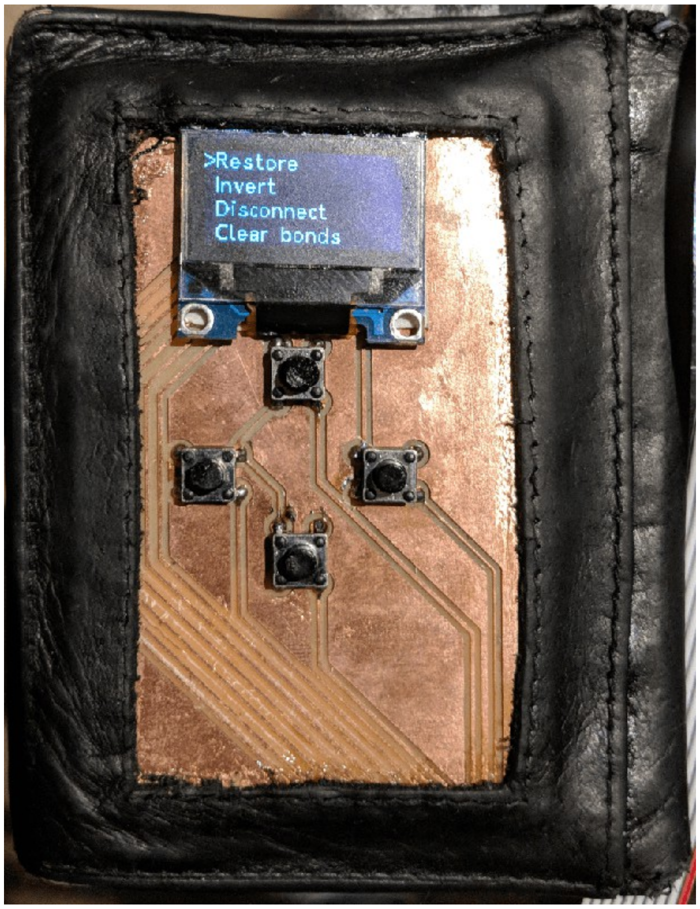
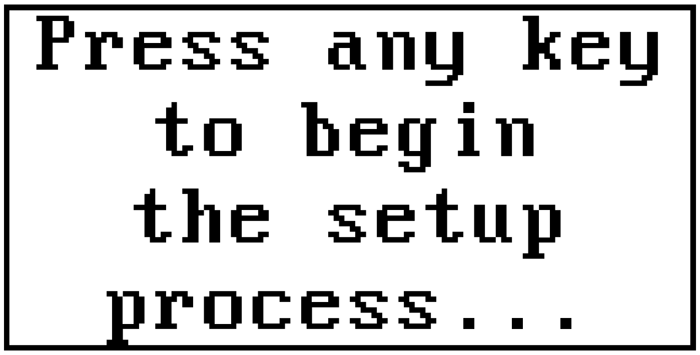


Figure 2.7: Hardware Wallet Mockup

The hardware wallet will have a screen and four buttons arranged in a plus formation.

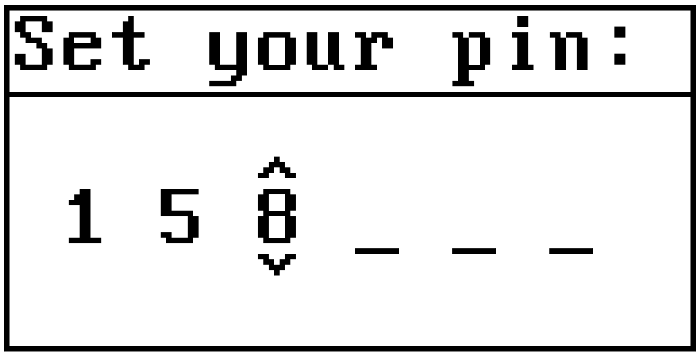
2.4.1 Setup



Press any key
to begin
the setup
process...

Figure 2.8: Setup Screen

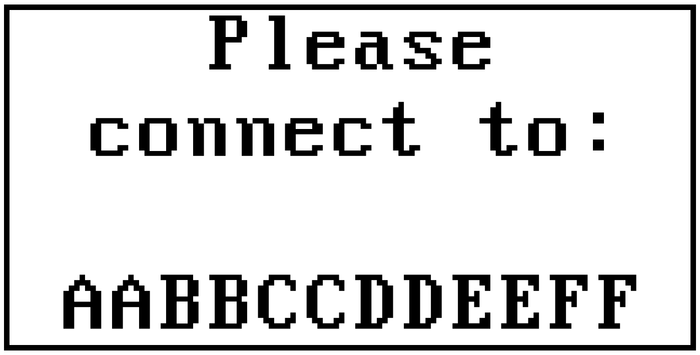
This screen will be displayed when the hardware wallet is powered on for the first time. As mentioned on the display pressing, any key will begin the setup process.


Set your pin:

1 5 8 _ _ _

Figure 2.9: Pin Screen

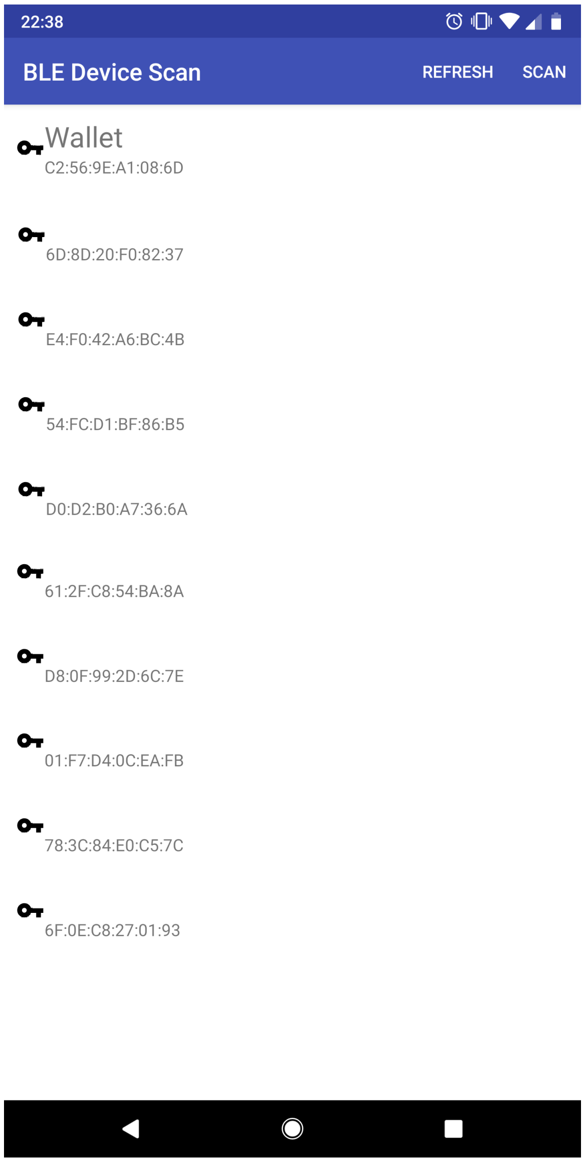
This screen allows the user to set a security pin to use for future connections. This prevents other people from being able to use the device. Pressing the up and down buttons will increase and decrease the value of the current digit respectively. Pressing right will move onto the next digit, and after all digits are entered progresses to the next screen. Pressing left will allow previous digits to be modified.


Please
connect to:

AABBCCDDEEFF

Figure 2.10: Connect Screen

The screen displays the name that will appear on a list of bluetooth devices allowing for easier pairing from a phone. The device will automatically advance to the next screen once it has detected a bluetooth connection.



Wallet
C2:56:9E:A1:08:6D

6D:8D:20:F0:82:37

E4:F0:42:A6:BC:4B

54:FC:D1:BF:86:B5

D0:D2:B0:A7:36:6A

61:2F:C8:54:BA:8A

D8:0F:99:2D:6C:7E

01:F7:D4:0C:EA:FB

78:3C:84:E0:C5:7C

6F:0E:C8:27:01:93

Figure 2.11: Phone Bluetooth Connect screen

On the mobile application, a list of available hardware wallets will be displayed. Upon pressing ”Pair”, the hardware wallet and the phone will be connected via Bluetooth.

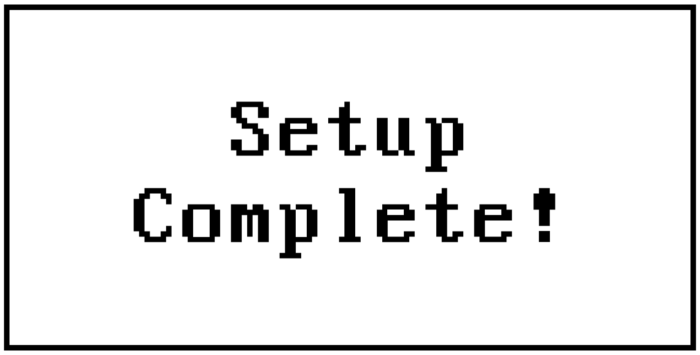


Figure 2.12: Setup Complete

Once the device is paired and the backup words have been verified then the setup is complete. Pressing any button will move onto the main menu.

2.4.2 Sending a Transaction

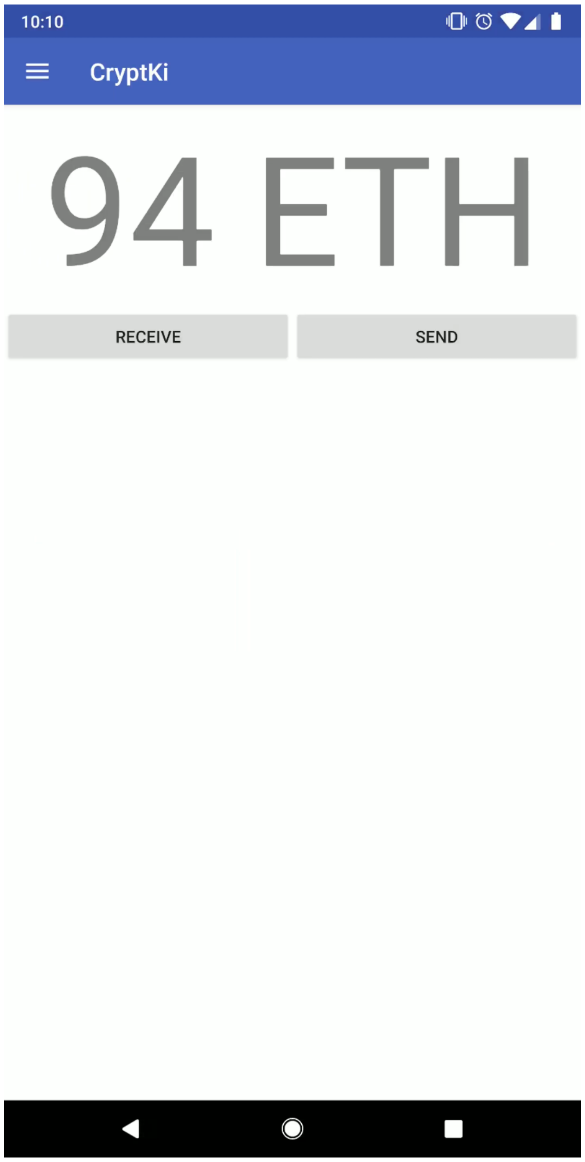
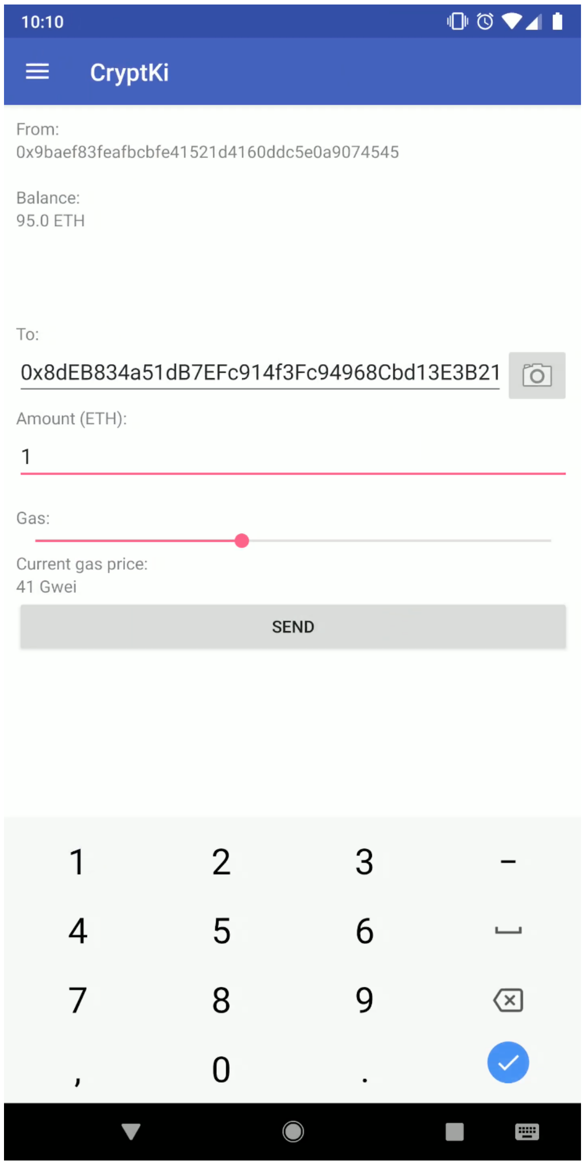


Figure 2.13: Phone Balance Screen

When the user opens the app or has received a transaction, the user can see their balance.



10:10

CryptKi

From:
0x9baef83feafbcbfe41521d4160ddc5e0a9074545

Balance:
95.0 ETH

To:
0x8dEB834a51dB7EFc914f3Fc94968Cbd13E3B21

Amount (ETH):
1

Gas:

Current gas price:
41 Gwei

Figure 2.14: Phone Sending a Transaction

On the mobile application, the user will be prompted to enter in a valid address to send funds to. After this, the user will then be prompted to enter in an Ethereum amount to transfer and the gas transaction fee.

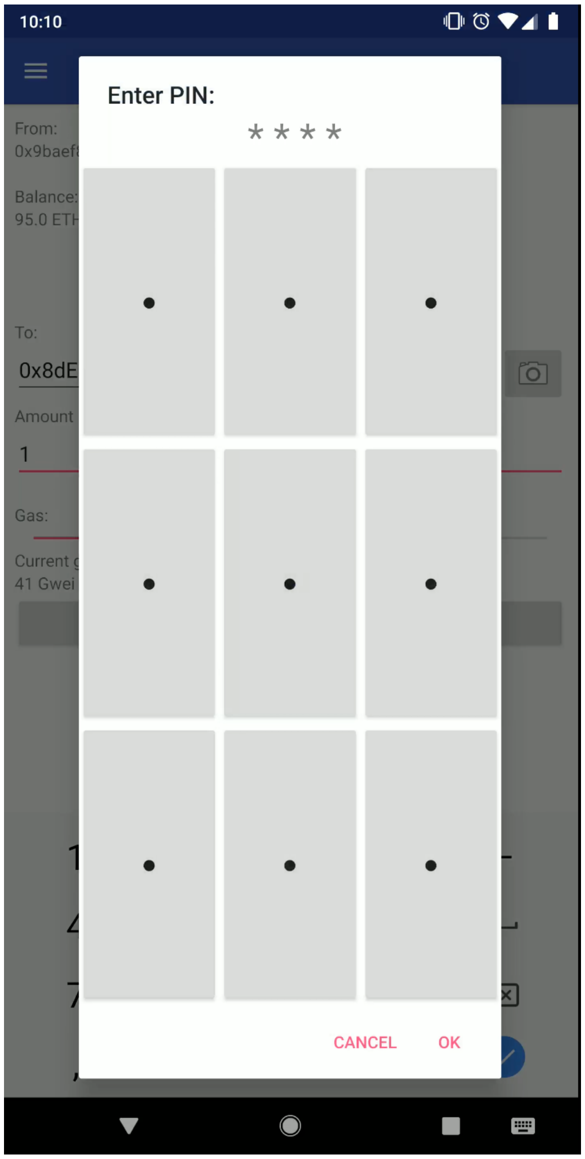


Figure 2.15: Phone PIN Screen

Before sending a transaction on the mobile application, the user will be prompted to enter a PIN that they created when they setup the device. This will ensure an extra level of security by preventing other people with access to your phone from sending funds.

