Real-world Deployability and
Usability of Bitcoin

Shayan Eskandari

A Thesis

in

The Department

of

Electrical & Computer Engineering

Presented in Partial Fulfilment of the Requirements for

the Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

October 2015

CONCORDIA UNIVERSITY

Division of Graduate Studies

This is to certify that the thesis prepared

By :   **Shayan Eskandari**

Entitled :   **Real-world Deployability and**
             Usability of Bitcoin

and submitted in partial fulfilment of the requirements for the degree of

Master of Applied Science

complies with the regulations of this University and meets the accepted standards
with respect to originality and quality.

Signed by the final examining committee :

_____ Chair
Dr. R. Raut

_____ Examiner
Dr. O. Ormandjieva

_____ Examiner
Dr. Y. Liu

_____ Supervisor
Dr. Abdelwahab Hamou-Lhadj

_____ Supervisor
Dr. Jeremy Clark

Approved by   _____

ABSTRACT

Real-world Deployability and

Usability of Bitcoin

Shayan Eskandari

We live in an era where Internet is one of the daily needs of human life. People use Internet banking instead of going to banks, they use email rather than postal mail.This leads to a robust digital way of living, but this also means people are trusting middle companies and third parties for their online services. The need of having a digital form of money that is not being controlled by one entity is plain to see.

Bitcoin is the first and the most popular decentralized virtual currency. It is based on cryptographic functions to remove the need of a central bank and regulates the generation of new units.

In this thesis, we would like to look at available tools to facilitate users in holding and using Bitcoin by a perspective on usability and security, and then evaluate the possibilities for a small business to accept Bitcoin payments. Our focus is on the usability of these tools and developing a useful framework for comparing and evaluating future tools. While many security tools have been studied from a usability perspective, our work is the first to look at Bitcoin.

Acknowledgements

I would like to express my gratitude to my supervisors, without whose their support and patience I could not have completed my master´s degree. I would like to thank Dr. Abdelwahab Hamou-Lhadj, for pushing me to find the subject I really enjoyed working on, and Jeremy Clark, whose expertise, understanding, and patience, added considerably to my graduate experience.

I would like to express my appreciation and thanks to the great co-authors that helped me with publishing the Bitcoin usability paper, David Barrera and Elizabeth Stobert.

A very special thanks goes out to Hamed Masoumi, for his valuable suggestions and cooperation to make Cafe Aunja the first café in Eastern Canada to accept Bitcoin.

I am also thankful to Montreal Bitcoin Embassy crowd, for offering a great environment to embrace the geek within me.

Last but not least, I would like to thank my parents and my sister for the support they provided me through my entire life.

Shayan Eskandari

Contents

List of Figures

Chapter 1

Introductory Remarks

We live in an era where Internet is one of the daily needs of human life and modern countries. Instead of going to banks, people use Internet banking and instead of sending a physical letter they use digital ways of communication. This leads to a robust digital way of living, but this also means people are trusting middle companies and third parties for their online services. The most important ones are banks and financial middle man (*e.g.,* credit card companies) and there has been many downsides to the trust, such as banks failing[1], government collapses that leads to the country's currency exchange rate decrease to pennies (*e.g.,* Zimbabwean dollar [HK09]) and many more examples on smaller scale. The need of having a digital form of money that is not being controlled by one entity is plain to see.

Bitcoin is the first decentralized virtual currency and by far has the most number

---

[1]List of bank failures in the United States (2008present) `https://en.wikipedia.org/wiki/List_of_bank_failures_in_the_United_States_(2008present)`

of users [Nak08]. It is based on cryptographic functions to remove the need of a central bank and regulates the generation of new units. Bitcoin is still in its early stages and there have not been that many practical applications of this digital currency that could offer an ultimate solution for financial problems.

In this thesis, we would like to look at available tools to facilitate users in holding and using Bitcoin by a perspective on usability and security, and then evaluate the possibilities for a small business to be able to accept Bitcoin payments. This could be a summary for the usage of any kind of a currency, as there only should be two entities involved in a monetary transaction, the payee and the payer.

Thesis Statement: End-to-end usable payment systems using Bitcoin, and its components, can be designed for real-world deployability while maintaining a strong notion of usability and security.

1.1   Background

The concept of digital cash was introduced by David Chaum in 1983 [Cha83], he continued the idea and founded a company named DigiCash[2] as a digital cash company. DigiCash filed for bankruptcy in 1998 and sold its assets to eCash Technologies, another digital currency company. In the same year, Paypal[3] emerged and other sys-

---

[2] https://en.wikipedia.org/wiki/DigiCash

[3] http://paypal.com

tems such as E-gold [4] followed, but due to unregulated use of E-gold, these companies were shut down in 2005 by the federal government of the United States. In 2008, Bitcoin was introduced to solve most of the problems of all the digital cash, which marked the start of digital currencies.

Bitcoin was an innovation because it was not a new digital version of cash (eCash) nor a commodity like gold (e-Gold), but it was its own currency with properties that were not seen in any other currencies before.

Even though the idea of having a decentral money is interesting, to the best of our knowledge, there has not been any published work on the usability of this new form of money. Thus a framework to be able to evaluate applications in this field with usability perspective is needed.

For Bitcoin to flourish, adoption must expand beyond developers and tech-savvy enthusiasts to novice users. Expansion solidifies the need for a usable, comprehensible approach to Bitcoin. If users cannot safely manage Bitcoin keys, it may result in the users' loss of funds and/or a poor reputation for Bitcoin, both of which could dissuade further user adoption.

1.2   Contributions

While this research is one of the first usability research on the subject of Bitcoin, our work provides a number of new contributions toward the evaluation of Bitcoin wallet

---

[4]https://en.wikipedia.org/wiki/E-gold

clients and payment systems, we summarize them here:

Bitcoin wallet comparison framework: We design a framework for comparing Bitcoin wallet clients and evaluate the existing tools.

Bitcoin point of sale comparison framework: with the focus on the available tools for businesses to accept Bitcoin as a method of payment, we analyze a small business's requirements using SCRAM [Sut03] a requirement engineering method [Dor90] and evaluate all the available options for Bitcoin payments.

Fully customizable open-source Bitcoin point of sale[5] : later we develop a customized Bitcoin point of sale specific to a small business needs as none of the available approaches could satisfy the needs. This software is available under GNU General Public Licience v2 and has already been used in other small businesses to accept Bitcoin.

Our focus in this thesis is on the usability of these tools and developing a useful framework for comparing and evaluating future tools. While many security tools have been studied from a usability perspective, our work is the first to look at Bitcoin.

1.3 Organization

In the next chapter (Chapter 2), we present some background on Bitcoin, the underlying protocol to the extent that it fits the scale of this thesis, and also some details about the methods used in the next chapters such as cognitive walkthrough [WRLP94]

---

[5]`https://github.com/shayanb/Bitcoin-PoS-PHP`

and requirement engineering.

In Chapter 3, we use cognitive walkthrough to evaluate the usability of Bitcoin wallet clients and then develop a framework for comparing existing and future wallet clients. This work is the first published usability paper on Bitcoin subject, mostly focused on how these clients handle key management that is the fundamental requirement of any Bitcoin wallet client.

In Chapter 4, we survey all the available tools for small businesses to accept Bitcoins and evaluate them based on the framework introduced in the same chapter. Then using a requirement engineering method, we list all the advantages and disadvantages of using each method for a small business and later we develop a fully customized open-source Bitcoin point of sale and implement it in real-world café[6] to accept Bitcoins. To our knowledge, this is the first café in Quebec, Canada that accepts Bitcoin.

And in the end we evaluate our contributions and discuss the future work needed in this field to have a more usable and robust system for holding and accepting Bitcoin as a method of payment.

---

[6] Cafe Aunja `http://aunja.com`

Chapter 2

Backgroud

2.1 Introductory Remarks

Bitcoin or any other digital form of money by nature is an interesting idea for the 21st century, but how practical and useful these would be to the people living in this era is what would make a difference. In this research, we tried to have a real world view of the implications that Bitcoin usage could have and also evaluated the existing approaches to holding, using and accepting Bitcoin as a digital form of money.

2.2 Bitcoin

Bitcoin is a cryptographic currency deployed in 2009 [Nak08], which has reached a level of adoption unrealized by decades of previously proposed digital currencies (from 1982 [Cha82] onward). Unlike many previous proposals, Bitcoin does not dis-

tribute digital monetary units to users. Instead, a public ledger maintains a list of every transaction[1] made by all Bitcoin users since the creation of the currency. A *transaction* in its simplest form describes the movement of some balance of the Bitcoin currency (XBT or BTC) from one or more accounts (called input addresses) into one or more accounts (called output addresses). Bitcoin addresses are indexed by the fingerprint of a public key from a digital signature scheme.[2] They are not centrally allocated or registered in any way—the addresses become active when the first transaction moving money into them is added to the ledger.

In Bitcoin, every standard transaction[3] must be digitally signed using the private signing key associated with each input address in the transaction. In order to spend Bitcoin, users require access to the signing key of the account holding their Bitcoin. Thus users do not maintain any kind of units of currency; they maintain a set of keys that provide them signing authority over certain accounts recorded in the ledger.

The ledger (known as the *blockchain*) is maintained and updated by a decentralized network using a novel method to reach consensus that involves incentivizing nodes in the network with the ability to generate (known as mining) new Bitcoin and collect transaction fees. The details of the Bitcoin consensus model are not relevant to this thesis, but we note that clients in the network participate in the consensus model by downloading and cryptographically verifying the integrity of the blockchain.

---

[1]Technically, a transaction specifies a short script that encodes how the balance can be claimed as the input to some future transaction.

[2]Elliptic Curve Digital Signature Algorithm (ECDSA) [Van92].

[3]From now on, the word transaction refers to a standard transaction unless otherwise specified

As of writing, the Bitcoin blockchain is roughly 30 GB in size.

Due to the large size of the Bitcoin blockchain, full download is infeasible for thin clients running on mobile devices, as well as some desktop clients. These clients connect to a semi-trusted nodes and only request transactions relevant to keys in their wallet. This technique, known as Simplified Payment Verification (SPV), eliminates the need to download and verify the entire blockchain but, when implemented incorrectly, can create privacy risks [GKGC14]. Also it has integrity risks, in the sense that it cannot forge transactions but could omit reports of relevant transactions.

In this section, we focus on introducing the essential constituents of Bitcoin that is needed in this research. Some details have been simplified to prevent going outside the scope of this thesis. Every aspect of Bitcoin that is missing from this introduction is explained through this thesis when the preliminary information of the usage is known to the reader.

2.2.1   Bitcoin Address

A Bitcoin address is a string of 26-35 alphanumeric characters that starts with "1" or "3", that contains digits, uppercase and lowercase letters with the exception of "O", "I" (Uppercase i) , "l" (Lower case L) and the number 0 to prevent visual ambiguity. Bitcoin addresses are commonly shared via QR-Code as it is easier to read with QR-code mobile scanners and is also implemented in most of the Bitcoin wallets as the main method to exchange addresses (see figure 2.1 that is a representation of the Bitcoin address 1shaYanre36PBhspFL9zG7nt6tfDhxQ4u). Bitcoin addresses are

Figure 2.1: QR-Code representing a Bitcoin address

derived from the equivalent ECDSA public key that will be explained shortly.

Public Key

In other words Bitcoin address is 160-bit hash of the public portion of the public and private ECDSA keypair. A public key is derived from the private key by some cryptographic functions (see Figure 2.2).

Private Key

A private key can be any 256 bit number from 0x1 to 0xFFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFE BAAE DCE6 AF48 A03B BFD2 5E8C D036 4140. Basically any number in this range would be valid as the input for secp256k1[4] ECDSA standard. This is the secret part of the Bitcoin address that should be kept secure and there are already different methods of securing the private key as discussed in

---

[4]Standards for Efficient Cryptography (SEC) `https://en.Bitcoin.it/wiki/Secp256k1`
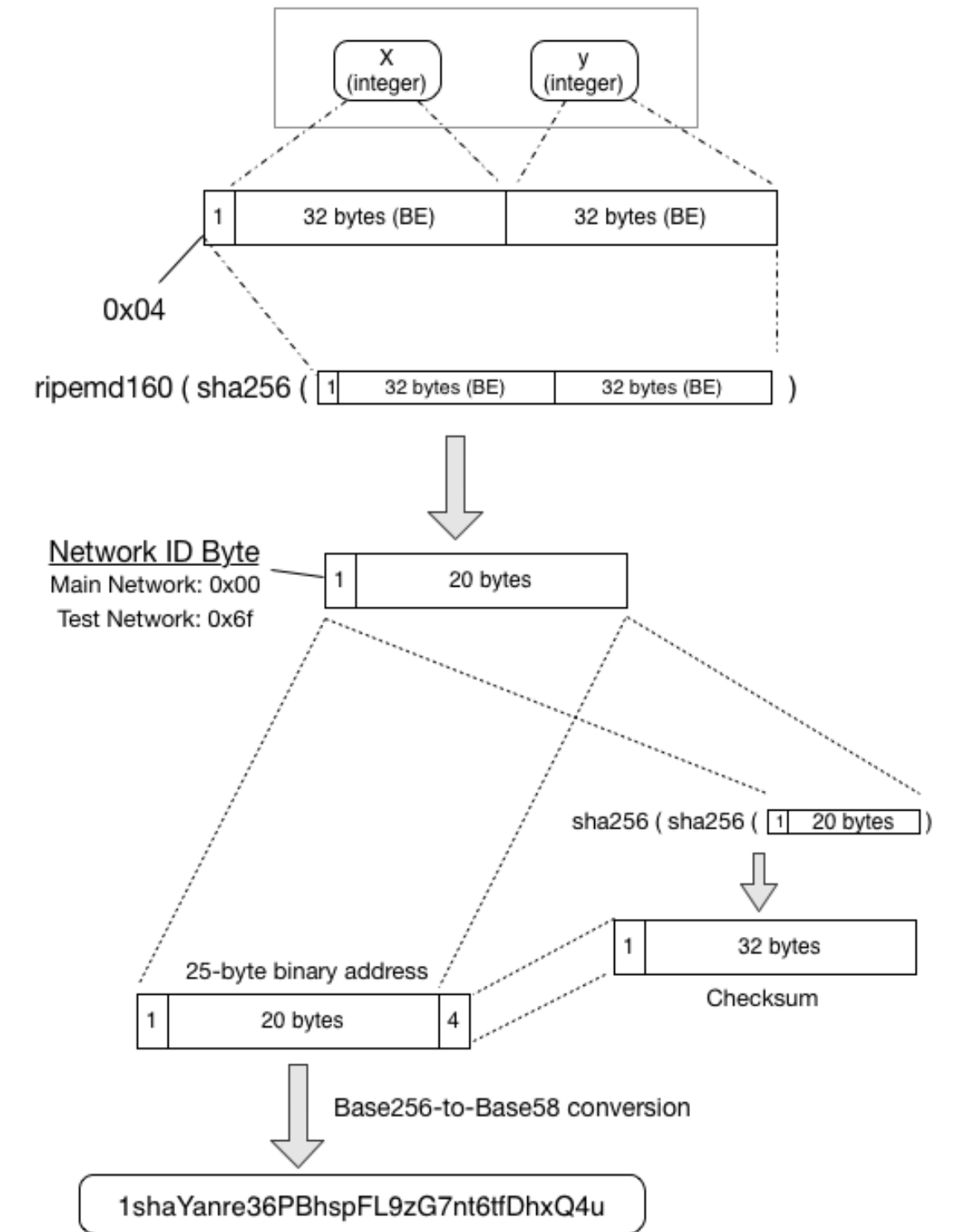
Figure 2.2: ECDSA Public key to Bitcoin Address [Wik]

Chapter 3. Anyone with the private key has the ability to sign a transaction and spend Bitcoins that are signed with the relevant public key (or Bitcoin address). Same as the Bitcoin addresses and the public keys, private keys have a shorter format called wallet import format (wif) that is used commonly by most Bitcoin wallet clients. It contains checksum bits and also some information about the public key associated to the private key. An example of a private key in wif format would be 5Kb8kLf9zgWQnogidDA76MzPL6TsZZY36hWXMssSzNydYXYB9KF, which would result in 1CC3X2gu58d6wXUWMffpuzN9JAfTUWu4Kj as the associated Bitcoin address.

To better understand how Bitcoin address is derived from the private key see Figure 2.2.

BIP32

As Bitcoin creator, Satoshi Nakamoto [Nak08] also points out, it is better to generate a new address for each transaction and receive the changes in a another new change address (The concept of change addresses is explained more in chapter 3) to use the pseudonymity of Bitcoin. This brought a challenge to Bitcoin wallet client designs. Keeping track of all the addresses in the wallet file and also the ability to back up the private keys, could get complicated when there are hundreds of keys in the wallet (more in Chapter 3).

