University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Protocol Improvements in CoinBlesk - A Mobile Bitcoin Instant Payment Solution

Alessandro De Carli
Birr, Switzerland
Student ID: 10-751-717

Supervisor: Thomas Bocek
Date of Submission: April 26, 2016

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

CoinBlesk ist eine von der Universität Zürich entwickelte mobile Zahlungslösung. Im Context dieser Arbeit wird das bestehende System auf verschiedenen Ebenen verbessert. Um sofortige Bitcoin Zahlungen zu ermöglichen muss ein Benutzer seine Bitcoins auf eine MultiSig-Adresse einzahlen. Die verbesserte Version entfernt das erforderliche Vertrauen durch den Kunden, dass sie ihre gesperrten Gelder zurückerhalten werden. Zudem bringt die neue Version Unterstützung für neue Kommunikationskanäle wie Bluetooth LE und Wi-Fi Direct sowie eine Abstraktion, jede Art von Kommunikationskanälen zu unterstützen. Im Rahmen der Arbeit werden die verschiedenen P2P-Kommunikationskanäle auf mobile Geräte gebenchmarkt und ausgewertet. Neben einer neuen Version von CoinBlesk veröffentlicht diese Thesis ein Werkzeug P2P-Kommunikationskanäle auf Android zu benchmarken.

CoinBlesk is a mobile payment solution developed by the University of Zurich. The scope of this thesis is to improve the existing system on various levels. By putting bitcoin funds on a multisig address a client can perform payments instantly. The improved version removes the required trust by the client that it will receive its locked funds back. Furthermore the new version brings support for new communication channels like Bluetooth LE and Wi-Fi Direct and an abstraction to support any kind of communication channels. In the scope of the thesis the various P2P communication channels, available on mobile devices, have been benchmarked and evaluated in the context of mobile payments. Besides a new version of CoinBlesk this thesis releases a tool to benchmark communication channels on Android.

Acknowledgments

At first I want to thank Dr. Thomas Bocek my supervisor for his support during my master thesis at the communications group of the university of Zurich. Thomas helped me to critically challenge design decisions and implement a very satisfying solution. His tireless support and passion for the matter was far beyond my expectations.

A big thank you also goes to Prof. Dr. Burkhard Stiller, who provided me the opportunity to write my thesis in his research group in a very interesting topic.

I would also like to thank Christian Killer for designing the application's user experience of the CoinBlesk 3.0 Android client and Michèle Steverlynck for proof reading this thesis.

Last but not least I'd like to thank my friends and family for their moral support throughout this thesis.

Contents

Abstract i

Acknowledgments iii

1 Introduction 1

1.1 Motivation . . . 1

1.2 Description of Work . . . 2

1.3 Thesis Outline . . . 2

2 Background and Related Work 3

2.1 Technology . . . 3

2.1.1 Bitcoin Transactions . . . 3

2.1.2 Blockchain . . . 4

2.1.3 Bitcoin Script . . . 5

2.1.4 Wireless Communication . . . 7

2.2 Security . . . 9

2.2.1 Double Spending . . . 9

2.2.2 Replay Attacks . . . 10

2.3 Swiss Financial Market Supervisory Authority . . . 10

2.4 Actual State Analysis . . . 11

2.4.1 CoinBlesk 1.0 . . . 11

2.4.2 CoinBlesk 2.0 . . . 12

2.5 Mobile Payment Systems . . . 12

2.5.1 Twint . . . 13
2.5.2 Paymit . . . 13
2.5.3 GreenAddress . . . 13
2.5.4 Comparison . . . 14

3 Design **15**

3.1 Architecture . . . 15
3.1.1 Server . . . 15
3.1.2 Client . . . 16
3.2 Protocol . . . 17
3.2.1 Multisignature Setup . . . 17
3.2.2 Instant Payment . . . 18
3.3 P2P Communication Abstraction . . . 20
3.3.1 Processing Steps . . . 20

4 Implementation **23**

4.1 WalletService . . . 23
4.2 LocalBroadcastManager . . . 24
4.3 Persisting Data on Client . . . 24
4.4 DERObjects . . . 25
4.5 Transferring Bitcoins . . . 26
4.6 AuthView . . . 26
4.7 System Integration . . . 27
4.8 Testing . . . 27
4.9 Dependencies . . . 28
4.10 Licenses . . . 29

5 Evaluation **31**

5.1 Limitations . . . 31

5.1.1 Trust . . . 31

5.1.2 Blockchain . . . 32

5.1.3 Bluetooth . . . 32

5.1.4 Wi-Fi Direct . . . 32

5.2 Benchmarks . . . 33

5.2.1 Communication Technologies . . . 33

5.2.2 Blocking Operations . . . 35

5.2.3 Overall Protocol . . . 37

5.3 Goals . . . 38

6 Summary and Conclusions **41**

Bibliography **41**

Abbreviations **47**

List of Figures **47**

List of Tables **49**

A Installation Guidelines **53**

B Application Screenshots **57**

B.1 CoinBlesk Android Client . . . 58

B.2 Android Communication Benchmark . . . 60

C Contents of the CD **61**

Chapter 1

Introduction

Over the years Bitcoin has gained a lot of media attention. Its transaction volume grew larger and larger every year. At the time of writing Bitcoin's market capitalization taken from coinmarketcap.com is 6.5 billion USD [1]. The Bitcoin system usage is still in an early state compared to the transaction volume of credit card companies like Mastercard and Visa. A possible reason why Bitcoin is not yet popular in real world micro payment transactions is its long transaction confirmation time.

To persist transactions Bitcoin uses a global blockchain. This blockchain is distributed on every participating node and contains all transactions ever made. The blockchain has the property that past entries cannot be modified. In order to decide which transaction is allowed to be persisted in the chain special Bitcoin nodes (also called miners) have to solve a cryptographic puzzle (also called proof of work). This cryptographic puzzle adjusts its difficulty based on the computing power of the complete network in a way that in average every ten minutes a new block can be created. Since Bitcoin is a payment system without a central authority the client does not need to trust anyone. Clients should accept a payment only if it is written in the blockchain. The conclusion is that a payment will take in average ten minutes if it is included in the next block. Ten minutes for a payment process is no problem for larger purchases but is too long for micro payments. The usual example used is the payment of a coffee, by the time the payment was accepted the coffee will be cold.

To solve this problem the Communications Systems Group at the University of Zurich created CoinBlesk in the scope of a master project in 2013 [2]. The first version enabled users to pay instantly with their smart phone using NFC. CoinBlesk's ultimate goal is to make Bitcoin a true alternative payment method for the real world.

1.1 Motivation

The first version of CoinBlesk ran in a client-server architecture, where the client was mostly a thin RESTful service consumer and the server was responsible for all communication with the Bitcoin network. Since the server could spend the money without the

client's permission various legal issues arose from this approach. CoinBlesk 2.0 addressed these issues and introduced a new concept working with multisig Bitcoin addresses which brought more responsibilities to the client. Both versions of CoinBlesk worked only in combination with NFC. Furthermore clients need to trust the server with both versions. Bitcoin by its nature is a system that does not require trust in a central authority. For this reason a low level of trust is a desirable property when designing a Bitcoin application.

This thesis' main goal is to further minimize the required trust level and to offer a production ready implementation that allows offline instant payments over Bluetooth LE, NFC and Wi-Fi Direct.

1.2 Description of Work

This thesis covers the design, implementation, and evaluation of an improved protocol for CoinBlesk using NFC, Bluetooth LE, Wi-Fi Direct. The existing CoinBlesk 2.0 version sets the baseline and is used to keep track of improvements. All implemented optimizations are discussed throughout the chapters. The new version created in the scope of this thesis is called CoinBlesk 3.0. CoinBlesk 3.0 introduces a generic abstraction to handle any kind of communication channel, simplifies the CoinBlesk protocol further and improves it in a way that no more client trust is required. All improvements are based on top of the Bitcoin protocol and integrate transparently with the system. The proposed design keeps its compliance with Swiss banking laws and offers a better protection to clients without a negative impact on usability.

1.3 Thesis Outline

The thesis begins with a short introduction, which includes motivation, a short description of what has been done and this outline. Then the background and related work chapter covers the basics needed to understand the decisions taken throughout the work. The design chapter documents all concepts and design decisions for CoinBlesk 3.0. The goal is to explain the improvements and reasons that led to these ideas. The implementation chapter is a high level view of the main developed components. This part explains the various decisions in context of the used SDK. The evaluation chapter discusses the benchmark results as well as discovered limitations. The thesis ends with a summary and conclusion, which on one side covers the main achievements and on the other side the open points that can be addressed with future work.

Chapter 2

Background and Related Work

The following chapter covers the background needed to understand the concepts and decisions made in the design and implementation part of this thesis. The first part covers the technologies used in a sufficient level of detail to explain the challenges and options that led to the implementation. The second part covers the most relevant attack vectors. The last part covers the available instant payment solutions together with their advantages and disadvantages.

2.1 Technology

This section starts with the bitcoin related used technologies. In the second part the various wireless communication technologies are explained.

2.1.1 Bitcoin Transactions

A bitcoin transaction consists out of one or more transaction inputs and one or more transaction outputs. The complete sum of the inputs has to be used up in the transaction. An input is unique and can never be used twice. Inputs are unspent outputs. Everything that is not included in an output will be converted to a transaction fee.

Figure 2.1 visualizes how inputs, outputs and transactions work together. Taking TX 2 as an example we see that we have 50k input and 40k output, this means 10k will be used as transaction fee. The higher a transaction fee, the faster it will be included in the blockchain. The transaction fee is payed to the miners for finding the next block. They will be optimizing their revenue by including transactions with a higher fee first.

This chain like structure of references is the fundament of the Bitcoin system. An output can be converted to an input if one can solve a riddle which is specified in Bitcoin Script (see Section 2.1.3). These riddles are often cryptographic riddles and require a secret to be solved. If a Bitcoin user wants to know his balance he will go through the public blockchain and check for unspent outputs he can unlock and sum these up.

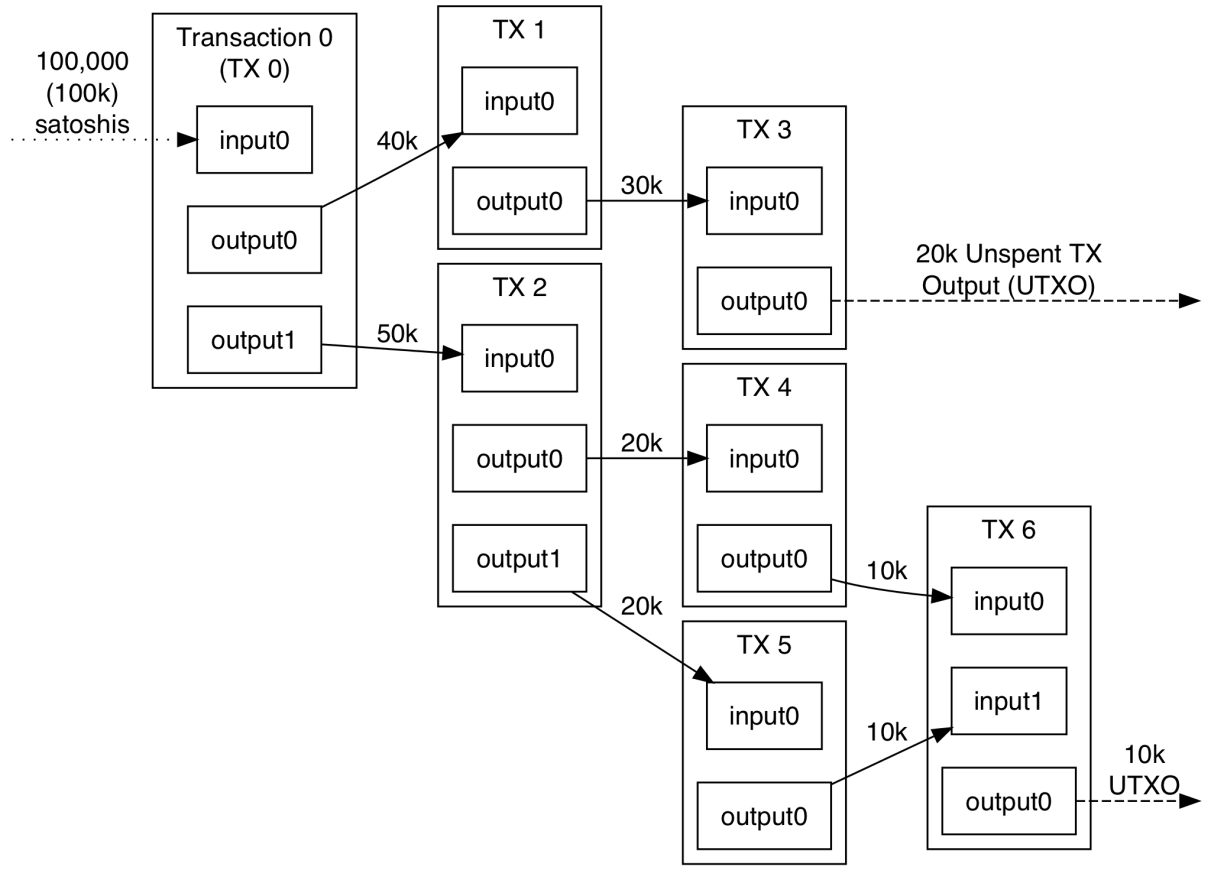

Figure 2.1: Schematic visualization of bitcoin transactions. [3]

This means if the blockchain changes the balance changes. For this reason once a block has been found it should be immutable.

2.1.2 Blockchain

The previous Section 2.1.1 explained how transactions are linked together. These links must be persisted in a way, that the complete Bitcoin system has a single source of truth. Having a single source of truth in a distributed system brings two major challenges:

- find way to update the data where no one can cheat
- make the persisted data immutable

Bitcoin addresses these challenges with the blockchain and a proof of work. Each block in a blockchain holds a reference to the previous block. This reference is a hash of the previous block. The hash of a block changes completely if its content changes. This chain of blocks brings the solution to the immutable data challenge, because if a previous block is changed the complete chain needs to change.

