



University of
Zurich^{UZH}

CoinBlesk, a Mobile NFC Bitcoin Payment System

Raphael Voellmy
Zurich, Switzerland
Student ID: 09-912-585

Supervisor: Thomas Bocek
Date of Submission: Aug 26th, 2015

Abstract

Even though Bitcoin has been around for several years now and has proved to be a reliable payment system, real-world adoption is still rare. The goal of this thesis is to provide a proof of concept of a payment system that is secure for users, fully compliant with the Swiss banking laws and convenient to use for both merchant and customer. This thesis provides a overview of the problem and challenges of the usage of Bitcoin in real-world environments, describes solutions to address these problems and includes an open-sourced implementation of a proof of concept of an NFC capable Android Bitcoin payment system.

Zusammenfassung

Obwohl Bitcoin seit mehreren Jahren existiert und sich als zuverlässiges Zahlungssystem bewiesen hat, ist die Anwendung in physischen Geschäften immer noch selten anzutreffen. Das Ziel dieser Arbeit ist es, einen Prototypen eines Zahlungssystems bereitzustellen, welcher sicher für die Benutzer, gesetzeskonform nach Schweizerischem Bankenrecht und benutzerfreundlich für Verkäufer und Kunden ist. Diese Arbeit gibt einen Überblick der Probleme und Herausforderungen beim Akzeptieren von bitcoins in physischen Geschäften und beschreibt Lösungsansätze um diese Probleme anzugehen und beinhaltet einen Open-Source Prototypen eines NFC fähigen Android Bitcoin Zahlungssystems.

Acknowledgments

I wish to express my sincere thanks to my supervisor Thomas Bocek for his tireless support during the course of this thesis.

Contents

Abstract	i
Zusammenfassung	iii
Acknowledgments	v
1 Introduction	1
1.1 Motivation	1
1.2 Description of Work	1
1.3 Thesis Outline	2
2 Background & Related Work	3
2.1 The Bitcoin System	3
2.2 Real-World Usage of Bitcoin	4
2.2.1 Legal Challenges	5
2.2.2 Technical Challenges	6
2.2.3 Usability Challenges	7
2.3 Requirements for a Bitcoin Payment System	7
2.3.1 Mitigating Risk of Volatility	7
2.3.2 Compliance with Swiss Banking Law	7
2.3.3 Performance and Convenience	8
2.4 Existing Solutions	9
2.4.1 GreenAddress: Shared Control of Bitcoins	9
2.4.2 BitPay: Instant Currency Swap and NFC Payments	9

3	Design	11
3.1	Client-Server Architecture	11
3.2	Multi-Signature Setup	12
3.2.1	The Property of Bitcoin and the Power of Script	12
3.2.2	P2SH Addresses	12
3.2.3	Hierarchical Deterministic Wallets	13
3.2.4	Preventing Double-Spending Attacks	15
3.3	Refund Transactions	15
3.3.1	Re-Deposits	16
3.4	NFC Payment Protocol	17
3.4.1	Payment Request	18
3.4.2	Send Request	18
3.5	Instant Currency Swap	20
4	Implementation	23
4.1	Wallet Service	23
4.2	Main Features and Views	24
4.2.1	Home Screen	24
4.2.2	Create Transaction	25
4.2.3	Transaction History	25
4.2.4	Settings	25
4.2.5	Pay In	26
4.2.6	Pay Out	26
4.2.7	Address Book	27
4.2.8	Send Out	27
4.3	Libraries & Tools Used	27

<i>CONTENTS</i>	ix
5 Evaluation	29
5.1 Scalability & Performance	29
5.2 Thesis Goals	30
5.3 Thesis Result	31
6 Summary and Conclusions	33
Bibliography	35
Abbreviations	37
Glossary	39
List of Figures	39
List of Tables	41
A Screenshots of the Android App	45
B Installation Guidelines	51
B.1 Installation of the Server	51
B.2 Installation of the Client	52
C Contents of the CD	53

Chapter 1

Introduction

1.1 Motivation

The adoption of Bitcoin in real-world stores progresses slowly, even though Bitcoin has the potential of being a fast, convenient and secure way of paying for goods. The total transaction volume of bitcoin only accounts for a tiny fraction of the mobile payments market. This despite of the fact that bitcoins are cheaper to accept for merchants compared to alternative payment systems, and Bitcoin protects the privacy of customers.

The goal of this thesis is to show that Bitcoin can be used in real-world environments in a convenient matter, without needing a banking license, while at the same time saving the merchant money and protecting the privacy of customers. On top of that, this thesis shows that only minimal trust in the payment system is required by the customers and that even in the case of total disappearance of the payment system operator, the customers can still claim their funds.

1.2 Description of Work

In 2014, three students of the University of Zurich developed an NFC capable mobile Bitcoin payment system called CoinBlesk. It was a centralized payment system, where clients interacted with a single Bitcoin Core instance running on a server [1]. During the course of six month, this original CoinBlesk implementation has been fundamentally restructured to a more decentralized and (almost) trust-less Bitcoin payment system. This new version, hereby called CoinBlesk 2.0, gives the control over the bitcoins back to the users, and is implemented in a way such that it is fully compliant with the Swiss banking laws. CoinBlesk 2.0 consists of an Android app and a Java server, both open-sourced and available at github.com [26].

1.3 Thesis Outline

Chapter 2 (Background & Related Work) describes the Bitcoin system in general, the current state of real-world adoptions of Bitcoin and tries to answer the question why the adoption rate is slow. Later, requirements of a Bitcoin payment system are specified that could improve the adoption rate of Bitcoin in real-world environments. Last but not least we look at existing services that already try to address the problems and partially solve the problems outlined in the beginning of this chapter. Later in chapter 3 (Design) the design of CoinBlesk 2.0 is explained, along with reasoning on *why* the system has been designed the way it was designed. The chapter 4 (Implementation) describes the end result of the proof-of-concept implementation of the system described in chapter 3.

Chapter 2

Background & Related Work

2.1 The Bitcoin System

Bitcoin is a publicly distributed ledger which, due to its incentive system, eventually reaches a consistent state as long as players act rationally. It is a combination of cryptography, P2P network and economics, solving the problem of consistency in distributed networks that remained unsolved until 2008. Currently the main application of this eventually consistent, distributed ledger is a digital currency called “bitcoin”. By convention, Bitcoin with an upper case B refers to the Bitcoin system as a whole while bitcoin written with a lower case b refers to the currency.

Bitcoin reaches a consensus by a processing power based voting mechanism called “mining”. By taking a hash of the previously confirmed Bitcoin transactions and a random nonce as input, the nodes in the network are constantly trying different nonces until a nonce is found such that the output of a defined hashing function is a hash that is lower than a certain number defined by the Bitcoin protocol. This number, called “difficulty”, is adjusted every two weeks to ensure that on average every 10 minutes a new hash meeting the defined criteria is found. As soon as a node found such a nonce, the node is allowed to create a “block” containing previously unconfirmed transactions and the nonce. The nonce is proof that the node confirming the transactions expended a certain amount of processing power to confirm the transactions. The transactions contained in this new block now have one confirmation by the Bitcoin network. The nodes in the network now use this new block and try to find a new nonce based on the hash of this newly broadcast block. This way, the Bitcoin “block chain” is formed and constantly expanded. All bitcoin transaction that are in the longest chain are considered valid. This means that an attacker trying to create a malicious block chain - e.g. to reverse confirmed transactions - would have to outperform the processing power of the entire network in order to form a malicious block chain that is longer than the non-malicious block chain. The nonce included in every block is a “proof-of-work”, proofing the network that a considerable amount of processing power has been expended to create the block. By only allowing a vote on the validity of transactions with a proof-of-work, Sybil-Attacks are prevented. The amount of identities in the Bitcoin network does not matter, only the amount of processing power counts.

Other than including the nonce, a miner has to follow a strictly defined set of rules when creating a new block. One - rather obvious - rule is that a bitcoin cannot be spent more than once. This means that if a node encounters the case that two different transactions spend the same bitcoins, the node needs to decide which one of the two transactions is valid. Attempting to spend the same bitcoins twice is called “double-spending attack” in the Bitcoin terminology. If the node follows all rules defined in the Bitcoin protocol, the node receives a reward from the network. This reward is an incentive for nodes to play by the rules. If a node decides to break the rules, other nodes in the network will reject the node’s block and the malicious node does not receive the reward. Since the reward for deciding on the validity of transaction is quite lucrative, there’s a constant competition which node is allowed to confirm transactions. Figure 2.1 shows the rapid growth of processing power invested in the Bitcoin network. The more processing power a node invests, the higher the probability of finding the next block and gain the reward [2].

Trying to outperform the entire network’s processing power is called a “51% Attack”. An entity performing such an attack could reverse transactions that have already been confirmed or simply deny to confirm any new transactions at all by only creating empty block. It can be assumed that the cost of performing a 51% attack roughly equilibrates the value that is generated from the Bitcoin network. Considering that a 51% attack would completely destroy the trust in the Bitcoin network and essentially render the whole system useless, the value of a bitcoin would probably plummet to a price close to zero. It can therefore be assumed that no rational entity would consider this attack and instead optimize revenue by playing by the rules [21]. History shows that this assumption has held true in the past, a 51% attack has never been successfully performed. There have been inconsistent states in the Bitcoin network, but only due to technical problems of Bitcoin protocol implementations, not targeted attacks [22] [23].

Since in today’s Bitcoin network it is almost impossible to create a new block alone, miners pool their processing power and try to create new blocks together. As soon as a block is found, the reward is split among the contributors, proportional to their expended processing power. This takes away the variance of block rewards for individual miners and allows receiving a constant income of bitcoins in exchange for processing power. In the past it has occurred that a mining pool has reached a share of processing power that almost exceeded 51% of the total processing power. Fortunately miners as well as pool operators themselves have an incentive to prevent a single entity to gain control over the network in order to maintain trust in the Bitcoin network and keep bitcoins valuable. Therefore the miners usually switch to a smaller mining pool as soon as the processing power of a mining pool nears 51%. One mining pool operator that came close to 51% of processing power even urged it’s clients to switch to a different mining pool [27].

2.2 Real-World Usage of Bitcoin

The real-world usage of Bitcoin (meaning the usage of Bitcoin in a physical setting, excluding online-shops or trading platforms etc.) is sparse and the adoption rate is mediocre at best. Bitcoin is far from being a standard payment method in physical brick-and-mortar

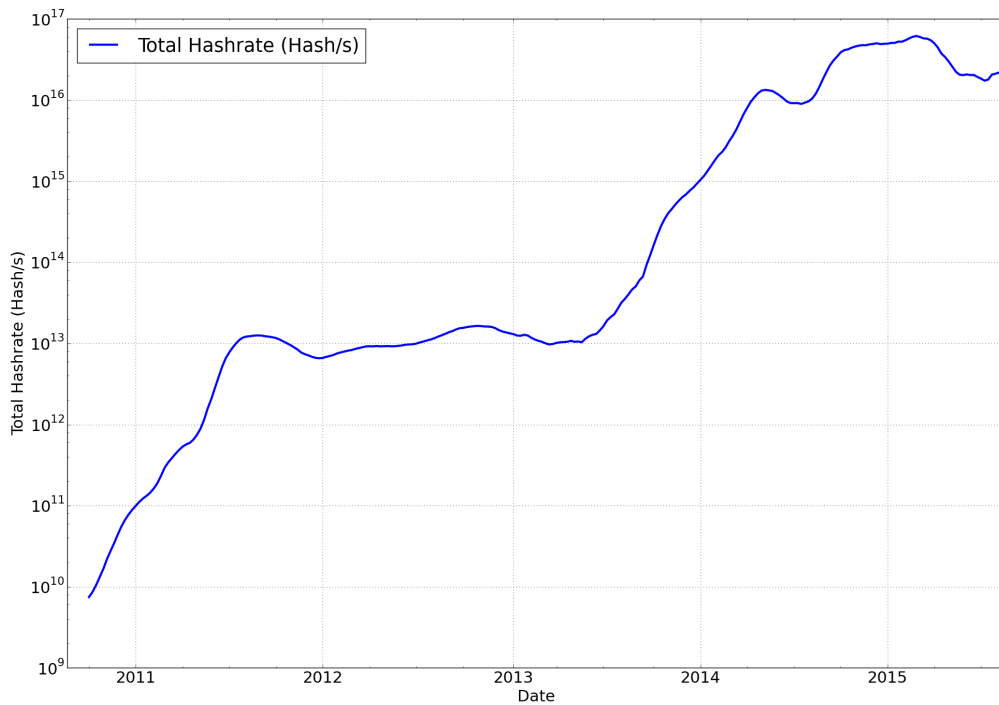


Figure 2.1: Estimated amount of hashes per second computed in the Bitcoin

stores. Even after years of Bitcoin's existence the world-wide number of stores accepting bitcoins is still below 7000, and the adoption rate is slowing down (see Figure 2.2) [24].

