University of Zurich^UZH

Mobile Bitcoin Payment System (MBPS) Federation

Mehmet Ali Bekooglu
Zofingen AG, Switzerland
Student ID: 06-920-771

Supervisor: Dr. Thomas Bocek, Guilherme Machado Sperb, Prof. Dr. Burkhard Stiller
Date of Submission: October 25, 2014

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

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

Bitcoin, a well-known peer-to-peer digital cryptocurrency is not yet feasible for everyday transactions like paying for a meal or a drink. The confirmation time for a Bitcoin transaction takes too long. Therefore, a mobile payment system has been developed to resolve this problem by using a centralized approach which allows to do fast Bitcoin payments by exchanging the transaction information of two participants over NFC. Two test runs have been conducted to evaluate the MBPS. The MBPS has performed successfully, especially the fast NFC payment and the intuitive user interface have received positive user feedback. But the client/server architecture has been complete opposite to the P2P nature of the Bitcoin network.
The goal of this thesis is to develop a federation system of servers. The servers can federate allowing users from other MBPS servers to achieve a instant payment over NFC. Furthermore, a web-frontend is developed to administrate a server in the federation.
The implementation has been tested and evaluated based on the time performance to do a cross-server payment in respect to the previous version. Compared to the previous version, a cross-server payment requires on the Nexus 7 47.8% longer to receive a given amount and and 36.7% longer for sending an amount to a user. Summarized, the payment took in average 1 to 1,5 seconds to complete.

Zusammenfassung

Bitcoin, eine bekannte digitale peer-to-peer Kryptowährung, ist noch nicht geeignet für alltägliche Zahlungen wie besipielswies für eine Mahlzeit oder ein Getränk. Die Bestätigung einer Bitcoin Transaktion dauert zu lange. Um dieses Problem zu lösen wurde ein mobiles Bezahlungssystem entwickelt, welches auf einem zentralisierten Ansatz basiert. Die Transaktionsinformationen zwischen zwei Teilnehmern werden dabei über NFC ausgetauscht. Zwei Testläufe wurden durchgeführt um MBPS auszuwerten. MBPS wurde erfolgreich ausgeführt, vor allem die schnelle Transaktionsabwicklung über NFC und die intuitive Benutzeroberflüche haben positive Rückmeldungen erhalten. Die Client/Server-Architektur hebt jedoch die Vorteile des dezentralen Netzwerkes Bitcoin wieder auf.
Das Ziel dieser Arbeit ist, ein Föderationssystem von Serveren zu entwickeln. Die Server können sich vereinigen um Teilnehmer von anderen MBPS Servern zu erlauben, eine sofortige Bitcoin Zahlung über NFC zu verrichten. Des Weiteren wurde eine Web-Seite entwickelt, welche die Administration eines Servers in einer Föderation erlaubt.
Die Umsetzung wurde getestet und ausgewertet. Die Zeit, die eine serverübergreifende Transaktion benötigt, wurde evaluiert und mit der vorherigen Version verglichen. Im Vergleich zur vorherigen Version, benötigt eine serverübergreifende Zahlung auf dem Nexus 7 47.8% länger um einen bestimmten Betrag zu erhalten und 36,7% länger um die Summe zu einem Teilnehmer zu schicken. Im Durchschnitt benötigt eine Transaktion im neuen System 1 bis 1.5 Sekunden.

Acknowledgements

I would like to thank all the people involved in this master thesis. First of all, I thank Prof. Dr. Burkhard Stiller for giving me the opportunity to write this thesis at the Communication Systems Group.
I would also like to thank Dr. Thomas Bocek and Guilherme Machado Sperb for providing the idea of this thesis and for the great support during the different phases in the thesis.

Contents

Chapter 1

Introduction

Bitcoin [1] is an open, fully distributed P2P digital currency that is gaining more and more popularity and even the governments are aware of Bitcoins [2, 3]. However, everyday transactions such as paying for a coffee or other kind of micro payments are not yet feasible when it comes to Bitcoins. The reason for this is that the seller (or the peer that is going to receive the Bitcoins for a service or good) has to wait an amount of time for the transaction to be confirmed. A transaction is confirmed when it is included in a block which is published to the network [4]. At this time, it takes in average around 10 minutes (with a few exceptions depending on the blockchain) for a block to be created [5]. This means that Bitcoin transactions can not be carried out instantly and both seller and buyer have to wait a quite large amount of time before closing the deal.
Based on the nature of the Bitcoin protocol and implementation, it is not enough to have only one confirmation since a number of blocks can be rejected in favour of another branch. A transaction should only be considered as confirmed after 6 blocks are verified [6]. This means that if the seller is not willing to risk losing Bitcoins (even in the scope of micro payments) s(he) has to wait for 6 confirmations. Based on the time, this requires both peers to wait more than 56 minutes before closing the deal. Thus, Bitcoins are not applicable in its current implementation and approach when it comes to everyday transactions, e.g. paying in a market or in a restaurant.
However, there is the approach of just broadcasting a transaction without waiting for any confirmation. This so called ”fast payment” takes a few seconds to be completed [7]. But there remains the risk of double spending. Therefore, this approach was not taken into consideration for the MBPS project in order to eliminate the double spending risk for the seller. To address and resolve these limitations mentioned above, a mobile Bitcoin payment system which allows exchanging Bitcoins instantly was developed within the scope of a master project and carried on to different master thesis. In addition, the whole project [8] focused on security aspects, two-way Near Field Communication (NFC), reducing the number of Bitcoin transaction by introducing a clearing center and an easy, intuitive user interface. This system has been successfully tested in two test runs with the UZH Mensa Binzmühle in Oerlikon [9]. The first test run took place for one week in February 2014 and the second test run in September was extended to three weeks. During these test runs, students, employees and visitors had the chance to test the mobile solution CoinBlesk (formerly MBPS) and pay their consumption with Bitcoins using CoinBlesk. To give the

test persons easier access to Bitcoins a Bitcoin exchange point was provided where test user could buy Bitcoins to top up their accounts or sell them again. In the second test run the user had additionally the opportunity to top up their user accounts with Bitcoins by a Bitcoin ATM (BTM) which was allocated by SBEX [10].

1.1 Motivation

The first test run of the MBPS project was accomplished with success. The MBPS was evaluated reliable and quite handy. The idea of paying the consumption by a device appeal to the costumer of the Mensa. Participants had to wait approximately 2-3 seconds for a payment confirmation with the MBPS. Nonetheless, during the development and especially in the test-run two main flaws were disclosed. The first flaw, regarding the handling of the application revealed that some test users had difficulties to properly do a payment over NFC. This can be attributed to the inexperience of using a device in this use case. One challenge a few participants had to figure out was how to establish a successful connection. This was caused mainly because of the poor feedback and support by the MBPS client. Furthermore, the small range of the NFC technology intensified the occurrence of failures. The second flaw, emerged due to the NFC protocol and the manufacturing of different NFC chips for Android devices. Thus, in the test run the MBPS application was limited to just a few devices of the Google Nexus models. In overall, the test run showed that the MBPS project had potential to be a good alternative for a payment system and either Bitcoin to be used in everyday transactions. Therefore, to improve the MBPS application three potential future works emerged for different areas [8]. These potential works were taken further as master thesis's.

One thesis addressed the improvement of the the MBPS client. To overcome the users inexperience with the NFC the user interface has to be enhanced with supporting feedbacks like progress dialogues. Further, a few existing views of the MBPS has to be adjust to the needs of the user. And, also instead just requesting Bitcoins which was good enough for the Mensa test run, users have to be allowed to send Bitcoins by entering the amount [11].
Another thesis focused to improve the flaws related to the NFC protocol. A feature has to be to retransmit packets which get lost due to the connection failure. Another feature, has to be the recognition of the sessions, e.g, during a payment a user can take a way the device to accept or reject the request and re-establish the NFC contact to proceed. Furthermore, the time for a transaction has to be improved by making it faster. A last requirement has to allow more devices to use the MBPS application with different NFC manufactures [12].
The last thesis which was the scope of this thesis addresses the approach of the centralized server and extent this by making it decentralized. This means, the MBPS application can exists with different instances and organization. Instances can manage their own user accounts with Bitcoins. Furthermore, instances can be federated to allow users from different organizations to perform a payment over NFC.

Thus, the goal of this thesis is to create a federation mechanism of MBPS servers. The current architecture is a client/server architecture, which is the complete opposite to the Peer-to-Peer (P2P) nature of the Bitcoin network. The idea is to create a federation based on MBPS servers to create a super-peer type of network. To enable payment in less than a second, a client/server architecture will be used. However, MBPS servers can federated to allow users from other MBPS servers to perform instance payments over NFC. The following example in Figure 1.1 illustrates the use-case: organization X and organization Y have a federated MBPS system. If a user of the server X is going to eat at the organization Y, the payment would go from Y's server to X's server and perform the transaction without any Bitcoins involved yet. The outstanding amount is then sent via the Bitcoin system once or twice a day between the two parties. The steps to perform the cross-server payment is shown in Figure 1.1 below. The first step is to initialise a NFC connection, then the payment request is send to the server of the Initiator, here *Y*. The server *Y* sends the information to the server *X* and the response is returned back to the user of *Y*. Furthermore, a web-frontend solution has to be developed to manage the federation between different instances or organizations.

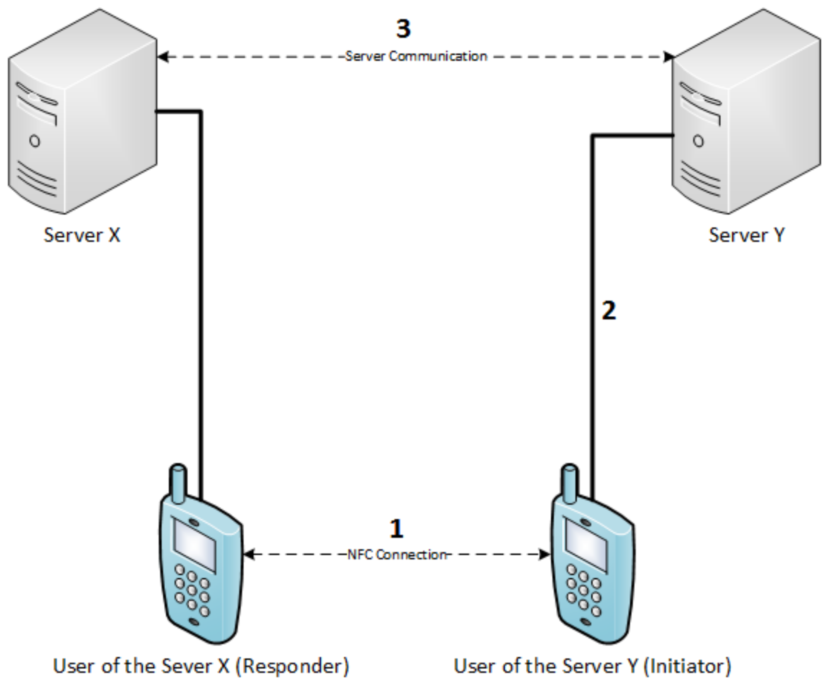

3
Server Communication
Server X
Server Y
2
1
NFC Connection
User of the Sever X (Responder)
User of the Server Y (Initiator)

Figure 1.1: Example of a Federation Mechanism

1.2 Description of Work

This thesis covers the design, implementation, and evaluation of the MBPS federation system. The existing MBPS version is used as a basis to implemented the new extensions. The thesis focuses mainly on the server side (Java/Tomcat/Spring), and there has to be a need to adapt the user interfaces of the Android Client. This thesis discusses in details the design advantages and disadvantages of the federation mechanism. In addition, the federation has to be adminstarted over web interface.

1.3 Thesis Goals

The goal of this thesis is to add a federation system to the existing MBPS. In order to complete this thesis, the following tasks are done:

- Analyse decentralized federation mechanims including escrow Bitcoin accounts [13]. The market mechanisms of the federation mechanism is discussed.
- Implement the federation mechanism into the current MBPS application. Develop a web-fronted for administration.
- Source code needs to be open sourced, well documented, and readable.
- Evaluate the solution in different scenarios with different MBPS servers.
- Evaluate he federation mechanism by deploying test servers.
- Use JUnits to test the federation mechanism.

1.4 Outline

Chapter 2 presents background information about the the MBPS applications (MBPS and CoinBlesk) used in the test-runs in February 2014 [8] and in September 2014 [11, 12]. It also covers the trust issue based on game theory and introduces escrow accounts to overcome the trust issue. Chapter 3 discusses related work. In Chapter 4 the design choices and the analysed use-cases for the federation solution is determined. Additionally, the approach and features to accomplished the defined goals are explained. Chapter 5 explains the architecture with main features. It also introduces the implementation and challenges during the development. The following Chapter 6 compares the payment transaction over different servers in benchmark to previous versions without a federation mechanism. The last Chapter 7 summarizes the key findings and draws a conclusion. It also outlines possible future works.
Aforementioned, in this Chapter the name of the MBPS has been changed to CoinBlesk before the test run in Spetember 2014 was held. In this thesis, the name MBPS is used

through out the thesis to not get confused. The name MBPS addresses both versions expect the discussion is related to CoinBlesk. The reason to choice the older name is based on the starting date of this thesis.

Chapter 2

Background

This Chapter provides background information about the MBPS and the improved application CoinBlesk. This gives a overview about the whole scope and key features of the MBPS project. Further, to comprehend this master thesis in Section 2.3 and 2.4 key functionalities are introduced.

2.1 MBPS

Aforementioned, the goal of the MBPS project was to develop a mobile Bitcoin payment system which allows exchanging Bitcoins instantly by overcoming the time constraint of the Bitcoin core network. A confirmation of a Bitcoin transaction is considered as confirmed after 6 blocks, which takes in average an hour [14]. However, the project focused in addition on security aspects, bidirectional NFC connection, and reducing the number of Bitcoin transaction by introducing a clearing center.
The drawback with the confirmation constraint was solved by introducing a centralized system which keeps all accounts of each user on the MBPS server. Therefore, the MBPS is based on a prepaid approach, where user can top up their accounts or transfer Bitcoins out of the system. This design allows users to do payment instantly without waiting for any Bitcoin block to be confirmed. Another reason for this approach was to omit the danger of double spending attacks [7, 15].
A further feature of the MBPS in the master project was the implementation of a clearing center. A user A pays several times within a day a given amount to user B or vice versa. Instead of generating for each transaction a Bitcoin transaction, the system can initiate a core Bitcoin transaction at a given time or for a given threshold. This kind of clearing center is preferable to keep the Bitcoin core transaction low, since transaction fees will play a bigger role in the future [4].
The only requirements for users to exert the MBPS and conduct a transaction among each other is by means of an mobile device running on an Android operating system (OS) Version 4.4.
The MBPS solution was tested in a real one week Mensa test-run at the Mensa Binz in

Oerlikon. Summarized the evaluation of the test-run and the conduct survey following results are stated [8]:

- Participants were pleased with a payment solution of the MBPS to pay for consumption at the Mensa. Especially, the simple use and transferring Bitcoins (technically the accounts are matched) from one user to another.
- Some drawbacks according the user interface and handling the device during a transaction with the NFC became apparent. There were some potential improvements detected.
- The restriction to Android 4.4 and the availability of only few devices (i.e., Nexus 5, Nexus 7, Nexus 10, and ACR122u) for the test run was criticised. It is to mention at the time when the test-run was conducted, the OS 4.4 had a marked share of 2% [16].

2.2 CoinBlesk

CoinBlesk is the follow-up payment solution based on the aforementioned MBPS application. Two different master thesis contributed to CoinBlesk application [11, 12]. The goal of the CoinBlesk application was to cover the exposed flaws from the MBPS and add additional features that were missing. But the goal of the MBPS is still to provide a mobile payment system that overcomes the time constraint of the Bitcoin core network. Summarized, these are the key features of the CoinBlesk application:

- Allow user to easier use the application especially regarding the NFC. During a payment with NFC a supporting feedback solution has to be provided. The user has to be informed if the NFC connection is still proceeding or if a payment is accomplished with success or not.
- The interface of the application has to be more intuitive and design has to be optimized. As a result, a user of CoinBlesk has to take less actions to be provided with accounts information and has to be much faster ready for a payment.
- The existing MBPS protocol using NFC has to be improved. Has to be more resilient when using devices from different vendors with different NFC chip manufactures.
- Retransmission of packets has to be implemented with a NFC handshakes, that can keep messages alive.
- The transmission and the signing of the request has to be delivered much faster than the MBPS application.

These goals where achieved with success and a second Mensa test run were conducted at the UZH Mensa Binzmühle in Oerlikon. The test run was set up for 2 weeks and was later extended to 3 weeks. In addition to the first test run a Bitcoin ATM provided by the SBEX [10] was positioned in the entry hall of the building Binzmühle. This allowed participants to get easy access to Bitcoins. The CoinBlesk application run with no major issues. The transaction were proceeded much faster and smoother than in the first test run. The waiting time was diminished to around 1 second and by enabling the auto-accept the transaction was even much faster. In comparison, the market share of Android devices running on KitKat (4.4) during the test run was between 20-22 % [17]. Besides the NFC chip manufacturer Broadcom, devices with NXP chips have also taken part in the test run. Except one incident with HTC One (M7) showed that the CoinBlesk application was not able to fully operated properly. For further details related to both applications see Appendix A.

2.3 Game Theory - Trust

The new feature a federation of autonomous servers targeting instant payments between each other brings further problems to be solved. Since Bitcoins have monetary value, the risk of losing money increases rapidly. Therefore, the relation between servers is a key point and to understand the risk of having cross-server payments game theory supported the idea of trust relation.
The game theory, developed in economics, addresses exactly the relationships between agents (individuals, organizations) in a particular model and predicts their optimal decisions with the use of applied mathematics. Game theory is more about decision-making, i.e., it is concerned with choices and strategies than seeking for the best solution [18]. Briefly, the game theory analyses to answer the following questions:

- What strategies are for this model possible?
- What solutions can be achieved?

The object of the study in game theory is the game, which can have a number of criteria (number of players, strategies per player, known information, etc.) and this can be used to classify the game [19].
For this thesis only the trust game is considered as applicable. The trust game is represented by the games Dictator and Ultimatum [20, 21]. Both games consists of two participants that are anonymous paired and obtain a certain amount of money. The first individual is told to send some amount of his money to an anonymous second individual. In addition, the first individual is also informed, that the money which is send will be tripled by the experimenter and passed to the second individual even if this money is zero. The same is told to the second individual [21, 22]. The predicted results under the standard economics assumption of rational self-interest, the first individual would choose to send nothing. Even with perfect information about the mechanics of the game. The first individual's action in this game describes the nash equilibrium. But according to the

results in the experiments [21, 22] the predicated results were violated for the first and second individual. In overall, the outcome in both experiments stated that the results varied sharply from those results predicated in the standard economic assumption of pure self-interest [23]. But the amount between the individual depended heavily on the level of social information from each other. For more detailed information and definitions about game theory and the experiments see Appendix B.
Based on the assumption of rational self-interest a federation of servers will fail without any additional precautions. The one responding strategy is the benefit that can be made without relaying on trust. Hence, the only solution is not to act in a decentralized network, which is not the solution this thesis is seeking. But the results of both experiments [21, 22] disproved the standard economic assumption and even than there is no guarantee servers will not act for their own self-interest in a network of homogeneous servers. In a federation mechanism For this scenario, there are several strategies possible from self interested servers to servers supporting the idea of trust in the network. The conclusion of the experiments also indicated that individuals paired with the same individual several times they formed a reputation of trustworthiness. A federation without precaution is a big concern and a kickback for systems operating especially with Bitcoins. Therefore, a model dealing with trust issues is indispensable for this thesis.

2.4 Escrow

Aforementioned, a model that includes trust and securities avoiding fraud has to be implemented. Hence, escrow systems have to be judged has reliable solutions for transactions between two untrusted parties. Escrow is well known in the information system (IT) as software escrow, key-escrow and in the databases but those were neglected and focused only on escrow accounts having a type of a bank account held by third parties. According to [24], escrow account is defined as:

> A financial instrument held by a third party on behalf of the other two parties in a transaction. The funds are held by the escrow service until it receives the appropriate written or oral instructions or until obligations have been fulfilled. Securities, funds and other assets can be held in escrow.

Based on the definition the third party acts as an neutral intermediary and ensures that the paid amount by the buyer will be transferred from the escrow account to the seller of a service or good, only when all conditions are satisfied by both parties. Since the payment transactions are done in Bitcoins, the escrow service has to operate in Bitcoins. There are several third party providers offering escrow accounts with Bitcoins as payment currency [25]. But in a federation of servers there is no clear assignments which server is in a buyer or in a seller role. Second, to find a third party which satisfies both server in a federation is difficult and it has to be expected that a server interacts with more than one server indicates more involved third parties. On a trust level, every instance has to be very caution when sending Bitcoins to a third party escrow partner. The danger of a

third party abscond with the received Bitcoins is high. In overall, an escrow account with a neutral third party operating with full trust from both sides (buyer and seller) is not applicable in this thesis and not considerate.
However, in the Bitcoin community there is a willingness to improve the Bitcoin transaction between individuals and this can be seen a possible solution for the escrow issues trusting a third party. The keyword is multisig, a multi-signature transaction which pioneered and formalized into the standard Bitcoin protocol in 2011 and 2012 [26]. The reason for this change of think was tp protect against the large number of Bitcoin services fall over and vanished, in the last years. For example, in October 2013 Silk Road an anonymous marketplace shut down and lost $2.7 million worth in BTC [27] or in February 2014 the world largest Bitcoin exchange platform Mt.Gox shut down, either, and lost Bitcoins worth of $470 million [28, 29, 30]. According to [26], the difference between current Bitcoin addresses with one associated private key (described as send-receive system) and the future use of Bitcoin addresses having multiple associated private keys with multi-signature changes the security aspect. Multisig major purpose is to protect consumers from finalized Bitcoin transaction. To get a better overview of the capabilities of multi-signatures in [31] it is described as followed:

> Bitcoin includes a multi-signature feature that allows a transaction to require the signature of more than one private key to be spent. It is currently only usable for technical users but a greater availability for this feature can be expected in the future. Multi-signature can, for example, allow an organization to give access to its treasury to its members while only allowing a withdrawal if 3 of 5 members sign the transaction. It can also allow future online wallets to share a multi-signature address with their users, so that a thief would need to compromise both your computer and the online wallet servers in order to steal your funds.

