UNIVERSIDADE FEDERAL DE SANTA CATARINA
INE - Departamento de Informática e Estatística
COMPUTER SCIENCE

RICARDO VIEIRA FRITSCHE

RECOMMENDATIONS FOR IMPLEMENTING A BITCOIN WALLET USING SMART CARD

Florianópolis, Brazil, 2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Fritsche, Ricardo Vieira
Recommendations for implementing a Bitcoin wallet using smart card / Ricardo Vieira Fritsche ; orientador, Jean E Martina, coorientador, Lucas M Palma, 2018.
90 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2018.

Inclui referências.

1. Ciências da Computação. 2. Hardware wallet. 3. Bitcoin. 4. Software and hardware integration . 5. Smart Card. I. Martina, Jean E. II. Palma, Lucas M. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Título.

RICARDO VIEIRA FRITSCHE

RECOMMENDATIONS FOR IMPLEMENTING A BITCOIN WALLET USING SMART CARD

Trabalho de Conclusão de Curso submetido ao Curso de Ciências da Computação para a obtenção do Grau de Bacharelado.

Prof. Dr. Jean Everson Martina
Orientador

Lucas Machado da Palma
Co-orientador

Florianópolis, 20 de outubro de 2018.

RICARDO VIEIRA FRITSCHE

RECOMMENDATIONS FOR IMPLEMENTING A BITCOIN WALLET USING SMART CARD

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de “Bacharelado em Ciências da Computação”, e aprovado em sua forma final pelo Curso de Ciências da Computação.

Florianópolis, ___ de _____ de 2018.

_____

Prof. Dr. Rafael Luiz Cancian
Coordenador do Curso

_____

Prof. Dr. Jean Everson Martina
Orientador

_____

Lucas Machado da Palma
Co-orientador

_____

M.e Lucas Pandolfo Perin
Banca avaliadora

_____

Douglas Marcelino Beppler Martins
Banca avaliadora

I dedicate this work to everyone who have lost cryptocurrencies that weren't held on cold storage.

APPRECIATION

I would like to praise my mother, Mara Vieira, who always supported my decisions, never judging. My father, Julio Cesar, for pointing the way into the computer science world since I was young. Gustavo Fritsche, my brother, for the support and brotherhood.

My business partner, Alexandre Augusto, thank you for sharing your thoughts on the importance of completing the course and your support to maintain the business while I couldn't deliver full attention to it.

Special thanks to my girlfriend, Lorrana Rezzieri, a very patient angel who listened to all my complains and annoyance, always supporting and remembering me that the finish line was close.

My cousin, João Guilherme, thank you for revising this work, studying, and drinking cold beers together after the exams.

Antônio Fröhlich, thanks for supporting the initial idea of this work and contributing with your time and insights.

Thank you Jean Martina for quickly accepting to orient this work and your wisdom. Lucas Palma, your validation and revision was fundamental for this work, thank you.

To finish I would like to register my acknowledgement to whomever produced knowledge and code that somehow contributed to my formation.

Ricardo Vieira Fritsche

“*Better late than never*”

ABSTRACT

Bitcoin is a decentralized peer-to-peer electronic cash system that allows any two willing parties to transact directly without the need for a trusted third party. The user's funds are protected by private keys that must be kept safe, preferably not on third party wallet services, but on hardware wallets, which are the best balance between very high security and ease of use. In this work we made a review on cryptography, the Bitcoin protocol and secure elements, then we dived into the project of hardware wallets, discussing different requirements and ways to construct one. Our proposed device uses an anti tamper Java Card to store the private keys. We considered variations of the device, one with a dedicated touchscreen and another with NFC to integrate with a mobile phone. We analyzed security aspects of the project, made recommendations and described some challenges. Finally, we implemented our own open source prototype, showing the architecture of the project, its components, the requirements, the APDU communication protocol and the results.

Keywords: Hardware wallet; Bitcoin; Software and hardware integration; Smart Card.

SUMMARY

INTRODUCTION

Almost ten years ago Satoshi Nakamoto introduced Bitcoin to the world. His peer-to-peer electronic cash system allows any two willing parties to transact directly with each other without the need for a trusted third party [1]. Bitcoin has grown rapidly, giving birth to a novel industry, involving other cryptocurrencies, blockchain technology and new economic dynamics [6]. On september of 2018 cryptocurrencies had a U$ 200 billion market capitalization, of which U$ 112 billion (55%) is Bitcoin [3].

Despite its huge success, one could say that Nakamoto's vision is not yet fulfilled, as the ecosystem presents signs of centralization, especially with the rapid growth of third parties companies that hold and manage the user's cryptographic keys, in other words, the user's funds [8]. In the final months of 2017 those major cryptocurrency exchanges have added more than 100,000 users per day [13].

Many users might prefer to leave their coins on an exchange or online wallet, as it seems to be easier and works almost like the current online banking solution [7]. The problem is that unlike banks, these services cannot offer any guarantees on the user's holdings. In fact, due to hacking, more than U$ 15 billion were lost since 2013 [2]. Beyond that, if the users are not the real owners of their funds, the whole idea of decentralization is lost, giving space to government censorship, retention of user's funds and other threats [14, 15].

For one to be the real owner of its coins, he must deal with public-key cryptography [7]. In the context of Bitcoin "*managing, controlling, and using cryptographic keys are complex tasks, and no clear solution has been proposed*" [8]. The best solution we have so far are called hardware wallets:

> *"Hardware wallets are the best balance between very high security and ease of use. These are little devices that are designed from the root to be a wallet and nothing else. No software can be installed on them, making them very secure against computer vulnerabilities and online thieves. Because they can allow backup, you can recover your funds if you lose the device."* [16]

The main hardware wallets on the market are the *Ledger Nano*, *Trezor*, *KeepKey* and *Digital BitBox* [5]. They have limitations on the number of cryptocurrencies supported and a relative high price, but the main concern is that they still present some security flaws, like the exposure of the communication channel [12] and the openness to a *Supply Chain* and *Evil Maid* attacks [71]. Despite that, they are continuously being improved by their manufacturers.

Considering the explosive growth of cryptocurrencies and the endless possibilities for a better future [17], it is necessary to give users more options to secure their funds and be independent of third parties. With the mentioned idea in mind, this academic work hopes to contribute with the cryptocurrency ecosystem by making a thorough description of some important aspects of managing keys and addresses, and also listing some important security aspects unique to Bitcoin. Not only that, but also to develop an open source prototype that might be a starting point for more robust wallets that are low cost, secure and can be adapted for different coins.

We analyze different hardware wallet projects possibilities, with and without a dedicated screen and develop a prototype based on smart card technology, more specifically on *Java Card*, the leading platform, with more than 10 billion devices deployed [19, 20]. These devices are tamper resistant [26] and some of them are Evaluation Assurance Level (EAL) certified [58]. We explore the solutions to encrypt the communication channel, preventing Man-in-the-Middle (MitM) attacks [12] by using password-authenticated secure channel protocol (SRP) [27] and public key cryptography.

It should be noted that our proposal is not a panacea, as it is based on technologies that have known flaws [18, 22, 23, 24], but are the best available for the price and are constantly being enhanced. Our implementation is focused on serving users that don't hold very large amounts of funds. For a more secure setup, users should use threshold cryptography [25], that in the future might be incorporated in our work.

OBJECTIVES

MAIN OBJECTIVE

The goal of this work is to provide recommendations for implementing a secure and low cost Bitcoin hardware wallet using a smart card.

SPECIFIC OBJECTIVES

1. Define the requirements and features for a Bitcoin hardware wallet, analysing the security aspects involved;
2. Project a general architecture for a Bitcoin hardware wallet and its communication protocol;
3. Implement a working prototype of a Bitcoin hardware wallet that runs on a Java Card (following the ISO 7816 standard) and uses web technology.

EXPECTED RESULTS

As mentioned earlier, we want to contribute with the cryptocurrency ecosystem by providing the development community a rich description of the components involved in a hardware wallet project, presenting security recommendations, suggestions for the architecture, the communication protocol and to provide an open source prototype. We hope this will help Bitcoin to achieve its original goal of a secure and decentralized system.

STRUCTURE OF THIS WORK

On the **Fundamentals** chapter we make a revision on the theoretical aspects of cryptographic (hash functions, message authentication code, key derivation, asymmetric cryptography), describe the workings of the Bitcoin protocol and detail the protocols to generate addresses and manage wallets. Then we also review related works (attacks in Bitcoin wallets, secure channels and usability analysis).

Next, on the **Development** chapter, we incrementally analyze different possibilities for the hardware wallet project and make specific security considerations.

We then advance to the prototype by defining the architecture and requirements, concluding with the presentation of how we coded the solution.

Finally on the **Conclusion** chapter we present the main findings and propose themes for future works that can be related with this project.

FUNDAMENTALS

CRYPTOGRAPHY

In this section we make a brief explanation of the main cryptography concepts used by the Bitcoin protocol and especially by the Bitcoin Improvement Proposals (BIPs) related to address generation and management [28].

Cryptographic Hash Functions

A cryptographic hash function takes a variable-length block of data as input and returns a fixed-size value called hash [30]. Any change in the message will have a very high probability to produce a distinct hash, allowing to check for data integrity and various others applications, notably digital signatures [31].

To be considered secure and strong, hash functions must fill some important requirements [30]:

1. To be deterministic, so the same message always produces the same hash;
2. To be fast to compute for any given message;
3. To be infeasible to find the original message by having only the hash (preimage resistant);
4. Given message **m1** and its hash, it is infeasible to find a message **m2** different from **m1** but with the same hash (second preimage resistant);
5. To be infeasible to find two different messages with the same hash (strong collision resistant).

SHA

The Secure Hash Algorithm (SHA) is a family of widely used hash functions [30] standardized by the National Institute of Standards and Technology (NIST) as well as the U.S. Federal Information Processing Standard (FIPS).

The SHA-1 is a 160-bit hash function which resembles the Message Digest Algorithm 5 (MD5) and since late 2005 was advised not be used anymore, in flavor of the SHA-2 variants [30], which has some similar characteristics to its predecessor, but has different block sizes, known as SHA-256 and SHA-512, with 32-bit and 64-bit

words, respectively. There are truncated versions of each standard, known as SHA-224, SHA-384, SHA-512/224 and SHA-512/256. The newest version is called SHA-3 (formerly Keccak), released in 2015 after a public competition among non-NSA designers. It supports the same hash lengths as SHA-2, but its internal structure differs largely from the rest of the SHA family. It is not meant to replace SHA-2, but to compose the family of SHA with more robust options [29].

The SHA-2 mechanism is very well explained on the FIPS publications, however it is out of the scope of this work to detail it. What is very important to note is that SHA-2 is at the heart of the Bitcoin protocol, being used for the Proof-of-work (PoW) algorithm as well as for generating address [1, 28], both of which will be detailed further ahead.

RIPEMD

Research and Development in Advanced Communications Technologies in Europe (RACE) Integrity Primitives Evaluation Message Digest (RIPEMD) is a family of functions based on the Message Digest Algorithm 4 (MD4), created by the academic community [31]. The main used variant is the RIPEMD-160, an improved version that generates 160 bits hashes. Currently, no successful efficient attacks against the hash functions are known [33].

HMAC

A Message Authentication Code (MAC) algorithm has two purposes: to verify the integrity and the authenticity of a message. It takes as input a variable-length message and a shared secret key and generates an authentication code, that will be appended to the message and sent. The recipient can use the shared secret key and the received message to generate an authentication code and compare with the appended received authentication code [30].

In recent years, the main approach to form a MAC is to combine a cryptographic hash function, such as SHA-256, in some fashion with a secret key [30], this is known as Hash-based Message Authentication Code (HMAC). The security of any MAC function based on an embedded hash function depends in some way on the cryptographic strength of the underlying hash function. The main strength

of HMAC is that it is proven to exist an exact relationship between the strength of the embedded hash function and the strength of HMAC, so to break the HMAC, one would have to break the underline hash function [31].

MAC can also be used to generate pseudorandom numbers of fixed length. This is explored by the Bitcoin protocol to generate deterministic wallets [11, 34].

PBKDF2

A Key Derivation Function (KDF) takes an input (usually a password or passphrase) and produces a secret key that can fulfil some required format, for example, a certain length and higher entropy. More than that, it is used to prevent brute force attacks, as two elements are present in the function: the use of a salt, avoiding the pre-computation of keys and the usage of iteration, requiring more computational power to execute the function [36].

In the year of 2000 RSA Laboratories published its Public-Key Cryptography Standards (PKCS) #5 (version 2.0), defining the Password-Based Key Derivation Function 2 (PBKDF2) [36], which is not particularly bound to any specific pseudorandom function and can generate keys of arbitrary sizes.

The PBKDF2-SHA512 variant is used by the BIP-39 to create a binary seed from the mnemonic code [35].

Asymmetric cryptography

Public-key (or asymmetric) cryptography is any cryptographic system where encryption and decryption are achieved by using two different keys, one public (that can be freely distributed) and another private (that should be kept secure) [30, 31].

The strength of public-key cryptography comes from mathematical functions known as trapdoor one-way functions, which are easy to compute in one direction, yet the opposite (or inverse), is very hard to be computed without all the needed parameters. Trapdoor functions are based on problems like prime number factorisation, the discrete logarithm problem and elliptic curve multiplication [39].

Various mechanisms for asymmetric cryptography were created (e.g. RSA, Diffie-Hellman, DSS, Elliptic Curve), providing useful encryption services for a wide

range of problems, though the most used are digital signatures and the establishment of symmetric keys used to encrypt the messages in a communication channel [31].

Digital signatures

One of most important instance of public-key cryptography is digital signatures [31], with applications on a wide range of areas, including secure e-commerce, legal signing of contracts, secure software updates, online banking and Bitcoin, where a transaction is securely signed, allowing only the owner of the signing private key to spend funds associated with a referenced public key [11].

Digital signature is based on mathematics and provides three important aspects [30, 31]:

1. Authentication: the receiver can verify that the message was created by the sender;
2. Integrity: the receiver can verify that the message was not altered in transit;
3. Non-repudiation: the sender cannot deny having sent the message.

It is also relevant to note that it should be computationally infeasible to generate a valid signature for any given message without knowing the private key, while verifying the signature using the public key should be trivial.

ECDSA

Elliptic Curve Digital Signature Algorithm (ECDSA) is based on elliptic curve cryptography (ECC). It is used as an alternative to RSA/DSA and other schemes for digitally signing messages, with the advantage that the same level of security can be achieved using a much smaller key size. For instance, a 256-bit elliptic curve public key should provide comparable security to a 3072-bit RSA public key, or a 160-bit private key in ECDSA compared to 1024-bit on DSA [30, 38, 39].

Bitcoin protocol has chosen the secp256k1 curve parameters defined in the Standards for Efficient Cryptography (SEC) to make all the ECDSA operations. Unlike the NIST curves, secp256k1's constants were selected in a predictable way, which significantly reduces the possibility that it might contain any kind of backdoor [37].

This scheme is based on the elliptic curve $y^2 \equiv x^3 + 7 \pmod{p}$, defined over a finite prime field Z_p. The private key is a cryptographically secure random unsigned

integer and the public key is a point on the curve, representing the multiplication (in ECC terms) of the generator point (specified on secp256k1) by the private key [39]. Given the public key, it is unfeasible to determine the private key. This is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP) and is basically what guarantees the security of ECC [31].

To sign a message using ECDSA, considering **n** as the order of $\boldsymbol{Z}_p$ [30]:

1. Calculate the **e** as the hash of the message (e.g. using SHA-256);
2. Let **z** be the $\mathbf{L}_n$ leftmost bits of **e**, where $\mathbf{L}_n$ is the bit length of the group order **n**;
3. Select a cryptographically secure random integer **k** from **[1, n-1]**;
4. Calculate the curve point ($\mathbf{x}_1$, $\mathbf{y}_1$) = **k** x **G**.
5. Calculate **r** = $\mathbf{x}_1$ **mod n**. If **r = 0**, go back to step 3.
6. Calculate **s** = k^{-1} (**z** + **r**$\mathbf{d}_A$) **mod n**. If **s = 0** go back to step 3.
7. The signature is the pair (**r**, **s**).

It is really important to choose a different cryptographically random number on step 3, otherwise the private key can be recovered, as the famous attack on the Sony PlayStation 3 has shown [40]. Another way to ensure that this integer is unique is by computing it with a KDF of the private key concatenated with the message.

BITCOIN

Bitcoin is a *Peer-to-Peer Electronic Cash System* initially developed by Satoshi Nakamoto in late 2008 to allow any two willing parties to transact directly with each other without the need for a trusted third party [1]. It is the world's first decentralized cryptocurrency, with a market capitalization of U$ 112 billion, representing a 55% slice from the U$ 200 billion cryptocurrency space [3]. It is built upon cryptography and peer-to-peer (P2P) technology, has its own currency – called bitcoin (BTC) – that isn't bounded to any asset on the real world and is completely open source [41].

Below are summarized the main general characteristics of Bitcoin [1, 5, 42]:

Transactions are public: All Bitcoin transactions are transmitted to the network to be propagated to all participating nodes (each node is connected only to some peers, but using the network effect, all nodes will receive the transaction). The

nodes maintain a distributed database called *blockchain*, which is a public ledger with all the transactions.

Frauds are infeasible: To prevent double spending of coins and reach global decentralized consensus, the nodes on the network execute a *proof-of-work algorithm* based on cryptography hashing.

Coins are secure: To transfer bitcoins to a recipient, one must own the ECDSA's private key that will sign the transaction and release the funds.

Payment irreversible: Once the owner transfer their Bitcoins, he will no longer have the power to retrieve them without the recipient's consent.

Cheap and fast: The transactions are designed to be cheap and fast (disregarding the whole energetic cost to mine coins) as there is not a chain of intermediates between the sender and the recipient.

Partially anonymous: Although all transactions are publicly known, Bitcoin uses addresses as pseudonyms, where the addresses are not tied to a person or company, but to a cryptography public key. It is possible to track and associate all the addresses using a serie of techniques, so the system is not completely private [7].

Coins are created by miners: The incentive for the nodes (called miners) to participate on the network comes from the fact that when they check the transactions and execute the proof-of-work algorithm before all others nodes, they are rewarded with freshly created bitcoins and the fees from the transactions.

Deflationary currency: Although coins are created on the process of mining, there is a limitation of total number of coins on the specification of the protocol, only 21 million bitcoins can be created, making Bitcoin deflationary.

Figure 1 shows how the different concepts described are related.

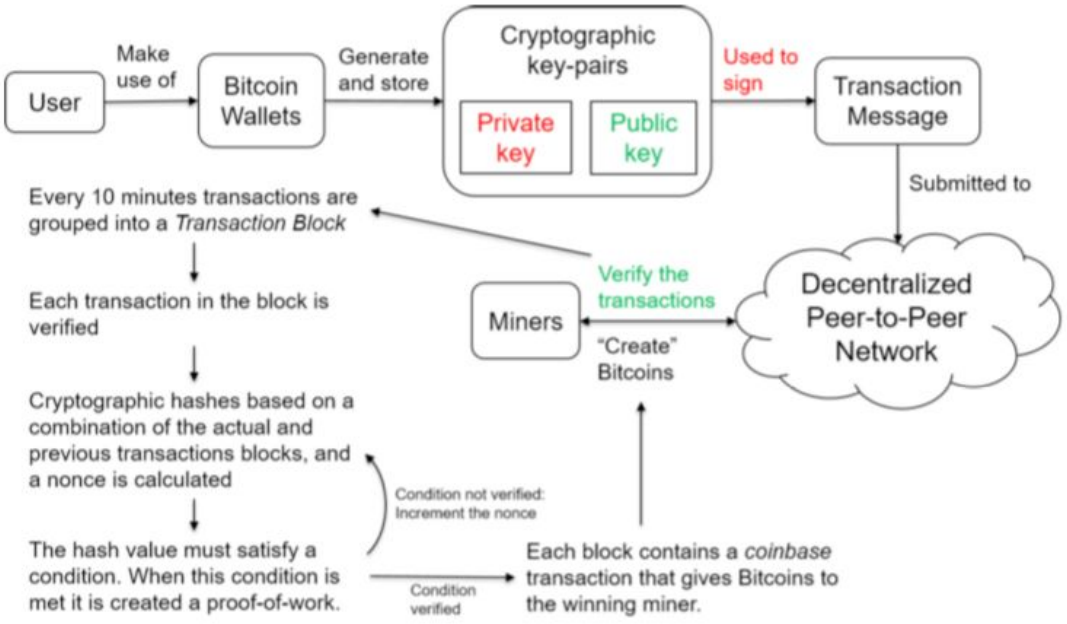


Figure 1: General view of the Bitcoin protocol and its participants.
From Miraje Gentilal, et al. (2017) [5]

Blockchain and mining

In the context of Bitcoin, the blockchain is the distributed database containing all the transactions between the users of the Bitcoin network, it is a public ledger that is resistant to modification [41]. Satoshi used the terms *block* and *chain* to describe what he calls the Timestamp Server [1], that works by grouping received transactions from the network into a block. This block will have a unique hash of its contents, a *Unix time* timestamp and a reference the previous accepted block, forming a chain of blocks that will be propagated through the network.

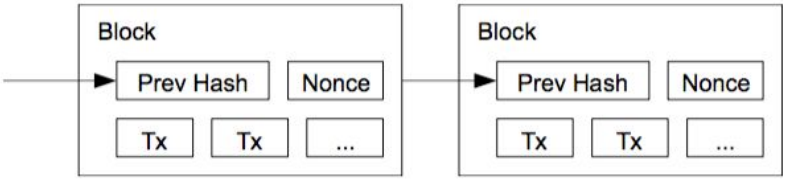


Figure 2: Chain of two blocks forming the blockchain.
NAKAMOTO, Satoshi. (2008) [1]

To prevent double spending of coins and other forms of misuse of the consensus rules, the nodes of the Bitcoin network must be able to verify the transactions that goes into the blockchain in a decentralized way. The solution is the use of a proof-of-work (PoW) system and a form of incentive to keep the nodes honest [1].