Despite Bitcoin having proved to be secure and less expensive than accepting credit cards, Bitcoin adoption in real-world stores is not taking off. A reason for this could be that Bitcoin payments are not yet convenient enough, fast enough or that merchants fear the volatility of the digital currency.

2.2.1 Legal Challenges

In Switzerland, payment system providers are regulated by the FINMA. According to the FINMA, a company moving and managing a certain amount of someone else's assets is considered a financial intermediary. Financial intermediaries are subject to anti money-laundering laws (AML) and need to be part of a self-regulatory organization (SRO) in order to legally operate [6]. According to a letter by the Federal Assembly, Bitcoins are considered an asset [7]. Consequently, if users of the original CoinBlesk implementation 1.0 held a certain amount of Bitcoins in their accounts, the CoinBlesk implementation would have been considered illegal under the Swiss banking law.

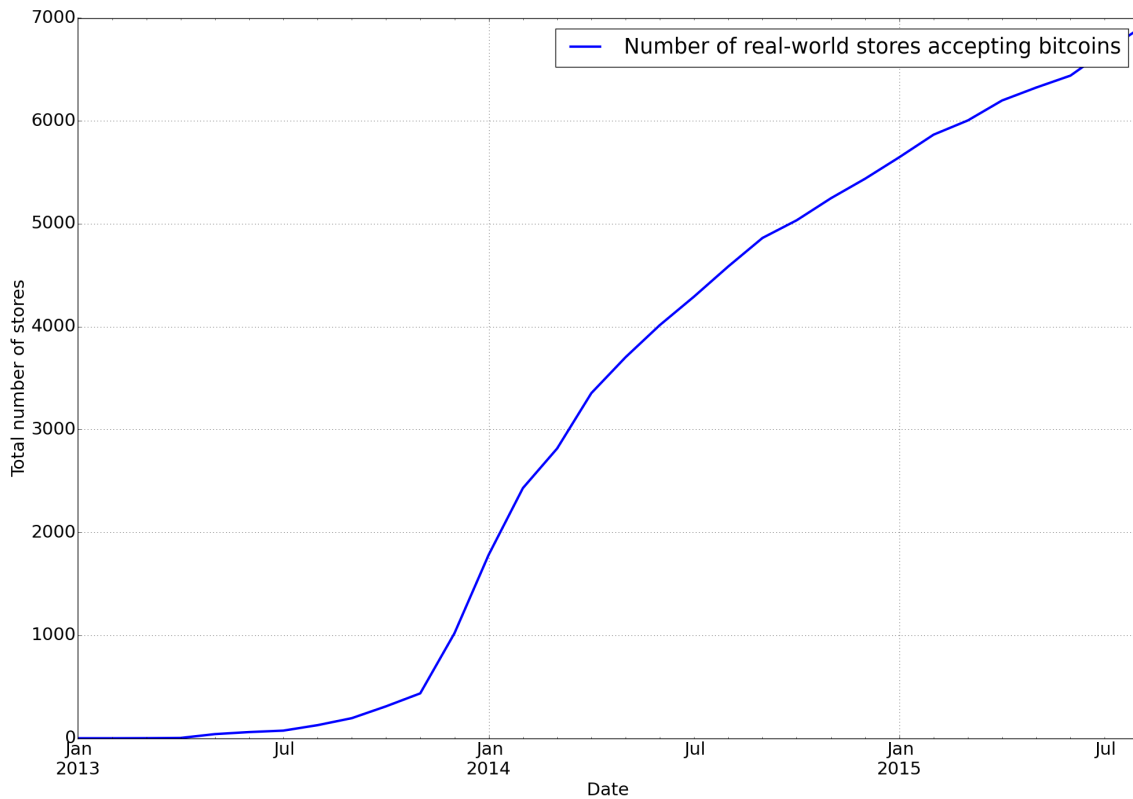


Figure 2.2: World-wide count of real-world stores accepting bitcoins between January 2013 and July 2015

2.2.2 Technical Challenges

As mentioned in Section 2.2.1, according to the Swiss banking law, managing someone else's funds without being part of an SRO is considered illegal. This means that in order to operate a Bitcoin payment system, the operators of the payment system cannot be in full control of the user's funds. Yet, if the user himself is in total control of his bitcoins, there is no way to stop him from attempting a double-spending attack. According to Satoshi Nakamoto, on average only after one hour a double-spend transaction can be ruled out with near 100% certainty [2]. This is clearly not feasible for a brick-and-mortar store. So the objective is to find a solution in the middle, where both the payment system provider *and* the user have some control over the bitcoins, but not full control.

Another challenge is securing the client's bitcoins. A company or server managing bitcoins is a magnet for hackers, as a breach of the security can yield high rewards. Again this problem is alleviated to a certain extent by sharing the control of the bitcoins between the payment system operator and the client as an attacker would have to gain access to both the payment operator's and the client's machine.

2.2.3 Usability Challenges

Bitcoin in its pure form is not human-friendly. E.g. bitcoin addresses are strings of random characters, containing both upper- and lowercase letters and numbers. Prices in traditional currency converted to bitcoins are numbers with up to 8 decimal places, so when a customer buys a coffee for USD 5, the bitcoin accepting waiter will ask for e.g. 0.01961946 bitcoins.

A challenge of developing a Bitcoin payment system is to abstract the inner workings of Bitcoin to the point where it's a convenient payment option not just for above average tech-savvy people. The Bitcoin payment system has to abstract the underlying workings of Bitcoin so that even people who have never heard of Bitcoin before are able to use it for payment.

2.3 Requirements for a Bitcoin Payment System

In this section three key requirements for a Bitcoin payment for system are outlined. The focus lies on requirements for Swiss merchants, but it can be assumed that similar if not identical requirements are applicable for other countries.

2.3.1 Mitigating Risk of Volatility

Compared to traditional currency, the price of bitcoins fluctuates wildly. Within January 1st 2014 and December 31st 2014, the peak of the bitcoin price was USD 985.68 and the lowest price USD 316.90. [3]. In the year 2013 the price of a bitcoin decreased over 80% in less than one week [4]. For merchants that have no interest in holding risky assets, bitcoin is not a feasible means of storing value, especially if operating on low margins.

A Bitcoin payment system for real-world, brick-and-mortar stores therefore has to protect the merchants from these price fluctuations. One way to solve this problem would be to hedge against a price fall by buying bitcoin options, but currently the market for such financial products in the bitcoin space is not mature enough. There is an exchanges that offer a market for bitcoin options, but the trading volume is tiny or not existent at all [5].

2.3.2 Compliance with Swiss Banking Law

As mentioned in section 2.2.1, according to the Swiss banking law a payment system operator usually needs to be a licensed financial intermediary in order to legally operate. Therefore, in order to operate a payment system such as CoinBlesk, it is required that the operators of the payment system are not actually managing the funds of its users. The users of the payment system still have to be able to reclaim their bitcoins in any case. Also it cannot be possible for the operator of the payment system to move their user's funds without their consent.

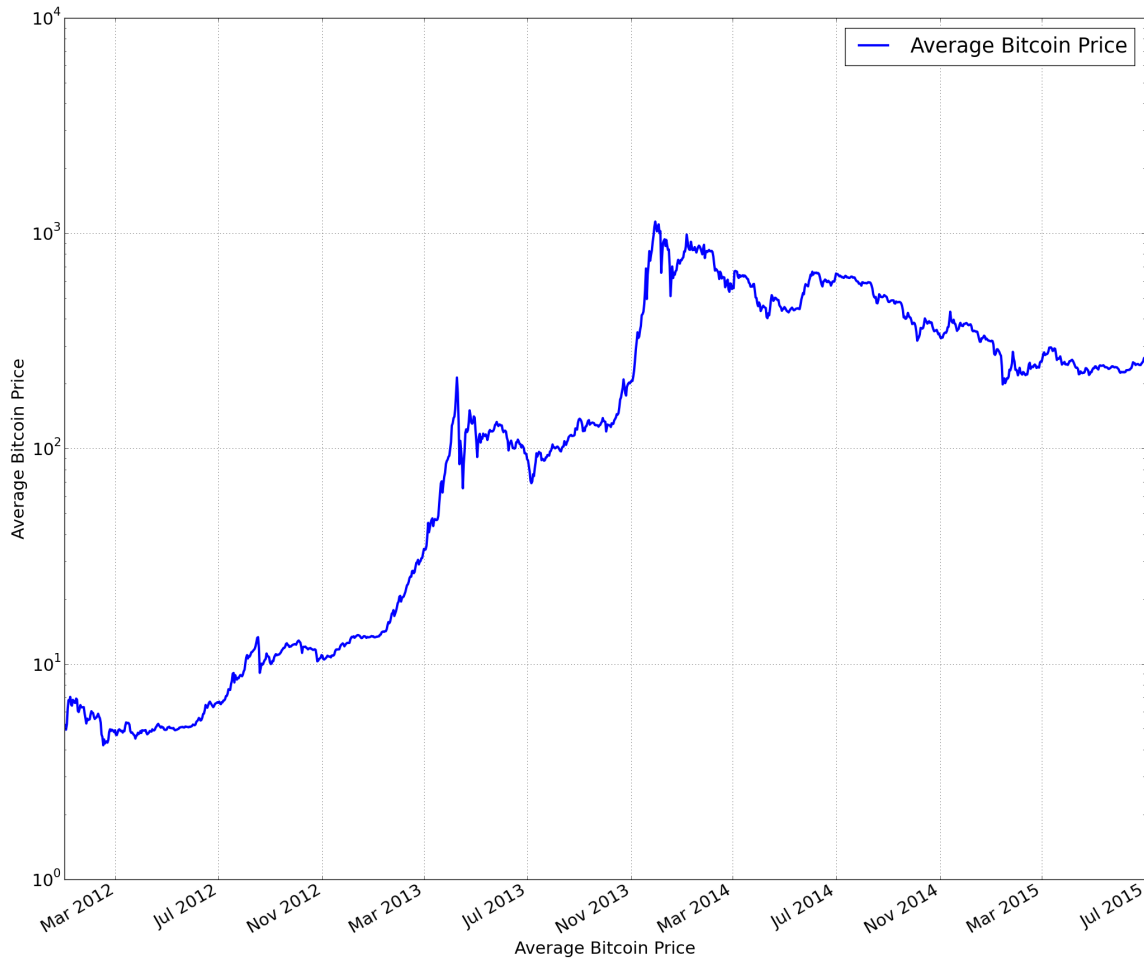


Figure 2.3: Average bitcoin price in USD between January 2012 and July 2015

2.3.3 Performance and Convenience

Paying cash works well and has proved to be a relatively convenient means of payment in the last centuries. A new payment system that replaces cash should bring additional advantages over cash. One advantage that an innovative payment system can bring to the table is the *speed* of processing a transaction. The faster a payment is processed at the counter of a store, the less time customers spend waiting in line. At the same time, the merchant has an incentive to minimize payment times, in order to decrease the number of employed cashiers and save on wages.

Even though cash is still the preferred way of payment in Switzerland, cashless payment options are slowly adopted by the Swiss. Today, cashless payment options such as credit and debit card account for almost 40% of the total transaction volume [8] [9]. It can be assumed that an important cause of this rise is due to the increased transaction speed and convenience of cashless payments. Exceeding the speed and convenience of cash payments is not a real challenge, therefore the Bitcoin payment system should aim at being even more convenient and faster than current cashless payment systems.

2.4 Existing Solutions

CoinBlesk 2.0 is not the first payment system addressing the requirements for a Bitcoin payment system for real-world merchants. In this section, existing solutions are shortly described and how they meet some the requirements outlined in Section 2.3.

2.4.1 GreenAddress: Shared Control of Bitcoins

GreenAddress is a wallet provider that already fully solves the requirement described in section 2.3.2 (Compliance with Swiss Banking Law) and partially solves the requirement described in section 2.3.3 (Performance and Convenience). It does so by giving the user only partial control over his funds, but providing a so-called “refund transaction” to ensure the user that the funds are reclaimable in any case, even in the case of total bankruptcy or disappearance of GreenAddress. Furthermore, sharing the control over funds provides an additional layer of security for the user. A hacker would have to gain access to the private keys of both GreenAddress and the GreenAddress user in order to steal bitcoins [15].