Since Bitcoin is an open source project, to make improvements to the proto-

11

col, Bitcoin Improvement Proposals (BIP) are introduced by developers to be imple-mented in the core code. One important proposal is BIP32 also known as Hierarchical Deterministic Wallets or HD wallets [Pieb]. BIP32 introduces the ability to gener-ate a tree of addresses from a single seed known as "Master Seed". A master node would be generated from the master seed and it is possible to branch it to multiple nodes shown as m/0, m/1, . . . , m/i , and then from each multiple addresses could be derived such as m/0/1, m/0/2, *etc.*. Each of these derivation has a path in the tree that is called "BIP32 Path", *e.g.,* second branch of the first branch of master seed m is shown as "m/0/1". Commonly the first branch would be used on different accounts and the second depth of nodes on the tree would be used as the addresses of each account. BIP32 simplifies the backing up process as there is just a seed to be backed up and it is easier to keep track of the addresses. It should be mentioned that there are two parts to BIP32, the private key that starts with "xprv" and the public part that starts with "xpub". BIP32 allows a simple terminal that does not contain the private keys to generate new Bitcoin addresses (for use in point of sale terminals) using the public portion of BIP32 seed. A visualization of how BIP32 was designed can be seen in Figure 2.3.

2.2.2 Bitcoin Wallet

This term has been conflated in Bitcoin sphere as both the file that contains the private keys and also the software client used to mange Bitcoin transactions. This concept is more discussed in Chapter 3. For the sake of simplicity, we use the term
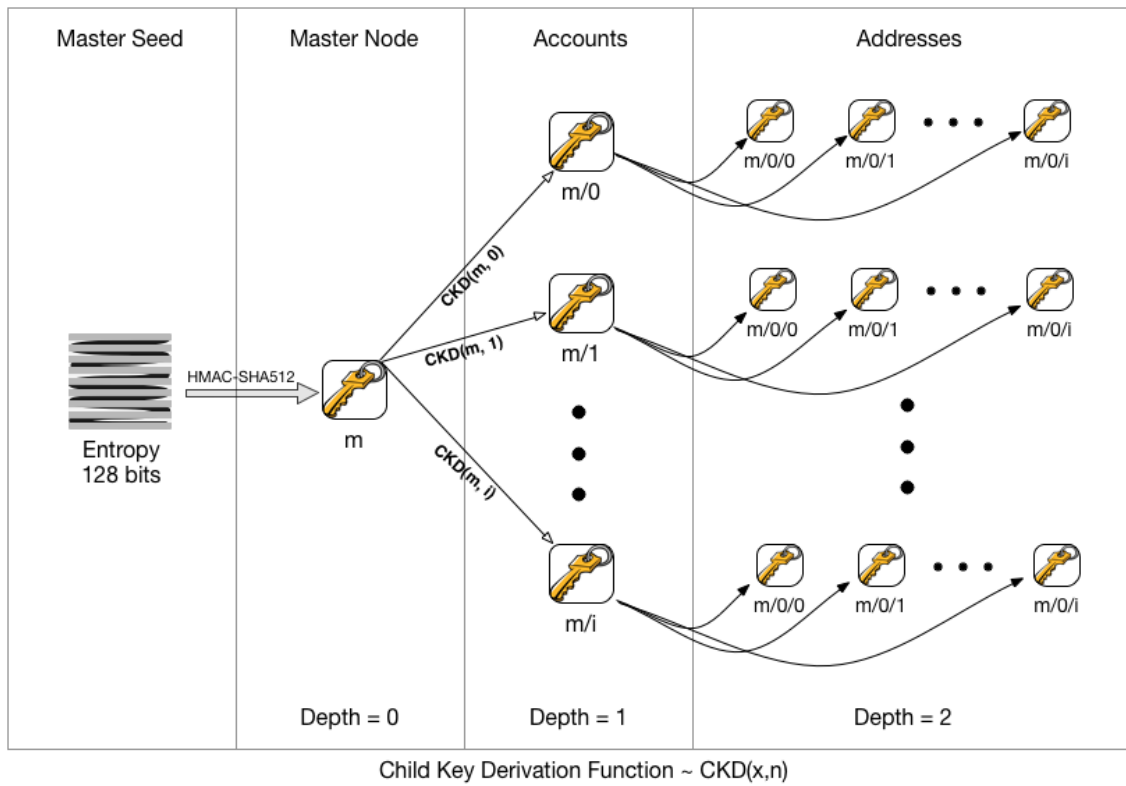
Figure 2.3: BIP32 - Hierarchical Deterministic Wallets [Piea]

Bitcoin wallet client as in the software used to sign and manage Bitcoin transactions, and Bitcoin wallet file to the file containing the private keys. The first and official Bitcoin wallet client is Bitcoin Core (Bitcoind as the daemon) that will be explained more on Chapter 3.

2.2.3 Confirmation

When a Bitcoin transaction broadcasts to the network, it should be included in a block by Bitcoin miners. Bitcoin miners are the computational devices, which are using their hashing power (computational power) to verify each and every transaction within Bitcoin network and include them in a block. A block is a group of transactions put together by a miner. Each block would be added to the blockchain with a hash referencing its previous block and everyone in the network should form the consensus that the hash value for the block and the previous one is correct. As soon as the transaction is broadcasted it has 0-confirmation, meaning it has not been added to any blocks yet. As soon as it gets included in a block it has 1 confirmation and this increases by each block that gets added to the blockchain. The miner gets rewarded by Bitcoin network for his work with 25 Bitcoins[5].

It is possible for a miner[6] to fork the blockchain and start to mine blocks on his

---

[5]Bitcoin generation algorithm rewards the miner based on a controlled supply algorithm `https://en.bitcoin.it/wiki/Controlled_supply`

[6]For the sake of simplicity of the language, in this thesis, we refer to the miners as "He", as in the operator of the mining devices. A miner is a computational device that could be operated by a single person or a group of people.
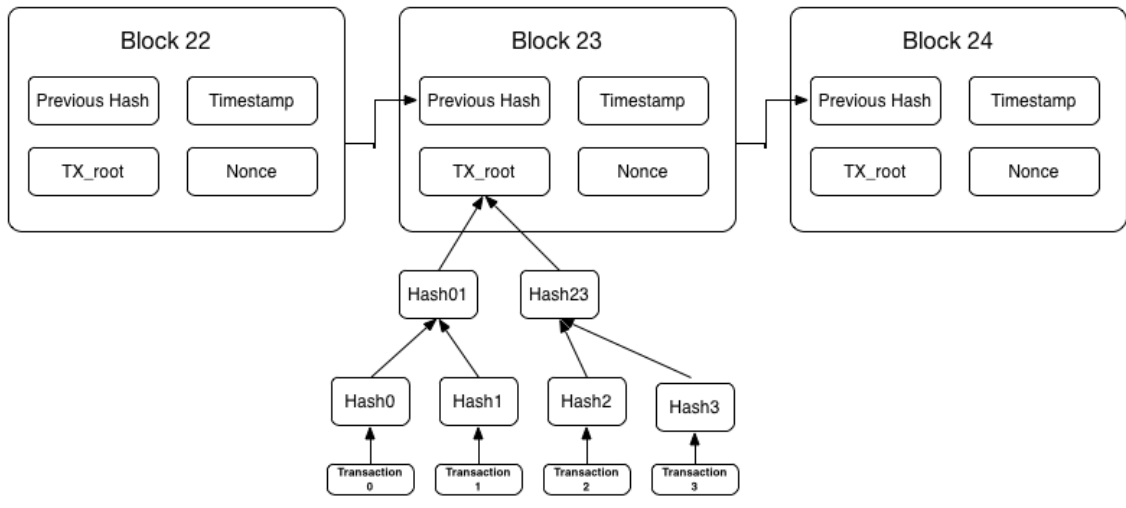
Figure 2.4: Bitcoin Blocks in the blockchain [Nak08]

own fork, here is when the consensus model is important as the coins that were mined in his fork are invalid in the Bitcoin network and there is no reference in the main blockchain that those coins exists. The consensus model dictates that if a node sees two forks of the blockchain it should chose the longest chain. This check happens on every new block that is added to the blockchain and seen by the node.

This is important to understand that it is possible for a 0-confirmation transaction to stay unconfirmed for a some time, depending on how much miner's fee is included in the transaction and some other factors. Miner's tend to chose the transactions with higher miner's fee to be included in the new block first, but this is not a universal fact. Thus the miner is in control to chose which transactions should be included in the blocks he mines[7].

---

[7]This is why 51% attack becomes noticable, as the miner that has 51% of the hashing power would be in control to reject transactions specific to some addresses or whitelist some others to be

2.2.4 Double Spend

One of the major innovations of Bitcoin was to solve a problem known as "double spending". For any digital data it is possible to duplicate the data and send it to two different entities, this could be a problem when this digital data has monetary value. Details of the solution would rely on a cryptographic description of how Bitcoin protocol works. However to put it in simple words, when spending some Bitcoins as an input to a transaction, the Bitcoin wallet should use the hash of those inputs that are already in the blockchain, when it broadcasts that transaction it will be stored on the memory of all the Bitcoin nodes as an unconfirmed transaction[8] until it gets included in a block by a miner. In this period of time, it is possible to use the same input in another transaction and broadcast it to the network, mainly because that input was not spent in any blocks yet. Although only one of these transactions could be included in a new block and the other one would be invalid and erased from the memory. This makes it complicated for Bitcoin payment processors to accept 0-confirmation transactions as they could not be fully trusted.

included in new blocks. however he will not be able to forge transactions.

[8]mempool

Chapter 3

A First Look at the Usability of Bitcoin Key Management

This chapter is adapted from published work by Shayan Eskandari, supervised by Jeremy Clark and co-authored by David Barrera and Elizabeth Stobert [EBSC15]

3.1 Introductory Remarks

In all of the excitement surrounding Bitcoin, it is easy to forget that the decentralized currency assumes a solution to the longstanding problem of usable public key cryptography for user authentication. Studies of the usability of key management [GM05, GMS$^+$05, SBKH06, GFFK06] have shown that there are numerous usability issues that prevent public key cryptography from being effectively leveraged

17

by end users. Managing, controlling, and using cryptographic keys are complex tasks, and no clear solution has been proposed.

Despite the known complexity in creating and managing cryptographic keys, the Bitcoin network and software clients use such keys extensively for many operations. For example, digital signatures, which require the Bitcoin software to read private keys into memory, are used to assert ownership over a specific set of Bitcoins. Thus, managing the same coins on multiple devices (*e.g.,* a desktop and a phone) requires the corresponding private keys to be copied to and made accessible on these devices.

The consequences of losing exclusive control over an account containing monetary value connects the threat of losing a Bitcoin private key to that of losing an online banking password. However, consumers in many countries are legally protected from any liability of banking credential loss. Furthermore, most bank transactions are traceable and reversible, making it difficult to extract value from stolen banking credentials (most techniques involve a mule [FH12]). Bitcoin transactions are also traceable, however they are not reversible. Stolen Bitcoins can thus not be centrally or automatically recovered. Bitcoin users typically have no legal protection against loss or theft, and while stolen Bitcoins could be traced as they change ownership,[1] several mechanisms exist for laundering Bitcoins and similar digital currencies [MGGR13, BNM+14].

In an effort to address some of the complexities of key management, developers

---

[1]Public keys associated with specific Bitcoins are publicly available in the Bitcoin blockchain, but the identities of users who control those keys are not.

of Bitcoin software have created a variety of innovative technologies ranging from password-derived keys to air-gapped computers to physical printouts of private keys in the form of 2D barcodes. However, since none of these proposals have been evaluated in the Bitcoin context, it remains unclear which techniques have usability advantages.

For Bitcoin to flourish, adoption must expand beyond developers and tech-savvy enthusiasts to novice users. Expansion solidifies the need for a usable, comprehensible approach to Bitcoin. If users cannot safely manage Bitcoin keys, it may result in the users' loss of funds and/or a poor reputation for Bitcoin, both of which could dissuade further user adoption.

In this paper, we aim to investigate the usability challenges surrounding key management in Bitcoin. To do this, we survey and categorize the most prominent Bitcoin key management proposals. Next we conduct an expert usability inspection technique known as a cognitive walkthrough [WRLP94] on popular examples of each proposal. Our goal is to identify overarching usability issues as well as advantages of specific proposals, allowing us to propose design recommendations for future Bitcoin clients. Specifically, the contributions of the paper are as follows:

- We perform a broad survey of six Bitcoin key management techniques which cover the vast majority of deployed Bitcoin software.
- Using the results from our survey, we propose an evaluation and comparison framework for Bitcoin key management techniques. The framework is based on 10 security, usability and deployability criteria, and enables direct comparison

of current and future key management proposals. Using our framework we find that certain properties, such as trust in a central party enable additional beneficial properties. We also find that the disadvantages of certain properties, such as malware protection, outweigh the relative benefits.

- We perform a cognitive walkthrough of six distinct Bitcoin clients and tools to identify usability issues while performing basic Bitcoin tasks (*e.g.,* viewing account balance, sending funds, *etc.*). We find 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.

This chapter is a longer version of the published paper [EBSC15].

3.2   Background

3.2.1   Bitcoin

One subtlety of Bitcoin's transaction architecture is that, in order to spend Bitcoins, the entire value of unspent outputs (*i.e.,* from previous transactions) must be spent. To accommodate this, Bitcoin clients automatically spend the full amount of unspent outputs and create multiple components in the transaction: one component will send part of the unspent coins to the intended recipient, and the other component will send the remaining inputs back to the sender as *change.* It is technically possible (and some clients behave this way) to send change back to the sending address.

However, to enhance anonymity, the reference client generates fresh addresses (and corresponding private keys) to receive the remaining transaction amount.

As more transactions are made, Bitcoin clients must keep track of multiple private keys for use in future transactions. Many clients prominently display a Bitcoin balance on the main screen, which represents the sum of all unspent outputs for which private keys are available.

One subtlety of Bitcoin transactions is that each XBT amount in the set of inputs must reference past transactions where the address received adequate XBT to cover the input amount. The complexity here is that a transaction must completely spend the amount received in these past transactions even if it is larger than the amount in the current transaction. To enable this, transactions will output the surplus XBT back to the sender as 'change.' Change can be sent back to the same input address, or to enhance privacy, it can be sent to a brand new address created by the sender's client (called 'change addresses').

3.2.2 Usability of Key Management

Passwords remain the most common form of user authentication [HvO12]. Private key-based authentication is rarely used by non-experts, and is typically never used as the default configuration in applications which support this authentication method. Transport Layer Security (TLS) client-side certificates have failed to reached widespread deployment. Secure shell (SSH) uses passwords by default, and recent efforts to reintegrate them in a different form (*e.g.,* origin-bound certificates [DCBW12])

still rely on passwords as the primary authentication mechanism.

Password managers, when configured to generate or store system-chosen random passwords, share at least one property of cryptographic keys: such passwords become something you *have* instead of *know.* However, if access to such a password is lost, online services generally offer account recovery mechanisms (*e.g.,* based on email). No such recovery mechanism exists for self-managed cryptographic keys.

The use of public key systems by non-experts that is closest to Bitcoin is arguably encrypted/authenticated email, in particular Pretty Good Privacy (PGP) and its open-source alternatives (*i.e.,* GPG and OpenPGP). Beginning with *Why Johnny Can't Encrypt* [WT99], the usability of public key technology has been well-studied from a usability perspective [GM05, GMS+05, SBKH06, GFFK06]. The findings of this literature are diverse but relevant observations include the following: (1) the metaphor and terminology behind public and private keys is confusing; (2) it is difficult to correctly obtain other users' public keys; (3) key migration between devices is difficult. This literature tends to focus primarily on encryption and not signatures, but we find some overlap to the work presented here [2].

3.3   Bitcoin Key Management Approaches

Before turning to a detailed usability evaluation, we evaluate from a systems perspective each category of tool for managing Bitcoin private keys. We highlight security

---

[2] "Why King George III can encrypt," *Freedom to Tinker* (blog), 6/6/2014.

and deployability issues, and note relevant details of the Bitcoin protocol that create complexities and potential discrepancies with users' mental models.

3.3.1   Keys in Local Storage

One way in which Bitcoin software manages several private keys is by storing these keys on the device's local storage, typically in a file or database in a pre-configured file system path. When a new transaction is created, the Bitcoin client can read the keys and immediately (possibly without any further user input) broadcast the transaction over the network. The reference Bitcoin client (Bitcoin Core), as well as certain mobile wallets (*e.g.,* Android Bitcoin Wallet) use this approach, storing private keys in a file (referred to as a *wallet*) inside the user's home or application directory.

Storing keys in a locally accessible file has several advantages. First, there is no additional cognitive load on users, since only the software must access the file. Second, a practically unlimited number of keys can be stored on disk due to the small size of keys. Third, the Bitcoin software can automatically generate keys and create transactions without additional input or actions from the user.

Storing keys locally also creates several threats, which the user must consider. For example, the file storing private keys can be read by any application with access to the user's application folder. Malware authors may be particularly interested in exploiting this key management approach, since access to the local file results in the adversary gaining immediate access to the victim's funds. One of the first examples of

private key-stealing malware was discovered by Symantec in 2011 [Sym], with many other similar malware examples following suit.

Users must be cautious to not inadvertently share their Bitcoin application folder (*e.g.,* through peer-to-peer file sharing networks, off-site backups or on a shared network drive). Physical theft, especially in the case of portable computers or smart-phones must also be considered. Similar to the storage of other sensitive files, threats to digital preservation [BKM05] should be taken into account. Examples include general equipment failure due to natural disasters and electrical failures; acts of war; mistaken erasure (*e.g.,* formatting the wrong drive or deleting the wrong folder); bit rot (*i.e.,* undetected storage failure); and possibly others. If storing private keys for a long period of time (*e.g.,* a trust fund or long-term savings), users must also preserve a specification of the file format to ensure the keys can continue to be read.