After the transactions in a block are validated, the PoW is required to make it hard to modify a block of transactions and it works by requiring the node to compute a cryptographic challenge before commiting the block. The header of the block contains a random nonce, among other information. The node must calculate the SHA-256 digest of the header and check if the computed hash contains a determined number of preceding zeros, if not, the node must increment the nonce and redo the calculation until it fulfils the requirement.

> *"Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it."* [1, page 3]

On the Bitcoin protocol, the number of zeros needed is adjusted every 2016 valid blocks to guarantee that the average time to compute a block is 10 minutes [42].

When a node computes the challenge, it can retrieve a reward by generating new coins for itself and collecting the fees from the transactions on the block. All this process of validating the blocks, executing the proof-of-work and collecting rewards is known as *mining*.

Transactions and authentication

Bitcoin transactions represent the transfer of coins between users of the network. The transactions are broadcasted and miners register them on the blockchain if they are valid according to the Bitcoin protocol [41]. To find out how much bitcoin one owns, it is necessary to check all the chain of transactions relating to one or more bitcoin address. The transactions have a defined structure, usually referencing previous transactions outputs as its input and bitcoin addresses with corresponding values as outputs [44].

A scripting system is used for transactions, making the protocol more versatile, like allowing to lock the amount for some time or to require multiple signatures to release the funds. The language of the script is simple, stack-based, and processed from left to right. It is intentionally not Turing-complete, with no loops [45, 47].

Figure 3 shows the schematic relationship of transactions and public key cryptography. In essence, to transfer bitcoins to a recipient, one must own the

ECDSA's private key that will sign the transaction and release the funds to the address indicated on the output. But not only that, as shown by Ken Shirriff [47], there are many details in the process of creating raw transactions with the correct signatures.

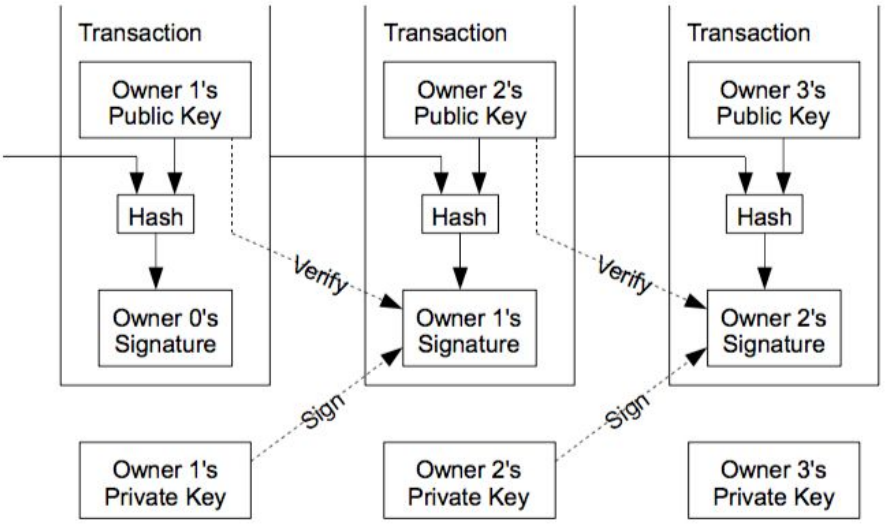


Figure 3: Relation of different Bitcoin transactions.
NAKAMOTO, Satoshi. (2008) [1]

Figure 4 shows an example of a real Bitcoin transaction. The amount referred in the input (5.31491729 BTC) was collected from 5 addresses and was sent to 3 different output addresses.

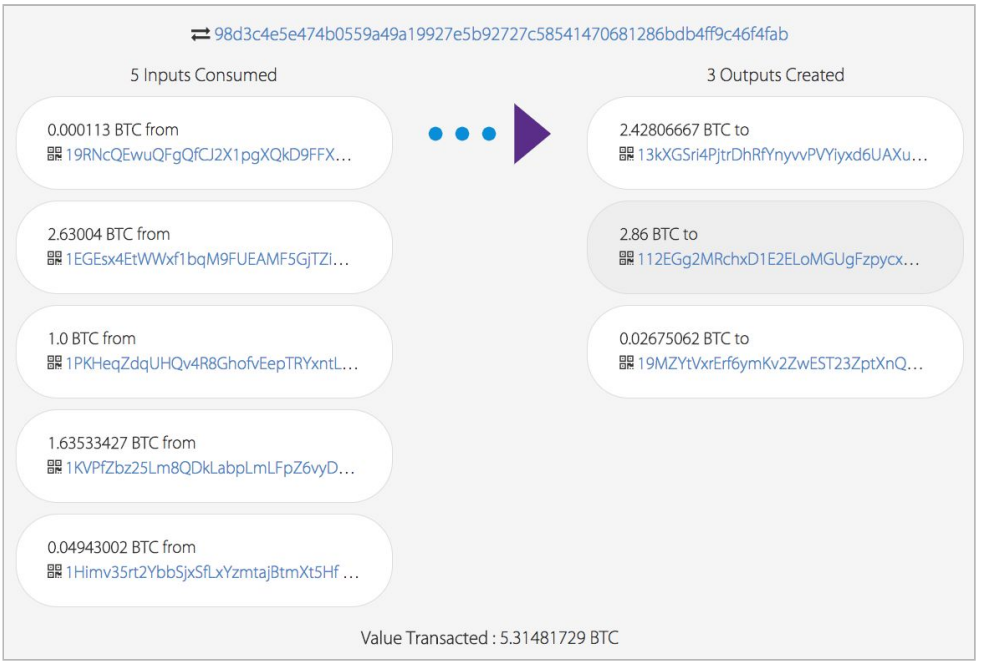


Figure 4: Example of Bitcoin transactions with its inputs and outputs [46]

KEYS MANAGEMENT, ADDRESSES AND WALLETS

Bitcoin is rooted on the cryptography ground, meaning that users must be able to deal with public key cryptography in order to use the protocol [41]. Managing private keys is a complex task and this is one of the biggest challenges for the widespread adoption of Bitcoin and other blockchain solutions, as shown by Shayan Eskandar, et al. [8].

To circumvent this situation, the Bitcoin developer's community have invested a lot of effort to make the operations related to address generation and key management more simple and user-friendly. In the next topics we explain how Bitcoin address are generated and how *wallets* come as a rescue for the ease of managing user's funds.

Bitcoin wallets

Bitcoin wallets generally refers to the client software used to manage bitcoin private keys, generate addresses, forge transactions and aggregate balance information from the Bitcoin network. Wallet may also refer to the data structure used to store and manage user's keys [43], but we will sticky with the broader definition.

Wallets comes in different formats and flavors, ranging from simple local storage of keys on a desktop computer, to mobile applications, dedicated hardware devices, paper wallets or even brain wallets [8]. All of them can be categorized as a *hot* or a *cold* wallet [5]. Further in this chapter we will detail both flavors, but first we must understand how Bitcoin addresses works.

Address generation

A Bitcoin address is an identifier that represents a possible destination for a Bitcoin payment and it is deeply related to a set of cryptographic keys from a user, as it is derived from the public key or from a series of transformations based on the private key [11, 43, 48, 49].

It contains 26-35 alphanumeric characters and can be generated at no cost by any user, without the need for an internet connection. Except the newer Bech32 [50]

format, all addresses are case sensitive. The addresses are Base58 encoded and have a checksum to prevent invalid address usage [49].

It is important to note that unlike email address, Bitcoin addresses should not be used in more than one transaction [48], as this brings serious privacy and security risks [51]. To prevent this, all software related to address generation should enforce fresh generated addresses.

There are currently three address formats in the Main network:

- Pay To Public Key Hash (P2PKH) which begin with the number 1, eg: 1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2
- Pay To Script Hash (P2SH) type starting with the number 3, eg: 3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy
- Bech32 type starting with bc1, eg: bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq

Figure 5 shows the necessary steps to generate a Bitcoin address from a ECDSA public key. Note that to prevent length-extension attacks, hashes are always applied twice [32].

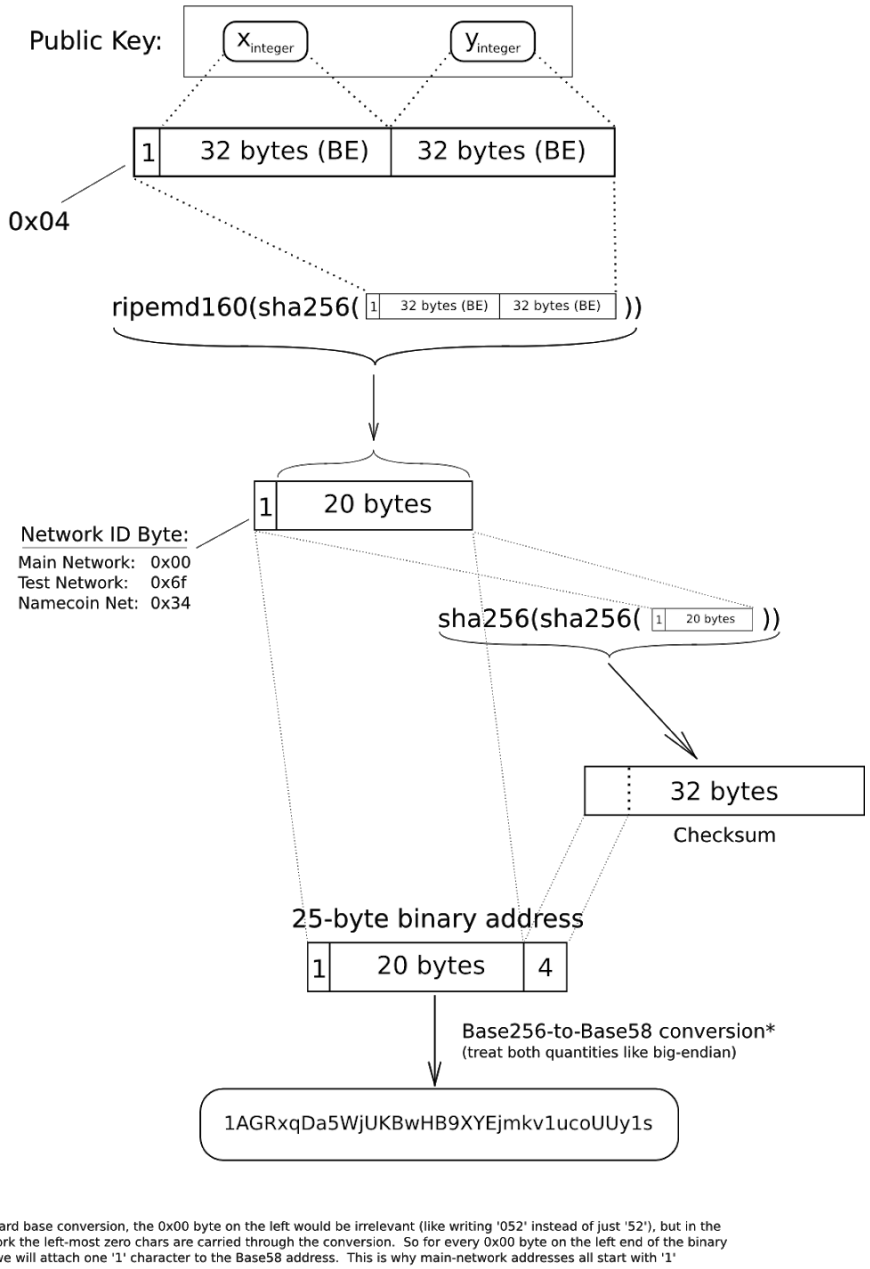


Figure 5: Conversion from ECDSA public key to Bitcoin address version 1 [49]

Deterministic generation of wallets

The first generation of Bitcoin wallets, like the Bitcoin Core Wallet, generates random keys and store them on the local filesystem, encrypted by a password or not [8, 43]. This approach presents headaches and serious security exposure. The user has to constantly backup the keys to other devices and trust that the computer is not infected by any kind of malware and that no unauthorized person has access to it.

To get around this situation, in the year of 2012 the BIP-32 was introduced, proposing a new way to generate addresses, know as Hierarchical Deterministic

Wallets (HD wallets) [34]. In this kind of wallet, any number of keys can be derived from a single master key, called *seed*, forming a hierarchic tree of accounts, each with its public and private keys, as shown on Figure 6.

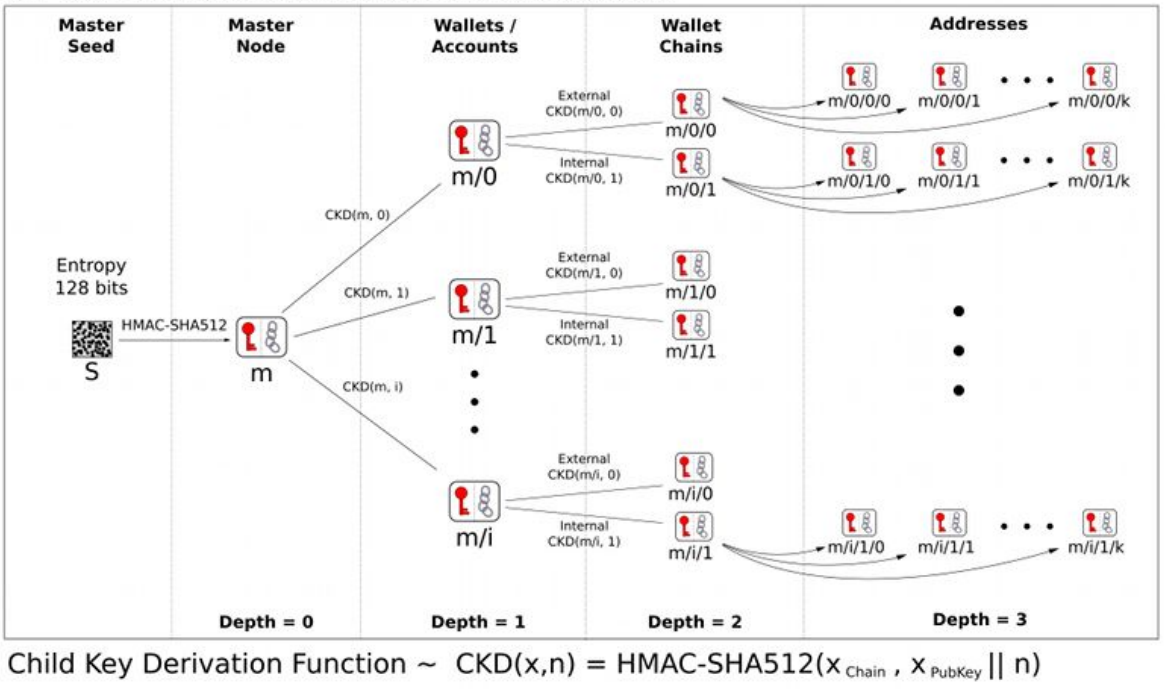


Figure 6: Hierarchical Deterministic Wallets with different depths [34]

This scheme has many of advantages over classic non-deterministic wallets [5, 11, 34, 43]:

Easy to backup: By making a backup of the seed just once, one can recreate the full tree of accounts at any given time.

Independence of branches: Each child account has its own extended private and public key, making it possible to share just part of the tree. For example, in a ecommerce site, it is possible to generate unique receiving addresses for each order by utilizing the extended public key the external chain of a single account.

Organizational structure: The tree structure can be given meaning, for example, allowing the headquarter of a business to monitor all the offices addresses with just one seed, but delegating control to each one of them independently.

Fresh addresses: The receiving and change addresses is always freshly generated, so it is never used in more than a single transaction.

Audit capability: As the full chain of address can be generated using a extended public key, the key can be handled to external auditors to check the balance of a tree, for example, when a exchange needs to prove its solvency.

As of 2018, practically all wallet client software supports HD wallets. The implementation details can be seen on the BIP-32 specification [34].

Seed phrase for recovering the master key

Another important improvement added to wallets is the ability to use seed phrase for generating deterministic keys (BIP-39) [35]. A seed phrase is easier to handle instead of raw binary or hexadecimal representations of a wallet seed. The sentence can be written on paper, spoken over the telephone (not adviced) or even registered on resistant metal cards and locked in traditional bank safes.

It is important to note that whomever has access to the seed phrase has full control over all the balance associated with the addresses derived. To mitigate the risk of exposing the funds, it is possible to extend the seed phrase with an additional password [52]. This also provides plausible deniability, because every password will generate a valid wallet.

The seed phrase is generated from a random number that is converted to a phrase with 12, 15, 18, 21 or 24 words from a standard wordlist, allowing different wallet software to generate compatible seed phrases. With 12 words the generated entropy is 128 bits and with 24 words the entropy is 256 bits. The order of the words matter and the phrase contain a checksum [35]. To make a HD wallet from a seed phrase, the words are used to derive a longer seed through the use of the key-stretching function PBKDF2 with HMAC-SHA512 [43].

Hot and cold wallets

All wallets can be subdivided in two main categories: *hot*, which is connected to the internet and *cold*, which isn't. Usually hot wallets are associated with everyday usage on desktop and mobile phone, providing more convenience over cold wallets, which were created for long-term storage of larger amounts of cryptocoins [5, 7, 10, 41].

All the facility from using a hot wallet comes with a major price: the risk of losing all the funds to hackers or the third parties responsible for the wallet service [8]. The Bitcoin history is full of such cases, in fact more than U$ 15 billion were lost since 2013 due to hacking [2]. It is important to remember that online exchanges are included in the category of hot wallets, despite holding most of its funds under cold storage, the final user is anyhow exposed to the mentioned risk.

Cold storage comes in different forms, here are some [53]:

Paper wallets: The keys are printed or written on paper.

Engraved in metal: The keys are engraved, etched, ablated or stamped in a piece of metal.

Stored digitally on data devices: The keys are recorded on a USB drive or other data storage medium.

Brainwallet: The seed phrase of a wallet is stored on one's own mind by memorizing it.

Dedicated hardware device: The keys are generated and stored in a secure hardware device designed from the root to be a wallet and nothing else [16].

We will focus on the hardware wallets, as they offer the best balance between very high security and ease of use. Besides storing the keys, these devices are capable of signing the transactions on its dedicated secure enclave, like a smart card, so the keys never leave the device. The main hardware wallets on the market are the Ledger Nano, Trezor, KeepKey and Digital BitBox [5]. All of them have a companion mobile or desktop application, usually are open source and support different cryptocoins [54].

Most of them have a dedicated screen and buttons, so that the user can check the transactions before confirming, copy the seed phrase securely and input additional password or other information without being exposed to malware on the computer that the wallet is connected [12].

Hardware wallets have a great track record but are not silver bullets [12, 54], they can be prone to hardware and software bugs, have insecure Random Number Generator (RNG), be exposed to a compromised production or shipping process and many others [54]. In this work we will approach some of this factors further ahead.

A note on the Bitfi hardware wallet

There is a new hardware wallet on the market called Bitfi, released in July of 2018 [55]. The creators of the wallet claims that it is unhackable, because "there is nothing to be hacked", as the seed phrase is never stored on the device, but instead, it is inputted by the user when he needs to sign a transaction, derivating the private keys on the fly. Although the device seems not to store the keys when turned off, there might be some kind of exploitable aspect on the communication channel or some other part of the device. They recommend the users to generate the seed words using the Diceware method [56] and store them on their own mind. This approach might be interesting, but we must consider that the general advice from community is that one should not rely on human mind to be the only storage place of such an important information [57]. As this kind of hardware wallet needs further investigation by the community, we will not consider this approach for our implementation and will focus on the requirement that the device keeps the keys on its secure memory. We must also take into consideration that if a greater level of security is required, the way to go should be threshold cryptography [25], that can be latter incorporated in our implementation.

SECURE ELEMENT

According to GlobalPlatform, a non-profit industry association focused on making specifications and certifications for the security field:

> *"A SE is a tamper-resistant platform (typically a one chip secure microcontroller) capable of securely hosting applications and their confidential and cryptographic data (for example cryptographic keys) in accordance with the rules and security requirements set by well-identified trusted authorities." [61, page 2]*

Secure elements comes in different flavors, depending on the requirements of the project. They can be embedded and integrated SEs, SIM/UICC, smart microSD as well as pocket-sized smart cards. With the exception of *Trezor*, the majority of the hardware wallets available employs a Secure Element (SE) in its design [54].

In this chapter we will give a brief overview of this technology.

Anti-tamper and certification

SEs are designed to be tamper resistant, not allowing an attacker to retrieve or modify information in the its secure memory [62]. Although it is getting harder to tamper with them, there are known flaws [18, 21, 22, 23, 24, 26] and different levels of security. However SEs still represent the best available practical solution for the price and are being constantly enhanced.

Kömmerling, O. and Kuhn M. [26] distinguish four major attack categories in SEs:

Microprobing can be used to access the chip surface directly, thus one can observe, manipulate, and interfere with the integrated circuit.

Software attacks use the normal communication interface of the processor and exploit security vulnerabilities found in the protocols, cryptographic algorithms, or their implementation.

Eavesdropping techniques monitor, with high time resolution, the analog characteristics of all supply and interface connections and any other electromagnetic radiation produced by the processor during normal operation.

Fault generation techniques use abnormal environmental conditions to generate malfunctions in the processor that provide additional access.

As there are many possible attacks and different ways of engineering a SE chip, it is hard to say that a chip is really anti-tamper or at least that the cost to tamper it is too high. The solution for that comes in the form of certification from external auditors. There are two main certificates in this area:

CC EAL: Common Criteria (CC) Evaluation Assurance Level (EAL), which has 7 levels, 1 being the most basic (and therefore cheapest to implement and evaluate) and 7 being the most rigorous (and most expensive) [63]. The *Ledger Nano S* has a CC EAL5+ certification [58].

FIPS 140-2: An NIST standard with the requirements for a secure cryptographic module containing 4 levels of certification, being 4 the highest level of security [64].

Java Card

Java Card is a platform released in 1996 to simplify the development of software for smart cards by offering portability and security [18, 19]. It is the leading platform, with more than 10 billion devices deployed [19, 20], mainly Subscriber Identity Module (SIM) chips for mobile phones and credit cards for the banking industry. Java Card products are based on the Java Card Platform specifications, which allows a Java Card applet to be written once and run on different smart cards running a virtual machine on top of its operational system (Figure 7).