The described m-of-n signatures can be explained in a concrete example, e.g., a 2-of-3 signature is associated with 3 private keys, and sending Bitcoins from this address requires signatures from at least 2 keys. The point is that there are total of 3 people(or entities) involved in a transaction and 2, usually a less number, must sign off. Such a case, prevents one person from having the ability to steal Bitcoins and ensures that a predefined set of persons have to agree whatever transaction is is taking place. The multisign enables several new contract designs including the multi-signature escrow [32].
There are two approaches to describe the multi-signature escrow. One option mentioned before, is the 2-of-3 multisig transaction. The buyer collects one of their own public keys, plus a public key each from the escrow party and the seller with the the requirement that the Bitcoins can be spent if 2 of them sign off. In this scenario, no party can control the Bitcoins. Even without needing the third party to be involved, the Bitcoins can be released by the buyer and seller. Only if buyer and seller cannot agree, then the third party must break the tie. In such a case, the third party can freely decide the matter, since only one other party has to sign off. This prevents the third party to abscond Bitcoins but there is still the issue that both the buyer and seller have to trust the third party.
Another approach is the two party escrow using a 2-of-2 multisig transaction. In this scenario the third party is removed and this leads to more complex behaviour. Therefore,

a scenario, where Bob does nothing (not signing), i.e., for Alice the Bitcoins in the escrow are lost. One option to avoid this, is to lock up some of Bob's Bitcoins along with Alices's. Technically, this means Bitcoins from Alice and Bob are collected together into a single multisig output. Constructing such a transaction is trivial as long Alice's and Bob's Bitcoins are hold on one centralized service. But this is unusable for accounts in different wallets. The second option is to setup an automatic release from the escrow at predefined points in the future, where the Bitcoins are either sent back to Alice, or split between both Alice and Bob. In a concrete example, this is done, first Alice creates the initial transaction sending Bitcoins into escrow and she signs the transaction in order to determine the transaction ID. Then, instead broadcasting the transaction, she keeps the transaction secret. Alice sends the transaction ID to Bob and tells Bob to sign the new transaction. This transaction will be locked, so that it can only be redeemed at a specified time in the future. After Alice gets back the signed transaction from Bob, she releases into the escrow. The only problem is if they can agree on the refund [33].

Summarized, the standard escrow accounts are unusable, because of the trust issue to third parties. The multisig escrow transactions are an improved form for more protection in the Bitcoin community. In addition, multi-signatures can eliminate the trust issue to third parties, even exclude third parties from an escrow transaction. For example, the 2-of-2 multisig escrow transaction makes it possible to sign an output with Bitcoins from both sides. Unfortunately, both sides have to have their accounts on the same service, otherwise it is not applicable.

Chapter 3

Related Work

This chapter looks on related systems which have similar characteristics. The goal is to be aware that the system implemented in this thesis is not unique in the sense of the research area but unique in the combination of already existing requirements.
However, the requirements of both applications MBPS and CoinBlesk are still valid and were analysed and compared in different reports [8, 11, 12] and will not be repeated. In addition to the security aspects the requirements which have to be meet are the following, (i) federation of autonomous systems, (ii) allow to exchange Bitcoins instantly, (iii) minimize risk of fraud (trust issue), and (iv) maintain trust relations according to specified parameters. Section 3.1 analyses P2P systems managing trust. Section 3.2 introduces a model dealing with fraud in P2P networks. The next Section 3.3 introduces a federated identity solution. Paying for consumptions with Bitcoins within seconds is covered in Section 3.4. The last Section 3.5 summarized this Chapter.

3.1 Trust models in P2P network

In decentralized P2P system, trust has acquired considerable interest to extend the security mechanism. Trust to Distrust (T2D) [34], PeerTrust [35], and Dynamic-Trust [36] are models dealing with trust issue in the P2P networks.
T2D is an approach to evaluate the behaviour from trustful to distrustful (Disposition of trust). Each node evaluates the Trust Set (TS) on a scale from max trust threshold to max distrust threshold which is defined as a trust behaviour coordinate. The TS is composed of the nodes that the node with the direct trust relation can evaluate. This trusted nodes then were sorted based on their trustworthiness [34]. Concrete, each peer sets the strength of the relations to other peers with a scale from trust to distrust. Afterwards, a neutral evaluation (by an administrator) based on *Trust Sort* generates a sorted list of trust. The process is used to determine several groups classified based on the trust relation. T2D is attractive for the distributed environment where propagation is computed using summation instead of average.

PeerTrust is a reputation based trust supporting framework [35]. It includes a coherent adaptive trust model for quantifying and comparing the trustworthiness of peers. By making use of a transaction based feedback system over a structured P2P network. PeerTrust describes three basic trust parameters and two adaptive factors to compute trustworthiness of peers. The parameters and factors are namely, the feedback a peer receives from other peers, the total number of transactions a peer performs, the credibility of the feedback sources, the transaction context factor, and the community context factor. Further, general trust metric is defined by combining these parameters. Peer-Trust helps to identify malicious peers by submitting a negative feedback on the peer. Further with the credibility factor of the feedback it tries to minimize fake or misleading ratings in reputation-based feedbacks. This can happen if peers give fake feedbacks about other peers to bias the trustworthiness of peers to their benefit.
The last trust management system is the Dynamic-Trust [36]. Dynamic-Trust is based on three-dimensional computation model, namely the initial-, the direct- and the recommended-trustworthiness. Besides this model an algorithm is designed to calculated the trust. The initial value is described as the peer's ability like storage capability computation power, number of resources and bandwidth. The direct trust consists of number of interaction, time of interaction and record of interaction. The recommended value is based on weight of recommendation, recommendation of distance and the length of the recommended path. According to the trustworthiness of a peer a threshold mechanism is designed, which limits the behaviour of the peers and controls malicious peers.

T2D, PeerTrust and Dynamic-Trust meet two out of four requirements, namely (i) and (iv). It is obvious that requirement (ii) is not meet of each of them. Either, requirement (iii) is not meet. T2D, does not take any precaution against transitive relations. Even if a transitive relation has been evaluated trustful, this does not mean it can be ensured the behaviour of this peer will be the same if the Bitcoin trade amount increases. Either, PeerTrust does not give any protection based on the Bitcoin trades. This reputation systems are more concerned to identify the malicious peers. This is done by rating a already happened negative behaviour. Concluded, this means a reputation based system can not ensure trustworthiness if monetary value is in play like Bitcoins. Dynamic-Trust designed a superior system, considering new entries and the existence of many trust paths. But has not considered the reliability of the trust path, so it cannot effectively prevent vicious recommendations, either.

3.2 A Preventing Fraud Trust Model in P2P Networks

The trust management system [37] is based on the idea of congestion control of "Additive Increase/Multiplicative Decrease". Thus, a negative experience impacts the recommendation bigger, then a successful one. By providing an adaptive punishment parameter in recommended trust (direct trust), based on recommendation credibility, concussive and collusive fraud can be noticed and influence the rating stronger then in usual recommendations. Concussive fraud is defined as few peers periodically provide good services for a period of time to gain other peer's trust, and then provide bad service in a particular transaction for its own benefit. Collusive fraud means, that few peers from a collusive group bias other peers trust relation by providing good feedbacks for peers in the group and bad feedbacks for peers outside of the group. Three of four requirements are meet. This system is not operating with Bitcoins or other currency. The (iii) is half meet, without an additional feature to minimize the loses. For example, if a provider abscond the Bitcoins on her or his server this will be enough reason not to join the MBPS federation because of trust issue.

3.3 Shibboleth

According to [38], Shibboleth is open-source software that provides single sign-on (SSO) capabilities across or within and between organizations. Shibboleth is developed as open source software and is released under the Apache Software License. The software implements widely used federated identity standards, to provide the user with a federated SSO and an attribute exchange framework. The basic concept of Shibboleth is based on the same core functions as every other web-based SSO system, expected Shibboleth provides SSO support to services outside of user's organization while still protecting their privacy [39].Shibboleth works as follows, a user authenticates with his or her organizational and the organization (or identity provider) passes the identity information to the service provider to enable an authorization decision [39].
The requirement (i) is meet but the other requirements are not meet due to the functionality the software provides. Therefore, the options of Shibboleth can be analysed and used for future works. That means, is there a possibility where the Shibboleth software can provide more security and less interactions. For example, a user signed in to an instance can reach an instance which shares a trust relation to the instance of the users account informations.

3.4 Have a Snack, Pay with Bitcoins

Based on a concept to address the drawback of Bitcoin payment taking quite a time, a fast transactions is introduced [7]. The evaluation result of using double-spending attacks showed that the success of such an attack had diminished to a rate of 0,09 %. In addition, a snack vending machine was modified to accept Bitcoin payments with the fast transaction confirmation. To achieve a fast transaction some countermeasures has to be set up. First, the merchant is ensured that the costumer (attacker) is not the only source of information. Therefore, the merchant has to connect to a sufficient large number of nodes in the Bitcoin network. Further, the merchant should not accept incoming connections. Thus the costumer is forced to broadcast it over the network. Additionally, a merchant

examines the propagation depth before accepting the transaction. Another issue which was solved, is the situation were the merchant needs to return money to the costumer. This situations appear, when transaction can be underspent and the funds are returned or overspent and the customer is entitled to change or cancelled by the costumer. To prevent double-spending attacks in these scenarios a automatic invalidation of the returned transaction is issued. This is called return chaining, where the merchant claims the created output as a result of the costumers created transaction and uses it as an input to the merchants created transaction. Based on the theory a trail with a vending machine was conducted. The costumer successfully achieved a payment for a consumption in less than 10 seconds.
Only the requirement (ii) is meet in this case. Here the Bitcoin transactions are based on zero-confirmation by applying the aforementioned concept. The other requirements are not meet due to the research area, which was addressing fast payment with core Bitcoin transaction.

3.5 Related Work Summary

Table 3.1 lists the result discussed in the previous Sections. As can seen in table there is no system existing that satisfies all requirements. There were not research subject that consists of a federation mechanism performing instant payment with Bitcoins, but the core Bitcoin transactions are transferred in a later time. The mentioned trust management systems in P2P networks are concerned to identify malicious peers to minimize the trust issue. But for this thesis a malicious peer has to be prevented to have chance to perform a fraud which results in a bad feedback. The federation deals with Bitcoins malicious peers have to be identified before a fraud was committed. Hence, the presented solutions are more applicable or suggestible for system operating without monetary values.

Requirements	T2D	Peer Trust	Dynamic-Trust	Preventing Fraud	Shib-boleth	Pay with Bitcoins
Federation of Autonomous Systems	√	√	√	√	√	X
Allow to Exchange Bitcoins Instantly	X	X	X	X	X	√
Minimize Risk of Fraud	X	X	X	(√)	X	X
Maintain Trust Relation	√	√	√	√	X	X

Table 3.1: Requirements

Chapter 4

Approach and Design

After illustration of background information and related application and systems, this Chapter covers the approach decisions and design choices. Section 4.1 discusses the selected approach and decisions based on key requirements. In Section 4.2 the different levels of the trust relation are introduced. Section 4.3 provides information about the different roles. The extensions for the Android Client is explained in Section 4.4. The last Section 4.5 describes the features for the web interface.

4.1 Approach

After two successful test-runs, this thesis takes the next step by extending the existing MBPS application with a federation mechanism. Based on the goals described in Section 1.3 a list of requirements is defined and discussed. The requirements for this thesis are shown in Table 4.1. Further requirements will be introduced in Section 4.4. Those requirements are based on the requirement 7 in Table 4.1.

Create a Federation Mechanism

Bitcoin is decentralized digital currency. Unfortunately, the MBPS application runs on a centralized server, which stores all Bitcoins paid in. The current situation has a few drawbacks, e.g., the trust concern of each registered user. Considering the circumstance that the server is running in another country as the registered user, the user depends on the legislation of this country and also on the behaviour of the provider. The context that a person sending Bitcoins There is always a risk for people sending or saving the owned money (here Bitcoins) on a place outside of his covered legislation makes it risky. A provider committing a fraud a user can be confronted with additional problems related to the legislation issues. Another well known situation is, when the Bitcoins amount increases on a centralized server, the attention to this server increases simultaneously and this attention can turn into a negative experience like the examples mentioned in Section

Table 4.1: Requirements

Number	**Title**	**Description**
1	Create a Federation Mechanism	A decentralized network of servers having instant payments with Bitcoins.
2	Select Server	The user has the decision to store her or his user information to a trusted server.
3	Trust Relation	The servers can share a relation based on a defined trust level. Each level has different purpose.
4	NFC Payment	User of different servers are allowed to do cross-server payment using NFC.
5	Server Account	A server maintains a server account of all servers sharing a trust relation. Cross-server payments are depended on this account.
6	Extend Android Client	New features and improvements are provided to meet the new situations on the client.
7	Create Web Interface	A web interface is provided for administration purposes and only users with the specific right is allowed to access.
8	User Roles	Users are distinguished between each other with the role s(he) owns. The roles have different purposes.

2.4. Those mentioned circumstances can be enough to feel uncertain against centralized system dealing especially with Bitcoins.
The introduced federation mechanism in Section 1.1 is the way out to a decentralized network. This allows to minimize the concerns related to centralized systems. Additionally, the features of the previous versions like instant payment and the approach of a clearing center are retained for the federation mechanism. In the federation mechanism an instant cross-server payment can be completed by two participants of different servers initialising a NFC connection. The feature of an instant transaction payment with Bitcoins is achieved due to the approach of a clearing center. In a cross-server payment no Bitcoins are involved or transferred to new owner, only the account balance is match to the new situation. The clearing center takes the responsibility to do once or twice a day (depends on the provider) a core Bitcoin transaction. In the best case, this requirement enables participants to register the user account only on one server and do cross-server payments. For risk-averse providers there is a also the alternative to build a small group of servers with high trustworthiness to each other. Even if this mentioned scenario is not preferable in the idea of a decentralized network it is still a much better solution than the previous version running on a centralized server. The small group example distributes the registered users and still let the user to interact with each other without being from the same server. And also users can decide with which server they do not want to have a interaction by rejecting payment request from user of this server.
Figure 4.1 illustrates a federation mechanism with servers maintaining trust relations. The figure shows that server A has relations with server B, C and D and the server F for example does not share a trust relation to any other server. This shows that the number of relations can vary from server to server. The relations in the figure are represented by arrows. In Figure 4.1 the four icons linked to the server icon represents 4 function of the MBPS application. The mobile payment solution is represented by a smart phone icon, the web interface is represented by the earth icon, a database storing the accounts

is represented by a cylinder icon and the real-time information exchange (e.g. exchange rate) is represented by the person icon.

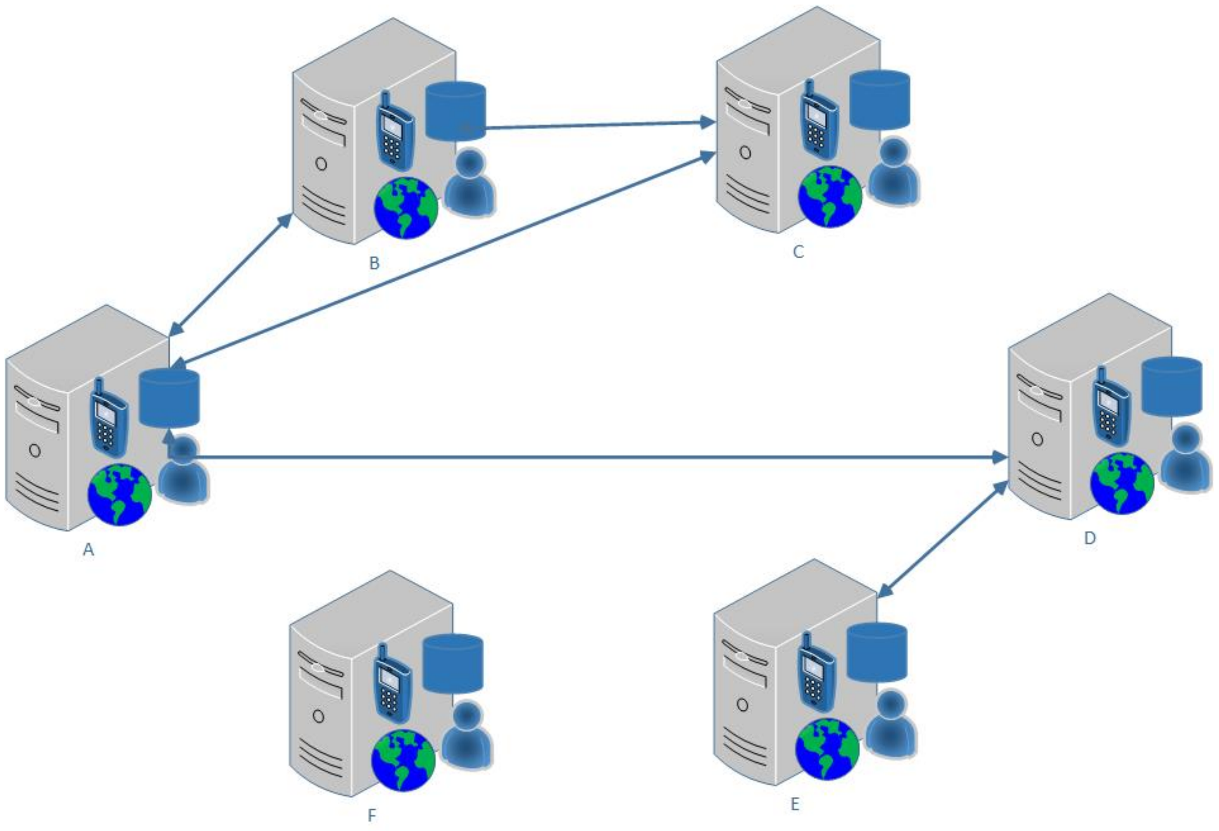


Figure 4.1: Federation of Servers

Select Server

A decentralized network of servers enables the feature for a user to decide which server can store the user account belonging to him. The possibility to select a server is a further instrument to minimize the uncertainty of a user by providing a choice to select. In centralized server approach the user is left only with two choices to participate or do not participate.
from the user perspective, a further advantage to select a server is related to the Bitcoin network. It has to be emphasized that every bad publicity concerning Bitcoins induces a negative or a cautious behaviour against Bitcoins and also against services or products with Bitcoins. Therefore, the possibility to select a server has an advantages that can be crucial for the reputation of MBPS. For example, a potential fraud or violation or even a non-accessibility of one server in a decentralized network does not effect the network as it would be in a centralized network. In a federation mechanism a user decides not only because of the instant payments with Bitcoins, the user also decides based on the reputation of the provider by selecting a server. In a centralized system, the user is not given a chance to decide for a server, instead the user is allowed to participate or

not. In decentralized system users can avoid untrusted servers or switch to an other server.

Trust Relation

With the federation mechanism additional concerns emerge. Especially, the approach of a clearing center involves risk factors. A clearing center reduces the number of transactions, e.i., several users of a server X and Y can do an undefined number of cross-server payments without a real core Bitcoin transactions is realized. The Bitcoin transaction is performed at a later time based on a defined time or a defined balance limit. The centralized server approach for the server in a federation is a requirement to allow instant payments. But before the defined time is reached or the balance limit is exceeded there is a timespan where one server holds Bitcoins that does not belong to the server any more. This reflects an uncertainty and especially because of the monetary value of Bitcoins the uncertainty converts to a trust issue between counterparties. Therefore a solution has been developed to eliminate this concern. Since the MBPS application including the federation mechanism is unique in the form, a trust relation system classifying each server by a trust level is designed as the most accurate solution. The trust levels are *No-Trust*, *Hybrid-Trust* and *Full-Trust*. These levels are discussed in Section 4.2 in more detail. It is also important to mention that even the trust game theory in Section 2.3 came to the conclusion a precaution has to be provided to overcome the trust issue, which is the trust relation system in the federation mechanism.

NFC Payment

By initiating a NFC connection between two Android devices a payment transaction is started. This is a main functionality of the MBPS application. For a payment with NFC only the Initiator has to be online. The data of the Responder is passed to the Initiator over NFC and the Initiator sends both data to the server. The data of the Responder is signed by a private/public key combination. This is a necessary security condition that prevents the initiator to manipulate the information to his benefit. Only the server is capable of verifying the authenticity of both users involved in the payment transaction.
In a cross-server payment between users from different servers both servers are involved in the payment transaction. Therefore the existing verification is extended with additional checks considering the trust relation. To satisfy the security conditions the server calls (the request and the response) are signed by the specific server with a private/public key combination. The signed payment is then verified by the other server.

Server Account

The server account is the core entity to allow and to create trust relations. The server account consists of several attributes. The attribute *Trust Level* introduces the level a

server shares with another server. Besides this attribute there exists further attributes like the *Active Balance*, *Balance Limit* and *User Balance Limit.* These attributes are all concerned with balance on Bitcoins.
The *Active Balance* determines if a server owns Bitcoins to another server or vice versa. A negative balance means the server expects Bitcoins to be paid in in the amount of the negative balance. A positive balance therefore the opposite. The *Balance Limit* is a fixed limit that cannot be exceeded. A cross-server payment is enabled until the defined *Balance Limit* is reached further payment request is rejected until a Bitcoin core transaction is done and neither of both servers expect any payins or payouts. The attribute *User Balance Limit* allows each user to do a cross-server payment up to the defined balance limit per day. This attributes gives a server additional flexibility to determine the most accurate trust relation combined with the trust level.

Extend Android Client

A user uses the Android Client to select a trusted server and create a user account. Further, the user does a cross-server payment by establishing a NFC connection. The previous version of the Android Client does not support these features and the Android Client has been improved and extended to satisfy the new requirements. The user can enter the url of the server with the most trust. This allows the Android Client to interact with the selected server. As long the user does not change the url, the Android Client interacts with existing server (the last entered url).
Before attempting a cross-server NFC payment the user has to know if the server of the other user is related by a trust relation. Therefore, the Android Client has been extended with an additional view that identifies the trust relation of servers sharing a trust. Besides, the trust level the user is also informed about the allowed balance limit and user balance limit.

Create Web Interface

In a federation mechanism servers interact with each other an to administrate the interactions a platform has to be offered. Since the start of the project the Android Client serves as a mobile payment system to transfer Bitcoins instantly. Therefore, an alternative solution has been collaborated and implemented beside the Android Client.
To make it to work a web interface serves as an administration tool to manage trust relations in the federation. The web interface has other priorities than the Android Client and therefore the users of each client have other roles. Further details on the design and functionalities of the web interface is provided in the Section 4.5.

Roles

In the current solution, there is no distinction made between the users of MBPS system. Every user can sign in by going the process of registration and verification. The introduction of the web interface makes the discussion of the user roles indispensable. The fact of the different priorities between these clients, the users have other responsibilities. Therefore, the most applicable solution includes the extension of the user into user roles. Each role grants different rights and responsibilities. The roles are *User*, *Admin* and *Both*. Section 4.3 provides the necessary information about the role system.

4.2 Trust Level Features

This Section introduces the different levels of trust. For each level the possibilities and constraints are discussed. The analysed and developed levels are *No-Trust*, *Hybrid-Trust* and *Full-Trust*. But before introducing each trust level in Subsection 4.2.1 the scenario of not having a trust relation is explained. In Subsection 4.2.2 the lowest level of trust is discussed. The *Hybrid-Trust* is introduced in Subsection 4.2.3 and includes the theory of escrow. Subsection 4.2.4 provides the *Full-Trust* the highest trust level. In the last Subscection 4.2.5 problems and constrains are explained.