The design of the new CoinBlesk 2.0 implementation is inspired by GreenAddress. Just as GreenAddress, the control of the users funds are shared between the user’s CoinBlesk app and the CoinBlesk server. The design of the CoinBlesk system is described in detail in Chapter 3.

2.4.2 BitPay: Instant Currency Swap and NFC Payments

BitPay is a company offering Bitcoin merchant solutions and end-user wallets. Their core business is converting bitcoins to traditional currency instantly after receiving bitcoins, and hereby protecting merchants from the volatility of bitcoins. For offering this service, BitPay raises a small fee of 1%.

Following the footsteps of Apple, BitPay recently added NFC support to their PoS (Point of Sale) software, allowing users to pay a bill with their Android wallet with only a single tap [17]. Though in the case of BitPay, NFC is a mere way to transmit the data traditionally provided by a QR-Code, containing the payment address, amount and perhaps a transaction memo. The Android app does not transmit a complete Bitcoin transaction over NFC [16]. The payment system also doesn’t protect merchants from a double-spend attack. Contrary to popular believe among the Bitcoin community, the merchant still carries the risk of a double-spend attack, not BitPay [18].

Chapter 3

Design

In this Chapter, the current implementation of the new CoinBlesk 2.0 payment system is explained and how it solves the requirements outlined in section 2.3.

3.1 Client-Server Architecture

Bitcoin itself is a P2P network, but CoinBlesk 2.0 follows a Client-Server architecture. The responsibilities of the server are quite limited though. Contrary to the original CoinBlesk 1.0 implementation [1], the server has no real notion of user accounts. The responsibilities of the server boil down to:

1. Signing client's transactions
2. Keeping track of the inputs that have been signed
3. Keeping track of the lock-times of signed inputs
4. Keeping track of which unspent outputs are owned by the clients

The responsibilities listed above are explained in detail in the following sections.

The server acts as a trusted intermediary of the clients. In an ideal world, instant transaction would be possible without *any* kind of trusted third party, but as of today there is no viable option to do so for the use case of CoinBlesk. A technique that could be built upon to allow trust-less instant transactions between clients are payment-channels. But payment-channels are intended for only two parties at a time, and funds need to be *allocated* in a payment channel to work correctly. Expanding the payment-channel protocol and applying it to a network of peers is very complex and out of the scope of this thesis. A project trying to create such a network of payment-channels is the Bitcoin Lightning Network, but the implementation is far from production ready. To be precise, the Lightning Network does not even work with the current Bitcoin protocol, due to transaction malleability [25].

3.2 Multi-Signature Setup

Before going into details of the CoinBlesk 2.0 Multi-Signature setup, a closer look is taken on the multi-signature capabilities of the Bitcoin protocol.

3.2.1 The Property of Bitcoin and the Power of Script

Compared to the traditional banking system, the notion of “property” is more complex in the Bitcoin world. While in the traditional banking system, money usually belongs to a single person or a group of people, the property of bitcoins is defined in a much more fine-grained way. Thanks to a Forth-like bitcoin scripting language (called “Script” [10]), almost arbitrary property-schemes can be created. For example, the redemption of bitcoins can be tied to the condition that at least two out of three signatures of a fixed set of three private keys are provided in order to spend bitcoins. In theory, the redemption of bitcoins can even be tied to an external state, such as the death of a relative (inheritance) or the price of a company’s share (option). Schemes like this are called “Contracts” in the Bitcoin space [11]. The Script defining the redemption conditions is called *RedeemScript*.

3.2.2 P2SH Addresses

In the original Bitcoin implementation by Satoshi Nakamoto, Script was already supported to some degree. Back then, the Script was part of a transaction output, which meant that the sender of the bitcoins had to specify the conditions for redemption of the funds [2]. In the case of multi-signature transactions, the sender of the bitcoins had to specify all the public keys of the receivers private keys [13]. This may be useful in some cases, but usually the payer does not care or know how exactly the funds are redeemed. It also meant that wallets had to implement special capabilities to create multi-signature transactions. Because of this shortcoming of the Bitcoin protocol, the lead developer of Bitcoin Core - Gavin Andresen - proposed the development of a new kind of standard transaction: The Pay to Script Hash (P2SH) transaction [12].

P2SH transactions shift the responsibility of defining the redemption conditions from the payer to the payee. At the time of payment, all the payer sees is a 160 bit hash of the Script. At the time of the redemption of the bitcoins sent to a P2SH address, the payee is responsible of providing the full Script, which is included in full in the ScriptSig of the input claiming the unspent output [12]. This has two main advantages:

- The payer doesn’t have to know the redemption conditions at the time of payment
- The redemption Script can be of almost arbitrary complexity, while the payer only has to know a 20 bytes long P2SH address that can easily be copied and pasted

These two advantages over the “traditional” multi-signature of the Bitcoin protocol have made multi-signature a feasible feature among companies offering Bitcoin services. As

of today, many Bitcoin wallets allow paying to P2SH addresses. As seen in Figure 3.1, adoption of this relatively new feature of the Bitcoin protocol is rising at a fast pace. As of August 2015, more than 9% of all bitcoins that have moved were moved to P2SH addresses. [14].

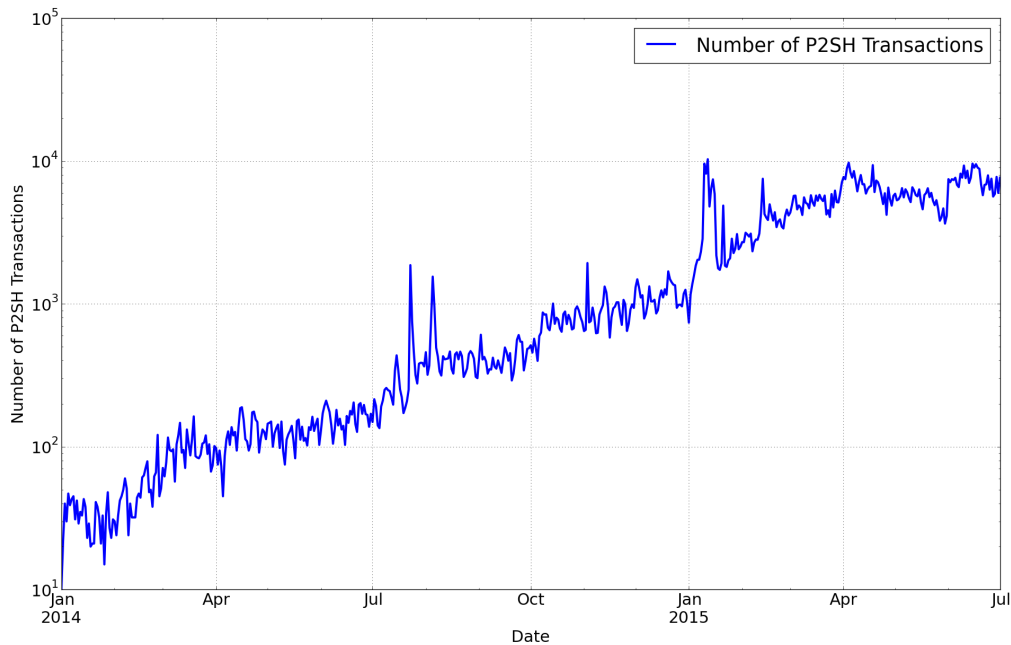


Figure 3.1: Number of P2SH transactions per day from January 2014 to July 2015

When the user deposits bitcoins to the CoinBlesk 2.0 app, the only deposit address is a P2SH address. The Script behind this P2SH address is a 2-of-2 multi-signature script. In other words, 2 different private keys are required to spend the bitcoins that are received on the CoinBlesk app. One of those private keys is stored on the Android device, the other key is stored on the CoinBlesk server. Whenever a CoinBlesk user creates a Bitcoin transaction, the transaction is first signed with the private key stored on the user's device, and afterwards by the CoinBlesk server. Without the signature of the server, the user is not able to spend the bitcoins. At first, this may sound alarming. What if the CoinBlesk server disappears? What if the private key of the server is accidentally deleted? The history of bitcoin service providers is not exactly calming, but the CoinBlesk system provides a feature that ensures its users that they will still be able to re-claim their bitcoins, even if the server's private key is deleted. This is achieved by providing a pre-signed, time-locked refund transaction to the user (see Section 3.3).

3.2.3 Hierarchical Deterministic Wallets

Hierarchical Deterministic Wallets (HD-Wallets) are a way to derive private/public key pairs deterministically from a seed. Traditionally, every private key is generated randomly on demand, and in order to back up a wallet, all private keys had to be backed up. Since best practice is to generate a new key pair for every transaction, old and heavily used

wallets had to back up long lists of private keys, which made it a tremendously tedious task to re-import the private keys into a wallet if the private keys were printed out and not stored on digital storage. With HD-Wallets, all the user needs to back up is the *seed*, and derive all private keys from it. With BIP32, the seed and key derivation has been standardized, so that a seed of one wallet works with other wallets as well [19].

HD-Wallets have a notion of paths. A path defines the “position” of a derived key. In other words, given the seed of a HD-Wallet and a path, the same key is derived deterministically. This is one of the key properties of HD-Wallets.

The 128-bit seeds of HD-Wallets are usually represented in the form of mnemonics. A mnemonic is a number of words that can easily be written down on a piece of paper. The words can be converted to bits and vice-versa. A 128-bit seed is translated to 12 words, which is reasonably easy to write down or even remember.

Another useful property of HD-Wallets is that the private keys can be completely “stripped away” from the seed. This means that, given the public seed of an HD-Wallet, an almost arbitrary amount of public keys can be derived from this seed. One useful application of this is e.g. an online shop that can generate unique addresses for each customer, without requiring any access to the private keys. Another useful application of public seeds are married wallets.

“Married wallet” is a term coined by Mike Hearn, developer of the Java Bitcoin library `bitcoinj`. Married wallets are called like that because Mike Hearn had a use case in mind of a married couple that share a bitcoin wallet together (much like today’s widely used joint bank accounts, just without the bank). A married wallet is the combination of multiple seeds in a single wallet. When creating a new receive address, the seeds are used to derive new public keys. From the public keys a RedeemScript can be created that defines how many signatures are required to redeem the bitcoins. Even though the term “married wallet” suggests that this concept only works with two seeds (or two public keys), the same concept can be applied to wallets with up to 15 seeds. The number of signatures required to spend the bitcoins is flexible, the RedeemScript can define a minimum of 15 required signatures, or only one. Addresses like this are usually called M-of-N addressed, M being the number of signatures required to spend the bitcoins, N being the total number of keys. When a new receive address is requested from a married wallet, a new RedeemScript is generated, from which a P2SH address is created. The public keys contained in the RedeemScript of the P2SH address are derived from the seeds of the married wallet.

Married wallets are a key feature of the design of CoinBlesk 2.0. As described in section 3.2.2, CoinBlesk 2.0 relies on P2SH multi-signature addresses. These addresses are obtained from a married wallet. The CoinBlesk clients only have one private seed, the other seed is a public-only seed from the CoinBlesk server. The receive addresses generated by the CoinBlesk app are 2-of-2 multi-signature P2SH addresses, meaning that the CoinBlesk clients themselves are not able to spend bitcoins without the signature of the CoinBlesk server.

3.2.4 Preventing Double-Spending Attacks

Since the CoinBlesk clients themselves are not able to spend bitcoins without the signature of the CoinBlesk server, the CoinBlesk server can to some degree control what transactions can be issued by the clients. This way, the CoinBlesk server can prevent the clients from attempting double-spending attacks, and therefore accept 0-confirmation transactions immediately. This is possible by keeping track of which inputs have been signed, and refusing to sign an input again that has already been signed in the past. This makes it impossible for clients to create a double spending transaction.

The CoinBlesk server also has to keep track of the currently unspent outputs of the clients. Otherwise, a client could fool the CoinBlesk server with a transaction that spends bitcoins that were never in the possession of the client.

If above conditions are met, the CoinBlesk server can with absolute certainty confirm a transaction immediately, without waiting for confirmations. As long as the clients trust that the CoinBlesk server has no malicious intentions, they can safely accept 0-confirmation transactions given the confirmation of the server.