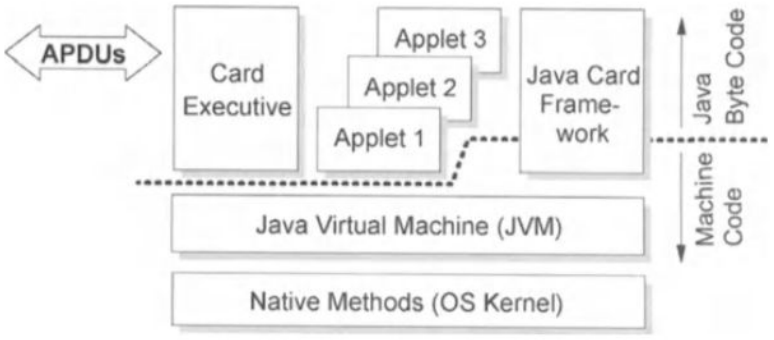


Figure 7: Software Stack of a Java Card [19]

Palma L. and Sousa L. highlight the main advantages of the platform:

Easy of use: Developers can focus on the application code rather then be concerned about hardware aspects of the smart cards.

Security: Java cards employ different layers of security to guarantee the isolation of the applets on the card, not allowing an applet to access its neighbor memory and storage.

Hardware independence: Applets can be compiled once and deployed on different compatible cards with the targeted Java Card API version.

Multiple applets: Different applets with completely independent functionality can coexist on the same card.

Compatibility: Java Cards are based on the ISO 7816 international standard for smart cards, managed jointly by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).

Java Card applets are developed in a tiny subset of the Java language with many of restrictions [66]. There is no support for char, double, float, long,

multidimensional arrays, dynamic class loading, threads, object cloning and there are different limitations on Java core API classes. Yet many of the familiar features of a Java application are available: including objects, inheritance, packages, dynamic object creation, virtual methods, interfaces, and exceptions. The smart card vendors can provide additional functionality by extending the default Java Card API.

Each applet on a card is uniquely identified by an Application ID (AID) (as specified in the 7816-5 ISO) and must extend the Applet abstract base class, which defines the methods used by the Java Card Runtime Environment (JCRE) to control the applet life-cycle (Figure 8). First the applet is downloaded to the card and the JCRE invokes the applet's static *Applet.install()* method, and the applet registers itself with the JCRE by invoking *Applet.register()*. When the applet is installed and registered, it is in the *unselected* state, available for *selection* and *processing* by responding to custom commands.

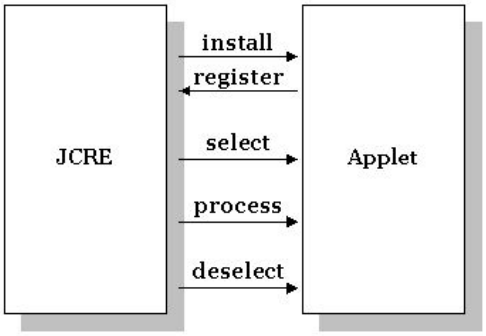


Figure 8: The Java Card Applet Life-Cycle Methods [66]

APDU

Beyond developing the application that runs on the smart card, one must implement an application that communicates with the applet using a card reader, USB interface or some wireless communication protocol [19, 22, 65, 66]. Those two pieces of software exchange information using the Application Protocol Data Unit (APDU), conforming with the ISO 7816-3 and 7816-4 (Figure 9). There are two kinds of APCU: *command* and *response*.

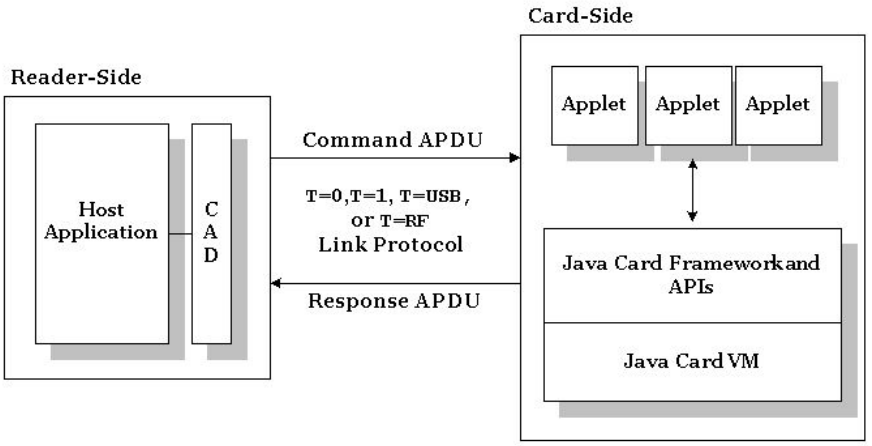


Figure 9: Communicating Using the Message-Passing Model [66]

The command APDU is sent by the reader to the card with a mandatory 4-byte header with the type of the command, command and parameters (CLA, INS, P1, P2) (Figure 10). A response APDU is sent by the card to the reader containing from 0 to 65.536 bytes of data, and 2 mandatory status word bytes (SW1, SW2) (Figure 11).

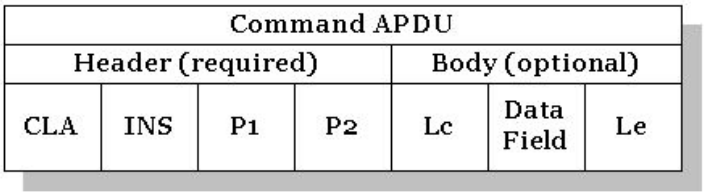


Figure 10: Structure of a command APDU [66]

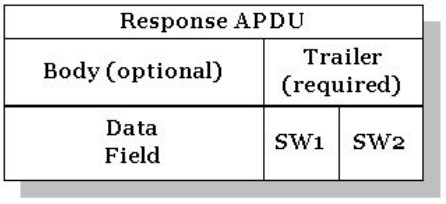


Figure 11: Structure of a response APDU [66]

Global Platform

GlobalPlatform is a non-profit association driven by over 100 member companies, from chip manufacturers to the communication industry [67]. The GlobalPlatform specifications consists of three parts [19]:

Card: Defines the personalization process and standard ways for customizing the card after it has been issued, also providing the card issuer with tight control of the card and of the process of loading new applications onto it.

Device: Also called terminal specification, focuses on the standardization of stand-alone payment terminals.

System: It is the latest addition to the family of specifications covering infrastructure, processes, and systems required to manage a multi-application card and its content.

The main components include the *Runtime Environment* consisting of a secure multi-application card runtime; the *GlobalPlatform API* which provides services like a secure channel for communication, the verification of the cardholder and personalization of the card; the *Card Manager* containing the ability to perform application installation and selection, command dispatching, card content management and Personal Identification Number (PIN) support; the *Security Domains* enabling the applications of various providers to share space on a card without compromising the security of any particular provider.

GPShell, *GlobalPlatformPro* and *OpenSC-Tool* are tools that implement the GlobalPlatform specification and allow the interaction with the Java Cards.

RELATED WORK

In this section we detail three related papers that were important for the development of this work and the collection of additional references.

Low-Level Attacks in Bitcoin Wallets

Gkaniatsou A., Arapinis M. and Kiayias A. released a paper in 2017 [12] where they bring to attention that the use of EAL5+ certified smart cards on a dedicated hardware wallet doesn't guarantee its security, as the low-level communication protocol can be exploited. They focus on the *Ledger Nano S* [58], by reverse-engineering the APDU communication protocol and mounting a series of MitM attacks (Table 1) to demonstrate the vulnerabilities.

a. Direct wallet attacks	**b. Transaction attacks**	**c. Account privacy attacks**
a.1 Access to the master private key;	b.1 Tamper the payment amount;	c.1 Account traceability.

A.2 Access to the keypool encryption key; a.3 Unauthorised access to the wallet; A.4 Alter the wallet security properties.	b.2 Tamper the payment address; b.3 Denial of service.	

Table 1: Attacks demonstrated by Gkaniatsou A., et al. [12]

The attacks are really serious and serve as a reminder of the risks of sending clear text data through an insecure channel. The authors propose to secure the communication channel selectively, by using symmetric cryptography only on pieces of data that must be confidential. They suggest the use of the Password Authenticated Key Exchange (PAKE) by Juggling protocol (j-PAKE) [59] which *"allows bootstrapping high entropy keys from the low-entropy user's PIN. In that way, we avoid storing secret data API side, ensure that fresh keys are used in each session and guarantee the user's presence at that session"* [12].

Their solution can be applied to any hardware wallet and will be considered in our implementation.

SRP on Java Card applets

As shown by Gkaniatsou A., et al. [12], having a secure communication channel is a crucial requirement for a hardware wallet device. With that in mind we investigated different possibilities to deploy this on a smart card and found the work of Hölzl, M., et al. (2015) [27] to be very enlightening. The authors design, implement and evaluate the use of the password-authenticated secure channel protocol (SRP) to protect the communication of a mobile application to a Java Card applet running on a smart card.

The SRP protocol is an augmented password-authenticated key agreement (PAKE) protocol, specifically designed to work around existing patents. It allows one party (client) to demonstrate to another party (server) that he knows the password, without actually sending the password, nor any other information from which the password can be retrieved, accomplishing what is known as zero-knowledge password proof [60]. The authors choose SRP-6a over other possibilities because it

is based on Diffie-Hellman key exchange, which is has a wider support on Java Card 3.0, but end up also presenting the SRP-5 elliptic curve variant [27].

The referred work presents a really high level of detail and effort in the optimization of the algorithms. The resulting performance of the SRP-6a variant is 1,600 ms on an external smart card, while the SRP-5 variant runned in 526 ms. The values might seem high, but it is important to note that most of the computation happens on the key agreement phase, which runs simultaneously with the password or PIN entry.

Usability analysis on wallets

One of the objectives of this work is to provide an implementation of a Bitcoin wallet, which will inevitably include a client user interface to serve as a glue between user's needs, the Bitcoin network and the secure hardware. Beyond our informal research on current wallets products, we selected an academic paper by Eskandari S., et al. [8] where the main wallets solutions until the year of 2015 are mapped, studied and compared, mainly in the point of view of usability and user experience.

Category	*Example*	Malware Resistant	Key(s) Kept Offline	No Trusted Third Party	Resistant to Physical Theft	Resistant to Physical Observation	Resilient to Password Loss	Resilient to Key Churn	Immediate Access to Funds	No New User Software	Cross-device Portability
Keys in Local Storage	Bitcoin Core			●		●	●	●	●		
Password-protected Wallets	MultiBit		○	●	○	●		●	●		
Offline Storage	Bitaddress	○	●	●			●				●
Air-gapped Storage	Armory	○	●	●		●	●	●			
Password-derived Keys	Brainwallet		●	●	○			●	●	●	●
Hosted Wallet (Hot)	`Coinbase.com`						●	●	●	●	●
Hosted Wallet (Cold)		○	●				●	●		●	●
Hosted Wallet (Hybrid)	Blockchain.info		○	○			●	●	●	●	●
Cash		●	●	●		●	●	●	●	●	●
Online Banking							●	●	●	●	●
Our solution		●	●	●	○	●	●	●	●	○	●

Table 2: A comparison of key management techniques for Bitcoin. Modified from *Eskandari S., et al. [8]*

The authors elaborate a framework composed of ten security, usability and deployability criterias to enable direct comparison of different key management solutions (Table 2). Then they define four main tasks (configure, spend coins, spend

coins on a secondary device, recovery of wallet) and comprehensively walkthrough the defined talks on six different wallet clients, evaluating the usability based on predefined heuristics.

Table 2 has received an additional last line with our solution to allow comparison with the others proposals. As the keys are stored on a secure element, which cannot be tampered, we gave *Malware Resistant* a full point (●). On the *Resistant to Physical Theft* aspect, we gave our solution the half point (○) as the device can be stolen but no keys can be extracted. Finally, on the *No New User Software* we also gave our solution the half point (○), considering that the user will access the wallet through a web interface, but it is necessary to install the *Connector* (described in the development chapter) software or at least an *extension* to the web browser.

Additionally, the authors conclude that *"the metaphors and abstractions used in the surveyed clients are subject to misinterpretations, and that the clients do not do enough to support their users."* [8], which is a very important indicative that in order to make a successful wallet solution, extra attention should be given to the user interface design, including the choice of texts, metaphors and additional help content.

Paradoxically, despite being a disruptive innovation, Bitcoin borrowed concepts from the old financial order, like *wallet*, that brought confusion to users, as these concepts actually have different meanings on the traditional system.

DEVELOPMENT

In this chapter we will effectively dive into the details of a Bitcoin hardware wallet project.

A hardware wallet can be constructed in many different ways, bringing unique security advantages, implementation challenges and costs of production for each version. In the first part of this chapter, we will analyze those differences. Next, we advance to a discussion of security aspects of the project. Finally, we will detail the planning and implementation of a working prototype using Java Card and a web user interface to interact with the device and the Bitcoin network.

Different possibilities for the hardware wallet

As previously mentioned, a hardware wallet can be built in many different ways, as shown by some products on the market: the *Ledger HW.1* [84] is a smart card that fits a USB port, the *Ledger Nano S* [58] is a version with a smart card, a little monochrome screen and two buttons, the *TREZOR Model T* [85] has a little full color touch screen but no secure element and the *Ledger Blue* [86] has an even bigger screen and more powerful processor.

As we've seen in the first chapter, it is really important that the private keys are kept on a secure element, making the smart card (Java Card) the basic requirement in our analysis. Another general precondition is the communication with an external computer (desktop, notebook or even an mobile phone), so the wallet can interact with the Bitcoin network and the user, usually through an user interface (web, desktop or mobile). This connection can be done with a smart card reader, an USB port, Near Field Communication (NFC) or even Bluetooth. We will generalize the term *connection* and detail when needed. This relationship is shown on Figure 12.

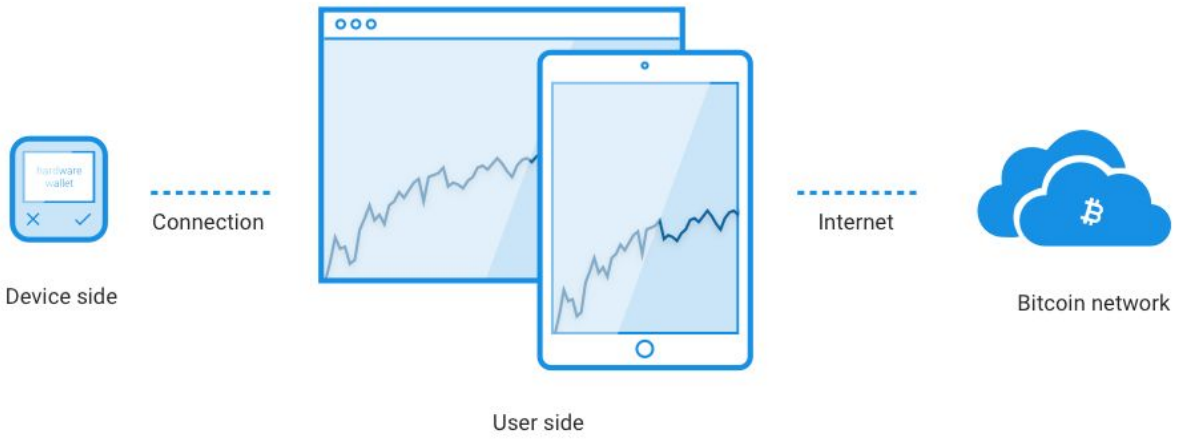


Figure 12: Relationship between the device, the user interface and the Bitcoin network.

When we talk about requirements, we will list the most basic ones. Requirements related to more advanced security aspects, like the encryption of the communication channel to prevent eavesdropping and MiTM attacks, will be detailed further in the second part of the chapter.

Smart Card + Connection

This is the most primitive and cheap version of the wallet, even though it will be able to perform the basic requirements of a Bitcoin wallet. One should choose a Java Card that can perform ECC natively to gain better performance.

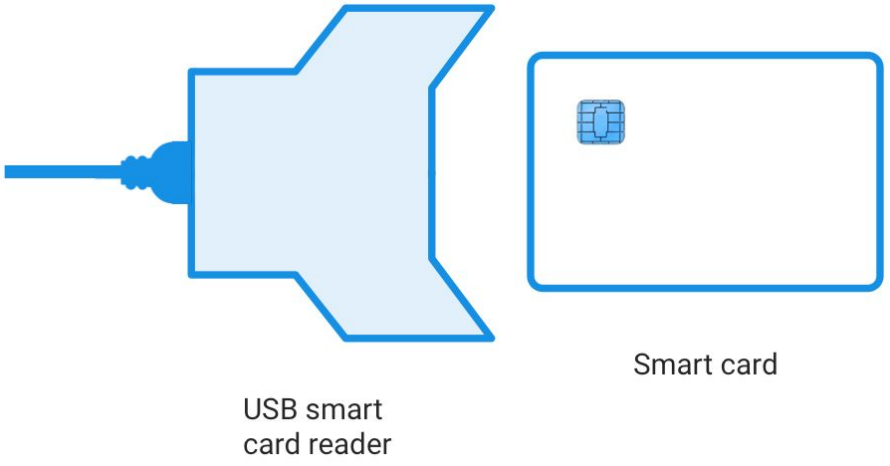


Figure 13: Simple smart card with the USB smart card reader.

If the Java Card doesn't have an USB interface, one must use a smart card reader to interact with it, making the project more expensive (Figure 13).

Example: NXP JCOP J3D081 EV1

Cost: U$ 10 (for small quantities)

Implementation: It is necessary to define all the commands available for the card, its different states and to develop the Java Card Applet using one specific version of the SDK.

Basic requirements

1. **SETUP**
   - 1.1. To define/change a PIN to access the device;
   - 1.2. To generate the initial master private key of the device using a True Random Number Generator (TRNG) and the seed words for recovery (BIP-39)[1];
   - 1.3. To recover a wallet by defining the master private key using the seed words;
   - 1.4. To verify that the seed words generates the master private key stored on the device.
2. **ADDRESS GENERATION**
   - 2.1. To generate deterministic BIP-32 address (public/private keys).
3. **SIGN TRANSACTIONS**
   - 3.1. To sign Bitcoin transactions using the keys of a BIP-32 path;
   - 3.2. To validate transaction details before signing (optional, adds more security).
4. **INTERACTION**
   - 4.1. To validate the user PIN;
   - 4.2. To take actions based on PIN policies, like to wipe the device if the PIN is wrong for 5 consecutive times.

[1] The seed words should be sent to the user only once, when the seed is generated.

Limitations

Exposure of the PIN and the seed words: This is the main limitation of this simplified version, as the user must interact with it using the screen and keyboard of a computer, which might be infected with viruses, keyloggers and other malware that can trick the user into signing a forged transaction, leak the wallet's seed words or the device PIN.

User confirmation can only be done by PIN: As the device has no buttons or touch screen, the user can only confirm to sign a transaction by entering the PIN on the computer device.

Firmware update restricted: To update the Applet on the card, one would have to have access to the deployment keys configured for that card, making the update only possible on the manufacturer.

Smart Card + Connection + Microcontroller

The lack of a screen introduces a big security flaw in the wallet, making it impractical for real world usage. As we couldn't find a smart card on the market that has a screen integrated and we must only use easily available parts to build our device, we need to make the integration with a screen ourselves. For that, we need a microcontroller that will be able to make the interconnection between the smart card, the screen, the optional buttons and the communication channels (like USB, Bluetooth, etc.).

This will increase the cost of the device and make the development more complex, but it is an unavoidable requirement, as we cannot trust a third party device screen (computer, notebook or even a mobile phone with Secure Enclave). An example is shown on Figure 14, the *Ledger Nano S* hardware wallet.

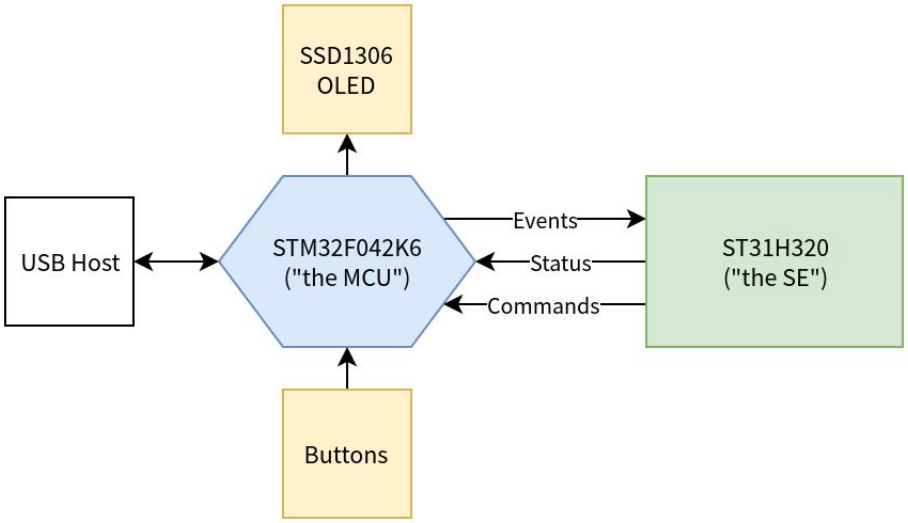


Figure 14: Integration of the hardware components of the Ledger Nano S using a microcontroller unit (MCU) [71].

We will need to write the code for the microcontroller, which can be done in many different ways, usually divided in two parts: a bootloader and the firmware. As this microcontroller is not a secure element, it will have to authenticate itself with the smart card to prevent malware on the code of the microcontroller. This is a complex topic that we will discuss on the Security Considerations section, further in this chapter. An important point for the project is that if we can find a process to validate that the code on the microcontroller has not been tampered with, then we can move some code that deal with Bitcoin transactions and address generation from the Java Card (which is hard to code and debug) to the firmware (which can be easier to program, debug and validate).

Smart Card + Connection + Microcontroller + Buttons + Screen

This configuration is the most common among the hardware wallets, as it allows one to safely store the private keys on the secure element and to offer a trusted screen and buttons.