4.2.1 Without Trust

Before going into the trust level, there is a case of not having a trust relation. This case is expected to be seen most of the time, especially in the earlier stage of then federation mechanism. It is logical behaviour that a provider running the MBPS will not form a trust relation with a server which is not known or not trusted. What does that mean for the users? Users have no choice but to created a further user account with any server that does not share a trust relation. But a user can still try to attempt a cross-server payment transaction without trust relation. The payment will be refused immediately. When the payment request is directed to the server, the first check is to find out which users are involved and if a user is unknown a check is done to identify if the server of the unknown user shares a trust relation. If this is not the case, the process is abort and refused.

4.2.2 No-Trust

The first level of a trust relation is the *No-Trust* level. The name introduces, there exist a trust level of the basis not trusting each other. No interaction exists in this level and compared to Subsection 4.2.1 few similarities pop up. The similarities are particularly related to NFC payment. Here, the transaction is abort and refused, either. Concluded,

the purpose of this level lies more in the security aspect than in the functionality. This means, the trust relation is the first step to be connected with other servers. After this level has been analysed the results showed that an interaction in this level is too costly. In this level the servers are not trusting each other to allow payment transactions with a core Bitcoin transaction at a later time.
A discussed solution during the design phase was that servers pay in an amount of Bitcoins into the other trusted server and vice versa. This balance serves as transaction payment for its users doing cross-server payment with *No-Trust*. This modelled scenario consisted of several challenges that has to be solved.
Before having a trust relation, there are steps involved to create a trust relation. To goal for the action to create a trust relation has to be easy. Only one server is needed to created a trust relation by knowing the url of the other server. The trust relation is created when one server enters the URL of the other server. Hence, on both servers a server account of the counterpart is created with the default level *No-Trust*. This way the creation of a trust relation is simple and transparent and does not needed the interaction of both servers. Besides creating a trust relation, the deletion of a trust relation can be done only with level.

4.2.3 Hyprid-Trust

The next level of the trust relation is the *Hyprid-Trust*. This level is the first level that actually allows to do cross-server payments. The process to achieve a successful *Hyprid-Trust* both servers have to agree on the trust level. The Initiator sends a request for this trust level and the Responder has the possibility to accept or decline the request. If two server share a *Hyprid-Trust*, the trust is backed by securities. This means both server have to give up the same amount of Bitcoins in front to allow cross-server payment transactions. The Bitcoins are paid into an escrow account and neither of both can charge back the Bitcoins without the agreement of the other server. The escrow account serves as a kind of punishment system if a violation is committed. In this case the both server will lose will lose the Bitcoins that were deposit. Mentioned in Section 2.4 there can be different kinds of escrow accounts created. The most preferable escrow account for the federation mechanism is the one without any third party involved. In the federation a server probably maintains several *Hyprid-Trust* relations and to find a third party that pleases both servers is to time consuming.
The escrow using 2-of-2 multisig transaction by locking up Bitcoins from both servers is the most adequate solution. In a 2-of-2 multisig transaction, both servers have to deposit the same amount of Bitcoin and signed the transaction. Hence, to be feasible with the escrow account the amount of cross-server payment transactions have to be less than the deposit. This can be achieved by the attribute *Balance Limit* of the server account, mentioned in Section 4.1. The Bitcoin value of the *Balance Limit* needs to be half of the deposited Bitcoins. If a server commits a fraud it will lose at least the double of the Bitcoin amount, which will be gained by the fraud. Unfortunately, the server not involved in the fraud will lose besides the expected Bitcoins the deposited Bitcoins, too. If both server resign from this trust level the Bitcoins on the escrow account have to be send back to the server it belongs.

Due to few constrains and problems, this level is constructed but not developed in this thesis. The reasons is discussed in Section 4.2.5.

4.2.4 Full-Trust

The highest trust level in the trust relation is *Full-Trust*. The procedure to build a *Full-Trust* relation is the same mentioned in Section 4.2.3. The servers in this level trust each other without attentional precautions or deposits. In this level the server can combine the trust level with the attributes *Balance Limit* and *User Balance Limit* to determine the most adequate *Full-Trust* level for each server. These attributes support the *Full-Trust* relation based on the commitment for the server to have a *Full-Trust* level. The *Balance Limit* is the limit for cross-server payments and it depends on the attribute *Active Balance*. The *Active Balance* is the current payment to the trusted relation. This means a server either owns Bitcoins (positive balance) or expects Bitcoins (negative balance) from a server, This attribute can change on both direction until the absolute value does not exceed the limit of *Balance Limit*. Further, the *User Balance Limit* allows user to do cross-server payments up to a defined limit. This limit counts only payment transaction made per day.
It is important to know that the attributes of *Balance Limit* and *User Balance Limit* do not need have the same limit in a trust relation between servers. The limits are based on the trustworthiness they inspire for each other. Though, *User Balance Limit* cannot be the same or even exceed the limit of the *Balance Limit*. The maximal limit for the *User Balance Limit* is fixed by 75% of the *Balance Limit*.

4.2.5 Constrains and Problems

The trust relation had to deal with few constrains and problems that influenced the decision making. The trust relation is based on a trust level mechanism in which two servers share a relation with each other. The relation between servers can share only the same level of trust. So, a server X shares a *Full-Trust* relation with server Y and server Y has to have the same relation with the server X. Another constrain is, a transitive trust relations does not exists. For example, server X shares trust relation to server Y and Z but Y and Z does not have a trust relation with each other. Thus, a cross-server payment between server Y and server Z will be rejected. Transitive relations are not considered trustfully due to the fact that the federation mechanism deals with monetary value in the form of Bitcoins.
The trust relations can be created and updated between servers. The creation process is accomplished by one server entering the information of the other server. The approval of the other server is not needed in creation process. Therefore, every creation of a trust relation is by default *No-Trust*. In this level a server does not have obligations associated to the other server. Further, the trust level upgrade can be successfully performed if both servers agree upon it. But a change on the level can only be accomplished if the

attribute *Active Balance* is equal zero. This means, there are no core Bitcoin transaction outstanding.
To delete a trust level only one server is needed to perform a deletion. But the trust level has to be *No-Trust*, otherwise it is not possible. A deleted server is not completely deleted only the deleted flag is changed to deleted. This assures that all transactions concerning the deleted server are traceable at any time. But for the trust relation an already deleted server can be undone by a request to re-created the trust relation. This approach has been chosen due to the uniqueness of the the url and only one server account can be created with a url.
Aforementioned, in Section 4.2.3, the *Hyprid-Trust* could not be developed in this thesis. Due to time constrains the development of the escrow account was not included in this thesis. There were question-marks if the additional analyse and implementation of the 2-of-2 multisig escrow account, which is seen as the most ideal case, would go beyond the scope of the thesis with respect to given time. To included the multi-signature escrow account, not only the successful creation is meant also the combination of multisig in respect to the web interface and server account which is holding the key information about the trust level and transactions. This means, a server has to have an option to created an escrow account and deposited the multisign Bitcoins may be with an option to select which contract pleases both servers. The steps which have to be modelled would be first the first server signs the transaction and sends that to the other server and the second server signs it, too, an sends the transaction back. Not to forget, the escrow account has to be coupled to the server account, the server account is the main entity for a cross-server payment in the federation mechanism.
An escrow account used with multi signature combined with the web interface needs a deeper analyse on an algorithm or design model to make the escrow account work with the existing web interface to federate cross-server payments on federation system.

4.3 Roles

With the federation mechanism a web interface has been developed for administration of the trust relation. The web interface and the Android Client is seperated by roles. Each role has its own priorities. In Section 4.3.1 the role *User* is discussed. The next Section 4.3.2 explained the role *Admin* followed by the role *Both* in Section 4.3.3.

4.3.1 User

The role *User* allows the user to access the MBPS application. The previous solutions operated only with this role, therefore no role was needed. The steps to become a user with the role *User* is: First the application is downloaded and then the registration and verification steps are performd. As *User* the user can sign in to MBPS application and perform several tasks, e.g., a transfer of Bitcoins into or out of the system, do a payment transaction with NFC and retrieve or alter information stored in the server. Furthermore, the role *User* is only restricted to the Android Client.

4.3.2 Admin

The role *Admin* is restricted to the web interface. The registration process for this role differs completely from the Android Client. The role *Admin* cannot be signed up by entering the credentials on a registration page. However, the first *Admin* is created when the MBPS is deployed on the server. The credentials of this user are known in advanced and should be changed afterwards. As an *Admin* the user can access every web site without any restrictions and even alter data as desired. But the main task is still to administrate and manage the server and trust relation.
Another difference between the roles *Admin* and *User* is the number of participants. For the role *User* there can be none up to an undefined number of users. The number of user depends on the willingness to participate. This case is not valid for the *Admin* role. At least there is one user that has this role within. The number of users can also increase to an undefined number of users, but that depends not only on the willingness to participate it also depends on other users who are already in the role of an *Admin* and decide to invite a person to join the MBPS application as an *Admin*. The invitation is sent by email. This is a necessary precaution to avoid a large number of users that should not have authorization. Another reason is to prevent users with hidden intentions to harm the server by altering trust relations. Concluded, the invitation process serve as an security condition to control the user that can join the system as *Admin*.
Lot of systems have usually one *Admin* for administration but MBPS allows more than one. This is based on the fact that maintaining a web interface takes quit a time and not every one is available at any time and more users in *Admin* role can manage and react much more efficient to changes.

4.3.3 Both

The last role *Both* inherits both roles, the *Admin* and the *User*. In this role a user can access the Android Client and the web interface with the same credentials. There is only one way to become *Both*. First, a user has to sign up on the MBPS application to be in the role *User*. After a successful verification, the *User* has to wait to be invited by an *Admin* to join for the administration. The second step is achieved by sending an email. The email address is the key factor to become the role *Both*. A user has to provide an email address for the registration and if the same email address is used for the invitation of the role *Admin*, the user becomes the role *Both*. The change from *User* to *Both* is done automatically and does not need an approvement of the user. The email serves as a notice for a role update. The opposite way is not possible and will be refused. An *Admin* can not use the same email for sign up on the Android Client.

4.4 Android Client

The extension of the Android Client is also a part of the federation mechanism. Before deciding for a final design mock-ups were designed and discussed internally with the involved people of the MBPS group. Since the changes to the client are just a few, it does not need an extensive evaluation procedure. The goal was that the improvements has to be easy do understand and the level of intuitiveness has to be the same like for the previous versions. Based on this mock-ups shown in Appendix C a decision was made. The views in consideration for improvements are the registration and the login view. Besides the improvements an additional view has been implemented. This extension supports the user with information about trust relation.
In the first design process the insertion of the server url in the login screen was chosen as a pop up screen, but later this decision was improved and the login screen was extended with additional text area. The text area seem to be the best decision. The login screen already has several text areas and further the user does not need to figure out where to enter the url. Another argument that supported the text area is that the url is always in the sight of the user. For the registration view the design was retained and also an additional text area for the server url was implemented. But between these two views there is a functional difference. The text area of the registration can be neglected by the user and the standard server, which is the server maintained by the CSG of the informatics department from the University of Zürich, will be chosen. But on the login screen, the user has to enter at least once the server url. The screenshots for the login and registration screens can be seen in the Appendix C in Section C.2.
Additional improvements to the history view were made. This improvement was a nice to have requirement. The improvement of bold text emphasises the important information in the view. The user can distinguish between a payment transactions made within or outside of the server. The cross-server payments are distinguished to internal payments with url listed besides the username of the user. Figure C.8 in the Appendix C shows the history screen.
Since the thesis goal is based on federation mechanism, a trust relation view is indispensable. The view can be visited over the menu navigation on the left above corner. The Figure C.7 in Appendix C shows the screenshot of the view. This view was implemented based on the history view, therefore a similarity is visible. Like in the history view this view lists all servers that share a trust relation. The information that is provided too each server are the url, the trust level, the *User Balance Limit* and a combination of the *Active Balance* and *Balance Limit* named after the *Balance Limit*. The combination of this balance limits provides an value that represents the current amount that is available. For example when the *Balance Limit* is at 1.0 Bitcoin and transaction were made with the value of 0.6 Bitcoins the combination shows a value of 0.4 Bitcoins in the view. This value is still available for further cross-server payments.

4.5 Web Interface

A web interface for administration has been developed. Before implementing the web interface an analysis and design process were conducted. Due to the time constraint and the scope of the thesis, the focus was on the features and the functions of the web interface. Therefore, the design for the user interface was not considered. Thus, the design has been kept simple over the whole web interface. In Subsection 4.5.1 the requirements for the web interface are illustrated. The Subsection 4.5.2 discussed the analyse and design phase. The features of the web interface are introduced in Subsection 4.5.3.

4.5.1 Requirements

These requirements in Table 4.2 are based on the requirement 7 in Table 4.1. The requirements is be explicitly explained in this Subsection. It will be discussed in the further Subsection 4.5.3.

Table 4.2: Requirements

Number	**Title**	**Description**
1	Authorization	The Web-page should be authorized for particular users.
2	Trust Relation	The trust relation shared with other servers has to be managed and an overview should be exists.
3	History	The server transactions have to be represented in a view.
4	Multiple Users	More then one user has the rights to sign in.
5	Accept/Decline requests	A server can accept or decline requests sent by other servers.
6	Logs	All users have to be informed about changes.

Section D.1 in Appendix D provides the mock-ups for the Web-page.

4.5.2 Analyse and Design

The focus lied on the implementation and not in the user interface. Therefore, no iterative procedure was conducted, either to evaluate the design decisions. First the requirements for the web interface was analysed and evolved. Based on the requirements the user interfaces were modelled and mock-ups were created. The created user interfaces as mock-ups can be seen in Section D.1 in the Appendix D. The mock-ups consists of the followed interfaces:

- Login interface to sign into the web page.
- Home interface consists of the most adequate information about the user credentials, Bitcoin balance, the last recent server transactions and the admin list. This page serves as default page after a successful sign in.

- Trust Relation interface retrieves and lists the server accounts sharing a trust relation. Each server consists of the url, payout address, balance limit, trust level and creation time.

- Server Account interface retrieves detailed information about the selected server. Additional to the previous view trust relation, the payout rules are visible.

- History interface provides information about the successful Bitcoin core transactions. It consists of the url, the timestamp, the amount and if it was a incoming or outgoing transaction.

- Users interface retrieve all users registered on the Coin Blesk server with the username.. email address and role.

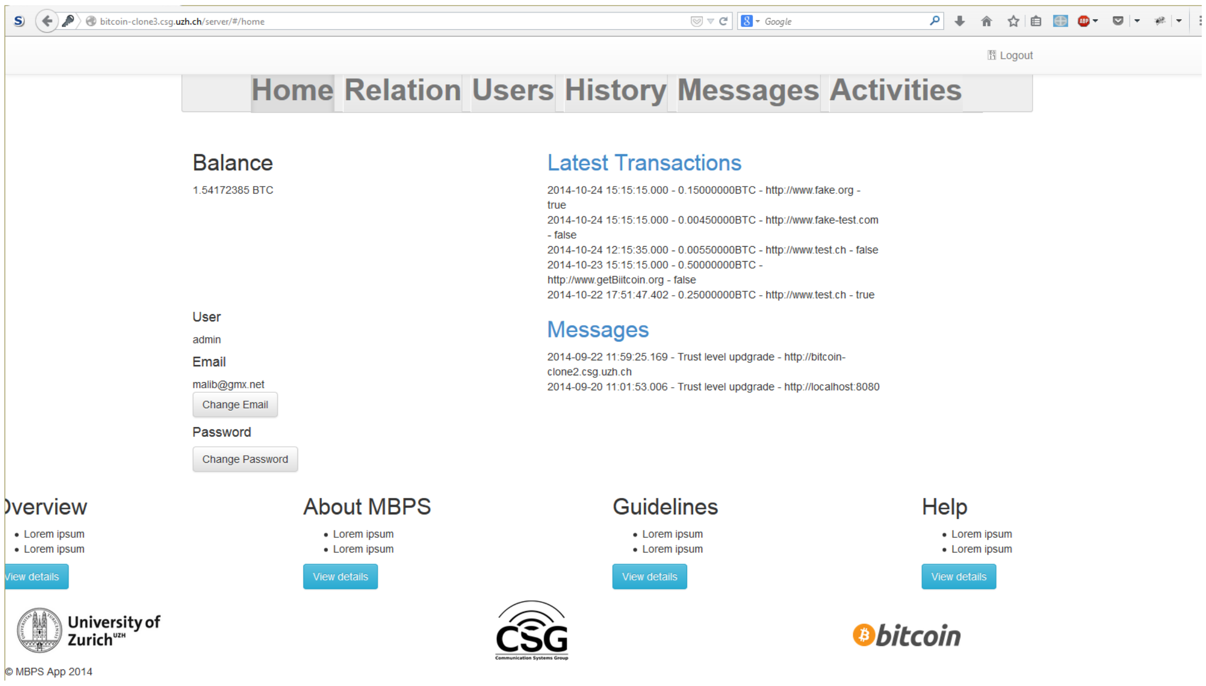
During the implementation of the web interface the requirements changed and the web interfaces had to be altered due to new requirements. Besides the already defined web pages in the design phase two additional web pages have been designed and developed. The *Activities* interface which logs every changes on the web interface. This features is supporting the admins in order to be aware of any change. The second interfaces is the *Messages* interfaces that notifies the admins about requests from trusted servers. Only not answered requested messages are list that have to be accepted or declined by an admin.

4.5.3 Features

In this Subsection the features of the web interface is discussed. The screenshots of the features are introduced in Appendix D. It has to be clear that most of the pages have the same design. Almost every page has the menu bar at the top and several links with descriptions on the the bottom. The links redirect the user to the overview, the description about the MBPS project, the guideline and the Help interface. The overview provides information about the master thesis. The about MBPS describes the project and the goals. The functionalities and features are presented in the guideline. The help supports the user with frequent asked questions (FAQ). In addition to that, the look-and-feel of the web interface has been kept very simple. Every action that involves a change is represented by a button. A click on the button opens a pop up dialogue and the dialogue is submitted if a change has to be done.

Home

After a successful login the user is redirected to this page. The home interface is the default page and its purpose is to support the user with the most adequate information. Figure 4.2 shows the user interface of the Home interface. The informations are, the sum of all Bitcoins in the system, the credentials of the logged user, the recent server transactions and the messages waiting for a response. This design differs in comparison to the mock-up introduced in Section 4.5.2. The admin list is replaced by the most recent messages. The messages have more weighted when it comes to adequate information. The credentials email and password can be update. Further, the recent server transactions can be used to ensure if any transaction have been made without redirecting to the History page. The last data informs the user about the last recent message that has to be responded.


bitcoin-clone3.csg.uzh.ch/server/#/home
Google
Logout
Home Relation Users History Messages Activities

Balance

1.54172385 BTC

Latest Transactions

2014-10-24 15:15:15.000 - 0.15000000BTC - http://www.fake.org - true
2014-10-24 15:15:15.000 - 0.00450000BTC - http://www.fake-test.com - false
2014-10-24 12:15:35.000 - 0.00550000BTC - http://www.test.ch - false
2014-10-23 15:15:15.000 - 0.50000000BTC - http://www.getBiitcoin.org - false
2014-10-22 17:51:47.402 - 0.25000000BTC - http://www.test.ch - true

User

admin

Email

malib@gmx.net

Change Email

Password

Change Password

Messages

2014-09-22 11:59:25.169 - Trust level updgrade - http://bitcoin-clone2.csg.uzh.ch
2014-09-20 11:01:53.006 - Trust level updgrade - http://localhost:8080

Figure 4.2: Home interface

Trust Relation

The trust relation is presented by two nested interfaces. The first interface includes a list of all trusted servers. Every server is represented based on the server url, the payin and the payout addresses, the active balance, the balance limit and the trust level. Those informations are illustrated for every server.
The second interface is nested and can be visited by clicking on a url of one of the listed servers on the first interface. The second page offers more information about the server account. The additional informations are the *User Balance Limit*, the existing payout rules and the latest server transactions base on this specific server. On this page an admin has the possibility to updated the trust level, to alter the *Balance Limit* and the *User Balance Limit* and created payout rules. The deletion of the trust level is also done in this interface and is distinguished by a different color to underline the consequence this action can have.

Users

This interface is simple and does not have any complex features. The interface retrieves the user accounts that are registered on the server. Each user consists of the username, email, payment address and role. The balance limit was not retrieved because the Bitcoin amount of a user belongs only to the user self and nobody should have the right to observe the Bitcoin amount of a user. Besides the retrieved information a user can invite further persons to join the system as an *Admin* by clicking on the button *Invite Admin* and entering the email address. Another button allows admins to send a email to all users at once. This feature can be used to notify all user about very urgent events. For example, a shut down of a server is a reason to inform all users in before.

History

The history interface lists all the successful server transaction. The server transactions are core Bitcoin transactions. It is enabled by predefined pay out rules combined with the attribute *Active Balance*. The server transaction consist of the Bitcoin amount, the server url, the timestamp and if the transaction was incoming or outgoing. This interface is modelled for the purpose to provide the user with traceable information about transactions which have been achieved with success.

Messages

This interfaces lists all requests of trusted servers. Only the request which are not answered are shown. The response can be performed by accepting or declining the request. The message consists of the subject, e.i., the reason of the request, a detailed description of the request and the server which sent the request, the tmestamp when the request was created and the buttons to accept or decline the request. In this stage of the thesis, only the upgraded of a trust level is a request that can be accepted or declined. There does not exists a time frame to answer the request. When a request is accepted or declined the other server is automatically informed about the decision which is logged and presented in the *Activities* view. However, an accept also includes an alteration on the data on both servers.

Activities

This interface supports all admins to be up-to-date related to the changes made. The activities interface shows all the interaction done (manipulation of data) by every admin or the received responses by the other servers. The interface consists of logs sorted after the timestamp with the most recent at the top. The subject of the activity is given, a detailed description, the server url of the server that had been changed, the admin that did the changed and the timestamp of the change.

Chapter 5

Implementation

This Chapter describes the implementation part of the thesis. To give a better understanding how MBPS works, Section 5.1 gives an overview over the architecture. The next Section 5.2 discussed essential scenarios of the thesis based on the technical part. The last Section 5.3 introduces the challenges regarding the implementation of the new features.