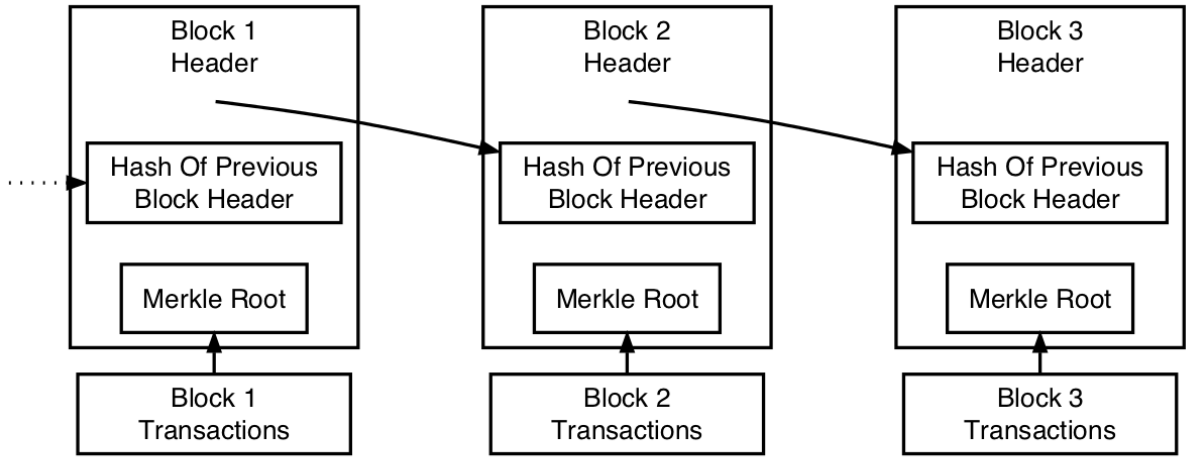

Figure 2.2: Schematic visualization of the Bitcoin blockchain. [3]

Figure 2.2 visualizes how blocks are linked together. By using hash functions references to previous blocks are made.

To decide which block will be the next in the chain a Bitcoin node has to solve a cryptographic puzzle. This puzzle can be solved only with brute force. This means a node can solve the puzzle only with trial and error, there is no more efficient way. The Bitcoin nodes that solve this cryptographic puzzle are called miners. The Bitcoin network sets the difficulty of the cryptographic puzzle such that a new block is created every 10 minutes in average. If a miner can proof to the network it solved the puzzle correctly, the complete network will include its block in the chain and try to find the next block. Miners that can include a block receive a reward and this is how bitcoins are created.

Bitcoin uses the blockchain to store transactions. A block is only valid if its transactions are valid (considering the complete blockchain). Invalid blocks (and thus invalid transactions) are ignored by the network.

Every valid block ever mined is stored on every Bitcoin node. This means every transaction ever made will be stored on every node as well. As Bitcoin is gaining popularity more transaction are made, which means the blockchain grows faster.

Figure 2.3 visualizes the number of Bitcoin transactions made every day. The number of transactions is growing fast, which makes the blockchain grow exponentially. This is not scalable.

2.1.3 Bitcoin Script

The Bitcoin transaction system is based on top of a scripting language called Bitcoin Script [5]. Everyone that wants to spend a previous transaction needs to provide the input that solves the set script. The provider of the input is the one that can define the

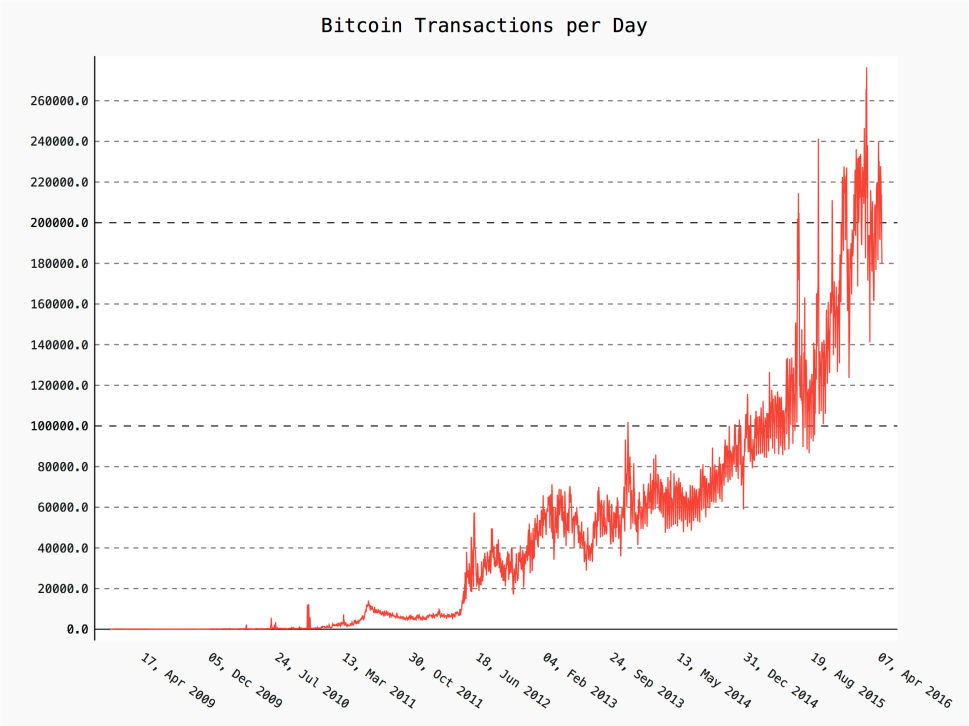

Figure 2.3: Bitcoin transactions per day. [4]

next script needed to spend the funds. Bitcoin Script is stack-based, not Turing-complete and does not allow loops [5]. An input is valid (and thus the transaction spendable) if after the execution the top stack item is true.

Bitcoin Script has various operations called Opcodes. By using these codes one can add conditions and logic to the spending of a given transaction. New Opcodes are introduced in the system by using Bitcoin Improvement Proposals (in short BIP). The purpose of a BIP is to propose improvements of the Bitcoin system. Everyone can write a new BIP, the work flow each BIP has to follow is described in BIP 1 [6]. If a consensus in the Bitcoin community was found a BIP becomes final and is implemented in the Bitcoin core client.

Pay to Script Hash

BIP 16 [7] introduced a new standard for defining transactions called pay to script hash (in short P2SH). In P2SH the sender of a transaction does not need to know the script used to spend the transaction. The only thing the sender needs to know is the hash of the script later on used to redeem the funds. The payment receiver is the one that has to specify the redeem script and the input for a valid execution. P2SH introduced a new Bitcoin address format specified in BIP13 [8].

Since version 2.0 of CoinBlesk [9] this feature is used to send money to a multisignature Bitcoin Address.

Multisignature Transaction

Bitcoin script allows to specify transactions that can require multiple signatures to spend them [10]. When setting up a multisignature (in short multisig) transaction one can decide how many signatures are required to spend the fund (N) and how many signatures are valid (M), which leads to a N-of-M multisig transaction. A 2-of-3 multisig transaction means, that in order to spend the funds 2 of the 3 specified valid signatures are required.

CoinBlesk uses a 2-of-2 multisig account. One secret is held by the client and the second required secret is held by the server. By having a dependency on the server no funds can be spend without his acknowledgement. CoinBlesk uses this feature to provide instant verifications of payments, because it can mitigate double spends (it simply won't sign already spend funds).

Time Locks

Bitcoin offers two ways to time lock funds. The first one was there since the beginning, the second one was included with BIP 65 [11] in February 2016:

nLockTime is a transaction parameter used to define when the transaction is allowed to appear earliest in the blockchain [12]. The point in time specified in the transaction is either defined in blockchain height or Unix time.

OP_CHECKLOCKTIMEVERIFY BIP 65 [11] introduced the Opcode `OP_CHECKLOCKTIMEVERIFY`. This Opcode allows to time lock outputs rather than transactions, this means in order to lock funds a user does not require to perform two transactions (pay on to a multisig address and request time locked refund transaction) but can achieve the same with a single transaction. The locking process becomes cleaner and more transparent because everyone that has access to the blockchain can read for how long the funds are time locked.

Since version 2.0 of CoinBlesk [9] nLockTime is used to create refund transactions. Section 2.1.3 described the mechanism of locking funds on a multisig address, the refund transaction makes sure that those funds can be retrieved on a later point in time even if the server is being uncooperative. With CoinBlesk 3.0 the refund transaction is requested before locking the funds, which mitigates a client extortion scenario. OP_CHECKLOCKTIMEVERIFY is a better suited solution for the problem CoinBlesk wants to solve but when this thesis started it was not yet supported by the Bitcoin system.

2.1.4 Wireless Communication

CoinBlesk is a system that uses an Android mobile application as its client. In order to provide offline P2P transactions clients communicate with each other using wireless technologies. The Android SDK provides connectivity API's to communicate with other

devices over NFC, Bluetooth and Wi-Fi [13]. The new version of CoinBlesk uses all of these technologies for the instant payment feature. The following sections will describe these technologies.

Near Field Communication

Near Field Communication (in short NFC) is a short range wireless technology. It allows devices to communicate within 10cm distance [14]. The technology is based on the RFID standard. NFC can be used in combination with passive tags that don't require any external power source. Its theoretical transmission rate of 100-800Kbit/s makes it only usable for small data transfers. NFC provides a fast handshake, which makes the technology very interesting for instant communication.

The NFC architecture supports three operation modes [15]:

Peer-to-Peer Mode This mode allows two NFC devices to communicate with each other. Both devices are active and communicate using a logical link control protocol.

Read/Write Mode This mode allows an active NFC device to read data from or write data to a passive device (*e.g.,* a NFC tag). The data exchanged has to follow the rules described in the NFC Data Exchange Format (in short NDEF) [16].

NFC Card Emulation Mode This mode allows a passive device to emulate an NFC smart card. The device in NFC Card emulation mode cannot initiate the communication, it can only reply to request made by the active device interacting with it.

CoinBlesk 3.0 uses NFC in Card Emulation Mode. The NFC API was included in the Android SDK since Android 2.3. Even though iOS devices have NFC hardware no API is provided to developers.

Bluetooth

Bluetooth (in short BT) is a short range wireless technology. It allows devices to communicate within 10m distance [17]. This distance allows BT to be used for a wireless personal area network (in short WPAN). In order to setup a Bluetooth connection a slave (client) connects to a master (server). After the connection is setup both parties can read and write data over the connection. Its theoretical transmission rate is 2.1Mbit/s.

In a scenario where two devices have never connected before one of the devices needs to get discovered first. A BT discovery takes a long time (10.24 seconds per inquiry) especially if there are many devices [18].

The BT API was included in the Android SDK since the very beginning [13]. iOS provides access to its BT hardware via the Core Bluetooth API [19].

Bluetooth Low Energy

Bluetooth Low Energy (also called Bluetooth Smart) is a wireless communication technology that allows data transmission up to 10m distance. BLE is not retro compatible with BT but uses the same 2.4 GHz band, which allows both technologies to share the same antenna.

In order to transfer data BLE uses the Generic Attribute Profile (in short GATT) protocol. A BLE server advertises it's characteristics and the client reads/writes these characteristics with GATT operations. This advertising system reduces the latency and thus makes BLE suitable for small instant data transfers.

BLE support was included in Android SDK since version 4.3 [13]. iOS provides BLE access through its Core Bluetooth SDK starting with iOS 5 [19].

Wi-Fi Direct

Wi-Fi Direct allows two Wi-Fi devices to setup a connection without a central access point. The range of Wi-Fi Direct is around 100m [17]. With a bandwidth of 54 Mbit/s (depends on standard used, *e.g.,* a/b/g/ac) [17] Wi-Fi Direct is best suited for larger data transfers.

To discover each other the clients send Wi-Fi probe requests [20]. After discovery the clients negotiate which device should be the group owner. A Wi-Fi Direct client authenticates its connection with another client (group owner) by using WPS [21].

Wi-Fi Direct support was included in Android SDK since version 4.0 [13]. iOS provides a limited Wi-Fi Direct access through its Multipeer Connectivity Framework starting with iOS 5 [22], this framework only allows iOS to iOS connections.

2.2 Security

When dealing with money transfers security is a main concern. The following section will go through the most relevant attack scenarios for a Bitcoin instant payment service.

2.2.1 Double Spending

Double spending is a common attack scenario in distributed ledgers. The idea is that a merchant is asking money from a malicious buyer, this buyer will then prepare two transactions with the same source of money. In the first transaction money is transferred to the merchant's account and the second transaction transfers the same money back to the buyer's account. The first transaction is sent to the merchant, which will accept the valid payment, while the second transaction is broad casted at the same time to the distributed network. The network has now two conflicting transactions and will need to

get a consensus, which one it should accept. If the network accepts the second transaction, the merchant will not receive any money even though it thought the payment succeeded. The attack was successful.

The Bitcoin system avoids double spending by using a blockchain. Every new block attached to the chain will only be accepted if it contains only valid non conflicting transactions (considering the whole chain). A Bitcoin block is created in average every 10 minutes. This means that the merchant should wait until the transaction is found inside a block (thus 10 minutes in average) before accepting it. There is a probability that multiple blocks get attached to the chain (forking). This happens if the network finds multiple blocks at roughly the same time. In this case the nodes will decide on their own which block is the valid for them and try finding the next block. When the next block is found all nodes will switch to the longest chain and reject the alternative blocks. A block's depth defines how many new blocks were chained after that block. The block's depth can be used as a measure of how safe a transaction is. In the Bitcoin system the recommendation is to wait until your transaction is in a block of depth 3. If transactions get instantly accepted the risk of double spending is very high [23].

2.2.2 Replay Attacks

A replay attack takes place when a potential attacker repeats a valid request maliciously. In the context of payments an example would be that a merchant could repeat its payment request thus receiving multiple times the same payment amount.

The Bitcoin system avoids replay attacks by using its transaction concept. A transaction always refers to an unspent transaction. A transaction is always either unspent or fully spent, there is no *partially spent*. In order to consume a transaction a merchant would need some secret input from the client [5]. This input is typically a signature. The merchant cannot reuse that signature to create other transactions with the client's funds because it is valid only for one transaction and that transaction will be in a spent state. This mitigates replay attacks.