Figure 15: Example of a bitcoin wallet asking for the confirmation before sending bitcoins.

The screen will be used to show important information for the user, for example the destination address and amount of a transaction that it is going to sign (see Figure 15), the seed words on the first setup, etc.

It is recommended at least two buttons, one to confirm and the other to cancel operations. It is possible to include number buttons (0 to 9) to allow the user to enter the PIN on the device itself or to use the screen to display random positions for the numbers so the user can enter the PIN on a untrusted computer without revealing it (more on this on the Security Considerations chapter).

Example:

- Secure Element: NXP JCOP J3D081 EV1: U$ 10 (for small quantities)
- Microcontroller and screen: STM32F429I (includes de STM32F429ZIT6 MCU and a 2.4' LCD screen): U$ 20 (for small quantities)

Cost: U$ 30.

Implementation: Beyond developing the Applet for the Java Card, a new requirement is the coding of the bootloader and the firmware of the microcontroller. There are open source solutions that can be used as a starting point, like the one found on the *micropython board* [69], the *Coldcard* [68], *Trezor* and *Ledger* wallets.

Additional basic requirements (smart card)

1. **INTERACTION WITH THE MICROCONTROLLER**

1.1. To verify authenticity of the firmware;

1.2. To establish an encrypted connection with the microcontroller (optional);

1.3. To respond with the details of the transaction that it is going to be signed;

1.4. To respond with random positions for PIN entry.

Basic requirements (microcontroller)

1. BOOTLOADER

1.1. To load and verify the firmware code;

1.2. To manage the memory of the microcontroller;

1.3. To make services available to the firmware, like USB access, screen access, buttons access, smart card access, etc.;

1.4. To validate itself with the smart card;

1.5. To validate and upgrade the firmware code.

2. FIRMWARE

2.1. To interact with the smart card, bridging the requests from the user side to the smart card side;

2.2. To display different information on the screen and deal with buttons interaction;

2.3. To upgrade the smart card Applet code.

Limitations

Still requires the user side computer: Although this version is more robust and complete, it requires the connection with a computer to access the Bitcoin Network and manage the wallet, as the processing power of the microcontroller is limited and the screen size is really small.

Introduces a big challenge on the validation of the code on the microcontroller: With the introduction of the microcontroller proxy, in addition to adding more code to the project, one must find a clever way to validate the code that runs on the firmware, so we can prevent *Evil Maid* and *Supply Chain* attacks.

Smart Card + Connection + Powerful Microcontroller + Buttons + Big screen + Internet access

If we add a bigger screen to the device, a better microcontroller and access to the internet through wireless connection, then we can offer the user a full experience with a Bitcoin wallet on the device itself, allowing a user to create and sign transactions, see the accounts balance and more.

This seems like a really good option, but there is a catch: we start to open differente of attack vectors as the software and hardware on the device gets more complex and fully exposed to the internet. Because of this we will not enter in the details of this version, but leave it as a provocation for the reader. Maybe there is a way to do it securely?

Smart Card + Connection (NFC) + Mobile Phone

This version is an alternative to the simplory version of just the smart card without any display screen or buttons. It could use NFC (that comes natively in a great number of smart cards) to interact with the user on the mobile phone screen. With the advance of the ARM's TrustZone™, others Trusted Execution Environment (TEE) solutions and the concept of Trusted User interface (TUI) [70] we could securely display the user seed words and require confirmations on the users mobile phone.

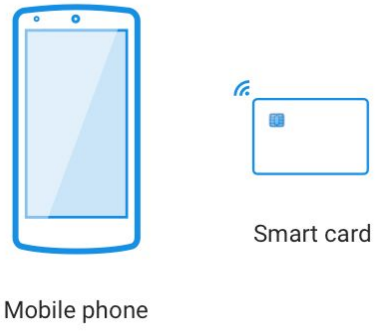


Figure 16: Smart card with NFC enabled and mobile phone as the interface.

This solution is really appealing, as the cost can be very small and the user interface can be natively mobile and easy to use.

Example: Java Card JCOP J3D081, from NXP EV1

Cost: U$ 10 (for small quantities)

Implementation: Would be necessary to make some changes on the Applet to support sending data via NFC and to wait for the user confirmation of actions. The user interface on the mobile phone would also need to be developed.

Limitations

Need to trust the user mobile phone: We must trust the user mobile phone, which is hard to do specially on the Android platform, where applications have more freedom to access the system resources without asking for specific permission.

Security considerations

In this section we discuss some specific security aspects of the project, presenting possible solutions.

Communication channel

As we've seen on the first chapter, it is really important to guarantee that the information flow is secure and encrypted, preventing MiTM attacks and its variations.

The first solution is to use the work of Hölzl, M., et al. (2015) [27], which executes a password-authenticated SRP, where the password is the user's PIN.

The second possibility is to use Elliptic Curve Diffie-Hellman (ECDH) with fresh public keys to establish a session shared-secret to encrypt the channel, ascertaining the integrity and authentication for each APDU. This solution is used by the *Status Wallet* [72]. The general steps would be as follows:

1. The client selects the application on card and the application responds with a public EC key.
2. The client sends a command to open a secure channel with its fresh session public key. The ECDH algorithm is used by both parties to generate a shared 256-bit secret.
3. The generated secret is used as an AES key to encrypt all further communication. Cipher Block Chaining (CBC) mode is used with a random IV

generated for each APDU and prepended to the APDU payload. Both command and responses are encrypted.

4. The client sends a command to verify that the keys are matching and thus the secure channel is successfully established.

This solution will do a good job, but won't prevent a well equipped hacker to intercept the communication bus between the client and the smart card, listening to the initial exchange of public keys.

PIN entering

One important aspect of the hardware wallet is the PIN protection, as it will be required to allow the signing of transactions and even to establish a secure channel. In our implementation we use the *OwnerPIN* class from the Java Card API. This class protects against attacks based on program flow prediction and is resilient to side channel attacks.

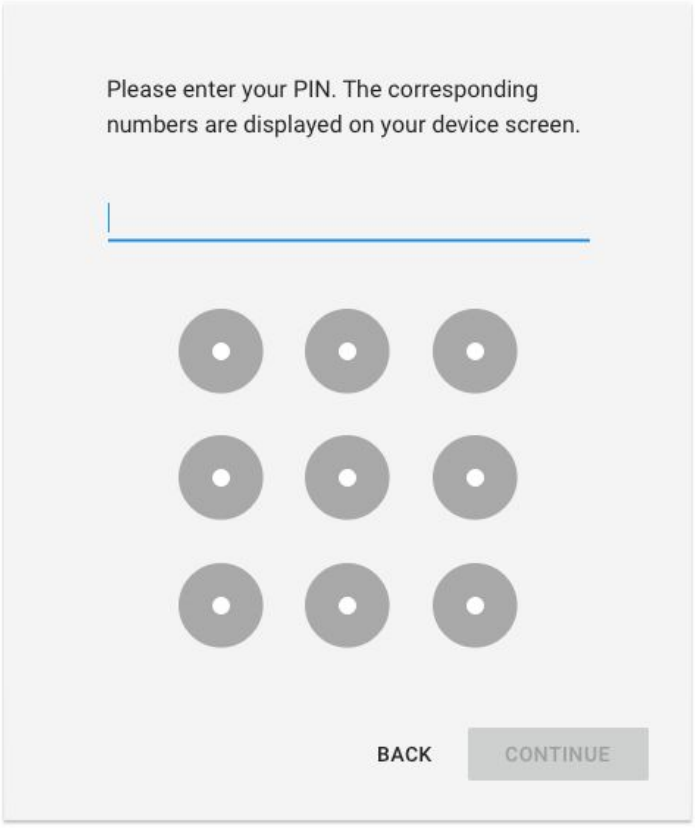
The snippet below demonstrates how to store and validated the PIN.

```
// Initialize the PIN class
OwnerPIN walletPin = new OwnerPIN(WALLET_PIN_MAX_ATTEMPTS, WALLET_PIN_MIN_SIZE);

// Store a new PIN on setup the the wallet
Util.arrayFillNonAtomic(scratch256, (short) 0, WALLET_PIN_SIZE, (byte) 0xff);
Util.arrayCopyNonAtomic(buffer, offset, scratch256, (short) 0, walletPinSize);
walletPin.update(scratch256, (short) 0, WALLET_PIN_SIZE);
walletPin.resetAndUnblock();

// Check if PIN is valid
if (buffer[ISO7816.OFFSET_LC] != walletPinSize) {
  ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
}
Util.arrayFillNonAtomic(scratch256, (short)0, WALLET_PIN_SIZE, (byte)0xff);
Util.arrayCopyNonAtomic(buffer, ISO7816.OFFSET_CDATA, scratch256, (short)0,
walletPinSize);
if (!walletPin.check(scratch256, (short)0, WALLET_PIN_SIZE)) {
  if (walletPin.getTriesRemaining() == 0) {
    reset();
  }
  ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
}

// Check if PIN is already validated
if (!walletPin.isValidated()) {
  ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
}
```

Note that the *reset()* method is called when there is no more tries remaining, effectively doing a hard reset on all the variables containing sensitive information, like the private key, the device key and the PIN.

Another relevant aspect of PIN entering is to try to prevent it from being captured by malware software on the user side. We've seen on the previous chapter that a device might have buttons and a screen. If it has two buttons and a screen, the user might use the buttons to cycle between the numbers and enter the PIN on the device itself. It might even have buttons for each number or a touch screen, also allowing the entering on the device itself.

In contrast, if we do not have a touch screen, we might reproduce a clever scheme created by *Trezor* [73] that does not expose the PIN even if the user screen is being captured. Each time the user must enter the PIN, the device generates a random position for the numbers and show them on the device screen. The user must look at the device and click on the dot positions shown on the user interface. Figure 17 shows this mechanism.



User interface on the computer

Hardware wallet

Figure 17: Entering the PIN on the device without revealing it to screen recorders.

Device personalization

Imagine an *Evil Maid* attack where someone steals your device and replaces it with a replica that can send data over the internet. Next time you start your device (which will be the replica) you will be asked to enter the PIN, which you will do, as you cannot differentiate the replica from your real device. Just after that, the replica will send your PIN though the internet to the attacker. Now he can use your PIN on the real device and steal your funds.

To prevent this scenario, we recommend that the hardware wallet can allow personalization, as show on Figure 18. If the device has a screen, a user can define a name or even an image for it, that will be show after the correct PIN is entered. If the user enters the PIN and the device doesn't show the correct name or image, the user can know that his original device is stolen and quickly use the recovery seed words to restore the wallet on another device and transfer the funds.

Figure 18: Device personalized with the name Richard and a Lion image.

Another possibility is to divide the PIN in two parts. The first part (2 digits) would be used to show two mapped words from the BIP-39 wordlist. The user enters the first part of the PIN and check the words, if they differ from the usual words, the device has been replaced. Note that the words are unrelated with the seed words.

Plausible deniability and wipe PIN

There is a famous hypothetical attack known as the *5 dollars wrench* attack, show in figure 19.

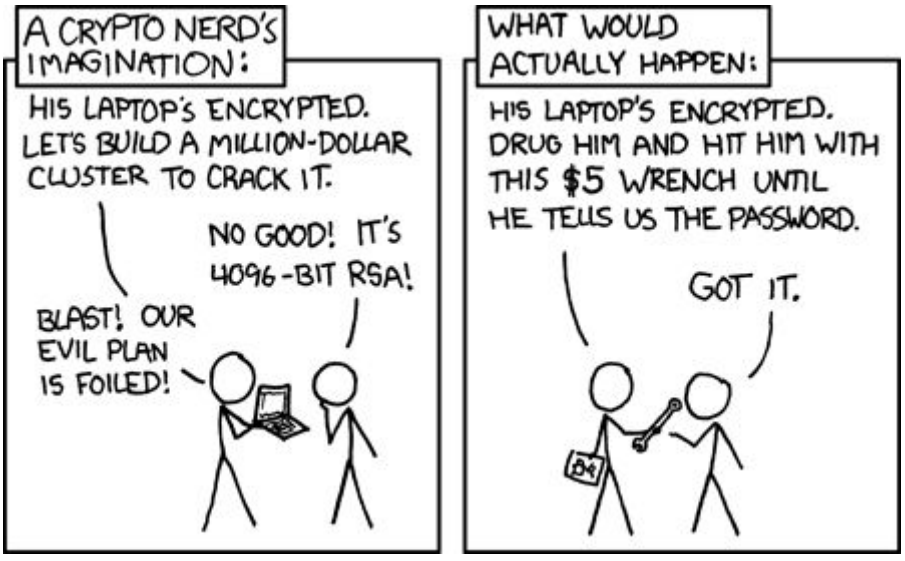


Figure 19: 5 dollars wrench attack.

To prevent against this kind of attack without using threshold cryptography, a user could have two – or more – Bitcoin wallets. One would be used to hold most of the user's funds and the other would have a smaller amount of coins. If someone asks for the password with a *5 dollars wrench* attack, the user could inform the password that reveals the second account.

This password would be an additional 25th word of the seed recovery phase. This word should be memorized by the user and would be asked every time the device is turned on. The idea is that every word entered would load a valid wallet, allowing the user to set up two or more accounts for his funds. This is also known as plausible deniability, which can be useful different scenarios, like a government seizure.

A related feature would be to allow a user to set up a *wipe* PIN, that when entered would reset the device memory to random bits, making it impossible to restore the wallet without the seed words.

Hardware level protection

To integrate a screen and optional buttons on the device we must include a microcontroller into the project. This requirement brings us out of the tamper proof world of smart cards and puts us in a condition where we must consider the hardware security aspects of the microcontroller and its integration with other components. It is out of the scope of this work to provide details on how to protect this additional hardware components from attacks, nevertheless we will quickly cite some basic possibilities (most of them will increase the cost of the project):

Disable debug interfaces: It is a good practice to disable the Joint Test Action Group (JTAG) or similar interface of the integrated microcontroller if it is included. This will help prevent unauthorized write access to the bootloader's memory area.

Active and passive intrusion detection: This mechanisms are found in some Hardware Security Modules (HSM), the idea is to continuous monitor the voltage, temperature, light, proximity, magnetic field and other aspects and acting on the detection of an anomaly, like clearing the memory associated with the private keys.

Protective shield on the integrated circuit: Active and passive shield mashes can be used to protect the probing of the microcontroller memory.

Hardware potting: An optional practice is to fill the device with a solid or gelatinous compound, improving the resistance to shock, vibration, moisture and corrosive agents. Usually epoxy is used.

Dust and water resistant: Although not related to protection against attacks, it is a interesting feature to provide at least IP67[2] (Ingress Protection) for a commercial hardware wallet, fully protecting against dust and immersion, up to 1m depth, for 30 seconds. This will increase the lifetime of the device, protecting the internal hardware components.

[2] The Ingress Protection Code (IEC standard 60529) classifies and rates the degree of protection provided against intrusion, dust, accidental contact, and water by mechanical casings and electrical enclosures. The standard aims to provide users more detailed information than vague marketing terms such as waterproof.

Attestation of genuineness

To prevent *Evil Maid* and mainly *Supply Chain* attacks, it is an important feature to be able to verify the authenticity of the device. One option is to generate a attestation key pair on the secure element on the manufacture process, recording the public key of the device on the database of the company making the device or even on the Bitcoin blockchain (our proposition). To check for the authenticity of the secure element, the user interface would send a challenge for the device to sign with the private attestation key, then it would check if the signature matches with the public key stored on the manufacturer database or the blockchain. If the device is not genuine, the user interface would warn the user about that.

Another aspect related to genuineness, is to check the integrity of the bootloader and the firmware of the microcontroller on devices that contains a screen and optional buttons. This is a big challenge and there seems to be no definitive solution. One option is to record the bootloader on a read only memory upon manufacturing and making it responsible for providing a writable memory space for the firmware, making the software updateable. The bootloader should communicate with the secure element and send a SHA-256 hash of a full memory read with the provided signature of the firmware. The secure element can then check if the signature matches with the manufacturer public key and the provided hash, and inform the bootloader about the result, that could be written to the device screen.

Prototype

As we've seen in the previous sections, a hardware wallet is a complex project, encompassing different software and hardware components. To go beyond the theoretical knowledge, we developed a functional prototype using a real smart card. The prototype didn't have a dedicated screen or buttons and didn't use a secure channel on its communication, although it checks for the attestation of the genuineness of the smart card.

Overview of the architecture

Figure 20 shows an overview of the general architecture of the project. There are four main pieces of software that should be written to support the full experience of the user with the wallet. These pieces will be detailed in the next section.

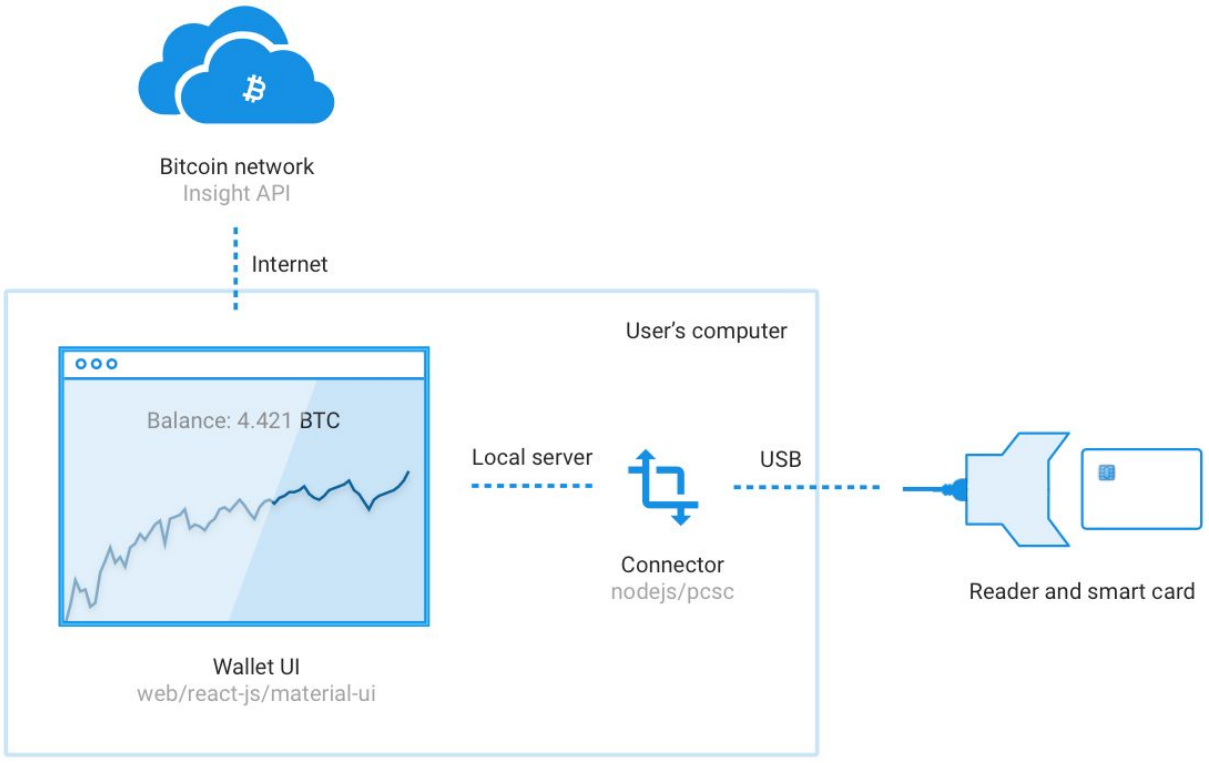


Figure 20: Overview of the architecture of the project.

The user flow would be as follows:

1. **Acquire a new device**: The manufacturer (the author of this work, in this case) will install the compiled Java Card Applet (*.cap* format) on a new smart card and lock the device to prevent the installation of additional Applets. Then

the manufacturer will access the smart card and ask for the public attestation key generated for the device upon the Applet first run (*install()* method). This key will be recorded in the Bitcoin blockchain (as previously proposed).

2. **Install the Connector software**: The user will navigate to the wallet website and be asked to install the *Connector* software.
3. **Access the wallet web interface**: After installing the *Connector*, the user will be able to access the smart card and proceed with the wallet setup. In this step the wallet user interface will check if the device is genuine.
4. **Setup or recovery wallet's accounts**: As the wallet has no account configured, the user will be asked to setup a new account (where he will asked to write down the new seed words) or recovery using previous generated backup seed words. Here he will also define a PIN to access the wallet.
5. **Sees the balance and history of transactions:** After entering the correct PIN, the user will be able to see all the transactions that happened in the account and the current balance. The interface will interact with the Bitcoin network and the smart card (asking for addresses), following the BIP-44 account discovery protocol [74].
6. **Generate new receiving addresses:** If the user wants to receive new funds from someone else, he can generate a fresh address.
7. **Transfer funds to other addresses:** The user can also send Bitcoins to other users, by defining the amount and destination address. The transaction will be generated and sent to the smart card to be signed after the correct PIN entering. The signed transaction can then be propagated to the Bitcoin network by the web interface.

To accommodate this flow we must define the detailed requirements for each of the four main components, which is done in the next section.

Main requirements

Hardware wallet

The main basic requirements of the wallet were defined in the start of this chapter. In addition to that, we will need to generate the attestation keys upon installation and to sign requests for attestation validation.

1. **SETUP**
   - 1.1. To define the PIN to access the device;
   - 1.2. To change the PIN; (PIN required)
   - 1.3. To generate the initial master private key of the device using a TRNG and the seed words for recovery (BIP-39)[3]; (PIN required)
   - 1.4. To recover a wallet by defining the master private key using the seed words; (PIN required)
   - 1.5. To verify that the seed words generates the master private key stored on the device; (PIN required)
   - 1.6. To generate the attestation of genuineness keys using a TRNG.
2. **ADDRESS GENERATION**
   - 2.1. To generate deterministic BIP-32 address (public/private keys). (PIN required)
3. **SIGN TRANSACTIONS**
   - 3.1. To sign Bitcoin transactions using the keys of a BIP-32 path; (PIN required)
4. **INTERACTION**
   - 4.1. To validate the user PIN;
   - 4.2. To take actions based on PIN policies, like to erase the device if the PIN is wrong for 5 consecutive times;
   - 4.3. To sign a challenge (SHA-256 hash) with the genuineness private key;
   - 4.4. To respond with the genuineness public key of the device;
   - 4.5. To erase the device if request by the user. (PIN required)

[3] The seed words should be sent to the user only once, when the seed is generated.