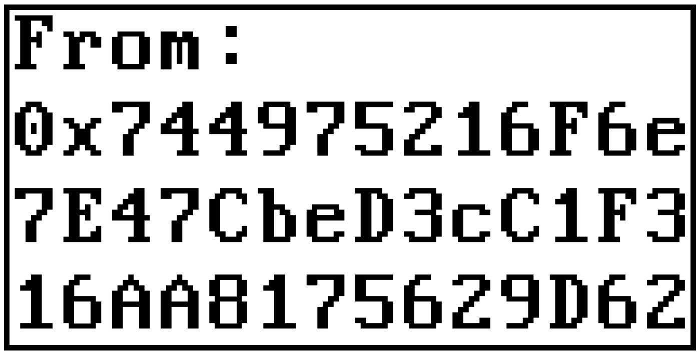

From:
0x744975216F6e
7E47CbeD3cC1F3
16AA8175629D62

Figure 2.16: Transaction From

When the user initiates a transaction from their phone, this screen will automatically be displayed on the device. Inspecting the address will allow the user to filter out any transactions they did not initiate. Pressing any button moves to the next screen.

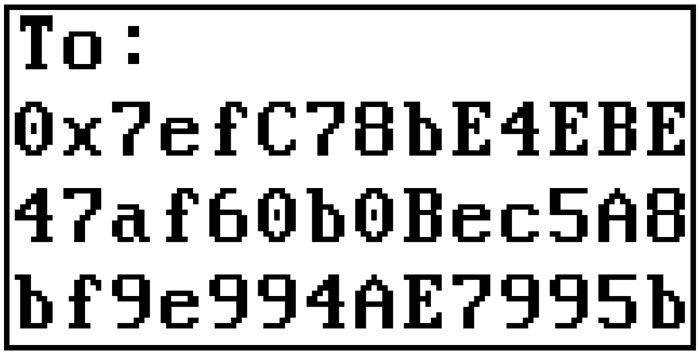

To:
0x7efC78bE4EBE
47af60b0Bec5A8
bf9e994AE7995b

Figure 2.17: Transaction To

The recipient will be displayed for confirmation. Pressing any button moves to the next screen.

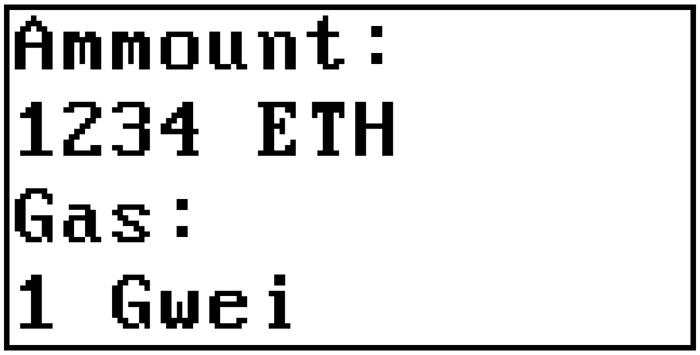

Ammount:
1234 ETH
Gas:
1 Gwei

Figure 2.18: Transaction Amount

The amount and gas (transaction fee) will then be displayed. Pressing any button moves to the next screen.

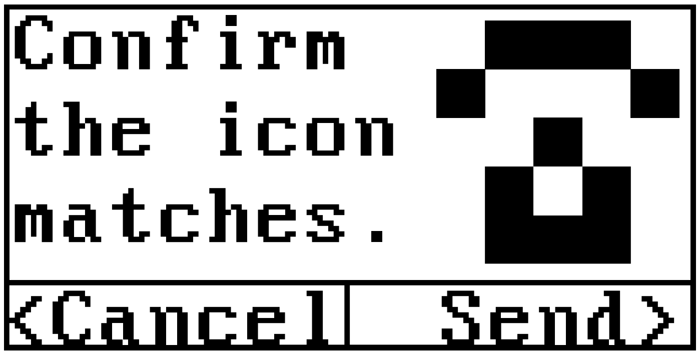


Figure 2.19: Transaction Confirm

At this point the user will be able to cancel or send the transaction. As an added level of security a verification image will be displayed allowing for quick visual confirmation. If the image matches the one generated on the phone then the information has not been tampered with. Pressing left cancels the transaction, pressing right sends it.

Figure 2.20: Sent Screen

When the user confirms the transaction, the sent screen will be shown.

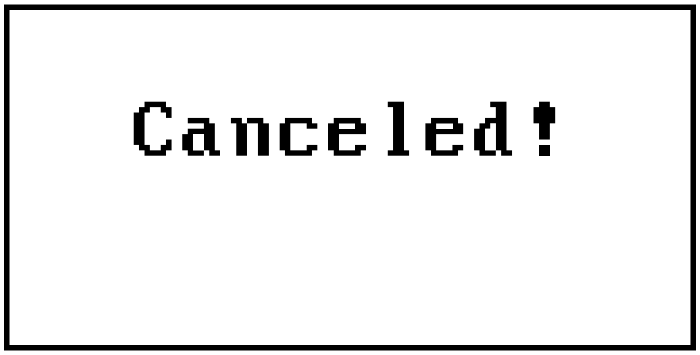


Figure 2.21: Canceled Screen

When the user denies the transaction, the canceled screen will be shown.

2.5 Architectural Diagram

This section describes how the individual software and hardware components interact to form the hardware wallet. The hardware wallet design is described in the block diagrams.

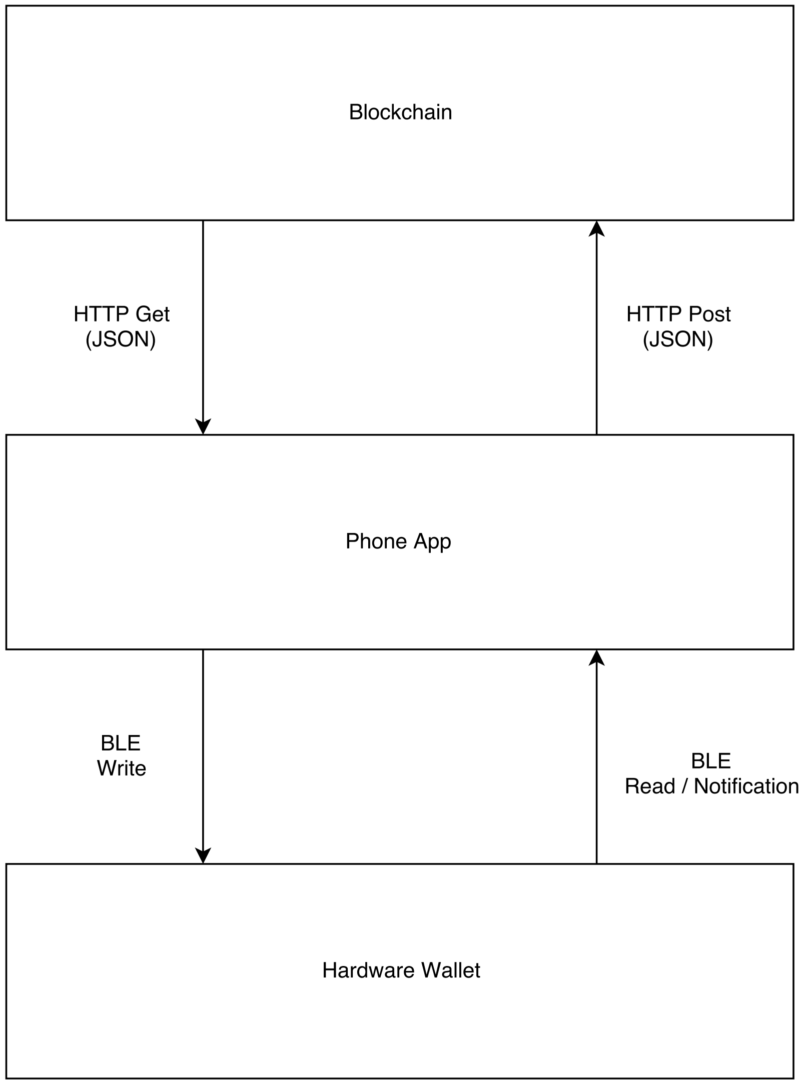


Figure 2.22: System Architectural Diagram

The system level architecture is described by figure 2.22. This diagram shows how each component interacts with every other component. The blockchain component is a server backend running Parity. Parity is a server software designed to communicate with the Ethereum blockchain. The phone app acts as a communication layer between the hardware wallet and the blockchain. The main function of the phone app is to provide a GUI for the user as well as sending and receiving data from the blockchain. The final component is the hardware wallet. The hardware wallet receives transaction information from the phone app via a BLE connection. The hardware wallet will then generate and sign a valid Ethereum transaction. Next, the wallet will send that transaction to the phone and then the phone will send the transaction to the blockchain.

2.6 Phone App

Originally Phone apps will be implemented for iOS as well as Android. Despite the different implementation methods, the development team plans on utilizing the same software architecture for both. The phone app will use a model view controller architecture to implement the functions described in the requirements section. A model view controller architecture is beneficial here because each of the functions are compartmentalized and are focused on one task.

The app itself will act as the controller for the two data models. If the user requests for a transaction to be submitted to the network, the phone app will send the required transaction information to the hardware wallet. The hardware wallet will then respond back with a valid Ethereum transaction. The phone app will then send that information to the blockchain and respond back to the user. By using a model view controller, the user will have a smooth experience.

The models in this system represent Ethereum data. The app communicates with the different models by using JSON remote procedure calls (RPC) and BLE. JSON RPC is a standardized method to communicate with servers. In this context, the phone app would utilize RPC to tell the server to send a message to the Ethereum blockchain. The message can range from sending transactions to getting network status. The phone app utilizes BLE to send transaction information to the hardware wallet. The wallet's responsibility is to manipulate that data and return a valid Ethereum transaction.

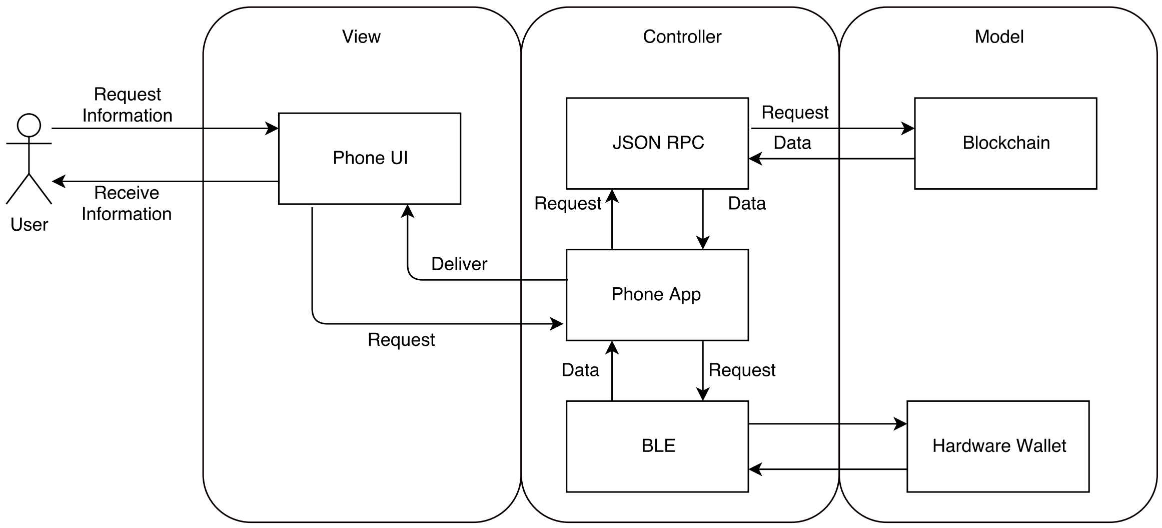


Figure 2.23: App Software Architecture

2.7 Block Diagrams

2.7.1 Level 0

This diagram describes overall diagram of hardware wallet. At a surface level, the hardware wallet takes in transaction information, which include: the amount of Ethereum to send, the address of the recipient, the gas amount used in the transaction, and a pin number to confirm the transaction. As a result of a successful pin match, a signed transaction with appropriate values are produced from the hardware wallet which will be sent back to the phone and then to the network.

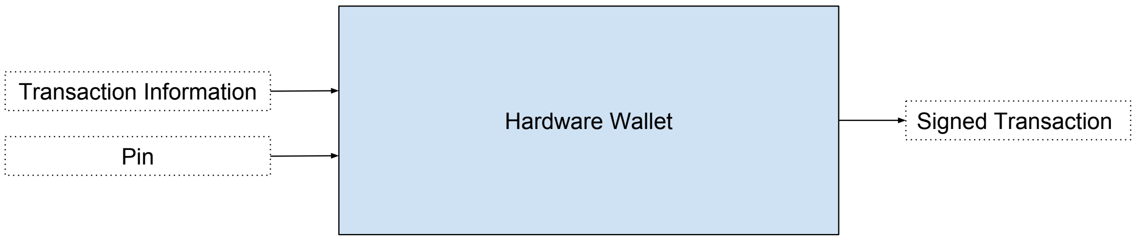


Figure 2.24: Level 0 Block Diagram

Table 2.1: Services

Module	Hardware Wallet
Inputs	Transaction info (68+ bytes) Pin (4-6 bytes)
Outputs	Signed transaction (100 bytes)
Functionality	Will sign a transaction

2.7.2 Level 1

This diagram describes the major components of the hardware wallet PCB. At its core is a microcontroller responsible for all computations. The microcontroller is connected to a secure element to retrieve keys for the wallet, and a wireless transceiver to transmit and receive information from the phone. It is possible for all three of these blocks to be contained within one SOC (System On a Chip). The microcontroller will receive user input via buttons attached over GPIO and display output on a screen connected to the SPI bus. A power supply will power the whole board.

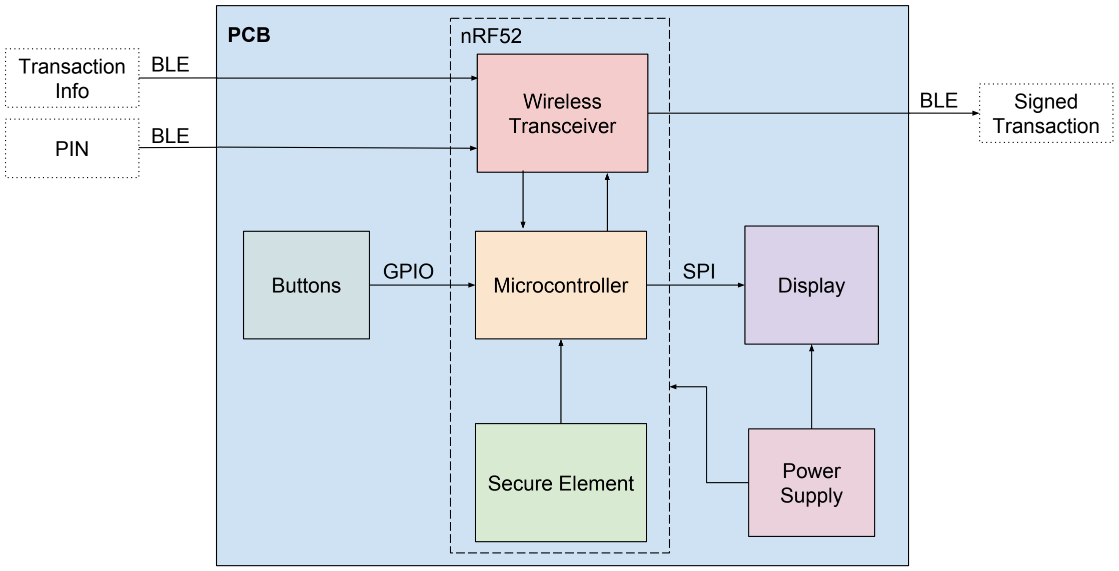


Figure 2.25: Level 1 Block Diagram

2.8 Cryptography Block Diagram

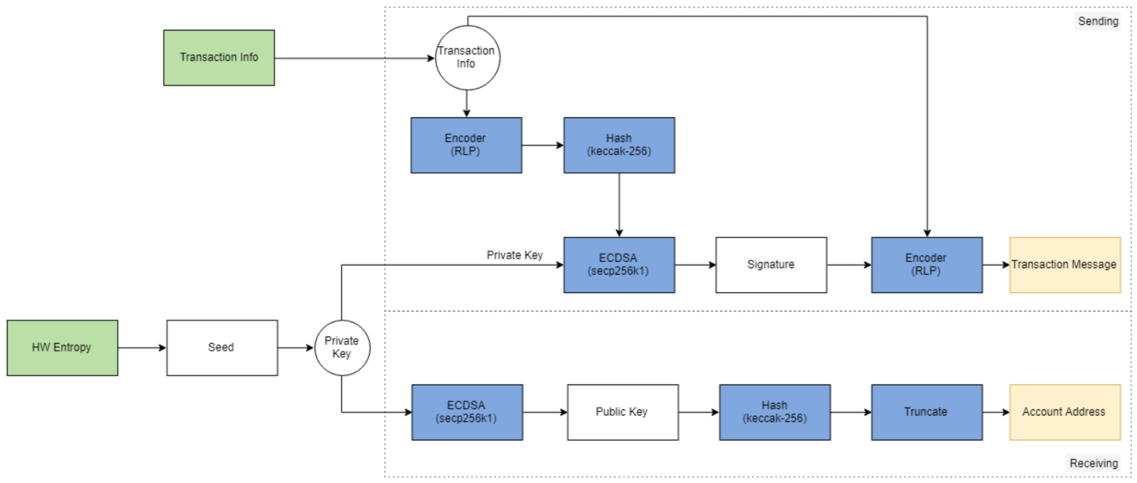


Figure 2.26: Cryptography Block Diagram

This implementation is the standard for all Ethereum compatible clients to use. Any modifications to the transaction generation or public address generation would create incompatibilities. The one area of possible modification would be to add additional steps between the HW Entropy and the Private Key. For instance in the future we could implement BIP32 to create Hierarchical Deterministic Wallets, allowing for multiple currencies off of a single seed.