3.3 Refund Transactions

All funds deposited on the CoinBlesk 2.0 app are held in a 2-of-2 multi-signature address. A positive aspect of this fact is that even if the operators of the CoinBlesk server are maliciously intended, they will never be able to steal user's funds. Further, if the CoinBlesk server is hacked and the private keys stolen, the hackers would need to gain access to the user's private keys as well in order to steal the bitcoins. Yet it also means that if the CoinBlesk server was to suddenly disappear for some reason, the clients are no longer able to spend their bitcoins. This is a major problem, because the Swiss law requires customers of a payment service to be able to gain full access to their funds in any case, should the operator of the payment system go bankrupt, or in the case of a Bitcoin service, be hacked. Secondly, the clients of CoinBlesk need to trust the system not to disappear.

The solution to this problem are refund transactions. A refund transaction is a pre-signed, time-locked transaction, that sends all the client's funds to an address exclusively controlled by the client. In the case of CoinBlesk, a refund transaction is automatically created by the app as soon as a new unspent output is in the wallet, meaning whenever bitcoins are received, or a transaction is created. The app then takes all the unspent outputs, creates a single transaction sending all bitcoins to an address of a private key that is derived from the client's private seed. The client signs this transaction and sends it to the server. The server checks that the transaction is in fact time-locked, signs it, and sends the fully signed transaction back to the client. The client is now in possession of a valid, fully signed refund transaction that becomes valid as soon as the time-lock expires. In case the CoinBlesk server suddenly disappears, the client can broadcast the refund transaction and regain control over the bitcoins.

One downside of this solution is that the client, at the time of sending bitcoins, has no way of knowing whether the server will actually sign the refund transaction or not. If the CoinBlesk operators were maliciously intended, they could simply refuse to sign the refund transaction. But from a game-theoretic perspective, the CoinBlesk operators gain nothing from this. They will not be able to steal user's bitcoins at any case, because this would require the client to comply and offer his signature to create a valid transaction. To summarize: At the time of depositing bitcoins on the CoinBlesk app, there is a short time - before the user receives the signed refund transaction - where the user actually has to trust the server to provide his signature for a refund transaction. The fact that the user has to trust the server for this is the key distinction from the CoinBlesk multi-signature scheme compared to so called "(micro)-payment-channels". In the payment-channel protocol, the refund transactions are signed by the counter party *before* any bitcoins are sent to the multi-signature address. Such a completely trust-less protocol is of course desirable, but it would add a lot of complexity to the system, both for the user and for the development of the system, and after careful consideration it was decided not to follow this path.

There's one more caveat regarding the refund transactions. As soon as the time-lock of the refund transaction expired, the client is in possession of a valid transaction that could potentially be used for a double-spending attack. Assuming Alice has a valid refund transaction, she could send Bob a bitcoin and immediately after broadcast the now valid refund transaction. In that case there is no guarantee on which transactions will be confirmed by the miners, the refund transaction sending all bitcoins back to Alice, or the payment to Bob. Therefore, some precautions have to be put in place to prevent such an attack. The solution for this problem is described in the following section.

3.3.1 Re-Deposits

The CoinBlesk server keeps track of the inputs that have been signed and their lock-time. Whenever a client asks the server to sign a transaction, the server first checks if any of the inputs have already been signed and, more importantly, if the time-lock already expired. If that is the case - meaning the refund transaction has become valid - the server will refuse to sign the input again. In this case, the client *must* broadcast the refund transaction, because the server will never again sign a transaction of the client. Needless to say, this state is not desirable. To prevent a client ever ending up in this state, the client is responsible for *re-depositing* the funds from time to time. Re-depositing means creating a transaction that sends all bitcoins owned by a user to an address of the same user. This may seem pointless at first, but by re-depositing bitcoins, the refund transaction is effectively invalidated. Transaction inputs are pointers to outputs of other transactions, and by re-depositing bitcoins, the inputs of the refund transaction point to outputs of transactions that have already been spent, therefore invalidating the refund transaction.

The task of re-depositing bitcoins is managed by the CoinBlesk app and does not require any user interaction. When the time lock is nearing its expiration, the CoinBlesk app automatically re-deposits all the user's funds, so that the refund transaction never actually becomes valid. The CoinBlesk app even wakes up once every day to check if the time lock is nearing the expiration and, if necessary, re-deposits the client's bitcoins in the background, completely transparent to the user.

Should the CoinBlesk server disappear, the client could of course no longer re-deposit the bitcoins, meaning that as soon as the time-lock expired, the app will automatically broadcast the refund transaction, which is the desired behavior in this case.

The downside of refund transactions is that it comes with a cost for the user. Since Bitcoin transactions are not free of charge, every time the client's bitcoins are re-deposited, the client has to pay a small transaction fee. This fee can be seen as a cost for security, much like a bank charging clients for a safety deposit box. But compared to the annual fee of CHF 200 for a safety deposit box at a Swiss bank [28], the annual fees for redepositing of bitcoins is negligible.

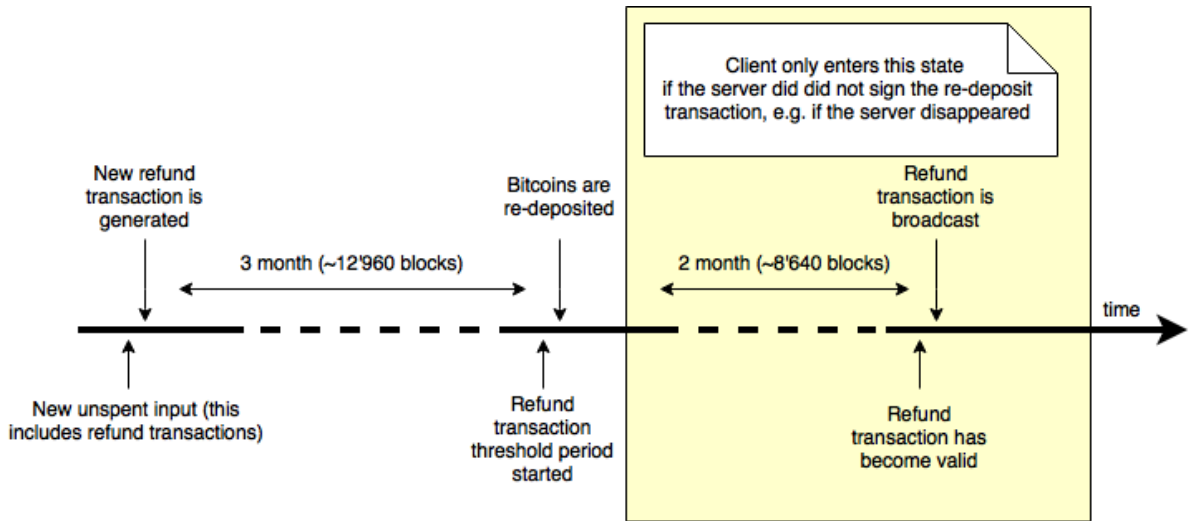


Figure 3.2: Time-line of refund transactions and re-deposits

3.4 NFC Payment Protocol

A key feature of the CoinBlesk system is its NFC capability. Not only allows CoinBlesk to pay instantly via NFC, it even allows payment if either the payer or payee - depending on if it's a payment or send request - is not connected to the Internet. This is especially useful for tourists not willing to pay high roaming fees or at places where Internet reception unreliable or not present at all. The following sections describe the implemented NFC payment protocol.

The NFC payment protocol covers two different use cases: One where a user requests money from an other user, and one where a user wants to send money to another user. The former case is henceforth called "payment request", the latter case is called "send request".

The NFC communication is based on a custom two-way NFC library for Android, developed by Thomas Bocek and Jeton Memeti[20].

3.4.1 Payment Request

Let's assume Alice requests one bitcoin from Bob and Bob is willing, and in possession of enough unspent bitcoins, to fulfill Alice's request. In this case, the protocol works as follows:

1. Alice starts *initializing*, which means that other NFC devices can now discover Alice's device's NFC tag.
2. Bob's device discovers Alice's NFC tag.
3. Alice sends a payment request to Bob, containing her public key, her user name, her bitcoin address and the amount of bitcoins she requests.
4. Bob checks the payment request and either rejects it and replies with a NOK, or accepts it and replies with a half-signed transaction, sending the amount requested by Alice to Alice's Bitcoin address.
5. Alice sends a signature request to the CoinBlesk server, containing the half signed transaction provided by Bob.
6. The CoinBlesk server checks the transaction and, if valid, signs the transaction, broadcasts it to the Bitcoin network, and sends the fully signed transaction back to Alice.
7. Alice is now sure that she will receive the requested bitcoins. She sends the fully signed transaction to Bob.

Figure 3.3 provides a graphical representation of the protocol described above.

Bob does not require a connection to the Internet for the payment. As all his transactions are stored on the device, he simply needs to create a transaction from his unspent outputs, sign it with his private key and send the half-signed transaction to Alice. A potential problem of that is that, if the connection breaks before Alice sends the payment confirmation to Bob, or if Alice is running a maliciously modified version of the CoinBlesk client that does not send any payment confirmations, Bob has no way of knowing whether his transaction was actually broadcast or not. Alice could then claim that she hasn't received the payment, and ask Bob to pay again. This is a risk that Bob has to bear in mind when paying without Internet access and assess for him self if he's willing to carry this risk or not.

3.4.2 Send Request

Assuming Alice possesses a Bitcoin and she wants to send it to Bob, the NFC protocol works as follows:

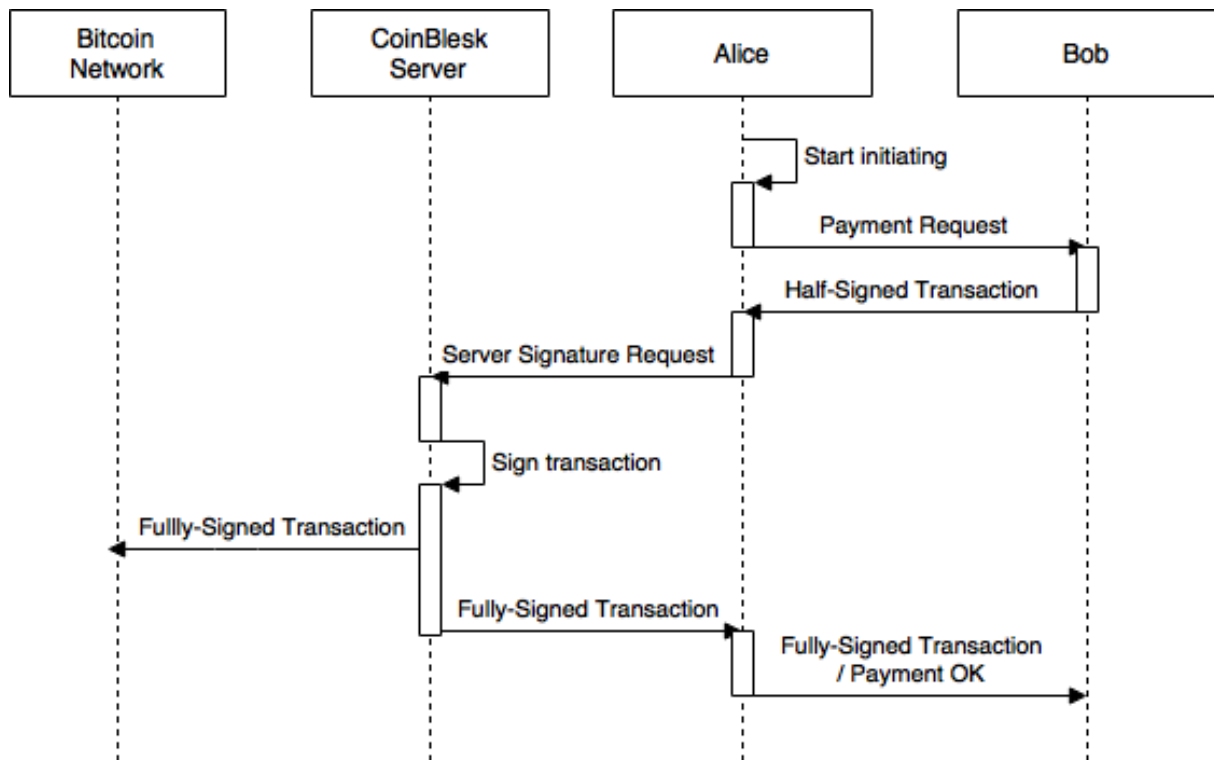


Figure 3.3: Sequence diagram of the NFC payment request protocol

1. Alice starts *initializing*, which means that other NFC devices can now discover Alice's device's NFC tag.
2. Bob's device discovers Alice's NFC tag.
3. Alice sends a send request to Bob, containing her public key and her user name.
4. Bob replies with his Bitcoin address, his public key and his user name.
5. Alice sends a signature request to the CoinBlesk server, containing a half signed transaction with an output sending the desired amount of bitcoins to Bob.
6. The CoinBlesk server checks the transaction and, if valid, signs the transaction, broadcasts it to the Bitcoin network, and sends the fully signed transaction back to Alice.
7. Alice sends the fully signed transaction to Bob.
8. Bob replies to Alice with a confirmation that the transaction has been received.