Connector

The user interface will be served as a web site on the user's browser. We must find a way to allow the browser to access the smart card. In order to communicate with an USB device (in our case the smart card reader), we have three options: use the *WebUSB API* [75], write an extension to *Chrome* or *Firefox* or use a custom piece of software, which is our choice.

We will develop a middleware called *Connector* that will make the bridge between the USB smart card reader and the browser, by making it run a local web server on the client computer. This software will have to be installed, but it offers a more generic solution that can later receive new features, like establishing a secure channel, simulate a smart card or to serve a desktop interface.

Table 3 shows the HTTP commands that should be supported by the *Connector* API. All requests uses HTTP *POST* and return JavaScript Object Notation (JSON*)* responses.

url method	parameters	result type	description
`/ping`		"knox"	Returns the string "PONG"
`/list`		Array<{`path`: string, `session`: string | null}>	Lists devices. If `session` is null, nobody else is using the device; if it is string, it identifies who is using it.
`/lock/PATH`	`PATH`: path of device	{`session`: string}	Locks the device at `PATH`.
`/unlock/SESSION`	`SESSION`: session to unlock	{}	Unlocks the device with the given session.
`/call/SESSION`	`SESSION`: session to call request body: the command	{response: string}	Sends a command to the device and returns the answer.

Table 3: API specification of the *Connector* component.

Wallet User Interface

As mentioned, this will be a web interface running on any modern browser. It will support the steps 2 to 7 from the architecture view section. It will interact with the Bitcoin network through the *Blockchain* API and with the device through the *Connector* API. Figure 21 shows the main actions that the interface should support in a workflow format.

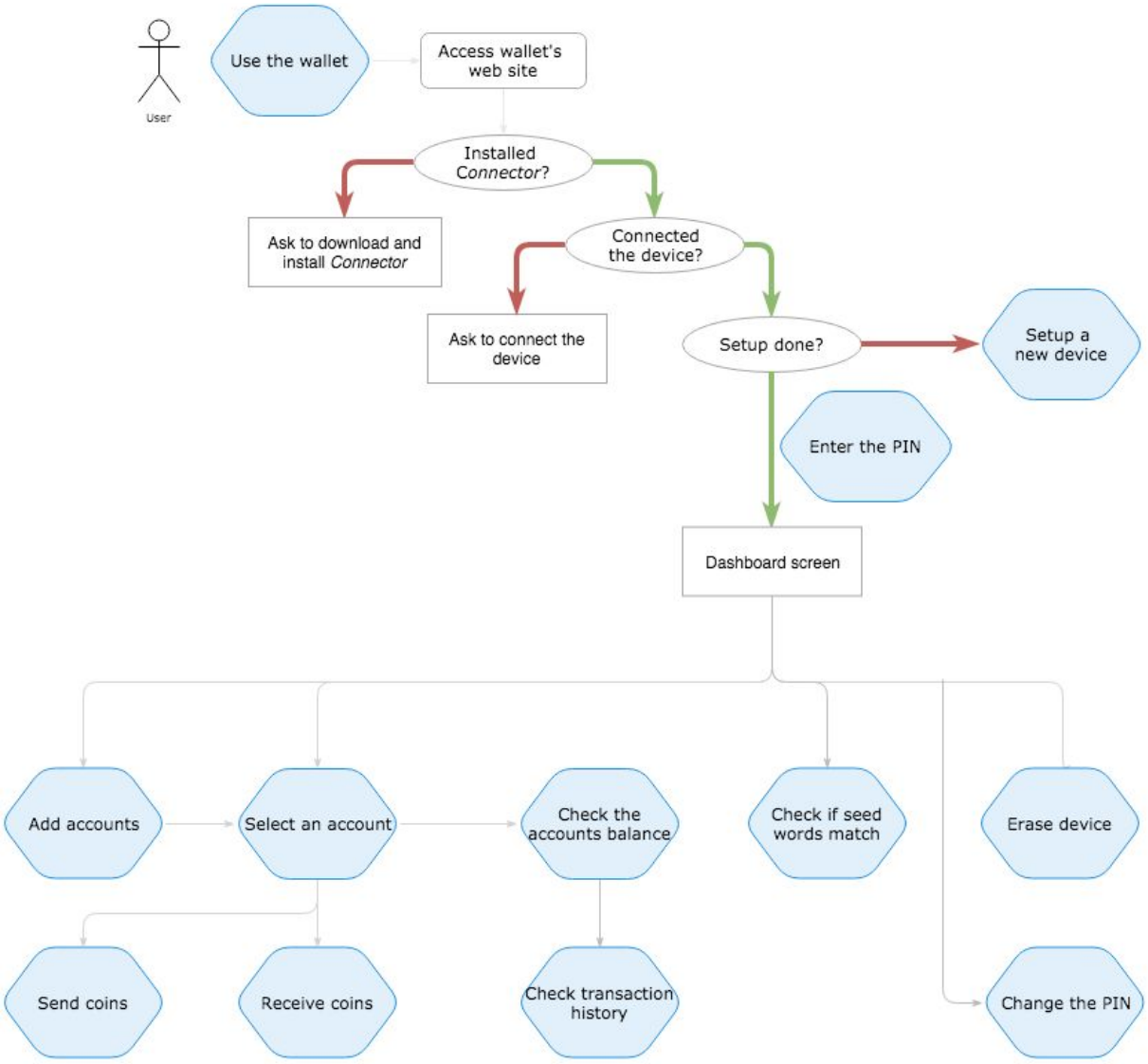


Figure 21: Workflow of navigation on the wallet user interface.

Figue 22 shows the workflow for setting up a new device.

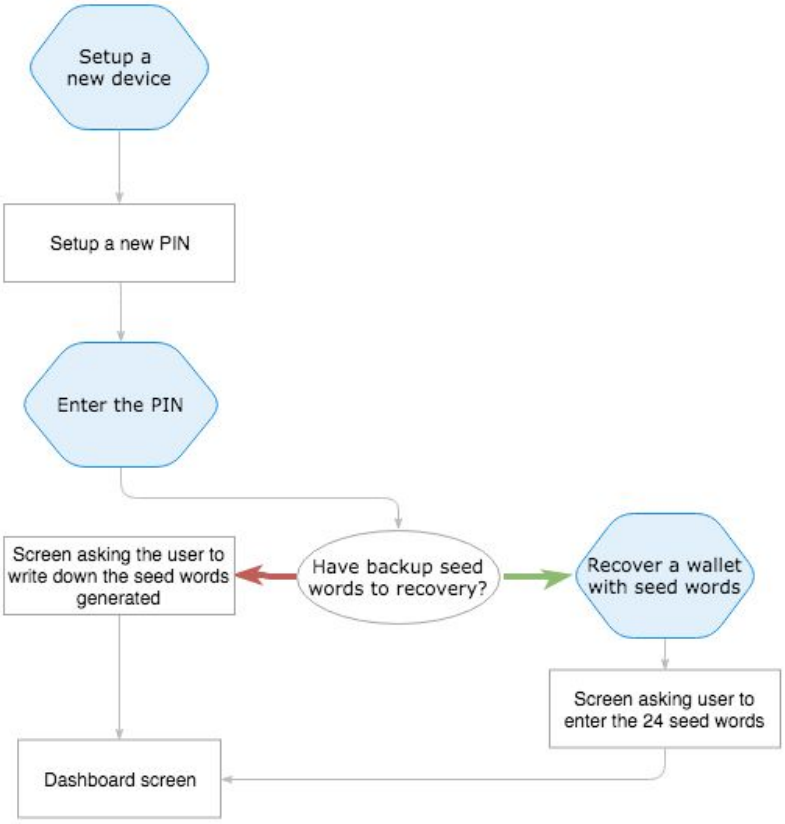


Figure 22: Workflow to setup a new device.

Blockchain API

The wallet web interface needs a way to communicate with the Bitcoin network. More than that, it needs a mechanism to quickly gather information about an address, like the transactions associated with it and the current balance. Given the actual Bitcoin's blockchain size of 173GB [76], this isn't an easy task.

The Blockchain API component should provide an Representational State Transfer (REST) HTTP interface with different endpoints that will be used by the web user interface. This Blockchain API should be served in fixed web domains and be maintained by the wallet manufacturer. In a real product there should be at least 2 completely independent deployments of this service (in different world regions) to provide additional redundancy.

We will use a widely adopted open source solution called *Insight-API*, maintained by *BitPay* [77]. Table 4 shows some of the endpoints available and Figure 23 an API response in JSON format.

url method	description
`/insight-api/txs/?address=ADDR`	Returns all the transactions related to an address
`/insight-api/addr/[:addr]`	Returns the details about an address, like the total balance.
`/insight-api/tx/[:txid]`	Returns the details about a transaction.
`/insight-api/utils/estimatefee[?nbBlocks=2]`	Returns the estimated fee required to make the transaction be included in the next *nbBlocks*.
`/insight-api/tx/send`	Broadcasts a transaction to the Bitcoin network.

Table 4: Some endpoints available on the Blockchain (Insight) API.


```
{
  "addrStr": "1FhNPRh1TxVidoKkWFEpdmK5RXw9vG1KUb",
  "balance": 5.0,
  "balanceSat": 5000000000,
  "totalReceived": 55.86,
  "totalReceivedSat": 5586000000,
  "totalSent": 50.86,
  "totalSentSat": 5586000000,
  "unconfirmedBalance": 0,
  "unconfirmedBalanceSat": 0,
  "unconfirmedTxApperances": 0,
  "txApperances": 4
}
```


Figure 23: JSON API response of an address endpoint.

Results

Hardware wallet

As the secure element we selected the NXP J3D081 EV1 smart card. It is a contact and contactless interface Java Card 3.0.1 running on Global Platform 2.2.1 with 80K of EEPROM. Table 5 shows the full specification.

Feature	Details
EEMemory	80KB
Javacard version	3.0.1
CC certification CC EAL	6+
Gobal Platform version	2.2.1
Delegate Management	Yes

DES/3DES (bit)	56/112/168
AES (bit)	256
RSA (bit)	2048
SHA (bit)	512
MD5	Yes
ECC	Yes
SSCD Type3 CC EAL	Yes
Flex with Plus (incl.Classic) and Desfire EV1	Yes

Table 5: Full specification of the J3D081 Java Card [78].

To load the Applet into the smart card we used the *GlobalPlatform Pro* [79]. In addition to that, we used a library called *jCardSim* [80] to simulate a smart card, making the development phase faster, as the simulator can be integrated in the unit tests. Our implementation was based on previous work by *Ledger* [58], *SatoChipApplet* [81], *Status Wallet* [83], *IsoApplet* [82].

We used unit tests to guarantee that the Applet is behaving according to the specification in the simulator and on the real card. The tests also helped us to map the command flow between the *Connector* and the smart card. Figure 24 shows the result of the tests running:

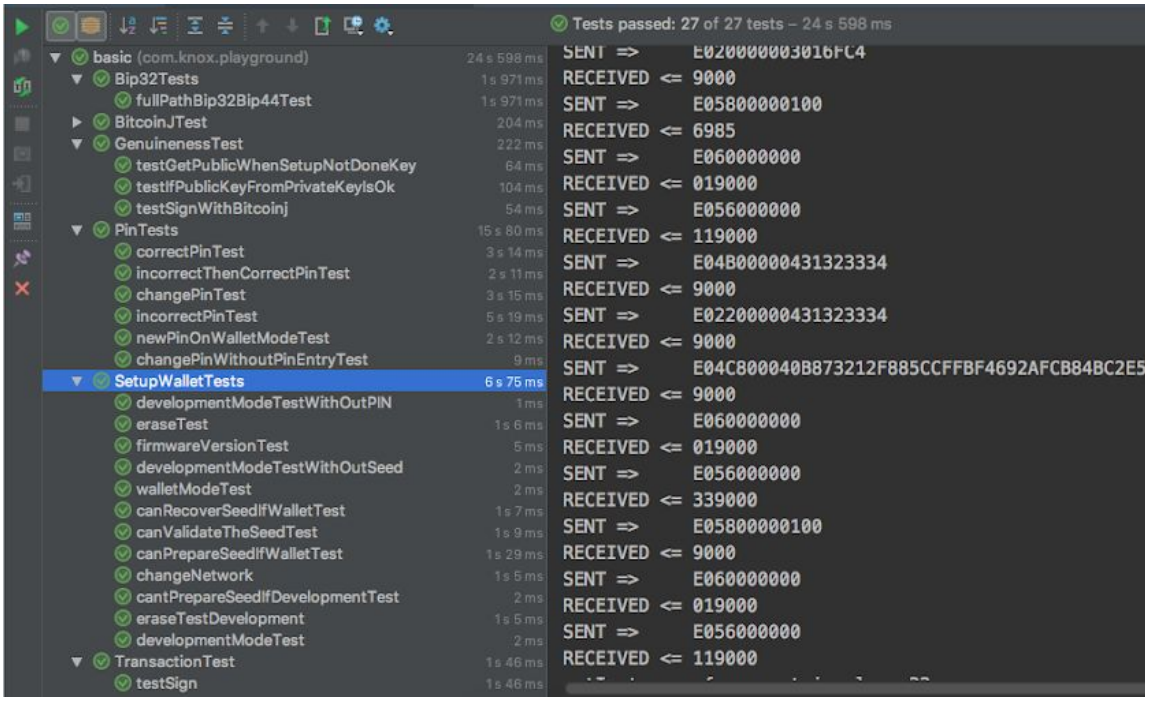


Figure 24: Unit tests running on the Applet.

At the manufacturer the Applet is *installed* and the *setup* method is called to put the wallet into *normal* (or *development*, used for tests) mode. Then the wallet goes to the final user that will plug it into the smart card reader and start to use, *defining the PIN* and *generating the seed*, where he will receive the recovery seed words (just once). After that the wallet is ready to be used. If it is erased or the PIN is entered incorrectly more than 5 times, the wallet clears the memory and returns to the setup state. Figure 25 shows this flow.

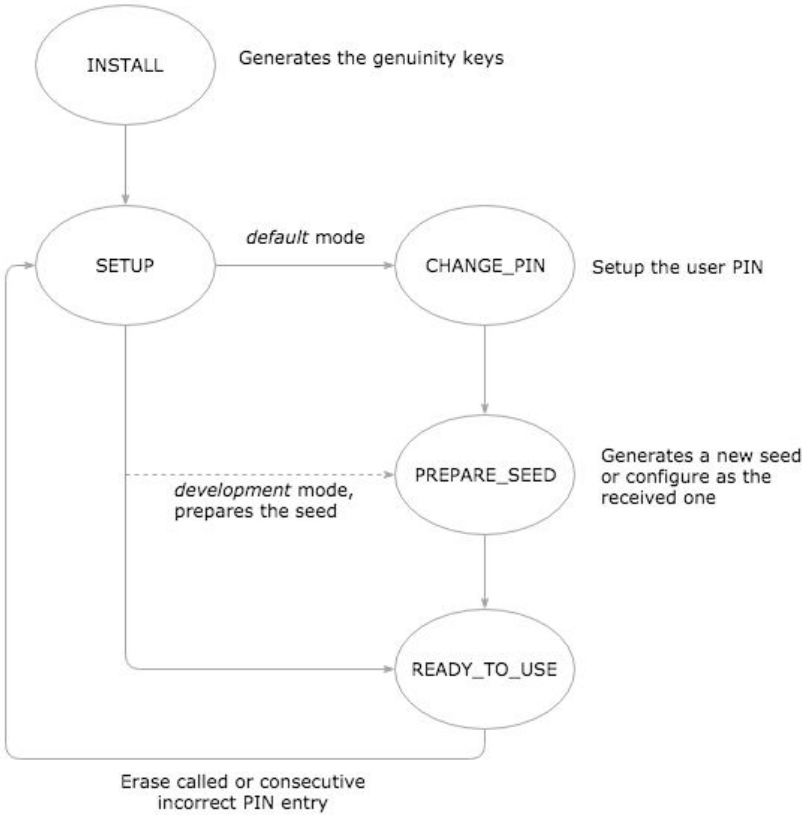


Figure 25: Possible states of the Applet.

APDU commands and responses

It was necessary to define and code the full list of commands and responses that the Applet should support. We followed the ISO 7816 as much as we could. The specification is shown in this subsection.

INSTALL

This is the first command to be called (only once). It will initialize the Applet instance and generate and the genuineness key.

CLA	INS	P1	P2	Lc	Data	Le
80	E6	00	00	00	--	00

SETUP

This is the second command to be called, it will setup the wallet. If the mode is development, the BIP-39 seed and PIN must be provided (this mode is used for unit testing and debugging).

CLA	INS	P1	P2	Lc	Data	Le

E0	22	00	00	var	Mode: 0x01 normal, 0x08 development bitcoinNetworkKeyVersion: 1 byte bitcoinNetworkKeyVersionP2SH: 1 byte PIN length: from 0x04 up to 0x20 PIN: var BIP-39 Seed: 64 bytes	00

Response data	**SW 1, 2**
-	Setup already done or mode is development and seed/PIN is not provided: SW_CONDITIONS_NOT_SATISFIED Mode, seed or PIN invalid: SW_DATA_INVALID

CHANGE_PIN

This is called to set or change the the wallet PIN. The verify method must be called again after change.

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**
E0	4B	00	00	var	PIN length: from 0x04 up to 0x20 PIN: var	00

Response data	**SW 1, 2**
-	Setup not done or old PIN is not entered correctly: SW_CONDITIONS_NOT_SATISFIED New PIN not provided: SW_DATA_INVALID

PREPARE_SEED

This is called to generate a new random BIP-39 seed recovery words, that will be used by the client to generate the BIP-39 seed that should be set on this command. If the client already has the backup seed words, they can be set here, skipping the first step. It can only be called once with the final BIP-39 seed. When generating the words, the data returned is actually a sequence of 16-bit big-endian integers with values ranging from 0 to 2047.

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**
E0	4C	00: generate random words 80: set the wallet seed	00	var	If P1 is 80: BIP-39 seed: 64 bytes	var

Response data	**SW 1, 2**

The seed words indexes (If P1 is 00): var	Seed already set or PIN not set/checked: SW_CONDITIONS_NOT_SATISFIED Seed invalid: SW_DATA_INVALID

VALIDATE_SEED_BACKUP

This is called to check if the user recovery seed words match the seed generated on the device.

CLA	INS	P1	P2	Lc	Data	Le
E0	52	00	00	var	BIP-39 Seed: 64 bytes	var

Response data	SW 1, 2
-	Seed not set or PIN not set/checked: SW_CONDITIONS_NOT_SATISFIED Seed does not match: SW_DATA_INVALID

VERIFY_PIN

This command is used to unlock the device using the PIN or to check the remaining attempts. After 5 invalid PIN submitted in a row, the device data is erased.

CLA	INS	P1	P2	Lc	Data	Le
E0	22	00 : verify PIN 80 : get remaining attempts	00	var	PIN	00

Response data	SW 1, 2
-	Pin incorrect or setup not done: SW_CONDITIONS_NOT_SATISFIED Pin length invalid: SW_WRONG_LENGTH Remaining attempts: 1 byte

GET_GENUINENESS_KEY

This is called to get the genuineness public key.

CLA	INS	P1	P2	Lc	Data	Le
E0	4D	00	00	00	-	var

Response data	SW 1, 2
Uncompressed genuineness public key (EC): 65 bytes	-

PROVE_GENUINENESS

This is called to check the genuineness of the device, by asking it to sign a challenge.

CLA	INS	P1	P2	Lc	Data	Le
E0	4F	00	00	20	Challenge: 32 bytes random challenge to be signed	var

Response data	SW 1, 2
The signed challenge: var	Challenge length invalid: SW_WRONG_LENGTH

GET_FIRMWARE_VERSION

This will return the version of the Applet running on the device.

CLA	INS	P1	P2	Lc	Data	Le
E0	C4	00	00	00	-	03

Response data	SW 1, 2
Firmware major version: 1 byte Firmware minor version: 1 byte Firmware patch version: 1 byte	-

GET_STATE

This will return the current state of the device (reference on Figure 24).

CLA	INS	P1	P2	Lc	Data	Le
E0	56	00	00	00	-	01

Response data	SW 1, 2
Device state: 1 byte STATE_INSTALLED = 0x00 STATE_SETUP_DONE = 0x11 STATE_PIN_SET = 0x22 STATE_READY = 0x33	-

CHANGE_NETWORK

This is called to change the Bitcoin network parameters.

CLA	INS	P1	P2	Lc	Data	Le

E0	50	00	00	02	bitcoinNetworkKeyVersion: 1 byte bitcoinNetworkKeyVersionP2SH: 1 byte	00

Response data	**SW 1, 2**
-	-

ERASE

This will erase the device memory.

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**
E0	58	00	00	00	-	00

Response data	**SW 1, 2**
-	PIN not set/checked: SW_CONDITIONS_NOT_SATISFIED

GET_PUBLIC_KEY

This is called to get the EC public key of a BIP-32 path.

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**
E0	40	00	00	var	Number of BIP-32 derivations (max 10): 1 byte First derivation index (big endian): 4 bytes ... Last derivation index (big endian): 4 bytes	var

Response data	**SW 1, 2**
Public Key length: 1 byte Uncompressed public key (EC): var Base58 bitcoin address length: 1 byte Base58 encoded bitcoin address: var BIP-32 Chain code: 32 bytes	PIN not set/checked: SW_CONDITIONS_NOT_SATISFIED Path params invalid: SW_DATA_INVALID

SIGN_TRANSACTION

This is called to sign a bitcoin transaction using the public key derived from a BIP-32 path. It will actually sign the 32 bytes SHA256 double hash for the transaction, that should be prepared outside.