2.9 Design Rationale

This hardware wallet is essentially a secure way to complete a transaction between the user and the blockchain. There were many options to choose from and those that were chosen were done so after on careful review of pros and cons of each.

2.9.1 Support for Multiple Cyrptocurrencies

For this project, we are initially targeting only Ethereum. Consideration was given to developing a mobile wallet for other cryptocurrencies like Bitcoin but we ultimately chose to focus on Ethereum due to the stability of the platform and the technology.

2.9.2 Localization

For the moment we are targeting English for our proof of concept as it is the main language of the group members. In the future both the app and hardware wallet could be translated into multiple languages.

2.9.3 Connecting to the Ethereum Blockchain

There are two ways to connect to the blockchain, a full node and a light node. Connecting via a full node provides the most up to date information because a full node stores the entire blockchain on the local machine. A light node stores just a portion of the blockchain on the local machine.

For a mobile application, a full node is impractical to implement due to the fact that an entire Ethereum blockchain is about 12 GB big, which is an unreasonable amount of data to store in a smart phone. Additionally, the smart phone would constantly need to stay synced and connected to other peers on the Ethereum network. These features are fine for a dedicated server but would negatively impact a smartphone user's experience.

A light node is better suited for a mobile application because of the way the technology stays up to date with the Ethereum blockchain. A light node only keeps track of data that the node deems relevant. In this scenario, a user can track a list of wallets and keep information about those wallets up to date. However, like the full node, the light node needs to be connected to peers in order to stay current.

Instead of running a full node or a light node on the user's smartphone, we will utilize a remote node. With this architecture, each instance of the mobile application will connect to the same server backend. The benefit of this configuration is that the server will constantly be connected to the Ethereum network and therefore will remain up to date with the current blockchain state. Furthermore, the main storage needs will be on the server instead of the mobile application. For the more privacy conscious, users will be able to host their own nodes.

2.9.4 Mobile Application Connection with Hardware Wallet

There are numerous ways to connect devices together. The other hardware wallets connect to a computer via a USB connection. For a mobile wallet, a wired connection is not practical. No consumer wants a wire connected to their phone whenever they want to make a transaction. The better solution is to use a wireless connection.

WiFi has the issue of changing IP addresses making it difficult for the phone and wallet to communicate over changing network conditions. Additionally there may be situations where the user is not near a wifi hotspot.

NFC is short range making it infeasible to do longer activities such as entering a PIN without moving the devices too far apart.

Bluetooth is a better solution as the phone and wallet create a persistent bond that does not rely upon other devices. Bluetooth is a commonly adopted standard and all major smartphones support the Bluetooth standard.

2.9.5 Power

In order to power the hardware wallet we will need some form of battery power. Alkaline batteries are too large for our portable form factor. Traditional Lithium Polymer (LiPo) batteries are also fairly large and require charge protection circuits. For these reasons we have opted to use a coin cell battery as they are small, cheaply available, and supply the required 3V to power our circuits. As an additional benefit the battery's discharge curve fits nicely within the acceptable range for both our microcontroller and screen meaning that we do not require any voltage regulators, reducing both cost, pcb space, and current draw.

2.9.6 Interacting with the Hardware Wallet

Because we want the user to be able to confirm transactions on the hardware wallet we need some method for the user to interact with the device. The competitors, Trezor and Ledger, both use two buttons to accomplish this. Unfortunately this leads to awkward sequences for the user. For instance, pressing both buttons at the same time to navigate to the next didgit. We believe that four buttons arranged in an up, down, left, right configuration will provide the most comfortable navigation between screens without taking up too much physical space.

2.9.7 BLE Services

The bluetooth services will be split by use. A main device service will be responsible for unlocking the device (by transmitting a pin) and changing settings such as the brightness.

A second service dedicated to the ethereum client will be created. This service will allow creating an account, listing accounts, and sending transactions. In the future additional services for more cryptocurrencies could be added.

Each service and characteristic must have a unique ID. To accomplish this we have a randomly generated 128bit vendor specific UUID listed in Appendix F.

As bluetooth can only send binary strings, we require structured data so we are utilizing msgpack. This provides a compact way to send arrays, dictionaries, strings, and numbers all in a single binary stream.

2.9.8 Security

Our main security is keeping the wallet's private keys, off phones and computers. By keeping them contained on a separate device running no other processes there are much fewer attack vectors. We are also enforcing a mandatory PIN code thus making any transactions require something you have (the wallet) and something you know (the pin). The device will automatically wipe itself after a number of incorrect PINs have been entered adding protection against thieves. In the event that the device is stolen the backup code created during setup can be used to restore the account to a new device preventing loss of funds. We have also enabled Man In the Middle (MITM) protection for our bluetooth connections helping to prevent interception, and as an additional layer display a checksum image that would no longer match if the data was somehow tampered with. No transactions are sent without explicit user interaction preventing any unauthorized use by other apps or malware. We will be deploying as many security technologies as possible on the device including signed firmware, and stack protection. We will use preexisting cryptographic libraries to reduce the chance of inadvertently adding implementation flaws.

2.10 Technologies Used

This section will describe the technologies used in this project and will be divided into three subsections. For all coding aspects, we will utilize Github to enforce version control.

2.10.1 Server Backend

The server utilizes Parity to run an Ethereum full node. Parity is written in Rust and provides the full functionality to communicate with the Ethereum blockchain. The main functionality that the project utilizes is the JSON RPC interface to communicate with the backend. Using Parity enabled us to focus on building the hardware wallet as well as the associated application because all of the server communication links are already built for us.

2.10.2 Mobile Application

The mobile application was initially be targeted toward Android users. For the Android app, we utilized Android Studio as the main IDE. Like Xcode, Android Studio has different emulation modes, documentation, as well as APIs. The main testing platform that we will use will be a Google Pixel 2 XL. Android Studio has two main languages: Java and Kotlin. We opted to use Kotlin as it provides easier access to null safety checks and other useful features that plain java does not provide.

2.10.3 Hardware Wallet

Components

We chose to use Nordic Semiconductor's NRF52 series chips for our microcontroller. These devices have a high clock speed, large amount of flash, and most importantly Bluetooth, and NFC support. Additionally the chip has a hardware random number generator and acceleration for many common cryptographic functions which will be useful for our project.

For the display we chose to use a small OLED screen. This provides a high degree of contrast increasing visibility and also features a lower current draw than other technologies. Because the screen uses SPI we are able to have quick redraws if necessary.

Software

To design our schematic and pcb we used Kicad in order to keep all aspects of our project open source. For the same reason we used the standard GCC toolchain to compile the firmware allowing development on any platform with no fees. We are using Nordic's software libraries to ease the firmware development.

Tools

For debugging and firmware upload we are using the J-Link Edu. Test revision boards were produced off campus.

2.11 Development Process

2.11.1 Hardware Wallet

Circuit Design

In keeping would our open and accountable spirit we designed our PCB in KiCAD, an open source electronics design suite [5].

Controller

The heart of this project is a microcontroller responsible for providing entropy to generate private keys, persistently storing keys, and computing cryptographic signatures to create transactions. Instead of also requiring a separate microcontroller for bluetooth communication, we leveraged the fact that Nordic provides a series of chips that contain all the required functionality in a single package. The specific chip we utilized was the NRF52840. [6]. As an added bonus the S140 Soft Device is prettified which will ease FCC testing if we apply for certification [7]. To integrate this chip into our design we utilized their reference design with a few minor modifications to ease manufacturing such as increase tolerances [8]. To make sure our modifications would not negatively impact the operation of the chip all of the PCB guidelines were followed [9]. The full schematic and PCB can be found in Appendix D. To program these chips we purchased a J-Link which is capable of flashing most ARM chips [10]. In order to spead up development while waiting on these PCBs to be produced and delivered we opted to order Nordic's Development Kits as a known working implementation for the software to be built upon [11].

Screen

In order to display the transaction information clearly we need a sharp screen with high contrast. Organic LED screens fit this roll perfectly with their 6500 nits of brightness. We decided specifically to use an SSD1306 based display due to their cheap price and wide availability [12]. Many of these screens feature a SPI interface allowing quick communication with microcontrollers. To interface with the screen without requiring a separate breakout board, the supporting hardware was integrated into our schematic [13]. The software protocol was determined from the chip's data sheet.

Power

The device is powered with a single lithium-ion coin cell battery. These batteries provide a nominal 3V power and drop off voltage of 2V which falls squarely in line with the acceptable input ranges for all the ICs we used. Because of this no voltage regulation circuitry needed to be designed.

Bluetooth

The first step to developing a bluetooth protocol was creating a unique 128 bit vendor ID that can be used for all of our functionalities. At the basis is a that is used to uniquely identify our devices. From there two services were created, one to managed the device (such as setup and unlocking) and another for managing ethereum wallets (listing accounts, and sending transactions). Both of these services have unique addresses based upon the vendor ID. In the future more services could be created for other cryptocurrencies or features. Each service has one or more characteristics that can be written to or read from. Unfortunately these characteristics act as raw byte streams and have no formatting. To add formatted data ontop of this protocol we implemented MessagePack, a serialization and deserialization protocol [14].

Firmware

Once we had working hardware and an established protocol, work began on the firmware. Most of the device firmware utilizes the NordicSDK and is based upon its example projects [15]. The micro-ecc library is required for some of its features [16]. The first order of business was getting bluetooth advertising working. After the device showed on our phones then work on implementing the new services began. The next step was getting text to display on the screen, followed by handling button presses. At this state we could then enter a pin. After that we had to get flash storage working so that the entered settings would be remembered between power losses. At this point the device could be successfully setup. The final part of the firmware development was the actually sign transactions. For the cryptography we utilized the trezor-crypto repository due to its many algorithms in the C language, large amount of testing, and real world usage [17]. In order to create Ethereum transactions we had to create functions that encoded and decoded RLP formatted messages. Our implementation follows the psuedocode found in the Ethereum Wiki [18]. After implementing these libraries we could then compute signatures.

2.11.2 Android App

UI

At first the App used mocked data in order to allow the UI to be developed while the firmware was in progress. The Android App follows many of the best practices, including using the Support Library to provide a consistent UI between platform versions [19]. We also utilize some newer libraries such as LiveData to keep the displayed information and backend model in sync [20]. To ensure that users don't type invalid transaction information such as malformed addresses or negative amounts, we added in the DataValidator library [21]. To persistently store data such as account icons, colors, and other metadata the Room library was used [22]. The ZXing library was used to handle the scanning of QR codes for ease of entering recipients [23]. The MaterialDrawer library provided the basis of navigation and account selection, and the ColorPicker library allowed choosing unique colors for each account [24] [25]. Finally, Splitties was used to provide miscellaneous niceties such as a simpler interface to preferences [26].

Bluetooth

After the firmware was in a working state, the bluetooth communication code began to be integrated. The BleGattCourotines library was chosen for our Bluetooth backend as it provides protection against race conditions and other issues that would result in messages otherwise not being delivered [27]. MsgPack-Java was used to provide the android side of the Msgpack protocol used to communicate with the hardware wallet [28]. After implementing these two libraries and using the specific addresses decided upon for our protocol, the communication with the hardware wallet was working and could retrieve real data from the device.

Ethereum Network

After getting communication with the wallet working, the final step was talking to the remote node. In order to broadcast transactions to the network we hosted a Parity node. This node has an open port that accepts formatted requests on the users behalf [29]. The communication between the phone and remote node is accomplished through Google's Volley library [30]. Together, Volley and Parity allowed us to retrieve account information, submit transactions to the network, and more. At this point we had a fully working demo.

2.12 Attack Vectors and Mitigations

This section details the different possible attack vectors for a mobile cryptocurrency wallet. The design group created the wallet with these considerations in mind to help create a more secure ecosystem. Without these considerations, massive loss of user trust and financial loss can occur. Most of the systems that we have implemented are variations of two factor authentication. The main reason for this design decision is because two factor authentication is a tried and true way to deter and protect against most attacks.

However, there are concessions to be made here. If an attacker has physical access to the hardware wallet, a sophisticated enough attacker can probably derive a private key from the internals of a wallet though power analysis or modifying the actual hardware in the wallet. Additionally, this system cannot protect against so called "rubber mallet" attacks. In this form of attack, the attacker threatens the user with physical harm (such as with a rubber mallet) in order to get the user to give up pin numbers or access codes.

The following sections describe how the design team responded and mitigated known attack vectors.

2.12.1 Pin Access

One of the biggest attack vectors that must be mitigated is if the user's pin is compromised. If the user's pin is compromised, it is possible for an attacker to initiate transactions with the hardware wallet. This may occur in a number of ways, the primary way being an attacker watches the user input the pin. This is mostly mitigated by having the mobile application provide a randomized number pad when inputting the user's pin [31]. This solution mitigates the problem by ensuring that the attacker cannot simply memorize the pattern that the user inputs. Overall, this adds an additional layer of security because the attacker needs to know the actual pin number rather than just observing the user.

Furthermore, if a user's pin is compromised, the attacker would also need access to either the mobile phone and the hardware wallet. Due to how the system is built, a transaction cannot be completed without both components. So, in addition to stealing your pin, the attacker would also have to steal the hardware wallet in order to steal funds from that particular hardware wallet.

2.12.2 Man in the Middle Attacks

Since we are utilizing BLE to transmit data, it is possible for an outside authority to inspect the data that is sent between the mobile phone and the hardware wallet. However, BLE sends encrypted packets by default, which adds a layer of security to the exchange. Furthermore, sensitive data is never transmitted over BLE. The two pieces of sensitive data are the Ethereum account's private key and the user's pin number. The data that is sent are the various transaction details such as, transaction destination, amount, nonce, etc. The data here is not inherently sensitive, but if the data is changed the desired outcome changes.

This brings us to the second aspect of a man in the middle attack. It is technically possible for the attacker to intercept a BLE packet mid flight, change the contents, then forward it to the original destination. Our system mitigates this problem by implementing two factor authentication for all transactions. The transaction request originates from the mobile device and those transaction details are sent to the hardware wallet. If the details are changed mid flight, the wallet will display the wrong transaction details. The user will then simply deny the transaction request. If the signed result is tampered with between the wallet and the phone, the the signature will no longer be valid, and the transaction thus void. This protects against any modification of the recipient, amount, or other transaction details from 3rd parties or the phone once the details have been confirmed on the wallet's screen by a user.

In the same vein, TREZOR and Ledger used to have a problem called a vanity attack. In this attack, the malicious actor would generate a new Ethereum address and match the last eight or so digits with the user's intended destination address. As a result, because the TREZOR and Ledger used to display only the last 8 digits of an address during the transaction verification it was possible for the user to send the funds to the wrong account. Albeit, only if the attacker was capable of intercepting the transaction request and changing the details before the user realized. A single vanity attack only takes a couple of hours to compute an address with the same 8 last digits as the target address [32]. The way to mitigate this type of attack is to display the full destination address in order to ensure to the user that that is the intended address. Another thing to note is that in order to spoof more than 8 digits, the computation time and cost will increase exponentially. Therefore, it is sufficient to display 12 of the 20 digits of the destination address because an attacker simply does not have the time or the resources to spoof all 12 digits.

2.12.3 Power Attacks

This particular section comes from a voltage analysis via USB of a Trezor wallet performed by Dr. Jochen Hoenicke [33].

If the user loses physical access to the hardware wallet, little can be done to prevent an attacker from stealing funds. In this example, the attacker managed to derive the private key of an address by examining the power consumption done during key generation. One way we mitigated this issue was by ensuring that no cryptographic functions occur if the device is locked. Another method to mitigate this attack is employing power smoothing hardware that masks the waveform and power consumption.

Power attacks are sophisticated attacks because it requires direct physical access to the hardware wallet and specialized equipment. Essentially, a power attack is an attempt to peer into the black box and reverse engineer the contents within. This attack is feasible, but again, unlikely to occur. The time and effort required to acquire the private key outweighs the potential financial gains.