2.3 Swiss Financial Market Supervisory Authority

The Swiss Financial Market Supervisory Authority (in short FINMA) was created in 2009 [24]. Its responsibilities are defined by the Financial Market Supervision Act (in short FINMAG). FINMA is relevant for CoinBlesk because bitcoin is legally defined as a currency. FinfraG Art. 4.2 [25] states that payment providers require a FINMA approval if the users' financial safety is at risk. Having bitcoins where the service's operator has full control over them is a financial risk. This sets the requirement of creating a system, where the funds cannot be spend without the client's permission.

2.4 Actual State Analysis

CoinBlesk 3.0 is the third version developed at the University of Zurich. This section describes the previous versions together with their strengths and weaknesses.

2.4.1 CoinBlesk 1.0

The first version of CoinBlesk was implemented in the scope of a master project at the University of Zurich [2]. In order to enable instant payments a user transfers its money to a bitcoin address held by the server itself. Once this first top up transaction was confirmed inside the blockchain, the user can start sending funds instantly to other users in the CoinBlesk 1.0 system. The server holding the private keys has a private ledger that books all instant transactions off-blockchain. If a user decides to leave the system he can simply transfer the remaining funds to an ordinary Bitcoin address. The client consumes these services with ordinary RESTful HTTP calls to the server. The communication between the instant payment parties takes place over NFC.

This approach has the following advantages:

Off Blockchain All instant transactions happen within the system. The blockchain is relieved by these instant payments. Only the first top up of the account or the last transfer of remaining funds are written to the blockchain.

Easy Client Logic All bitcoin related operations are handled by the server. The client logic can be held very slim since all informations are retrieved from the server via HTTP calls.

Lightweight Server Operations Instant transactions happen off-blockchain and are basically database entries. Cryptographic operations happen only when the remaining funds are sent back.

On the other hand the following disadvantages arise:

Single Point of Failure The server holds a central database where all transactions are booked. Furthermore all private keys are stored in plain text in the same database.

Not FINMA Compliant After the top up took place the server has full power over the funds. The server's operator becomes a financial intermediary thus is subject to various FINMA laws. The University of Zurich cannot comply to those laws [9].

Online Both parties of an instant transaction require internet access to communicate to the server.

CoinBlesk 1.0 is no longer available and maintained.

2.4.2 CoinBlesk 2.0

The second version of CoinBlesk was implemented in the scope of a bachelor thesis at the University of Zurich [9]. The goal was to address the main disadvantages available in CoinBlesk 1.0. In order to enable instant payments a user transfers its money to a multisig Bitcoin address. One private key is held by the client and one private key is held by the server. Every instant payment has to be signed by the server and the client in order to be valid. Once the transaction is valid it is broad casted to the blockchain. After every transaction the remaining funds are returned with a time locked transaction (refund transaction). This gives the user the certainty, that in case of a server failure he will be able to retrieve its funds after the time lock expired.

The assumption is that the server will never sign a transaction output twice. This assumption mitigates the double spending issue and allows a user to accept such a transaction instantly. The communication between the instant payment parties takes place over NFC.

This approach has the following advantages:

Resilient to Server Failures If the server fails the client's funds can be retrieved when the time locked refund transaction is mature.

FINMA Compliant The server cannot spend the money without the signature of the client. This means the server's operator is not classified as a financial intermediary, thus no FINMA laws apply.

Build on top of Bitcoin The protocol uses Bitcoin script and is transparent to other Bitcoin clients.

On the other hand the following disadvantages arise:

Blockchain Every instant payment transaction is recorded into the blockchain.

Client Extortion The client receives its time locked refund transaction only after having locked up its funds. If the server decides not to sign the refund transaction, the funds will only be spendable if the server collaborates. This means the server can extort the client [26]

NFC only To support an offline use case, communication takes place over NFC. A device without NFC cannot perform an instant payment.

CoinBlesk 2.0 is no longer available and maintained.

2.5 Mobile Payment Systems

The following section covers the various solutions available to Swiss users for mobile payments.

2.5.1 Twint

Twint offers contact less payments through Bluetooth LE for iOS and Android [27]. A dedicated Bluetooth LE beacon is installed on the merchant's site. The client has an application and communicates with this beacon. Two modes are implemented. In the online mode the device talks directly to the backend server to accept a payment. In the offline mode the device talks to the backend through the merchant's internet connection [28]. CoinBlesk 2.0 uses a similar solution to allow offline payments.

Twint uses a centralized architecture and users have to top up their accounts before performing payments. This is similar to what used to be the case in CoinBlesk 1.0. No Bitcoin concepts are used, to provide this service. Twint has to be a financial intermediary and follow Swiss banking laws.

2.5.2 Paymit

Paymit was the first big player in the Swiss market offering mobile payments. Its target audience was primarily private persons sending money to each other. In order to accept and send payments a user has to provide a credit card and/or bank account. Paymit links this information in the backend with the users' mobile phone number. Sending money becomes very easy because it requires only the recipients mobile phone number and the amount. The user receives payment confirmations and requests instantly, just like with an SMS.

A solution for merchants was introduced in 2016 [29]. To pay merchants Paymit works with QR codes. The implication of this approach is that no offline payments are possible for now. Paymit uses a centralized architecture, expenses are billed directly to the credit card and payments are transferred to the bank account. No Bitcoin concepts are used, to provide this service. Paymit is a software that has to be operated by financial intermediaries and follow Swiss banking laws.

2.5.3 GreenAddress

GreenAddress is a bitcoin wallet with many similarities to CoinBlesk 2.0. By using a multisig account with the GreenAddress server, payments can be approved instantly [30]. Unlike CoinBlesk 2.0 GreenAddress offers nLockTime transactions before the client tops up the account or sends payments, which mitigates the client extortion scenario from the start. All transactions made with GreenAddress happen with a direct connection to the server, there is no instant payment support when the client is offline.

GreenAddress focuses on person to person payments, no merchant POS solution is offered by the company at the time of this writing. GreenAddress can operate without being a financial intermediary, because funds cannot be used without the client's approval.

2.5.4 Comparison

Table 2.1 compares the various mobile payment systems described above. The ”Offline” dimension describes whether a buyer can be offline during payment. The ”Pay to Phone” dimension describes if the system allows the user to send money directly to a contact only by knowing his phone number. The ”Uses Bitcoin” dimension describes if the system is build on top of Bitcoin. The ”Operator Controls Funds” dimension describes if the service operator has full control of the clients’ funds. The ”Opensource” dimension describes whether the complete system is opensource.

When comparing the non-Bitcoin systems the main difference is the offline support which only Twint offers. When comparing the Bitcoin systems ther are two differences the first is that GreenAddress offers a way to send funds directly to contacts and the second is CoinBlesk is opensource.

	Twint[27]	**Paymit**[29]	**GreenAddress**[30]	**CoinBlesk**
Offline	yes	no	no	yes
Pay to Phone	yes	yes	yes	no
Uses Bitcoin	no	no	yes	yes
Operator Controls Funds	yes	yes	no	no
Opensource	no	no	no	yes

Table 2.1: Comparison of different mobile payment systems

Chapter 3

Design

3.1 Architecture

CoinBlesk follows a Client-Server architecture. Since Bitcoin is a P2P network ideally no server would be required. The only reason why CoinBlesk uses a server is to mitigate possible double spending attacks. The goal is to find a way to provide this functionality by keeping the server as thin as possible. Furthermore the trust level required by a client should also be reduced to a minimum.

3.1.1 Server

In CoinBlesk 2.0 many responsibilities were moved away from the server to the client by using multisig Bitcoin addresses. With the help of time locked refund transactions the client has the certainty to receive its money back in case of server failure. This refund transaction was given to the client after having fully signed and broad casted the transaction. By doing so the server can extort the client by not giving him it's time locked refund. Without the refund transaction the client's money can only be spend in collaboration with the server. On one hand this requires trust on the client side that the server won't extort him. On the other hand the server needs to keep track of the refund transactions it issued, because in case of failure during payment the client might not be able to receive its refund transaction thus would need to request a new one.

The new design generalizes the server in a way that it gives more freedom to the client and simplifies its functionality. The only thing the server now needs to do is provide signatures and allow clients to check whether a certain transaction can be considered to be instant or not.

The CoinBlesk server's provides the following functionality:

Key Exchange The client needs to be able to exchange public keys with the server in order to setup a multisig Bitcoin address. Every client's public key needs to have

an asymmetric key pair stored on the server side. The key pair and client's public key need to be persisted and accessible only by the server.

Sign Transactions Every transaction sent by the client needs to be signed with the appropriate key pair stored in the previous step. The signature has to be sent back to the client. Already verified transaction outpoints (thus spent outpoints) will not be signed and throw an error.

Verify Transactions The payment receiver can verify whether the transaction it received can be considered instant or not. The verify step follows the transaction chain recursively and breaks when it finds a confirmed transaction in the blockchain. The transaction is considered to be instant iff:

- All relevant transaction outpoints where signed only once without a time lock.
- All relevant time locked transaction outpoints don't become spendable within a given threshold.

The Key Exchange functionality is only required when bootstrapping. This leaves just two very general tasks for the server.

3.1.2 Client

The client itself has a lot of freedom with the new protocol. It can decide on it's own what kind of Bitcoin transactions it wants to do. The only requirement is that it sends its transaction to the server for the signature. This allows the client to be compatible with any other Bitcoin client.

When it comes to instant transactions the client receiving the money has to take one additional step, namely asking the server whether the transaction can be considered instant or not. It's the client's responsibility to decide what to do with this information.

The CoinBlesk client's provides the following functionality:

Key Exchange Before any instant payment can be made the client needs to perform a key exchange with the server. Only after this key exchange the client can set up its P2SH address.

Create Transactions Any kind of transaction can be created as long as it is valid. Every transaction originating from the P2SH address needs to be sent to the server to get the signature. When sending the transaction to retrieve the signature the transaction can be sent without the client's signature. This means the server signs a transaction without being able to spend it.

Verify Transactions The client receiving the money has to ask the server whether the transaction can be considered to be instant or not.

Similar to the server the client needs to perform only two tasks when it comes to instant payments. Because the server will accept and sign every transaction the client can create its refund transaction at any time using the generic sign interface. This simplification allows CoinBlesk to be easier to adopt by external applications because there is no need to implement a complex work flow:

- If a third party application (*e.g.,* a web shop) wants to be able to accept instant payments it just needs to use the verify interface and ask the server for information. Which is only one request.
- If a third party application (*e.g.,* an exchange) wants to be able to send instant payments to CoinBlesk clients it just needs to send whichever kind of transaction it wants to the server to ask for a signature. Which is only one request for the actual payment. Setting up a P2SH account requires the additional key exchange request.

3.2 Protocol

The new protocol can be split into two parts. The first part is where we exchange keys with the server, the second part is where we actually perform the instant payment.

3.2.1 Multisignature Setup

In the key exchange part the client communicates directly with the server via internet. This part can be seen as a registration to the CoinBlesk system. The goal is that client and server can setup a common multisig address. No confidential information is exchanged during the key exchange only public keys are transferred.

Figure 3.1 shows the sequence diagram of the key exchange. The interaction begins with the client generating an asymmetric key pair. The client sends its public key to the server. The server checks whether it already has a key pair associate to the client's public key, if not it generates one. The server returns the associated public key. At this point both client and server can generate the P2SH address and start watching on the Bitcoin network for related activity. Before the client pays on to this address he can generate a time locked refund transaction and send it to the server to sign. The client then verifies the refund transaction and if everything is OK it will store the exchanged informations and broad cast its first payment onto the multisig address.

It's worth noting that because the client gets his refund before sending money to the multisig account he cannot be extorted. This is the only reason why this communication can take place over an insecure channel. If the client would immediately send its money to the multisig address without requesting a prior refund a MITM attack scenario can take place and the client can be extorted, because there is no way he can get his money back.

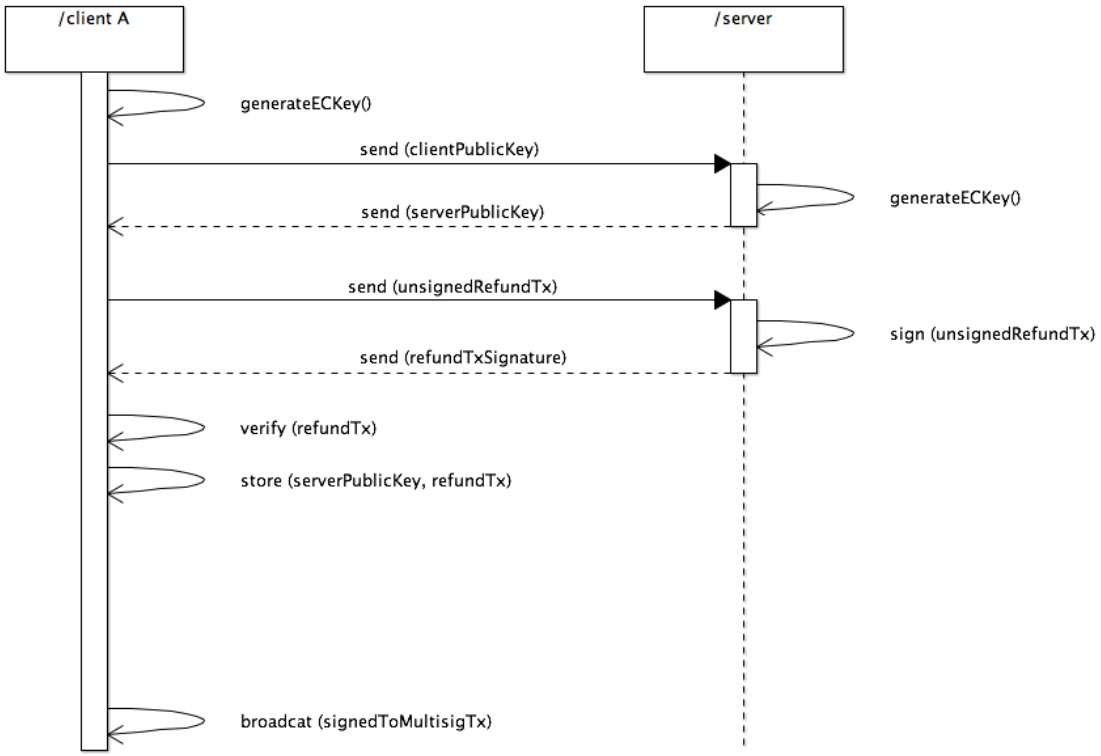

Figure 3.1: Multisignature setup sequence diagram

3.2.2 Instant Payment

In the case of instant payment the buyer uses the merchant as proxy to communicate with the server. The buyer uses P2P communication technologies (*e.g.,* NFC, Bluetooth LE, ...) to interact with the merchant. The buyer does not require an internet connection. Using the merchant as a proxy implies that the data sent is either encrypted end to end or not confidential. The protocol was designed in a way that no confidential information is transferred.