The reference Bitcoin client pre-generates keys in a batch of 100 (these keys are known as the keypool). When a transaction is made, the next available key is selected from the keypool for receiving change. The keypool is then periodically refilled with a new batch of keys as necessary. This *key churn* requires users to periodically create new backups of their key storage file to ensure that new keypool keys are stored.

The user must also be wary of *key churn* as the Bitcoin Core client sends change to new addresses. By default, it creates private keys in batches of 100 (called a keypool). This has the unfortunate side-effect that backups become obsolete after the user churns through their current keypool. The user interface of Bitcoin Core does not display change addresses or give any indication that they are being used,

and so it is quite natural that a novice users' mental model will not account for this behaviour, and they will not act accordingly to ensure they re-backup `wallet.dat` each time they deplete the keypool (another event that is not communicated to the user in any way). To address key churn, alternative Bitcoin clients return all change to the same address or derive all change addresses, called a *deterministic wallet*, from a single key.

Another disadvantage of using Bitcoin Core is that it requires a copy of the entire blockchain to validate the balance associated with each of the keys it will create. At the time of writing, the blockchain is 35 GB.[3] For a new installation, it is not uncommon for it to take days to obtain a local copy of blockchain from the Bitcoin peer-to-peer network.

3.3.2 Password-protected (Encrypted) Wallets

Certain Bitcoin clients allow a locally stored wallet file to be encrypted with a key derived from a user-chosen password or passphrase. Password-protected wallets appear to address only *physical* theft of the underlying storage device, requiring brute-force of the password if the file containing private keys is stolen. Password protection seems less useful in the case of *digital* theft; if malware can be installed on to the device storing the wallet, it is reasonable to assume a keystroke-logging module would be present, limiting or nullifying the benefits of the password protection.

Password-protected wallets share the advantages and disadvantages of non-encrypted

---

[3]`https://blockchain.info/charts/blocks-size`

wallets (see Section 3.3.1), with a few subtle differences. Password-protected wallets trade recoverability and usability for the mitigation of physical theft. If the password is forgotten, users lose the balance of their password-protected wallet since no mechanism exists for recovery[4]. For day-to-day use, users must unlock the wallet by entering their password when new transactions are made.

The trade-off of a password-protected wallet is that users can lose their XBT by forgetting the password protecting their wallet. No recovery mechanism exists (as this mechanism could itself be exploited in the case of theft) short of exhaustive search, which is an available service.[5]

Password-protected wallets may mislead the user to believe that the password itself provides access to their funds regardless of the location of the device storing the wallet, as would be congruent with a traditional mental model for web-based online banking. Users may be surprised to discover that they cannot access their funds at a new device by simply entering their encryption password; the wallet file must also be transferred to the new device.

3.3.3   Offline Storage of Keys

To further protect Bitcoin private keys from malware-based threats, wallets can be stored offline on some form of portable media, such as a USB thumbdrive. Keeping

---

[4]Of course, exhaustive search of the password space is theoretically possible, and is available as a service: `http://www.walletrecoveryservices.com`

[5]`http://www.walletrecoveryservices.com`

keys offline enables the use of traditional physical security techniques (*e.g.,* storing the drive in a fire-proof safe) to protect the wallet. However, offline storage has the drawback of making the wallet inaccessible for immediate use by software, preventing users from spending funds unless the offline storage media is nearby. As expected, offline storage can be used for backup, but all copies of the wallet must be kept offline for the full benefits of theft-protection to be realized. Prior to offline storage (wallet creation) and after storage (future transactions), the wallet will be exposed on a computational device, potentially to malware.

Figure 3.1: Offline storage - Paper Wallet

An interesting case of offline key storage is paper wallets (see 3.1) where private keys are printed onto paper typically in the form of a 2D barcode (*e.g.,* a QR code) or as a long sequence of characters. Barcodes facilitate reading the key back into a Bitcoin client by, for example, scanning the code with a smartphone camera. Securing a paper wallet is similar to securing cash, which most users should be comfortable with. However, funds can be stolen from a paper wallet by simply observing the QR code

(*e.g.,* on live television[6]), which is not possible with physical money. Thus transporting a paper wallet securely requires that the printed contents remain unobservable at all times. Users must remember that a paper wallet does not contain the funds itself, but rather enables signing authority over a set of Bitcoins. For example, if a paper wallet is discarded after funds are spent, the paper wallet still provides access to any future funds that may be sent to that address.[7]

Finally, users still need to be cautious of key churn and that spending XBT from a paper wallet does not result in XBT being sent to a change address not included in the paper wallet.

As with any long-term storage, users must preserve software capable of decoding the QR code in the event that the paper wallet generation service is unavailable when attempting to reload keys onto a device. As of writing, many Bitcoin clients as well as offline storage solution use a common "wallet import format", which involves manipulating an ECDSA private key by performing cryptographic hashes, adding a checksum for integrity, and encoding the resulting string into Base58.[8]

---

[6] "A Bloomberg TV Host Gifted Bitcoin On Air And It Immediately Got Stolen," *Business Insider*, 10/23/2013.

[7] "Five Ways to Lose Money with Bitcoin Change Addresses," *Bitzuma* (Blog), 17/03/2014.

[8] Base58 avoids the use of characters such as "0, O, I, and l" which may look visually similar, and also avoids punctuation characters which may trigger software (*e.g.,* e-mail clients) to perform line breaks.

3.3.4 Air-gapped Key Storage

In offline storage, we assume the device or media holding private keys cannot perform computations such as creating digital signatures. We distinguish this type of storage from air-gapped storage, where wallets are stored on a secondary device that generates, signs, and exports transactions, but this secondary device is never connected to a network. When spending Bitcoins using an air-gapped device, a transaction is created from the air-gapped device and the resulting signed output transported (usually through portable media) to an Internet-enabled device for transmission onto the Bitcoin network.

An air gap improves theft-resistance by never directly using a private key on an Internet-connected device. However, air gapped devices are capable of actually executing malware if infected. Malware may jump the air gap by infecting the portable media used to export signed transactions.

While not literally an air gap, hardware security modules (HSMs) emulate the properties of an air gap by isolating the key material from the host device, and only exposing the ability to sign transactions. Bitcoin-specific HSMs are under active development at the time of writing and a few have been recently released (*e.g.,* Trezor[9]).

Note that the consequences of obtaining access to the private keys are not much different from accessing a transaction-signing oracle for the wallet—both allow the current balance of Bitcoin to be stolen. However, future funds may be protected if

---

[9]`http://www.Bitcointrezor.com`

access to the signing oracle is non-persistent.

3.3.5   Password-derived Keys

Thus far, all key management solutions have required users to maintain cryptographic keys. The remaining two solutions enable users to access their Bitcoin with a password instead.

The first approach is to derive cryptographic keys from a user-chosen password (*e.g.,* using PBKDF2 [RSA], manipulating the output to produce a valid Bitcoin private key). The disadvantage of using this approach directly is that only one resulting keypair is created, requiring the user to select a new (different) password for a new keypair.

A more robust approach is described in the Bitcoin Improvement Proposal 32 [Pieb], and is known as a Hierarchical Deterministic (HD) Wallet. HD wallets deterministically derive a set of private keys from a master secret (a randomly chosen passphrase). These keys can derive new private keys. The deterministic nature allows the password holder to view the balance, as well as spend the funds, of any sub-account derived from the password. However, if the private key on one of the sub-accounts is compromised, only the funds sent to that sub-key (or sub-keys derived from it) may be stolen.

Password-derived wallets are targeted at loss-prevention and simpler cross-device access. The challenges of preserving access to a digital file are no longer necessary as long as the wallet can be re-generated from a memorized password. The pri-

mary drawback of a password-derived wallet is that weak user-chosen passwords can be found through unthrottled exhaustive search since a fingerprint of the associated public key will be in the global public ledger if the account holds any amount of Bitcoin. Rainbow tables [Oec03] for password-derived keys have been developed.[10] Finally, it remains unclear whether memorization poses an advantage over maintaining a digital file when preventing loss—a forgotten password will orphan all funds in the account.

3.3.6 Hosted Wallets

A final approach to key management is to host user accounts on a third-party web service. In this case, the service maintains possession of the private keys. Hosted wallet web services provide the user with access to transactional functionalities through standard web authentication mechanisms, such as a password or two-factor authentication, and may also offer password recovery mechanisms. Bitcoin smartphone applications that act as clients to hosted wallets benefit from reduced application complexity (*i.e.,* no need to perform cryptographic operations on the device) and brick and mortar bank-like user interfaces. Currency exchange services that allow Bitcoin to be exchanged with fiat currency effectively provide this service, as do web services deployed specifically to host wallets.

The popularity of hosted wallets appears to be justified, since these services pro-

---

[10]D. Martyn. "Bitcoin 'Brainwallets' and why they are a bad idea," *Insecurety Research* (sic) (Blog), 26 Mar 2013.

vide the closest experience to traditional online banking. However, their use has also been hampered by high profile breaches and fraud. Users' funds have been unrecovered from services such as Mt.Gox and Bitcoinica, while popular exchanges such as BTC-E have suffered losses but fully reimbursed users. Thefts and losses from/by third party services are catalogued online[11] and include over 40 events involving losses greater than 1000 XBT.[12]

It is natural to expect hosted wallet services will become primary targets of attack since these services typically hold large amounts of Bitcoin. Offloading the task of key management to a third-party requires users to assume the risk that the service could be breached and funds lost, in exchange for a traditional online banking-style user experience.

As a counter-measure to theft, hosted wallet providers often keep only a small float of their holdings online (called *hot storage*) and store the majority of their holdings offline in *cold storage*. This has the drawback of causing delays in transactions for users if the hot storage amount is exhausted. Hosted wallet services may also allow audits, where they cryptographically prove possession of sufficient Bitcoin to match their liabilities.

Another approach that falls under the hosted wallet category is a hybrid hosted wallet. Hybrid wallets use client side encryption (typically in Javascript) to encrypt all private keys and sensitive data. The web service is then only used for broadcasting

---

[11]https://Bitcointalk.org/index.php?topic=576337.0

[12]At the time of writing, 1000 XBT ¿ 650 000 USD.

transactions to the network and for displaying the user's balance (which requires inspecting the entire blockchain).

Other than server side encryption and security measures, It uses client side encryption (javascript) to encrypt all the private keys and sensitive data with user's password and sends the encrypted data as a random base64 string to the server. With this implementation, there is no access to the private keys and the final balance from anyone whom have access to the server's data [13]. Blockchain.info uses this implementation for its hosted wallet.

3.4   Evaluation Framework

In this section, we systematize the major category-wide issues we have uncovered in describing the various key management approaches used by Bitcoin clients. We present an evaluation framework based on 10 criteria as shown in Table 3.1 and discussed in the following subsections. This framework both summarizes the advantages and disadvantages of the various approaches we have evaluated, while also providing a benchmark for evaluating future key management proposals. The framework is adapted from a similar framework for evaluating password replacement schemes [BHvOS12].

---

[13]`http://Bitcoin.stackexchange.com/questions/5249/how-secure-is-blockchain-info`

Category	Example	Malware Resistant	Key(s) Kept Offline	No Trusted Third Party	Resistant to Physical Theft	Resistant to Physical Observation	Resilient to Password Loss	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		●	●		●	●	●	●	●	●

Table 3.1: A comparison of key management techniques for Bitcoin (contrasted with traditional financial services). ● indicates the category of client is awarded the benefit in the corresponding column. ○ partially awards the benefit. Details provided inline.

3.4.1 Evaluation Criteria

We briefly enumerate the criteria used to evaluate each proposal in the framework below.

Malware Resistant. Malware designed to steal Bitcoin wallets and related passwords has been observed in the wild. Wallets that are not stored on an Internet-connected device, or devices capable of performing computations are considered malware resistant (●), unless creating a transaction involves transferring to a computational device (○).

Key Stored Offline. For archival storage of infrequently used keys, keys not directly accessible from an Internet-connected device—either due to being offline (●) or online but password-protected (○)—are preferable.

No Trusted Third Party. All Bitcoin key management tools are trusted to a certain extent. This criteria considers the absence of a persistent trusted third party (●) that maintains direct signing authority over a user's Bitcoin.

Resistant to Physical Theft. If the cryptographic keys are stored on some media or device that can be physically stolen, we do not consider the tool to be resistant to physical theft. Within our framework, the only tools meeting this requirement rely on a human memorized password being necessary for key recovery. These are awarded (○) since passwords tend to be weak and may not adequately resist unthrottled guessing.

Resistant to Physical Observation. Physical observation, such as observing key strokes or capturing QR codes with a camera, may result in access to a user's Bitcoin

account.

Resilient to Password Loss. If passwords are used (○), the loss of a password could result in some Bitcoin becoming unrecoverable if it is a necessary authentication factor in obtaining access to the signing key. For solutions where funds are held by third parties, these entities could provide a password recovery/reset mechanism (●).

Resilient to Key Churn. Assuming the client sends change from transactions to a newly created change addresses, a tool is resilient to key churn if it can maintain access to the funds even after exhausting the initial keypool (●). Tools not awarded this benefit are not guaranteed to maintain persistent access to new change addresses, and any balance sent to these addresses may be lost.

Immediate Access. Key management mechanisms that maintain direct access to the wallet enable Bitcoin to be transacted immediately (●). We award this benefit to techniques that require a user to enter a password. We omit the benefit for techniques that require data to be obtained from external storage medium or secondary device.

No New User Software. Some approaches require users to install new software on their system, for which the user may not have suitable permission, or software may not be developed for their specific platform (*e.g.,* some mobile platforms). By contrast, some tools can be executed from widely available software such as any standards-compliant web browser (●).

Cross-Device Portability. A key management technique is cross-device portable (●) if it allows easy sharing of the a Bitcoin address across multiple devices with

minimal configuration or usability issues due to complexities like key churn.

3.4.2 Discussion

Table 3.1 demonstrates that key management approaches provide varying levels of security and convenience, with no single approach being obviously superior to others. One possible takeaway from our evaluation and comparison is that users can benefit heavily by offloading key management to a trusted party (*e.g.,* hosted wallets). The lower right side of the chart focuses on usability properties that are already present in traditional financial services (*i.e.,* resilient to password loss, no new software, cross-device portability). These properties are difficult to obtain if users independently manage their keys through one of the local storage techniques. Of course, the disadvantage of trusting a third party is that Bitcoin funds are now bound by a contractual agreement between users and the hosted wallet provider, negating one of the primary features of Bitcoin: a fully decentralized currency. Users in countries lacking regulatory maturity for digital currencies should exercise caution when trusting a third party with large amounts of Bitcoin.

Based on our analysis, users can be given the concrete advice of treating digital currency much like they would treat fiat currency: keeping small amounts in ready-to-spend form (*e.g.,* local storage or online hosted walled) mimicking cash, and keeping larger sums in more difficult to access but more secure storage (*e.g.,* air-gapped or offline storage) mimicking a savings account or trust fund. Barber *et al.* [BBSU12] suggest the use of "super wallets" where users essentially run their own personal bank.

A super-wallet keeps keys across multiple devices and requires all (or a subset using a threshold scheme) to be present to transfer funds to sub-wallets. Pre-configured transfers of small amounts can be authorized to move funds to sub-wallets that can be used for day-to-day spending. While the idea of super-wallets is intuitive, the implementation of such a scheme could introduce high levels of complexity.

3.5 Usability Evaluation of Bitcoin Clients

3.5.1 Methodology

We used a series of cognitive walkthroughs [WRLP94] to evaluate the usability of six Bitcoin clients. Cognitive walkthrough is a form of expert evaluation where an expert (or group of experts) steps through the design to evaluate aspects of its usability. The focus of the walkthrough is on the novice user and emphasizes *learnability* which is defined in ISO/IEC 25010 as the "degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use" [SQu]. At each step, the evaluators ask three questions: Will the user see what to do? Will the user see how to do it? And once it is done, will the user know if they have performed the correct action? If the answer to any of these is no, the evaluator will record the violation. When using multiple evaluators, the aggregated result will be the union of violations found by each evaluator.

We chose to use cognitive walkthroughs for several reasons. First, it allowed us to

choose and compare standard tasks on disparate tools, and gave us easily compared insight into the common problems and successes of different Bitcoin clients. The cognitive walkthrough also allowed us to keep the focus on the novice user. The goal of our evaluation was to uncover problems specific to key management within Bitcoin software rather than to evaluate the usability of the clients themselves.

A number of usability evaluation methodologies employ expert review. We use a cognitive walkthrough [WRLP94], which has been used previously to study closely-related subjects: public key technology [WT99] and software configuration [CvOA07]. A cognitive walkthrough is premised on the idea that users learn through exploration of the software, instead of reading manuals. They attempt to perform the task they want completed and rely on the interface to intuitively guide them through proper design, interface cues, and feedback.