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**
E0	54	00 : prepare message 80 : sign message	00	var	Number of BIP-32 derivations (max 10): 1 byte	var

					First derivation index (big endian): 4 bytes ... Last derivation index (big endian): 4 bytes Hash: 32 bytes hash to be signed	

Response data	**SW 1, 2**
The signed hash: var	Path params invalid: SW_DATA_INVALID Hash length invalid: SW_WRONG_LENGTH

Connector

This component starts a local HTTP server on port 28281 and listens to calls made from whitelisted domains or localhost. Java 1.8 was used for listening and responding to HTTP requests. The interaction with the smartcard was done using *pcsc* and direct access to the simulator.

The Connector is a simple proxy, it reads commands sent from the web user interface, send to the smart card and send back the response. We designed an abstract *Transport* class and an concrete implementation called *TransportUsb* that actually interacts with the smart card reader. Figure 26 shows the server running and intercepting commands on the debug mode.

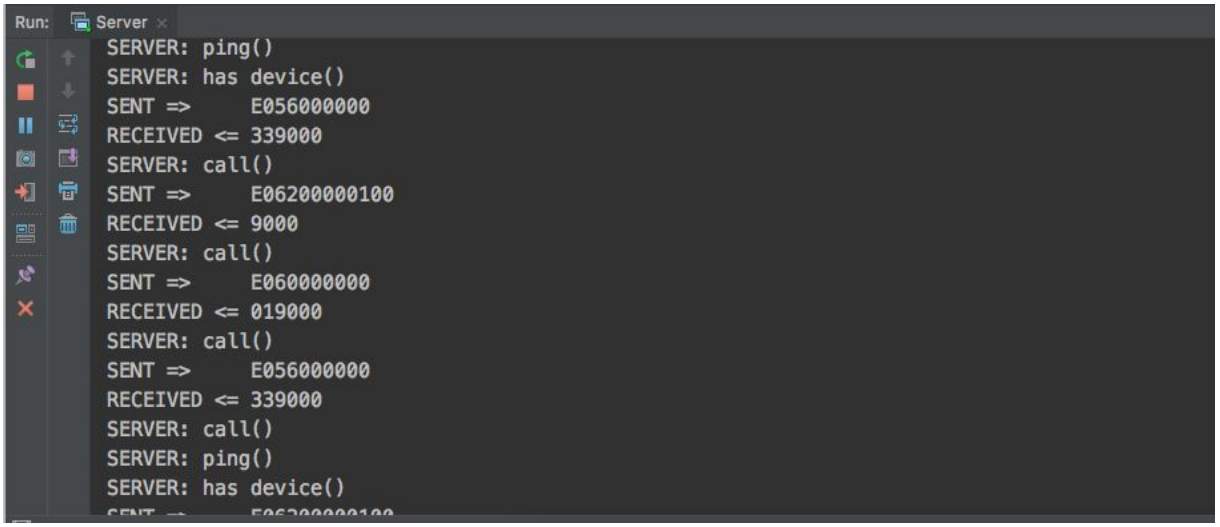


Figure 26: *Connector* running on localhost.

A class called *SecureDevice* was coded to abstract the interaction with the device, offering high level methods that can be used with different transports, like the

TransportUsb or the *TransportHTTP* (which is used by the web user interface to communicate with the *Connector*). This class can be used on the *Nodejs* context or on the browser. The methods supported are shown on Figure 27.

Figure 27: *Methods supported by the SecureDevice class.*

On a real product context, we would package the *Connector* into an software installer that would install and run the program as a background task upon the initialization of the operational system.

Wallet User Interface

The interface is a Single Page Application (SPA) developed using a modern set of web libraries:

- *React* to define the components and its behaviours;
- *MobX* for state management of the application;
- *Material-ui* for the Material Design components;
- *Webpack* for bundling the assets;
- *Yarn* for dependence management;
- *Jest* for running the unit and integration tests.

Before implementing the components, mockups of the interface were drawn using the workflow of the requirements section as a reference. Figures 28 to 32 shows some of the screens. Note on Figure 29 that we try to educate the user with important information regarding the seed words. Knox is the fictional name of the wallet.

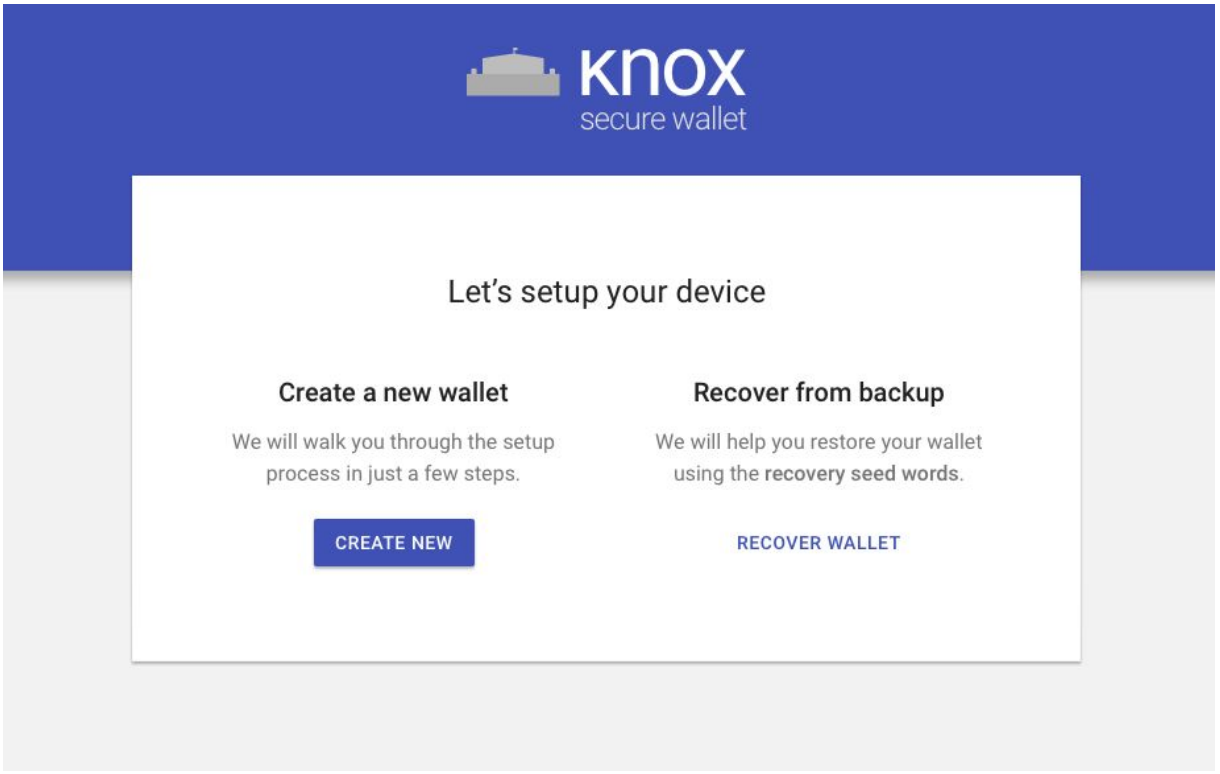


Figure 28: Setup screen giving two options for the user.

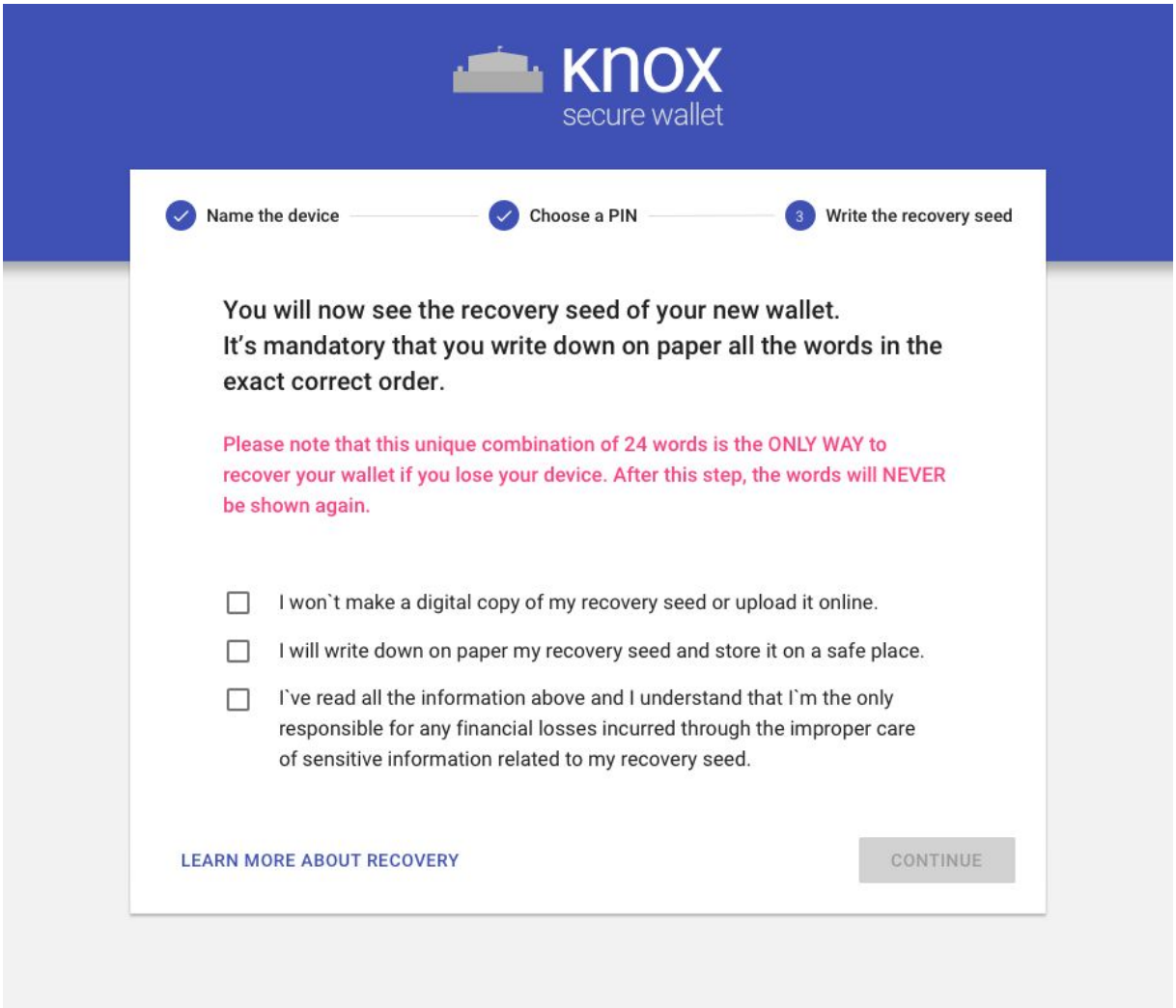


Figure 29: Educating the user about the importance of the recovery seed words.

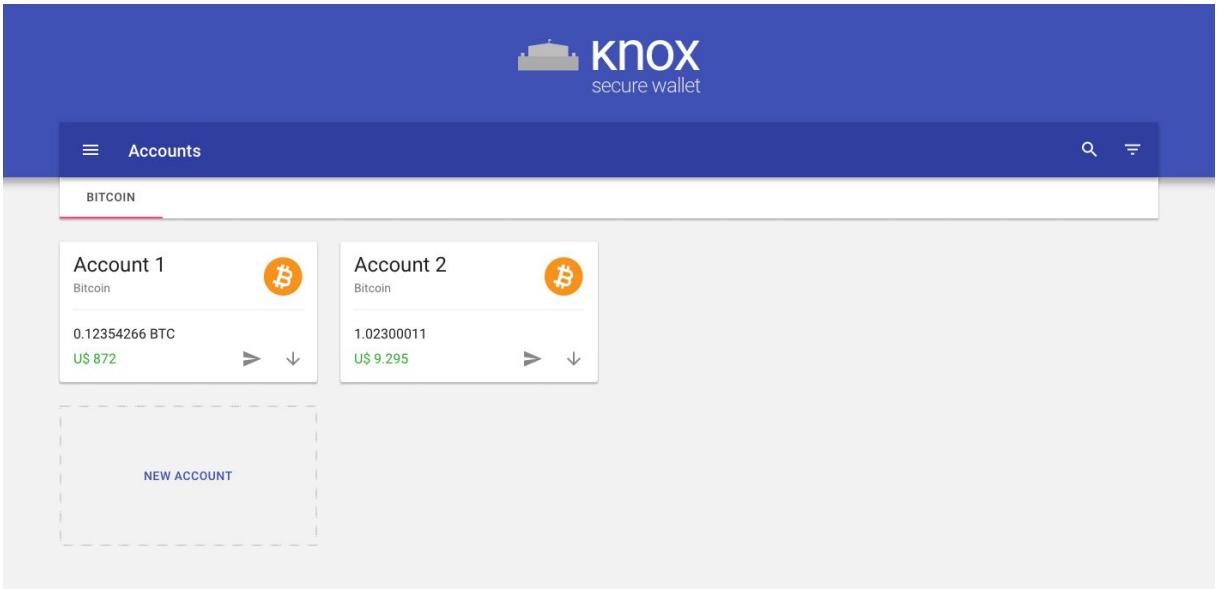


Figure 30: Dashboard with two Bitcoin accounts.

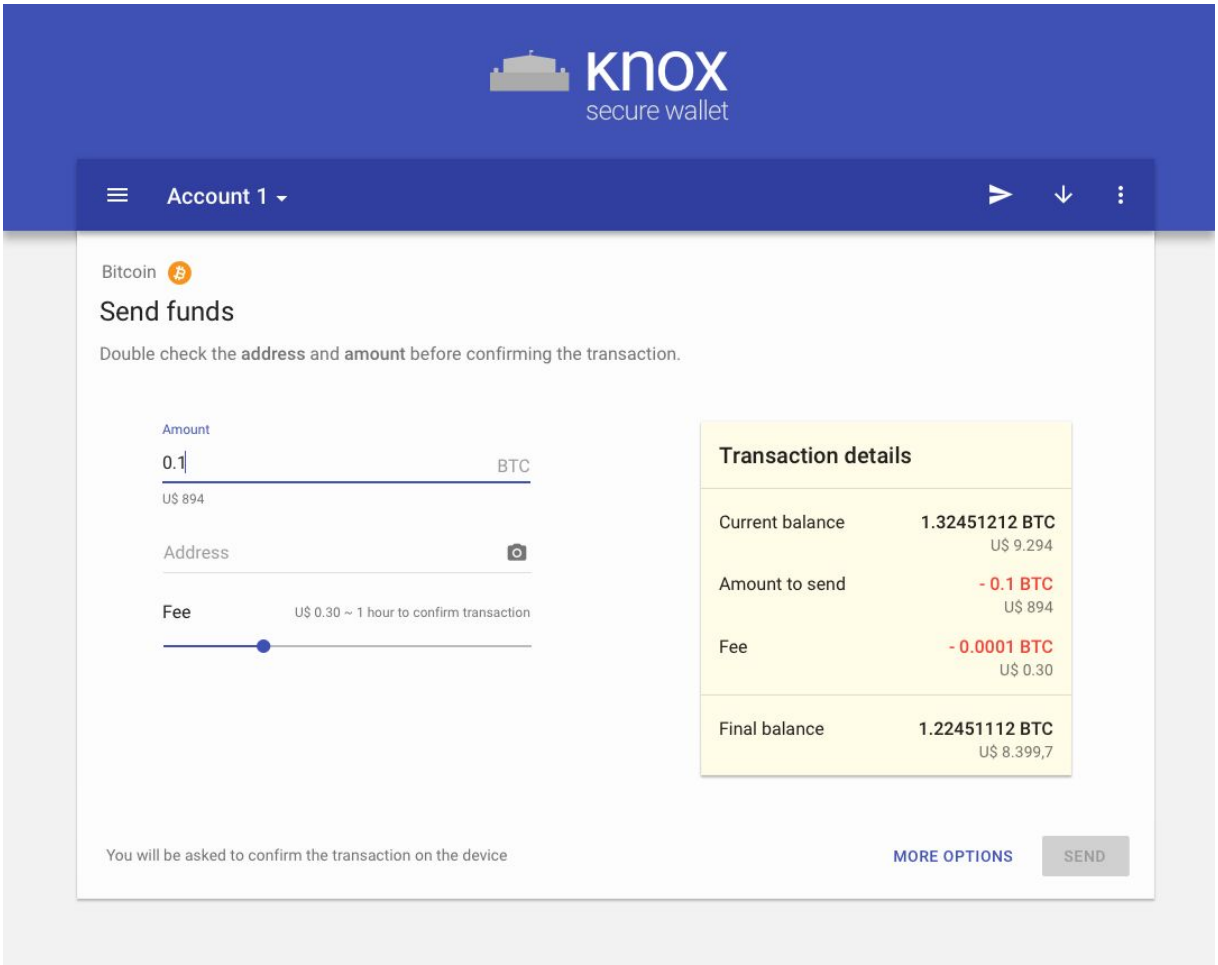


Figure 31: Send funds to an Bitcoin address.

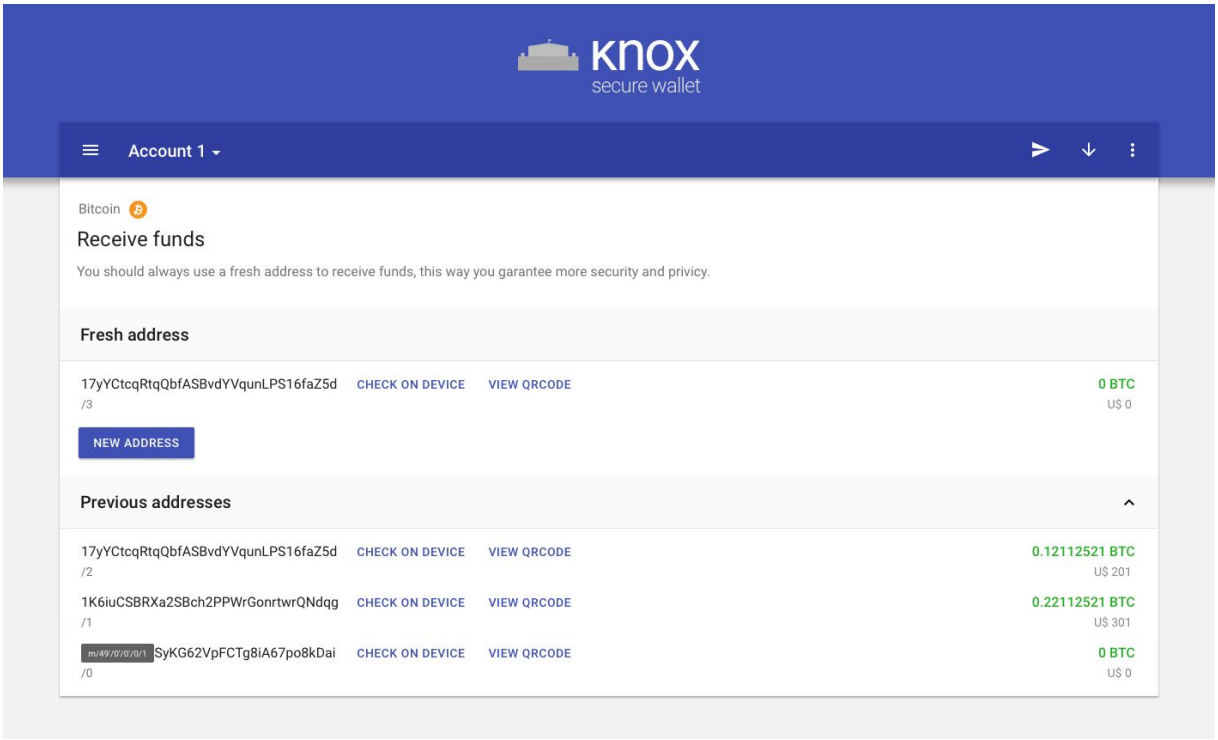


Figure 32: Generating fresh addresses to receive funds.

The user interface is easily extensible to deal with more robusts wallet versions, like the one with its own screen and buttons. It could also support more than one cryptocurrency, specially the ones compatible with the Bitcoin protocol.

Blockchain API

As previously mentioned, we used the *Insight-API* [77] as the Blockchain API component. Besides the API, we installed the *Insight blockchain explorer*, show on Figure 33. On the first run of the Insight-API, it will connect to Bitcoin nodes and start to download and sync the blockchain data, which can take a few hours to complete. After that, the url endpoints are fully accessible. To download and start the component, one should run the commands bellow:

```
npm install -g bitcore@latest
bitcore create mynode
cd mynode
bitcore install insight-api
bitcore install insight-ui
bitcore start
```

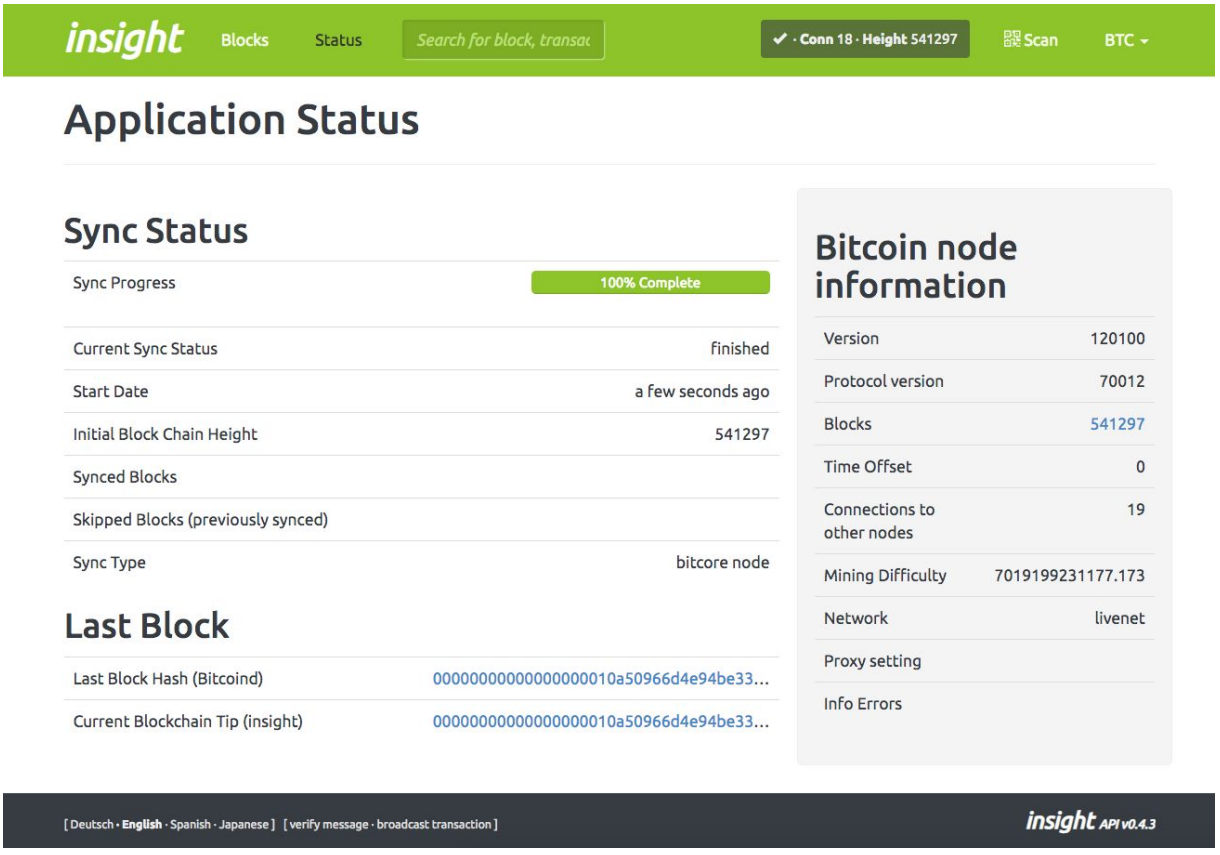


Figure 33: *Insight* blockchain explorer fully synced.