5.1 Architecture

The architecture of the MBPS has proven to be reliable and therefore no essential changes were done. The MBPS architecture shown in Figure 5.1 consist of several parts. The previous tools and features will be discussed first and then the new features and changes will be explained.
The middle part of the application a Java web server is running on a Apache Tomcat 7.0. This is hosted on a virtual Unbuntu machine. The server is reachable over a secure HTTPS connection. The Java web server is developed with the spring framework version 3.2.10 [40]. Authorization and access is managed by the spring-security version 3.2.5 [41]. The spring-security is also responsible for the role validation. Every request is checked if the user that sent the request has the right role, if not a 403 HTTP-Status is send back [42]. To persist the information the PostgreSQL database instance is used [43]. The Object-relational mapping (ORM) Hibernate is used to connect the implementation part with the database [44]. The left above part of Figure 5.1 presents the Android Client. This part has not changed and the application runs on Android 4.4. The client is the interface for the user to connect to the server by means of a mobile device. Still the user is capable to manage his user account, initiate transactions (pay with or receive Bitcoins), pay out Bitcoins to another wallet, etc. The left below part the Web Browser has to separated functions. The Support part serves as support to communicated with user using the mobile devices over the email address. The views are build in JSP format [45]. The other part will be explained later. The connection between the server and the Bitcoin network is established with Bitcoin client bitcoind (Bitcoin core) [46]. This is used to initiate transactions from the server to the outside world or listen for incoming

transactions into the server. The library Azazar Bitcoin-JSON-RPC is used to connect the server with the bitcoind by using RPC commands [47]. The right above part of the Figure 5.1 shows two external server connected to the server. The upper server Bitstamp is used to retrieve the current exchange rate USD/BTC from one of the leading Bitcoin exchanges [48]. The lower server Bitcoin exchange center is used to immediately exchange Bitcoins on different Bitcoin platforms [49].

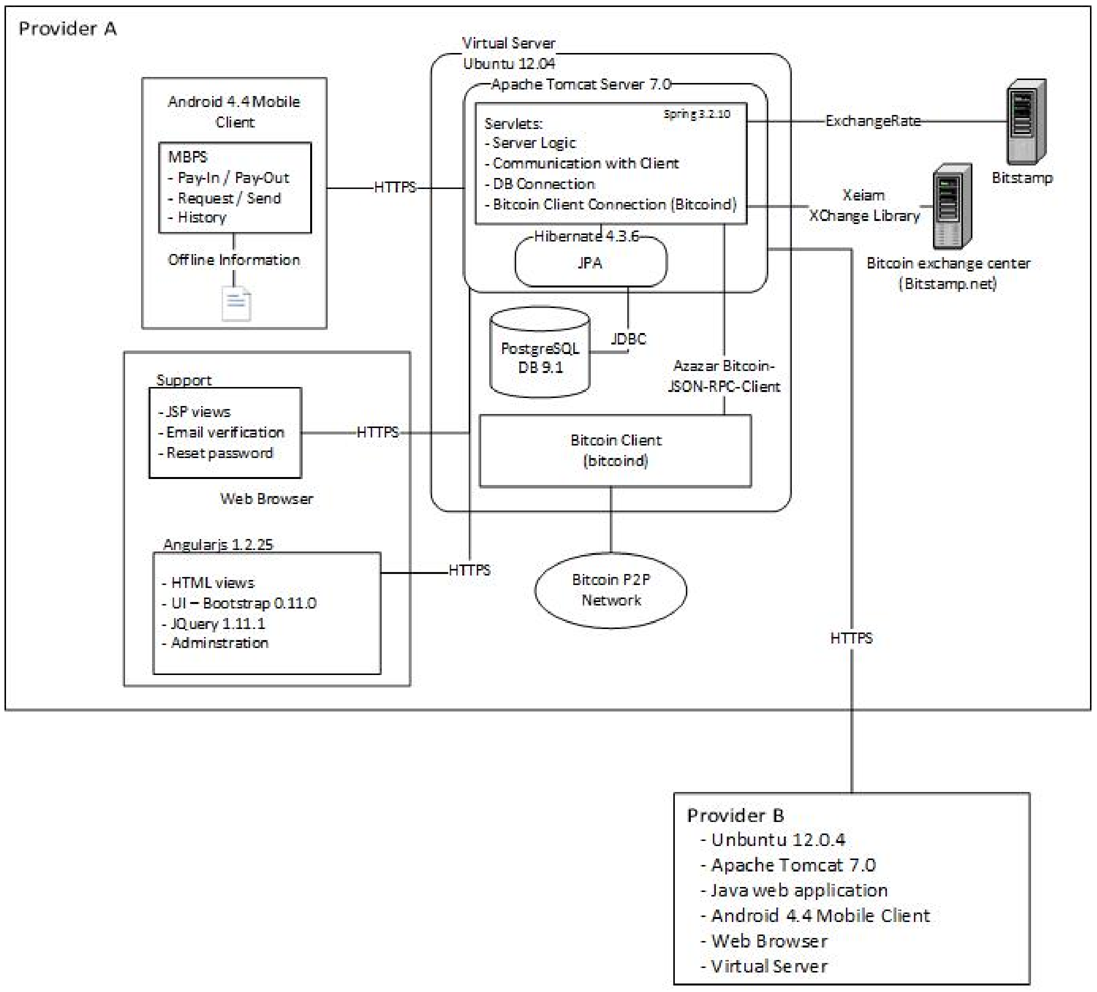


Figure 5.1: Architecture

The architecture had also changes to satisfy the new requirements or to improve the performances. The changes can be seen by comparing the architecture in report [8] with the architecture in Figure 5.1. To boost the performance the jackson message converter for converting objects [50] to JSON and back has been replaced by an own implementation of a Object mapper offered by the supervisor. The performance was tested and it showed a clear performance improvement. Especially the duration of a NFC transaction fall under one second, for more infromation see report [12]. A further change implementing the Java Persistence API (JPA) [51] between the Hibernate and the Java implementation lead to more flexibility. As long the JPA is the interface between the Java implementation and the ORM, the ORM can be changed if desired without further changes. The JPA provides a POJO persistence model for object-relational mapping.

Besides the improvements the architecture was extended by the introduction of the web interface. The Web Browser in Figure 5.1 includes the Angularjs as the core front-end tool for visualising and manipulating the view. It also maintains the communication with the back-end (server) [52]. Angularjs is a open source web application framework on the client site. It is developed by Google as a single-page application. The Angularjs consists of the script languages CSS, HTML and mainly JavaScript. In a single-page application there is one page which is rendered with the first request of the web page. The content is then load into this page and for every request that involves a new view the content is switch with the respective view. This framework is build on the model-view-controller (MVC) design pattern to separates the part of the code to their responsibility. The views have been developed with HTML. The server call from the front-end to the back-end is done by core Angular service $http [53]. The $http communicates between a remote HTTP server via the browser's XMLHttpRequest object or via JSONP. The authentication between the Android Client and the Server works based on tokens send in the header of a request to identify the user on the server. In order to make it work for the web-frontend and the server a Cross Origin Resource Sharing (CORS) [54] Filter on the server as well as the requesting $http of Angularjs must explicitly allow certain flags in the Header. To enable for cookies to pass through the Angularjs and the CORS Filter on the server must expressly have the credentials flag *withCredentials* enabled. Further, to enable authentication the flag *Authorization* has to be set, too. Further, Figure 5.1 shows two server A and B connected to each other with a secure HTTP connection.

5.2 Scenarios

In this Section two use cases have been introduced. The use cases are emerged with the implementation of a federation mechanism. In Subsection 5.2.1 the scenario of creating a trust relation is explained followed by the scenario doing a cross-server payment in Subsection 5.2.2. It is important that both described scenarios are synchronized requests expecting a response in a specified time.

5.2.1 Create Trust Relation

A trust relation can be create only involving one active server in the creation process. Figure 5.2 shows the creation cycle of a trust relation. A user of server A enters the information about the server B over the web page and sends a request to the server A. On server A the request is prepared with the server url of A, the email address of the Initiator and the server key pair of the server A. Those information are sent to the server B. B creates a server account based on the received information. The server B prepares a response consists with the same data (server url and key pair of B) and sends the response back to A. Server A creates a user account based on the received information. After the creation of the server account additional request have to be exchanged consisting of sensible data. The payout address is the sensible data. The payout address for the other

server is the created payin address on the server account. This piece of information was not sent in the previous request due to the sensitiveness of the payout address. When the payout address is lost or even stolen the existing Bitcoins on this address are gone, too. And also without a payout address no core Bitcoin transaction can be established between the servers and no cross-server payments are allowed, either. Therefore, the risk that the request could be manipulated or even replaced is a big concern.
After exchanging the initial server information to create a server account and the public keys, a second request is created by server *A*. The request consists of the payout address for *B* and the server url of *A*. The url serves as identification. This request is signed by the private key of *A* and send to *B*. *B* verifies the received request with the stored public key of *A* and the payout address is persisted into the database. Server *B* repeats the same steps and responds with a signed response object having the payout address for *A* and the url of *B*. Server *A* also verifies the response and persists the payout address.
But in the case the request fails in the first place the information is stored in the "*Server-AccountTask*". The *ServerAccountTask* stores failed requests between server s to repeat the request on a later time without a user involved. These stored information are processed each hour until the request can complete with success. Figure 5.2 illustrates the creation cycle of a trust relation.

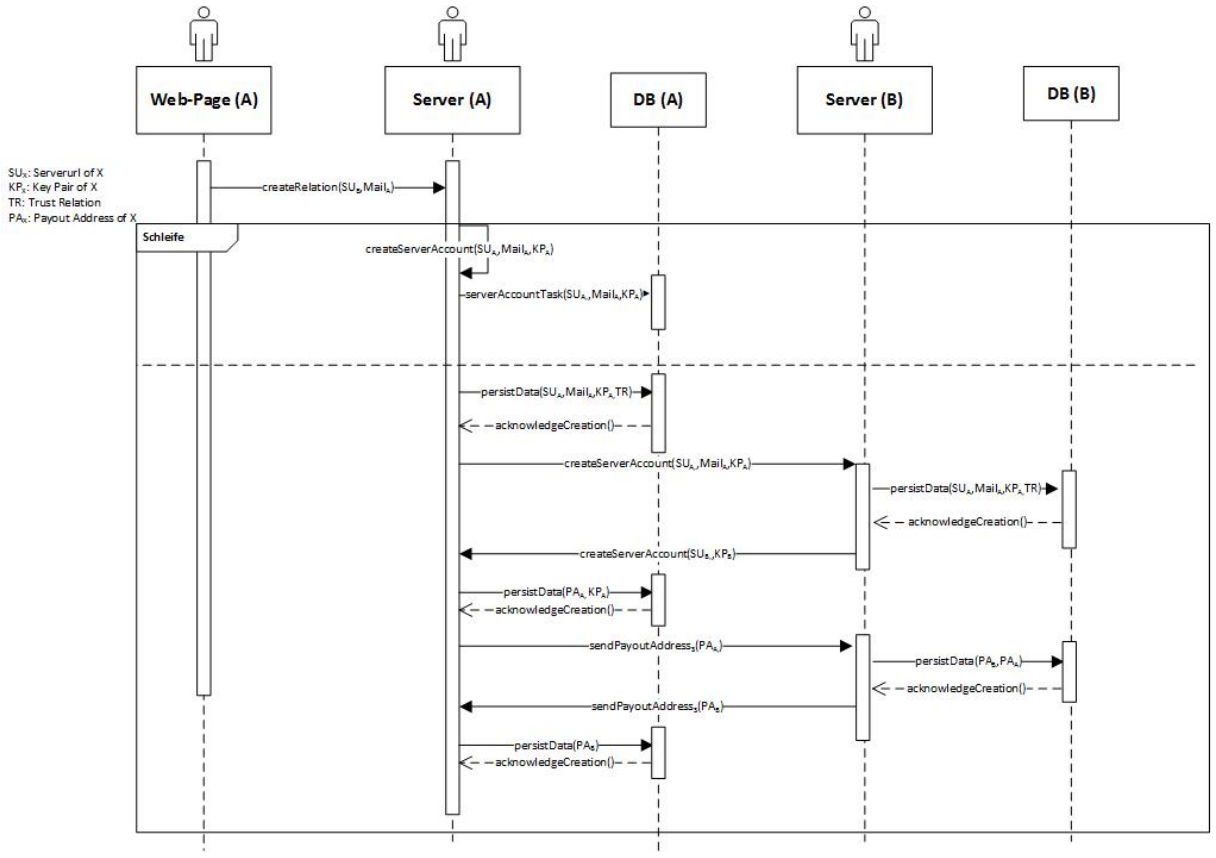

Web-Page (A)
Server (A)
DB (A)
Server (B)
DB (B)
SU_X: Serverurl of X
KP_X: Key Pair of X
TR: Trust Relation
PA_X: Payout Address of X
Schleife
createRelation(SU_B,Mail_A)
createServerAccount(SU_A,Mail_A,KP_A)
serverAccountTask(SU_A,Mail_A,KP_A)
persistData(SU_A,Mail_A,KP_A,TR)
acknowledgeCreation()
createServerAccount(SU_A,Mail_A,KP_A)
persistData(SU_A,Mail_A,KP_A,TR)
acknowledgeCreation()
createServerAccount(SU_B,KP_B)
persistData(PA_A,KP_A)
acknowledgeCreation()
sendPayoutAddress_S(PA_A)
persistData(PA_B,PA_A)
acknowledgeCreation()
sendPayoutAddress_S(PA_B)
persistData(PA_B)
acknowledgeCreation()

Figure 5.2: Create Trust Relation

5.2.2 Cross-Server NFC Payment

This is the scenario to request Bitcoins from another user. It represents the "Receive Payment" scenario from MBPS but with an extension of allowing users to do cross-server payments. In this scenario, the servers *Seller* and *Buyer* have a *Full-Trust* relation and Android Clients Are involved.

First, the devices are hold together to start a NFC connection. After exchanging messages over NFC the information are send to the server of the *Seller*. On the server the accounts of both users are loaded. But if one of the users are not found a check is done to find out if the unknown user belongs to a server that shares a trust relation with the level at least *Hyprid-Trust*. In the next step the *Balance Limit* and the *User Balance Limit* are checked and verified. Both limits can not be exceeded. A customer payment request is created after the checks were successful. The payment request consists of payment information of both involved users, the amount, the timestamp of the payment and the url of the server that send the request. Before the request is passed to the *Buyer* the request is signed by the server *Seller* to ensure this request is sent by the *Seller*. The server *Buyer* checks that the limits for the balances are not exceeded and verifies the signed request. After that, the information are persisted in the server and the a customer payment response is created and signed by the *Buyer*. The response is retuned back to server *Seller*. *Seller* verifies the signed object and checks if the received information are same in comparison to the send information. The server *Seller* then persists the information and acknowledges the user *Seller* and *Buyer* about the payment success. In this scenario of cross-server payment the send and received objects are signed twice. Once the users signed the created server payment with the key pair and once the server signed the created customer payment with the key pair.

5.3 Challenges

In this Section the challenges which were faced during the implementations are discussed. The challenges that are introduced are related to the implementation of the web-site wit respect to Angularjs, the database entries, the attribute *Active Balance* of the server account and finally about writing JUnit tests.

5.3.1 Communication between Angularjs and the Server

Aforementioned, Angularjs communicates by a core Angular service $http. The content of the request between the back-end and the front-end is exchanged as JSON objects. The structure of the Angulrjs request differs to the request from the server, but since both can handle JSON objects, the communication was done with JSON objects. A further

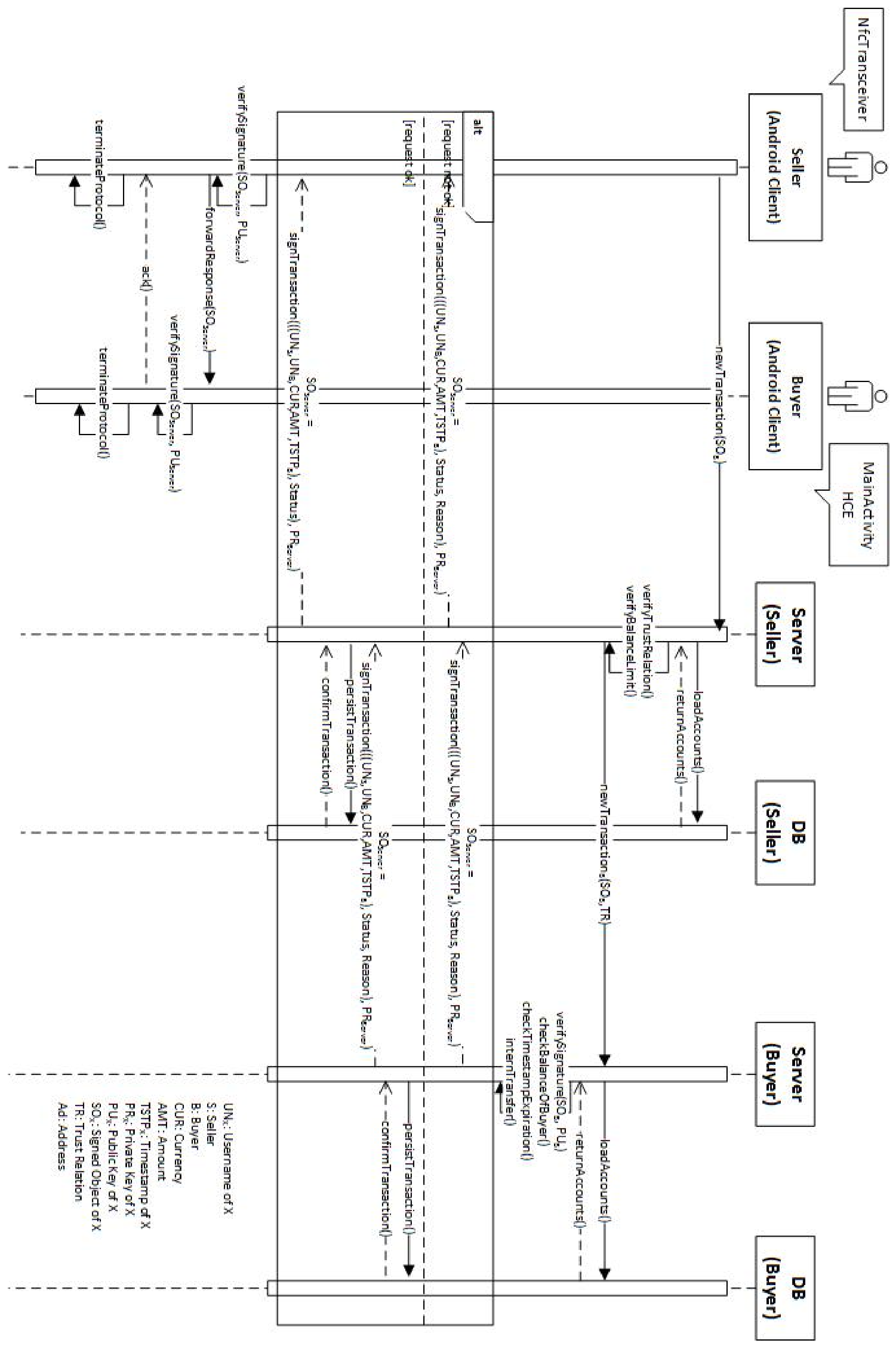


Figure 5.3: Mean and Standard Deviation

customization had to be made, before the communication could work. Explained in Section 5.1 certain flags had to be enabled permanently in the header of a request. Also, a CORS filter was implemented on the server that states to the browser that cookies are permitted. It is important that the CORS filter has to be validated before the security filter of the spring-security is processed. Otherwise the request would fail.
Another challenge was the authorization of the user. In this thesis a users has different roles and a role permits the user for certain requests. The check of the role is done by the spring-security. The reason to verify the roles on the server was decided because the spring-security was already responsible for the authorization between the server and the Android Client. Only a simple modification was enough to allow the spring-security to authorize users based on their roles.
A further challenge during the implementation was, that the spring framework has to handle different forms of views. Spring has a build-in view resolver that looks for the requested view and returns it. Unfortunately this resolver neglects other resolvers. This means it does not pass the request to other resolver instead sends back an error. Therefore a second resolver was implemented for the HTML views. This resolver had to be modified and placed in front of the internal resolver. So, if the implemented resolver did not find the respective view it pass the request to the internal resolver.

5.3.2 Extend Database Entries

The implementation of a federation mechanism also included information that had to be stored on the database. New entries has been created like the server account that stores the information about the trusted servers, server transaction that stores successful performed core Bitcoin transactions and server account tasks that stores failed request to other servers which have to be repeated. The challenge here was to decide if a new entry was needed or an alteration of the existing entry was good enough. But it was decided that the data from the Android Client and web interface are not mixed and therefore new entries were created for data coming from the web interface.
Besides the new entries existing entries has been altered, to meet the requirements of this thesis. In the previous version the username consisted only with the entered name. In the federation mechanism the username has been altered and the new username consists of the entered name and the server url of the server that holds the user accounts. These strings are separated by the symbol of "@". This symbol is forbidden to be used in the entered name an the url. For example, a username can look like this: "*muster@http://muster.com*". The name is the first part followed by the symbol that serves as separation and the last part is the absolute url of the server which the user is registered. This changes helped to identify server of the user. Especially, during a cross-server payment the username is used to identify the server of a unknown user. A further alternation is done to the transaction entry. This entry has been extended with two attributes. The attributes are the server urls of the payee and the payer of a payment transaction. This information helps to check if the *User Balance Limit* was exceeded by a user of a cross-server payment.

5.3.3 Active Balance

A successful cross-server payment has to be stored to do a core Bitcoin transaction on a later time. This means an attribute has to be implemented that understands if a payout or a payin is needed. Therefore, the attribute *Active Balance* is created. This attribute is the only balance attribute that can have a positive or a negative amount. The *Active Balance* has a default value of zero. A value zero means for the Server there are no outstanding or incoming transactions. With the balance limit zero either of both server have no obligations to own or expected any Bitcoins. Therefore, a new created trust relation has an *Active Balance* of null. And if a change is related to the server account the change can only be succeed if the *Active Balance* is zero, otherwise the change will be rejected. This is a security condition, e.g., a server wants to downgrade the trust level of *Full-Trust* to *No-Trust*, so the server could have to possibility to deleted the trust relation if the server has outstanding Bitcoins to send.
If the *Active Balance* is greater than zero, the server owns this amount to another server. An outgoing Bitcoin core transaction is expected. But if the *Active Balance* is negative the server expects this amount from another server. An incoming Bitcoin core transaction is expected.

5.3.4 Test Cases

An implementation has to have test cases to ensure that the written implementation has no bugs and the source code is functional correct. In the previous projects the test cases were implemented by manipulating the entries of the existing database. This is not productive if the database already holds sensitive data that cannot be altered. Therefore, for the test case in this thesis DbUnit was used which allows the manipulation of data without touching the database with the sensitive data. DbUnit is a JUnit extension that puts your database into a known state between tests [55]. The data that has to be used and alter are generated in a XML file. During the test run the entries in the XML files are put in the memory of the processor and can be compared or altered.