For our cognitive walkthrough, we defined a set of core tasks involving key management that a typical user needs to perform. We compared the results of each walkthrough against a standard set of evaluation guidelines, combining aspects of an heuristic evaluation [Nie92] with the walkthrough in order to interpret our results.

Each of the following four tasks was independently performed by 2 experts to evaluate each tool:

T1 Configure a new Bitcoin address and obtain its balance. This task involves launching the Bitcoin client (or logging into one if hosted online) for the first time. After a new address has been generated (either explicitly or transparently

in the background), the user should be confident that the address' balance is XBT 0.00000000. The user should also be able to find their receiving Bitcoin address.

T2 Spend Bitcoin. Send some amount of Bitcoin to an arbitrary (but valid) Bitcoin address. This task requires the user to create a new transaction, entering relevant information such as recipient, amount, *etc.*

T3 Spend Bitcoin from the same address as above, but on a secondary device. This task may require copying private keys to the secondary device, entering passwords on multiple devices, or logging in to a hosted wallet provider on a different browser.

T4 Recover from the loss of the main credential. In the case of locally stored keys, this task involves restoring a file from backup. Otherwise this task involves recovering from password loss.

Since the focus of our walkthrough was on configuration and learnability, we used a set of heuristics first developed for a usability evaluation of Tor [CvOA07]. We chose to use these guidelines because like the anonymity software, successfully managing Bitcoin involves the application of complex cryptographic knowledge in an everyday activity. The set of guidelines, from [CvOA07], are:

G1 Users should be aware of the steps they have to perform to complete a core task.

G2 Users should be able to determine how to perform these steps.

G3 Users should know when they have successfully completed a core task.

G4 Users should be able to recognize, diagnose, and recover from non-critical errors.

G5 Users should not make dangerous errors from which they cannot recover.

G6 Users should be comfortable with the terminology used in any interface dialogues or documentation.

G7 Users should be sufficiently comfortable with the interface to continue using it.

G8 Users should be aware of the application's status at all times.

Cognitive walkthroughs are primarily relied on when the breadth of the evaluation makes a user or field study prohibitive to run due to time and cost. We examine six Bitcoin key managers, from configuration through transaction authorization through key recovery. If the results of the cognitive walkthrough narrows the field significantly, user studies are an appropriate follow-up for detailed examination of the most challenging set of tasks within one or two solutions. Thus while our result can be considered a first-pass at the problem, we felt the richness of the result merits sole presentation.

3.5.2 Evaluated Clients

Real-world evaluation of the general approaches detailed in Section 3.3 is difficult. Thus, we select six distinct Bitcoin clients or utilities that implement the key management approaches described. For the purposes of our usability evaluation, each client was evaluated in its default configuration on OS X unless otherwise stated.

Keys in Local Storage. The reference Bitcoin client, Bitcoin Core [Bit], is a cross-platform client that stores keys locally (optionally encrypted with a password). Bitcoin Core is the first recommended client on the `Bitcoin.org` website.

Password-protected (Encrypted) Wallet. We use the MultiBit [Mul] client (also recommended on `Bitcoin.org`) since it provides a more convenient way to encrypt with a user-chosen password.

Offline Storage. We use paper wallets as offline storage. While paper wallets can be as simple as printing private keys on to paper, we select the paper wallet creation website `Bitaddress.org` [Poi]. Bitaddress allows users to generate new randomized keys in their web browsers, and then print QR encoded keys.

Air-gapped Storage. We select the Bitcoin Armory [Arm] client which includes functionality for creating an offline wallet that can be used to sign and export transactions.

Password-derived Keys. One of the simplest ways to create a password-derived key is on the Brainwallet [bra] website. The site allows users to enter a passphrase which is converted into a private key.

Hosted Wallets. We use `Blockchain.info` [Blo] as our hosted wallet provider. As of writing, Blockchain.info advertises the management of over 2.5 million user wallets.

3.6 Results

The following is the full details of our walkthroughs, which expands on the shorter version presented in the conference version of this paper.

3.6.1 Keys in Local Storage (Bitcoin Core)

We begin with an evaluation of Bitcoin Core, the original Bitcoin wallet client, which uses locally-stored keys. We assume the user has downloaded and installed the Bitcoin Core client (it has a straight forward wizard installation procedure).

T1: Configure. Bitcoin Core transparently generates a new set of addresses on first run, but shows no notification to the user that this has occurred (fails G3). The receiving address can be found under the *Receive coins* tab, but this could be easily confused with the *Addresses* tab which contains a contact list of other user addresses (fails G2).

To retrieve the account balance, Bitcoin Core must be online and the user must wait until a full copy of the blockchain has been downloaded. Except for a small status indicator on the bottom-right side of the window that shows a small red cross in-between two black windows, there are no other messages to show the user that the application should be online. Due to the size of the blockchain, this may take hours to days to complete. A status bar displaying "Synchronizing with network" shows the progress of the blockchain download (achieves G8), but the terminology may be too technical for novice users, With a mouse over the icon, it says '0 active connections

43

to Bitcoin network' which is likely unfamiliar language that does not help resolve the error (fails G4 and G6). Once the blockchain has been downloaded, the balance is displayed on the *Overview* tab (achieves G3).

T2: Spend. Spending Bitcoin is straightforward since the keys are readily available to the Bitcoin Core client. Users spend Bitcoin by navigating to the *Spend* tab (achieves G1 and G2). Since our focus is on key management, we do not evaluate the actual completion of transactions (which may have additional usability issues). We focus on ensuring the key is available to the software tool (which is not so straightforward with *e.g.,* offline storage).

T3: Spend from Secondary Device. Installing Bitcoin Core on a secondary device creates a new set of keys. Users may not understand that the keys must be copied to the secondary device (fails G1), and if so, what file must be copied (fails G2). The correct procedure is to back up the `wallet.dat` with the 'backup wallet...' option in the 'File' tab of the first installation and chose a directory to save the `wallet.dat`. Next the user must securely transfer this file to the secondary device, and no guidance is provided on how to do this (fails G2) or the dangers of transferring it through an insecure mechanism (fails G5).

Assuming the user has transferred `wallet.dat` to the secondary device, she could try looking for import options in the newly installed wallet client, or drag and drop the `wallet.dat` into the client, but she would fail to do so as no import option exists. The documentation is inadequate here as well—there is actually nothing in the help menu except a debug window that is for advance user to tweak the application (fails

44

G2 and G6)!

The only mechanism to activate the wallet on a secondary file is to actually over-write `wallet.dat` on the secondary device with `wallet.dat` from the first. It is unlikely any novice user would be able to complete this step. It is actually even difficult to find the path to copy `wallet.dat` to on the new device—this could be possible by searching the local file system for `wallet.dat`, which might not succeed due to non-searchable system reserved folders or not knowing the exact file name (spotlight does not return any result for `wallet.dat`). More likely, the user will search online.[14] On OS X, the path is `/Users/User/Library/Application Support/Bitcoin/wallet.dat`.

The next step is to replace the new `wallet.dat` with the one from the primary device. It should be noted that the name of the file should be exactly `wallet.dat` for the Bitcoin Core to be able to read the file. Some of errors that the user might encounter during this procedure are:

- The user might copy `wallet.dat` from the primary device wallet client path instead of the one exported through the back up option. This could cause a corrupted `wallet.dat` that is not readable by the secondary device's Bitcoin Core. This is due to Bitcoin Core's procedure to lock `wallet.dat` while it is in use. The error is recoverable by repeating the process correctly (fails G4).

- User should wait for the Bitcoin Core on the secondary device, to download and

---

[14]`https://en.Bitcoin.it/wiki/Data_directory`

sync the Blockchain from the P2P network to be able to authorize a transaction.

- On the secondary device, the final balance might be wrong and there would be the need to resynchronize and rescan the blockchain to have the correct final balance (fails G3).

Finally, this process must be repeated if either client exhausts their keypool. If both do, there is no way to merge the new keys in the keypool, and replacing one `wallet.dat` with the other will lead to unrecoverable funds (fails G5).

We note that replacing the key file may require a re-scan of the blockchain to display the correct balance (fails G3).

T4: Recovery. If only one device is used, there is no way to recover from loss of the key file (*e.g.,* due to a disk failure, file corruption, or loss of the device itself; fails G5). If the user backed-up the key file, the process for recovering from loss is equivalent to that of T3 above.

3.6.2   Password-protected Wallets (MultiBit)

Although it is possible to encrypt the `wallet.dat` with a password in Bitcoin Core, it is not the default option nor is there any cue to do so. Instead we evaluate the MultiBit client, where one of the recommended first steps is to password protect the wallet file. MultiBit is a popular client in particular for its use of SPV[15] for lightweight blockchain validation that can complete within minutes instead of, relative to Bitcoin

---
[15]Simplified Payment Verification [Nak08]

Core, days.

T1: Configure. On first run, a welcome page contains an explanation of common tasks that can be performed with MultiBit—where the send, request and transaction tabs are and how to password protect the wallet file (achieves G1 and G2). The client provides help options for other functionalities with direct and non-technical guides (achieves G6).

MultiBit automatically generates a new receiving address on first run, but does not notify the user (fails G3). Reading the newly generated address requires navigation to the *Request* tab, which displays "Your address" as well as a copy of the QR code of address.(partially achieves G2).

The interface shows the status of the program (online, offline, or out of sync) on the bottom left status bar, the balance of the user's wallet on the upper left, and the latest price of Bitcoin on the upper right of the window (achieves G8). The interface seems to minimize jargon and technical vocabulary (achieves G6). As it is mentioned on the *welcome page* of the application, every option in MultiBit has the ability to show help tips by hovering the mouse over that option (achieves G6 and G7).

The displayed balance is not necessarily current until synchronization is completed, however there is no direct cue in the balance area indicating this (achieves G8).

T2: Spend. The user must navigate to the tab labeled *Send*, as instructed on the welcome screen (achieves G1 and G2). If the client is not synced, the send button is disabled (achieves G4). If it is synced, the user fills out the transaction details,

destination address and amount and clicks send. The client prompts the user for the decryption password (achieves G2). An incorrect password displays the error 'The wallet password is incorrect' but otherwise allows immediate and unlimited additional attempts. Entering the correct password authorizes the transaction (achieves G3).

T3: Spend from Secondary Device. On the primary device, the user must navigate to the *Options* menu, and select *Export private keys* under tools (fails G1 and G2). The interface displays a wizard requesting an export password as well as a file system path for the exported file to be saved. If the user attempts to save the exported file without password, a warning is displayed in red: 'Anyone who can read your export file can spend your Bitcoin' (achieves G5). By having a password-protected file exported, the user can securely copy the file to the secondary device with some protection against interception. After clicking to export, wallet file is saved in the given path and the client checks that the file is readable (achieves G4 and G5).

On the secondary device, the user must select *Import private keys* from the *Options* menu. After selecting the previously exported file, the wizard confirms the completion of the import (achieves G4) and the balance is updated to reflect the newly imported keys. The user can proceed to create a new transaction as in T2. MultiBit sends change to the originating address, so keypool churn is not an issue.

T4: Recovery.. As with Bitcoin Core, recovery is not possible if no backup of the wallet file was made. Creating a backup and importing it follows the same procedure as T3. As expected, both the password and the backed up wallet are necessary for recovery.

3.6.3 Air-gapped Key Storage (Armory)

Bitcoin Armory is an advanced Bitcoin wallet that allows the wallet to be stored and managed on an offline device, while supporting the execution of a transaction through an online device. Armory is also used on the online device to obtain the blockchain and broadcast the transaction created on the offline device. It is possible to use some other online application to implement the airgap, however this is the recommend method and the one we will consider.

T1: Configure. The user begins by installing Bitcoin Armory on the offline computer. On the start, the welcome page offers the option to 'Import Existing Wallet' and 'Create Your First Wallet!' (achieves G2). The user creates her wallet, with passphrase-protection being a mandatory option. Armory asks to verify the passphrase and warns the user not to forget her passphrase (achieves G5). After this step, a backup window pops up with the options to print a paper wallet or save a digital backup of the wallet, and also warns the user if he decides not to backup his wallet (achieves G5). After proceeding, the user must click on 'Receive Bitcoins' to prompt the client to generate the Bitcoin address in the wallet file (fails G3,G4). By contrast, most clients do this step automatically on launch.

In order to see the balance of the account, the device must be online and synced (fails G2). Thus the user must use the online device, not the offline device, to check her balance. Users can click on the 'Offline Transaction' button, which offers a short documentation of the steps to be taken to sign a transaction and in doing

so, explains the offline/online distinction relevant to checking a balance. Within the 'Wallet' window, there is an option to 'Create Watching-Only Copy.' The language is difficult (fails G6): this option allows a copy of only the Bitcoin addresses to be exported, not the private keys, for use on the online computer to display an updated balance for each address. The exported file can be copied to the online computer.

We assume the user has installed Armory on the online computer. It should be noted that Armory only works side-by-side with Bitcoin Core and uses Bitcoin Core to synchronize and read the downloaded blockchain (fails G1). A pop-up window will alert the user when 'the blockchain' has been downloaded (partially achieves G3). Armory displays a 'Connected' cue in green in the bottom-right when it connects to Bitcoin Core. Upon launching Armory, the user should click on 'Import Existing Wallet' and she is prompted to import either a digital backup or watch-only wallet. She should chose the watch-only back up file that has been copied from the offline computer. After the application is done syncing, the balance is displayed on the main window under 'available wallets' (achieves G8).

T2 & T3: Spend. With an air gap, the distinction between primary and secondary devices is less clear given that the basic setup itself includes two devices: one online and one offline, but authorization of transactions uses the offline device. To authorize a transaction, the user may begin from Armory on the online or offline device (may not fully achieve G2). On the either device, the user should click on 'Offline Transactions' in the main window which displays a very detailed description of the steps involved (achieves G1, G2, and G6). On the online computer, the user

clicks the option: *Create New Offline Transaction.* The user will be asked to enter the transaction details to generate an unsigned transaction as a file. The user must transfer this file to the offline computer. As mentioned in this step's documentation, the unsigned transaction data has no private data (the exact data will ultimately be added to the public blockchain) and no harm can be done by an attacker who captures this file (achieves G5) other than learning the transaction is being prepared.

On the offline computer, the user clicks on *Offline Transactions* and then *Sign Offline Transaction* which prompts the user for the unsigned transaction data file. Armory asks the user to review all the transaction information, such as the amount and the receiving addresses (achieves G5). By clicking on the *sign* button signed transaction data can be saved to a file. Text at the top of the window describes the current state of the file (signed) and what must be done (move to online device) to complete the transaction (achieves G1 and G2).

The signed file should be transferred to the online computer and be loaded through the same offline transaction window. When a signed transaction is detected, the *Broadcast* button becomes clickable. By clicking on broadcast, the user can once more review transaction details, and receive confirmation that the Bitcoins have been sent (achieves G3 and G8).

T4: Recovery. Like Bitcoin Core and MultiBit, Armory requires a backup of the wallet to have been made. Without this backup, recovery is impossible. Armory encourages backups at many stages (achieves G1 and G2). Armory provides many prompts for the user to back up her wallet keys. At the time of creating the wallet,

there are multiple windows and alerts conveying the importance of a back up, with options for digital and paper copies. Even if user decides not to back up her wallet at this stage, she is provided a persistent 'Backup This Wallet' option (achieves G4). In the backup window, there are a number of options to back up: a digital copy, paper copy, and others. By clicking on the 'Make Paper Backup' for example, the paper backup is shown to the user containing a root key that consist of 18 sets of 4-characters words and a QR code. To restore the paper wallet backup, on the main page, the user can click on 'Import or Restore Wallet' and select 'Single-sheet Backup' option. She will be prompted to input the Root Key from the paper wallet. The 'Digital Backup' option provides an unencrypted version of the wallet file that can be securely stored on portable media. Recovering from a lost wallet with a digital backup involves selecting 'Import Digital Backup or watch-only wallet' from the 'Import or Restore Wallet' window as explained in core task 1. Armory also enables the user to test the backups to ensure there is no error in the backup file (achieves G5) through the same import procedure.

3.6.4   Offline Storage (Bitaddress)

There are different methods to use for offline storage of a Bitcoin wallet. For our evaluation, we consider the use of a paper wallet. Specifically we use the Bitaddress web-service, a popular Bitcoin paper wallet generator. Many paper wallet generators exist, however Bitaddress, at the time of writing, is the first search result for 'Bitcoin paper wallet' on Google.

T1: Configure. Upon visiting the `bitaddress.org`, the user is asked to move the mouse or enter random characters in a text box to generate a high-entropy random seed to be used to generate a private key associated with the Bitcoin address (achieves G1 and G2). Once enough entropy has been collected, the site redirects the user to a page that shows the receiving Bitcoin address and it's associated private key (achieves G3). The public key (Bitcoin address) is labeled *Share* in green text and the private key *Secret* in red (helping achieve G5). In general Bitaddress uses non-expert terminology and simple instructions (achieves G6). To ensure the web service does not retain a copy of the users' key (the generation is done client-side in Javascript), the user should complete the process offline.

After printing, the user has a Bitcoin receiving address and, as mentioned in the documentation, the balance can be checked through a Bitcoin Block Explorer[16], such as `blockchain.info`. The user uses this site to search for her Bitcoin address and checks her balance. Although it is documented that the private key must remain secret, a user may inadvertently expose the private key by placing the paper wallet where it can be observed or by searching the website for the private key instead of the public key.