Source code

The source code of this project is publicly available at GitHub on the following repository: https://github.com/ricardovf/knox-wallet. Be advised that this project is only a prototype and isn't ready to be used in a real product.

Our Java Card Applet was based on open source code. We register our gratitude for the authors of the following projects:

1. Ledger Java Card Applet [58]
2. Status Wallet [83]
3. SatoChipApplet [81]
4. IsoApplet [82]

CONCLUSION

In this academic work we explored the exciting and novel world of Bitcoin, focusing on the problem of storing the user's private keys that give access to the funds, while providing the best balance between very high security, ease of use and independence of third parties, fulfilling Satoshi Nakamoto's vision of a decentralized peer-to-peer electronic cash system.

After making a review on cryptography used on Bitcoin, the Bitcoin protocol and secure elements, we dived into the project of hardware wallets, discussing different requirements and ways to construct a device. The device uses an anti tamper Java Card to store the private keys. We considered variations of the device, one with a dedicated touchscreen and another with NFC to integrate with a mobile phone. We analyzed security aspects of the project, made recommendations and described some challenges. Finally, we implemented our own open source prototype, showing the architecture of the project, its components, the requirements, the APDU communication protocol and the results.

The prototype constructed is limited in contrast to the more complete versions discussed, with a dedicated screen and an encrypted communication channel, but it can be further developed to accommodate the full requirements and be deployed into the world as a complete product.

Hardware wallets projects are complex, involving integrated circuits and software that must communicate and integrate securely, considering different attack vectors, without letting out the importance of the final cost and usability. Nevertheless, we must constantly encourage the enhancement of the current solutions and develop novel approaches, making cryptocurrencies a practical everyday solution to contrast with the traditional financial system.

Future work

As we concluded, the project of a real, secure and fully functional hardware wallet is very complex, embracing topics in different areas of computer science. We

believe there is a rich set of possibilities for future work related to this project, most of them on the practical side of extending and validating the security of the wallet:

1. Build prototype of the wallet with an integrated touchscreen;
2. Build a prototype using a contactless smart card (NFC) and a mobile phone as the device screen;
3. Systematically attack the wallet to find vulnerabilities and propose corrections;
4. Run usability test with real users to improve the wallet friendliness;
5. Implement the support for multiple cryptocurrencies, including ones not directly derived from Bitcoin Core;
6. Implement the support for multisignature scheme;
7. Find solutions to securely guarantee the genuineness of the integrated microcontroller;
8. To write an updated book on Java Card development utilizing open source libraries and tools, as the materials on the topic are very fragmented and diffuse.

REFERENCES

[1] NAKAMOTO, Satoshi. **Bitcoin: A peer-to-peer electronic cash system**. 2008.

[2] NEURON. **List of cryptocurrency exchange hacks**. Available at: <https://rados.io/list-of-documented-exchange-hacks/>. Accessed on may 29, 2018.

[3] COINMARKETCAP. **Global Charts**. Available at: <https://coinmarketcap.com/charts/>. Accessed on sep 14, 2018.

[4] BARIVIERA, A.F., BASGALL, M.J., HASPERUÉ, W. & NAIOUF, M. 2017. **Some stylized facts of the Bitcoin market**. Physica A: Statistical Mechanics and its Applications 484:82–90.

[5] GENTILAL, M., MARTINS, P. & SOUSA, L. 2017. **TrustZone-backed bitcoin wallet**. Proceedings of the Fourth Workshop on Cryptography and Security in Computing Systems - CS2 '17.

[6] Monopoly without a Monopolist: An Economic Analysis of the Bitcoin Payment System

[7] CONTI, M., E, S.K., LAL, C. & RUJ, S. 2018. **A Survey on Security and Privacy Issues of Bitcoin**. IEEE Communications Surveys & Tutorials :1–1.

[8] ESKANDARI, S., BARRERA, D., STOBERT, E. & CLARK, J. 2015. **A First Look at the Usability of Bitcoin Key Management**. Proceedings 2015 Workshop on Usable Security.

[9] BAMERT, T., DECKER, C., WATTENHOFER, R. & WELTEN, S. 2014. **BlueWallet: The Secure Bitcoin Wallet**. Lecture Notes in Computer Science :65–80.

[10] VAN DER HORST, L., CHOO, K.-K.R. & LE-KHAC, N.-A. 2017. **Process Memory Investigation of the Bitcoin Clients Electrum and Bitcoin Core**. IEEE Access 5:22385–22398.

[11] COURTOIS, Nicolas T.; VALSORDA, Filippo; EMIRDAG, Pinar. **Private Key Recovery Combination Attacks: On Extreme Fragility of Popular Bitcoin Key Management, Wallet and Cold Storage Solutions in Presence of Poor RNG Events**. 2014.

[12] GKANIATSOU, A., ARAPINIS, M. & KIAYIAS, A. 2017. **Low-Level Attacks in Bitcoin Wallets**. Information Security :233–253.

[13] YOUNG, Joseph. **Exponential Growth: Cryptocurrency Exchanges Are Adding 100,000+ Users Per Day**. Available at: <https://cointelegraph.com/news/exponential-growth-cryptocurrency-exchanges-are-adding-100000-users-per-day>. Accessed on may 29, 2018.

[14] MAY, Timothy. **The Crypto Anarchist Manifesto**. Available at: <https://www.activism.net/cypherpunk/crypto-anarchy.html>. Accessed on may 29, 2018.

[15] WIKIPEDIA. **Crypto-anarchism**. Available at: <https://en.wikipedia.org/wiki/Crypto-anarchism>. Accessed on may 29, 2018.

[16] BITCOIN. **Securing your wallet**. Available at: <https://bitcoin.org/en/secure-your-wallet>. Accessed on may 29, 2018.

[17] SEETHARAMAN, A., SARAVANAN, A.S., PATWA, N. & MEHTA, J. 2017. **Impact of Bitcoin as a World Currency**. Accounting and Finance Research 6:230.

[18] BARBU, Guillaume. **On the security of Java Card platforms against hardware attacks**. 2012. Tese de Doutorado. Telecom ParisTech.

[19] HANSMANN, Uwe et al. **Smart card application development using Java**. Springer Science & Business Media, 2012.

[20] ATKINS, Steve. **JCF celebrates 15 year anniversary in Singapore – 10 billion Java Cards deployed worldwide**. Available at: <https://contactlessintelligence.com/2012/09/21/jcf-celebrates-15-year-anniversary-in-singapore-10-billion-java-cards-deployed-worldwide/>. Accessed on may 29, 2018.

[21] GKANIATSOU, Andriana et al. Getting to know your card: reverse-engineering the smart-card application protocol data unit. In: **Proceedings of the 31st Annual Computer Security Applications Conference**. ACM, 2015. p. 441-450.

[22] BOZZATO, Claudio et al. **APDU-level attacks in PKCS# 11 devices**. In: International Symposium on Research in Attacks, Intrusions, and Defenses. Springer, Cham, 2016. p. 97-117.

[23] DE KONING GANS, Gerhard; DE RUITER, Joeri. **The smartlogic tool: Analysing and testing smart card protocols**. In: Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on. IEEE, 2012. p. 864-871.

[24] HOGENBOOM, Jip; MOSTOWSKI, Wojtek. **Full memory attack on a Java Card**. 2009.

[25] GOLDFEDER, Steven et al. **Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme**. 2015.

[26] KÖMMERLING, Oliver; KUHN, Markus G. **Design Principles for Tamper-Resistant Smartcard Processors**. Smartcard, v. 99, p. 9-20, 1999.

[27] HÖLZL, Michael et al. A password-authenticated secure channel for App to Java Card applet communication. **International Journal of Pervasive Computing and Communications**, v. 11, n. 4, p. 374-397, 2015.

[28] GITHUB. **Bitcoin Improvement Proposals**. Available at: <https://github.com/bitcoin/bips> Bitcoin Improvement Proposals. Accessed on oct 10, 2018.

[29] WIKIPEDIA. **SHA-2**. Available at: <https://en.wikipedia.org/wiki/SHA-2>. Accessed on jun 5, 2018.

[30] WILLIAM, Stallings. **Cryptography and network security: principles and practice**. Prentice-Hall, Inc, p. 23-50, 1999.

[31] PAAR, Christof; PELZL, Jan. **Understanding cryptography: a textbook for students and practitioners**. Springer Science & Business Media, 2009.

[32] STACKEXCHANGE. **Why does Bitcoin use two hash functions (SHA-256 and RIPEMD-160) to create an address?**. Available at: <https://bitcoin.stackexchange.com/questions/9202/why-does-bitcoin-use-two-hash-functions-sha-256-and-ripemd-160-to-create-an-ad>. Accessed on jun 5, 2018.

[33] WIKIPEDIA. **Hash function security summary**. Available at: <https://en.wikipedia.org/wiki/Hash_function_security_summary>. Accessed on jun 6, 2018

[34] GITHUB. **BIP32**. Available at: <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>. Accessed on jun 6, 2018

[35] GITHUB. **BIP39**. Available at: <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>. Accessed on jun 6, 2018

[36] KALISKI, Burt. **PKCS# 5: Password-based cryptography specification version 2.0**. 2000.

[37] CERTICOM, Standards for Efficient Cryptography. **SEC 2: Recommended Elliptic Curve Domain Parameters**. Version 2.0. 2010.

[38] BITCOIN. **Secp256k1**. Available at: <https://en.bitcoin.it/wiki/Secp256k1>. Accessed on jun 11, 2018.

[39] MISTRY, M.. **An Introduction to Bitcoin, Elliptic Curves and the Mathematics of ECDSA**. 2015

[40] WIKIPEDIA. **Elliptic Curve Digital Signature Algorithm**. Available at: <https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm>. Accessed on jun 11, 2018.

[41] ANTONOPOULOS, Andreas M. **Mastering Bitcoin: unlocking digital cryptocurrencies**. O'Reilly Media, Inc.. 2014.

[42] WANG, Di. **Secure Implementation of ECDSA Signature in Bitcoin**. Information Security at University College of London. 2014.

[43] ANTONOPOULOS, Andreas M. **Mastering Bitcoin: Programming the open blockchain**. O'Reilly Media, Inc. 2017.

[44] BITCOIN. **Transaction**. Available at: <https://en.bitcoin.it/wiki/Transaction>

[45] BITCOIN. **Script**. Available at: <https://en.bitcoin.it/wiki/Script>

[46] BLOCKCYPHER. Bitcoin Address. Available at: <https://live.blockcypher.com/btc/address/112EGg2MRchxD1E2ELoMGUgFzpycxM2gSU/>. Accessed on oct 10, 2018.

[47] SHIRRIFF, Ken. **Bitcoins the hard way: Using the raw Bitcoin protocol**. Available at: <http://www.righto.com/2014/02/bitcoins-hard-way-using-raw-bitcoin.html>. Accessed on oct 10, 2018.

[48] BITCOIN. **Address**. Available at: <https://en.bitcoin.it/wiki/Address>. Accessed on oct 10, 2018.

[49] BITCOIN. **Technical background of version 1 Bitcoin addresses**. Available at: <https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses>. Accessed on oct 10, 2018.

[50] GITHUB. **BIP173**. Available at: <https://github.com/bitcoin/bips/blob/master/bip-0173.mediawiki>. Accessed on oct 10, 2018.

[51] BITCOIN. **Address reuse**. Available at: <https://en.bitcoin.it/wiki/Address_reuse>. Accessed on oct 10, 2018.

[52] BITCOIN. **Seed phrase**. Available at: <https://en.bitcoin.it/wiki/Seed_phrase>. Accessed on oct 10, 2018.

[53] BITCOIN. **Cold storage**. Available at: <https://en.bitcoin.it/wiki/Cold_storage>. Accessed on oct 10, 2018.

[54] BITCOIN. **Hardware wallet**. Available at: <https://en.bitcoin.it/wiki/Hardware_wallet>. Accessed on oct 10, 2018.
[55] BITFI. **Bitfi cryptocurrency wallet**. Available at: <https://bitfi.com>. Accessed on oct 10, 2018.
[56] WIKIPEDIA. **Diceware**. Available at: <https://en.wikipedia.org/wiki/Diceware>. Accessed on oct 10, 2018.
[57] BITCOIN. **Brainwallet**. Available at: <https://en.bitcoin.it/wiki/Brainwallet>. Accessed on oct 10, 2018.
[58] LEDGER. **Ledger Nano S**. Available at: <https://www.ledger.com/products/ledger-nano-s>. Accessed on oct 10, 2018.
[59] HAO, Feng; RYAN, Peter. **J-PAKE: authenticated key exchange without PKI**. In: Transactions on computational science XI. Springer, Berlin, Heidelberg, 2010. p. 192-206.
[60] WIKIPEDIA. **Secure Remote Password_protocol**. Available at: <https://en.wikipedia.org/wiki/Secure_Remote_Password_protocol>
[61] GLOBALPLATFORM. **Introduction to Secure Elements**. Available at: <https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Secure-Element-15May2018.pdf>. Accessed on oct 10, 2018.
[62] WIKIPEDIA. **Tamper resistance**. Available at: <https://en.wikipedia.org/wiki/Tamper_resistance>. Accessed on oct 10, 2018.
[63] WIKIPEDIA. **Common Criteria**. Available at: <https://en.wikipedia.org/wiki/Common_Criteria>. Accessed on oct 10, 2018.
[64] WIKIPEDIA. **FIPS 140-2**. Available at: <https://en.wikipedia.org/wiki/FIPS_140-2>. Accessed on oct 10, 2018.
[65] PALMA, Lucas, HENRIQUE, Luiz. **Adição de funções de criptografia simétrica em um applet de código aberto com suporte via middleware OpenSC**. 2017.
[66] ORACLE. **An Introduction to Java Card Technology - Part 1**. Available at: <https://www.oracle.com/technetwork/java/javacard/javacard1-139251.html>. Accessed on oct 10, 2018.
[67] GLOBALPLATFORM. **Welcome to GlobalPlatform**. Available at: <http://globalplatform.org/>. Accessed on oct 10, 2018.
[68] COLDCARD. **Coldcard wallet**. Available at: <https://coldcardwallet.com/>. Accessed on oct 10, 2018.
[69] MICROPYTHON. **MicroPython**. Available at: <https://micropython.org/>. Accessed on oct 10, 2018.
[70] DYKE, Rob. **The Benefits of Trusted User Interface**. Available at: <https://www.trustonic.com/news/blog/benefits-trusted-user-interface/>. Accessed on oct 10, 2018.
[71] RASHID, Saleem. **Breaking the Ledger Security Model**. Available at: <https://saleemrashid.com/2018/03/20/breaking-ledger-security-model/>. Accessed on oct 10, 2018.
[72] STATUS. **Status hardwallet**. Available at: <https://hardwallet.status.im/>. Accessed on oct 10,

2018.

[73] TREZOR. **Entering your PIN**. Available at: <https://doc.satoshilabs.com/trezor-user/enteringyourpin.html>. Accessed on oct 10, 2018.

[74] GITHUB. **BIP44: Account discovery**. Available at: <https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki#Account_discovery>. Accessed on oct 10, 2018.

[75] GITHUB. **WebUSB API**. Available at: <https://wicg.github.io/webusb/>. Accessed on oct 10, 2018.

[76] STATISTA. **Size of the Bitcoin blockchain from 2010 to 2018, by quarter (in megabytes)**. Available at: <https://www.statista.com/statistics/647523/worldwide-bitcoin-blockchain-size/>. Accessed on oct 10, 2018.

[77] GITHUB. **The bitcoin blockchain API powering Insight**. Available at: <https://github.com/bitpay/insight-api>. Accessed on oct 10, 2018.

[78] CARDLOGIX. **NXP JCOP J2A080 v2.4.1 Rel 3 Java Card 80K**. Available at: <https://www.cardlogix.com/product/nxp-j2a080-java-card-80k-2-4-1-rel-3-jcop/>. Accessed on oct 10, 2018.

[79] GITHUB. **GlobalPlatformPro**. Available at: <https://github.com/martinpaljak/GlobalPlatformPro>. Accessed on oct 10, 2018.

[80] GITHUB. **jCardSim**. Available at: <https://github.com/licel/jcardsim>. Accessed on oct 10, 2018.

[81] GITHUB. **SatoChipApplet**. Available at: <https://github.com/Toporin/SatoChipApplet>. Accessed on oct 10, 2018.

[82] GITHUB. **IsoApplet**. Available at: <https://github.com/philipWendland/IsoApplet>. Accessed on oct 10, 2018.

[83] GITHUB. **Status hardware wallet source code**. Available at: <https://github.com/status-im/hardware-wallet/>. Accessed on oct 10, 2018.

[84] AMAZON. **Ledger HW1**. Available at: <https://www.amazon.com/Ledger-HW-1-HW-1/dp/B015N9Z4DU>. Accessed on oct 10, 2018.

[85] TREZOR. **Trezor Hardware Wallet**. Available at: <https://trezor.io>. Accessed on oct 10, 2018.

[86] LEDGER. **Ledger Blue**. Available at: <https://www.ledger.com/products/ledger-blue>. Accessed on oct 10, 2018.

APPENDIX I - PAPER

Recommendations for implementing a Bitcoin wallet using smart card

Ricardo V. Fritsche, Lucas M. Palma, Jean E. Martina

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Campus Universitário Trindade Cx.P. 476 / CEP 88040 – Florianópolis – SC – Brazil

ricardo@grad.ufsc.br, lucas.palma@posgrad.ufsc.br, jean.martina@.ufsc.br

Abstract. *On the Bitcoin peer-to-peer electronic cash system the user's funds are protected by private keys that must be kept safe. In this work we made a review on cryptography, the Bitcoin protocol and secure elements, then we dived into the project of hardware wallets, discussing different requirements and ways to construct one. Our proposed device uses an anti-tamper Java Card to store the private keys. We considered variations of the device, one with a dedicated touchscreen and another with NFC to integrate with a mobile phone. We analyzed security aspects of the project, made recommendations and described some challenges. Finally, we implemented our own open source prototype, showing the architecture of the project, its components, the requirements, the APDU communication protocol and the results.*

1. Introduction

Bitcoin is a peer-to-peer electronic cash system that allows any two willing parties to transact directly with each other without the need for a trusted third party [1]. Bitcoin has grown rapidly, giving birth to a novel industry, involving other cryptocurrencies, blockchain technology and new economic dynamics [2]. Many users might prefer to leave their coins on an exchange as it seems to be easier and works almost like the current online banking solution [3]. The problem is that if the users are not the real owners of their funds, the whole idea of decentralization is lost, giving space to government censorship, retention of user's funds and other threats [7].

The viable alternative for the users is to use hardware wallets, devices that offers the best balance between very high security and ease of use [23]. The main wallets on the market have limitations on the number of cryptocurrencies supported and a high price, but the main concern is that they still present some security flaws, like the exposure of the communication channel [6] and the openness to a Supply Chain and Evil Maid attacks [21].

In this academic work we did a thorough description of some important security aspects of managing keys and Bitcoin addresses. Not only that, we developed an open source prototype that might be a starting point for more robust wallets that are low cost, secure and can be adapted for different coins. We analyzed different hardware wallet projects possibilities, with and without a dedicated screen. We explore the solutions to encrypt the communication channel, preventing Man-in-the-Middle (MitM) attacks [6]

by using password-authenticated secure channel protocol (SRP) [12] and public key cryptography.

2. Fundamentals

In this chapter we make a brief explanation of the main cryptography concepts used by the Bitcoin protocol. We also detail the protocol itself, explore Java Cards and related works.

2.1. Cryptography

Cryptographic hash function takes a variable-length block of data as input and returns a fixed-size value called hash [13]. Any change in the message will have a very high probability to produce a distinct hash, allowing to check for data integrity and various others applications, notably digital signatures [14]. They present different properties: deterministic, fast to compute, preimage resistant, second preimage resistant and strong collision resistant [13]. The Secure Hash Algorithm (SHA) is a family of widely used hash functions that is present in the Bitcoin protocol.

Message Authentication Code (MAC) algorithm has two purposes: to verify the integrity and the authenticity of a message [13]. To form a MAC one can use a cryptographic hash function, such as SHA-256, combined with a secret key, this is known as Hash-based Message Authentication Code (HMAC) [13]. They also can be used to generate pseudorandom numbers of fixed length, which is explored by the Bitcoin protocol to generate wallets.