Chapter 6

Evaluation

One main objective the instant payment between users of the MBPS application has been the focus for the evaluation. Therefore, the evaluation addressed the time required to successfully complete a payment over a NFC connection. The performance of two scenarios have been evaluated and compared. The first scenario is a payment from users registered on the same server. This scenario has been already evaluated in a previous report [12]. But the evaluation was focused on the Payment Library and the NFC Library. The second scenario is a payment between users from different servers.
In Section 6.1 the setup of the evaluation is explained. The next Section 6.2 explains the performance of the payment on the same server. In Section 6.3 the performance of cross-server payment is evaluated. The Section 6.4 discusses the performance measurement on the server. The last Section 6.5 analyses the performances measurements and a conclusion is drawn.

6.1 Setup

The setup for the performance measurements have been conducted with the devices Google Nexus 7 [56] and LG G3 [57]. The NFC of Nexus 7 works with a Broadcom chip and the NFC of LG G3 with a NXP chip. In both scenarios the complete time required has been split into steps nested in each other. This steps has been needed to identify the required times for a comparison. The measurements for each evaluation part was repeated twenty times on both devices as a payer and as a payee. A second performance measurement has been conducted to evaluated the required time for the atomic steps of the server. The measurements for the payee and payer for each device has been repeated 10 times. The amount of repetition for the second evaluation was lesser due the fact that the required time for server was not affected by the specific features of both devices. Further, the total time required to completed a payment has been based on an auto-complete mode. In practice the devices used for the evaluation haven been tapped to each other without an accept or refuse possibility.
It has to be clear to distinguish the evaluation of this thesis from the evaluation in report

[12]. Another remark to mention is the Android's Host-based Card Emulation (HCE) mode of operation. The Initiator, acting as transceiver, in a NFC connection sends a message and receives a response in the same turn. Due to the fact that the Android API lacks to offer functionalities for measurements when the Initiator or the Responder is reading and writing data this thesis had to chose other beginning points for the measurement. Based on the Android API, the time for the Initiator can be measured before passing data to the NFC channel and stop measuring when the response arrives. The measurement for the Responder can be start after a message has arrived and before the response is passed to the NFC channel.
In this thesis only the time for the Initiator has been taken to account. Hence, the focus does relay on the total time required to completed a NFC payment but from the initialising point of the NFC up to the success notification. It is important to pinpoint that the steps between the scenario of users from the same server differ in opposite to the users from different servers. The evaluation goal is to show if a federation of servers still operate applicable for the MBPS project. Therefore, the new scenario is compared to the previous version and possible improvements can be quantified. The time discussed in both scenarios are based on the arithmetic average with a standard deviation measured in milliseconds. Another important aspect is, the evaluation has been conducted with HTTP requests instead of the required HTTPS. The reason is the during the implementation HTTP requests were used. For the production the HTTPS is required.

6.2 Performance Measurement NFC Payment

This Section first introduces the payee and payer role for the scenario involving one server to completed a NFC payment. Then the steps on the server is discussed. Afterwards, a comparison is drawn.

6.2.1 Payee Role

The performance measurements for the payee role are shown in Figure 6.1. The figure illustrates the arithmetic average and the standard deviation for the Nexus 7 and LG G3. The results for the Nexus 7 are shown in the Tables G.1 and G.2. The Tables G.5 and G.6 consists of the results for the LG G3. The last bar in Figure 6.1 is the total time required to complete a payment between users. The *NFC Exchange*, the *Server Call* and *NFC Notification* are a part of the total time. But the *Server Process* is nested in the *Server Call* step. The first step is proceed on the devices, the second step consists on the server and on the devices. *Server Call* is called on the device but the most time is consumed by the server which is the step *Server Process.* The *NFC Exchange* is the time needed to create both payment requests of both involved users. This step consists of multiple atomic steps which are not considerate for the evaluation. The major pieces are the NFC handshake, the time a device has been discovered until the handshake completes by a message returned by the payer. Further, the payment request is created by exchanging data with the payer. The data in the payment request is serialized to keep

the message shorter and send to the server. The *Server Call* describes the required time to send a payment request and receive a payment response back from the server. The *Server Process* consists of different verification and manipulation steps, see Section 6.4.1. After the server validated the data a signed payment response is sent back. On the client side, the payment response is verified by the payee and serialized in order to forward it to the payer. At the end the success notification is shown on both devices. This is the step *NFC Notification*.
Compared to the total time required, the most time consumed the step *NFC Exchange* with an average of 372ms (± 9.9ms) on the Nexus 7. The performance measurement for this step on the LG G3 took in average 424ms (± 33.9ms). This time is considered as justifiable, since it consists of the message exchange between the devices. First, the prepared message is send to the payer, the payer then signed the request and returns back to the payee. The payee serialized the request and forward the request to the server. This steps make around 46% percentage of the total time to complete. Since both devices have been proceeding the same atomic steps and have been evaluated with each other, the recognisable difference in *NFC Exchange* has to be device specific.
The next step *Server Call* took in average 312ms (± 23.3ms) on the device Nexus 7 and in average 369ms (± 67.2ms) on the device LG G3. This time cannot be compared efficiently, due to the reason that the measured time consists of HTTP requests to the server. The last step *Server Process* is the only nested step in the *Server Call* step without the HTTP request. This time describes the start and the end of different checks and manipulations. On the Nexus 7 this step took in average 176ms (± 22.6ms). The performance measurement for the LG G3 took in average 159ms (± 15.6ms). The difference of around 15ms between the devices is not a big factor and compared to the standard deviation of both devices it can be seen as same. But considering the step *Server Process* to the step *Server Call* there is a clear difference between the time the data is sent and returned to the client. On the Nexus 7 the step *Server Process* makes up to 56.6 % percentage of the measured time for *Server Call* but only 43.1 % percentage on the LG G3. Since both devices were evaluated on the same WLAN at the same time, there can be found an explanation without further evaluation based on the difference. The step *NFC Notification* took in average on the Nexus 7 121ms (± 11.3ms) and on the LG G3 178ms (± 35.4ms). The LG G3 took longer than Nexus 7 and this is based on the same reason like for the *NFC Exchange*, device specific.
Concluded, the performance measurement for the total time on the Nexus 7 took in average 807ms (± 46.7ms) and for the LG G3 it took 976ms (± 134.4ms). The difference is due to the specific features on both devices. The LG G3 took in average around 150ms longer to complete the NFC payment. The evaluation shows that both devices completed a NFC payment transaction under one second.

6.2.2 Payer Role

The Figure 6.1 shows the performance measurements for the payer role. The steps in this role are exactly the same mentioned in the Subsection 6.2.1. The last bar in the figure represents the measured time for the total time required to complete a payment. The results for the arithmetic average and the standard deviation in the Figure 6.1 for the

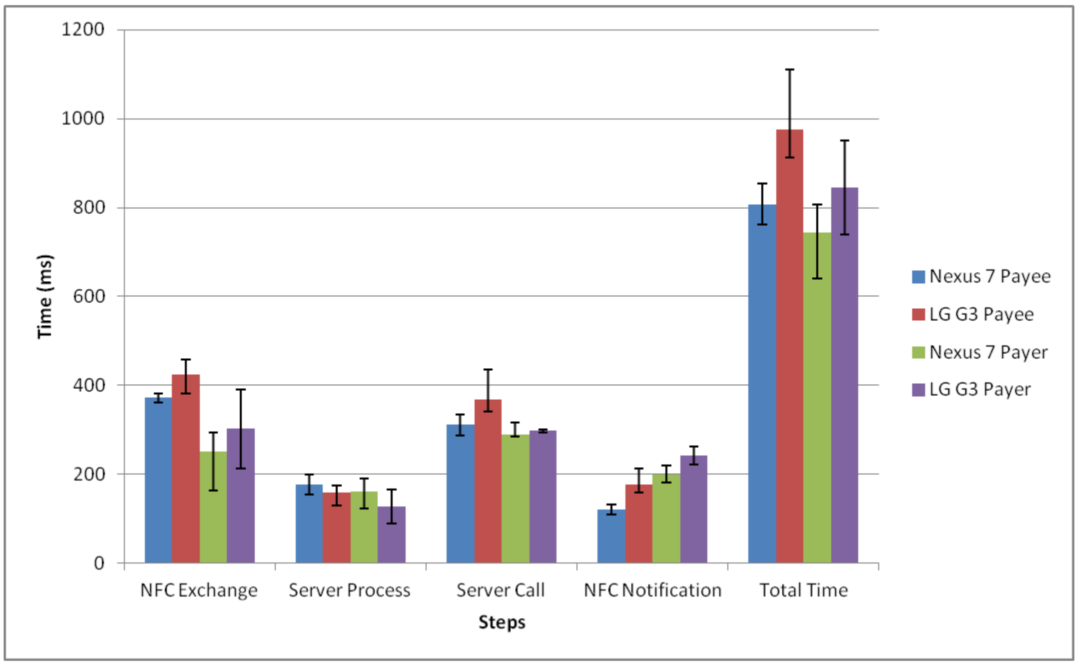


Figure 6.1: Mean and Standard Deviation NFC Payment

devices Nexus 7 and LG G3 are illustrated in the Tables G.3, G.4, G.7 and G.8. The *NFC Exchange*,, *Server Call* and *NFC Notification* are a part of the total time. The *Server Process* is nested in the *Server Call* step. The *NFC Exchange* is the time needed to create both payment requests. This step consists of multiple atomic steps. The major pieces are the NFC handshake, from the time a device has been discovered until the handshake completes by the second handshake message returned by the payee. Further, the payer requests the payee's username. After the username of the payee is deserialized the payer creates a payment request. This request is signed and send to the server. On the server, the request is verified, checked and manipulated. Finally, a signed payment response signed by the server is returned back. The client of the payer verifies the signature and serializes the payment response to forward the response to the payee. At the end the success notification is shown on both devices. This is the step *NFC Notification*.
For the Nexus 7 the most time consuming step is the server call. The server call made up 38.9% of the total time required followed by the step *NFC Exchange* with 33.8%. The arithmetic average for the step *NFC Exchange* took 252ms (± 41.2ms). The arithmetic average of the device LG G3 for the step *NFC Exchange* resulted in 302ms (± 89.1ms). It is important to mention that the atomic steps for the *NFC Exchange* differ for the payee and payer role. One important difference is, in the payee role there is the step where the Responder (payer) has the possibility to accept or reject the payment and if the request is accepted a signed payment object is created. But in the payer role only the username of the Responder (payee) is needed to create a payment request. However, for the LG G3 the steps *NFC Exchange* and *Server Call* required the same time to complete. 35.7% of the measured total time was covered by the step *NFC Exchange* and the *Server Call* covered 35.2%. The arithmetic average for the step *Server Call* took 290ms (± 26.4ms) for the Nexus 7. The LG G3 took in an average 298ms (± 3.5ms). In consideration that this steps consists of HTTP requests and responses to the server, it is difficult to compare

and explain the time performance.
The step *Server Process* is the only nested step in the *Server Call* step without the HTTP request. The arithmetic average for the Nexus 7 took 161ms (± 28.8ms). The performance measurement for the arithmetic average for the LG G3 took 128ms (± 37.5ms). In percentage the step *Server Process* on the Nexus 7 made up 55.6% of the measured time of the step *Server Call*. The same step for the LG G3 made up 42.9% of the complete time required for the *Server Call*. The last step the *NFC Notification* took in average 201ms (± 19.5ms) on the Nexus 7 and 243ms (± 19.8ms) on the LG G3.
Concluded, the performance measurement for the total time on the Nexus 7 took in average 744ms (± 63.4ms) and for the LG G3 it took 845ms (± 104.7ms). The difference is due to the specific features on both devices. Considering the calls on the server on both devices required the same time to complete it has to be related to the devices.

6.2.3 Comparison Payee and Payer

Figure 6.1 compares the performance measurements for the role payee and payer. The required time to complete the payment for the payer took longer than the time to completed a payment for the payer. The step *NFC Exchange* does the difference with the payee role the arithmetic average took 372ms (± 9.9ms) and 424ms (± 33.9ms) and for the payer role the average took 252ms (± 41.2ms) and 302ms (± 89.1ms). In the payee role the Responder needs to created a signed payment request which is send to the server. But for the payer role the Initiator only needs the username to completed the task. The messages exchanged between the Initiator and Responder for the payer is shorter than the one exchanged between the payee as Initiator. The steps *Server Call* took in average the same time but this results can probably vary in a further test based on the HTTP request. The measured step *Server Process* to completed a payment took in average for both devices and for both roles around the same time. In Figure 6.1 the total time to completed a payment was around 800ms. The payee role took in average a bit longer than the payer but this is mostly related to the step *NFC Exchange*. Further, the step *NFC Notification* in the payee role consumed in average less time than as payer. This has to do with devices specific features, that have to be evaluated focused on the different devices.

6.3 Performance Measurement Cross-Server Payment

This Section introduces the payee and payer role for the scenario involving two servers. This scenario is based on the federation mechanism (see Section 4.1) in which a cross-server payment can be succeeded between two servers sharing a trust relation. Besides the already existing steps in the previous versions additional steps have been implemented to enable a cross-server payment transaction. Below the payee and payer role is discussed followed with the comparison part.

6.3.1 Payee Role

The performance measurement for the cross-server payment shown in Figure 6.2 consists of multiple steps. The figure shows the arithmetic average and the standard deviation for each step and the total time. The last bar indicates the total time required to complete a payment. Besides the measured total time. the further steps and the nested steps in the figure are based on the results in the Tables G.9, G.16 for the Nexus 7 and in the Tables G.13 and G.14 for the LG G3. The steps *NFC Exchange*, *Server Call*, *Server Process* and *NFC Notification* are the same steps evaluated in the Subsection 6.2.1 in the payee role. The additional new steps are all nested into each other and consist in the step *Server Process*. The nested step the *Trust Relation Process* is the beginning point for a cross-server payment. This step is proceeded if the Responder (payer) is registered on another server sharing a trust relation. The *Trust Relation Process* step itself consists of atomic steps that are responsible for checks, server calls related to the trusted server and verification of the returned payment response. The *Trust Relation Process* consists of the steps *Trusted Server Call* and this again consist of the step *Trusted Server Call*. Further steps which are not evaluated in this scenario can be seen in Section 6.4. The *Trusted Server Call* is defined as the time needed to send the payment request to the trusted server and receive a response is return. The last step *NFC Notification* verifies and serialized the response in order to forward it to the Responder. At the end the success notification is shown on both devices. The step *Trusted Server Process* is proceeded by the other server. Figure 6.2 shows the nested step of another step always on the left side of the relevant bar. Expect the nested step is proceed on a different platform.
In a cross-server payment the arithmetic average for the step *NFC Exchange* took 339ms (± 70.7ms) on the Nexus 7 and 505ms (± 73.5ms) on the LG G3. There is a visible difference between the arithmetic average of the two devices which can not be explained with the measured results and further measurements focused on the devices specific difference have to be evaluated. However, with an average of 28.4% for the Nexus 7 and 34.5% for the LG G3 this step was not the most time consuming step. The most time consuming step was the *Server Call* that took half of the time required for a payment to complete (58.6% Nexus 7 and 52.00% LG G3). The arithmetic average for the Nexus 7 took 699ms (± 106.1). The LG G3 had an arithmetic average of 759ms (± 82.0ms). The results of the measured time can be explained with the argument that in a cross-server payment two servers are involved during a payment.
The nested step of the *Server Call* the *Server Process* is the time required to completed all the atomic steps consist in the server. The Nexus 7 had an arithmetic average of 478ms (± 166.2ms) and the LG G3 an average of 469ms (± 72.1ms). The most considerate step in the *Server Process* is the *Trust Relation Process*. This step is only proceeded if a cross-server payment is started. For both devices the arithmetic step for the *Trust Relation Process* took 304ms (± 87.7ms) respectively 350ms (± 106.8ms). The required time for this step covers more than the half of the *Server Process*. This can be explained due to the check and verification part based to verify the trusted server. Further, the server call to the trusted server has had also an role , why the *Trust Relation Process* took in average over 300ms on both devices. The server call to the trusted server had an arithmetic average of 247ms (*pm* 64.3ms) and 304ms (± 116.0ms). This nested step covers for the Nexus 7 and LG G3 more than 80% of the *Trust Relation Process* step. Skipping the HTTP request the trusted server call took in average 205.5ms (± 57.3ms)

for the Nexus 7 and 269ms (± 133.0ms) for the LG G3. Considering the checks and data manipulations in this step, this step has been fast. The atomic steps in this step are evaluated in Section 6.4. The last step *NFC Notification* which is proceeded on the device took in average 153ms (± 12.7ms) on the Nexus 7 and 189ms (± 12.0ms) on the LG G3. Compared to the total time required this step make up 12.8% and 12.9%.
Concluded, the performance measurement for the total time on the Nexus 7 took in average 1193ms (± 164.0ms) and for the LG G3 it took 1460ms (± 169.7ms). The huge time difference cannot be explained without further evaluation. The performance measurement for the total time showed that both devices needed between 1 and 1,5 seconds to completed a cross-server payment.

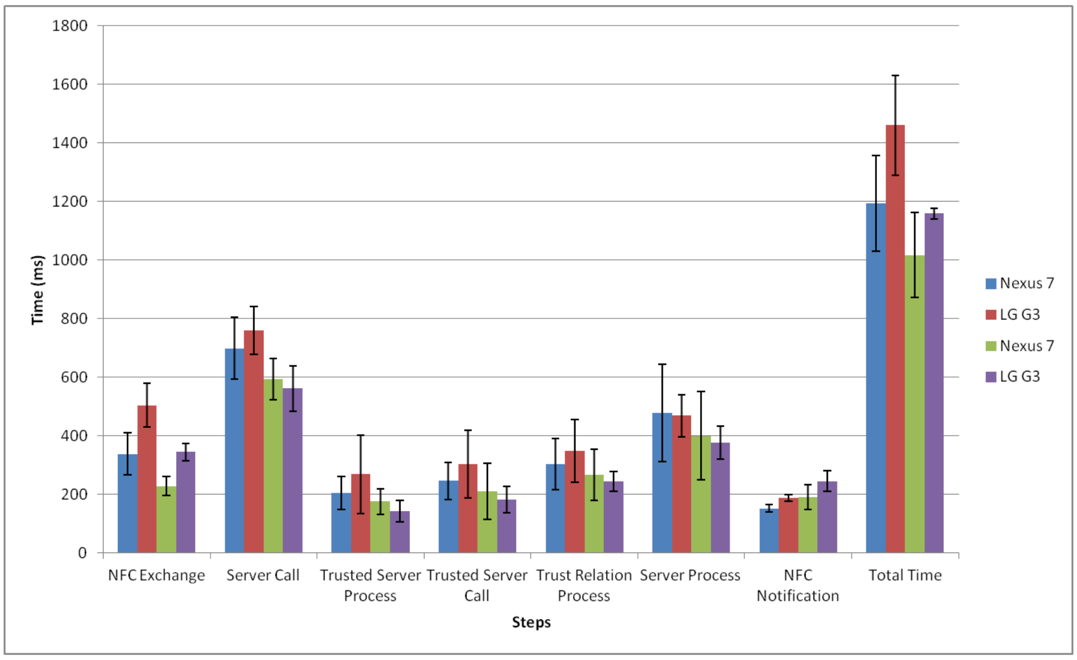


Figure 6.2: Mean and Standard Deviation Cross-Server NFC Payment

6.3.2 Payer Role

The Figure 6.2 shows the performance measurements for the payer role. The steps in this role are the same mentioned in the Subsection 6.3.1. The last bar in the figure represents the measured total time required to complete a payment. The results for the arithmetic average and the standard deviation for the devices Nexus 7 and LG G3 are illustrated in the Tables G.11, G.12, G.15 and G.16. The *NFC Exchange* and the *Server Call* are steps nested in the total time. The *Server Process* is a nested step of the *Server Call*. The *NFC Exchange* is the time to initialize a NFC connection and exchange messages through the NFC until the prepared payment request is passed to the server. Only the *NFC exchange* and *Server Call* can be seen as atomic steps of the measured total time. The steps *Server Process*, *Trust Relation Process*, *Trusted Server Call* and *Trusted Server Process* are nested into each other. The last step *NFC Notification* verifies and serialized the response in order to forward it to the Responder. At the end the success notification

is shown on both devices. Figure 6.2 shows the nested step of another step always on the left side of the step that consists it. Expect the nested step is proceed on a different platform.
The *NFC Exchange* step for the Nexus 7 and LG G3 took in average 229ms (± 31.8) and 345ms (± 29.7ms). The LG G3 again needed in average more time to complete this step. The next step *Server Call* consumed in average half of the complete total time. The Nexus 7 took in average 594ms (± 70.7ms) and the LG G3 had an arithmetic average of 563ms (± 77.1ms). Compared to the total time the LG G3 makes up 48.5% and the Nexus 7 58.4% of the measured average of the total time.
The nested step *Server Process* took in average 400ms (± 150.6ms) and 378ms (± 18.4ms) for the Nexus 7 and LG G3. This step consists of a server call to the other involved server. The next step in the figure the *Trust Relation Process* is proceed only if a cross-server payment is established. This step checks the server account information of the involved server and sends the payment request to this server. the arithmetic average of the step *Trust Relation Process* for both devices took 267ms (± 42.4ms) and 246ms (± 36.8ms). In consideration of the standard deviation both devices took in average the same time to completed this step. In comparison to the time to completed the *Server Process* the *Trust Relation Process* make up 66.8% of the time for the Nexus 7 and 65% for the LG G3. The arithmetic average for the *Trusted Server Call* on the Nexus 7 took 211ms (± 95.5ms) and on the LG G3 183ms (± 43.8ms). Without the the HTTP requests the required time on the other server took in average 176ms (± 42.4ms) on Nexus 7 and 143ms (± 36.8ms) on LG G3. The step *NFC Notification* took in average 192ms (± 42.4ms) and 246ms (± 34.6ms).
Concluded, the performance measurement for the total time on the Nexus 7 took in average 1017ms (± 145.0ms) and for the LG G3 in average it took 1159ms (± 18.4ms). The performance measurement for the total time for the payer role took around 1 second which is a good performance.

6.3.3 Comparison Payee and Payer

Figure *fig:evalDiffComparison* compares the performance measurements for the role payee and payer in a cross-server payment. The required time to complete a cross-server payment took for both cases in average more than one second. The request (payee) of Bitcoins needed in average more time to completed has sending the Bitcoins (payer) to another user. For the Nexus 7 the case requesting Bitcoins took in average 1193ms and the sending Bitcoins took in average 1017ms. However, for the LG G3 the requesting took in average 1460ms and the sending 1159ms. For the Nexus 7 there is no difference in time to request or send Bitcoins but the LG G3 had in average around 250-300ms longer to proceed a request compared to a send. The time differences can be seen in Figure 6.2. The first step *NFC Exchange* needed in average much longer for the payee scenario than for the payer. The difference is related to the message exchange when the Initator is a payee or a payer. Additional, there LG G3 took for both scenarios in average longer than the Nexus 7 for the first step and this is related to the specific feature to the devices. This need to be evaluated in a further research scope. There exists also a time difference in the step *Trusted Server Process* which affects the average times of all steps that includes this step. For the payee the step *Trusted Server Process* took on both devices in average 206ms and 269ms and for the payer it took in average 176ms and 143ms. This has to do with an atomic step in *Trusted Server Process* which is discussed in Section 6.4. It can be seen that the steps *Trust Relation Process* and *Trust Server Call* have the same form like the *Trust Server Process* that also indicates that the time difference is related to the step proceeded by the trusted server. Further, the *NFC Notification* took for the LG G3 for both roles longer in compared to Nexus 7. This has to do with the specific features of a device.