T2 & T3: Spend. Since the keys are printed on paper, there is no difference between authorizing from a primary or secondary device so we collapse the analysis of core tasks 2 and 3.

To send funds from a Bitcoin address that has been stored on a paper wallet, as

---

[16]webservice that provides access to the blockchain

it is mentioned in the documentation, the user has to import her private key in one of the wallet clients available, such as Armory or the Blockchain.info hosted wallet discussed below. If the user inputs the private key address into a client that returns change to newly generated addresses, she must export these new addresses to a new paper wallet or she will lose the surplus when she removes the wallet from the client (fails G5). If the user fails to remove the wallet from the client, a second copy is maintained increasing her exposure to theft (but reducing her exposure to key loss).

The process to import a key from a paper wallet depends on the client. For Blockchain.info, after making an online account, the user navigates to the 'Import/Export' tab and uses the option 'Import Using Paper Wallet, Use your Webcam to scan a QR code from a paper wallet.' It is also possible to type in the private key in the 'Import Private Key' text field. After this step, the address now is hosted on the online wallet and is the same as core task 2 in the Hosted Wallet section (Section 3.6.6) below.

T4: Recovery. Loss of a paper wallet makes the funds unrecoverable (fails G5). Bitaddress prompts the user to acknowledge this fact (also mentioned in its short documentation) when creating a paper wallet (achieves G1).

3.6.5   Password-Derived Keys (Brainwallet)

The most popular and complete implementation of a deterministic wallet with password-derived keys, at the time of writing, is Brainwallet.

T1: Configure. The Brainwallet website displays by default a pre-generated address corresponding to an empty passphrase. The passphrase input field displays "Long original sentence that does not appear in any song or literature. Never use empty passphrase. (SHA256)", but there is no corresponding documentation explaining the purpose of the passphrase or how it relates to the generated key (fails G1, G2, G6). As characters are typed, a new key is generated. User may not notice that generation of keys is happening dynamically, possibly preventing the user from noticing that the task is complete (fails G3). The user should replace the default passphrase with her own, hopefully ensuring her passphrase is not a commonly used phrase or anything that could be brute forced by an offline dictionary attack[17] as this passphrase is sufficient to access the funds stored in the resulting Bitcoin address. On entering the desired passphrase, the public and private keys are displayed on the same page.

Once the address has been generated, retrieving the balance of that address requires the use of an external service, but no suggestions are provided on the site (fails G1 and G2). The interface does display a number of other fields (*e.g.,* additional encodings of the public key) which may not be meaningful to novice users (fails G6 and G7).

T2 & T3: Spend. Spending Bitcoins from a password-derived wallet requires the user to import the private key into another client. The user should experience similar usability challenges as those detailed in the Offline Storage client above.

T4: Recovery. Forgetting the password of a password-derived key leads to funds

---

[17] "Bitcoin Brainwallets and why they are a bad idea", *Insecurety Research* (blog), 3/26/2013.

becoming unrecoverable (fails G5). Users will typically return to the same website (*i.e.,* the Brainwallet website) to extract private keys, but this may not be possible if the site is inaccessible (fails G5).

3.6.6   Hosted Wallets (Blockchain.info)

A variety of online services offer online hosted wallet clients to users. We use the popular Blockchain.info webservice for our evaluation.

T1: Configure. The user navigates to the Blockchain.info site and creates a new wallet by providing an email address and a (min) 10 character password (achieves G1 and G2). A message warning the user about the importance of not forgetting the password is displayed during registration (achieves G5). Next, a *Wallet Recovery Mnemonic* is shown to the user as a backup in case the password is forgotten. The balance and address are immediately displayed (achieves G3).

T2 & T3: Spend. Hosted wallets are accessible from any web browser, so creating transactions from many devices is straightforward. The user logs in to the site, clicks *Send money* (achieves G1 and G2). After filling in the required fields, the user is informed that the Bitcoins have been sent (achieves G3). Some of Blockchain.info's error messages may be too technical for novice users. For example, *No free outputs to spend* is displayed when transactions are created without sufficient funds (fails G6).

T4: Recovery. To recover from a forgotten password, a wallet recovery mnemonic may be provided on the login page. By clicking the *Recover Wallet* button, the site

will ask for the mnemonic phrase and the email address send the new credentials (achieves G1 and G2). Another recovery option is to proactively make backups and import them in case recovery is needed. To do so, in the main wallet page, user has to click *Import/Export* and exporting either an encrypted or unencrypted backup. Unencrypted backups should be kept in a secure storage. There are different options for the unencrypted backup procedure that could confuse the user and might result in an unrecoverable backup (fails G5 and G6)—the back up is shown on a text field that the user has to copy and paste into a text file to be able to save it on her computer (fails G2, G3 and G7). To restore the backups, there is an 'Import Wallet' option.

3.7 Discussion[18]

3.7.1 Metaphors

Bitcoin naturally invites a metaphor to traditional currency. This metaphor is often used in the clients (*e.g.,* send coins, receive coins, wallet), but does not always support their usability. The coin metaphor fails in both of the ways that user interface metaphors traditionally fail [CKM87]: aspects of Bitcoin transactions do not easily fit the coin metaphor, and conversely, encourages users to overextend the metaphor. Both of these lead to confusion on the part of users.

One way in which the metaphor of physical coins fails is in the sending and receiving of Bitcoin. In the physical world, the same physical token is almost always

---

[18]This section is largely the work of the co-authors of the paper but is included for completeness.

used to represent the same unit of currency (*i.e.,* giving money to a friend involves handing them the coin). However, when Bitcoins are exchanged, the private key is not transferred along with the balance. Private keys remain in possession of the sender, and can be reused and associated with new coins at a later time.

Many of the evaluated clients use the word "Send" to describe authorizing (digitally signing) a transaction, and private keys are not mentioned in any of the evaluated clients at the moment of transaction. It may appear counter-intuitive that this is a bad thing, but never mentioning the existence of keys may cause further confusion. The password-protected wallets, (*e.g.,* Multibit) require the user to input their password, but do not clarify the reason for the password.

Addresses are another metaphor that relate to the issue of transacting. The evaluated clients use the word "Address" to refer to the public key associated with a private key held by some user. This seems to be a relatively successful metaphor: it emphasizes the public nature of the public key, and also divorces the user's perception of a relationship between the public and private keys. To momentarily extend the metaphor, a user is accustomed to the idea that they will need to share their address in order to receive an item. However, the private key is more akin to the key to their mailbox, and a user would never think that they should share their mailbox key in order to receive mail to an address.

Another pervasive metaphor in the evaluated clients is the Bitcoin "wallet", where the user's Bitcoins are stored. The wallet metaphor is deeply entrenched in the foundations of Bitcoin. The reference client, *Bitcoin Core*, stores private keys in a

file named wallet.dat and the *MultiBit* client invites users to "create your first wallet!" on first launch. The hosted clients also use the metaphor; Blockchain.info prominently shows a Wallet tab, under which users are invited to "Create My Free Wallet". The wallet metaphor is descriptive for users, but fails to encompass the complexity of a user's collection of keys. In reality, the Bitcoin wallet contains private keys, but the term wallet is used to describe both the file storing the private keys, and the main interface of Bitcoin clients (as in Blockchain.info). This main interface sometimes includes a variety of other information, such as transaction history, address book, currency exchange rates, *etc.*

3.7.2   Abstractions

Abstraction and automation are complex issues for security software. Often, security is too complex to be completely automated, and the problem cases are often punted to the user (*e.g.,* in the case of TLS certificates [BvOP$^+$09]).

On first run, all of the evaluated clients transparently generate keypairs without informing the user. This behaviour continues as new transactions are made, where clients generate new addresses with no user notification (*e.g.,* for receiving change). It is unclear how well this abstraction works: while users do not need to be burdened with the knowledge of each private key, there are still situations in which a user might need to manage those keys, and the abstraction prevents users from doing so. Recovery from key loss depends on the existence of an up-to-date backup. While backup sounds like a simple task, in many of the evaluated clients, it involves finding

59

the right menu (MultiBit), or the right file (Bitcoin Core). Some clients do prompt the user to back up their wallets (*e.g.,* Bitcoin Armory), but with the private keys so completely abstracted away, users may not even understand what they are backing up, or why. Key churn, and the consequent need for semi-regular backups complicate the issue even farther.

The abstractions made in Bitcoin clients are sometimes beneficial for users, such as in the case of displaying a user's balance. A user's Bitcoin balance is typically made up of many small amounts corresponding to many private keys. However, most of the evaluated clients abstract these balances into a single figure. This highlights a usability disadvantage of paper wallets – the user must manage these multiple balances manually, and there is no method of seeing an aggregate balance when multiple paper wallets are in use.

3.7.3   Technical Language and Content
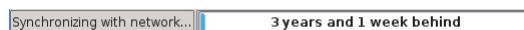
(a) Bitcoin Core
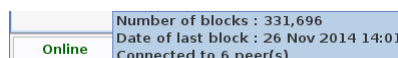
(b) MultiBit

(c) Armory

Figure 3.2: Screenshots of technical language displayed by two different clients.

When performing our evaluation, we identified multiple occurrences of highly

60

specialized or technical language used in the Bitcoin clients. These instances of technical language are confusing, particularly to novice users who are unlikely to be aware of either the jargon, and for whom the language will not help clarify the issues. The language itself highlights the complexity of the tasks associated with Bitcoin, and the difficulty of explaining them simply.

Examples of such language included messages in MultiBit and Bitcoin Core that referenced the client being "out of sync" or "synchronizing with network" (see Figure 3.2a) referring the process of downloading a full copy of the blockchain or retrieving relevant blocks from a trusted peer. A related message in MultiBit (Figure 3.2b) and Armory displayed the number of blocks that had been downloaded, as well as the number of connections to the Bitcoin network. These messages are intended to communicate that clients may benefit from faster transaction notifications when connected to more peers, but since peer connectivity is difficult for users to control, there is little benefit in communicating these ideas with the user. Similarly, the number of blocks independently has little significance to most tasks performed by an end user. We suggest that not only could this language be clarified, but that the interfaces could also streamline the amount of information that is presented to the user on every screen.

We also noticed that some clients used highly technical language when they could have used the metaphor to provide a simpler explanation to users. When attempting to authorize a new transaction on Blockchain.info with insufficient funds, the web interface displayed "no free outputs to spend". This error message is confusing, and

would be more easily understood if it referred to the lack of coins instead of the lack of outputs. Similarly, essential actions such as importing or exporting keys were often buried behind advanced or debug menus.

In the evaluated clients, there were often few resources to which users could turn for help. In the cognitive walkthroughs, the answer to the question "will the user know what to do?" was almost always unclear. Interface cues and features such as tool tips, wizards, or other contextual help were almost entirely lacking. Some actions were guided (*e.g.,* Multibit's prompted backups or create your first wallet), but many actions such as obtaining the balance of a paper or password-derived address were unsupported by help or documentation.

3.8   Conclusion

Bitcoin's usability limitations, particularly those related to key management, pose challenges to its rising popularity. In our evaluation, we found that developers in the Bitcoin ecosystem are making innovative attempts at solving the decades-old problem of usable key management. While some of these techniques seem promising, we find that tasks involving key management can be mired in complex metaphors and confusing abstractions.

Further investigation is needed to better understand and address these issues. A user study would give insight into exactly how these problems are affecting users and it would be interesting to investigate how expert users are (apparently successfully)

handling these challenges. Bitcoin presents a new opportunity for public key cryptography to become mainstream, and our evaluation is a first step towards achieving usable key management in decentralized crypto-currencies.

We covered the major categories of Bitcoin Wallet clients in this chapter. In the next chapter, we will try to evaluate each category for point of sale terminals for a small business to accept Bitcoin payments. It should be mentioned that the framework that was used in this chapter is specifically tailored for evaluating Bitcoin wallet clients and is not directly usable for other Bitcoin related software systems including point of sale terminals.

Chapter 4

Bitcoin Point of Sale Terminals: Evaluation and Deployment

4.1 Introductory Remarks

In previous chapter we evaluated the major categories of Bitcoin wallet clients for the user. In this chapter we survey and evaluate the existing Bitcoin payments for a business to accept Bitcoin. We use SCRAM [Sut03] requirement engineering method to develop the most suitable Point of Sale (PoS) for a small business based on our evaluation framework. We would use PoS for point of sale terminal through out this chapter. Although we borrowed several concepts used in the previous chapter, the problems of wallets and PoS are distinct enough to necessitate a new evaluation framework specific to PoS.

One aspect of Bitcoin is that there is no specific entity backing up the currency,

whoever that is using Bitcoin gives value to it. As a Bitcoin enthusiast one of the goals is to have more places to accept Bitcoin, however this has been an issue for the business owners to implement a simple, yet fully functional PoS to be able to accept Bitcoin as a method of payment. This chapter discusses the approach we use to evaluate existing Bitcoin point of sales, our proposed approach, and the implementation of Aunja PoS in a Café in Montreal[1].

In order to do so, we started by eliciting the requirements of a payment system for a small business, and then researched the available options to see if they meet our requirements. Then, we put together a framework to compare these PoS with different criteria in security, privacy, usability and deployability categories. In the end, we implemented Aunja PoS for this small business that could be used in any other similar businesses as the payment method to accept Bitcoin.

4.2 Requirments Engineering

Requirement engineering is the process of defining, documenting and maintaining requirements and is a subfield of software and system engineering. The term was first used in 1979 [Alf79] and then became a general term with the publication of an IEEE Computer Society tutorial [Dor90]. Requirements Engineering is the first phase in waterfall software development process [Roc70]. Depending on the type of the system being developed, the methods differ.

_____

[1] Cafe Aunja `http://aunja.com`

Probably the best way to analyze the requirements of a small business is to model real world descriptions and stories in scenarios. Scenarios are examples of real world experiences that we could use to model what is needed in the system. That is why scenario-based requirements engineering method named SCRAM was chosen.

4.2.1 SCRAM

We used SCRAM (Scenario-based Requirements Analysis Method) as our framework to gather the requirements of this system. SCRAM defines four phases of requirement engineering and has been shown to be a great requirement engineering framework. This method consists of four phases:

- **Initial requirements capture and domain familiarisation:** This is done by interviewing and fact-finding methods to have a full understanding of how the business works.

- **Storyboarding and design visioning:** This is done by making storyboards and walkthroughs to show to the business and get feedback on feasibility.

- **Requirement exploration:** This uses the early prototypes and designs to get critiques from the business and validate the requirements.

- **Prototyping and requirement validation:** This is done by developing fully functional prototypes and continues refining the requirements until the product is acceptable by the business.
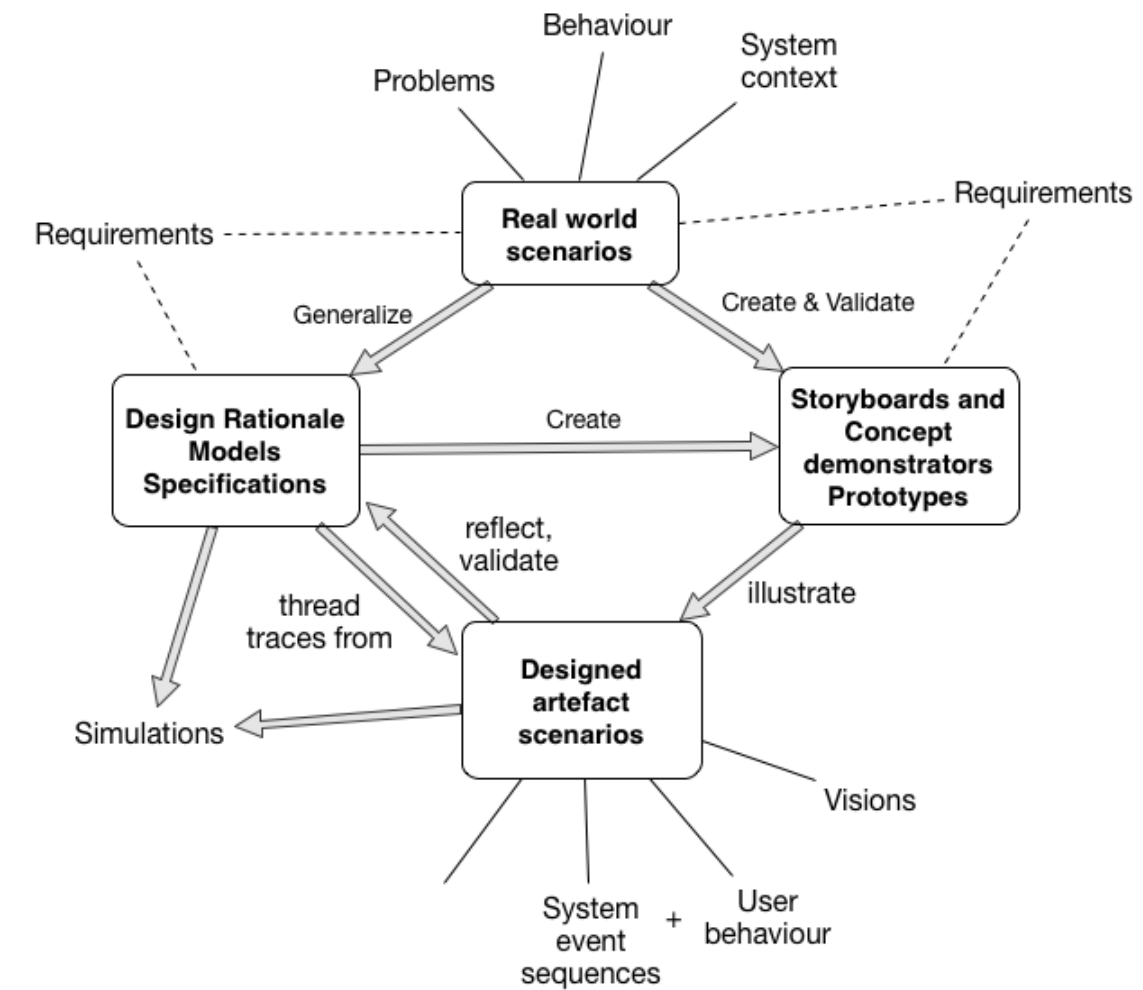
Figure 4.1: Role of scenarios and their relationship to requirements specifications and prototypes [Sut03]

Phase 1: Initial requirements capture and domain familiarisation. We asked the café owner, two employees and two customers for a scenario involving Bitcoin payment in the café to create the common "normal use case". The differences between the scenarios were insignificant thus the exceptions to this normal use case are not valid. One exception was power failure, and because even the current accepted payment methods such as Visa would fail, it was not considered as a valid scenario, that being said, there are methods to mitigate this that will be discussed later in the thesis such as browser-based PoS.