2.12.4 Cold Boot Attacks

This attack is a side-channel attack that attempts to read data from physical memory. This is possible because even though physical memory is volatile, the stored data does not immediately decay upon loss of power. If an attacker uses this method on our hardware wallet, it is possible for the attacker to derive the private key. However, the attacker would need physical access to the device, will require the device to be unlocked, and specialized equipment. We did not explicitly create a mitigation method for this problem due to the low likelihood of this attack happening. One way to mitigate this attack is to encrypt all memory contents, however, the device would require more power and slow down due to the additional instructions necessary to encrypt/decrypt every operation requiring memory. Additionally, the decryption key would need to be stored in memory, so for this scheme to be viable, the processor must load the decryption key, decrypt the data, do an operation, then reincrypt the data. This is an impractical solution because an attacker can simply cut power when the data is decrypted.

Lastly, a more practical solution is to simply prevent any operation unless the device is unlocked. If the device is unlocked then it most likely means that the user is around the device and the device is "safe". However, this is not a perfect solution because if the pin is compromised, the hardware wallet would start doing operations and therefore be susceptible to a cold boot attack.

2.13 Test Plans

This section will detail how the different software and hardware components were tested. The server used a test blockchain to maintain a clean testing platform. The main benefit of a test blockchain is that no other users can use it. Other people can not add transactions or add network load. This keeps the network in a deterministic state.

2.13.1 Mobile application

The mobile application has a couple of distinct modules that was tested. One of the main modules is the code that will handle the various remote procedure calls (RPC). The various RPC were tested using unit testing. Each call can be individually tested because there is documentation on expected input/output for each RPC.

Another module that the app has is a BLE communication module. The main testing that was done was to ensure accurate transmission of data. Fortunately, the Bluetooth stack has error recovery which was used to make sure that the app can communicate with the wallet. Again, the main testing here is unit testing because each of the BLE communication methods needed to be checked and verified for accurate data.

Another aspect that was tested is the UI. There are testing frameworks built into Android and were utilized to test the mobile app. The purpose of these tests was to ensure that the UI behaved as expected. Furthermore, we tested the app using blackbox testing and as if we were an end user using the application.

2.13.2 Hardware wallet software

The hardware wallet has a different suite of testing procedures that was done. Firstly, the wallet was tested to generate Ethereum public/private keys. The testing procedure was done by writing unit tests. The desired outcome for the tests were determined by the public/private key generation done by official Ethereum clients. Other unit tests that were done were the mnemonic phrase tests. The wallet needed to be able to generate these phrases accurately and must be able to display these accurately.

The wallet settings need to be tested as well. Specifically, factory reset, display brightness, and Bluetooth connectivity needs to be tested and verified to be working.

2.13.3 System Testing

After testing each independent component, full system testing will occur. The testing procedure needs to include sending and receiving information directly from the blockchain. This can be done by writing an initialization file for the blockchain. The initialization file in conjunction with wiping the test blockchain ensures that all of our tests will be run from the same starting point. We can also add scripts to populate the test blockchain with initial data to simulate real conditions.

Chapter 3

Discussion

3.1 Social Concerns

The creation of our hardware wallet will undoubtedly raise social concerns about our product. Specifically, the security of the device and the technologies used in securing funds. Our device aims to be as secure as possible with multiple security features to protect from various attack vectors. The wallet acts as a part of two-step authentication and will provide security for a user's funds.

3.2 Economics Considerations

Introducing our hardware wallet for cheaper price can create more competition in the hardware wallet market which are sold at relatively high prices. This competition can help drive prices for hardware wallets down which makes them more accessible to the general public. Making the product also allows everyone to make online transactions safer in the midst of controversy about the safety of the cryptocurrencies where companies with low securities had their cryptocurrency funds stolen. This step towards more safety in cryptocurrencies can further promote the use of such currency knowing that it will not only be convenient, but also safer.

3.3 Usability

One of the main features for our hardware wallet is mobility. Most existing hardware wallets works via a connection to a laptop or desktop through a USB cable. This means transactions can only be made with a computer, which can be inconvenient at times. If a user wants to use cryptocurrencies on the go, then the hardware wallet must be portable and powered on its own for better usability which our hardware wallet accomplishes. Considering everything is moving towards digitization and portability is becoming a key factor into our daily lives, there is a high chance that cryptocurrencies will one day be integrated into daily use at coffee shops, restaurants, etc. That is why our group decided to increase the usability by using coin battery to power the hardware wallet, which will last for hours of use. Each transaction will only take seconds of use at a time, so this battery is more than enough to power it for very long time.

3.4 Manufacturing

Our implementation of a hardware wallet uses Nordic Semiconductor's NRF52 series chips as a microcontroller. This microcontroller fulfills all of our needs while being cheap to purchase. Our design also allows for software updates without a need to purchase an entirely new hardware wallet. The wallet also features a standard Litium Polymer battery which will provide hours of use while also being cheap and readily available.

3.5 Lifelong Learning

Working on the hardware wallet has prepared us for the future of cryptocurrencies. The group understands how blockchain and Ethereum work and its potential in the future. The group also understands the various security features implemented in the hardware wallet.

Chapter 4

Conclusion

4.1 Lessons Learned

The biggest lesson learned was learning how to integrate multiple discrete systems to work well together. During our implementation, each individual system worked fine but combining the three modules caused things to break in unexpected ways. This problem was mostly mitigated through good coding practices and open communication.

Specifically, we learned a lot about the functions in the Bluetooth Protocol. We learned the limitations of Bluetooth as well as how different devices handshake and communicate with one another. We learned a lot about how the Ethereum network signs, validates, and mine transactions as well as how to communicate with the blockchain. Again, communicating data and systems to different modules was the main lesson learned because all of these amazing systems exist in the world already, it is up to new developers to understand the old system so that a new system can interlink and build.

4.2 Future Development

This project could turn into a business. Given the right resources and project guidance, the hardware wallet can be deployed to communicate with the real Ethereum network. Before that happens, there are multiple aspects that can be improved on this project. For instance, an iOS app should be developed so that almost all smartphone users can be included. Improvements on the PCB and manufacturing can be made so that this can become a commercial product as well as testing the hardware wallet on our custom PCB instead of using Nordic's testing kit. Additionally, more firmware can be added so that the mobile wallet can support different cryptocurrencies like Bitcoin. Multiple account support for Ethereum could also be implemented for the hardware wallet. Future development work will be posted on `CryptKi.com`

4.3 Summary

In this project, we built a mobile hardware wallet application that is capable of communicating with the Ethereum blockchain. The project can send transactions and other, independent clients on the blockchain can verify that the transaction that we generate is a valid transaction. There are two distinct parts that we created, the mobile application and the mobile hardware wallet. The mobile application can view transactions, initiate a new transaction, and explore the Ethereum blockchain. The mobile application implements various security features in order to maintain data integrity such as pin randomization and two factor authentication.

The mobile hardware wallet is a hardware based solution that uses an nRF-52840 microprocessor from Nordic Semiconductor. The wallet was programmed with C and implements the algorithm to generate new Ethereum wallets. The wallet is capable of storing the private key for Ethereum accounts and is capable of verifying transactions and sending transactions received through BLE. The user is also able to recover a private key. Upon address creation, the wallet generates a mnemonic code for the user. The hardware wallet implements the features set forth by the requirements.

Overall, the system is fairly robust with few known security risks. Further penetration testing must be done to determine how secure the hardware wallet really is. The current system has multiple ways to maintain security while not being cumbersome to the user. Simple two factor authentication and data integrity measures are enough to mitigate all but the most sophisticated attack vectors.

Bibliography

[1] http://www.ethdocs.org/en/latest/.

[2] https://www.coindesk.com/information/what-is-ethereum/.

[3] TREZOR, "Official TREZOR Shop," https://shop.trezor.io.

[4] Ledger, "Ledger Wallet - Products list: hardware wallets and accessories," https://www.ledgerwallet.com/products.

[5] "KiCad EDA," http://kicad-pcb.org/.

[6] Nordic Semiconductor, "nRF52840 Product Specification," http://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.0.pdf, Mar. 16, 2018.

[7] ——, "S140 SoftDevice Specification," http://infocenter.nordicsemi.com/pdf/S140_SDS_v1.1.pdf, Mar. 20, 2018.

[8] ——, "nRF52 Series Reference Layout," http://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.nrf52%2Fdita%2Fnrf52%2Fpdflinks%2Fref_layout.html, May 30, 2018.

[9] Kristin, "General PCB design guidelines for nRF52 series," https://devzone.nordicsemi.com/tutorials/b/hardware-and-layout/posts/general-pcb-design-guidelines-for-nrf52, Jul. 011, 2016.

[10] SEGGER, "J-Link / J-Trace User Guide," https://www.segger.com/downloads/jlink/UM08001, May 24, 2018.

[11] Nordic Semiconductor, "nRF52840 Preview Development Kit User Guide," http://infocenter.nordicsemi.com/pdf/nRF52840_PDK_User_Guide_v1.2.pdf, Jan. 19, 2018.

[12] Solomon Systech Limited, "128 x 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller," https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf, Apr. 2008.

[13] Rene van der Meer, "SSD1306 1.3 OLED SPI breakout board," https://blog.oshpark.com/2017/07/25/ssd1306-1-3%E2%80%B3-oled-spi-breakout-board/, Jul. 25, 2017.

[14] https://github.com/ludocode/mpack.

[15] "Software Development Kit for the nRF51 Series and nRF52," https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF5-SDK.

[16] https://github.com/kmackay/micro-ecc.

[17] https://github.com/trezor/trezor-crypto.

[18] https://github.com/ethereum/wiki/wiki/RLP.

[19] “Support Library,” https://developer.android.com/topic/libraries/support-library, May 8, 2018.

[20] “Livedata Overview,” https://developer.android.com/topic/libraries/architecture/livedata, Jun. 12, 2018.

[21] https://github.com/Ilhasoft/data-binding-validator.

[22] “Save data in a local database using Room,” https://developer.android.com/training/data-storage/room/, May 15, 2018.

[23] https://github.com/journeyapps/zxing-android-embedded.

[24] https://github.com/zsmb13/MaterialDrawerKt.

[25] https://github.com/TurkiAlkhatib/Material-Color-Picker.

[26] https://github.com/LouisCAD/Splitties.

[27] https://github.com/Beepiz/BleGattCoroutines.

[28] https://github.com/msgpack/msgpack-java.

[29] Parity Ethereum Documentation, “JSON RPC API - Wiki,” https://wiki.parity.io/JSONRPC.

[30] “Volley overview,” https://developer.android.com/training/volley, Apr. 25, 2018.

[31] SatoshiLabs, “Entering your PIN - TREZOR User Manual 1.0 documentation,” https://doc.satoshilabs.com/trezor-user/enteringyourpin.htm, May 30, 2018.

[32] K. Kreder, “Hardware Wallet Vulnerabilities,” https://blog.gridplus.io/hardware-wallet-vulnerabilities-f20688361b88, Oct. 24, 2017.

[33] J. Hoenicke, “Extracting the Private Key from a TREZOR,” https://jochen-hoenicke.de/trezor-power-analysis, Nov. 15, 2015.

[34] A. ”bunnie” Huang, “Keeping Secrets in Hardware: the Microsoft XBox^{TM} Case Study,” May 26, 2002.

[35] R. Carbone and C. Bean and M. Salois, “An in-depth analysis of the cold boot attack,” Jan. 2011.

[36] https://github.com/ethereum/blockies.

[37] Bluetooth Special Interest Group, “GATT Overview,” https://www.bluetooth.com/specifications/gatt/generic-attributes-overview, 2018.

[38] “Using Gradle - Kotlin Programming Language,” https://kotlinlang.org/docs/reference/using-gradle.html.

Chapter 5

Appendix

Appendix A

Risk Analysis and Timeline

A.1 Risk Analysis

The following table illustrates the possible risks associated with the successful creation of the hardware wallet. The table also analyzes the consequences of these risks as well as their likelihood, severity, potential impact, and strategies to mitigate each one.

Probability describes how likely the risk might happen throughout the course of developing this system on the scale of 0 to 1. Severity is a scale from 0 to 10 to describe how severe the consequence of the risk happening. Finally, impact scales from 0 to 10 describing the actual impact the risks will cause on the project. We may likely face extension in development time or incomplete functionality at the end of development as the number on this impact scale increases.

Table A.1: Risk Table

Risk	**Consequences**	**Prob.**	**Severity**	**Impact**	**Mitigation**
Bugs in code	Creates vulnerability in our code that may be used to exploit the code and data used in our code	.7	8	5.6	Create rigorous test to ensure that the code can not be broken. The code is also made open source for anyone to find and report the bugs.
Change of Ethereum protocol	The way our mobile device interact with Blockchain must be updated to meet the Ethereum standard protocol	.3	5	1.5	Make UI simple and intuitive. Make Navigation button.
Part Unavail-ability	Other similar and compatible parts must be substituted	.2	5	1	Use common high availability parts.
Manufac-turing Issues	Switch to another manufacturer	.15	4	.6	Use well known and trusted manufacturers
Data loss	Required to re-implement or re-create data that was lost	.01	10	.1	Use git or back-up services to recover from in case if data is lost
Parity develop-ment ceased	Other method must be implemented to replace the Parity, which is the method currently in use by our device, to interact with the Blockchain	.01	7	.07	Use Mist as an alternative to Parity to connect to Blockchain.

A.2 Budget

Table A.2: Budget

Item	**Price**
OSX Dev fee (x4)	$400
ARM Dev kits (x4)	$185
Bluetooth Sniffer	$50
J-Link EDU Programmer	$60
PCB Manufacturing funds	$150
Microprocessors	$50
OLED screens	$50
Batteries	$30
Passives	$20
Lightning Cables	$30
TOTAL	$1035

A.3 Development Timeline

Fall Quarter	Week1	Week2	Week3	Week4	Week5	Week6	Week7	Week8	Week9	Week10
Problem Statement	▨	▨	▨							
Project Description			▨	▨						
Determine main components			▨	▨	▨					
Funding Proposal				▨	▨					
Ethics Report				▨	▨	▨	▨			
Design Schematics						▨	▨			
Design Report								▨	▨	▨
Winter Quarter	**Week1**	**Week2**	**Week3**	**Week4**	**Week5**	**Week6**	**Week7**	**Week8**	**Week9**	**Week10**
Order Parts	▨	▨								
Revised Design Doc	▨	▨	▨							
Phone App Development		▨	▨	▨	▨	▨	▨	▨	▨	▨
Initial Board Manufacture			▨	▨	▨					
Initial Prototype			▨	▨	▨					
Conference Registration				▨						
Additional Revisions						▨	▨	▨	▨	▨
Testing						▨	▨	▨	▨	▨
Spring Quarter	**Week1**	**Week2**	**Week3**	**Week4**	**Week5**	**Week6**	**Week7**	**Week8**	**Week9**	**Week10**
Ethics Report	▨	▨								
Thesis			▨	▨	▨					
Final Design Report						▨	▨	▨		
Prepare for Design Conference									▨	▨

Figure A.1: Year Development Timeline

Appendix B

Bill of Materials

See Table B.1 on the next page.