Key Derivation Function (KDF) takes an input (usually a password or passphrase) and produces a secret key that can fulfil some required format, for example, a certain length and higher entropy. More than that, it is used to prevent brute force attacks, as two elements are present in the function: the use of a salt, avoiding the pre-computation of keys and the usage of iteration, requiring more computational power to execute the function [16].

Digital Signatures are one of most important instances of public-key cryptography [14], with applications on a wide range of areas, including secure e-commerce, legal signing of contracts, secure software updates, online banking and Bitcoin, where a transaction is securely signed, allowing only the owner of the signing private key to spend funds associated with a referenced public key [5]. They present three important aspects: authentication, integrity and non-repudiation. It should be computationally infeasible to generate a valid signature for any given message without knowing the private key, while verifying the signature using the public key should be trivial.

Elliptic Curve Digital Signature Algorithm (ECDSA) is used as an alternative to RSA/DSA and other schemes for digitally signing messages, with the advantage that the same level of security can be achieved using a much smaller key size [13]. Bitcoin protocol has chosen the secp256k1 curve parameters defined in the Standards for Efficient Cryptography (SEC) to make all the ECDSA operations [17].

2.2. Bitcoin

Bitcoin is a Peer-to-Peer Electronic Cash System initially developed by Satoshi Nakamoto in late 2008 to allow any two willing parties to transact directly with each other without the need for a trusted third party [1]. All Bitcoin transactions are transmitted to the network to be propagated to all participating nodes forming a distributed database called blockchain. To prevent double spending of coins and reach global decentralized consensus, the nodes on the network execute a proof-of-work algorithm based on cryptography hashing. To transfer bitcoins to a recipient, one must own the ECDSA's private key that will sign the transaction and release the funds. The transactions are designed to be cheap and fast (disregarding the whole energetic cost to mine coins). Bitcoin uses addresses as pseudonyms, where the addresses are not tied to a person or company, but to a cryptography public key. When miners execute the proof-of-work algorithm before all others nodes, they are rewarded with freshly created bitcoins and the fees from the transactions. There is a limitation of total number of coins on the specification of the protocol, only 21 million bitcoins can be created, making Bitcoin deflationary [18].

2.3. Bitcoin wallets

Bitcoin wallets generally refers to the client software used to manage bitcoin private keys, generate addresses, forge transactions and aggregate balance information from the Bitcoin network. Wallets comes in different formats, ranging from simple local storage of keys on a desktop computer, to mobile applications, dedicated hardware devices, paper wallets or even brain wallets [4].

The first generation of Bitcoin wallets, like the *Bitcoin Core Wallet*, generates random keys and store them on the local filesystem, encrypted by a password or not [4, 18]. This approach presents headaches and security exposure. The user has to constantly backup the keys to other devices and trust that the computer is not infected by any kind of malware and that no unauthorized person has access to it. Hierarchical Deterministic Wallets (HD wallets) [15], on the other hand, represent a better solution. In this kind of wallet, any number of keys can be derived from a single master key, called seed, forming a hierarchic tree of accounts, each with its public and private keys. To be able to backup the wallet, the seed phrase is generated from a random number that is converted to a phrase with 12, 15, 18, 21 or 24 words from a standard wordlist, allowing different wallet software to generate compatible seed phrases. With 12 words the generated entropy is 128 bits and with 24 words the entropy is 256 bits. The order of the words matter and the phrase contain a checksum [24]. To make a HD wallet from a seed phrase, the words are used to derive a longer seed through the use of the key-stretching function PBKDF2 with HMAC-SHA512 [18].

2.4. Java Card

Secure elements (SE) are hardware parts designed to be tamper resistant, not allowing an attacker to retrieve or modify information in the its secure memory [20]. They are resilient to microprobing, software attacks, eavesdropping and fault generation [11].

Java Card is a SE platform released in 1996 to simplify the development of software for smart cards by offering portability and security [8, 9]. Java Card products are based on the Java Card Platform specifications, which allows a Java Card applet to be written once and run on different smart cards running a virtual machine on top of its operational system.

2.5. Related work

Gkaniatsou A., Arapinis M. and Kiayias A. released a paper in 2017 [6] where they reverse-engineer the APDU communication protocol of the *Ledger Nano S* hardware wallet [19], mounting a series of attacks to demonstrate the vulnerabilities. The attacks are really serious and serve as a reminder of the risks of sending clear text data through an insecure channel.

With that in mind we investigated different possibilities to deploy a secure channel on a Java Card and found the work of Hölzl, M., et al. (2015) [12]. The authors design, implement and evaluate the use of the password-authenticated secure channel protocol (SRP).

Beyond our informal research on current wallets products, we selected an academic paper by Eskandari S., et al. [4] where the main wallets solutions until the year of 2015 are mapped, studied and compared, mainly in the point of view of usability and user experience. Table 1 show our proposal as an additional last line to allow comparison with the others categories. As the keys are stored on a secure element, which cannot be tampered, we gave Malware Resistant a full point (•). On the Resistant to Physical Theft aspect, we gave our solution the half point (◦) as the device can be stolen but no keys can be extracted. Finally, on the No New User Software we also gave our solution the half point (◦), considering that the user will access the wallet through a web interface, but it is necessary to install the *Connector* (described in the Development chapter) software or at least an extension to the web browser.

Table 1: A comparison of key management techniques for Bitcoin. Modified from Eskandari S., et al. [4]

Category	*Example*	Malware Resistant	Key(s) Kept Offline	No Trusted Third Party	Resistant to Physical Theft	Resistant to Physical Observation	Resilient to Password Loss	Resilient to Key Churn	Immediate Access to Funds	No New User Software	Cross-device Portability
Keys in Local Storage	Bitcoin Core			•		•	•	•	•		
Password-protected Wallets	MultiBit		◦	•	◦	•		•	•		
Offline Storage	Bitaddress	◦	•	•			•				•
Air-gapped Storage	Armory	◦	•	•		•	•	•			
Password-derived Keys	Brainwallet		•	•	◦			•	•	•	•
Hosted Wallet (Hot)	`Coinbase.com`						•	•	•	•	•
Hosted Wallet (Cold)		◦	•				•	•		•	•
Hosted Wallet (Hybrid)	Blockchain.info		◦	◦			•	•	•	•	•
Cash		•	•	•		•	•	•	•	•	•
Online Banking							•	•	•	•	•
Our solution		•	•	•	◦	•	•	•	•	◦	•

3. Development

In this chapter we explore different possibilities to build a hardware wallet, discuss relevant aspects considerations and present our open source prototype.

3.1. Different possibilities for the hardware wallet

A hardware wallet can be constructed in many different ways, bringing unique security advantages, implementation challenges and costs of production for each version. As we have seen in the first chapter, it is really important that the private keys are kept on a secure element, making the smart card (Java Card) the basic requirement in our analysis. Another general precondition is the communication with an external computer (desktop, notebook or even a mobile phone), so the wallet can interact with the Bitcoin network and the user.

The most basic version does not include a screen, although it has the lowest cost. To make it more secure, we must include a screen. For this we need a microcontroller that will be able to make the interconnection between the smart card, the screen, the optional buttons and the communication channels (like USB, Bluetooth, etc.). This will increase the cost of the device and make the development more complex, but it is an unavoidable requirement, as we cannot trust a third-party device.

3.2. Security considerations

Independent of the different constructed wallet, there are common security considerations that must be taken care of:

1. Protect the communication channel using SRP or other method;
2. Protect the device with a PIN using the Java Card PIN SDK;
3. Allow to enter the PIN on an untrusted device without revealing it by showing the keyboard numbers in a random fashion on the device;
4. Allow the personalization of the device to prevent *Evil Maid* attacks;
5. Make it possible to use plausible deniability by allowing any PIN to generate a valid wallet;
6. Protect the hardware against attacks by disabling debug ports, monitoring the components actively and passively and potting of device;
7. Be able to verify the genuineness of the SE by recording its unique public key on the Bitcoin blockchain.

3.3. Prototype

To go beyond the theoretical knowledge, we developed a functional prototype using a real smart card (Java Card). The prototype did not have a dedicated screen or buttons and did not use a secure channel on its communication, although it checks for the attestation of the genuineness of the smart card. Figure 1 shows an overview of the general architecture of the project.

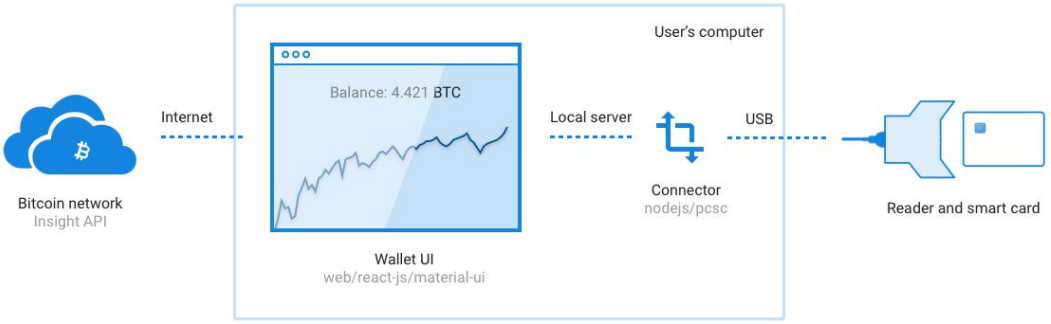


Figure 1: Overview of the architecture of the project.

The secure element selected was the NXP J3D081 EV1 smart card. It is a contact and contactless interface Java Card 3.0.1 running on Global Platform 2.2.1 with 80K of EEPROM. All the requirements (Figure 2) were implemented and tested with unit and integration tests. All the APDU commands and responses were documented.

SETUP

1.1. To define the PIN to access the device;
1.2. To change the PIN; (PIN required)
1.3. To generate the initial master private key of the device using a TRNG and the seed words for recovery (BIP-39)[1]; (PIN required)
1.4. To recover a wallet by defining the master private key using the seed words; (PIN required)
1.5. To verify that the seed words generates the master private key stored on the device; (PIN required)
1.6. To generate the attestation of genuineness keys using a TRNG.

ADDRESS GENERATION

2.1. To generate deterministic BIP-32 address (public/private keys). (PIN required)

SIGN TRANSACTIONS

3.1. To sign Bitcoin transactions using the keys of a BIP-32 path; (PIN required)

INTERACTION

4.1. To validate the user PIN;
4.2. To take actions based on PIN policies, like to erase the device if the PIN is wrong for 5 consecutive times;
4.3. To sign a challenge (SHA-256 hash) with the genuineness private key;
4.4. To respond with the genuineness public key of the device;
4.5. To erase the device if request by the user. (PIN required)

Figure 2: Requirements of the hardware device.

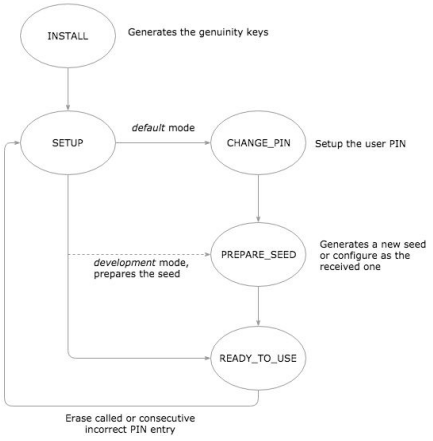


Figure 3: Possible states of the Applet.

We developed a middleware called *Connector* that makes the bridge between the USB smart card reader and the browser, by making it run a local web server on the client computer. This component starts a local HTTP server on port 28281 and listens to calls made from whitelisted domains or localhost. Java 1.8 was used for listening and responding to HTTP requests.

Figure 4 shows the main actions that the interface should support in a workflow format. These actions were used to draw and then implement the web interface (see Figure 5) as a Single Page Application (SPA). To communicate with the Bitcoin blockchain we used a widely adopted open source solution called *Insight-API*, maintained by *BitPay* [22].

The source code of this project is publicly available at GitHub on the following repository: *https://github.com/ricardovf/knox-wallet*. Be advised that this project is only a prototype and isn't ready to be used in a real product.

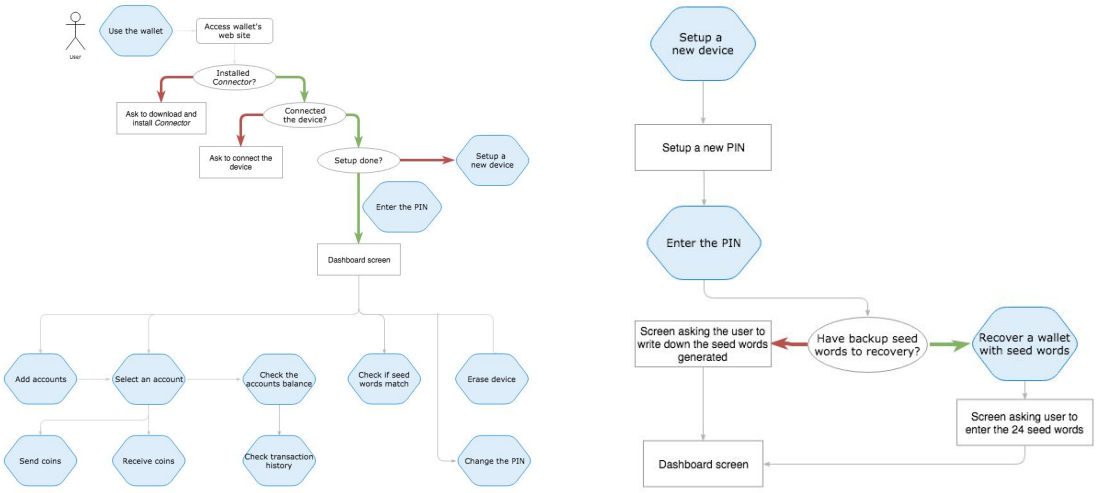


Figure 4: Workflow of navigation and setup on the wallet user interface.

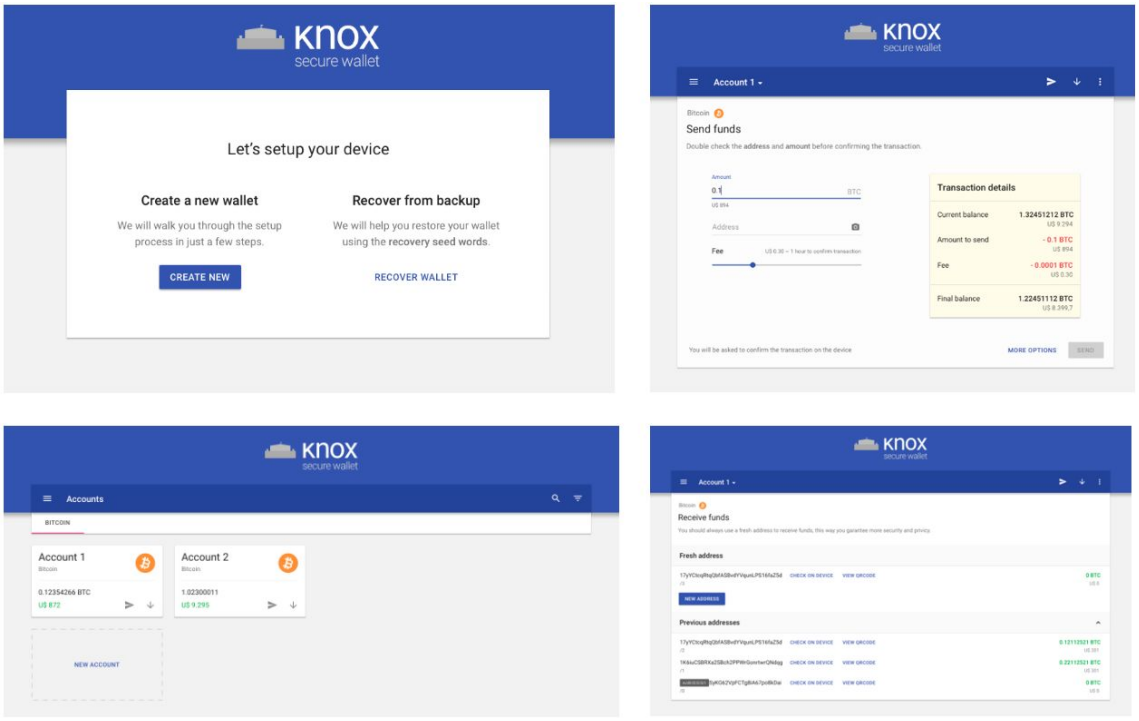


Figure 5: Setup, send funds, dashboard and receive funds screens.

4. Conclusion

We dived into the project of hardware wallets, discussing different requirements and ways to construct a device that uses an anti-tamper Java Card to store the private keys. We considered variations of the device and analyzed the security aspects of the project, making recommendations and describing some challenges. Finally, we implemented our own open source prototype, showing the architecture of the project, its components, the requirements, the APDU communication protocol and the results. The prototype constructed is limited in contrast to the more complete versions discussed, with a dedicated screen and an encrypted communication channel, but it can be further

developed to accommodate the full requirements and be deployed into the world as a complete product.

Hardware wallets projects are complex, involving integrated circuits and software that must communicate and integrate securely, considering different attack vectors, without letting out the importance of the final cost and usability. Nevertheless, we must constantly encourage the enhancement of the current solutions and develop novel approaches, making cryptocurrencies a practical everyday solution to contrast with the traditional financial system.

We believe there is a rich set of possibilities for future work related to this project, most of them on the practical side of extending and validating the security of the wallet:

1. Build prototype of the wallet with an integrated touchscreen;
2. Build a prototype using a contactless smart card (NFC) and a mobile phone as the device screen;
3. Systematically attack the wallet to find vulnerabilities and propose corrections;
4. Run usability test with real users to improve the wallet friendliness;
5. Implement the support for multiple cryptocurrencies, including ones not directly derived from Bitcoin Core;
6. Implement the support for multisignature scheme;
7. Find solutions to securely guarantee the genuineness of the integrated microcontroller;
8. To write an updated book on Java Card development utilizing open source libraries and tools, as the materials on the topic are very fragmented and diffuse.

References

[1] NAKAMOTO, Satoshi. **Bitcoin: A peer-to-peer electronic cash system**. 2008.
[2] Monopoly without a Monopolist: An Economic Analysis of the Bitcoin Payment System
[3] CONTI, M., E, S.K., LAL, C. & RUJ, S. 2018. **A Survey on Security and Privacy Issues of Bitcoin**. IEEE Communications Surveys & Tutorials :1–1.
[4] ESKANDARI, S., BARRERA, D., STOBERT, E. & CLARK, J. 2015. **A First Look at the Usability of Bitcoin Key Management**. Proceedings 2015 Workshop on Usable Security.
[5] COURTOIS, Nicolas T.; VALSORDA, Filippo; EMIRDAG, Pinar. **Private Key Recovery Combination Attacks: On Extreme Fragility of Popular Bitcoin Key Management, Wallet and Cold Storage Solutions in Presence of Poor RNG Events**. 2014.
[6] GKANIATSOU, A., ARAPINIS, M. & KIAYIAS, A. 2017. **Low-Level Attacks in Bitcoin Wallets**. Information Security :233–253.
[7] MAY, Timothy. **The Crypto Anarchist Manifesto**. Available at: <https://www.activism.net/cypherpunk/crypto-anarchy.html>. Accessed on may 29, 2018.
[8] BARBU, Guillaume. **On the security of Java Card platforms against hardware attacks**. 2012. Tese de Doutorado. Telecom ParisTech.
[9] HANSMANN, Uwe et al. **Smart card application development using Java**. Springer Science & Business Media, 2012.
[10] ATKINS, Steve. **JCF celebrates 15 year anniversary in Singapore – 10 billion Java Cards deployed worldwide**. Available at:

<https://contactlessintelligence.com/2012/09/21/jcf-celebrates-15-year-anniversary-in-singapore-10-billion-java-cards-deployed-worldwide/>. Accessed on may 29, 2018.
[11] KÖMMERLING, Oliver; KUHN, Markus G. **Design Principles for Tamper-Resistant Smartcard Processors**. Smartcard, v. 99, p. 9-20, 1999.
[12] HÖLZL, Michael et al. A password-authenticated secure channel for App to Java Card applet communication. **International Journal of Pervasive Computing and Communications**, v. 11, n. 4, p. 374-397, 2015.
[13] WILLIAM, Stallings. **Cryptography and network security: principles and practice**. Prentice-Hall, Inc, p. 23-50, 1999.
[14] PAAR, Christof; PELZL, Jan. **Understanding cryptography: a textbook for students and practitioners**. Springer Science & Business Media, 2009.
[15] GITHUB. **BIP32**. Available at: <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>. Accessed on jun 6, 2018
[16] KALISKI, Burt. **PKCS# 5: Password-based cryptography specification version 2.0**. 2000.
[17] CERTICOM, Standards for Efficient Cryptography. **SEC 2: Recommended Elliptic Curve Domain Parameters**. Version 2.0. 2010.
[18] ANTONOPOULOS, Andreas M. **Mastering Bitcoin: Programming the open blockchain**. O'Reilly Media, Inc. 2017.
[19] LEDGER. **Ledger Nano S**. Available at: <https://www.ledger.com/products/ledger-nano-s>. Accessed on oct 10, 2018.
[20] WIKIPEDIA. **Tamper resistance**. Available at: <https://en.wikipedia.org/wiki/Tamper_resistance>. Accessed on oct 10, 2018.
[21] RASHID, Saleem. **Breaking the Ledger Security Model**. Available at: <https://saleemrashid.com/2018/03/20/breaking-ledger-security-model/>. Accessed on oct 10, 2018.
[22] GITHUB. **The bitcoin blockchain API powering Insight**. Available at: <https://github.com/bitpay/insight-api>. Accessed on oct 10, 2018.
[23] BITCOIN. **Securing your wallet**. Available at: <https://bitcoin.org/en/secure-your-wallet>. Accessed on may 29, 2018.
[24] GITHUB. **BIP39**. Available at: <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>. Accessed on jun 6, 2018