As the café already have other payment systems in place, there is no need to go through the café's business plan or any other specification to check for conflicts. The only change is to implement another payment system at cashier's desk (see figure 4.2). However, there are requirements in the PoS system that need to be met, such as realtime Bitcoin to fiat money exchange and obvious alert of successful or failed payments.

Phase 2: Storyboarding and design visioning. Based on the information gathered from Phase 1 and further analysis, such as user survey on the design, storyboard was developed, see figure 4.3.

Phase 3: Requirements exploration. We developed a proof of concept[2] , capable of doing a simple Bitcoin payment. The Bitcoin exchange rate and the amount of transactions were hard coded and the transaction would be executed manually. We asked the employees to run a mock purchase with the demonstrator to see how they
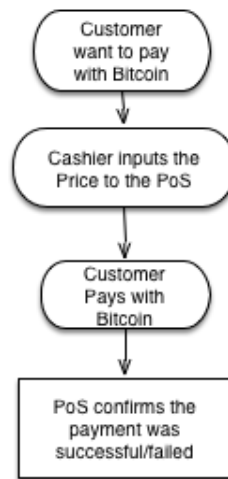
---

[2] concept demonstrator [Sut03]
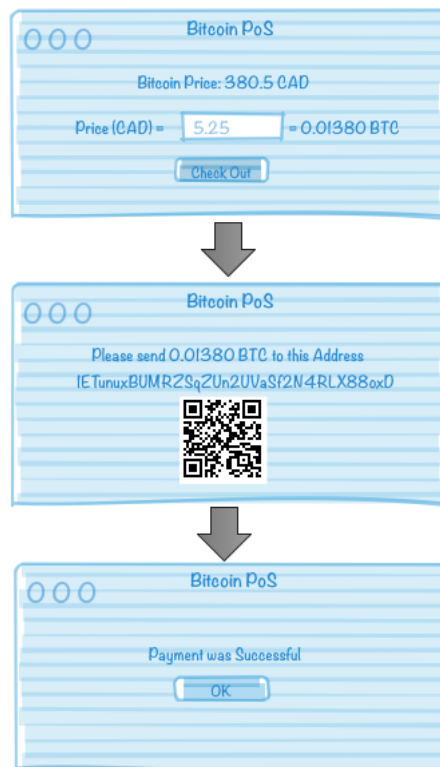
Figure 4.2: Phase 1 - Normal Use Case

Figure 4.3: Storyboard - User Interface first sketch

would interact with the system and received feedback. As Bitcoin concepts might be ambiguous for the new user, there should not be any interactions with Bitcoin concepts and terminology. After the transaction was done, the owner pointed that there is the need for a central logging system that could be checked from time to time for the daily transactions.

Phase 4: Prototyping and requirements validation. We used the feedback gathered from phase 3 to make the first prototype. The prototype retrieved the Bitcoin exchange rate in realtime and the employee only had to input the dollar amount in the PoS. This made it possible to keep the Bitcoin terminology out of the scope of the training for the employees. However on the first prototype, to show the successful payments, the system was showing the transaction on a Blockchain explorer[3], using web-based APIs. This was not clear for a novice user on what the state of the system is. On the second round of prototyping, we designed an interface to show that the transaction has been broadcasted to the Bitcoin network and would use that knowledge to inform the employee of the state of the payment.

---

[3]`http://blockchain.info`

4.3 Decision Framework

We propose a framework specialized for Bitcoin point of sale systems to score the system with a set of requirements based on usability, deployability, privacy and security. These are not a final set of requirements for a general purpose system, however in the case of Bitcoin payments for a small business these will suffice. It worths mentioning that these requirements are a subset of Systems and Software Quality Requirements known as ISO/IEC 25010:2011 [SQu].

We start by using our scenario based requirement engineering approach and adding the required non-functional requirements (*e.g.,* maintaining payee's privacy, data encryption). These requirements will be used to score each system described in this section, gathered in Table 4.1. For simplifying the figure, we use three score indicators. (●) for a complete score on the requirement, (○) if the requirements has not met completely and empty space if it is not satisfying the need. For some of the requirements the scoring system might be confusing (*e.g.,* low cost to run) which will be explained later.

4.3.1 Usability

There are different aspects of usability that should be considered. One is how the PoS is accessible for the employees and the other is technical matters of the implementation.

- **User Friendly:** The payment process should not be technical or complex for a café employee, a simple training for the employee would be enough for them to be able to accept Bitcoin. Also there should be a clear, mutual understanding when the payment is finalized. A PoS that has all of these features would score ($\bullet$), having some would result in scoring ($\circ$).

- **Time-Efficient:** the process of payment should not take significant amount of time more than the common payment systems such as Visa payments. If the process takes the same time as credit card payments it would score ($\bullet$), anything less than that would be ($\circ$) or none.

- **Fair Exchange Rate:** there should be a easy and fair approach for the payer and payee to have a consensus on fiat currency to Bitcoin exchange rate. If the price is retrieved from commonly accepted sources it would score ($\bullet$).

- **Availability:** all employees should be able to do the Bitcoin payment process without the need to know any credentials. If it's on a public domain for anyone to access it will score ($\bullet$), if it needs some private information it will score ($\circ$) and if it needs credentials it will score none.

4.3.2 Deployability

We use this category to state the requirements regarding the implementation of the system and branching. In the case of small businesses, the ability to manage multiple branch systems might not be a really important aspect. That being said, we will

score the systems for future work and hence to have a more complete framework.

- **Low Cost to Run:** PoS should be implemented in a way that is accessible with one of the currently owned devices of the café such as the cashier computer, the PoS terminal[4] or mobile devices. There should not be a need for buying new hardware or expensive software. For this requirement, we would score a (●) to a free of monetary cost system, and a (○) score to a moderate amount of spendings.

- **Enables Branching** The ability to install the point of sale on multiple branches, meaning the installation process for another branch of the business should not require modification on the PoS. Configuration might be needed to differentiate two branches in the system. If the PoS is packaged and easy to install on the second branch of the business it will score (●), if needs some modification (○) and if it is the same procedure to install it as the first one it will score none.

4.3.3 Privacy

Privacy is important in the payment system in the sense that no information should be leaked from any of the payers nor payee to the other party. This should be one of the fundamental requirements for any payment system.

- **No Information leakage:** There should not be any sensitive information available to the customer when she wants to pay with Bitcoin. These informa-

---

[4]The common PoS that accepts Visa/Debit Cards

tion could be the infrastructure of the business's network or a private domain used for accounting purposes. If it leaks any sensitive information it will score none and if it leaks some non-sensitive information it will score (○).

- **Maintains Payee's Privacy:** The payer should not be able to see how much the payee has received prior or after her payment but just her own amount of payment. If there is no link between the payments visible to the payer the PoS will score (●).

- **Maintains Payer's Privacy:** The payee should not be able to see how much the payer owns. This is one of the challenges that has not been fully solved [AKR+13]. It is the payer's responsibility to manage her funds and addresses in a sense that there is no privacy leak. All the PoS' in this evaluation scored (○) as this is outside the scope of the payee's PoS. This property is included to have a complete framework to evaluate future software.

- **Confidential Payments list:** The ability to see the payments list, only available for the manager by an authentication method, such as a password-protected panel. If the PoS offers a report page for the manager it will score (○), if the report page could have hierarchal authentication for employees with limited access it will score (●).

4.3.4 Security

Security might not be the café owner's priority as he might not have a deep understanding of this concept in payment systems nor in Bitcoin sphere. Anyhow it is one of the most important aspects in any financial payment system and also usable Bitcoin applications. Security of the system represents more than just the PoS code, it includes the environment that PoS is being used, the people using the software and the operating environment of the software [HLMN].

- **No 3rd-Party Trust:** There should be as little 3rd party trust as possible to accept and hold Bitcoin. Full trust to a third party will result in scoring none, some trust on the main functionality of the PoS result in (○) and no trust will result in (●) score.

- **Data Encryption:** In the case of any attacks on the service, there should be security measures that makes sure the attacker will not be able to have access to the private keys and transfer Bitcoin. Only if all the sensitive data is encrypted the PoS will score (●).

- **No Software Dependency:** The system should use as little dependencies as possible to minimize the attack vector on the server. This also falls into the deployability category as more dependencies could lead to the need of having a more complex system for implementation. If the PoS needs complex set of software or hardware to work it will score none, and if it could be executed in

a browser[5] without the need to run any other software it will score (●).

4.4 Evolution of PoS proposals

We will first go through the available options and why we chose to develop a custom PoS for this purpose.

There exist multiple payment systems which mostly suit the online markets (*e.g.,* e-commerece) and not a physical point of sale [6]. We list all the available approaches to accept Bitcoin payments for a physical business, and not as an e-commerce business.

4.4.1 One Bitcoin address - QR Code

One of the suggested ways for small businesses to accept Bitcoin is to hold one Bitcoin address and print out the QR code of that address near the cash register. In this way, the customer could scan the QR code and input the dollar value on his Bitcoin wallet and pay the business with the equivalent Bitcoins.

Usability It is not user friendly as it puts the employee in a position that she needs to know how Bitcoin transactions work and she needs to prepare,receive and check the payment manually (User friendly: none). This makes the time spent on the payment longer than normal payment systems (Time-efficient: none) . Same goes for

---

[5]In order to use a software PoS a mobile device or a computer is needed and we assume a web browser is by default installed on these devces

[6]`https://en.Bitcoin.it/wiki/How_to_accept_Bitcoin,_for_small_businesses`

the fair exchange rate, She should come to an agreed exchange price with the customer and this needs a deeper understanding of Bitcoin and finance (Fair exchange rate: ○, because the payee and payer should reach a consensus on the price). Thus technical training is required for each employee responsible for handling Bitcoin payments. As long as the QR-code print is visible to the payer, it is available to pay (availability: ●).

Deployability The cost to implement this method is almost zero (Low cost to run: ●), in monetary and time value. However, as mentioned in usability section, the time spent on each transaction fails for regular use. In case there are multiple branches, more print outs suffice to have multiple point of sales (Enables branching: ○).

Privacy This method provides no privacy for the seller (Payee's privacy: none). As all the Bitcoin transactions are publicly available in the Blockchain, anyone with the knowledge of the receiving Bitcoin address could see all the received payments, thus anyone could have access to the reporting page that is the payments received by the printed address (Confidential Payments list: none).

Security Other than the system holding the private key, not much security concern is applicable to this approach (No 3rd-party trust: ●). The private key should be kept in a secure place, preferably a cold storage unless the funds should be transferred to another address (*e.g.,* to exchange for cash). There are no software or data involved thus there is no software dependency (Data Encryption: none, No software dependency: ●).

4.4.2 Hardware Terminals

There are multiple hardware terminals available for accepting Bitcoin, however due to the high cost to run (*e.g.,* Coinkite[7] PoS are for sale at the starting price of 970USD), they have not been used in most of small businesses and have not been reviewed before. Also the fast changing technology made most of the terminal provider companies move to mobile or web-based solutions.

Usability The interfaces of each of the provided terminals are different. The most popular ones mimic the look and behaviour of a normal point of sale terminal used by credit card companies. However adding a new device to the payment routine would make it less user friendly and arises the need for training the employees (User friendly: ○). The time and availability of the payment through a hardware terminal should be the same as credit card payments if not lower (Time-efficient: ●) . The customer, nor the payee has any control over the exchange rate and it is provided by the PoS terminal operator (Fair exchange rate: ○). The device is accessible to anyone who has access to the other payment terminals (Availability: ●).

Deployability Due to the high costs these devices have, they score low in our framework (Low cost to run: none). Also in case there are multiple branches of the business, there should be one devices bought for each branch this makes the costs even higher (Enables branching: none).

Privacy Accepting Bitcoin with a hardware terminal should persevere the privacy the same as the regular credit card terminals, however the payees privacy depends

---

[7]`https://coinkite.com/store/products/all`

on the implementation of the Bitcoin payment system (Payee's privacy: ●). The terminal providers also offer similar interface to credit card terminals to list the payments (Confidential Payments list: ●).

Security The payee has no control over his private keys nor holds the funds (No 3rd-party trust: none), thus he needs to trust the third-party company that provided the terminals to keep the funds safe, and will receive the payments upon the agreed time frame with probably small transaction fees. As for other aspects of security, we assume the back-end implementation keeps the private keys encrypted and secure (Data encryption: ●). There are security risks involved in adding new hardware or software to the cashier's computer that will fall out of the scope of this chapter (No software dependency: none).

4.4.3 Online Merchant Services

Most of these services are focused for online businesses and don't have any implicit implementation for a physical payment system. One of the most famous ones, on the time of writing, is Bitpay[8] that takes 0% fees unlike some others competing companies, but they all have their own advantages. One other similar company is Coinbase[9] that charges 1% on exchanging Bitcoins to fiat currency.

Usability Implementing a Bitpay payment is straightforward and easy to implement. There are not many jargon or technical options for the employee (User friendly:

---

[8]`https://bitpay.com`

[9]`http://coinbase.com`

•). They have their own exchange rate (Fair exchange rate: ○) that the business owner could set to exchange to cash as soon as he receives payments, this will remove the possible effect that Bitcoin price volatility could have on the payments. It requires some credentials to access the PoS page (Availability: ○).

Deployability The only thing required by this approach is a smart phone or a small computer that users could interact with and browse to the Bitpay payment page, preferably with a touchscreen for easier price input and user interaction, as the interface is designed for touchscreen devices (Low cost to run: •). It is easy to add more branches to the original account or even make a new account for the second branch (Enables branching: •).

Privacy Bitpay another approach for preserving the privacy. As they generate a new address for each transaction, the payee's privacy is safe(Payee's privacy: •). However there has been reports of account suspensions because the payments were coming from flagged Bitcoin addresses (*e.g.,* black markets[10] or LocalBitcoins [11]), meaning that there was malicious activities on that Bitcoin address such as money laundry or buying drugs from online site. In this case, the privacy, as the sense that we are evaluating, is being held but maybe not in he aspects needed in a payment system. In order to view the payments, business owner should log in to his account and view the payments but other employees cannot see the list using any other accounts (Confidential Payments list: ○)

---

[10]Darknet Blackmarkets`https://en.wikipedia.org/wiki/Darknet_market`

[11]Peer to peer Bitcoin trading site `http://localBitcoins.com`

Security Every aspect of the payment system is implemented by Bitpay, they offered one of the most secure payment systems so far and there has been no big hacks reported (Data encryption: ○) . However, user has no control over his private keys and all the keys are being stored on Bitpay servers (No 3rd-party trust: none) which means complete trust to a third party. As they are a web-based solution, a device with a browser is enough to use their PoS (No software dependency: ●)

4.4.4 Self Hosting PoS

Another option is to run a customized wallet as the point of sale service. There are multiple options for this case and it depends on the features needed for managing the Bitcoin addresses. It is still possible to use a 3rd party for some of the functionalities like address generation or PoS interface. For the sake of simplicity we cover two popular methods, one using Mycelium Gear and another is a full custom self-hosting wallet using available open source software.