Figure 3.4 provides a graphical representation of the protocol described above.

Just as in the payment request scenario, only one party requires Internet access, namely the sender of the bitcoins. By extracting the signatures of the server from the inputs of the fully signed transaction and the public seed stored on Alice's device, Alice is able to verify that the transaction was in fact signed by the server. As long as Alice trusts the server, she can safely accept the payment and consider it instantly confirmed.

One caveat of the protocol, just as in the payment protocol, is that if the NFC connection breaks for some reason before the fully signed transaction was sent to the receiver, the receiver has no idea whether the transaction will arrive or not. Only after the receiver connected to the Internet and is notified about the transaction from the Bitcoin network this uncertainty is resolved.

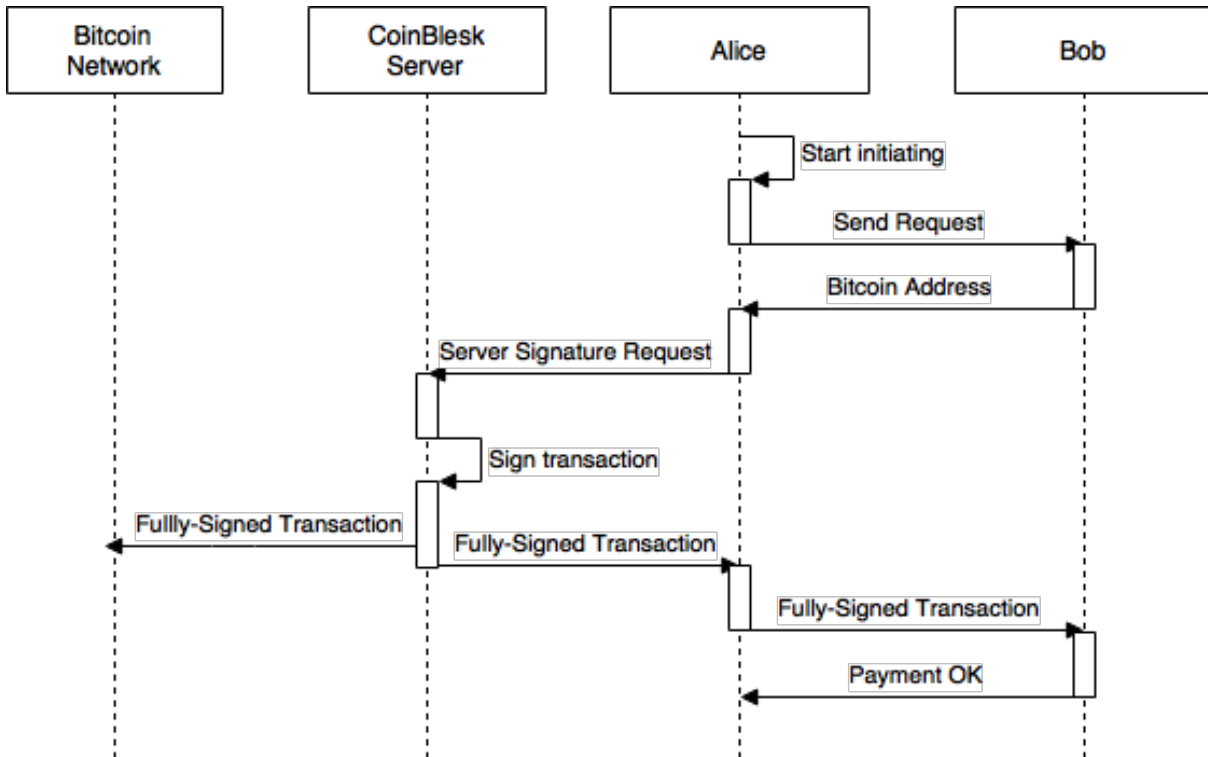


Figure 3.4: Sequence diagram of the NFC send request protocol

3.5 Instant Currency Swap

A merchant using the CoinBlesk payment system to accept payments can immediately swap bitcoins for traditional currency. If a user that has the “merchant mode” activated receives bitcoins, the equivalent amount of bitcoins is immediately sold on a Bitcoin exchange. The user also has the ability to set up a “backup exchange” that will be used if the sell order of the primary Bitcoin exchange failed for some reason.

Ideally, the bitcoins that are received on a device with the merchant mode activated would be sent to the exchange and then sold. Unfortunately instant transactions to exchanges are not possible; most exchanges require 3 confirmations before the bitcoins are accredited to the account. Consequently, for the merchant mode to work correctly, the CoinBlesk user needs a “buffer” of bitcoins deposited on the Bitcoin exchange, some of which are sold whenever the user receives bitcoins. If the bitcoin buffer on the exchange is used up, the instant currency swap is no longer functioning. Furthermore the user is responsible for manually sending bitcoins from the CoinBlesk app to the Bitcoin exchange at the end of the day in order to “fill up” the bitcoin buffer. In a future release of CoinBlesk, an

automatic filling of the bitcoin buffer may be a useful feature to minimize the effort for merchants to accept bitcoins.

Chapter 4

Implementation

In this Chapter, implementation details such as key parts of the application as well as the user interface of the CoinBlesk 2.0 system are described.

4.1 Wallet Service

The core of the CoinBlesk app is a wallet service that runs in the background. It is responsible for handling all bitcoin related tasks. This ensures that the sometimes quite CPU intensive bitcoin tasks are always run on a background thread and never on the UI thread. Further, the Bitcoin functionality is completely decoupled from the rest of the app, which in theory makes it possible to replace the bitcoin protocol implementation e.g. with a different Java Bitcoin library.

Responsibilities of the wallet service include:

- Creating Bitcoin transactions
- Providing information about the current balance
- Notifying listeners about bitcoin events, such as received transactions
- Providing information about the current block chain synchronization status
- Creating refund transactions
- Storing wallet activity on the device
- Saving transaction meta data to the local SQLite database of the app, such as the user names of recipients and senders of bitcoins, or whether a transaction was a NFC transaction
- Creating an mnemonic seed for backing up the wallet's bitcoins

Since the wallet service can be quite CPU intensive, it may take a few seconds until the service is running. The app is implemented in a way that the bitcoin functionality is only enabled as soon as the wallet service is ready. This means that e.g. when the user wants to send bitcoins, the send button is grayed out at first or the user's balance is not immediately visible, because the wallet service is not ready yet. For future versions of CoinBlesk, it may be worth caching some information of the wallet service, such as the user's balance, or trying to improve the performance of the wallet service in order to achieve a better, smoother user experience.

The original goal of this project was to create an app that is a light-weight client, without being a full fledged bitcoin client. The idea was to just store the private keys on the android device, and use these keys to sign transaction hashes that are provided by the CoinBlesk server. In theory this could work, but at the same time this would require a lot more trust in the CoinBlesk server:

- The server could create any transaction without the client knowing to which address the bitcoins are sent
- The server could create invalid refund transactions, and giving the clients a false sense of security

There is no way of fully validating Bitcoin transactions without being connected to the Bitcoin network or without the usage of a third party service. Since BitcoinJ is used for other, popular mobile wallets as well, the decision was made to not comply with the original goal of keeping the client as light-weight as possible and instead run BitcoinJ on the clients. BitcoinJ turned out to be quite resource-intensive and slowed down the app which has a negative impact on the user experience. In the end it's a trade-off between trusting the server and user experience. This is a very common trade-off among Bitcoin applications.

4.2 Main Features and Views

The primary goal by developing the CoinBlesk app was to make it as user-friendly as possible. The app intentionally does not offer any "Bitcoin-expert" functionality, such as viewing transaction details, importing private keys, etc. The functionality of the app is limited to what is needed in order to be a useful, compliant payment app, and nothing more.

The following Section give a short overview of the features and views the CoinBlesk app currently offers, and ideas how the functionality of the views could be further improved.

4.2.1 Home Screen

On the home screen, the user has an overview of the most important information:

1. The current progress of the block chain synchronization.
2. The current balance, both in bitcoins and in Swiss Francs
3. The most recent transactions

Below this information, a big button is shown which will redirect the user to the view described in Section 4.2.2 where the user can create a new payment or send request.

4.2.2 Create Transaction

The “Create new Transaction” view offers 3 buttons:

1. A “Send Bitcoins” button
2. A “Request Bitcoins” button
3. A “Send Bitcoins Directly to Known Receiver” button

The first two buttons are used to create an NFC payment or send request (see Section 3.4.1 and 3.4.2). The last button allows the user to send bitcoins to a known receiver from the address book.

4.2.3 Transaction History

The transaction history shows a list of all transactions of the user. It shows the amounts both in bitcoins and Swiss Francs. For each transaction, there’s also a small description of the transaction. For transactions that were received not via NFC but from the Bitcoin network, the number of confirmations is shown as long as the number of confirmations does not exceed 6 confirmations.

For NFC transactions from the CoinBlesk system, which are instantly confirmed, the number of confirmations is not shown, and the description makes clear that this transaction was an *instant transaction*. Further, the CoinBlesk transaction also shows from which user or to which user the bitcoins have been sent. This information is obtained from a SQLite database that stores meta data for each transaction.

4.2.4 Settings

The settings view exposes a number of basic settings to the user:

- The bitcoin unit in which all bitcoin amounts are displayed. The user can choose between BTC (bitcoins), mBTC (milli-bitcoins) and μ BTC (micro-bitcoins)

- The user name of the user
- A transaction fee for receiving a payment
- The amount of Swiss Francs to auto-accept a payment
- The number of transactions to show on the main view
- The wallet backup
- The merchant mode settings, which contains the API credentials for the exchange where the bitcoins are exchanged for traditional currency if the merchant mode is activated

Since users are internally identified by their public keys, changing the user name will not affect the address book entries of other clients. E.g. if Alice is a favored contact of Bob, and Alice changes her user name from “Alice” to “NotAlice”, Bob will still auto-accept payment requests from Alice.

The wallet backup item in the settings redirects the user to a separate wallet backup view. It offers two different functionalities: First, it shows the user his personal mnemonic seed that can be used to restore a wallet (all bitcoins and transaction history) in case the user lost his device, or switched devices. This mnemonic seed is extremely sensitive: Anyone that knows this mnemonic seed can steal all bitcoins of the user. Considering this sensitivity, a button to copy the mnemonic seed to the clipboard (as offered by the pay in view for the Bitcoin address of the user) is intentionally missing. Since the clipboard of the device is readable by any installed app, it poses a security risk to copy the mnemonic seed to the clipboard.

Even though there is a check mark to activate the merchant mode, the merchant mode is not actually activated until the user has entered all required API credentials for an exchange. The check mark is only there so that a user can disable the merchant mode but still keep the entered API credentials saved on the device. This allows quickly enabling and disabling the merchant mode. Details on the merchant mode are described in Section 3.5.

4.2.5 Pay In

On the pay in view, the user can see the bitcoin address where bitcoins need to be deposited in order to “top up” the CoinBlesk app with bitcoins. The user also has the ability to display a QR-Code of the bitcoin address, send an email containing the bitcoin address or simply copy the address to the clipboard of the device.

4.2.6 Pay Out

The pay out view provides the ability to send out bitcoins to an arbitrary Bitcoin address. This is a means of sending bitcoins out of the CoinBlesk system. The pay out view also

offers the feature to scan a QR-Code of a bitcoin address, so that the user doesn't have to type the bitcoin address of the receiver manually.

A future feature of the pay out view could be that the CoinBlesk app also understands the NFC tags of BitPay terminals (see section 2.4.2) and other NFC capable wallets.

4.2.7 Address Book

The address book view is a simple list of all CoinBlesk users that the app has interacted with via NFC. The user has the ability to “favor” a contact in the list, which allows auto-accepting payment requests by that user. There is also a button that allows removing all contacts in the list that are not favored.