6.4 Performance Measurement for the Server

In this Section the server performance during a payment over NFC is measured. In the Section 6.2 and 6.3 the specific feature of each devices had an impact on the total required time to complete a payment. For this scenario only the steps on the server is evaluated. This means, the measurement starts when the server received a request object and returned a response object to the client. This Section first introduces performance measurement for the scenario when both involved users are registered on the same server. Second the performance measurement for the cross-server payment is discussed.

6.4.1 NFC Payment

The Figure 6.3 shows the arithmetic average and the standard deviation of the essential steps on the server during a payment over a NFC. The last bar indicates the total time required to complete the steps on the server. A payment request is send by the Client to the server. After the request is serialized the step *Get Users* looks for both users received with the request. In the next step the signed request is verified. The server checks if the payment request is from the user who claims to be. The server stores the public key of each user. Thus, the server can verify users registered on the server. After the verification, the data is stored into the database. This is the step *Persist DB*. The last step in the Figure 6.3 is the *Create Payment Response* which creates a response with the information that had been stored. This request is signed by the server for security reasons and send back to the Client. The results for the steps are based on tables in Section G.4.In this scenario both users the Initiator and the Responder are registered on the same server.
The arithmetic average of the *Server Process* in a payee role took 155ms (± 21.9ms) with the Nexus 7 and 145ms (± 30.2ms) with the LG G3. For the payer the arithmetic average was 128ms (± 21.4ms) and 127ms (± 27.3ms). The step *Get Users* took in average 10ms and 11ms as a payee and payer. The different devices had also no influence. For both devices and roles this step took in average the same time. The *Verify Signed Users* had an arithmetic average 33ms (± 6.1ms) and 37ms (± 19.6ms) as payee for Nexus 7 and LG G3. As payer in average it took 34ms (± 7.6ms) and 34ms (± 9.7ms) on the Nexus 7 and LG G3. The step *Persist DB* had on both devices and also for both roles the same average. The last step *Create Payment Response* 19 (± 13.2ms) 14ms (± 4.2ms) 15ms (±

5.0ms) 12 (± 0.8ms). Compared, the step *Verify Signed Users* and *DB Persists* consume the most time. But the *Verify Signed Users* takes double the time to complete as payee than as payer. This can be explained, that in the payee there exists two signed object, One by the Initiator and one by the Responder. For the payer only the username of the Responder is needed and only one object is signed. Further, the sum of the steps in the Figure 6.3 make around 65% up to 75% of the measured time of *Server Process*. There exists a gap that consumes a lot compared to the 4 described steps. An explanation for the gap is, ther is the time which is needed to serialize a object o send it back to the client consumes a part of the gap. But the gap is still there and one reasonable argument can be the time an object needs to each step of the code even the step which are not performed needs time. Another will be the programming language itself has settings which leads to that. The real reason can not be said and a second evaluation has to be conducted to verify the gap.

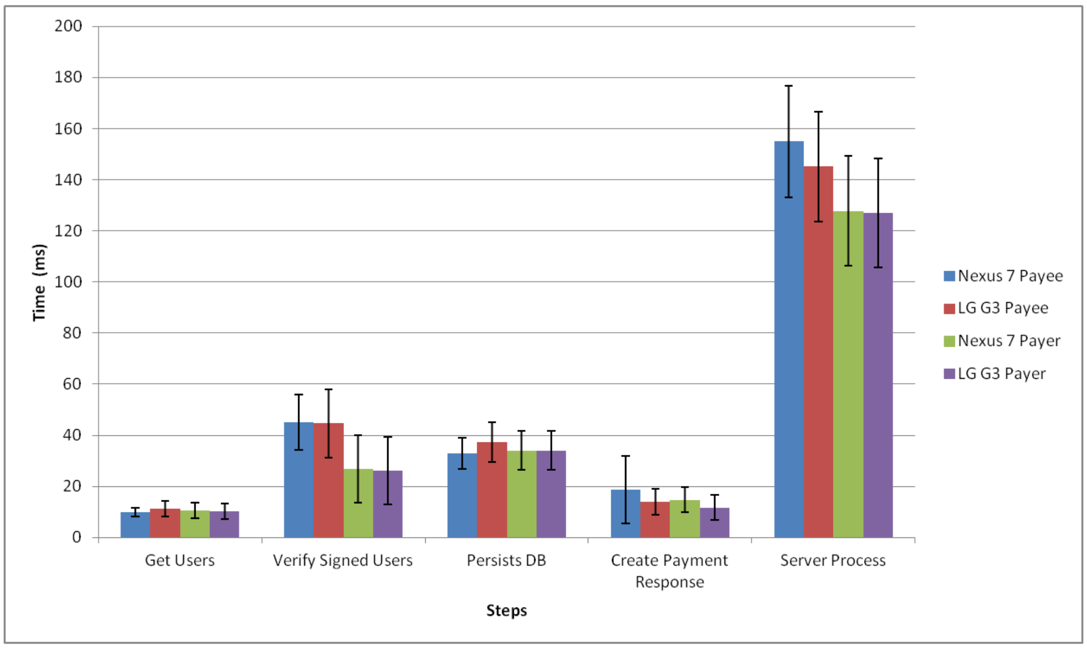


Figure 6.3: Mean and Standard Deviation Same Server

6.4.2 Cross-Server Payment

The Figure 6.4 shows the arithmetic average and the standard deviation of the essential steps on the server during a cross-server payment. The last bar indicates the total time required to complete the steps on the server. On the figure the steps which are part of another steps are on the left site of the step that consist this step. Further, the bars are divided into 2 group of colors. The two group of color represents the two server which are involved in the cross-server payment. The arithmetic average and the standard deviation are based on the results of Tables in Section G.5 Appendix G. Besides the step *Server Process* the steps *Trust Relation Process* and the step *Trusted Server Process* are essential steps on the server.. The *Trust Relation Process* consists of the steps:

- *Server Check Accounts*: Checks if the balance limits are not exceeded and if the trust relation allows a payment.
- *Sign Server*: A payment request is signed by the server, *Server Call*, sends the payment request to the trusted server and receives the payment response.
- *Server Call*: Sends the payment request to the trusted server and receives the payment response.
- *Decode Verify Server*: Serialize the received response and verifies the signature.

The step *Trust Relation Process* is only proceeded if a cross-server payment has been initialized. The other step *Trusted Server Process* is the time required for the trusted server to do cheeks, verifications and manipulations. The atomic step are followed:

- *Trusted Server Get Users*: Returns the user which belong to the trusted server and also the server information of the server that sent the request.
- *Trusted Server Verify Server*: Verifies that the signed payment request is from the server that claims to be.
- *Trusted Server Check Accounts*: Checks if the balance limits are not exceeded and if the trust relation allows a payment.
- *Trusted Server Verify User*: Verifies the signed payment object of the users.
- *Trusted Server Persists DB*: Stores the changes into the database.

There are more steps involved which are not shown in Figure 6.4. This means the sum of the existing steps are not the total time of the *Trusted Server Process*. The same is valid for the step *Server Process*. Minor steps have been ignored.
The total time measured for the *Server Process* took in average 438ms (± 56.2ms) and 418ms (± 36.2ms) as a payee and 391ms (± 32.2ms) and 408ms (± 408ms) as a payer. More than half of the required time consumed the step *Trust Relation Process* with 303ms (± 60.7ms) and 281ms (± 24.7) as a payee and 256ms (± 30.8ms) and 263ms (± 22.5ms) as a payer. The *Trust Relation Step* consists of the server call to the other server. Hence, the step *Trust Relation Process* consumes the most time on the server for a cross-payment process. The step *Trusted Server Verify Users* has a time difference if the payment is initiated as a payee compared to a payer. In a payer role only the payer request is signed and send to the server. The Responder (payee) sends his name to the Initiator for a payment and therefore the trusted server has no payee request to verify.
Compared the measured performance of the step *Trusted Server Process* to the *Server Process* the process accomplish by the trusted server consumed one third of the required time. This is justifiable, the *Trusted Server Process* has multiple atomic steps to perform before the response returns to the server of the Initiator. The atomic steps on the trusted server are repetition of the atomic steps on the server of the Initiator. This means more than half of the consumed time of *Server Process* is performed twice. Once the server of the Initiator and once the trusted server proceeded the steps. The sum of the nested

steps (*Get Users*. *Trust Relation Process*, *Verify Signed Users*, *Persists*) of *Server Process* makes up around 85% of the total time required for *Server Process*. The difference can be explained with three further steps wich were not part of this evaluation. The steps *Decode Request* at the beginning, *Created Payment Response* the payment response which is returned back to the client and the *Transfer Object*, the object which serilaized and send back. The *Transfer Object* consists of the object *Create Server Payment*.

6.5 Summarize

This Section summarize the Chapter and essential key findings are discussed::

- The NFC payment has been performed fast. A usual payment between users registered on the same server have performed in average of around 800ms. This means from the start of the NFC up to the success notification. As a payer the measured performance took in average around 50ms -100ms longer which is irrelevant.

- A cross-server payment performed in average around 1 up to 1.5 seconds. Considering that that two servers were involved during the payment request this time is acceptable and applicable for instnat payments with Bitcoins. The cross-server payment performed much faster than the MBPS used in the test-run in February 2014.

- The difference of the cross-server payments compared to usual payments is related to the step *Trust Relation Process* see Figure 6.2. This step is only performed when two users of different servers with on a trust relation do a payment over NFC. In average this step took around 300ms as payer and 250ms as payee. The step is has been the most time consuming step on the server.

- When compared the devices Nexus 7 and the LG G3, there was a time difference for the payer and for the payee as Initiator. The Nexus 7 outperformed the LG G3 in every scenario. The main reason was that the LG G3 took in average more time to perform for the step *NFC Exchange* and for the step *NFC Notification*. Hence, both mentioned steps are proceeded over the device. The time difference has to be device specific. Further evaluation should be considered on the NFC Library with different devices.

- Approximately the *Server Call* consumed half of the required time to completed a cross-server payment. This is due to nested HTTP request and also the steps which are repeated on the trusted server. The evaluation on the server showed that the total sum of the steps does not make the complete step of the *Server Process*. This gaps have to be evaluated and checked if it can be optimized.

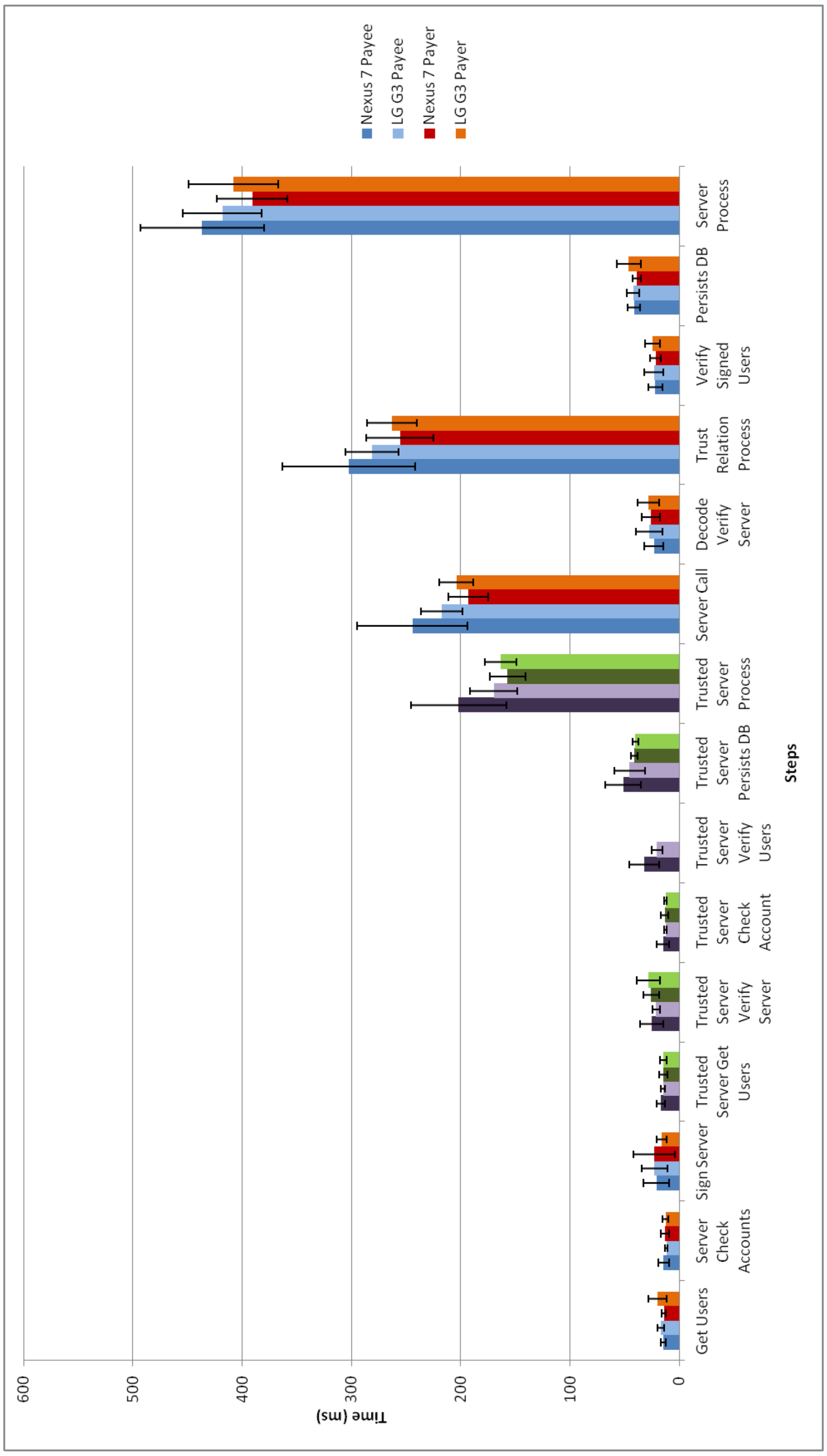


Figure 6.4: Mean and Standard Deviation

Chapter 7

Summary and Conclusions

Two test runs were conducted with success. Participants had the chance to pay their consumption with Bitcoins by means of an Android device. To use the MBPS application participants created a user account and topped their accounts with Bitcoins. The Bitcoins were hold on a server maintained by the CSG group. The server served as a centralized entity to store the user accounts and the Bitcoins paid in. This client/server architecture is a complete opposite to the P2P nature of the Bitcoin network. Further, a centralized server has disadvantages compared to a decentralized network based on accessibility and distribution.
The goal of this thesis is to create a federation for MBPS servers. To enable payment in less than a second, a client/server architecture will be used. However, MBPS servers can federate allowing users from other MBPS servers. Furthermore, a web-frontend has to be developed to administrate server in the federation network. Finally, participant have to be allowed to chose a federation based on their own interest by means of an Android Client. The performance of allowing cross-server payments has to be evaluated in comparison to the previous version. To achieve these goals a trust relation model has been analysed, developed and tested. further, a web interface has been developed to administrated the trust relation between servers. For the web interface Angularjs has been used as framework on the front-end. To distinguish users of the web interface and the Android Client a role system has been implemented. Furthermore, the Android Client has been extended to allow the user to decide which server stores the user account. Finally, the NFC payment process has been extended to allow cross-server payments.
The cross-server payment has been evaluated with respect to the performance using the previous version as baseline. The goal of the evaluation was to compare the new cross-server payment and achieve a result close to the current performance. The result pointed out that a cross-server payment took in average between 1 and 1.5 seconds. For the payee as Initiator the Nexus 7 had an arithmetic average of 1193ms to completed a payment compared to a usual payment with an average of 807ms. This is a increase of 47.8%. For the scenario as payer the Nexus 7 had an arithmetic average of 1017ms compared to the usual payment with an average of 744ms to completed a payment. This is an increase of 36.7%. The performance results for the cross-server payment is still in the range of the required time around 1 seconds. The server call consumed the most time of the total measured time for a cross-server payment with around 65%. The reason the server call claims

the most time is because of the second server. The server of the Initiator calls the server of the Responder to verify and accept the payment. In addition, it has been observed that the steps to satisfy the payment success have been repeated by the second server. A further performance difference has been observed between the two devices Nexus 7 and LG G3 both were used for the evaluation. The Nexus 7 outperformed the LG G3 in each scenario. This has to be based on the specific features of each device.
For a future work, the next step is to make further evaluations on the key findings. This means an analyse has to be conducted to reduce the repeated steps on both server in consideration of the system Shibboleth [38]. Further, the devices specific features should be analysed and further devices should be involved to the evaluation. Besides the improvement of the findings the *Hyprid-Trust* with an escrow account should be analysed and implemented into the system. There should be a solution that operates mostly over the existing web-fronted to create and remove an escrow account. The Bitcoin multisig should be included to the development of the escrow account. Further, the user interface of the web-site should be improved and new feature should be developed. For example, the user gets rights to manage the user account besides the server accounts. Finally, a point-of-sale should be integrated to the web-fronted, to allow customers like the Mensa to manage their Bitcoins. This means they should get the right to access the web-site and manage the exchange of the Bitcoins to another excepted currency by including a provider to the web-site over the API.

Bibliography

[1] “Bitcoin.” `http://bitcoin.org/en`. Accessed: September 2014.

[2] “Neue zürcher zeitung, winklevoss-zwillinge planen bitcoin-fonds.” `http://www.nzz.ch/aktuell/digital/tyler-und-cameron-winklevoss-fonds-bitcoin-1.18109996`. Accessed: September 2014.

[3] “Das geld, das aufstreckt.” `http://www.tagesanzeiger.ch/digital/internet/Das-Geld-das-aufschreckt/story/29480474?track`. Accessed: September 2014.

[4] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System.” `https://bitcoin.org/bitcoin.pdf`. Accessed: September 2014.

[5] “Blockchain.” `http://blockchain.info/en/stats`. Accessed: September 2014.

[6] “Bitcoin Wiki, Confirmation.” `https://en.bitcoin.it/wiki/Confirmation`. Accessed: September 2014.

[7] T. Bamert, C. Decker, L. Elsen, R. Wattenhofer, and S. Welten, “Have a Snack, Pay with Bitcoins,” *13th IEEE International Conference on Peer-to-Peer Computing (P2P), Trento, Italy.*, September 2013.

[8] J. Memeti, M. A. Bekooglu, and S. Kaeser, “Bitcoin in Practice,” 2014.

[9] “Mensa uzh binzmühle.” `http://www.mensa.uzh.ch/standorte/mensa-uzh-binzmuehle.html`. Accessed: September 2014.

[10] “Sbex - swiss bitcoin broker.” `http://www.sbex.ch`. Accessed: September 2014.

[11] S. Kaeser, “Improving the Mobile Bitcoin Payment System (MBPS),” Master’s thesis, 2014.

[12] J. Memeti, “Protocol Design and Implementation for a Fast and Reliable Mobile Bitcoin Payment System (MBPS) with two-way NFC,” Master’s thesis, 2014.

[13] “Two-factor bitcoin.” `https://bit2factor.org/`. Accessed: June 2014.

[14] M. Rosenfeld, “Analysis of hashrate-based double-spending,” 2012.

[15] A. E., C. s., and K. G., “Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin,” 2012.

[16] "Android 4.4 kitkat market share grew just 0.4 percent in january." http://www.theinquirer.net/inquirer/news/2326994/android-44-kitkat-market-share-grew-just-04-percent-since-january. Accessed: September 2014.

[17] "Most common sdk versions." http://www.appbrain.com/stats/top-android-sdk-versions. Accessed: September 2014.

[18] "Game theory." http://www.investopedia.com/terms/g/gametheory.asp. Accessed. September 2014.

[19] T. L. Turocy and B. von Stengel, "Game theory"," 2001.

[20] G. W., S. R., and S. B., "An experimental analysis of ultimatum bargaining," *Economic Behavior and Organization*, vol. 3, pp. 367–388, December 1982.

[21] B. J., D. J., and M. K, "Trust, reciprocity, and social history," *Games and Economic Behavior*, vol. 10, pp. 122–142, July 1995.

[22] B. M. and U. J.C., "Does the trust game measure trust?," *Economics Letters*, vol. 115, no. 1, pp. 20–23, 2012.

[23] "Bestiary of behavioral economics/trust game." http://en.wikibooks.org/wiki/Bestiary_of_Behavioral_Economics/Trust_Game. Accessed: September 2014.

[24] "Escrow." http://www.investopedia.com/terms/e/escrow.asp. Accessed: September 2014.

[25] "Bitcoin escrow service." https://en.bitcoin.it/wiki/Bitcoin_Escrow_Service. Accessed: September 2014.

[26] "Multisig: The future of bitcoin." http://bitcoinmagazine.com/11108/multisig-future-bitcoin/. Accessed: September 2014.

[27] "Silk road shut down, alleged owner arrested." http://bitcoinmagazine.com/7362/silk-road-shut-down-alleged-owner-arrested/. Accessed: September 2014.

[28] "Transaction malleability: Mtgox's latest woes." http://bitcoinmagazine.com/10093/transaction-malleability-mtgoxs-latest-woes/. Accessed: September 2014.

[29] "Joint statement regarding mtgox." http://blog.coinbase.com/post/77766809700/joint-statement-regarding-mtgox. Accessed: September 2014.

[30] "Almost half a billion worth of bitcoins vanish." http://online.wsj.com/news/article_email/SB10001424052702303801304579410010379087576-lMyQjAxMTA0MDIwNzEyNDcyWj. Accessed: September 2014.

[31] "Securing your wallet." https://bitcoin.org/en/secure-your-wallet. Accessed: September 2014.

[32] "Contracts." `https://en.bitcoin.it/wiki/Contracts#Theory`. Accessed: September 2014.

[33] "The future of bitcoin escrow." `http://www.opine.me/future-of-bitcoin-escrow/`. Accessed: September 2014.

[34] R. Saadi, J.-M. Pierson, and L. Brunie, "T2d: a peer to peer trust management system based on disposition to trust," in *SAC '10 Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 1472–1478, ACM New York, March 2010.

[35] L. Xiong and L. Liu, "Peertrust: supporting reputation-based trust for peer-to-peer electronic communities," vol. 16, Knowledge and Data Engineering, IEEE Transactions on, July 2004.