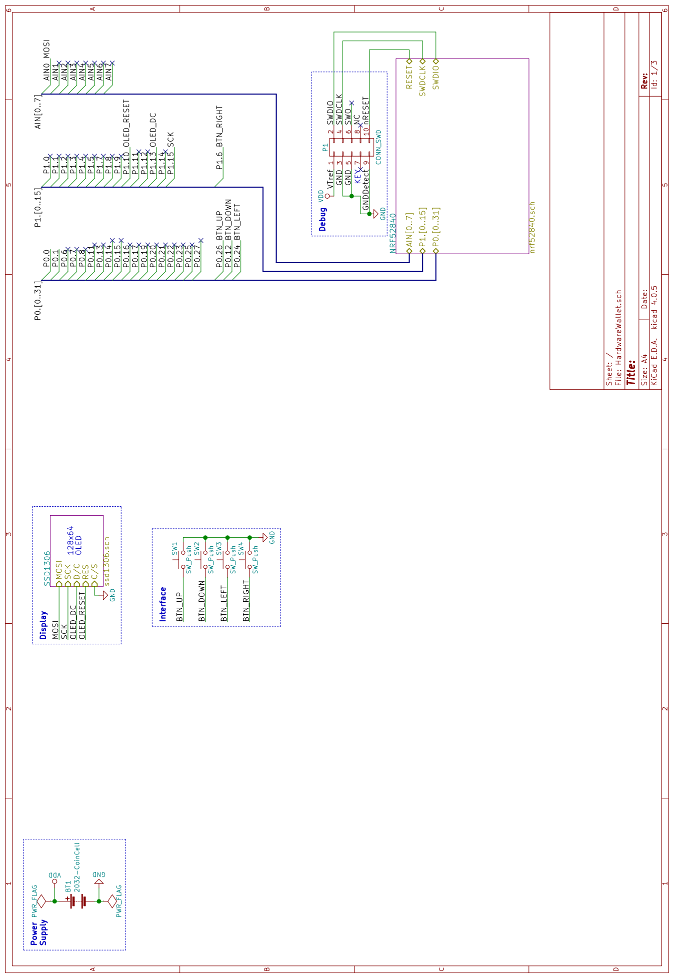
Table B.1: Bill of Materials

Reference	**Value**	**Footprint**
AEN1	NFC	
BT1	2032-CoinCell	
C1	12pF	SMD 0402
C10	N.C.	SMD 0402
C11	100pF	SMD 0402
C12	100nF	SMD 0402
C13	N.C.	SMD 0402
C14	1.0uF	SMD 0402
C15	1.0uF	SMD 0402
C16	47nF	SMD 0402
C17	12pF	SMD 0402
C18	12pF	SMD 0402
C19	1uF	SMD 0402
C2	12pF	SMD 0402
C20	1uF	SMD 0402
C21	4.7uF	SMD 0402
C22	2.2uF	SMD 0402
C23	1uF	SMD 0402
C24	1uF	SMD 0402
C3	0.8pF	SMD 0402
C4	0.5pF	SMD 0402
C5	100nF	SMD 0402
C6	4.7uF	SMD 0402
C7	100nF	SMD 0402
C8	100nF	SMD 0402
C9	N.C.	SMD 0402
Ctune1	TBD	SMD 0402
Ctune2	TBD	SMD 0402
L1	3.3nH	SMD
L2	10uH	SMD
L3	15nH	SMD
R1	390k	SMD 0603
SW1	SW_Push	SMD
SW2	SW_Push	SMD
SW3	SW_Push	SMD
SW4	SW_Push	SMD
U1	nRF52840-QIAA	
U2	ER-OLED013-1	
X1	32MHz SMD	SMD 2016 4Pads
X2	32.768kHz	SMD 3215 2Pads

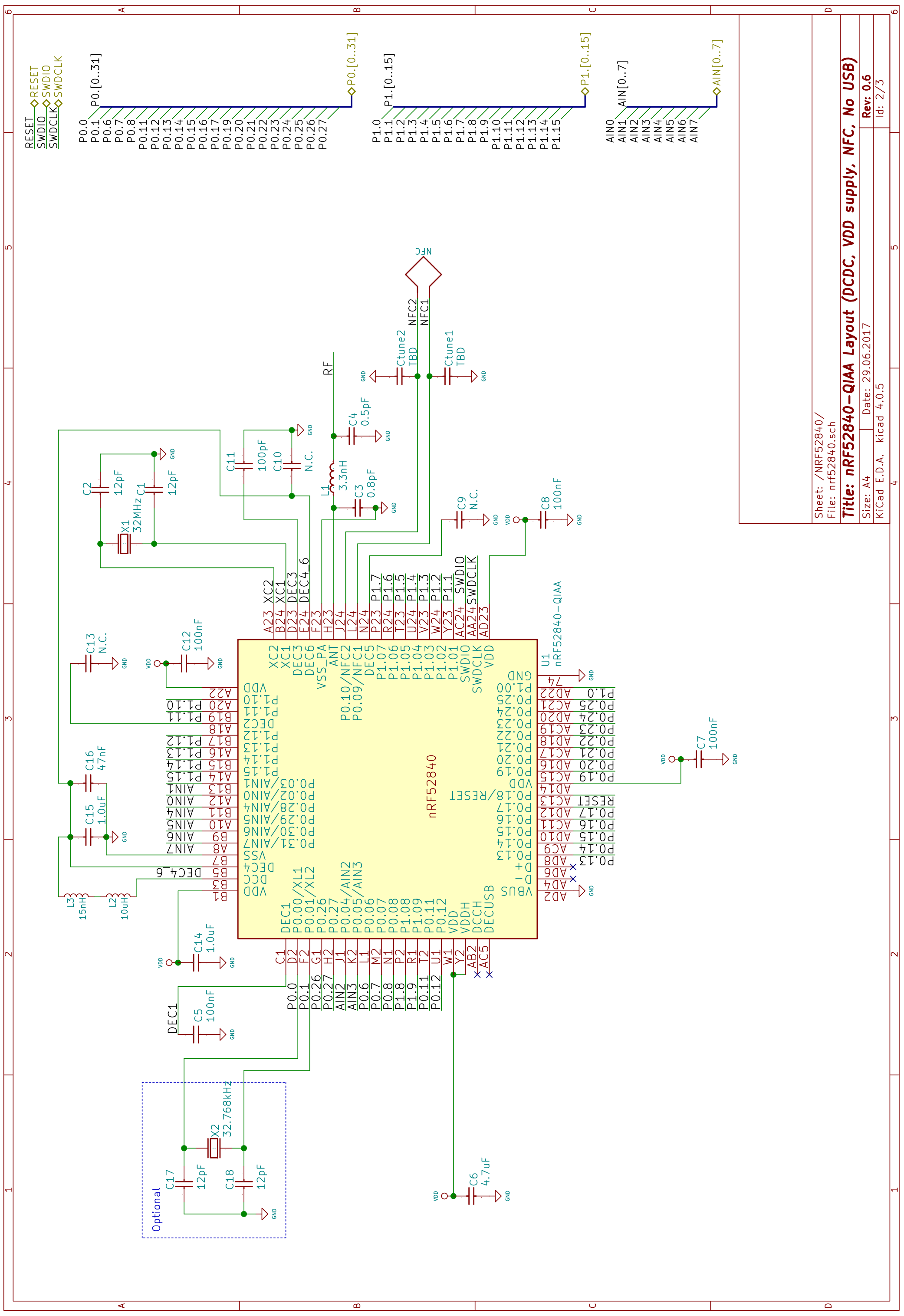
Appendix C

Schematics

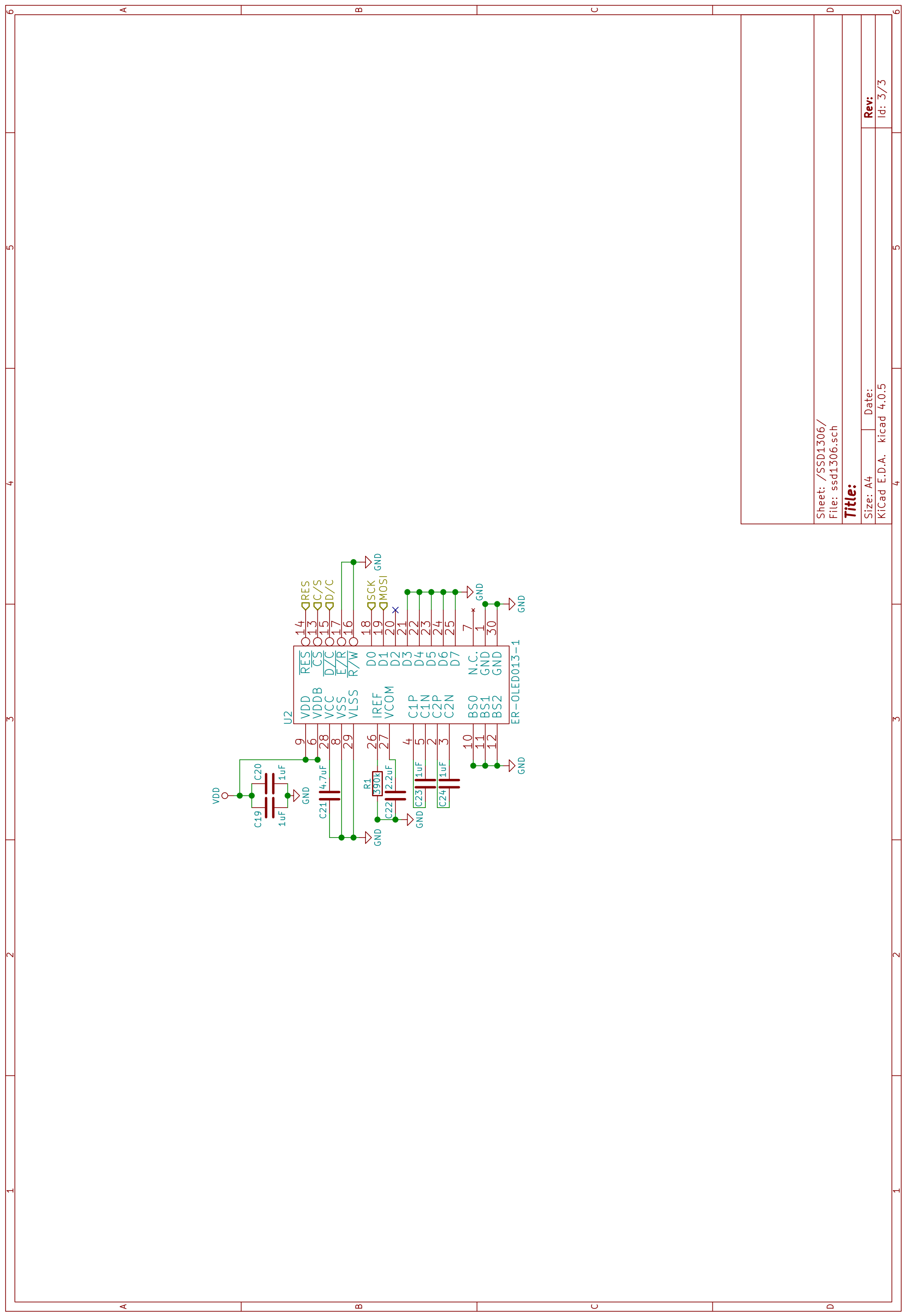
C.1 Top Level



C.2 Controller



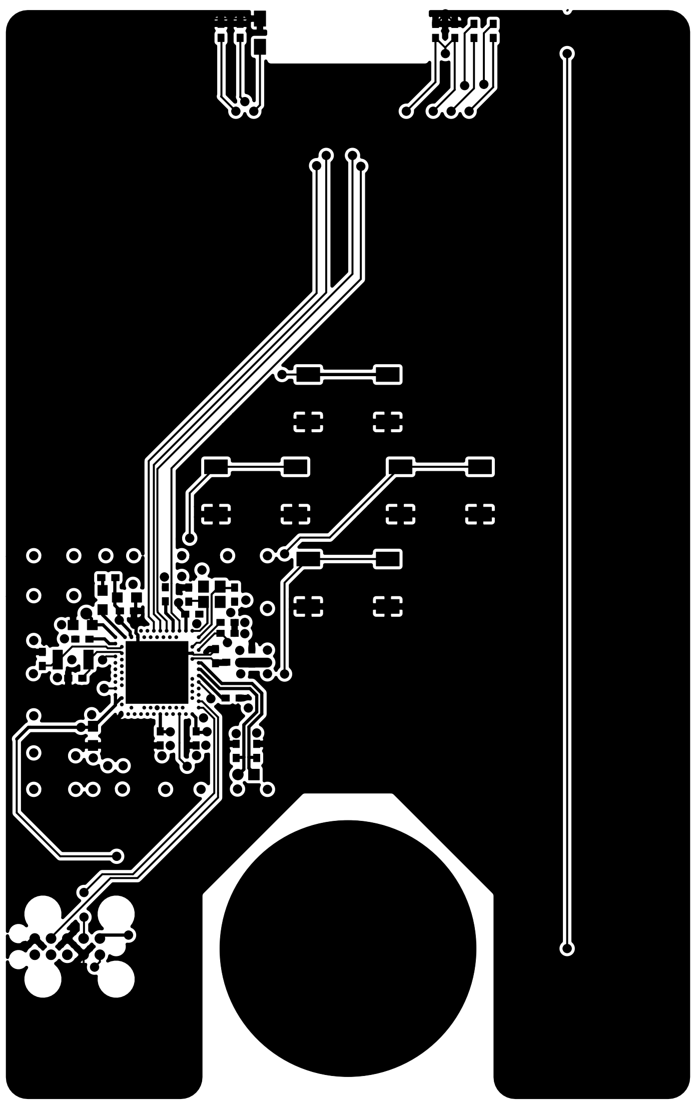
C.3 Screen Interface



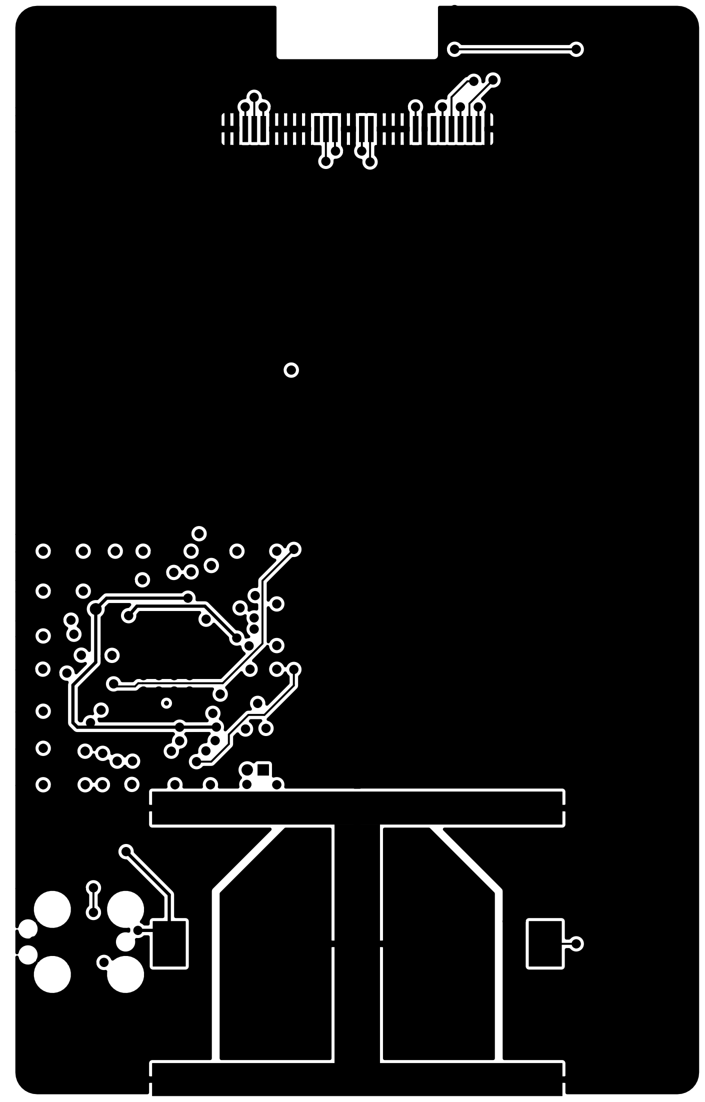
Appendix D

PCB Layout

D.1 Top Layer



D.2 Bottom Layer



Appendix E

Storage Allocation

Flash size: 1 MiB
Page size: 4 KiB
Write unit: 4 bytes

E.1 Page Allocation

Table E.1: Flash Pages

Name	**Pages**	**Start**	**End**
Soft Device	34	0x00000	0x21FFF
Software	40+	0x22000	VAR
-	-	VAR	0xEFFFF
Ethereum Wallets	1	0xF0000	0xF0FFF
-	-	0xF1000	0xFFFFF

E.2 Ethereum Wallets

Table E.2: Ethereum Wallet Flash Storage

Name	**Offset**
ETHADDR_COUNT	0x0000
ETHADDR_LIST	0x0004

Appendix F

Bluetooth Services

Base UUID: 0x3DE9AE84488146D89B9263075B10FE8F

Table F.1: Services

Name	**UUID**
Device Service	0x1000
Ethereum Service	0x2000

F.1 Device Service

Table F.2: Device Service Characteristics

Name	**UUID**	**Properties**	**Request**	**Response**
unlock_methods	0x1001	Read	auth_type_t[]	
unlock	0x1002	Write, Notify	uint8_t[]	bool

F.2 Ethereum Service

Table F.3: Ethereum Service Characteristics

Name	**UUID**	**Properties**	**Request**	**Response**
listAccounts	0x2001	Read, Notify		eth_address_t[]
newAccount	0x2002	Write	eth_address_t[]	
sendTransaction	0x2003	Write, Notify	eth_transaction_t	eth_hash_t

Appendix G

Data Types

G.1 auth_type_t

Type	Code
AUTH_PIN	0
AUTH_PASS	1

G.2 eth_address_t

uint8_t[20];

G.3 eth_hash_t

uint8_t[32];