However the buyer has to be sure he is talking to the merchant. If this condition is not met a potential attacker can change the payment request to contain its address.

Figure 3.2 shows the sequence diagram of an instant payment. The interaction begins with the buyer discovering the merchant. The merchant sends the payment request to the buyer. At this point the user needs to verify that its client is really talking to the right merchant. On the buyer's and the merchant's devices an authentication view is shown. If both show the same authentication view the user can accept and continue the payment process, because he knows that he is paying to the correct address.

The buyer now sends the unsigned transaction through the merchant to the server and

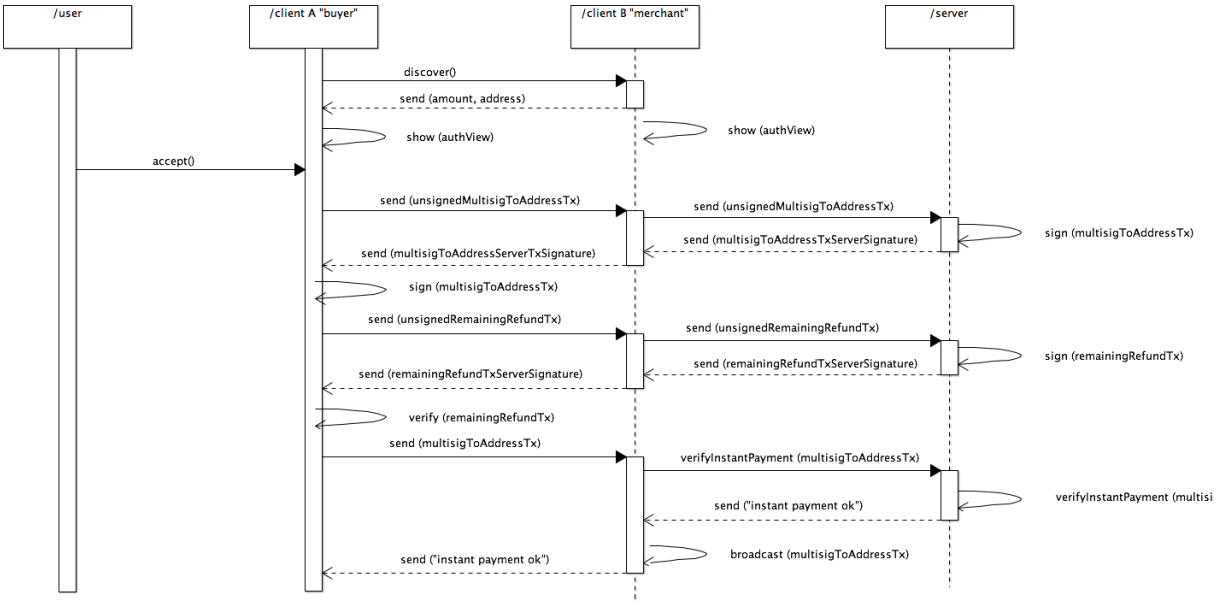

Figure 3.2: Instant payment sequence diagram

requests a signature. With this signature the buyer now can create the fully signed transaction and a refund transaction for his remaining funds. The buyer now requests a signature for the refund transaction through the merchant to the server. After having received the refund transaction signature the buyer cannot be extorted anymore and will send its fully signed transaction to the merchant. The merchant will ask the server to verify the transaction in order to know if he can accept the funds instantly.

It's important to remark that if the last verification succeeds the used transaction outpoints cannot be signed again by the server.

Refund Transactions and Instant Transaction Verification

CoinBlesk 3.0 simplifies the server in a way that it will sign every transaction it receives. This means no special interface is required to create and sign refund transactions. It's the client's responsibility to take care of its refund transactions. The client can decide on its own how long the refund transaction should be locked, there is no server implied limit. The only thing that the server does is store these transactions. If the merchant asks if a certain transaction is valid for instant payments the server goes through that list and checks the time locks for a given threshold (*e.g.,* four hours).

Figure 3.3 shows an activity diagram describing the recursive flow for checking if a transaction can be considered instant or not. This is the flow executed by the server when a merchant asks to verify a certain transaction. The recursion starts at the *is parent instant* step and converges when the transaction is either not a multisig transaction or was included in the blockchain. Since transactions are chains and loops are conceptually not possible (see Section 2.1.1) this recursion will find an end.

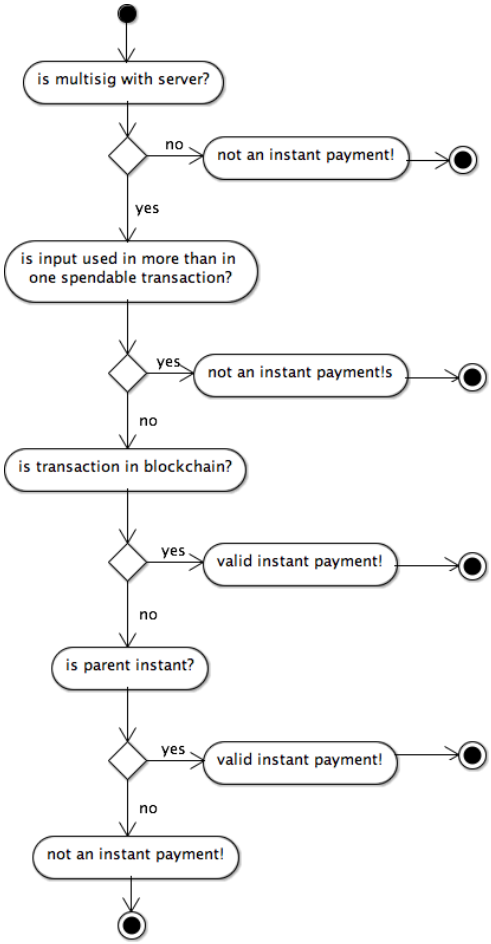

Figure 3.3: Instant payment verification

Since the refund transactions don't expire the client can simply store them and redeem them whenever the lock time passed.

3.3 P2P Communication Abstraction

This version of CoinBlesk comes with a client that supports different P2P technologies. Having different technologies introduces the challenge of finding an appropriate abstraction that allows the encapsulation of protocol related logic into reusable components. Missing to do so results in an unmaintainable code, every protocol change would require a change for every technology (thus a change at four places). Some of the technologies transfer data in streams (Wi-Fi Direct, Bluetooth) while other technologies transfer data in byte arrays of fixed sizes (NFC, Bluetooth LE).

3.3.1 Processing Steps

Every interaction described in Section 3.2.2 has in common that some data is sent as an input. This input is processed and an output is returned. The output again will serve

as an input for the next step. For this reason the CoinBlesk 3.0 client defines a `Step` interface that follows the strategy design pattern. A class implementing this interface has to give the implementation of the method `byte[] process(byte[] input)`. This abstraction puts the whole protocol logic into reusable components, that can be used for every communication technology. The complete protocol uses seven steps. Four steps are on the merchant's side and three steps are on the buyer's side. The following list explains every step in detail. The entries written in cursive are steps that are handled by the merchant, the others are the ones handled by the buyer.

`PaymentRequestSendStep` *This step starts the protocol. It is the only step where no input is required. The output of this step is the actual payment request.*

`PaymentRequestReceiveStep` This step receives as an input the payment request. The payment request is authorized. The output is the unsigned transaction.

`PaymentAuthorizationReceiveStep` *This step receives as an input the unsigned transaction. The unsigned transaction is sent to the server for a signature. The output is the server's signature.*

`PaymentRefundSendStep` This step receives as an input the server transaction signature. The server signature allows to completely sign the transaction and create a refund transaction for the remaining funds. The output is the unsigned refund transaction.

`PaymentRefundReceiveStep` *This step receives as an input the unsigned refund transaction. The unsigned refund transaction is sent to the server for a signature. The output is the server's signature.*

`PaymentFinalSignatureSendStep` This step receives as an input the server refund transaction signature. The buyer now has its completely signed refund transaction and sends the previously created fully signed transaction. The output is the fully signed transaction.

`PaymentFinalSignatureReceiveStep` *This step receives as an input the fully signed transaction. The transaction is sent to the server for verification. No output is expected.*

The beauty of having steps is that one can easily change the protocol by adding some new steps or removing ones. For example if we say that we want to speed up the protocol rather than having a trust less payment, we can remove the steps `PaymentRefundSendStep` and `PaymentRefundReceiveStep`.

Chapter 4

Implementation

The following chapter goes through the most relevant concepts and decisions made during implementation. The goal is to explain why certain things where made in a specific way and how those things work.

4.1 WalletService

The `WalletService` is a pure Android `Service`. It holds the bitcoinj wallet and runs the various needed threads to stay in sync with the Bitcoin network. The service is started only once by the `MainActivity` but bound multiple times by different fragments that use its functionality. The `WalletService`'s responsibility is to:

- Bootstrap the bitcoinj wallet
- Broad cast a message when transactions change
- Broad cast a message when balance change
- Broad cast a message when exchange rate change
- Fetch the most current exchange rate
- Give access to unspent outputs
- Give access to the transaction list
- Give access to the wallet's Bitcoin address
- Commit transactions to the Bitcoin network

All broadcast operations are done using the `LocalBroadcastManager`.

4.2 LocalBroadcastManager

Android has a system wide message bus called BroadcastManager. This message bus is used by the system to allow communication with applications. Message listeners have to implement the `BroadcastReceiver` interface and register for relevant events using an `IntentFilter`.

The same message bus concept can be used in the context of inter component communication within the same application. `LocalBroadcastManager` is a message bus that can be accessed only within the same application. This implementation of CoinBlesk uses this pattern extensively for decoupled communication between the different components.

For example when the `WalletService` receives a new relevant transaction from the Bitcoin network it will use `LocalBroadcastManager` to broadcast this event. All fragments interested in this event are listening to the broadcast and will thus update their view immediately and always show up to date information.

It would be possible to implement a proprietary observer pattern in order to provide the same functionality, but when communicating between components with different life cycles memory leaks are very common. `LocalBroadcastManager` comes pre shipped with Android and can deal with life cycles without many thoughts.

4.3 Persisting Data on Client

Android does not provide a simple way to persist Java objects out of the box. Many heavyweight frameworks and libraries exist, but they come with a big overhead. For data persistence I've chosen to create a simple object storage library that would allow a user to access its data asynchronously and synchronously. The complete API uses generics and is type safe. The only dependency needed is GSON [31], which is used for object serialization.

The storage object is a singleton and needs to be initialized with the root path. This allows it to be easily reused throughout the application. The library implements a filtering system based on a strategy pattern. A developer can define its own filter with per-object logic by implementing the `Filter` interface. Every public object storage operation starts an own thread with callbacks to return results. To allow synchronous calls `CountdownLatch` is used for blocking until the callback returns.

Every Java object can be stored if GSON is able to serialize it. The only requirement the storage has is that the object implements the `UuidObject` interface. This is needed in order to allow a generic unique identification of objects (thus avoiding problems like duplicates). All objects are lazy loaded from disk on the first access, after the first access the objects are cached in memory unless explicitly cleared with the provided API. Nothing is written to disk until the `commit()` method is called. `commit()` serializes the in-memory object map to a JSON file.

In order to make this storage functionality reusable among different projects the source code was packaged in a library. The opensource library is called Simple-Object-Storage and can be found on GitHub [32].

4.4 DERObjects

CoinBlesk version 2.0 serialized the data needed for transfer in JSON. JSON has the nice property of being easily readable by humans and offers great libraries to work directly with Java objects, but this comes at the price of size and computation overhead. Especially when serializing byte arrays (such as cryptographic signatures) the overhead becomes remarkable because the payload needs to be converted in a UTF-8 representation.

CoinBlesk 3.0 offers support for Bluetooth LE and NFC. These communication channels have a very limited bandwidth and MTU. In order to support instant payments the payload size needs to be kept to the bare minimum. When choosing a new serialization standard these requirements needed to be satisfied:

- Multiple types need to be serialized and deserialized in a generic way. Developers should work with objects and not care about the underlying byte arrays.
- The payload can be fragmented in small parts, the receiving part needs to know when the fragmented payload is complete.
- Minimal serialization overhead.
- The serialization needs to be easily extensible with new types.

The binary standard Abstract Syntax Notation One (in short ASN.1) offers a syntax that allows describing different default types and can be easily extended with new types. Together with DER encoding objects can be transferred with very little overhead. CoinBlesk 3.0 includes a DER serializer and deserializer which allows to transfer Java objects. The implemented parser converts objects of any size and makes them transferable. For all objects used in our case the overhead was reduced to 2 bytes per object. The current implementation supports these types:

DERObject This is the base class for every type. Any kind of byte array payload can be included with this one.

DERInteger This type allows to transfer `BigInteger` objects. Elliptic cryptography uses `BigInteger` for the key and for the signature.

DERString This type allows to transfer `String` objects of any size.

DERSequence This type allows to encapsulate other `DERObject`s as its children. If a message contains more than one type, the message start will be a DERSequence.

Sending elliptic curve digital signatures (in short ECDS) is a common task in the instant payment protocol. If we take this example encoded in DER this would be a `DERSequence` containing two `DERIntegers`, the overall overhead is six bytes.

The serialization together with the step design explained in Section 3.3.1 allows CoinBlesk 3.0 to abstract the communication part in a way that every possible communication channel can be used.

4.5 Transferring Bitcoins

Version 3.0 moved CoinBlesk from a proof of concept application to a mobile wallet application. This transition requires the application to include all features expected from a mobile bitcoin wallet application. Sending bitcoins to other CoinBlesk users can be done using the instant payment feature, but to facilitate transfers to non CoinBlesk users a simple send dialog with QR scan support is implemented. The send dialog also allows users to enter the amount and recipient's address manually.