4.4.5 Mycelium Gear

Mycelium Gear [12] is a service offered by "Mycelium" group that offers a widget as an interface to the user and a service that would use the BIP32 public key provided on the Admin panel to generate new addresses securely. This means that they don't hold any private keys, but still uses the same set of paths for address generation as their Mycelium Mobile wallet uses.
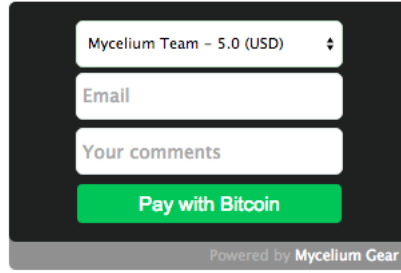
---

[12]https://gear.mycelium.com/

Figure 4.4: Mycelium Gear Widget

Usability Mycelium Gear is designed in a way to suit e-commerce business needs. It should be customized to suit a physical business PoS (User friendly: ∘) . There are no fees related to using this service, the only usability issue is that the BIP32 path[13] that is generated by the PoS widget, sometimes is different with the ones being checked by the mobile wallet client, so there might be some payments missing from the available credits in the application that is actually hard to retrieve if the path is unknown. They offer fast verifications on 0-confirmation transactions (Time-efficient: ●) and it's possible to chose from a list of supported exchanges to retrieve the Bitcoin exchange rate from (Fair exchange rate: ∘). A unique URL is needed to access the payment page and the employees should be aware of this link (Availability: ∘).

Deployability This method would be simple to implement but somehow more complicated to customize as there's not that much access to the code to be able to customize for business needs. Although the cost-to-run depending on the implementation could be almost zero (Low cost to run: ∘). The only deployability downside is that the payee is forced to use Mycelium Mobile wallet to manage his payments, how-

---

[13]see 2.2.1

ever doing so makes it easy to use the PoS in other branches and dedicate different accounts to each branch (Enables branching: ●).

Privacy As Mycelium Gear uses BIP32 to generate a new address for each transaction request the payees privacy is held (Payee's privacy: ●). However, there is no user management for the report page, If the customer closes the successful payment page, the employee would not be able to check if the payment was received or not unless he has the administrator password to check the transaction list (Confidential Payments list: ○).

Security Nothing related to the PoS holds any private information or keys that might be in danger of getting hacked, so there's no trust in any 3rd party in this sense. Although all the private keys would be in the Mycelium mobile wallet that is not prone to mobile malwares or hardware failure (No 3rd-party trust: ○,Data Encryption: ○) . Also this would be the weak point that if the hacker steals the phone, he has full access to all the available funds and also the future payments if stay unnoticed. The only software dependency is that the user is forced to use Mycelium mobile wallet (No software dependency: ○)

4.4.6 Aunja PoS

Depending on the requirements, it's possible to use integration of some open source software to build a fully custom self-managed Bitcoin PoS. The details of this custom PoS will be discussed in Section 4.5 and the scoring is discussed in Section 4.5.1.

Usability: As this is a fully customized PoS we could use the scenario based re-

quirement engineering method to implement a system that meets the business needs.

Deployability: cost-to-run this system depends on the requirements and how it is implemented. There might be some time needed to implement the prototype and change the bugs on the next round of requirement engineering when we get the feedback of the business owner and employees.

Privacy: We could implement the system with all the privacy measurements that need to be satisfy for the business owner. New address generation for each transaction would be basic need to have a good private PoS.

Security: Same as Privacy, it is possible to keep in mind all the security features when implementing this system. One of the basic needs is that the private keys should not be easily accessible, either kept offline or encrypted if they are stored on the online server and also there should not be any trust in any 3rd party as it is not needed on such a system. Although it should be mentioned that anytime that a third party code is being executed, we are basically trusting the developer for that software. However, in this case all the code used is open source and reviewed.

4.4.7   Desicion result

As you can see in table 4.1 there is no perfect solution out of the box for a small business to start accepting Bitcoin. After discussing the advantages and disadvantages of each method with the business owner, we decided to implement our own custom PoS using available open source software. This way it would be easy to incrementally change the PoS system with the customer and employees feedback to meet the needs

of the business.

4.5   Design and Implementation

There are multiple approaches for implementing Aunja PoS. We first have to see what programming language we want to use and under which environment. One of the lower cost methods would be to use a computer on the café's network as the web server but the maintenance and support would be really hard as the network might go down, or overwhelmed by the high number of connected devices and would not function properly. Uptime is one of the most important aspects for a payment system. The next low cost solution is to use shared hosting to host the wallet server and design a web based payment interface for the employees and also a reporting page for the business owner to track the Bitcoin payments. This made our decision easier to chose a programming language, the most common programming language supported by most shared hostings is PHP[14].

PHP is a server-side scripting language designed for web development and can be mixed with HTML to have more tools for interface design. It can be used with MySQL[15] as the database backend.

---

[14]PHP originally stood for Personal Home Page, it now stands for PHP: Hypertext Preprocessor`https://secure.php.net/manual/en/history.php.php`

[15]Structured Query Language

Table 4.1 — A comparison of Point of Sale gateways.

Category	User Friendly	Time-Efficient	Fair Exchange Rate	Availability	Low Cost to Run	Enables Branching	Maintains Payee's Privacy	Maintains Payer's Privacy	Confidential Payments list	No 3rd-Party Trust	Data Encryption	No Software Dependency
QRCode	○	●	○	●	●	○		○		●		●
Hardware Terminal	●	●	○	●	○	●	●	○	●	●	●	
Online Merchant Services	○	●	○	○	○	●	●	○	○	○	○	●
Mycelium Gear	○	●	○	○	○	●	●	○	○	○	○	○
Aunja PoS	●	●	●	●	○	○	●	○	●	●	●	○

Table 4.1: A comparison of Point of Sale gateways. ● indicates the category of client is awarded the benefit in the corresponding column. ○ partially awards the benefit. Details provided inline.

4.5.1 Implementation measurements

After multiple rounds of surveying employees and customers to understand their needs and also researching the subject, here is the break down of the results.

Usability

- **User Friendly (•):** The interface should be minimal and simple, with the ability to show the exchange price of Bitcoin to fiat currency, input box for the price in dollars, estimation of Bitcoin amount equivalent to the price and a note section to jot down the details of the transaction. As for the user facing interface, it should be simple, showing all the required information such as Bitcoin amount, the exchange rate and the QRCode for the deposit Bitcoin address. Both interfaces should indicate when the transaction is complete.

- **Time-Efficient (•):** It should not take more than normal payment system to initiate the payment. A web based interface would have the advantage that it can be loaded from any device with good speed, depending on the Internet speed. Also to verify the payment it should not take a long time. It also need to use fast verification methods to indicate that the payment is propagated (broadcasted to the Bitcoin network). Knowing that a propagated transaction is not same as confirmed transaction but is an accepted risk for low volume transactions.

- **Fair Exchange Rate (•):** After doing our research on this we found the web-

site called Bitcoinaverage[16] that offers a good combination of all the exchange prices to come up with an average daily price to be used as the fair exchange rate.

- **Availability (•):** The payment interface should be open to public in the sense that it could be loaded on any device.

Deployability

- **Low Cost to Run (○):** The only costs associated with this implementation would be the annual cost of the shared hosting that nowadays is less than 100 dollars for an unlimited web host. For the sake of this research, there would be no other implementation and development costs.

- **Enables Branching (○):** For now there's no plan to have more branches for this business, but depending on the implementation, to have another branch it would be as easy as running another instance of the application on the server.

Privacy

- **No Information leakage:** The payment interface does not reveal any information about the backend nor the business' internal detail.

---

[16]"BitcoinAverage.com is the first aggregated bitcon price index that was initially launched in August 2013 with a goal to aggregate rates from all available Bitcoin exchanges around the world and provide a weighted average Bitcoin price." `https://Bitcoinaverage.com`

- **Maintains Payee's Privacy (●):** There should be a new address generated for each transaction request so no one can see how much the business have received in Bitcoin prior or after each transaction.

- **Maintains Payer's Privacy (○) :** This would be the payers Bitcoin wallet client responsibility and it would be out of the scope of this PoS system.

- **Confidential Payments list (●):** There should be a reporting and administration interface designed, only accessible to the business owner or designated personals.

Security

- **No 3rd-Party Trust (●):** There should not be any sensitive usage of 3rd parties in the system, it should work as a stand alone system.

- **Data Encryption (●):** All the private keys should be encrypted and then stored on the server.

- **No Software dependency (○):** There should not be any software dependency on the payment page for the business. The software dependencies on the server side should all be included in the package as open source software. The only piece of software required to use a PoS on a mobile device should be a web browser and we don't consider this as a software dependency.

4.5.2 Open source libraries and software applications

There are multiple approaches to implementing the PoS. After the requirement engineering phase, we chose PHP as our main programming language to code this project. This narrows down the options to a few open source projects. As we examined different PHP Bitcoin projects, we chose the following as the base of our PoS software:

Bitcoin libraries

- **Bitcoin SCI:** Bitcoin Shopping Card Interface

- **PHP Elliptic Curve library**[17]**:** Used as a dependency to Bitcoin SCI to generate Bitcoin addresses.

- **Bitcoin-prices**[18]**:** Display Bitcoin prices in human-friendly manner in fiat currency using Bitcoinaverage.com market data

After searching the Internet, we decided to use "Bitcoin SCI: process Bitcoin transactions with PHP" as the software to use as our Bitcoin core. It is originally designed to be integrated in e-commerce websites but it could be easily modified to meet our needs.

Bitcoin SCI (Bitcoin Shopping Cart Interface [19]): is a set of libraries and tools that enables the user to process Bitcoin transactions with only PHP.

---

[17]http://matejdanter.com

[18]https://github.com/miohtama/Bitcoin-prices

[19]http://bitfreak.info/?page=tools&t=bitsci

Figure 4.5: Bitcoin SCI (Bitcoin Shopping Cart Interface)

This is not a complete project to process payments. The first decision was to use this package for building the prototype and then if we failed to modify the package to meet out needs, use another approach, however we could make it suit the needs and Bitcoin SCI was used in the end product.

A break down of the tools Bitcoin SCI gives us are as follow:

- **Bitcoin Address generation:** Bitcoin SCI uses PHP Elliptic Curve library to generate new secure Bitcoin addresses (set of public and private keys)

- **Private key encryption:** using phpseclib library, all the private information (Bitcoin private keys, transaction details) are stored encrypted

- **Payment Confirmation:** It uses APIs from a blockchain explorer site [20] to

---

[20]blockexplorer.com

confirm receiving payments.

- **Input Interface:** even though this package was meant to be used as an e-commerce payment system, it has the basic tools and methods to build the price input page

However it lacks some other features that should be added:

- **Database:** In order to have management and report page, saving the transaction details into a database is a must.

- **Fair Bitcoin Exchange rate:** It uses a predefined source to obtain the exchange rate of Bitcoin and it's not possible to set different currencies as the input

- **User-Friendly interface:** All the interfaces are poorly designed and need to be modified to suit the PoS system.

- **Report Page:** We need a report page with authentication in place.

- **Input Validation:** Other than security perspective of input validation, this is needed because of the way we want the PoS to work. It should alert the employee if she has done something wrong before going to the next page and adding a failed transaction to the database.

- **Cash out option:** As all the private keys are stored encrypted in the server, we need a way to cash out the available Bitcoins and send them to another

Bitcoin address. It's possible to retrieve the private keys of each Bitcoin address separately from the tool, but it's not scalable to multiple weekly transactions.

Bitcoin-prices This library allows us to use Bitcoinaverage.com prices as our main source of price conversion, and it gives nice tools for interface design, such as the ability to switch between different currencies by just clicking on the price. This allows us to reach a fair exchange rate that is also shown in different currencies in case it was needed.

Encryption libraries

- **phpseclib**[21]: used for private key encryptions.

We used this library mainly because it was already included in the Bitcoin SCI package as a dependency, but later on when we added the database functionality, we needed a library for encryption purposes that they were all included in phpseclib.

Interface libraries

- **Sweet Alert** [22]: A Beautiful replacement for javascript's "Alert",

This is a nice Javascript library that we used to make the interface more user-friendly. Also in the case of data validation, we needed a simple and nice way to inform the employee that she made a mistake on the form and the mistake should be

---

[21]`http://phpseclib.sourceforge.net`
[22]`http://t4t5.github.io/sweetalert`

fixed. For this case Javascript is the best option in the sense that it could validate the inputs on the browser before sending it to the server.

4.5.3 Prototyping

With the full knowledge of the requirements and a few sketches of the interface, we started developing the PoS. Although the first prototype was ready to launch within a week, we did 3 prototypes in the month after that, each had bugs fixed and features added as we surveyed and obtained feedback from the employees on each round of prototyping.

Here is a short description of the implemented functionalities:
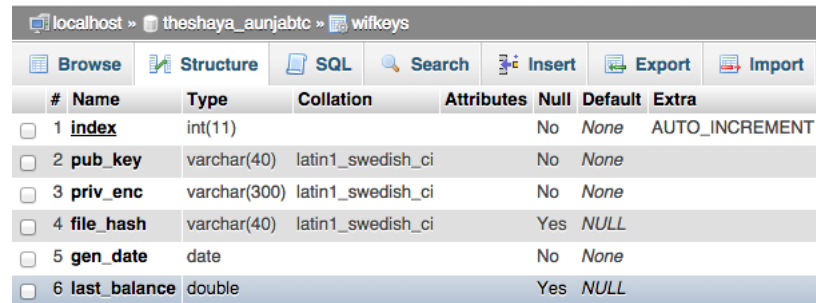
PoS main functionalities

The PoS was hosted on a shared hosting service named Host Monster [23]. They offer low cost annual plans that offer PHP and MySQL which are the requirements that we need. Then we started working with Bitcoin SCI to add the database functionality and defined tables for transaction requests and payments on MySQL.

I designed three tables (first prototype had two) for the purpose of this PoS. One table is for the Bitcoin key pairs to be stored and will be used to decrypt and export the private keys when required (see Figure 4.6). The other table holds the information regarding each transaction (Figure 4.7), later on to prevent spam requests and test cases from making the table unnecessary big, we designed a temporary transaction
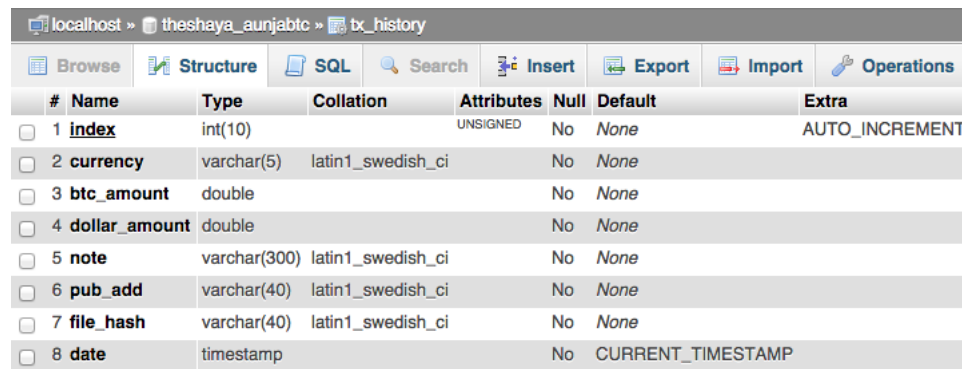
---

[23]http://hostmonster.com

table to hold the data of each transaction before the payment is validated. As soon as the payment is flagged as valid, the relevant data would be moved to the main table and will be removed from the temporary table.

Figure 4.6: Structure for wifkeys table that holds the Bitcoin key pairs

Figure 4.7: Structure for transaction history table

Bitcoin SCI uses a file-based method to store the keys and transaction details, as a backup method, we kept that in place and stored the file_hash detail of each entity for future references.

Other tasks were involved in integrating the above mentioned open source projects into each other to have a complete solution package.

One of the features that were added on the second round of prototyping was the

Figure 4.8: Aunja PoS - First View

ability to show the Bitcoin price in USD other than the default CAD, this was added

with the usage of Bitcoin-prices library. The other was to add the "Notes" field to

be able to add invoice ID or the items that the customer bought. It was possible to

implement a drop down menu with all the café's menu options to be added to the list

but as we discussed this solution with the café owner, he mentioned that the items

in the menu might not stay the same during the year and also there might be price

changes, so that approach was not suitable for this business, although it might be a

good option for an e-commerce site.

Figure 4.9: Aunja PoS - Payment

Private reporting page

One other aspect of the requirements was a reporting page, this was based on the feedbacks from the café's owner and his preferences.

Id	Date	Bitcoin Amount	Sale Dollar Amount	Note	Bitcoin Address
1	2015-04-18 14:12:33	4.68 CAD	3.5 CAD		1PKaQvzm3dXAHxYs7sb48ytahxxQzU8wKj
2	2015-05-07 13:46:15	29.33 CAD	23 CAD		17GTN1N2PKbyAfoSXtSkdfgHPeXwvVmY9x
3	2015-06-22 18:31:55	5.54 CAD	4.6 CAD		1FfYcmWiSnBmKdhEVRhsiVAC9GB6pZK3da
4	2015-06-22 18:32:17	6.75 CAD	5.6 CAD		12JKE6nGigN9VKuJQUvZSooekj5kPZ9dKT
Summary		46.30 CAD	36.7 CAD		

Figure 4.10: Report Page

One of the important fields added later to the report page was the "Sale Dollar Amount". The reason was that Bitcoin price is really volatile compared to other currencies and the café owner did not want to risk losing money by accepting Bitcoin. As you can see in (Figure 4.12), the Sale Dollar amount is less than the Bitcoin amount. In this period of time, the owner could have had more profit on the sales because of the increase in Bitcoin prices, but this would be considered as a risk that he did not want to take. So as an agreement, we decided to lock the price of each sale on the sale time to be paid the same amount as if he was selling his products with cash[24], Thus on the second prototype of the report page, this field was added for accounting purposes. This is real-time exchange, However as an agreement, cashing out the Bitcoins would happen in monthly basis or within a threshold (*e.g.,* when reached 100 dollars).