G.4 eth_transaction_t

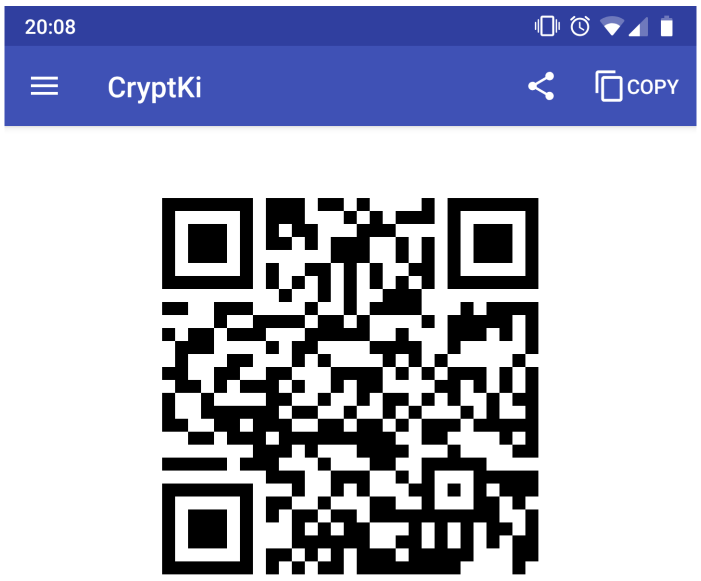
```
struct {
    eth_address_t from;
    eth_address_t to;
    uint64_t nonce;
    int gas;
    int gasPrice;
    int value;
    uint8_t *data;
};
```

Appendix H

User Manual

H.1 Phone App

H.1.1 Receiving Funds



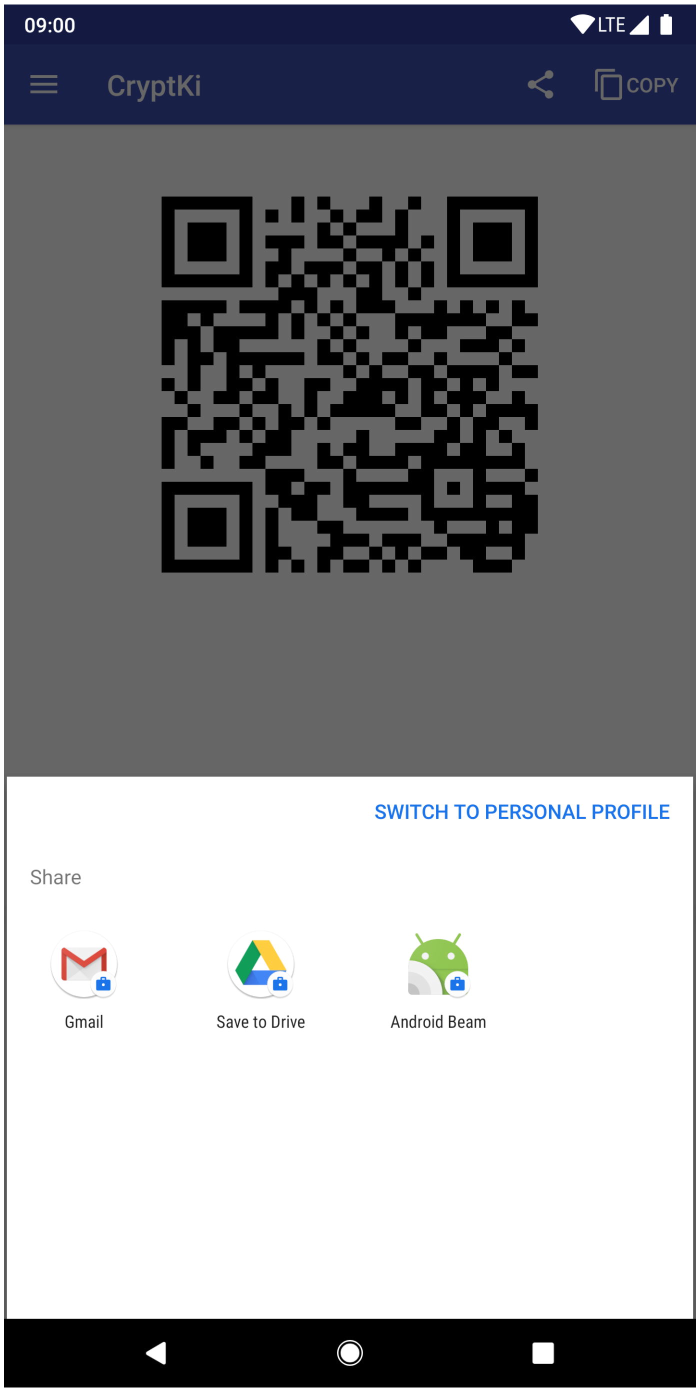


Figure H.2: Share Account Address

This screen is accessed by clicking the receive button on the main page. The displayed QR code can be scanned by other users to enter your address into their app. The QR code may also be shared by email, SMS, or other apps by tapping the icon in the top right corner.

H.1.2 Drawer

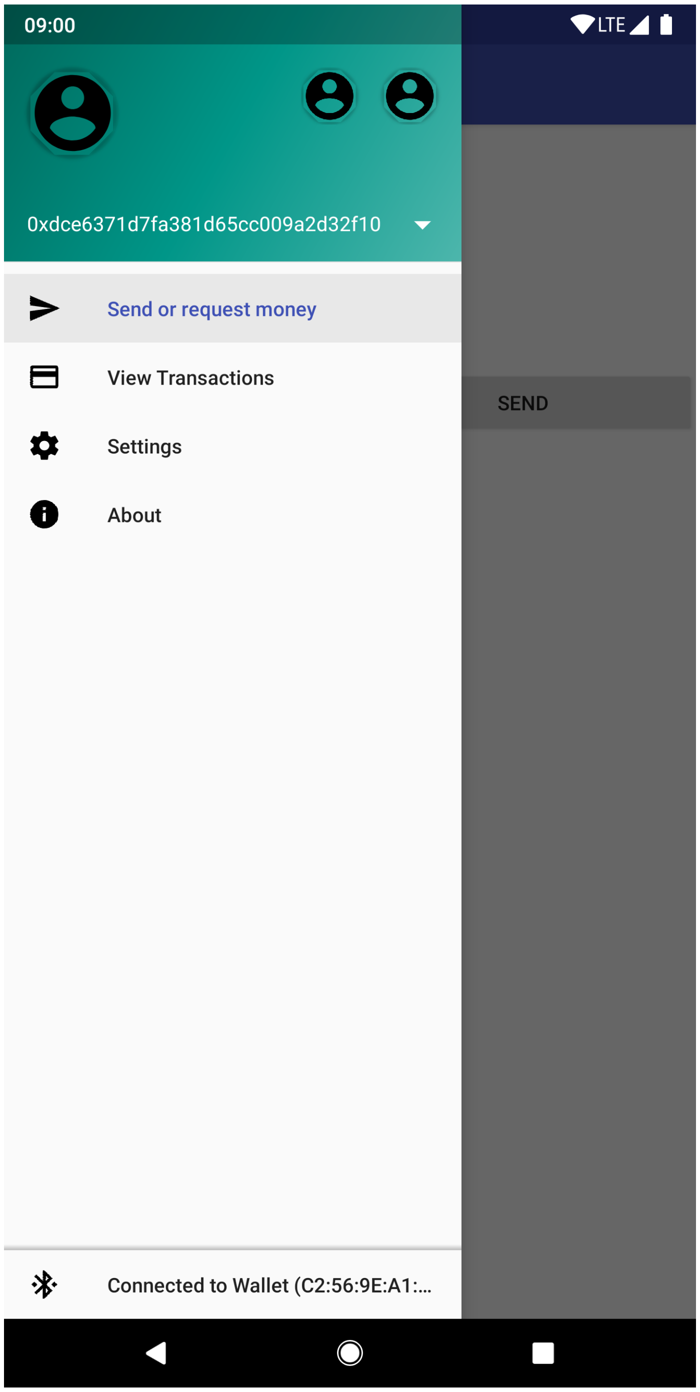
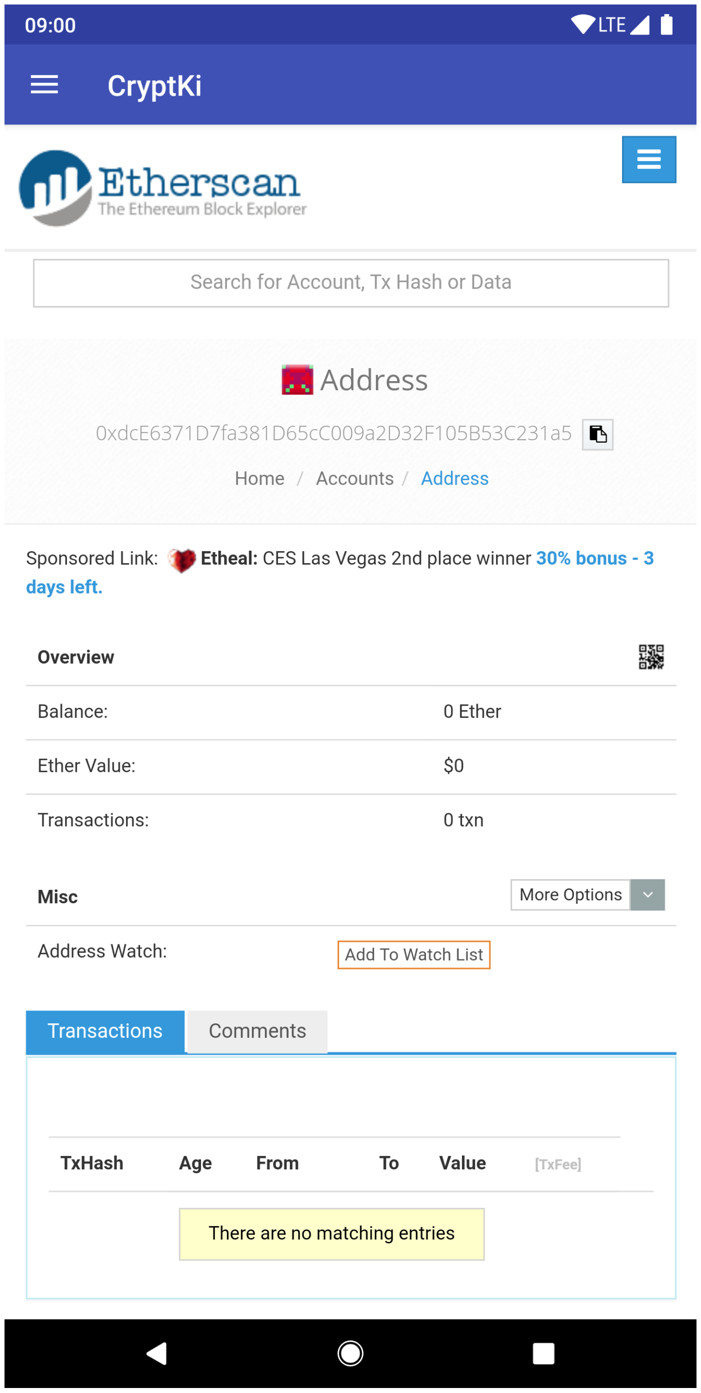

09:00
LTE
0xdce6371d7fa381d65cc009a2d32f10
Send or request money
View Transactions
Settings
About
SEND
Connected to Wallet (C2:56:9E:A1:...

Figure H.3: App Drawer

The app drawer can be accessed by swiping from the left side of the phone screen or by tapped the 3 lined icon in the top left corner. It provides access to many of the apps less used features.

H.1.3 Viewing Transactions


09:00

CryptKi

Etherscan
The Ethereum Block Explorer

Search for Account, Tx Hash or Data

Address

0xdcE6371D7fa381D65cC009a2D32F105B53C231a5

Home / Accounts / Address

Sponsored Link: **Etheal:** CES Las Vegas 2nd place winner **30% bonus - 3 days left.**

Overview	
Balance:	0 Ether
Ether Value:	$0
Transactions:	0 txn

Misc More Options

Address Watch:	Add To Watch List

Transactions Comments

TxHash	Age	From	To	Value	[TxFee]

There are no matching entries

Figure H.4: Transactions List

This screen is accessed by click the transactions button in the app drawer. It is powered by etherscan.io

H.1.4 Switching Accounts

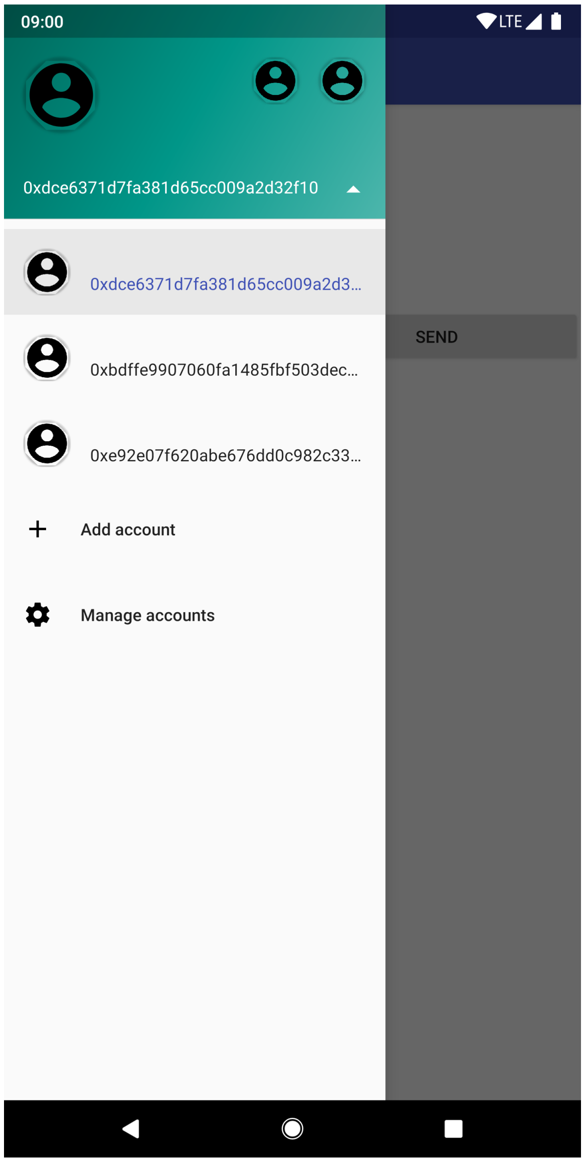
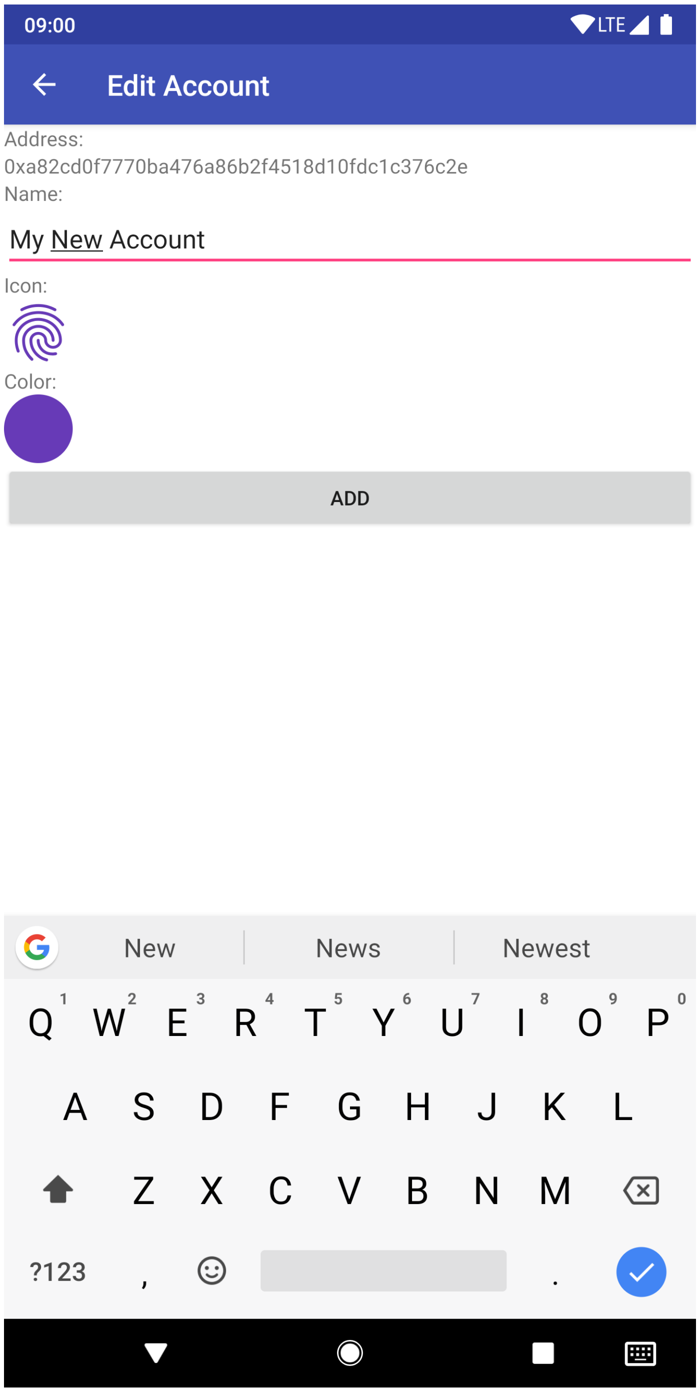

09:00
LTE
0xdce6371d7fa381d65cc009a2d32f10
0xdce6371d7fa381d65cc009a2d3...
0xbdffe9907060fa1485fbf503dec...
0xe92e07f620abe676dd0c982c33...
Add account
Manage accounts
SEND

Figure H.5: Accounts Screen

This screen is accessed by clicking the accounts item in the navigation drawer. Upon clicking an account will be chosen. The balance on the main page, and account on the sending and receiving screens will be updated accordingly.