[36] X. Liu, M. Zheng, and Y. Liu, "Dynamictrust: three-dimensional dynamic computing model of trust in peer-to-peer networks," in *GEC '09 Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pp. 1–6, ACM New York, June 2012.

[37] S. Liu, Y. Yu, J. Xu, and Z. Huang, "A preventing fraud trust model in p2p networks," in *Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), 2012 IEEE 26th International*, pp. 2305 – 2311, IEEE, May 2012.

[38] "Shibboleth." `http://shibboleth.net/`. Accessed: September 2014.

[39] "What is shibboleth?." `http://shibboleth.net/about/`. Accessed: September 2014.

[40] "Spring." `http://spring.io/`. Accessed: October 2014.

[41] "Spring-security." `http://projects.spring.io/spring-security/`. Accessed: October 2014.

[42] "10 status code definitions." `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`. Accessed: October 2014.

[43] "PostgreSQL." `http://www.postgresql.org`. Accessed: October 2014.

[44] "Hibernate." `http://hibernate.org`. Accessed: September 2014.

[45] "Javaserver pages technology." `http://www.oracle.com/technetwork/java/javaee/jsp/index.html`. Accessed: October 2014.

[46] "bitcoind." `https://en.bitcoin.it/wiki/Bitcoind`. Accessed: October 2014.

[47] "Bitcoin-JSON-RPC-Client." `https://bitbucket.org/azazar/bitcoin-json-rpc-client/wiki/Home`. Accessed: October 2014.

[48] "Bitstamp." `https://www.bitstamp.net/`. Accessed: October 2014.

[49] "Xeiam XChange." `http://xeiam.com/xchange`. Accessed: October 2014.

[50] "Mappingjacksonhttpmessageconverter." `http://docs.spring.io/spring-framework/docs/4.0.x/javadoc-api/org/springframework/http/converter/json/MappingJacksonHttpMessageConverter.html`. Accessed: October 2014.

[51] "Java persistence api." `http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html`. Accessed: October 2014.

[52] "Angularjs." `https://angularjs.org/`. Accessed: October 2014.

[53] "Angular http service." `https://docs.angularjs.org/api/ng/service/$http`. Accessed: October 2014.

[54] "Cross-origin resource sharing." `http://www.w3.org/TR/cors/`. Accessed: October 2014.

[55] "About dbunit." `http://dbunit.sourceforge.net/`. Accessed: October 2014.

[56] "Nexus 7 made for what matters.." `http://www.google.ch/nexus/7/`. Accessed: October 2014.

[57] "Lg g3." `{http://www.lg.com/ch_de/mobiltelefone/lg-D855-g3}.\newblockAccessed:October2014.\bibitem{hce2}``{Host-basedCardEmulation}.`
`''\url`https://developer.android.com/guide/topics/connectivity/nfc/hce.html. Accessed: October 2014.

[58] "Advanced card systems ltd., acr122u usb nfc reader." `http://www.acs.com.hk/en/products/3acr122u-usb-nfc-reader`. Accessed: Semptember 2014.

[59] "Game theory." `http://www.cdam.lse.ac.uk/Reports/Files/cdam-2001-09.pdf`. Accessed: September 2014.

[60] "The basics of game theory." `http://www.investopedia.com/articles/financial-theory/08/game-theory-basics.asp`. Accessed: September 2014.

List of Figures

List of Tables

Appendix A

Comparison MBPS and CoinBlesk

In Section A.1 few screenshots of MBPS and CoinBlesk are introduced. Section A.2 presents numbers and facts about both test runs. More information about MBPS and CoinBlesk are provided in the reports [8, ?].

A.1 Screenshots Mobile Client

The figures from A.1 to A.6 representing views of the MBPS application. The main view in figure A.3 has features for current balance and buttons to redirect the user to the pay (accepting/declining request), the receive (requesting Bitcoins) and the history (made transactions) view.

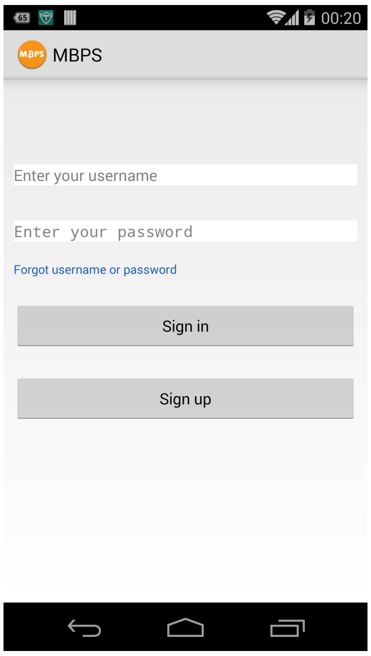


Figure A.1: Login

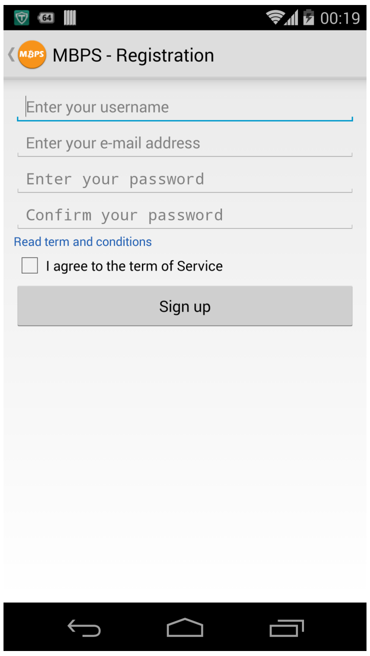


Figure A.2: Registration

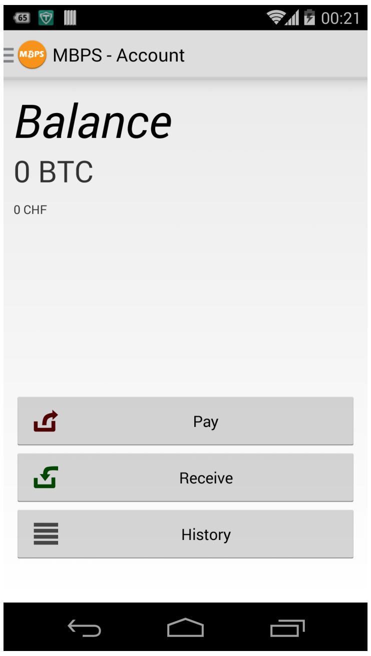


Figure A.3: Main

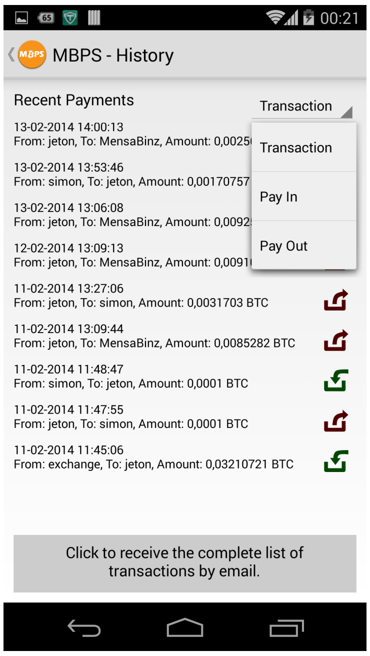

00:21
MBPS - History
Recent Payments
Transaction
13-02-2014 14:00:13
From: jeton, To: MensaBinz, Amount: 0,0025
13-02-2014 13:53:46
From: simon, To: jeton, Amount: 0,00170757
13-02-2014 13:06:08
From: jeton, To: MensaBinz, Amount: 0,0092
12-02-2014 13:09:13
From: jeton, To: MensaBinz, Amount: 0,0091
11-02-2014 13:27:06
From: jeton, To: simon, Amount: 0,0031703 BTC
11-02-2014 13:09:44
From: jeton, To: MensaBinz, Amount: 0,0085282 BTC
11-02-2014 11:48:47
From: simon, To: jeton, Amount: 0,0001 BTC
11-02-2014 11:47:55
From: jeton, To: simon, Amount: 0,0001 BTC
11-02-2014 11:45:06
From: exchange, To: jeton, Amount: 0,03210721 BTC
Transaction
Pay In
Pay Out
Click to receive the complete list of transactions by email.

Figure A.4: History

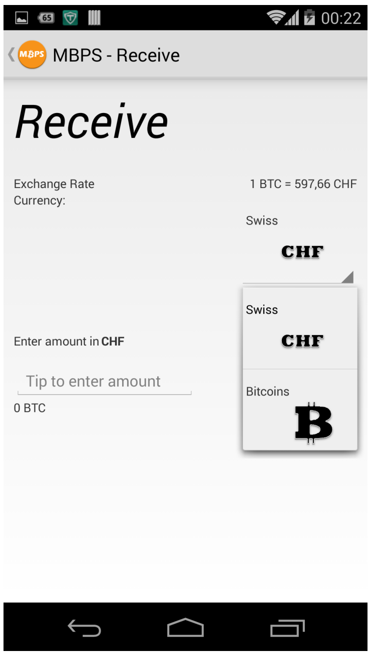

00:22
MBPS - Receive
Receive
Exchange Rate 1 BTC = 597,66 CHF
Currency: Swiss CHF
Enter amount in CHF
0 BTC

Figure A.5: Request

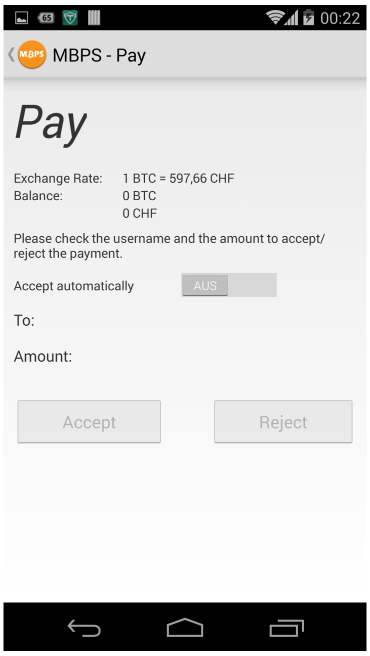
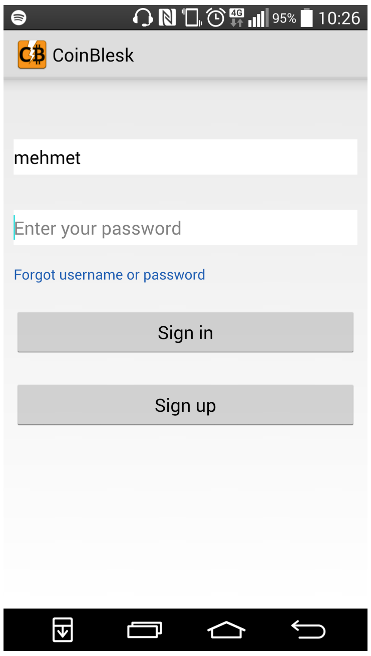

00:22
MBPS - Pay
Pay
Exchange Rate: 1 BTC = 597,66 CHF
Balance: 0 BTC
0 CHF
Please check the username and the amount to accept/reject the payment.
Accept automatically AUS
To:
Amount:

Figure A.6: Accept/Decline request

The views of CoinBlesk makes it obvious that views have been redesigned to the need of the users. The main view in figure A.9 consist of the current balance, the last transaction made, an animation bring out that the device is ready for a NFC transaction, and a button that redirects the user to the payment option view see figure A.11. In the payment option view the user can select which option will be performed to transfer Bitcoins. The send view in figure A.13 is a new features that was not available in the MBPS application. The user can send the entered amount to another user by holding the devices together. By the way, there is no view to redirect for accepting or declining a request. This one is replaced by the animation aforementioned. In addition, the views in the figure A.14 and A.15 are new and support the user during the NFC connection with feedback.


10:26

CoinBlesk

mehmet

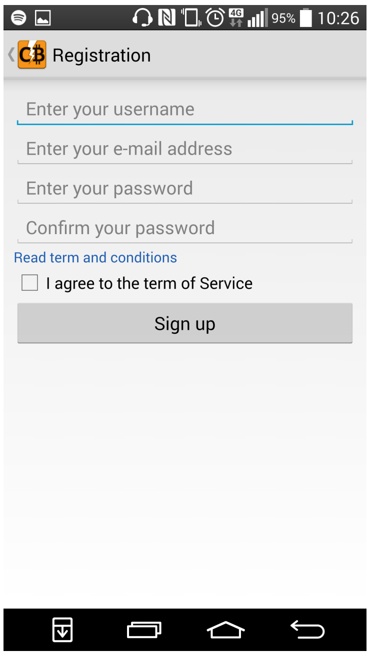
Enter your password

Forgot username or password

Sign in

Sign up

Figure A.7: Login


10:26

Registration

Enter your username

Enter your e-mail address

Enter your password

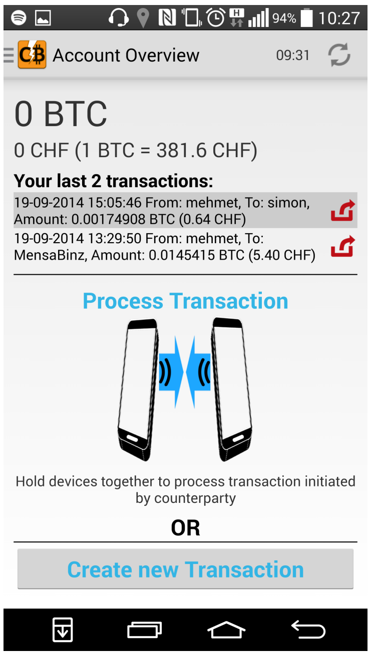
Confirm your password

Read term and conditions

I agree to the term of Service

Sign up

Figure A.8: Registration


10:27

Account Overview 09:31

0 BTC

0 CHF (1 BTC = 381.6 CHF)

Your last 2 transactions:

19-09-2014 15:05:46 From: mehmet, To: simon, Amount: 0.00174908 BTC (0.64 CHF)

19-09-2014 13:29:50 From: mehmet, To: MensaBinz, Amount: 0.0145415 BTC (5.40 CHF)

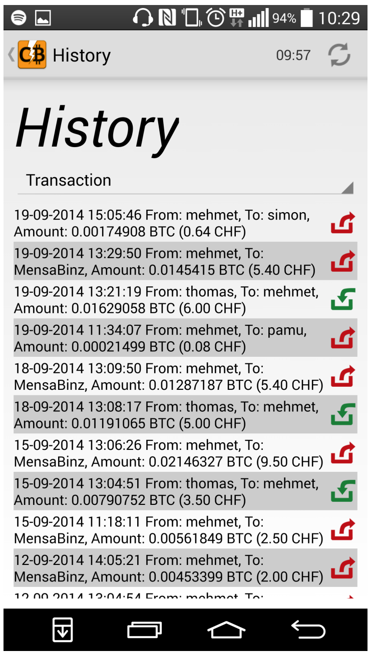
Process Transaction

Hold devices together to process transaction initiated by counterparty

OR

Create new Transaction

Figure A.9: Main


10:29

History 09:57

History

Transaction

19-09-2014 15:05:46 From: mehmet, To: simon, Amount: 0.00174908 BTC (0.64 CHF)

19-09-2014 13:29:50 From: mehmet, To: MensaBinz, Amount: 0.0145415 BTC (5.40 CHF)

19-09-2014 13:21:19 From: thomas, To: mehmet, Amount: 0.01629058 BTC (6.00 CHF)

19-09-2014 11:34:07 From: mehmet, To: pamu, Amount: 0.00021499 BTC (0.08 CHF)

18-09-2014 13:09:50 From: mehmet, To: MensaBinz, Amount: 0.01287187 BTC (5.40 CHF)

18-09-2014 13:08:17 From: thomas, To: mehmet, Amount: 0.01191065 BTC (5.00 CHF)

15-09-2014 13:06:26 From: mehmet, To: MensaBinz, Amount: 0.02146327 BTC (9.50 CHF)

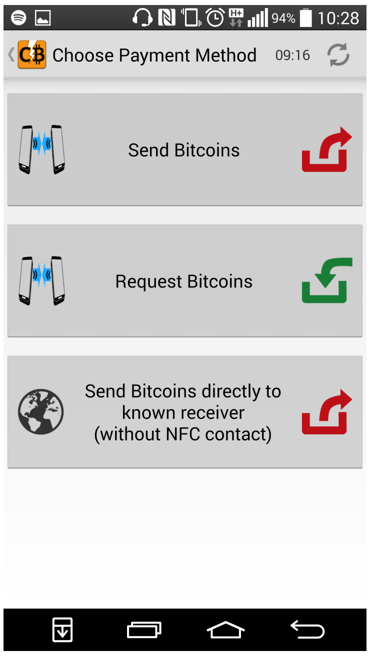
15-09-2014 13:04:51 From: thomas, To: mehmet, Amount: 0.00790752 BTC (3.50 CHF)

15-09-2014 11:18:11 From: mehmet, To: MensaBinz, Amount: 0.00561849 BTC (2.50 CHF)

12-09-2014 14:05:21 From: mehmet, To: MensaBinz, Amount: 0.00453399 BTC (2.00 CHF)

12-09-2014 13:04:54 From: mehmet, To:

Figure A.10: History


10:28

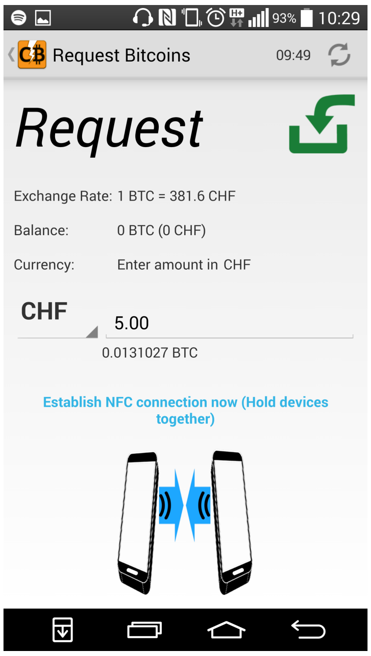
Choose Payment Method 09:16

Send Bitcoins

Request Bitcoins

Send Bitcoins directly to known receiver (without NFC contact)

Figure A.11: Choose Payment method


10:29

Request Bitcoins 09:49

Request

Exchange Rate: 1 BTC = 381.6 CHF

Balance: 0 BTC (0 CHF)

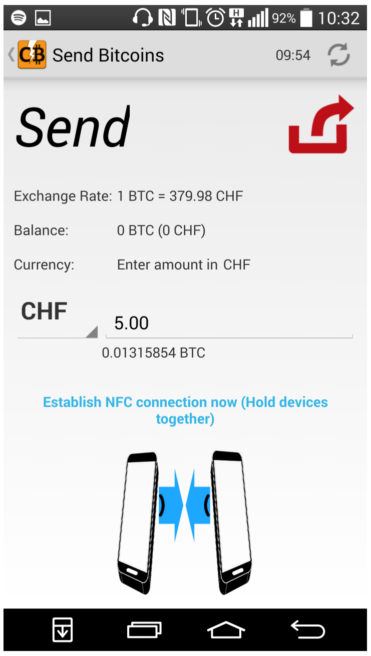
Currency: Enter amount in CHF

CHF 5.00

0.0131027 BTC

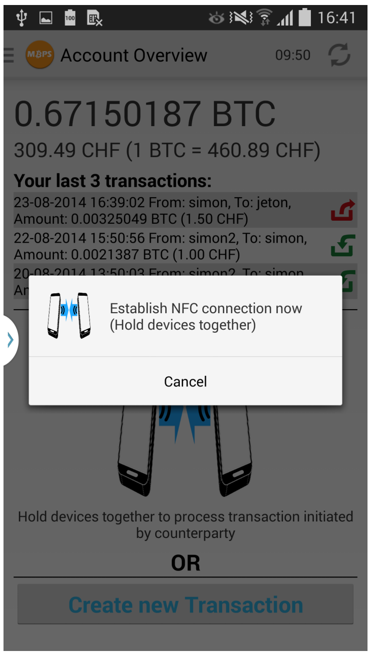
Establish NFC connection now (Hold devices together)

Figure A.12: Request



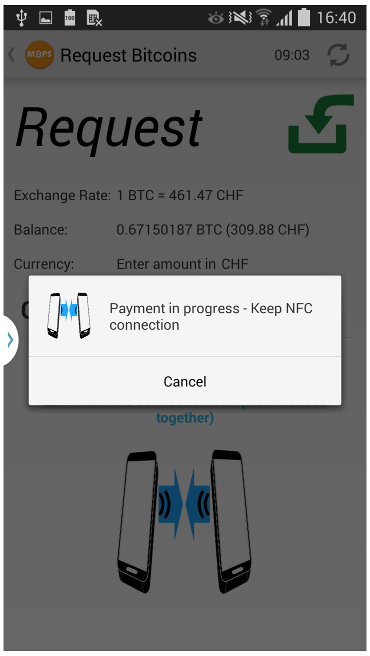
Exchange Rate: 1 BTC = 379.98 CHF
Balance: 0 BTC (0 CHF)
Currency: Enter amount in CHF
CHF 5.00
0.01315854 BTC

Figure A.13: Send



0.67150187 BTC
309.49 CHF (1 BTC = 460.89 CHF)
Your last 3 transactions:
23-08-2014 16:39:02 From: simon, To: jeton, Amount: 0.00325049 BTC (1.50 CHF)
22-08-2014 15:50:56 From: simon2, To: simon, Amount: 0.0021387 BTC (1.00 CHF)
20-08-2014 13:50:03 From: simon2, To: simon
Establish NFC connection now (Hold devices together)
Hold devices together to process transaction initiated by counterparty

Figure A.14: Establish NFC



Exchange Rate: 1 BTC = 461.47 CHF
Balance: 0.67150187 BTC (309.88 CHF)
Currency: Enter amount in CHF
Payment in progress - Keep NFC connection

Figure A.15: Payment in Progress

A.2 Results

To understand the scope of the first and second test run important numbers and facts are provided.

A.2.1 Test Run in February 2014

Duration	**Date**	**Mensa equipment**	**Required OS**	**Available Devices**
1 week	10.02 - 14.02	Nexus 10 tablet with NFC reader ACR122u [58]	Android 4.4. (kitkat) with HCE	Nexus 5, 7, 10

Table A.1: Pre-requirements of first Test Run

A.2.2 Test Run in September 2014

Duration	**Date**	**Mensa equip-ment**	**Required OS**	**Available Devices**
3 week	01.09 - 19.09	Nexus 10 tablet with NFC reader ACR122u [58]	Android 4.4. (kitkat) with HCE	All Devices expect HTC One(M7)

Table A.2: Pre-requirements of second Test Run

A.2.3 Result

	February 2014	September 2014
Transactions in the Mensa	42	117
Total Value in CHF	187.5	774.95

Appendix B

Game Theory

Section B.1 is a glossar of definitions used in game theory [59, 60]. Section B.2 provides more information about the experiments in [21, 22].