When it comes to receiving bitcoins CoinBlesk 3.0 offers three options:

Contactless This option uses the CoinBlesk specific instant payment described in Section 3.2.2.

E-Mail This option generates an E-Mail with the default E-Mail client containing Bitcoin address and desired amount.

QR Code This option generates a QR code containing the Bitcoin URI described later in Section 4.7.

The first option is only compatible with another CoinBlesk client. The latter two work with other Bitcoin wallets as well.

4.6 AuthView

CoinBlesk 2.0 supports instant payments via NFC. Due to its short range man in the middle attacks with NFC are possible but not probable. With the introduction of new wireless technologies like Bluetooth LE or Wi-Fi Direct man in the middle scenarios become very realistic. This opens up a new requirement for an authentication method. In the use case of CoinBlesk we have to make sure, that recipient and amount of a payment are not changed along the way.

Section 3.2.2 describes a step where the user is prompted to verify if the device is talking to the right merchant. This authorization step is required to mitigate possible man in the middle attacks. CoinBlesk 3.0 introduces the AuthView to solve this problem. The

AuthView's goal is to generate a pattern that is easily comparable for the human eye even if displayed on small screens.

The view takes an input of arbitrary data and generates a SHA-256 hash. All of the 32 bytes are then used to generate the pattern. The first 3 bytes represent the RGB color of the background. The rest of the bytes is used to decide if two dots are connected or not. The algorithm is deterministic, which means it will always return the same pattern for the same input. The pattern does not have the full 32 bytes of entropy, because no difference is made if dot 1 is connected with dot 9 or dot 9 is connected with dot 1. Since the input data is hashed even the slightest change in the data will result in a completely different pattern.

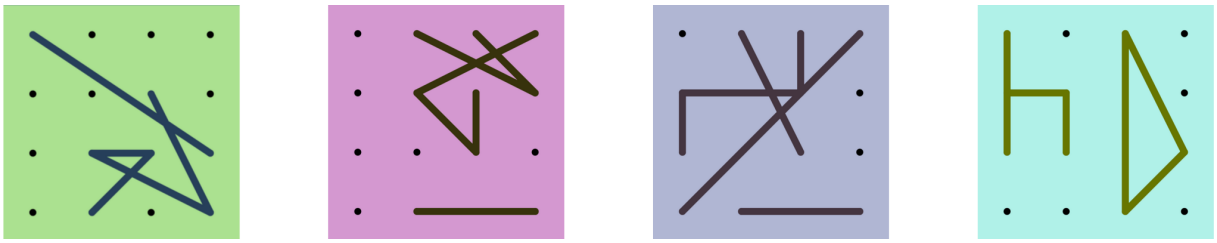

Figure 4.1: Four AuthViews

Figure 4.1 shows four AuthViews with very similar input. One can clearly see the difference of the patterns even if they are displayed on small screens. In CoinBlesk 2.0 no such view was required because it only supported NFC.

4.7 System Integration

The CoinBlesk 3.0 Android application registers the standard URI scheme used for bitcoin transfers. The URI is specified in BIP 0021 [33] and has the following syntax:

```
bitcoin:<address>[?amount=<amount>][?label=<label>][?message=<message>]
```

By registering this scheme with the application, the Android system opens the application whenever a user tries to open a Bitcoin URI. This means that the URI can be written on a NFC tag or QR code. When the user scans it the CoinBlesk application opens directly on the correct screen and includes the relevant data. The CoinBlesk application is not required to run in the background for this feature.

4.8 Testing

Android provides different types of unit tests:

Local Unit Tests This type runs on the local development machine. No Android hardware or emulator is required to run these tests. It can be used if the Android specific classes can be mocked easily. This is ideal to test business logic which seldom is bound to Android functionality.

Instrumented Unit Tests This type runs on Android hardware or emulator. Running these tests can be cumbersome because every code change needs to be compiled and packaged in an Android APK, this APK is then installed on the device and executed. The execution results are transmitted back via the Android Debug Bridge (in short ADB). This kind of tests are used if the Android specific functionality cannot be mocked.

CoinBlesk 3.0 has mostly local unit tests and some instrumented unit tests. The object storage described in Section 4.3 has a unit test coverage of 82%, furthermore some performance benchmarks are available in form of unit tests. The unit tests in the payment module of the Android application covers only parts of the features. Throughout the thesis no efficient method of testing the P2P communication programmatically could be found. The challenge lies in the fact that the communication technologies can only be tested with hardware and two devices are required. This communication part was tested manually, which is not a satisfying solution because this turned out to be very time consuming. The communication parts that could be tested with unit tests like the DERParser were unit tested.

The steps described in Section 3.3.1 where tested as a pipe inside an instrumented unit test.

4.9 Dependencies

In CoinBlesk 3.0 the biggest dependency for server and client remains bitcoinj. Bitcoinj is used in simplified payment verification mode (in short SPV) which makes it feasible for mobile devices to interact with the Bitcoin network. In SPV mode the client does not download the complete blockchain, it downloads only the block headers. By asking a trusted node for transactions matching a certain pattern, the client can then download only the relevant transactions. Bitcoinj has itself a dependency on spongy castle which is a Android compatible clone of the crypto library bouncycastle.

The server uses the Spring framework for providing its endpoints. Data is persisted to the server's database with Hibernate ORM. For serialization/deserialization CoinBlesk 3.0 uses Google's library GSON. The client uses the REST library Retrofit for communication with the server. The implemented QR scanner on the client uses ZXing.

All dependencies used are managed with gradle and are open source.

4.10 Licenses

CoinBlesk 3.0 is an open source project released under the permissive Apache License 2.0. The source code is hosted on GitHub and publicly available. All used dependencies are itself open source. Here the dependecies and their licenses:

- Bitcoinj: Apache License 2.0
- GSON: Apache License 2.0
- Android: Mostly Apache License 2.0, some parts have other licenses (*e.g.,* Linux Kernel GPLv2)
- Retrofit: Apache License 2.0
- ZXing: Apache License 2.0
- prettytime: Apache License 2.0
- Spring Framework: Apache License 2.0
- Hibernate: LGPL

Chapter 5

Evaluation

Many of the evaluation results were collected during the CeBit 2016 in Hanover [34]. The University of Zurich sponsored a booth CoinBlesk was demonstrated. During a full week roughly 200 transactions were made, this was the first real world test for the new implementation.

5.1 Limitations

This section goes through all the limitations that were discovered during implementation and testing. Some of the limitations were discovered only during our real world test at CeBit.

5.1.1 Trust

From a buyer's perspective the new protocol reduces the trust level required in the server to zero. The buyer always receives its refund transaction before it decides to reveal its signature. This mitigates the extortion scenario. There is only one party left that requires trust and it is the merchant. The party receiving a payment requires a small level of trust that the buyer together with the server are not malicious and try to perform double spending attacks. Since we are using multisig addresses, two parties are required to sign a transaction for it to be valid. If both parties decide to perform a double spending attack nothing can stop them.

As stated in Section 2.2.1 the risk of double spending can be very high for clients accepting payments instantly. Since the only service provided by our server is the double spending mitigation in case of an attack the server would loose its credibility immediately. Furthermore an attack scenario requires both parties to be malicious, which reduces its probability further.

At the end of the day it's a trade off between trust and usability. But considering that in the real world instant payments are usually small a merchant would only risk a small

amount before noticing that something is wrong with the server. For larger payments the merchant can still wait for confirmation in the blockchain before accepting the payment.

5.1.2 Blockchain

The current implementation writes every transaction immediately to the blockchain. This is an issue because the blockchain does not scale (see Section 2.1.2). A transaction has to pay a fee for it to be accepted by the miners and included in the blockchain. The fee is not related to the spent amount it's related to the transaction size and how many other transactions are waiting to be included in the next block. Besides technical scaling issues this fee turns out to be quite large considering the small amounts sent instantly. For example at CeBit we were selling chocolate bars for 5.- EUR the fee at that time was around 0.04 EUR. One would need to pay the same amount of fee for a transaction of 2.- EUR which would mean a 2% fee.

5.1.3 Bluetooth

During development Bluetooth in RFCOMM mode turned out to work very stable. The discovery time was around ±6 seconds, the actual data transfer was very fast. Considering that almost every Android device comes with Bluetooth pre shipped, this was a viable trade off. The issue appeared when we tested Bluetooth in a place with many other Bluetooth devices. The discovery became very unreliable and slow.

The problem is that with the Android SDK Bluetooth does not allow the device to advertise its service in a way that a filtered scan is possible (like it is the case for Bluetooth LE). Every device in advertise mode will be listed, the client needs to request from every one of them if a specific service is available or not. For example at CeBit the instant payment over Bluetooth was impossible.

Another limitation is that if the merchant wants to make himself discoverable over Bluetooth he gets a system dialog he needs to accept. The only way to suppress this dialog on Android is by rooting the device. This impacts the user experience, because it requires user interaction.

5.1.4 Wi-Fi Direct

Wi-Fi Direct turned out to work stable only in certain cases. For instance if one of the devices is tethering, our implementation allows payments only in one direction. Sometimes the client needs several retries to establish a connection via Wi-Fi Direct.

Another limitation is that the first time a buyer connects to a merchant via Wi-Fi Direct, the merchant needs to accept a system dialog. The only way to suppress this dialog on Android is by rooting the device. This impacts the user experience, because it requires user interaction.

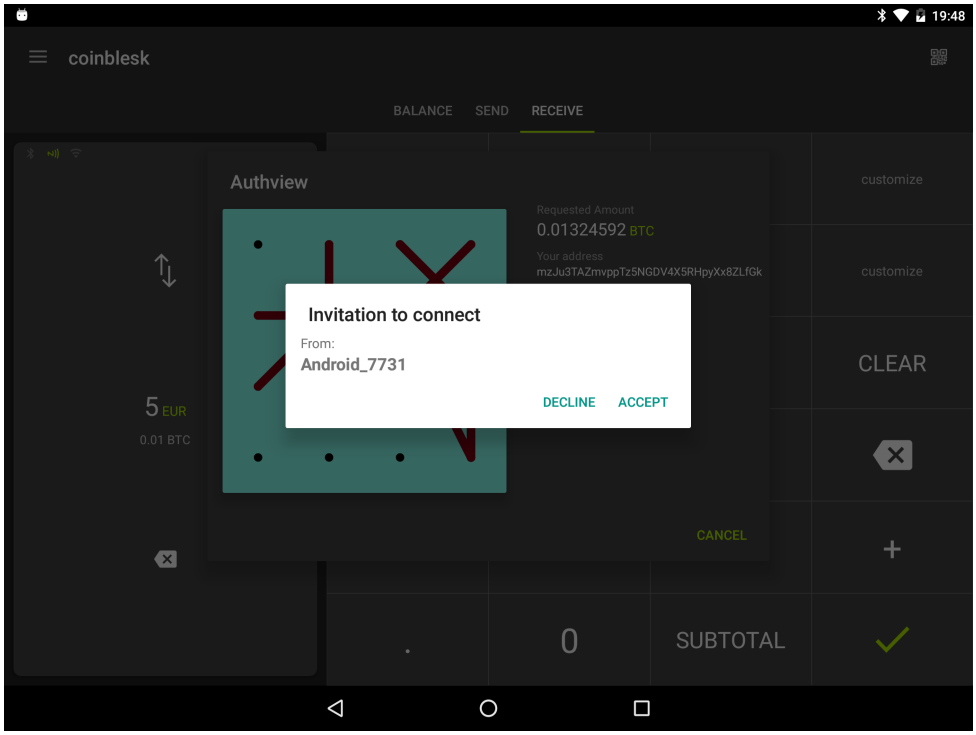

Figure 5.1: Wi-Fi Direct System Authorization View

Figure 5.1 shows a screen shot of a merchant receiving a authorization request for Wi-Fi Direct. This dialog is shown by the system. The developer has no way to suppress the dialog with a clean API.

5.2 Benchmarks

This section starts with some benchmarks taken only for the communication technologies, then goes over to the most CPU intensive and blocking operations and in the end covers the actual payment protocol as a whole.

5.2.1 Communication Technologies

Android does not offer any tool to benchmark the P2P communication technologies. For this reason the open source application AndroidCommunicationBenchmark was created in the scope of this thesis. AndroidCommunicationBenchmark can be found on GitHub [35]. The very simple application allows the user to set the payload size and then select which technology to benchmark. The benchmark consist in sending the given payload back and forth between the two devices. The application automatically takes care of fragmentation in case the payload size exceeds the MTU. Each benchmark result consists of three times:

Discovery Time : Time used to find the other peer.

Connection Time : Time used to connect to the other peer.

Transfer Time : Time used to transfer payload back and forth.

Discover time together with connection time are defined as latency. The results where measured with a OnePlus One running Android 5.1.1 and a Nexus 9 running Android 6.0.1.

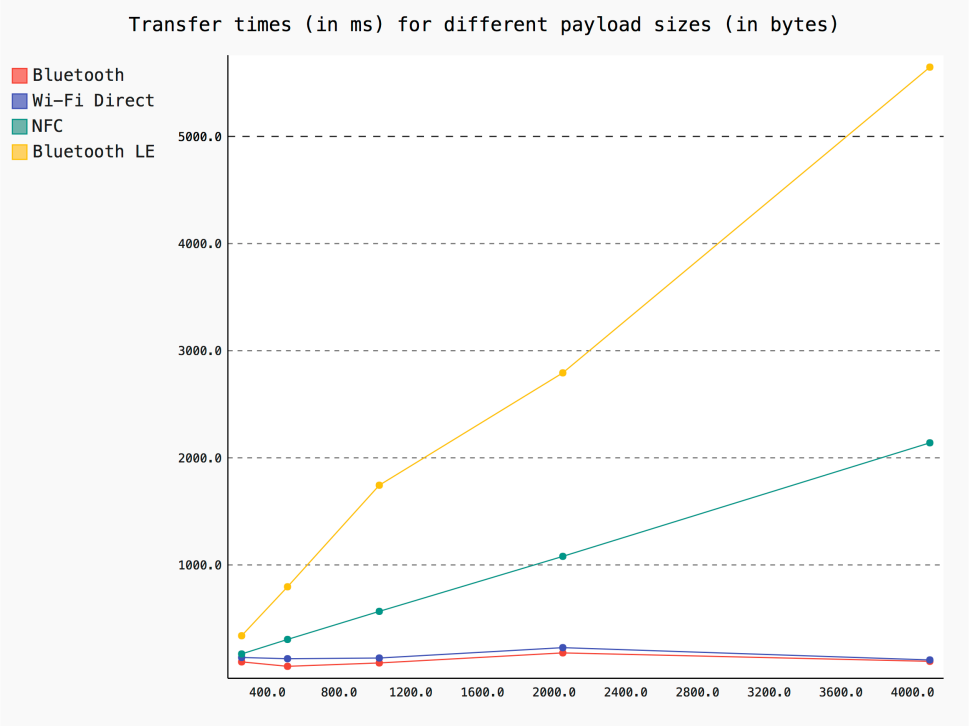

Figure 5.2: Transfer times for different payload sizes

Figure 5.2 shows us how the transfer times (thus time without connection setup) develop for the different technologies. One can clearly see, that Bluetooth LE has a hard time with larger payloads (MTU used is 300 bytes). NFC shows similar results. Bluetooth and Wi-Fi Direct on the other hand are as expected not impressed with a payload of 4096 bytes.