A useful feature that is missing in the current implementation would allow the user to create a payment by tapping on an address book entry. In the current implementation, the user can only create a transaction to an other user in the address book in the send out activity (see Section 4.2.8).

4.2.8 Send Out

The send out view allows a user to send bitcoins to a user in the address book. The bitcoin address of the user in address book entry is not visible to the user but saved internally. The functionality of this view is very similar to the pay out view, only instead of sending bitcoins to a Bitcoin address, the bitcoins are seemingly sent to an other user of the CoinBlesk system.

4.3 Libraries & Tools Used

The most important Java libraries used for the CoinBlesk system are bitcoinj and the Spring Framework.

Bitcoinj a popular Bitcoin library. Bitcoinj offers a simplified payment verification mode (SPV mode), which makes it a viable option for mobile devices. SPV mode means that blocks are not downloaded in full, but only their headers. The blocks are only downloaded completely if they contain transactions that are relevant to the wallet's addresses, meaning transaction that either send from or receive bitcoins from an address that belongs to the wallet.

The Spring Framework is a Java library that offers countless useful tools when developing a Java application, such as dependency injection, database abstraction, web controllers, and many more. The Spring Framework is configured using Java Beans and Java Annotations.

Chapter 5

Evaluation

In this Chapter, the scalability and performance of CoinBlesk 2.0 is shortly assessed. Later, the original thesis goals, as imposed by the thesis supervisors, are listed and evaluated to what degree they have been fulfilled.

5.1 Scalability & Performance

The scalability of the CoinBlesk app has not been tested, so it is not clear if the payment system is ready for hundreds or thousands of clients. Since the server's most resource draining responsibility is to keep track of the client's unspent bitcoins, it can be assumed that this is the bottleneck of the server's scalability. There does not seem to exist an extensive scalability test of bitcoinj though, so any assumptions on the scalability of bitcoinj is difficult.

Regarding the fact that the scalability of CoinBlesk hasn't been tested, the CoinBlesk system has to be seen as a prove-of-concept, and not a production ready system.

Compared to the performance of the centralized CoinBlesk 1.0 version, the performance of CoinBlesk 2.0 has definitely suffered. While the CoinBlesk app used to be a simple client that merely sent commands to the CoinBlesk server, the CoinBlesk 2.0 app carries far more responsibilities and is a full-fledged bitcoin wallet. It therefore has to contain all the code to interact with the Bitcoin network. Furthermore, a lot more data has to be sent through NFC now compared to the original CoinBlesk 1.0 version. With the centralized approach, all that had to be sent through NFC was a payment command that contained a signature of the client sending the bitcoins, where now an almost complete bitcoin transaction, containing a RedeemScript for each input, has to be sent through the NFC channel. Depending on the number of inputs, the size of such a transaction could exceed 1 MB for a single transaction, which will be both sent and received through NFC on both clients.

This is where a future version of CoinBlesk could improve a lot. Instead of sending the complete bitcoin transaction over NFC, the clients should only send the signatures

required to spend the unspent outputs over NFC. This would require the server to keep track of the RedeemScripts though, which would add a layer of complexity to the current CoinBlesk server implementation. This feature was out of scope for this thesis.

5.2 Thesis Goals

The following table lists the thesis goals as listed in the thesis description and assesses the degree of fulfillment.

Goal 1	Become familiar with CoinBlesk
Result	Fully completed
Goal 2	Propose a design and discuss it with the supervisors
Result	Fully completed
Goal 3	Design and implement multisig with a micro payment channel and time lock in CoinBlesk.
Result	Partially completed: The CoinBlesk 2.0 implementation uses multi-signature and time-locked transaction to secure the client's bitcoins. However, the instant transaction feature of CoinBlesk 2.0 does <i>not</i> use the payment-channel protocol. After careful consideration together with the supervisor, it was decided that using the payment-channel protocol is not feasible for CoinBlesk 2.0
Goal 4	Source code needs to be open source, well documented, and readable
Result	Partially completed: Due to lack of time in the end of the thesis, some parts of the source-code remain not as well documented as they should be. The code is open-sourced and available on GitHub.
Goal 5	Use JUnit to test where possible
Result	Partially completed: Test coverage of the project is far from 100%. The major features, especially any code that handles bitcoins, is tested using JUnit.
Goal 6	Write a well-structured report that includes motivation, background information, related work, design decisions, implementation details, deep evaluation (also to related work), and conclusion. For the motivation, interesting aspects are the legal situation of holding/managing private keys in traditional online wallets. Does it become necessary to obtain a banking license for managing and holding these private keys, and is this still the case with multisig keys?
Result	Completed

Table 5.1: List of thesis goals and their state of completion

5.3 Thesis Result

The original description of the thesis reads as follows:

This thesis covers the design, implementation and evaluation of multisig with a micro payment channel and time lock in CoinBlesk. The focus is on a two out of two multisig address. Optionally, an automatic renewal of the micro payment channel after the lock time expired can be implemented. One of the challenges is to not use BitcoinJ on the client side (Android) and keep the Android client as light as possible. Thus, interaction with the Bitcoin network should be done on the server side only. Furthermore, the existing NFC channel from CoinBlesk should be reused. Adaptations to this NFC channel can be made with the help of the supervisors. Backwards compatibility is not required as not many users are using CoinBlesk at the moment. However, data needs to be migrated and reused in any case.

The same use case as in [1] should be covered as well: payment in the cafeteria. The buyer can top up his account with a multisig address. The server can check if this money is blocked for a specific period of time and accept (micro) transactions coming from the client. The server keeps these transactions and before the time lock expires, submits the last transaction to the Bitcoin network to claim the Bitcoins spent by the client. Since the server is advancing money, it is crucial that the transactions will be submitted. Thus the architecture has to be built to make the submission DoS resistant. The goal is to keep the CoinBlesk transaction below 1 second.

The result of this work is CoinBlesk 2.0 that can be used in real life. For that purpose, a demonstrator should be built that can be shown during Scientifica 2015 (4-6 September). Similar to the cafeteria test run, a tablet will be used for the cashier and external NFC transceiver is used to communicate with the client. This setup will be built with the help of the supervisors. Furthermore CoinBlesk relies on one exchange site: Bitstamp. History has shown that those exchange sites are not 100% reliable (e.g., 1 week outage of Bitstamp). An optional task is to make CoinBlesk more reliable by including other exchange sites.

The developed CoinBlesk 2.0 system meets most of the criteria described above. The system is ready for the Scientifica 2015 test run and will be used by the visitors. All client funds are exclusively “stored” on 2-of-2 multi-signature addresses. Bitstamp is no longer the only exchange used for the currency swaps, an additional backup exchange can be specified in the app settings.

Yet, the CoinBlesk 2.0 diverted from some of the goals specified in the thesis description. Mainly the fact that BitcoinJ is running on the Android devices and that the system does not implement the payment-channel protocols. The reasons for this have been elaborated in the previous chapters.

Chapter 6

Summary and Conclusions

During the course of six month CoinBlesk 2.0, an open-source, ready-to-use Bitcoin payment system with NFC capabilities has been developed. It is the upgrade of the Bitcoin payment system CoinBlesk 1.0 that has been developed by students if the University of Zurich in 2014. The Android app can be downloaded from the Google Play store. The code is available on GitHub [26].

The CoinBlesk 2.0 implementation proves that Bitcoin can be used securely and conveniently in real-world environments and that Bitcoin as a payment technology has the potential of making mobile payments more secure, more private and more convenient than commonly used payment systems.

CoinBlesk 1.0	CoinBlesk 2.0
Server-Client Architecture	P2P/Server-Client Hybrid Architecture
Server has full control over bitcoins.	Server and client have shared control over bitcoins.
Client cannot claim bitcoins if server is unavailable.	Client can reclaim bitcoins with refund transactions even if the server is unavailable.
Sign up required.	No sign up required. Server has no notion of accounts.
Compliant with Swiss banking <i>to a certain point</i>	Fully compliant with Swiss banking laws
Commands with signatures are sent via NFC to send bitcoins.	Actual Bitcoin transactions are sent via NFC.
Bitcoin transactions are <i>off-chain</i> , server responsible for accounting.	Bitcoin transactions are registered in the block chain. Inconsistencies are (almost) impossible.
Single exchange is used for selling bitcoins.	Fallback exchange can be set in case the primary exchange is shut down.

Table 6.1: Comparison of CoinBlesk 1.0 to CoinBlesk 2.0

Bibliography

- [1] Jeton Memeti, Mehmet Ali Bekooglu, Simon Kaeser; Bitcoin in Practice; Master Project at University of Zurich, 3/2014
- [2] Satoshi Nakamoto: Bitcoin: A Peer-to-Peer Electronic Cash System, <https://bitcoin.org/bitcoin.pdf>, 10/31/2008
- [3] bitcoinaverage.com, <http://bitcoinaverage.com>, 8/8/2015.
- [4] cryptowat.ch, <http://cryptowat.ch>, 8/8/2015.
- [5] coinut.com, <http://coinut.com>, 8/8/2015.
- [6] FINMA: Rundschreiben Finanzintermediation nach GwG, <https://www.finma.ch/de/~media/finma/importiertedokumente/regulierung/anhoerungen/26-rundschreiben-finanzintermediation-nach-gwg/entwurf-rs-finanzintermediation-nach-gwg-20100611-de.pdf?la=de>, 2010
- [7] Swiss Confederation: Bericht des Bundesrates zu virtuellen Whrungen in Beantwortung der Postulate Schwaab (13.3687) und Weibel (13.4070), <http://www.news.admin.ch/NSBSubscriber/message/attachments/35361.pdf>, 6/25/2014
- [8] The Swiss National Bank: Bargeldloser Zahlungsverkehr und Geldausgabeautomaten, http://www.snb.ch/ext/stats/statmon/pdf/defr/C2_Kundenzahlungsverkehr.pdf, 2015
- [9] The Swiss National Bank: Noten- und Mnzumlauf, http://www.snb.ch/ext/stats/statmon/pdf/defr/A2_Noten_u_Muenzumlauf.pdf, 2015
- [10] BitcoinWiki: Script, <https://en.bitcoin.it/wiki/Script>, 8/9/2015
- [11] BitcoinWiki: Contract, <https://en.bitcoin.it/wiki/Contract>, 8/9/2015
- [12] Gavin Andresen: Pay to Script Hash, <https://en.bitcoin.it/wiki/Contract>, 3/1/2012
- [13] Bitcoin Developer Guide: Multisig, <https://bitcoin.org/en/developer-guide#multisig>, 8/9/2015
- [14] p2sh.info, <http://p2sh.info/>, 8/9/2015

- [15] GreenAddress.it: P2SH 2nd Address Wallet service implementation, <https://ghgreenaddress.files.wordpress.com/2014/04/greenaddressp2sh2of2hd-61.pdf>, 8/11/2015
- [16] Reddit, /u/WACOMalt, https://www.reddit.com/r/Bitcoin/comments/2l9l19/bitpay_brings_nfc_to_bitcoin_in_new_bitcoin/clteo1v, 11/2014
- [17] BitPay: Bitcoin Checkout: One-Tap Mobile Bitcoin Payments, <https://blog.bitpay.com/bitcoin-checkout-one-tap-mobile-bitcoin-payments/>, 11/2014
- [18] BitcoinTalk, user Steve <https://bitcointalk.org/index.php?topic=135985.msg1451602#msg1451602>, 11/2014
- [19] Pieter Wullie, Hierarchical Deterministic Wallets <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>, 2/11/2012
- [20] GitHub: coinblesk/android-kitkat-nfc-library <https://github.com/coinblesk/android-kitkat-nfc-library>, 2/11/2012
- [21] Raphael Voellmy: Averting the tragedy of the Blockchain http://www.academia.edu/8865335/Averting_the_tragedy_of_the_Blockchain, 12/1/2013
- [22] bitcoin.org: 11/12 March 2013 Chain Fork Information <https://bitcoin.org/en/alert/2013-03-11-chain-fork>, 5/16/2013
- [23] BitcoinWiki: Common Vulnerabilities and Exposures, CVE-2010-5139 https://en.bitcoin.it/wiki/Common_Vulnerabilities_and_Exposures#CVE-2010-5139, 5/16/2013
- [24] coinmap.org <https://coinmap.org/>, 8/15/2015
- [25] Joseph Poon, Thaddeus Dryja: The Bitcoin Lightning Network <https://lightning.network/lightning-network-paper.pdf>, 7/17/2015
- [26] GitHub: CoinBlesk <https://github.com/coinblesk>, 8/17/2015
- [27] Coindesk: Bitcoin Miners Ditch Ghash.io Pool Over Fears of 51% Attack <http://www.coindesk.com/bitcoin-miners-ditch-ghash-io-pool-51-attack/>, 1/9/2014
- [28] Credit Suisse: Save Deposit Box <https://www.credit-suisse.com/media/pb/docs/ch/privatkunden/schrankfach-1134-en.pdf>, 25/8/2015

Abbreviations

SRO	Self-Regulatory Organization
AML	Anti Money-Laundering
P2P	Peer-to-Peer
P2SH	Pay to Script Hash
PoS	Point of Sale
BIP	Bitcoin Improvement Proposal
SPV	Simplified Payment Verification
DAO	Data Access Object
DoS	Denial of Service

Glossary

FINMA Swiss Financial Market Supervisory Authority

Double-Spend Attack Spending the same bitcoins to two different parties at the same time

Forth An imperative stack-based computer programming language

Bitcoin Core The reference implementation of the Bitcoin protocol. Also called “Bitcoin Reference Client” or “BitcoinD”

HD-Wallet Hierarchical Deterministic Wallet

bitcoinj A Bitcoin library written in Java

Script A Forth-like Bitcoin scripting language

RedeemScript A Script that defines conditions to redeem unspent bitcoins

0-Confirmation Transaction A Bitcoin transaction that has been broadcast to the Bitcoin network, but not included in a block

Mnemonic Seed A list of words that can be converted to bits. From these bits, private/public key pairs can be recreated deterministically. Mnemonic seeds are standardized and are used to back up bitcoin wallets.