H.1.5 Creating a New Account


09:00

Edit Account

Address:
0xa82cd0f7770ba476a86b2f4518d10fdc1c376c2e

Name:

My New Account

Icon:

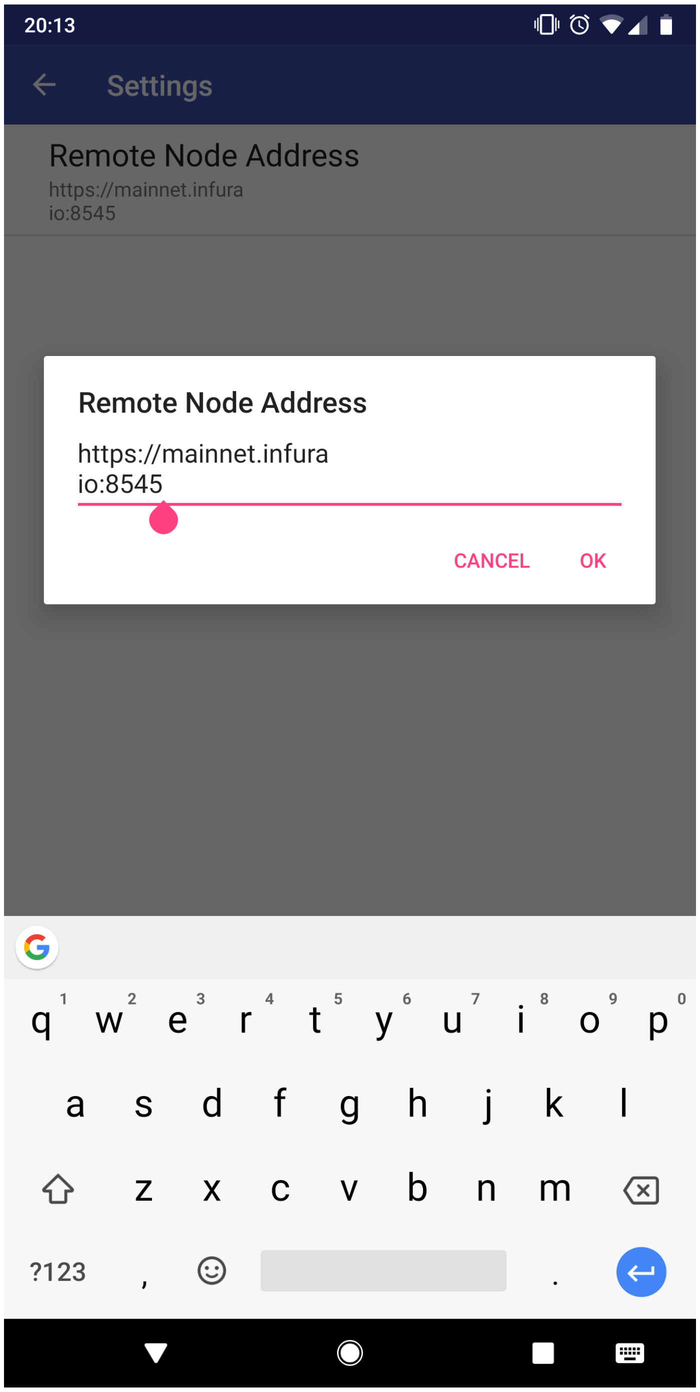
Color:

ADD

Figure H.6: New Account Screen

This screen is accessed by clicking the add account button on the account switcher. Here the user can fill in a human readable name for the account number in addition to an icon and color. These features help make the random addresses more recognizable and memorable.

H.1.6 Changing Nodes



Remote Node Address
https://mainnet.infura
io:8545

Remote Node Address
https://mainnet.infura
io:8545

Figure H.7: Changing Ethereum Node

This screen is accessed by clicking the remote node option in the app settings. The remote node is your connection to the outside world and is responsible for transmitting transactions on your behalf and retrieving your balance from the network. Although a working node is entered by default, you are welcome to host your own node for increased privacy.

H.2 Hardware Wallet

H.2.1 Settings

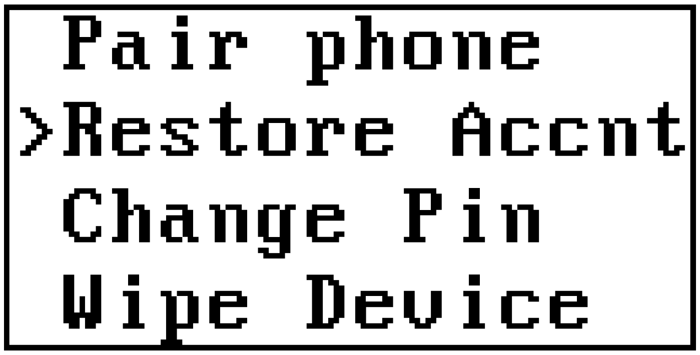


Figure H.8: Settings Screen

This screen will be displayed after setup or after unlocking the device on subsequent power-ons. The screen will display actions that can be executed without the use of a phone. Pressing up and down will move the > symbol, indicating the selected item. Pressing right will choose the selection and navigate to a new screen.

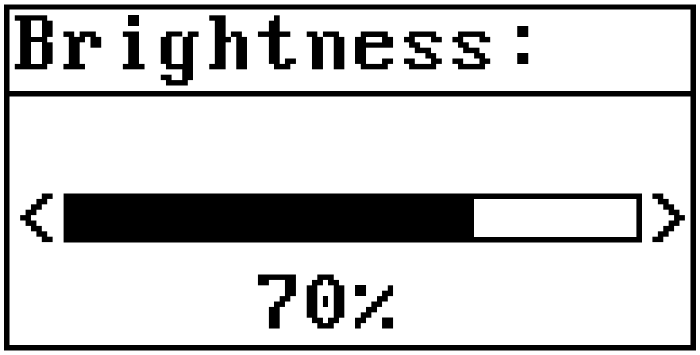


Figure H.9: Brightness Setting

This screen will be displayed after selecting the brightness option. Pressing left will decrease the brightness, and pressing right increase. Pressing up or down will return to the previous screen.

H.2.2 Backup

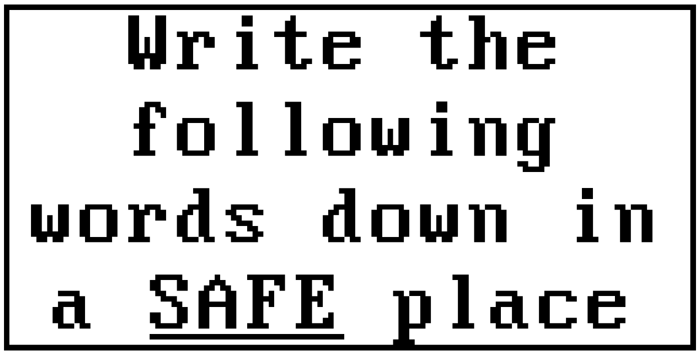

Write the following words down in a SAFE place

Figure H.10: Start Backup

For security reasons a backup key is created. This key allows the user to restore any accounts that were created onto another device, preventing the loss of funds from theft or destruction. Pressing any button will begin the process. This screen starts the process of sharing up the secret key that will be used to recover the user's account.

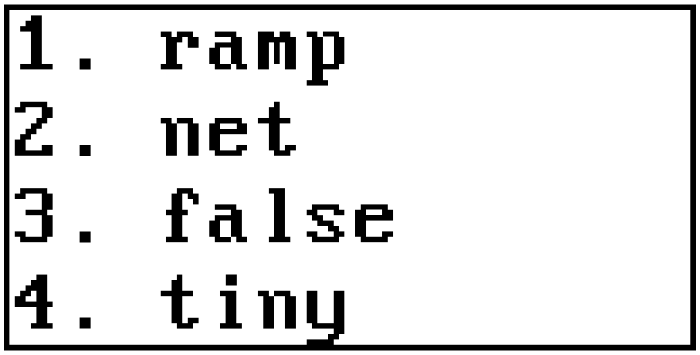

1. ramp
2. net
3. false
4. tiny

Figure H.11: Backup Words Screen

The screen will display a list of words. Pressing right will show the next set of four, and pressing left will return to the previous set. After all 12 words have been written down this process is finished.

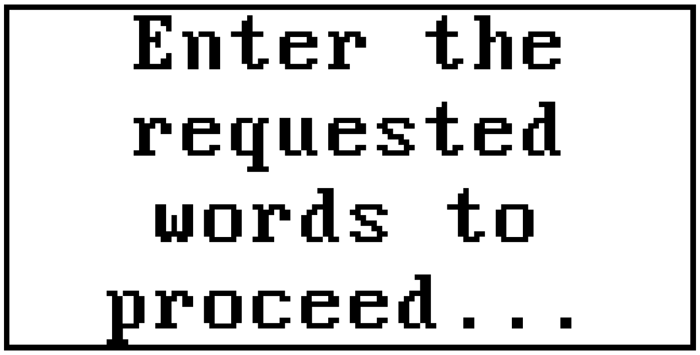
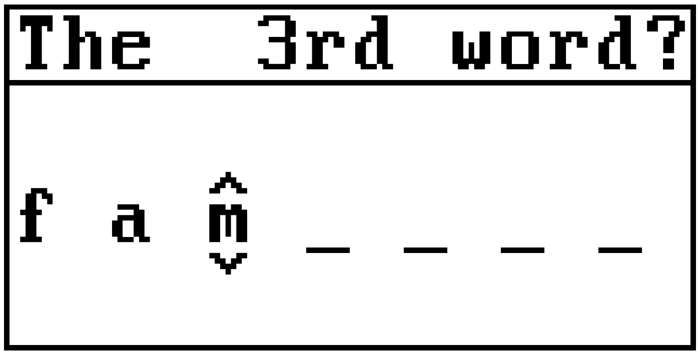

Enter the requested words to proceed...

Figure H.12: Verify Backup

To ensure the user actually saved the secret words, the device will ask for the user to enter the words previously displayed.



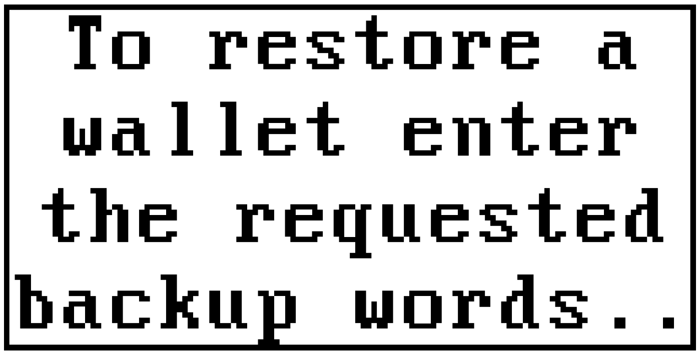
The 3rd word?

f a m _ _ _ _

Figure H.13: Restore Screen

To enter a word, pressing up and down will navigate through the alphabet. Pressing left and right will change positions. If the word input is correct, the setup will proceed. If it is not, the user will be prompted to do the backup process again.

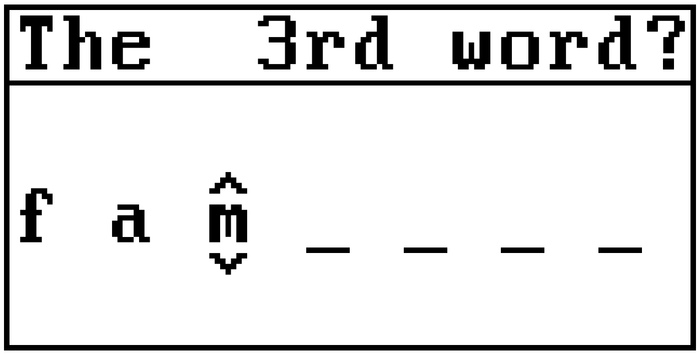
H.2.3 Restore



To restore a wallet enter the requested backup words..

Figure H.14: Start Restore

This screen marks the start of restoring a wallet. Pressing any button begins the process.

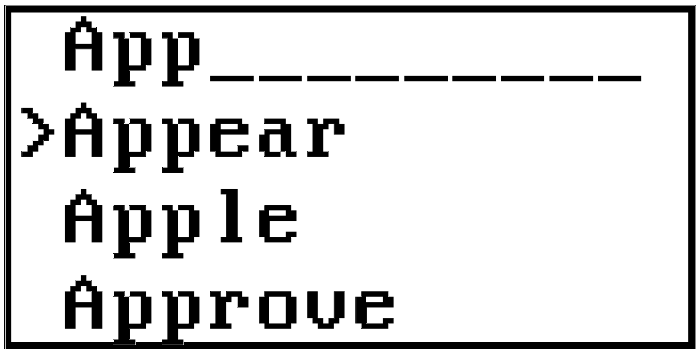


The 3rd word?

f a m _ _ _ _

Figure H.15: Restore Screen

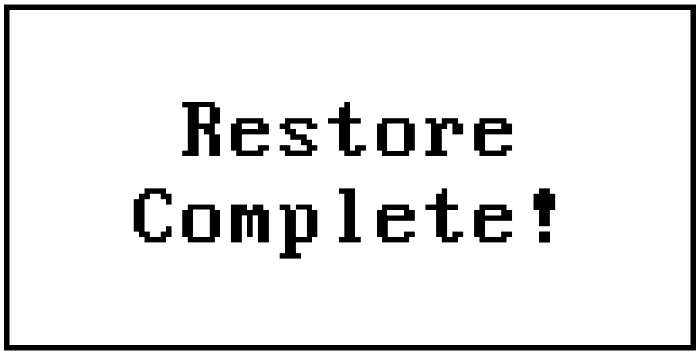
Users will select characters by pressing the up and down buttons and move between different letters with the left and right arrows. Once sufficient letters have been entered for a word a screen with suggestions will appear (see Figure H.16). Pressing left while on the left most character will returning to the previous word in case it was entered incorrectly.



```
 App_________
>Appear
 Apple
 Approve
```

Figure H.16: Suggestions Screen

Pressing up and down will let the user choose from the suggested words. Once a work has been selected by pressing right, the user will return to Figure H.15 to begin entering the next word.



```
  Restore
 Complete!
```

Figure H.17: Restore Complete

Once all words have been entered the restore process is complete.

Appendix I

Privacy Policy

CryptKi built the CryptKi app as a open source app. This SERVICE is provided by CryptKi at no cost and is intended for use as is.

This page is used to inform website visitors regarding our policies with the collection, use, and disclosure of Personal Information if anyone decided to use our Service.

If you choose to use our Service, then you agree to the collection and use of information in relation with this policy. The Personal Information that we collect are used for providing and improving the Service. We will not use or share your information with anyone except as described in this Privacy Policy.

The terms used in this Privacy Policy have the same meanings as in our Terms and Conditions, which is accessible at CryptKi, unless otherwise defined in this Privacy Policy.

Information Collection and Use

For a better experience while using our Service, we may require you to provide us with certain potentially personally identifiable information, including but not limited to bluetooth scanning and camera access. The information that we request is not retained on your device and is not collected by us in any way. The information is soley used in the process of initiating transactions.

The app does use third party services including Google Play Services that may collect information used to identify you.

Links to privacy policies of third party service providers used by the app

- Google Play Services
- Crashlytics

Log Data

We want to inform you that whenever you use our Service, in case of an error in the app we collect data and information (through third party products) on your phone called Log Data. This Log Data may include information such as your devices's Internet Protocol ("IP") address, device name, operating system version, configuration of the app when utilising our Service, the time and date of your use of the Service, and other statistics.

Cookies

Cookies are files with small amount of data that is commonly used an anonymous unique identifier. These are sent to your browser from the website that you visit and are stored on your devices's internal memory.

This Services does not uses these "cookies" explicitly. However, the app may use third party code and libraries that use "cookies" to collection information and to improve their services. You have the option to either accept or refuse these cookies, and know when a cookie is being sent to your device.

Service Providers

We may employ third-party companies and individuals due to the following reasons:

- To facilitate our Service;
- To provide the Service on our behalf;
- To perform Service-related services; or
- To assist us in analyzing how our Service is used.

We want to inform users of this Service that these third parties have access to your Personal Information. The reason is to perform the tasks assigned to them on our behalf. However, they are obligated not to disclose or use the information for any other purpose.