Figure 5.3 shows us where the different technologies spend the most time when transferring a payload of 1024 bytes starting with an unconnected state. Bluetooth and Wi-Fi Direct spend a lot of time before sending the first byte. The huge difference in connection time vs. discovery time can be explained with the actual implementation of the benchmark. Wi-Fi Direct interprets the setup of the actual TCP socket as connection time (the actual Wi-Fi handshaking is included in the discovery time). Bluetooth interprets the setup of the Android `BluetoothSocket` as connection time. Bluetooth LE has a relatively high connection time, this is most probably because during this time the two peers negotiate a new MTU, without MTU negotiation the default is 23 bytes, which make transfer times even higher. NFC is the clear winner when it comes to transfer of small payload sizes (which is the most relevant case for CoinBlesk 3.0).

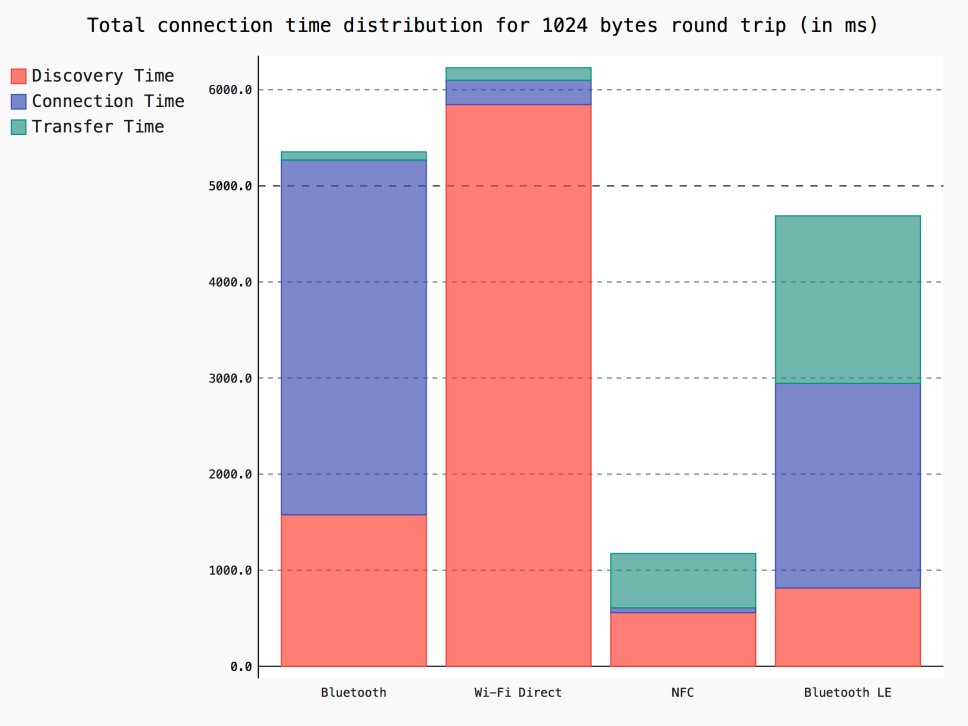

Figure 5.3: Time distribution for a 1024 bytes payload

The presented results where collected in lab experiments, with optimal conditions (devices very near to each other and not to much interference). In the real world Bluetooth takes if there are n discoverable devices in average $(n/2) * (discoverytime + connectiontime)$, which showed us to be an unpractical solution in crowded environments like CeBit.

5.2.2 Blocking Operations

To benchmark each step individually the `ProtocolStepsTest` was created. This test simulates the entire instant payment without P2P transfer overhead. The entire test runs on a single device. The purpose of this benchmark is to identify how long every step takes to complete. The following results were measured using a Nexus 9 tablet.

Figure 5.4 shows us a box plot diagram of the speeds of each step. Most time is used in `PaymentAuthorizationReceiveStep`. This is the step where the server is requested to sign the transaction. It is interesting to see, that in the `PaymentRefundReceiveStep` when the server is requested to sign the second time it takes only half the time. One explanation for this, is that since this is the first time a request is made to the server the server requires some *warm up* time. Every step where the local client is signing transactions computation takes between 80ms (`PaymentRefundSendStep`) and 60ms (`PaymentFinalSignatureSendStep`). These results confirmed the assumption that no alternative to bouncycastle is required to achieve performance gains. The very first step performs only serialization and no cryptographic computations. This confirms that the implemented DERObject serialization performs very fast. The detail information of what

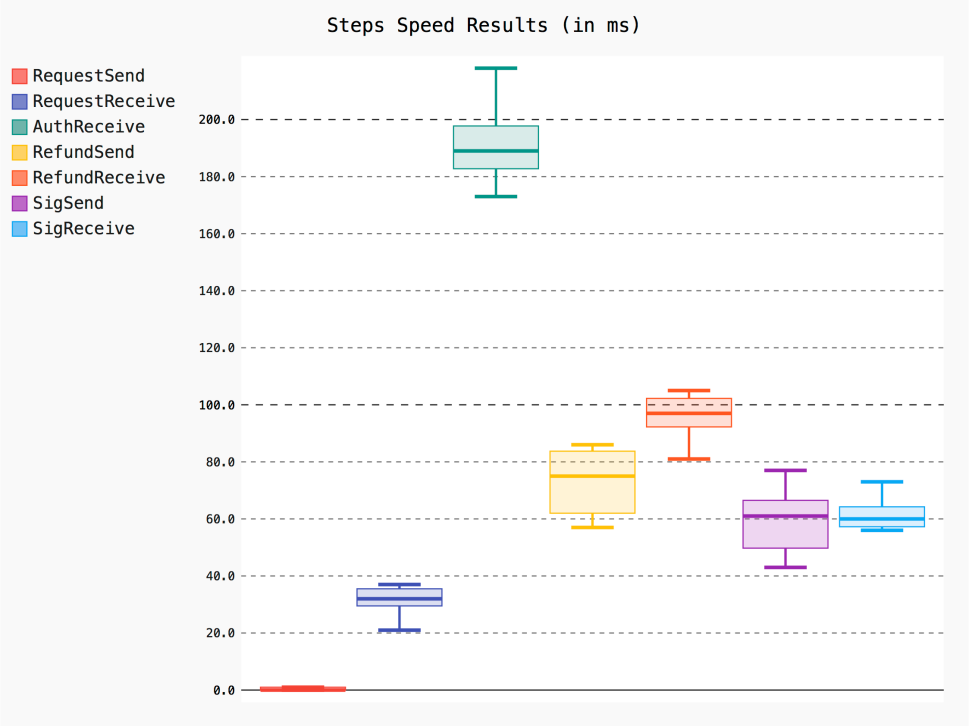

Figure 5.4: Protocol Steps Speed Benchmark

every step is doing can be found in Section 3.3.1. In total the instant payment protocol has a median of 516ms, a minimum of 441ms and a maximum of 565ms.

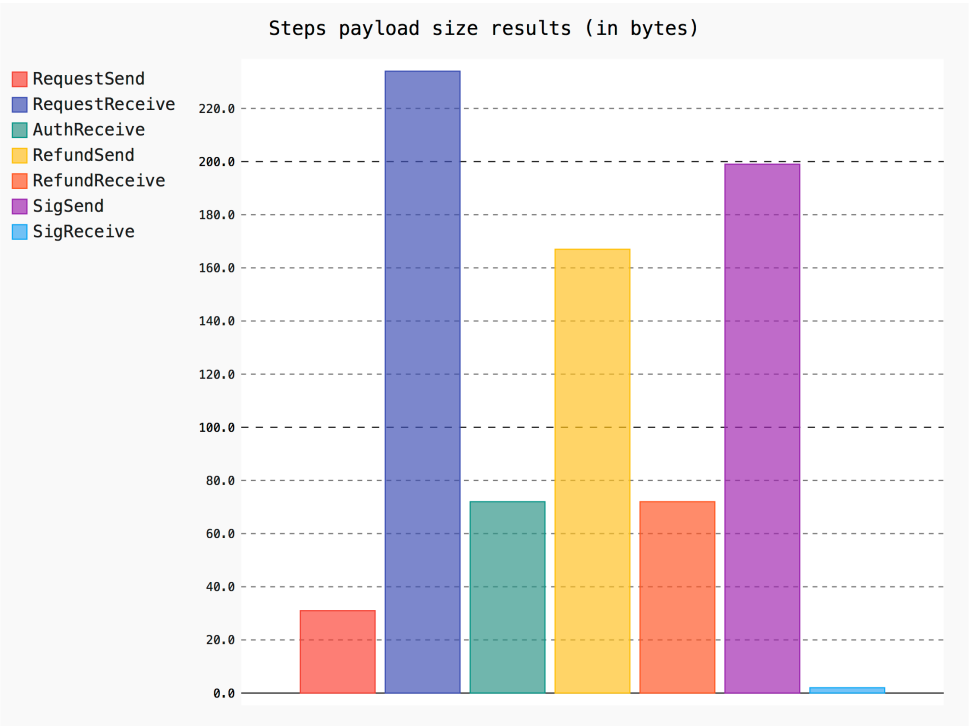

Figure 5.5: Protocol Steps Size Benchmark

Figure 5.5 shows a bar chart of the different payload sizes produced by the steps. These

payloads are the raw bytes that later need to be transferred through the P2P communication channel. Considering all communication channels the minimal MTU is set by NFC with 261 bytes and every step stays below this limit. An external USB NFC reader is also supported which has an MTU of 53 bytes, but since not every user would have such a device, it is counted as an exception. In total 777 bytes are transferred throughout the instant payment. The measured data can be found on the CD delivered with this thesis.

5.2.3 Overall Protocol

The last benchmark in this thesis is the overall instant payment protocol. The following results were measured using a Nexus 9 tablet running Android 6.0.1 (as the merchant) and a Nexus 5 running Android 6.0.1 (as the buyer). The time measurement starts when the `PaymentRequestReceiveStep` was triggered and ends when the `PaymentFinalSignatureReceiveStep` completed. This means the discovery time is not included in the measurement.

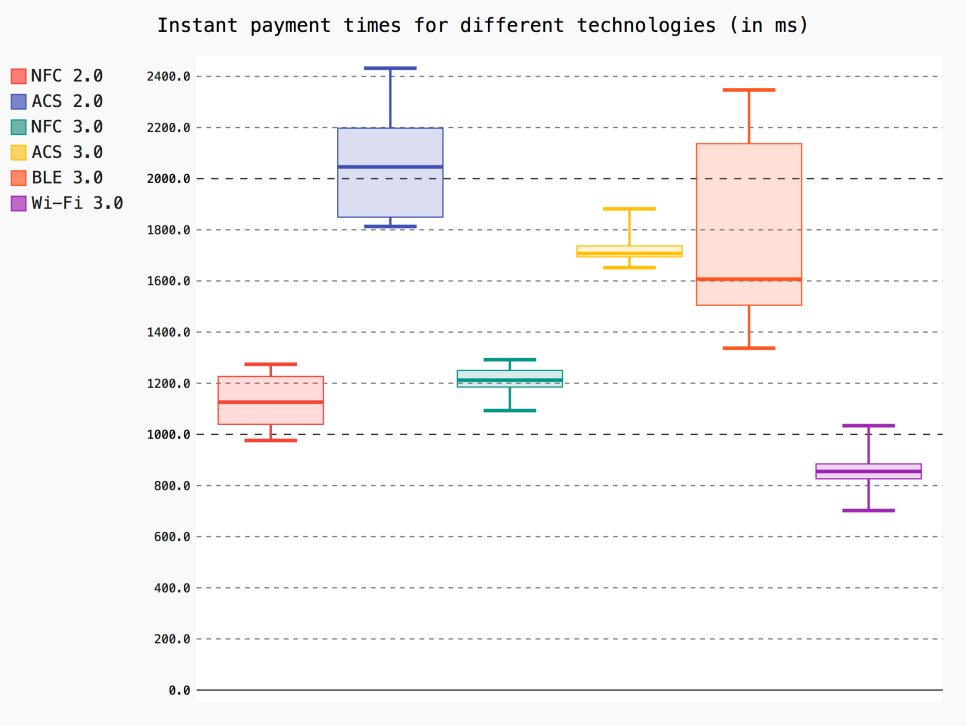

Figure 5.6: Instant Payment Times

Figure 5.6 shows the times for the complete instant payment. The CoinBlesk 2.0 NFC and ACS instant payments were included as well in order to make a comparison of the improved protocol implementation. The ACS reader is an external USB NFC reader that can be attached to an Android Device via OTG. ACS has the limiting property of having an MTU size of 53 bytes, the fragmentation overhead clearly gets reflected in the payment times (for version 2.0 and version 3.0). When compared with CoinBlesk 3.0, the NFC implementation shows almost the same performance. ACS on the other hand

is slightly faster in CoinBlesk 3.0. But the big difference is that in CoinBlesk 2.0 only the payment request and signatures were transferred and CoinBlesk 3.0 now additionally transfers the refund transaction before signing the transaction. This overhead costs two additional round trips. Bluetooth LE takes the longest time to complete, this can be explained on one hand with the slow round trip times and on the other hand with MTU negotiation which takes place on the first contact. Wi-Fi Direct feels very fast to the user, as soon as he accepted the request the payment completes.