Sybil-Attack An attack on P2P networks where identities are forged by a single entity in order to fake a certain consensus within the network

(Micro-)Payment-Channel A protocol that allows trust-less instant off-chain (micro-)transactions

Off-Chain Not reflected on the block chain

Block Chain A public ledger containing every Bitcoin transaction that ever occurred

Transaction Malleability Changing the signature of a Bitcoin transaction in a way that the transaction is still valid but has a different transaction hash

List of Figures

2.1	Estimated amount of hashes per second computed in the Bitcoin	5
2.2	World-wide count of real-world stores accepting bitcoins between January 2013 and July 2015	6
2.3	Average bitcoin price in USD between January 2012 and July 2015	8
3.1	Number of P2SH transactions per day from January 2014 to July 2015 . . .	13
3.2	Time-line of refund transactions and re-deposits	17
3.3	Sequence diagram of the NFC payment request protocol	19
3.4	Sequence diagram of the NFC send request protocol	20
A.1	Main view when synchronizing with the block chain	45
A.2	Main view with transactions	46
A.3	Choose payment view	47
A.4	Payment request view	48
A.5	transaction history view	49
A.6	Settings view	50

List of Tables

5.1	List of thesis goals and their state of completion	30
6.1	Comparison of CoinBlesk 1.0 to CoinBlesk 2.0	33

Appendix A

Screenshots of the Android App



Figure A.1: Main view when synchronizing with the block chain

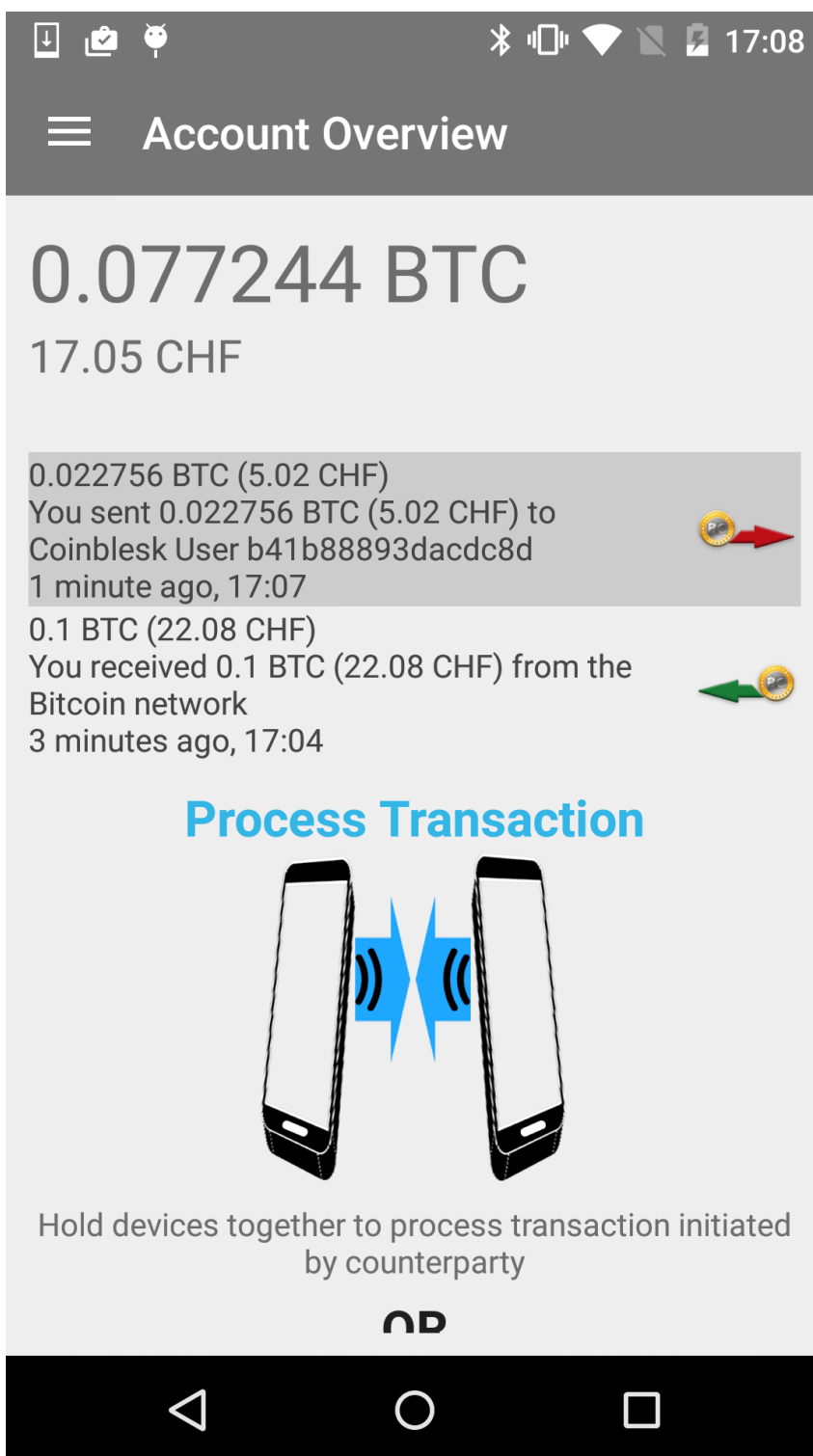


Figure A.2: Main view with transactions

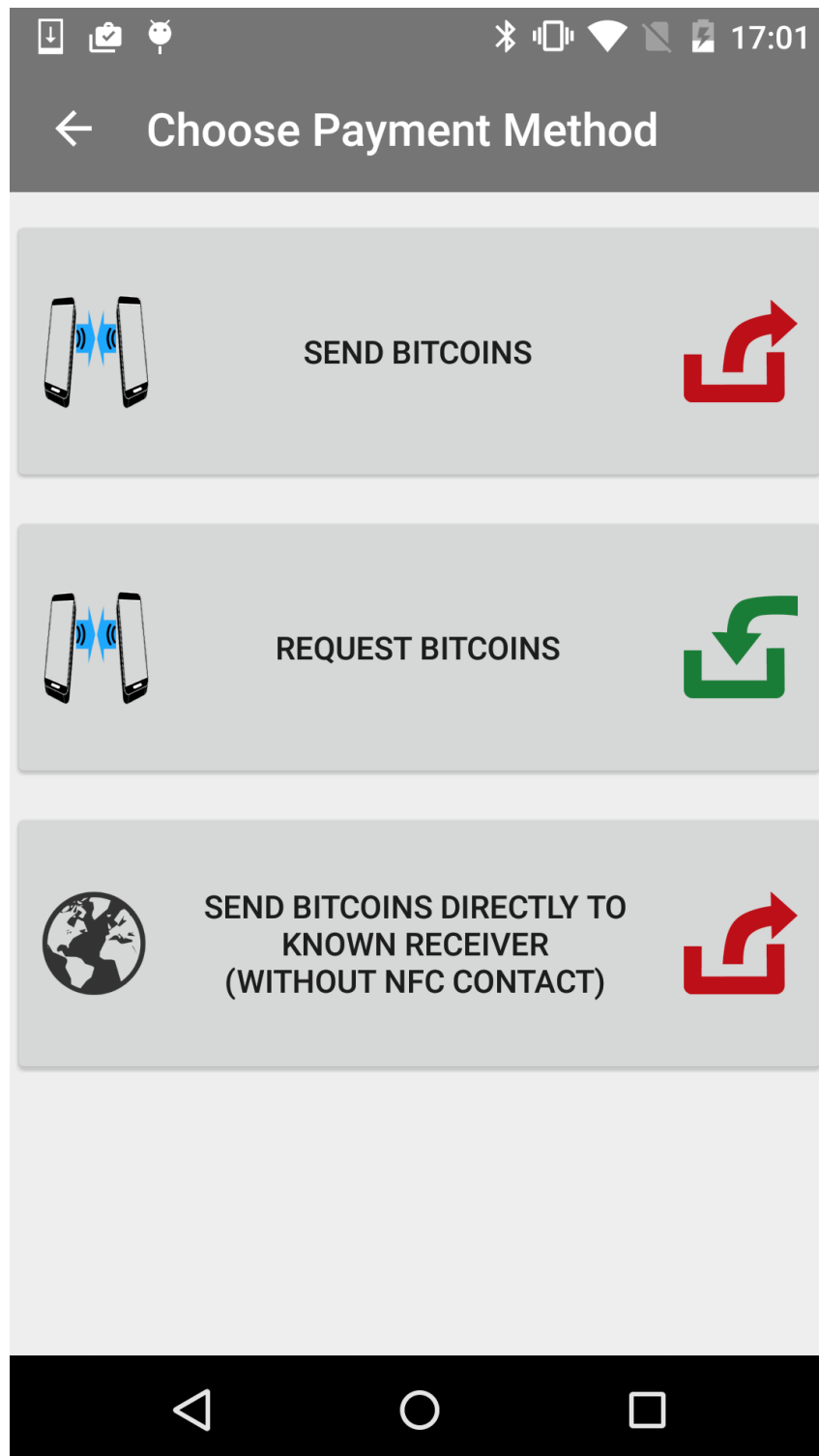


Figure A.3: Choose payment view

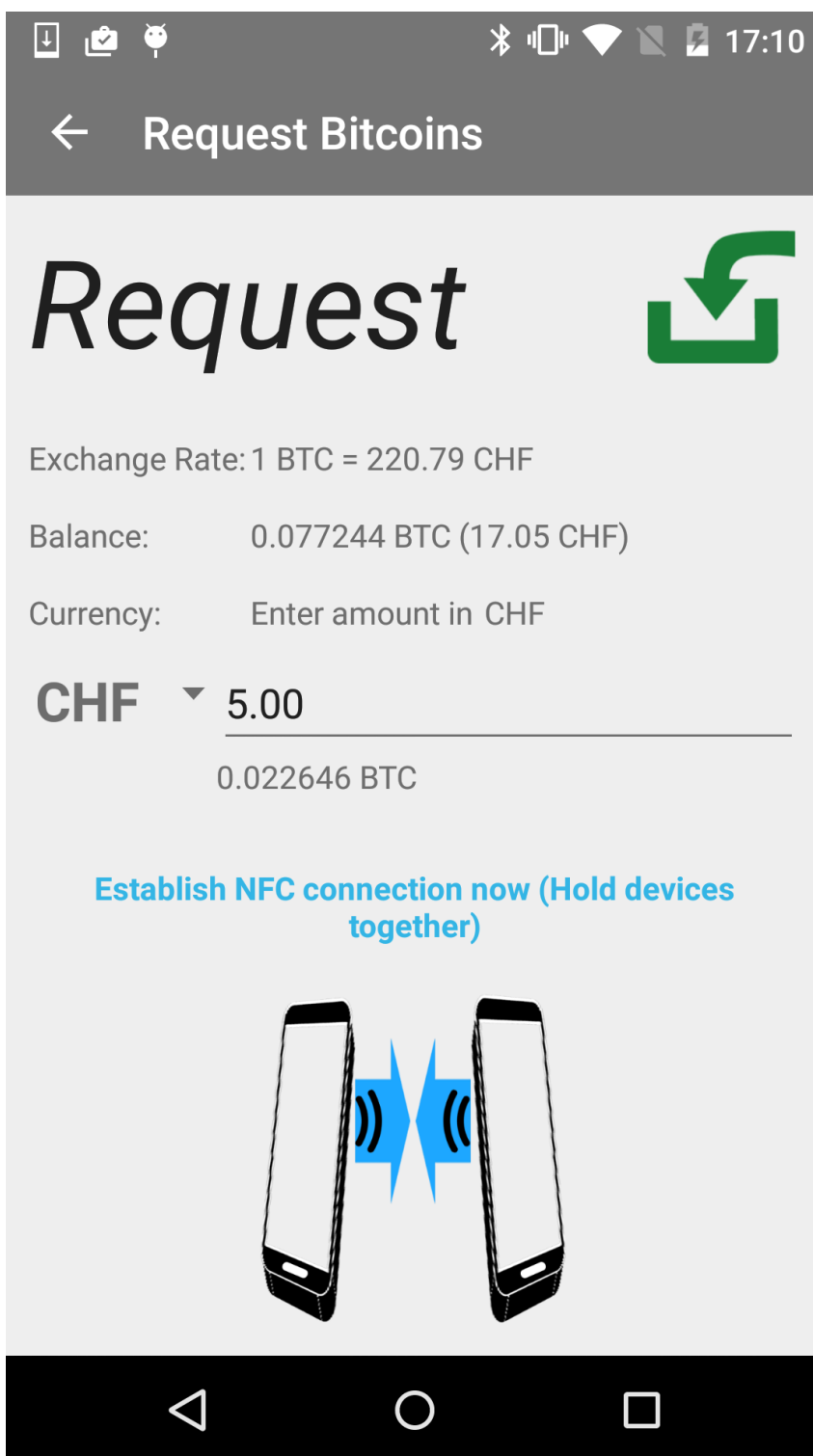


Figure A.4: Payment request view

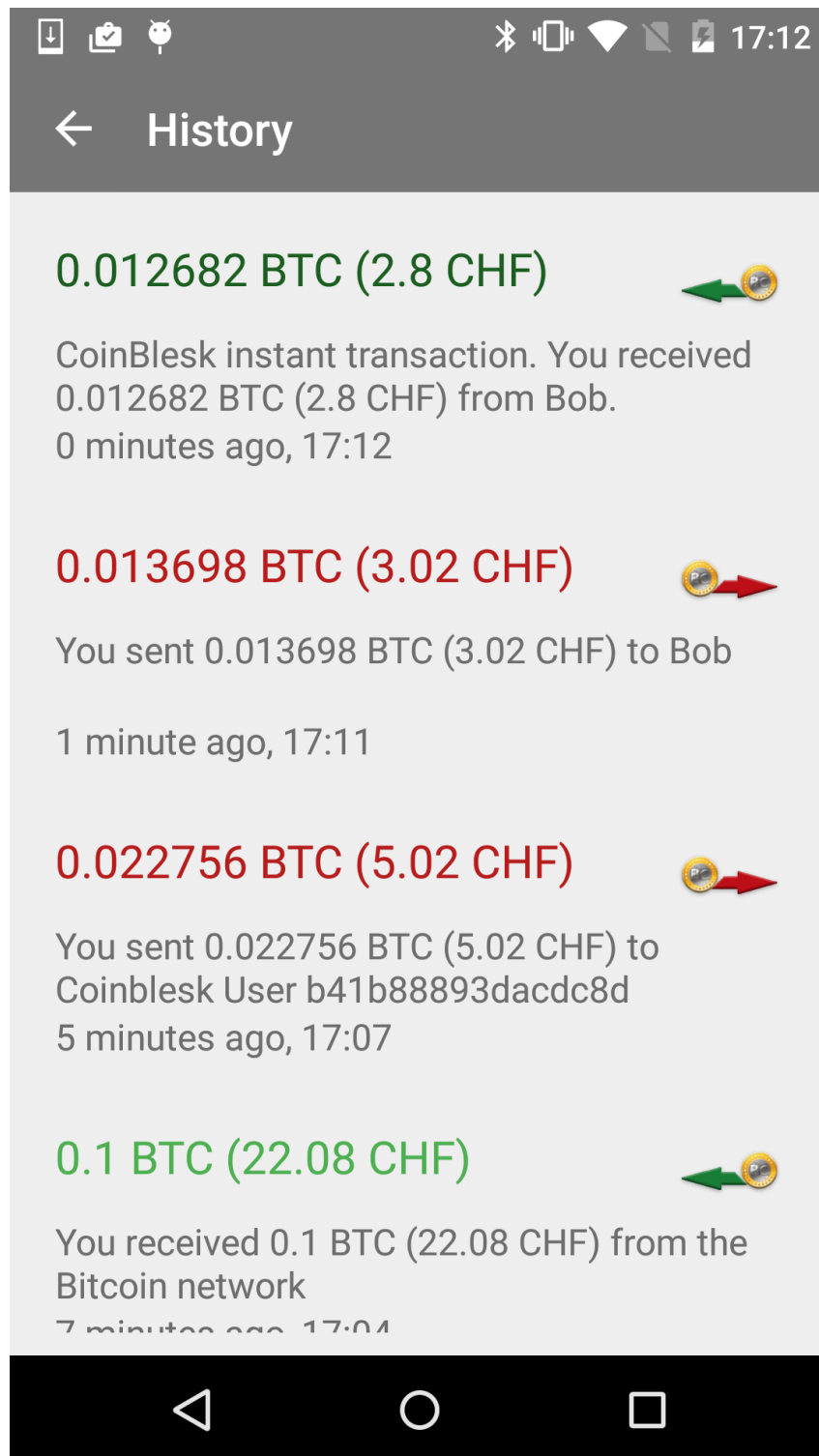


Figure A.5: transaction history view

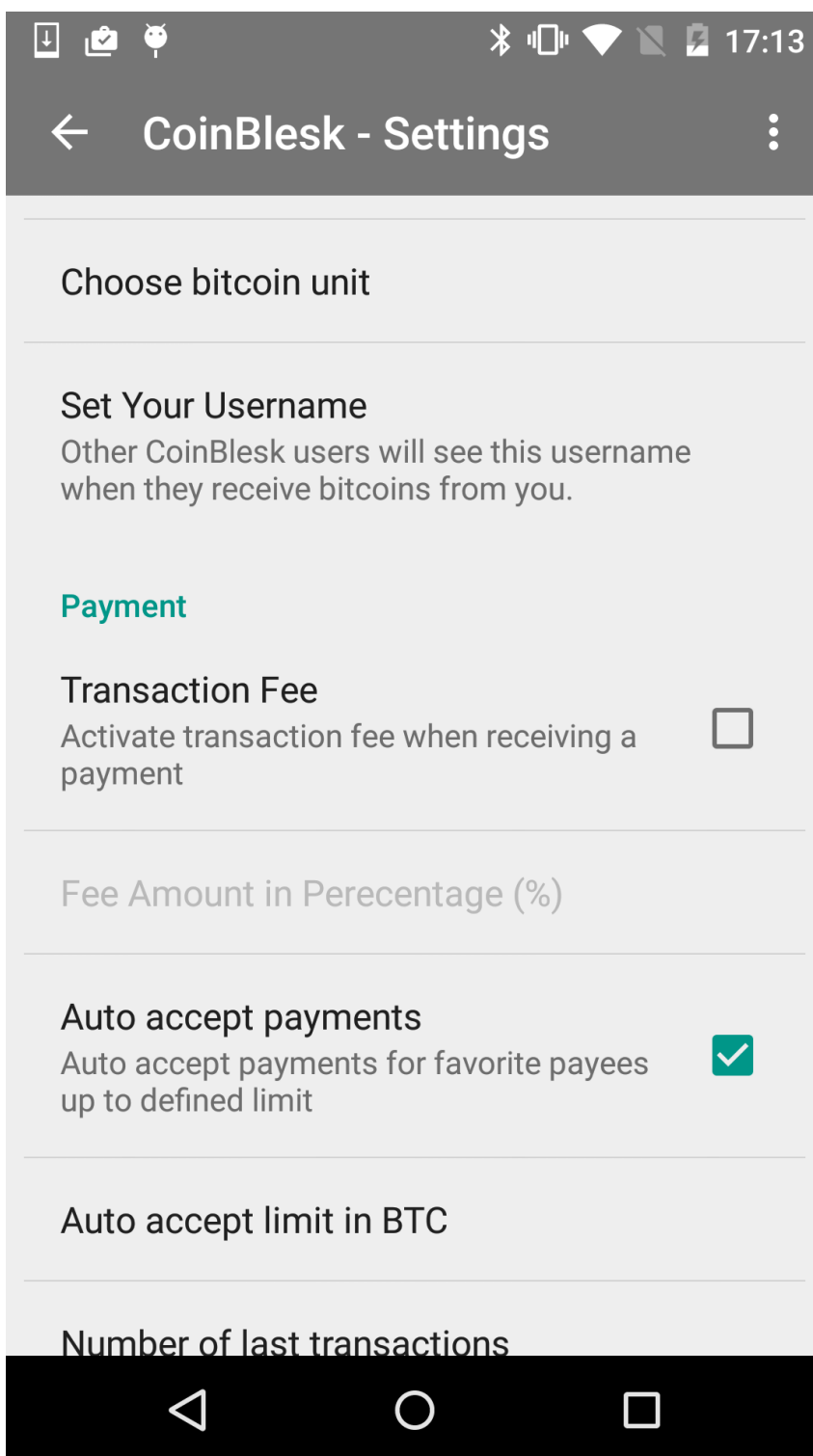


Figure A.6: Settings view

Appendix B

Installation Guidelines

B.1 Installation of the Server

The following instructions are for Debian based systems.

In order to install and run the CoinBlesk server, the following dependencies have to be installed:

- Java 7 or higher
- Tomcat 7 or higher
- Maven
- PostgreSQL Server and Client