Security

We value your trust in providing us your Personal Information, thus we are striving to use commercially acceptable means of protecting it. But remember that no method of transmission over the internet, or method of electronic storage is 100% secure and reliable, and we cannot guarantee its absolute security.

Links to Other Sites

This Service may contain links to other sites. If you click on a third-party link, you will be directed to that site. Note that these external sites are not operated by [me|us]. Therefore, I strongly advise you to review the Privacy Policy of these websites. I have no control over, and assume no responsibility for the content, privacy policies, or practices of any third-party sites or services.

Children's Privacy

This Services do not address anyone under the age of 13. We do not knowingly collect personal identifiable information from children under 13. In the case we discover that a child under 13 has provided us with personal information, we immediately delete this from our servers. If you are a parent or guardian and you are aware that your child has provided us with personal information, please contact us so that we will be able to do necessary actions.

Changes to This Privacy Policy

We may update our Privacy Policy from time to time. Thus, you are advised to review this page periodically for any changes. We will notify you of any changes by posting the new Privacy Policy on this page. These changes are effective immediately, after they are posted on this page.

Contact Us

If you have any questions or suggestions about our Privacy Policy, do not hesitate to contact us.

This privacy policy page was created at `privacypolicytemplate.net` and modified/generated by App Privacy Policy Generator

Appendix J

Source Code

The current source code of the wallet is over 3000 lines, and the android app an additional 3000 lines. Because of this the full source code will not be placed in this document. After publication it will be released to git with a link available on `CryptKi.com`. Instead the most important routines and defines will be listed here.

J.1 Transaction Signing

```
#define CHAIN_ETH_MAINNET 0x01
#define CHAIN_PARITY_DEV 0x11

int eth_signTransaction(eth_hash_t* dest, eth_transaction_t transaction) {
    char pk_str[65];
    a2hexstr(pk_str, privkey, ETH_PRIVKEY_SIZE);
    NRF_LOG_INFO("PK: %s", pk_str);

    // Conversions
    transaction.value *= 1000000000000000000; //convert eth->wei
    transaction.gasPrice *= 1000000000; //convert gwei->wei

    uint8_t nonce_bytes[32], gasPrice_bytes[32], gas_bytes[32], value_bytes[32];
    size_t nonce_len    = bytes(nonce_bytes,    transaction.nonce);
    size_t gasPrice_len = bytes(gasPrice_bytes, transaction.gasPrice);
    size_t gas_len      = bytes(gas_bytes,      transaction.gas);
    size_t value_len    = bytes(value_bytes,    transaction.value);


    // Calculate RLP length
    uint32_t rlp_length = 1; //off by one?
    rlp_length += rlp_calculate_length(nonce_len,         nonce_bytes[0]);
    rlp_length += rlp_calculate_length(gasPrice_len,      gasPrice_bytes[0]);
    rlp_length += rlp_calculate_length(gas_len,           gas_bytes[0]);
    rlp_length += rlp_calculate_length(ETH_ADDRESS_SIZE,  transaction.to[0]);
    rlp_length += rlp_calculate_length(value_len,         value_bytes[0]);
    if (chain_id) {
        rlp_length += rlp_calculate_length(1, chain_id);
        rlp_length += rlp_calculate_length(0, 0);
        rlp_length += rlp_calculate_length(0, 0);
    }


    // Compute RLP
    size_t data_size = 0;
    sha3_256_Init(&keccak_ctx);
    hash_rlp_list_length(rlp_length);
    hash_rlp_field(nonce_bytes,    nonce_len);
    hash_rlp_field(gasPrice_bytes, gasPrice_len);
    hash_rlp_field(gas_bytes,      gas_len);
    hash_rlp_field(transaction.to, ETH_ADDRESS_SIZE);
    hash_rlp_field(value_bytes,    value_len);
    hash_rlp_length(data_size,     transaction.data[0]);
    hash_data(transaction.data,    data_size);
    if (chain_id) {
        hash_rlp_number(chain_id);
        hash_rlp_length(0, 0);
        hash_rlp_length(0, 0);
    }

    // Compute Signiture
    uint8_t hash[32], sig[64];
    uint8_t v;
    keccak_Final(&keccak_ctx, hash);

    int rc = ecdsa_sign_digest(&secp256k1, privkey, hash, sig, &v,
    ethereum_is_canonic);
    if (rc != 0) {
        NRF_LOG_INFO("Invalid signature");
    }

    if (chain_id) {
        v = v + 2 * chain_id + 35;
    } else {
        v = v + 27;
    }

    uint8_t r[32], s[32];
    memcpy(r, sig+0,  32);
    memcpy(s, sig+32, 32);

    // Form transaction message
    rlp_length += rlp_calculate_length(1, v);
    rlp_length += rlp_calculate_length(32, r[0]);
    rlp_length += rlp_calculate_length(32, s[0]);
    rlp_length -= 3; //Fix size. something with no chainid here

    sha3_256_Init(&keccak_ctx);
    hash_rlp_list_length(rlp_length);
    hash_rlp_field(nonce_bytes,    nonce_len);
    hash_rlp_field(gasPrice_bytes, gasPrice_len);
    hash_rlp_field(gas_bytes,      gas_len);
    hash_rlp_field(transaction.to, ETH_ADDRESS_SIZE);
    hash_rlp_field(value_bytes,    value_len);
    hash_rlp_length(data_size,     transaction.data[0]);
    hash_data(transaction.data,    data_size);
    hash_rlp_field(&v, 1);
    hash_rlp_field(r, 32);
    hash_rlp_field(s, 32);

    int tx_len = rlp_length+2;
    static uint8_t tx[ETH_HASH_SIZE];
    memcpy(tx, keccak_ctx.message, tx_len);

    char tx_str[223];
    tx_str[0] = '0';
    tx_str[1] = 'x';
    a2hexstr(&tx_str[2], tx, tx_len+2);
    NRF_LOG_INFO("Transaction: %s", tx_str);

    memcpy(dest, tx, tx_len);
    return tx_len;
}

static size_t bytes(uint8_t *dest, uint64_t val) {
    size_t count = 0;
    while ( val > 0 ) {
        dest[count] = val & 0xFF;
        val >>= 8;
        count++;
    }

    for (uint8_t i=0; i<count/2; i++) {
        uint8_t temp = dest[i];
        dest[i] = dest[(count-1)-i];
        dest[(count-1)-i] = temp;
    }

    return count;
}
```

J.2 Public Address Generation

```
uint32_t eth_listAddresses(eth_address_t *addresses) {
    uint32_t count;
    eth_addresses_count(&count);
    NRF_LOG_INFO("num accounts: %d", count);

    eth_address_t from;
    eth_getAddress(from, privkey);
    memcpy(&addresses[0], from, ETH_ADDRESS_SIZE);

    return count;
}
```

J.3 RLP Encoding

```
static int ethereum_is_canonic(uint8_t v, uint8_t signature[64])
{
    (void) signature;
    return (v & 2) == 0;
}

static inline void hash_data(const uint8_t *buf, size_t size)
{
    sha3_Update(&keccak_ctx, buf, size);
}

static void hash_rlp_length(uint32_t length, uint8_t firstbyte)
{
    uint8_t buf[4];
    if (length == 1 && firstbyte <= 0x7f) {
        /* empty length header */
    } else if (length <= 55) {
        buf[0] = 0x80 + length;
        hash_data(buf, 1);
    } else if (length <= 0xff) {
        buf[0] = 0xb7 + 1;
        buf[1] = length;
        hash_data(buf, 2);
    } else if (length <= 0xffff) {
        buf[0] = 0xb7 + 2;
        buf[1] = length >> 8;
        buf[2] = length & 0xff;
        hash_data(buf, 3);
    } else {
        buf[0] = 0xb7 + 3;
        buf[1] = length >> 16;
        buf[2] = length >> 8;
        buf[3] = length & 0xff;
        hash_data(buf, 4);
    }
}

static void hash_rlp_list_length(uint32_t length)
{
    uint8_t buf[4];
    if (length <= 55) {
        buf[0] = 0xc0 + length;
        hash_data(buf, 1);
    } else if (length <= 0xff) {
        buf[0] = 0xf7 + 1;
        buf[1] = length;
        hash_data(buf, 2);
    } else if (length <= 0xffff) {
        buf[0] = 0xf7 + 2;
        buf[1] = length >> 8;
        buf[2] = length & 0xff;
        hash_data(buf, 3);
    } else {
        buf[0] = 0xf7 + 3;
        buf[1] = length >> 16;
        buf[2] = length >> 8;
        buf[3] = length & 0xff;
        hash_data(buf, 4);
    }
}

/*
 * Push an RLP encoded length field and data to the hash buffer.
 */
static void hash_rlp_field(const void *_buf, size_t size)
{
    const uint8_t *buf = (const uint8_t*)_buf;
    hash_rlp_length(size, buf[0]);
    hash_data(buf, size);
}

static void hash_rlp_number(uint32_t number)
{
    if (!number) {
        return;
    }
    uint8_t data[4];
    data[0] = (number >> 24) & 0xff;
    data[1] = (number >> 16) & 0xff;
    data[2] = (number >> 8) & 0xff;
    data[3] = (number) & 0xff;
    int offset = 0;
    while (!data[offset]) {
        offset++;
    }
    hash_rlp_field(data + offset, 4 - offset);
}

static int rlp_calculate_length(int length, uint8_t firstbyte)
{
    if (length == 1 && firstbyte <= 0x7f) {
        return 1;
    } else if (length <= 55) {
        return 1 + length;
    } else if (length <= 0xff) {
        return 2 + length;
    } else if (length <= 0xffff) {
        return 3 + length;
    } else {
        return 4 + length;
    }
}
```

J.4 Oled Interface

```
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define LINE_HEIGHT 16

ret_code_t ssd1306_init(void);
void ssd1306_uninit(void);
void ssd1306_pixel_draw(uint16_t x, uint16_t y, uint32_t color);
void ssd1306_rect_draw(uint16_t x, uint16_t y, uint16_t width, uint16_t height,
    uint32_t color);
void ssd1306_display_screen(void);
void ssd1306_display_invert(bool invert);
void ssd1306_clear_display(void);


void oled_init();
void oled_clear();
void oled_redraw();
void oled_invert(bool invert);
void oled_write(uint8_t x, uint8_t line, const char *text);
int a2hexstr(char *dest, const uint8_t *src, int len);


typedef struct screen_t {
    void (*on_load)(void *data);
    void (*on_unload)(void);
    bool (*on_evt)(bsp_event_t event);
    struct screen_t *prev_screen;
    void *data;
} screen_t;

void screen_load(screen_t *screen, void *data);
void screen_set_prev(screen_t *screen);
void screen_prev();
void screen_evt(bsp_event_t evt);
```

J.5 BLE Defines

```
#define APP_FEATURE_NOT_SUPPORTED        BLE_GATT_STATUS_ATTERR_APP_BEGIN + 2     /**<
    Reply when unsupported features are requested. */
#define APP_ADV_INTERVAL                 300                                      /**<
    The advertising interval (in units of 0.625 ms. This value corresponds to 187.5
    ms). */
#define APP_ADV_TIMEOUT_IN_SECONDS       180                                      /**<
    The advertising timeout in units of seconds. */

#define SECURITY_REQUEST_DELAY           APP_TIMER_TICKS(400)                     /**<
    Delay after connection until Security Request is sent, if necessary (ticks). */

#define APP_BLE_OBSERVER_PRIO            1                                        /**<
    Application's BLE observer priority. You shouldn't need to modify this value. */
#define APP_BLE_CONN_CFG_TAG             1                                        /**<
    A tag identifying the SoftDevice BLE configuration. */

#define MIN_CONN_INTERVAL                MSEC_TO_UNITS(100, UNIT_1_25_MS)         /**<
    Minimum acceptable connection interval (0.1 seconds). */
#define MAX_CONN_INTERVAL                MSEC_TO_UNITS(200, UNIT_1_25_MS)         /**<
    Maximum acceptable connection interval (0.2 second). */
#define SLAVE_LATENCY                    0                                        /**<
    Slave latency. */
#define CONN_SUP_TIMEOUT                 MSEC_TO_UNITS(4000, UNIT_10_MS)          /**<
    Connection supervisory timeout (4 seconds). */

#define FIRST_CONN_PARAMS_UPDATE_DELAY   APP_TIMER_TICKS(5000)                    /**<
```

```
    Time from initiating event (connect or start of notification) to first time
    sd_ble_gap_conn_param_update is called (5 seconds). */
#define NEXT_CONN_PARAMS_UPDATE_DELAY    APP_TIMER_TICKS(30000)                    /**<
    Time between each call to sd_ble_gap_conn_param_update after the first call (30
    seconds). */
#define MAX_CONN_PARAMS_UPDATE_COUNT     3                                         /**<
    Number of attempts before giving up the connection parameter negotiation. */
```

J.6 SDK Defines

```
// Defaults in <SDK>/config/sdk_config.h

#define APP_TIMER_ENABLED 1
#define BLE_ADVERTISING_ENABLED 1
#define BUTTON_ENABLED 1
#define FDS_ENABLED 1
#define GPIOTE_ENABLED 1
#define NRF_BALLOC_ENABLED 1
#define NRF_BLE_CONN_PARAMS_ENABLED 1
#define NRF_BLE_GATT_ENABLED 1
#define NRF_CLI_ECHO_STATUS 1
#define NRF_CLI_ENABLED 1
#define NRF_CLI_LOG_BACKEND 1
#define NRF_CLI_RTT_ENABLED 1
#define NRF_FPRINTF_ENABLED 1
#define NRF_FSTORAGE_ENABLED 1
#define NRF_LOG_ENABLED 1
#define NRF_LOG_ERROR_COLOR 2
#define NRF_LOG_USES_COLORS 1
#define NRF_LOG_WARNING_COLOR 4
#define NRF_QUEUE_ENABLED 1
#define NRF_SDH_BLE_ENABLED 1
//#define NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE 1408
#define NRF_SDH_BLE_PERIPHERAL_LINK_COUNT 1
#define NRF_SDH_BLE_VS_UUID_COUNT 1
#define NRF_SDH_ENABLED 1
#define NRF_SDH_SOC_ENABLED 1
#define NRF_SECTION_ITER_ENABLED 1
#define NRF_STRERROR_ENABLED 1
#define PEER_MANAGER_ENABLED 1
#define NRF_SDH_BLE_GATT_MAX_MTU_SIZE 250
#define SEGGER_RTT_CONFIG_DEFAULT_MODE 1 //TRIM

#define CLOCK_ENABLED 1
#define NFC_AC_REC_ENABLED 1
#define NFC_BLE_OOB_ADVDATA_ENABLED 1
#define NFC_BLE_PAIR_LIB_ENABLED 1
#define NFC_BLE_PAIR_MSG_ENABLED 1
#define NFC_CH_COMMON_ENABLED 1
#define NFC_EP_OOB_REC_ENABLED 1
#define NFC_HS_REC_ENABLED 1
#define NFC_LE_OOB_REC_ENABLED 1
#define NFC_NDEF_MSG_ENABLED 1
#define NFC_NDEF_RECORD_ENABLED 1
#define NFC_PAIRING_MODE 4 // NFC_PAIRING_MODE_GENERIC_OOB
#define NFC_T2T_HAL_ENABLED 1
#define RNG_ENABLED 1

#define NRF_GFX_ENABLED 1
#define NRF_SPI_DRV_MISO_PULLUP_CFG 1
#define NRF_SPI_MNGR_ENABLED 1
#define SPI0_ENABLED 1
#define SPI0_USE_EASY_DMA 0
#define SPI_ENABLED 1

#ifdef DEBUG_NRF
#define NRF_LOG_DEFERRRED 0
```

```
#define NRF_LOG_DEFAULT_LEVEL 4 //Debug
#define NRF_BLE_PAIR_LIB_LOG_ENABLED 1
#define HAL_NFC_CONFIG_LOG_ENABLED 1
#endif
```

Appendix K

Glossary

ARM Advanced RISC Machines.

BLE Bluetooth Low Energy.

Blockchain A data type similar to a linked list but previous nodes (blocks) are immutable.

CCCD Client Characteristic Configuration Descriptor.

CLI Command Line Interface.

Ethereum A type of cryptocurrency.

GAP Generic Access Protocol.

GATT Generic Attribute Profile.

HCI Host Control Interface.

HTTP Hypertext Transfer Protocol.

iOS A mobile operating system developed by Apple.

JSON JavaScript Object Notation.

JTAG Join Test Action Group.

MCU Microcontroller Unit.

NFC Near Field Communication.

OOB Out of Band.

PCB Printed Circuit Board.

RAM Random Access Memory.

RISC Reduced Instruction Set Computer.

ROM Read Only Memory.

RPC Remote Procedure Call.

SOC System On a Chip.

SWI Software Interrupt Instruction.