5.3 Goals

The following goals were extracted from the original thesis description. During the implementation of CoinBlesk 3.0 new requirements and features came up, which made some of the goals less relevant or even obsolete. The results are based on a personal subjective rating.

Goal	**Result**
Implement a NFC detection mechanism to differentiate between NXP and Broadcom. As the later chipset does not have a transmission limit, fragmentation can be omitted.	*Not accomplished.* The CoinBlesk 3.0 communication abstraction supports fragmentation and reduces packet sizes to below 261 bytes, which makes this goal obsolete.
Implement concurrency and send data as soon as a bit is ready. *E.g.,* message sending and signing can be run in parallel.	*Partially accomplished.* The CoinBlesk 3.0 protocol design reduced the communication to six steps that depend on each other. Wherever possible parallelization was used.
Use a binary diff format instead of sending the full transaction. This requires a deep understanding of the Bictoin pay-to-script transaction support.	*Accomplished.* The CoinBlesk 3.0 protocol design extensively makes use of Bitcoin Script. The transferred data is using a diff format and does not transfer unused data. With the serialization to DERObjects the overhead was drastically reduced.
Investigate the time to create signatures on mobile devices. If performance issues arise, evaluate different PKI implementations (native vs. Bouncycastle).	*Partially accomplished.* CoinBlesk 3.0 uses bitcoinj in combination with bouncy castle, the evaluation has shown us, that most performance gains could be achieved by improving the protocol rather than improving the crypto library.
By far the largest transaction message is the refund transaction. Investigate which communication channel can handle such a message in a reasonable time.	*Accomplished.* All implemented channels can now transfer the refund transaction in a reasonable time.

Source code needs to be open sourced, well documented, and readable.	*Accomplished.*
Evaluate the solution in different scenarios with different devices. Describe for each optimization the performance gain.	*Partially accomplished.* CoinBlesk 3.0 was tested on various devices including smart watches. All optimizations have been well described, but not all performance gains.
Use JUnits where possible to test the protocol. Also try to cover losing packets and protocol restarts with these test cases.	*Partially accomplished.* JUnit is used to test the new protocol. No solution could be found to test the protocol automatically on the actual hardware network connection.
Write a well-structured report that includes motivation, background information, related work, design decisions, implementation details, deep evaluation (also to related work), and conclusion.	*Accomplished.*
If the thesis results are very good, publications of solution and result are planned and the student should support the submission. Furthermore, the student is required to support a potential CeBit 2016 booth, where CoinBlesk with these improvements are presented.	*Accomplished.*

Chapter 6

Summary and Conclusions

CoinBlesk 3.0 introduces various improvements on different levels. The most fundamental change is the protocol design itself. By making sure that the client always holds a refund transaction before locking money no more trust is required in the server. The only party left with trust is the merchant. But considering the merchant is using the service to receive the ability to perform instant payments, the benefits overweights the required trust. Furthermore the merchant can only be attacked if buyer and server would conspire together against the merchant.

The next big improvement lies in the abstraction of the used ad hoc wireless communication technology. CoinBlesk 3.0 brings a stable support for NFC, Bluetooth LE and WiFi-Direct. The evaluation has shown us, that the default Bluetooth in RFCOMM mode can be used but is very unpractical due to its limitations. The proposed abstraction is extensible and can be used for other communication standards in future.

The introduction of these new communication technologies required the development of an easy authentication method. The AuthView covers the requirement with a secure human friendly solution. The collected feedback during the CeBit 2016 show us that the AuthView adds value and is perceived as a secure and understandable solution.

Benchmarks in Section 5.2 show that the new protocol is faster (ACS) or equally fast (NFC) compared to the previous version, even though an additional step is made during the instant payment. This additional step can be removed in a future version by using `OP_CHECKLOCKTIMEVERIFY`. Together with the UX improvements made by Christian Killer CoinBlesk 3.0 introduces a production ready alternative to other known Android bitcoin wallets.

An unsolved challenge remains the fact that the blockchain is growing too fast. The design changes and simplifications in the protocol are a first step towards a solution for trust less off blockchain transactions. CoinBlesk 3.0 is still broadcasting every transaction to the blockchain. Another open point with lower priority is a way to allow CoinBlesk to use multiple servers and move the entire system towards a more P2P architecture.

Bibliography

[1] CoinMarketCap.com. *Crypto-Currency Market Capitalizations*. URL: https://www.coinmarketcap.com/ (visited on 04/11/2016).

[2] Simon Kaeser Jeton Memeti Mehmet Ali Bekooglu. *Bitcoin in Practice; Master Project at University of Zurich*. 3–2014.

[3] Bitcoin.org. *Developer Guide - Bitcoin*. URL: https://bitcoin.org/en/developer-guide#block-chain-overview (visited on 04/09/2016).

[4] *Bitcoin Number Of transactions Per Day*. URL: https://blockchain.info/en/charts/n-transactions (visited on 04/24/2016).

[5] BitcoinWiki. *Script - Bitcoin Wiki*. URL: https://en.bitcoin.it/wiki/Script (visited on 03/26/2016).

[6] Taaki Amir. *BIP 1 - BIP Purpose and Guidelines*. Aug. 19, 2011. URL: https://github.com/bitcoin/bips/blob/master/bip-0001.mediawiki (visited on 04/15/2016).

[7] Andresen Gavin. *BIP 16 - Pay to Script Hash*. Jan. 3, 2012. URL: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki (visited on 04/15/2016).

[8] Andresen Gavin. *BIP 13 - Address Format for pay-to-script-hash*. Oct. 18, 2011. URL: https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki (visited on 04/15/2016).

[9] Raphael Voellmy. "CoinBlesk, a Mobile NFC Bitcoin Payment System". PhD thesis. Aug. 26, 2015.

[10] BitcoinWiki. *Multisignature - Bitcoin Wiki*. URL: https://en.bitcoin.it/wiki/Multisignature (visited on 03/28/2016).

[11] Todd Peter. *BIP 65 - OP_CHECKLOCKTIMEVERIFY*. Oct. 1, 2014. URL: https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki (visited on 04/15/2016).

[12] BitcoinWiki. *NLockTime - Bitcoin Wiki*. URL: https://en.bitcoin.it/wiki/NLockTime (visited on 04/15/2016).

[13] Android. *Connectivity | Android Developers*. URL: http://developer.android.com/guide/topics/connectivity/index.html (visited on 03/28/2016).

[14] Vedat Coskun, Busra Ozdenizci, and Kerem Ok. "A Survey on Near Field Communication (NFC) Technology". In: *Wireless Personal Communications* 71.3 (Dec. 1, 2012), pp. 2259–2294. ISSN: 0929-6212, 1572-834X. DOI: 10.1007/s11277-012-0935-5. URL: http://link.springer.com/article/10.1007/s11277-012-0935-5 (visited on 03/26/2016).

[15] Klaus Finkenzeller. *RFID-Handbuch: Grundlagen und praktische Anwendungen induktiver Funkanlagen, Transponder und kontaktloser Chipkarten*. 3., aktualisierte und erw. Aufl. München: Hanser, 2002. 446 pp. ISBN: 978-3-446-22071-3.

[16] NFC Forum. *NFC Data Exchange Format (NDEF)*. July 24, 2006.

[17] J. S. Lee, Y. W. Su, and C. C. Shen. "A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi". In: *33rd Annual Conference of the IEEE Industrial Electronics Society, 2007. IECON 2007*. 33rd Annual Conference of the IEEE Industrial Electronics Society, 2007. IECON 2007. Nov. 2007, pp. 46–51. DOI: 10.1109/IECON.2007.4460126.

[18] R. W. Woodings et al. "Rapid heterogeneous ad hoc connection establishment: accelerating Bluetooth inquiry using IrDA". In: *2002 IEEE Wireless Communications and Networking Conference, 2002. WCNC2002*. 2002 IEEE Wireless Communications and Networking Conference, 2002. WCNC2002. Vol. 1. Mar. 2002, 342–349 vol.1. DOI: 10.1109/WCNC.2002.993519.

[19] Apple. *Bluetooth for Developers - Apple Developer*. URL: https://developer.apple.com/bluetooth/ (visited on 03/28/2016).

[20] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano. "Device-to-device communications with Wi-Fi Direct: overview and experimentation". In: *IEEE Wireless Communications* 20.3 (June 2013), pp. 96–104. ISSN: 1536-1284. DOI: 10.1109/MWC.2013.6549288.

[21] M. Conti et al. "Experimenting opportunistic networks with WiFi Direct". In: *Wireless Days (WD), 2013 IFIP*. Wireless Days (WD), 2013 IFIP. Nov. 2013, pp. 1–6. DOI: 10.1109/WD.2013.6686501.

[22] Apple. *Multipeer Connectivity Framework Reference*. URL: https://developer.apple.com/library/ios/documentation/MultipeerConnectivity/Reference/MultipeerConnectivityFramework/ (visited on 04/15/2016).

[23] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. "Double-spending Fast Payments in Bitcoin". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. New York, NY, USA: ACM, 2012, pp. 906–917. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382292. URL: http://doi.acm.org/10.1145/2382196.2382292 (visited on 04/02/2016).

[24] The federal Council. *CC 956.1 Federal Act of 22 June 2007 on the Swiss Financial Market Supervisory Authority (Financial Market Supervision Act, FINMASA)*. URL: https://www.admin.ch/opc/en/classified-compilation/20052624/index.html (visited on 04/23/2016).

[25] *SR 958.1 Bundesgesetz vom 19. Juni 2015 über die Finanzmarktinfrastrukturen und das Marktverhalten im Effekten- und Derivatehandel (Finanzmarktinfrastrukturgesetz, FinfraG)*. URL: https://www.admin.ch/opc/de/classified-compilation/20141779/index.html (visited on 04/23/2016).

[26] David Zimbec. "Two Party double deposit trustless escrow in cryptographic networks and Bitcoin." In: Apr. 7, 2014.

[27] TWINT. *TWINT – Die App für Mobile Payment*. TWINT AG. URL: https://www.twint.ch/ (visited on 04/15/2016).

[28] AdNovum. *AdNovum - TWINT: Sicheres Bezahlen per Smartphone*. URL: http://www.adnovum.ch/wissen/focus/projekte/twint_sicheres_bezahlen_per_smartphone.html (visited on 04/15/2016).

[29] SIX. *Privatkunden – Paymit*. SIX. URL: http://www.paymit.com/de/home/individuals.html (visited on 04/15/2016).
[30] GreenAddress. *GreenAddress Wallet protects your bitcoins*. URL: https://www.greenaddress.it/ (visited on 04/15/2016).
[31] *google/gson*. GitHub. URL: https://github.com/google/gson (visited on 04/23/2016).
[32] *dcale/Simple-Object-Storage*. GitHub. URL: https://github.com/dcale/Simple-Object-Storage (visited on 04/23/2016).
[33] Matt Corallo Nils Schneider. *BIP 65 - OP_CHECKLOCKTIMEVERIFY*. Jan. 29, 2012. URL: https://github.com/bitcoin/bips/blob/master/bip-0021.mediawiki (visited on 04/15/2016).
[34] *CeBIT: Big Data, Cloud, Internet of Things, Mobile und mehr*. Mar. 14, 2016. URL: http://www.cebit.de/ (visited on 04/23/2016).
[35] *dcale/AndroidCommunicationBenchmark*. GitHub. URL: https://github.com/dcale/AndroidCommunicationBenchmark (visited on 04/23/2016).

Abbreviations

POS	Point Of Sale
UX	User Experience
MTU	Maximum Transmission Unit
TX	Transaction
UTXO	Unspent Transaction Output
API	Application Programming Interface
SDK	Software Development Kit
BIP	Bitcoin Improvement Proposal
P2SH	Pay to Script Hash
Multisig	Multisignature
P2P	Peer to Peer
NDEF	NFC Data Exchange Format
BT	Bluetooth
WPAN	Wireless Personal Area Network
BLE	Bluetooth Low Energy
NFC	Near Field Communication
SPV	Simplified Payment Verification
ADB	Android Debug Bridge
ORM	Object Relational Mapper

List of Figures

2.1 Schematic visualization of bitcoin transactions. [3] 4

2.2 Schematic visualization of the Bitcoin blockchain. [3] 5

2.3 Bitcoin transactions per day. [4] 6

3.1 Multisignature setup sequence diagram 18

3.2 Instant payment sequence diagram 19

3.3 Instant payment verification 20

4.1 Four AuthViews 27

5.1 Wi-Fi Direct System Authorization View 33

5.2 Transfer times for different payload sizes 34

5.3 Time distribution for a 1024 bytes payload 35

5.4 Protocol Steps Speed Benchmark 36

5.5 Protocol Steps Size Benchmark 36

5.6 Instant Payment Times 37

B.1 CoinBlesk Android Client Screenshots 58

B.2 CoinBlesk Android Client Screenshots 59

B.3 Android Communication Benchmark Screenshots 60

List of Tables

2.1 Comparison of different mobile payment systems 14

Appendix A

Installation Guidelines

CoinBlesk Android Client

CoinBlesk

Usage

Gradle

You can build and install CoinBlesk by using the included gradle wrapper. Install first the required local dependencies into your local repository:

```
$ git clone https://github.com/dcale/Simple-Object-Storage.git
$ cd Simple-Object-Storage
$ chmod +x gradlew && ./gradlew clean build install # for *nix
$ ./gradlew.bat clean build install # for Windows
$ cd ..
$ git clone https://github.com/coinblesk/coinblesk-shared-resources.git
$ cd coinblesk-shared-resources
$ chmod +x gradlew && ./gradlew clean build install # for *nix
$ ./gradlew.bat clean build install # for Windows
```

Once you installed all dependencies you can attach an Android device and run:

```
$ git clone https://github.com/coinblesk/coinblesk-client-gui.git
$ cd coinblesk-client-gui
$ chmod +x gradlew && ./gradlew installDebug # for *nix
$ ./gradlew.bat installDebug # for Windows
```

Simple Object Storage

This is a Java/Android library, that allows you to store/persist and retrieve objects.
The library's only dependency is gson, which is used for object serialization.
The core idea was to have a clean storage API that keeps things as simple as possible.