Another added feature was the ability to check each transaction on the blockchain.

---

[24]this method is actually one of the common methods recently used by Bitcoin payment processors.

98

If the café owner clicks on any of the Bitcoin addresses related to each sale, he would be redirected to a blockchain explorer site and he can see if the transaction went through or not.

Figure 4.11: A canceled sale - this means that the request was made on the Aunja PoS interface to generate an address, but the customer never sent the Bitcoins. Probably a customer changed his mind and paid via another payment method

Another feature request was the ability to decrypt and export the private keys of those addresses that has some balance. This has been done for the admin page that is out of the scope of this chapter.

Aunja PoS has been made open source and available to public[25] under GNU General Public License v2 and has already been used in other small businesses to accept Bitcoin.

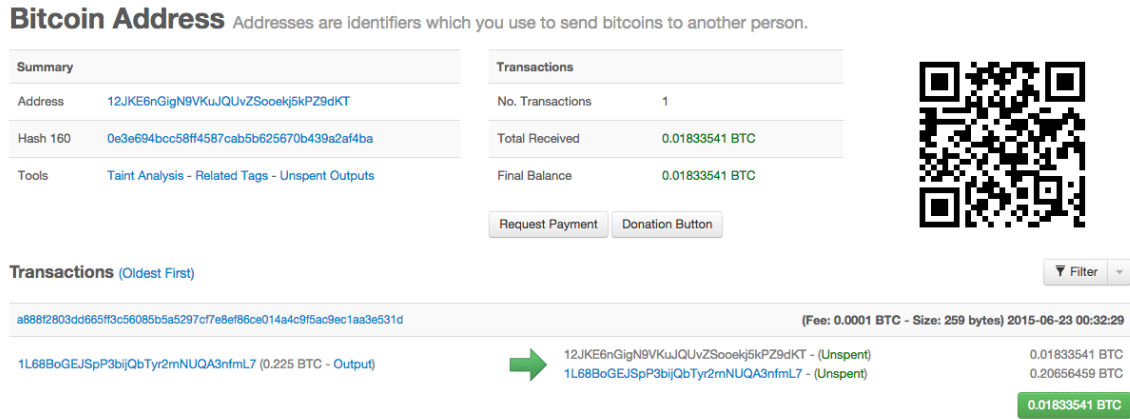[25]https://github.com/shayanb/Bitcoin-PoS-PHP

Figure 4.12: A Complete Sale - This shows that 0.01833541 BTC (approximately 5.5 CAD on the time of sale) was deposited in the address generated by the Aunja PoS

4.5.4 Training

We tried to make the interface as simple as possible for the employees. There is no jargon or technical requirements to use Aunja PoS, but some details specific to Bitcoin transactions have to be taught to the employees to be able to recover from human errors while a transaction is being processed. Other than in-person training that was done with every employee, a manual was made (Figure 4.13) and was attached to the cashier's counter for future reference by all café employees.

4.6 Real-world Deployment

Café Aunja started accepting Bitcoin with Aunja PoS on Oct 23, 2014, and was one of the first cafés in eastern Canada that accepts Bitcoin. During the first month, there was more than 10 Bitcoin payments and it has been working ever since.
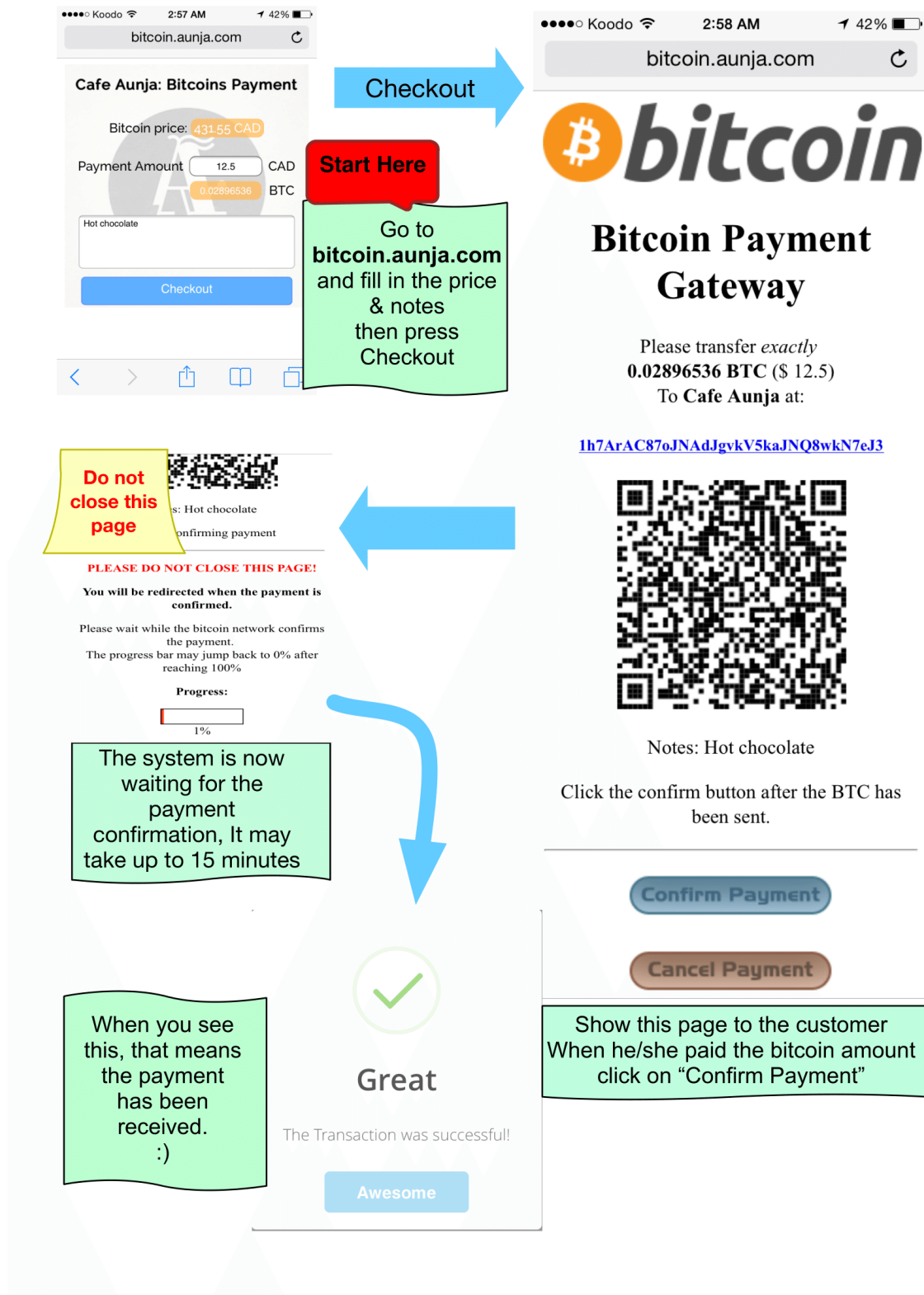
Figure 4.13: PoS - Step by step manual for Bitcoin payments

4.6.1   Lessons learned

One of the missing features that should be implemented in such a system is a fast verification method. In early Bitcoin PoS, for each payment, the customer needs to wait 10 minutes in average for the transaction to be confirmed and included in the blockchain. To remedy this, we solved this issue by flagging the transactions as successful as soon as the transaction is broadcasted to the Bitcoin network, also known as, 0-confirmation transaction. This could work for a PoS in a café as the volume of each transaction is small and it's not risky to take 0-confirmation transactions, Another newly introduced method to overcome the confirmation time is by using the Bitcoin Lightening Network [PD15] but the technical details on lightening network goes beyond the scope of this thesis. However this is still an open problem to remedy the risk for higher value transactions and prevent double spend attacks [KAC12] [BDE+13].

Bitcoin and Bitcoin transactions are still new concepts for most people. We encountered a countless number of questions from customers to explain what Bitcoin is and how it works and mostly they became more interested to know more about Bitcoin when they observed a payment done with the Bitcoin PoS, mostly because they would not reveal any personal information with each payment.

Another interesting lesson is the concept of locked price that is the price of Bitcoin for each sale is locked to the exact exchange rate at the time of the transaction. This means if the customer paid 0.01 Bitcoin for a coffee that was 3 dollars in the time

of the purchase, the business owner will be paid the exact same amount of 3 dollars and it should not differ if the Bitcoin price has been increasing or decreasing from the time of the purchase to the time he chooses to cash out the receiving Bitcoin. This makes the acceptance of Bitcoin payments for the business risk-free.

Now to test the system on production, it was time for the first coffee, to the best of our knowledge, in eastern Canada to be bought with Bitcoin (Figure 4.14).

index	currency	btc_amount	dollar_amount	note	pub_add	file_hash	date
1		0.00231064	1	testing	19sTMpGrdNC1hsxz5dRxeUigGDhSCUB1JF	377c562aa7fb1aea0c550a479692f869	2014-10-17 19:12:02
2		0.01013034	4.5	Hot chocolate, First bitcoin payment	138tN3PaKXQJTY697cMijbGvVaj15Fcw7J	3558408149a322c087b670d2fde90e35	2014-10-19 19:41:56

Figure 4.14: Database details of the first coffee bought with Bitcoin in the café

Figure 4.15: Café Aunja Started to accept Bitcoin on Oct 23, 2014

Chapter 5

Conclusion

In this thesis, we summarized all the existing Bitcoin wallet clients and put together a framework to evaluate the upcoming software. We concluded that there is no perfect wallet client yet and we have a long way to go to have a perfect solution. We also evaluate the Bitcoin point of sales that accepted Bitcoin and again there was no perfect solution for a small business to easily accept Bitcoin. We developed and deployed a custom point of sale for small businesses and published it as an open source application that has already been used by a few more small businesses all over the world.

These are small steps towards having a framework to evaluate Bitcoin software, wallet clients and payment processors, but they are necessary first steps.

Bibliography

[AKR+13]   Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *Financial Cryptography and Data Security*, pages 34–51. Springer, 2013.

[Alf79]   Mack Alford. *Software Requirements Engineering Methodology*. Wiley Online Library, 1979.

[Arm]   Armory. Armory Secure Wallet. `https://bitcoinarmory.com`.

[BBSU12]   Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to Better — How to Make Bitcoin a Better Currency. In *Financial Cryptography*, 2012.

[BDE+13]   Tobias Bamert, Christian Decker, Lennart Elsen, Roger Wattenhofer, and Samuel Welten. Have a snack, pay with bitcoins. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–5. IEEE, 2013.

[BHvOS12] J Bonneau, C Herley, P C van Oorschot, and F Stajano. The quest to replace passwords: a framework for comparative evaluation of web authentication schemes. In *IEEE Symposium on Security and Privacy*, 2012.

[Bit] Bitcoin Core Developers. Bitcoin Core. `https://bitcoin.org`.

[BKM05] Mary Baker, Kimberly Keeton, and Sean Martin. Why traditional storage systems don't help us save stuff forever. In *HotDep*, 2005.

[Blo] Blockchain Team. My Wallet Be Your Own Bank. `https://blockchain.info`.

[BNM+14] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *Financial Cryptography*, 2014.

[bra] brainwallet. Brainwallet. `https://brainwallet.github.io/`.

[BvOP+09] Robert Biddle, P. C. van Oorschot, Andrew S. Patrick, Jennifer Sobey, and Tara Whalen. Browser interfaces and extended validation ssl certificates: An empirical study. In *CCSW*, 2009.

[Cha82] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, 1982.

[Cha83]   David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[CKM87]   J Carroll, W Kellogg, and R Mack. *Interface metaphors and user interface design.* 1987.

[CvOA07]   Jeremy Clark, Paul C. van Oorschot, and Carlisle Adams. Usability of anonymous web browsing: An examination of tor interfaces and deployability. In *SOUPS*, 2007.

[DCBW12] M. Dietz, A. Czeskis, D. Balfanz, and D. S. Wallach. Origin-bound certificates: A fresh approach to strong client authentication for the web. In *USENIX Security*, 2012.

[Dor90]   Merlin Dorfman. System and software requirements engineering. In *IEEE Computer Society Press Tutorial.* Citeseer, 1990.

[EBSC15]   Shayan Eskandari, David Barrera, Elizabeth Stobert, and Jeremy Clark. A first look at the usability of bitcoin key management. In *Workshop on Usable Security (USEC)*, 2015.

[FH12]   D. Florencio and C. Herley. Is everything we know about password stealing wrong? *IEEE Security & Privacy*, 10(6), 2012.

[GFFK06]   Shirley Gaw, Edward W. Felten, and Patricia Fernandez-Kelly. Secrecy, flagging, and paranoia: Adoption criteria in encrypted email. In *CHI*, 2006.

[GKGC14] Arthur Gervais, Ghassan Karame, Damian Gruber, and Srdjan Capkun. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *ACSAC*. ACM, 2014.

[GM05] Simson L. Garfinkel and Robert C. Miller. Johnny 2: A user test of key continuity management with S/MIME and outlook express. In *SOUPS*, 2005.

[GMS⁺05] Simson L. Garfinkel, David Margrave, Jeffrey I. Schiller, Erik Nordlander, and Robert C. Miller. How to make secure email easier to use. In *CHI*, 2005.

[HK09] Steve H Hanke and Alex KF Kwok. On the measurement of zimbabwe's hyperinflation. *Cato J.*, 29:353, 2009.

[HLMN] C B Haley, R Laney, J D Moffett, and B Nuseibeh. Security Requirements Engineering: A Framework for Representation and Analysis. *IEEE Transactions on Software Engineering*, 34(1):133–153.

[HvO12] Cormac Herley and Paul C van Oorschot. A Research Agenda Acknowledging the Persistence of Passwords. *IEEE Security & Privacy*, 10(1):28–36, 2012.

[KAC12] Ghassan Karame, Elli Androulaki, and Srdjan Capkun. Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. *IACR Cryptology ePrint Archive*, 2012:248, 2012.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zero-coin: Anonymous Distributed E-Cash from Bitcoin. In *IEEE Symposium on Security and Privacy*, 2013.

[Mul]   MultiBit Team. MultiBit. `https://multibit.org`.

[Nak08]   S Nakamoto. Bitcoin: A peer-to-peer electionic cash system. Unpublished, 2008.

[Nie92]   Jakob Nielsen. Finding usability problems through heuristic evaluation. In *CHI*, 1992.

[Oec03]   Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *CRYPTO*, 2003.

[PD15]   Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. Technical report, Technical Report (draft). https://lightning. network, 2015.

[Piea]   Pieter Wuille. BIP32 Hierarchical Deterministic Wallets. `https:// github.com/bitcoin/bips/blob/master/bip-0032.mediawiki`.

[Pieb]   Pieter Wuille. BIP32: Hierarchical Deterministic Wallets. `https:// github.com/genjix/bips/blob/master/bip-0032.md`.

[Poi]   Pointbiz. JavaScript Client-Side Bitcoin Wallet Generator. `https:// bitaddress.org`.

[Roc70]   WW Rocye. Managing the development of large software systems: con-
           cepts and techniques. In *IEEE WESTCON*, 1970.

[RSA]   RSA Laboratories. PBKDF2 (Password-Based Key Derivation Function
           2). `http://tools.ietf.org/html/rfc2898`.

[SBKH06]   Steve Sheng, Levi Broderick, Colleen Alison Koranda, and Jeremy J.
           Hyland. Why Johnny still can't encrypt: Evaluating the usability of
           email encryption software. In *SOUPS (Poster)*, 2006.

[SQu]   ISO SQuaRE. ISO/IEC 25010:2011 Systems and software engineering –
           Systems and software Quality Requirements and Evaluation (SQuaRE)
           – System and software quality models. `http://www.iso.org/iso/`
           `catalogue_detail.htm?csnumber=35733`.

[Sut03]   A. Sutcliffe. Scenario-based requirements engineering. In *Requirements
           Engineering Conference, 2003. Proceedings. 11th IEEE International*,
           pages 320–329, Sept 2003.

[Sym]   Symantec. Infostealer.Coinbit. `http://www.symantec.com/security_`
           `response/writeup.jsp?docid=2011-061615-3651-99`.

[Van92]   S. Vanstone. Responses to NIST's Proposal. *Communications of the
           ACM*, 35:50–52, 1992.

[Wik]   Wiki.   Technical   background   of version 1 Bitcoin   addresses.
           `https://en.bitcoin.it/wiki/Technical_background_of_version_`
           `1_Bitcoin_addresses`.

[WRLP94]   C. Wharton, J. Rieman, C. Lewis, and P. Polson. The cognitive walk-
           through method: a practitioner's guide. In *Usability Inspection*. Wiley
           & Sons, 1994.

[WT99]   A. Whitten and J. D. Tygar. Why Johnny can't encrypt: a usability
           evaluation of PGP 5.0. In *USENIX Security*, 1999.