Install the dependencies with the following commands:

```
sudo apt-get install openjdk-8-jdk maven postgresql postgresql-client
sudo apt-get install tomcat8 tomcat8-admin
```

Start the PostgreSQL server and create the coinblesk database:

```
sudo service postgresql start
sudo -u postgres createdb coinblesk
```

In order to deploy CoinBlesk to tomcat, make sure you have a Tomcat user with username "admin" and no password, that has the role "manager-script":

```
<!-- {$TOMCAT_HOME}/tomcat-users.xml -->
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="manager-script"/>
  <user username="admin" password="" roles="manager-script"/>
</tomcat-users>
```

Make sure to restart tomcat after you changed the tomcat-users.xml file.

Side note: If you don't want to change your Tomcat users, you can also change the tomcat credentials in the pom.xml file of the coinblesk-server project, and edit the tomcat user credentials in the tomcat7-maven-plugin settings. The same applies for deploying on a remote Tomcat instance. The documentation for the tomcat7-maven-plugin can be found at <http://tomcat.apache.org/maven-plugin-2.0/tomcat7-maven-plugin/deploy-mojo.html>.

All server settings are loaded from the context.xml file of the Tomcat directory. A sample context.xml file is included in this project (SAMPLE.context.xml).

Create a directory for coinblesk and enter it:

```
mkdir coinblesk && cd -
```

Clone the projects "coinblesk-shared-resources" and "coinblesk-server":

```
git clone https://github.com/coinblesk/coinblesk-shared-resources.git
git clone https://github.com/coinblesk/coinblesk-server.git
```

Install the modules in your local maven repository:

```
mvn -f coinblesk-shared-resources/pom.xml clean install
mvn -f coinblesk-server/pom.xml clean install
```

Deploy the CoinBlesk server by running:

```
mvn -f coinblesk-server/pom.xml tomcat7:deploy-only
```

After that, the CoinBlesk server should be running at <http://localhost:8080/coinblesk>

B.2 Installation of the Client

Import the client project in Android Studio, compile and run it.

Appendix C

Contents of the CD

The CD contains this document and the source code of the CoinBlesk 2.0 payment system.