Usage

Gradle

The easiest thing to try use this lib is installing it to the local maven repo. At the moment I don't host the lib in a repository. Installing the lib to your local repo is easy:

```
$ git clone https://github.com/dcale/Simple-Object-Storage.git
$ cd Simple-Object-Storage

# if you have gradle installed
$ gradle clean build install
# or if you don't have gradle installed:
$ chmod +x gradlew && ./gradlew clean build install # for *nix
$ ./gradlew.bat clean build install # for Windows
```

To use the lib in your project, you can add the following dependency:

```
dependencies {
    compile 'ch.papers:objectstorage:1.1.0'
    ...
}
```

API

The best documentation on how the lib should be used are its Unit Tests. The first thing you want to do to persist an object is to implement the UuidObject interface of directly extend the AbstractUuidObject. The reason you have to do this is because the library relies on UUID's for accessing specific objects:

```
public class TestModel extends AbstractUuidObject {
        private final String name;
        private final String description;

        public TestModel(String name, String description) {
            this.name = name;
            this.description = description;
        }
}
```

Storing an Object

AndroidCommunicationBenchmark

App for testing latency and bandwidth of Android's various communication channels (WiFi, Bluetooth, Bluetooth LE, NFC).

Usage

Gradle

You can build and install AndroidCommunicationBenchmark by using the included gradle wrapper. Install first the required simple object storage lib into your local repository:

```
$ git clone https://github.com/dcale/Simple-Object-Storage.git
$ cd Simple-Object-Storage
$ chmod +x gradlew && ./gradlew clean build install # for *nix
$ ./gradlew.bat clean build install # for Windows
```

Once you installed simple object storage you can attach an Android device and run:

```
$ git clone https://github.com/dcale/AndroidCommunicationBenchmark.git
$ cd AndroidCommunicationBenchmark
$ chmod +x gradlew && ./gradlew installDebug # for *nix
$ ./gradlew.bat clean installDebug # for Windows
```

Appendix B

Application Screenshots

B.1 CoinBlesk Android Client

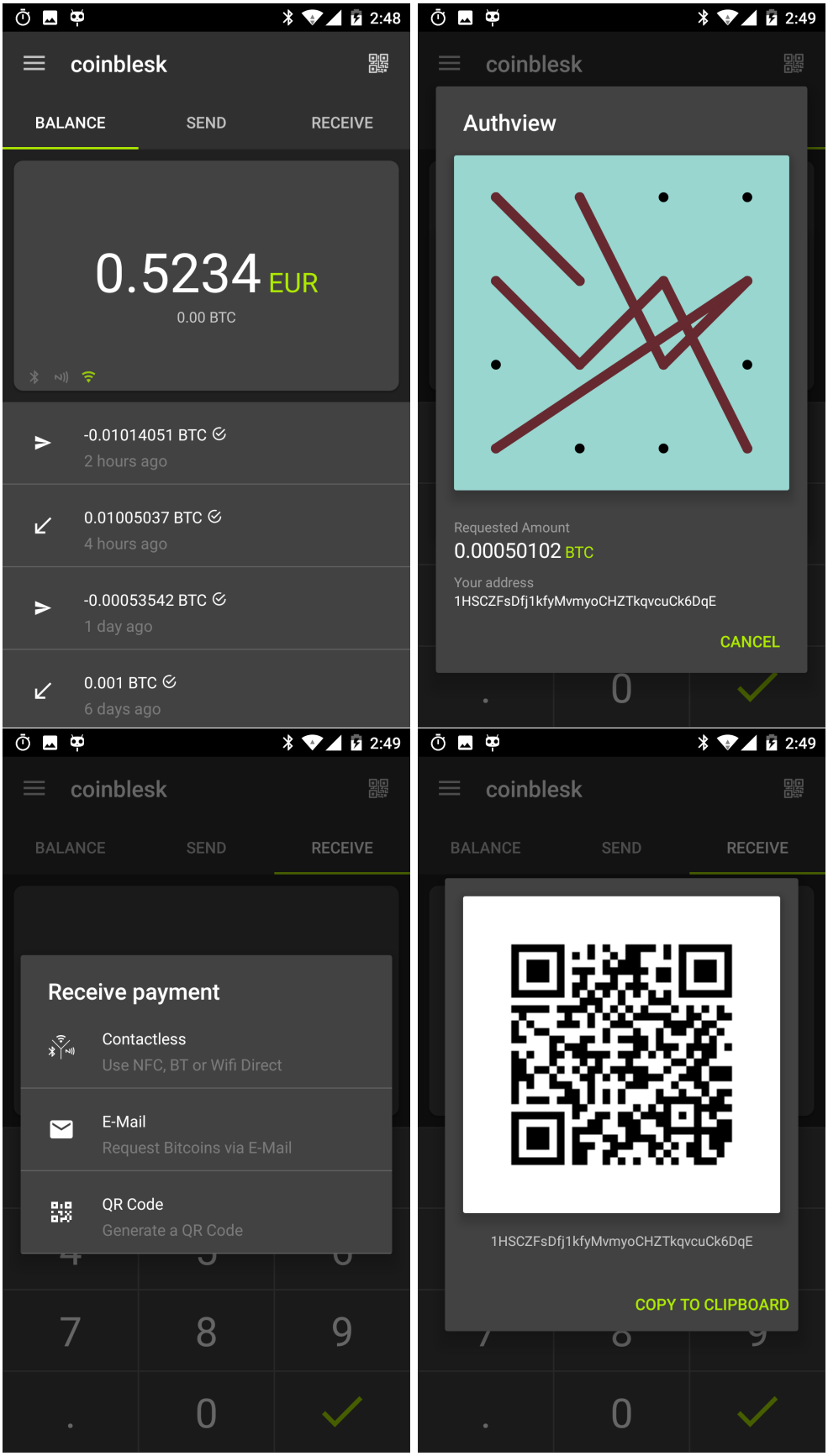

Figure B.1: CoinBlesk Android Client Screenshots

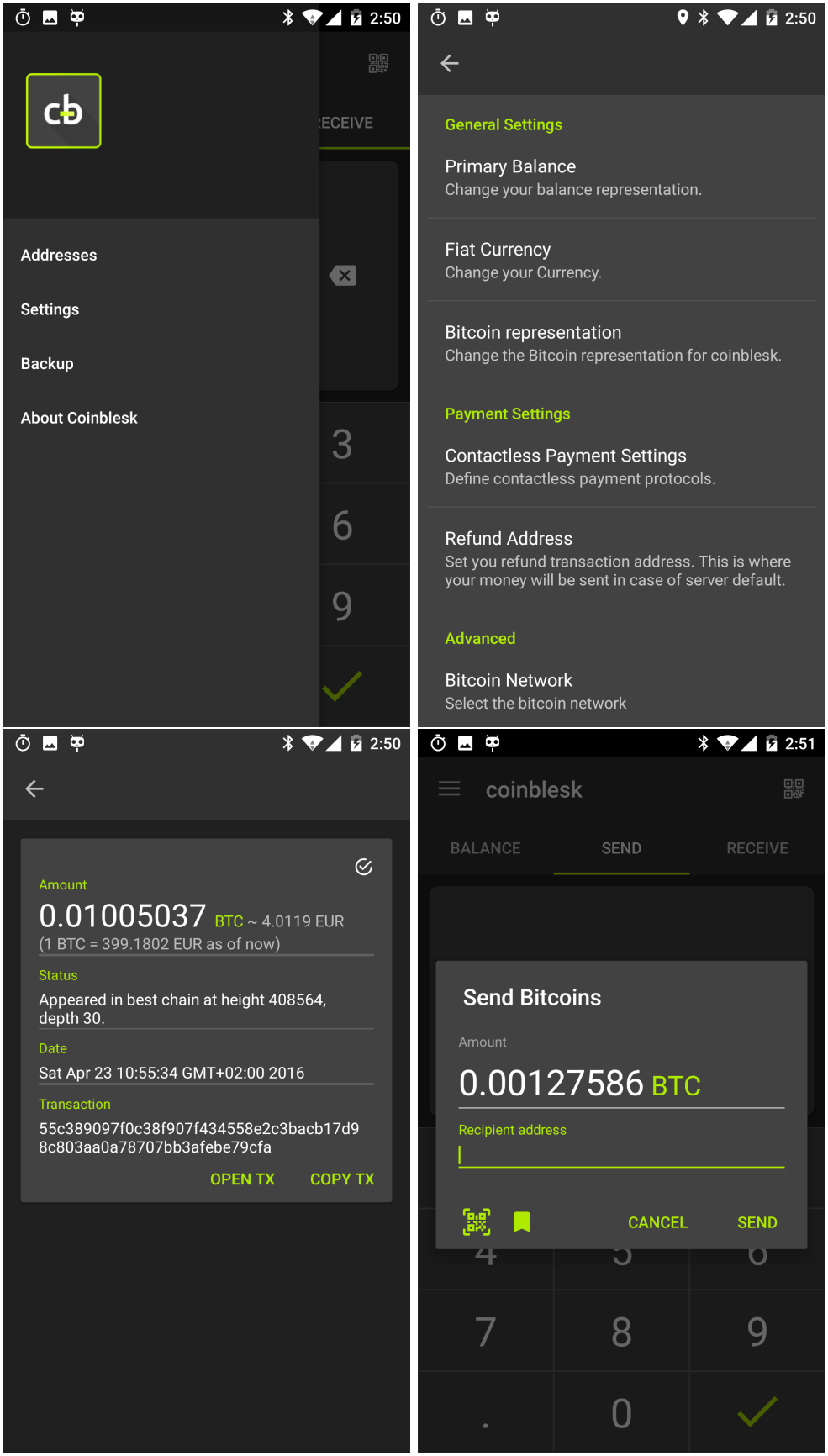

Figure B.2: CoinBlesk Android Client Screenshots

B.2 Android Communication Benchmark

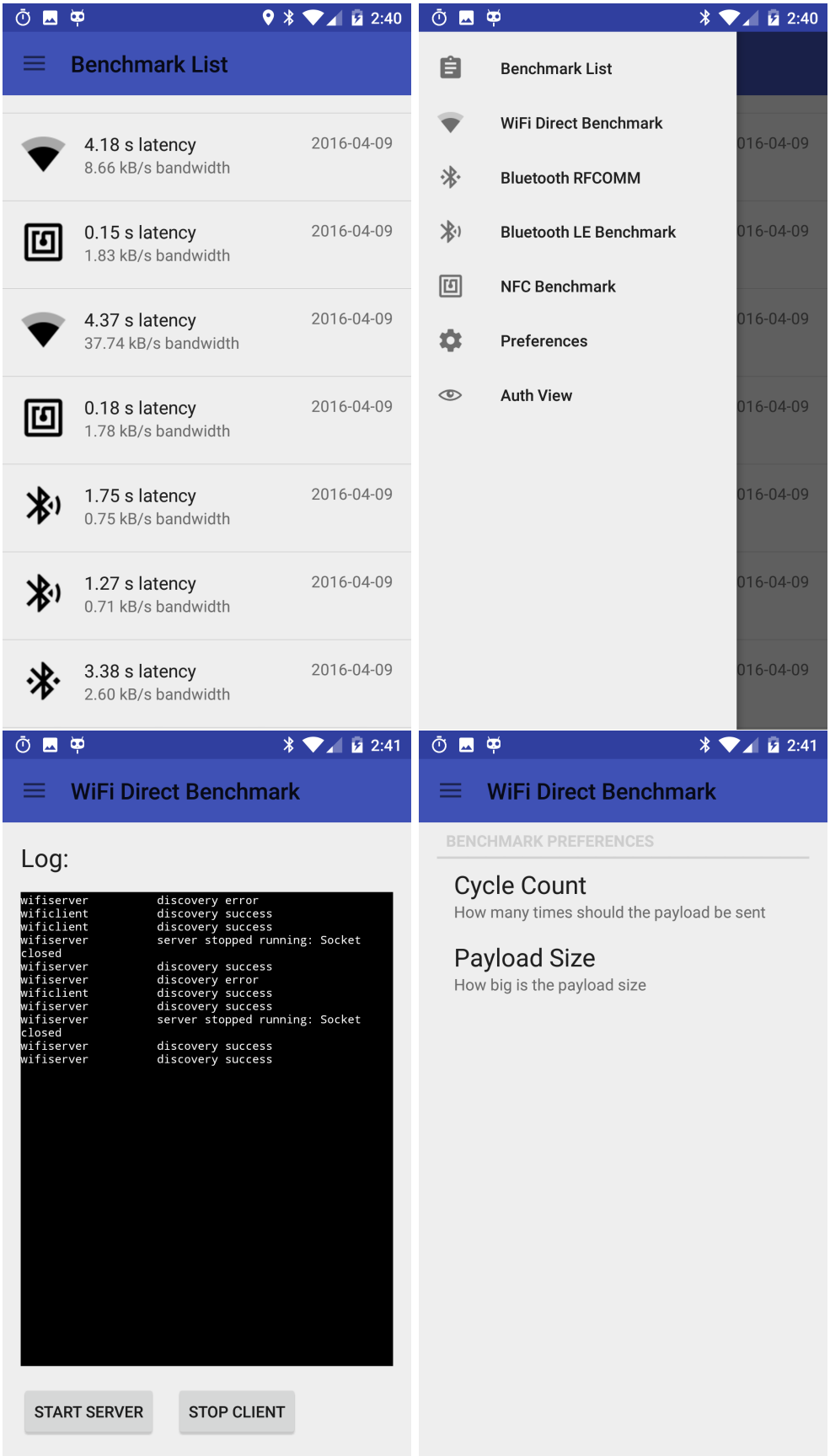

Figure B.3: Android Communication Benchmark Screenshots

Appendix C

Contents of the CD

- Latex source files for this document
- Raw data used for evaluation
- Scripts used for data extraction
- Scripts used for plotting
- Source code CoinBlesk client
- Source code Simple-Object-Storage
- Source code AndroidCommunicationBenchmark
- Source code for UML diagrams
- Apache 2.0 license file