B.1 Glossary

Backward induction
The process of deducing a problem or a scenario backwards from the end to determine a sequence of optimal actions.

Game
A game is the formal description of a set of circumstances that has a result dependent on the actions of decision maker of a strategy.

Game Theory
Game theory is the formal study of decision-making.

Information set
The information available at a given time in the game. Perfect information is known and it describes that at any point every action that haven been made is known to all.

Nash Equilibrium
A Nash equilibrium is a list of strategies, where the optimal outcome of a game is one where player has an incentive to deviate from his or her chosen strategy after considering an opponent's choice in which no player can unilaterally change his strategy and get a better payoff.

Payoff
A payoff is a number (utility) that a player receives from arriving at a particular outcome.

The expected payoff incorporates the player's attitude towards risk.

Player (Individual)
A player is an agent who makes decisions in a game.

Rationality
The attitude of a player seeking the maximum payoff.

Strategy
A strategy is one of the given possible actions of a player.

B.2 Experiments

In a standard economical assumption individuals act in their own self interest. To prove this assumption several experiments were conducted. Below two experiments are introduced which set up a trust game to observe how individuals act in relation to the standard economic assumption.

First Experiment - 1995 ([21]))
The procedure in this game is as followed: Player 1 and 2 are endowed with 10 Dollar. Player can decide how much of the 10 Dollar will be transferred to the player 2. The experimenter triples the amount and sent. Player 2 is informed about the transfer of player 1 and decides how much will be send back. In standard prediction player 1 will send nothing and player 2 will do the same. The same is valid for backward induction. The results were followed:

- 32 games were tried.
- 30 games violated under the standard economic assumption.
- Vast majority of palyer 1 make a transfer.
- Player 2 sends back money.
- reflects unconditional kindness (altruism) or trust
- In average, player 1 gets back the amount that is sent

Second Experiment - 2010 ([22])
The setup of this experiment was done by first-year undergraduate students at the university of Lausanne. The first player (mover) were endowed with 10 Swiss francs and the second player altered between three option of 0, 10 and 20 Swiss francs. In addition, both player were informed about the situation of the other paired player. The same procedure was valid here like in experiment above and the result were followed:

- The game was done in 4 sessions. The session were organized in two methodological dimensions: manual vs internet-based games and between-subject design vs within-subject design.
- First player send in average 7.04 of their 10 Swiss francs to the second player and second player send 11.02 Swiss francs back.
- Only a small fraction conform to the perfect equilibrium with pure self-interest, by giving nothing. 11 % of first players and 20 % of second players in four sessions fulfil the selfishness.

Summarized
The experiments in both tests disprove the assumption under standard economics. One explanation why the players send money to another player can be related to reputation of trustworthiness. For example, when the trail was run multiple time between the same players, there is a reliableness build. But, as the majority of first players send money to the second, even on the first run of the game, an alternative explanation to the results is needed.
An alternative approach, denies that self-interest is the sole motivator to human actions. As Berg states, "Evolutionary models predict the emergence of trust because it maximizes genetic fitness." [21]. This will explain the action under the the trust game, at least for the first player. The explanation for the second player might depend on the evolutionary innateness of trust. Berg suggests that the response of reciprocation is triggered by the second player viewing that the fisrt players intend is an attemot to use trust to improve the outcome for both [23].

Appendix C

Android Client Mock-ups and Screenshoots

Section C.1 introduces the mock-ups created during the analysing and design process. The final screenshoots are presented in Section C.2.

C.1 Mock-Ups

Figure C.1 shows the the extension made to the Login screen. The description of of "*Server URL eingebn*" enables the user to enter the URL of the trusted server. After tapping on this description a pop up shows up. Figure C.2 is the pop up where the user can enter the URL. This design allows the user to not be concerned with server url an instead focus on the sign in. Once entered the url this url will be remembered until it is changed.
Figure C.3 shows the extensions of an additional column. This column is used to enter the url of the server that should persists the user account of the user. The next Figure C.4 shows an view which does not exists in the previous solutions. This view shows the different trust relation with other servers. The name, the trust level, the active balance and the balance limit is shown.

C.2 Screenshots

From Figure C.5 the improvements are shown. The Figure C.5 and C.6 shows the login and the registration screens. The improvements are made very simple and match to the current layout. Figure C.7 shows the new view that represents the trust relations between the servers. The most important informations for a user is given back. There are additional improvements done which were included during the implementation. Those improvements should support the user to focus on the important information. Figure C.8 shows the new look and feel of the history screen.

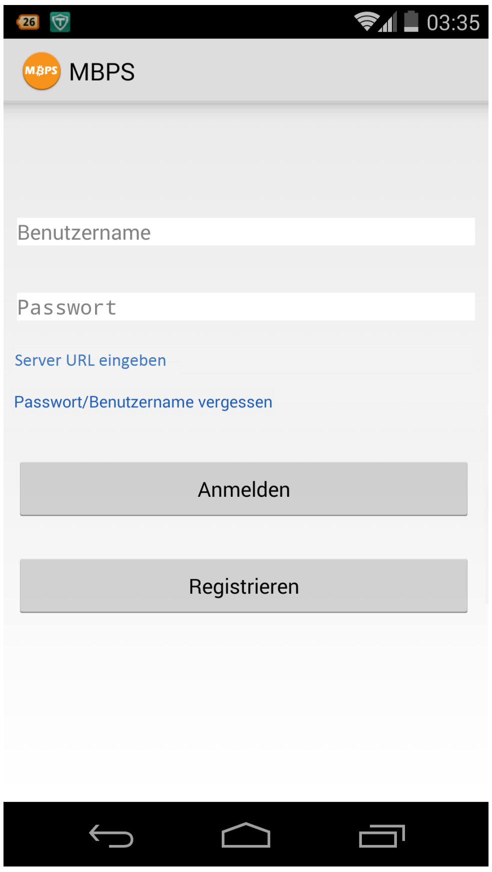


Figure C.1: Login Screen

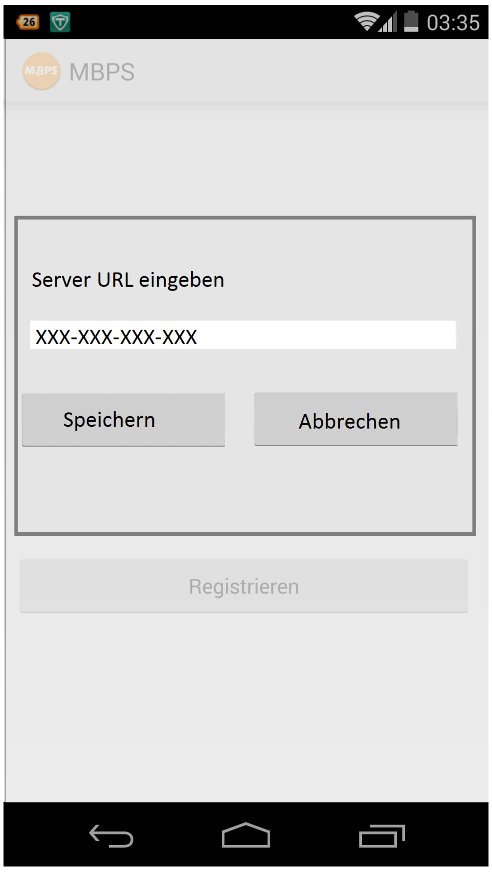


Figure C.2: Enter Server URL

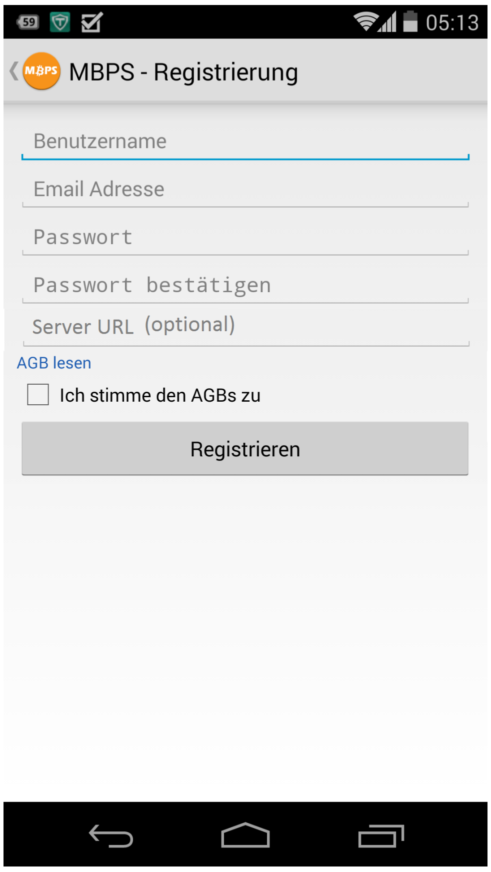
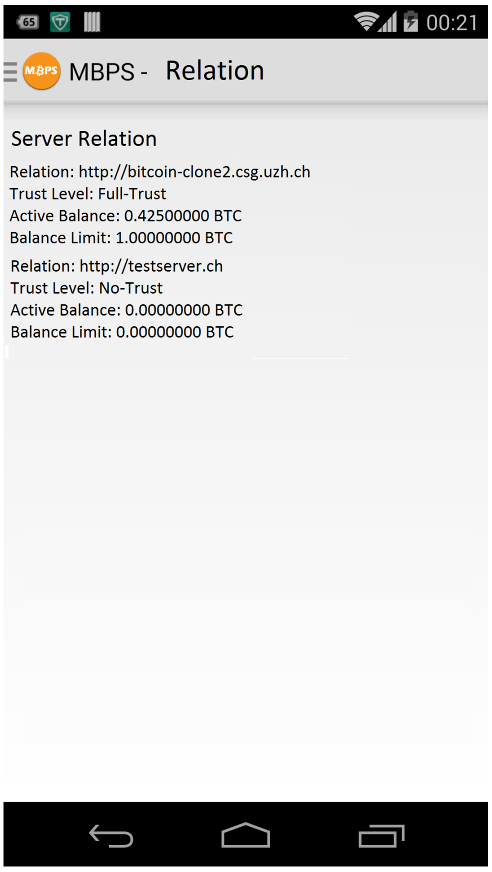


Figure C.3: Login Screen

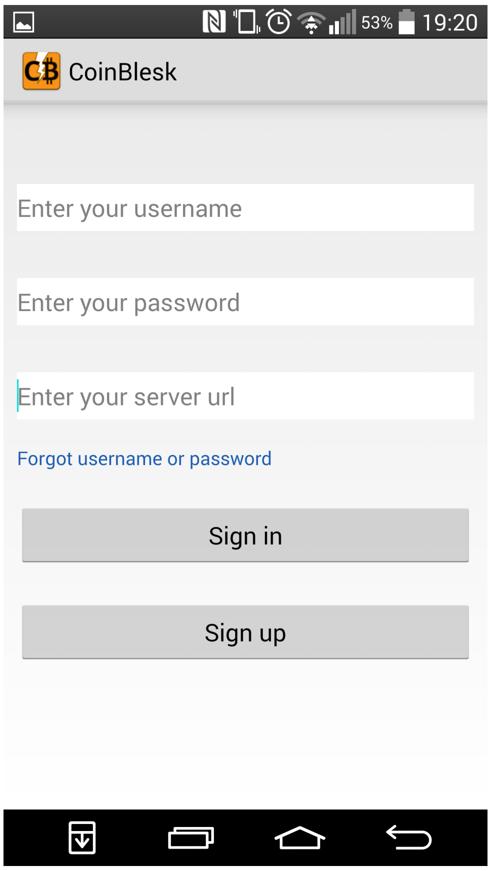


Server Relation

Relation: http://bitcoin-clone2.csg.uzh.ch
Trust Level: Full-Trust
Active Balance: 0.42500000 BTC
Balance Limit: 1.00000000 BTC

Relation: http://testserver.ch
Trust Level: No-Trust
Active Balance: 0.00000000 BTC
Balance Limit: 0.00000000 BTC

Figure C.4: Enter Server URL


53% 19:20

CoinBlesk

Enter your username

Enter your password

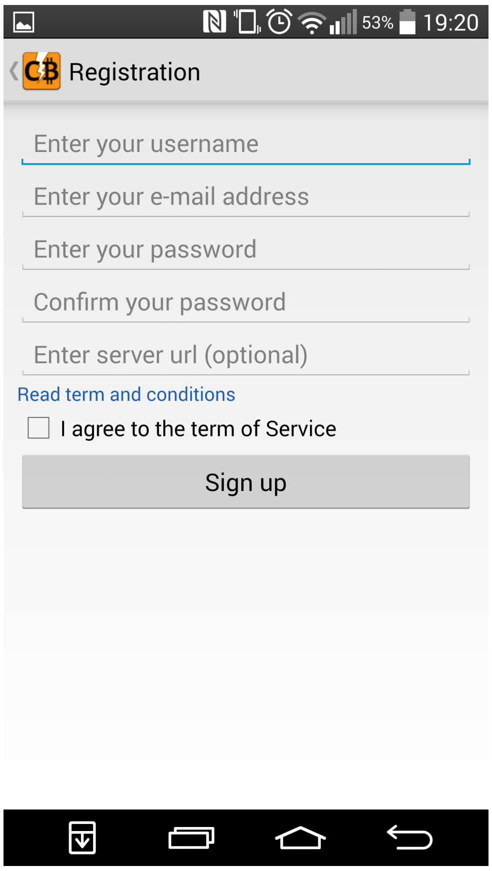
Enter your server url

Forgot username or password

Sign in

Sign up

Figure C.5: Login Screen


53% 19:20

Registration

Enter your username

Enter your e-mail address

Enter your password

Confirm your password

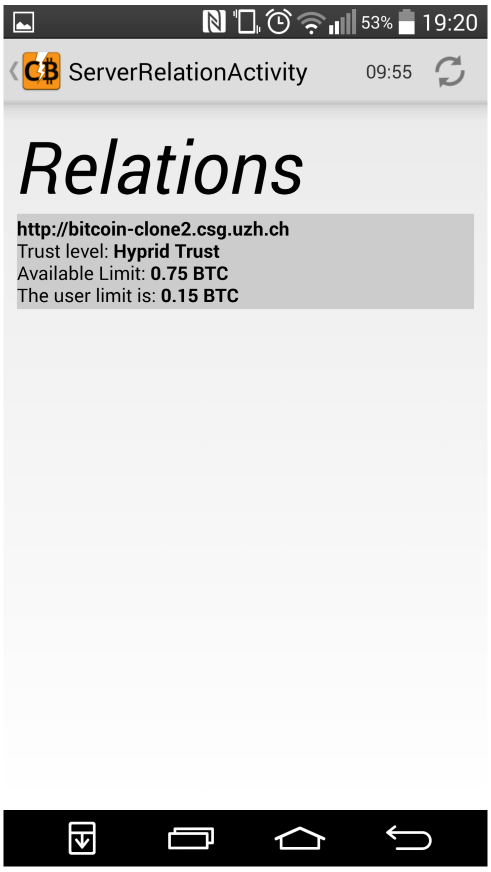
Enter server url (optional)

Read term and conditions

I agree to the term of Service

Sign up

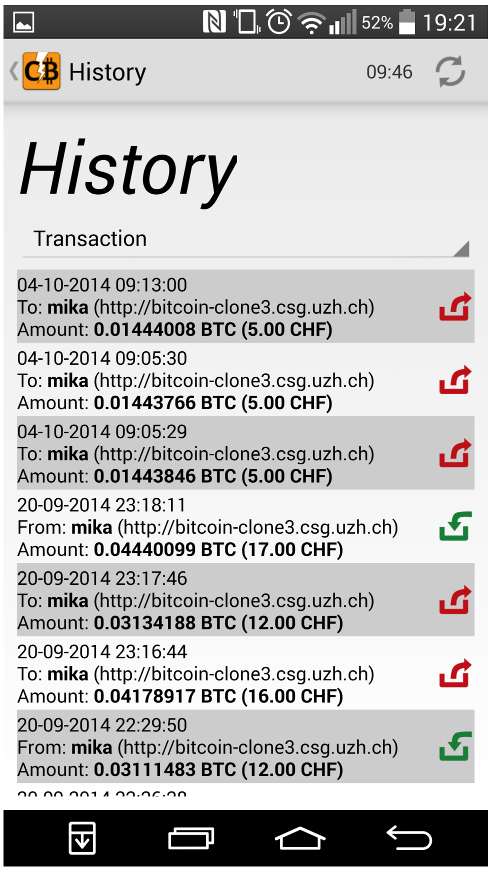
Figure C.6: Registration Screen



Relations

http://bitcoin-clone2.csg.uzh.ch
Trust level: **Hyprid Trust**
Available Limit: **0.75 BTC**
The user limit is: **0.15 BTC**

Figure C.7: Relation view



History

04-10-2014 09:13:00
To: **mika** (http://bitcoin-clone3.csg.uzh.ch)
Amount: **0.01444008 BTC (5.00 CHF)**

04-10-2014 09:05:30
To: **mika** (http://bitcoin-clone3.csg.uzh.ch)
Amount: **0.01443766 BTC (5.00 CHF)**

04-10-2014 09:05:29
To: **mika** (http://bitcoin-clone3.csg.uzh.ch)
Amount: **0.01443846 BTC (5.00 CHF)**

20-09-2014 23:18:11
From: **mika** (http://bitcoin-clone3.csg.uzh.ch)
Amount: **0.04440099 BTC (17.00 CHF)**

20-09-2014 23:17:46
To: **mika** (http://bitcoin-clone3.csg.uzh.ch)
Amount: **0.03134188 BTC (12.00 CHF)**

20-09-2014 23:16:44
To: **mika** (http://bitcoin-clone3.csg.uzh.ch)
Amount: **0.04178917 BTC (16.00 CHF)**

20-09-2014 22:29:50
From: **mika** (http://bitcoin-clone3.csg.uzh.ch)
Amount: **0.03111483 BTC (12.00 CHF)**

Figure C.8: History view

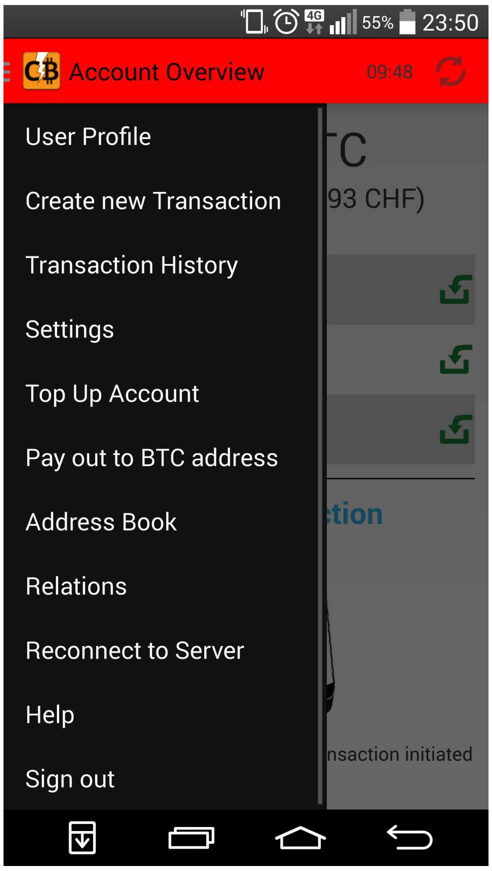


Figure C.9: Navigation menu

Appendix D

Web Page mock-ups and Screenshoots

The mock-ups and screenshots of the Web-page are illustrated in this Section. Section D.1 shows the mock-ups generated during the analyse phase.

D.1 Mock-Ups

The Login interface D.1 is very simple and has two text areas that serve as sign in. The home interface is the standard interface if logged. There is all important information shown for the user. The Bitcoin amount in the whole system, the last server transactions, credentials of the logged user and the admin list. The user can updated the email address or the password. Additional new users can be invited to join. The Figures D.3 and D.4 are related to the trust relation. The first picture shows a list of trusted servers. Besides the name it consists of further attributes concerning about different kind of balances. The second interface is a more detailed look on one server sharing a trust relation. The purpose for this interface is to manage the relation between each other. This means altering the trust level, the limit for the balance, the payout rules and further more. The history interface lists the accomplished server transaction ordered by the most current transaction shown in Figure D.5. Figure D.6 illustrates all user from the system. The listed users are mixed and can be in the role of *User* or *Admin* or *Both*. The last two interfaces are neglected, due to the fact that those interfaces serve only as information pools.

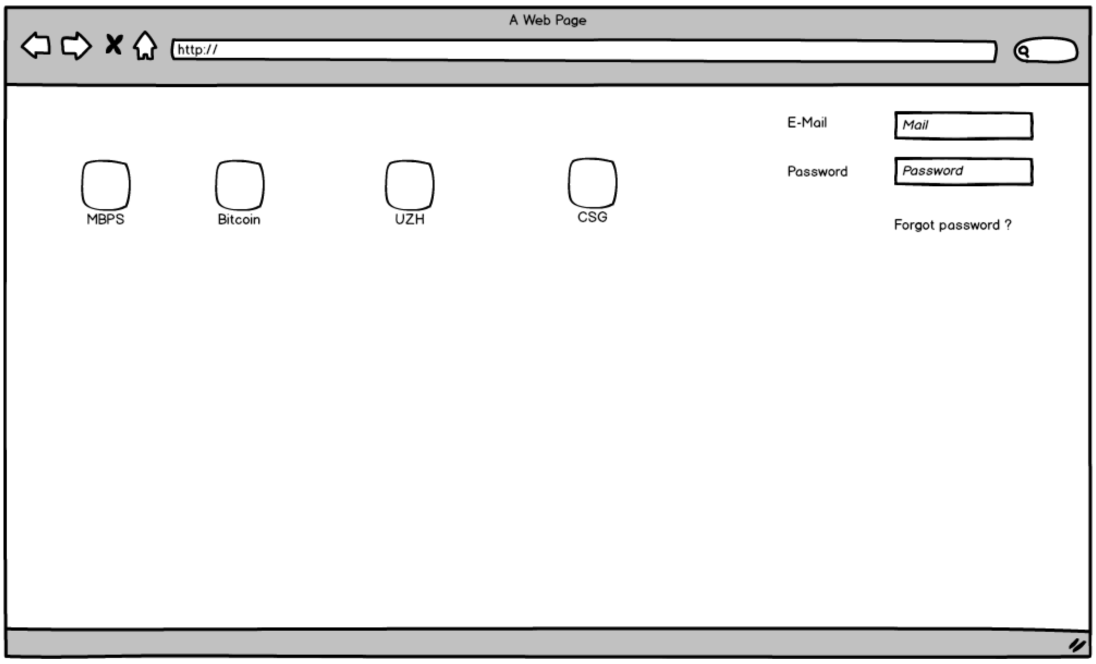


Figure D.1: Mock Login

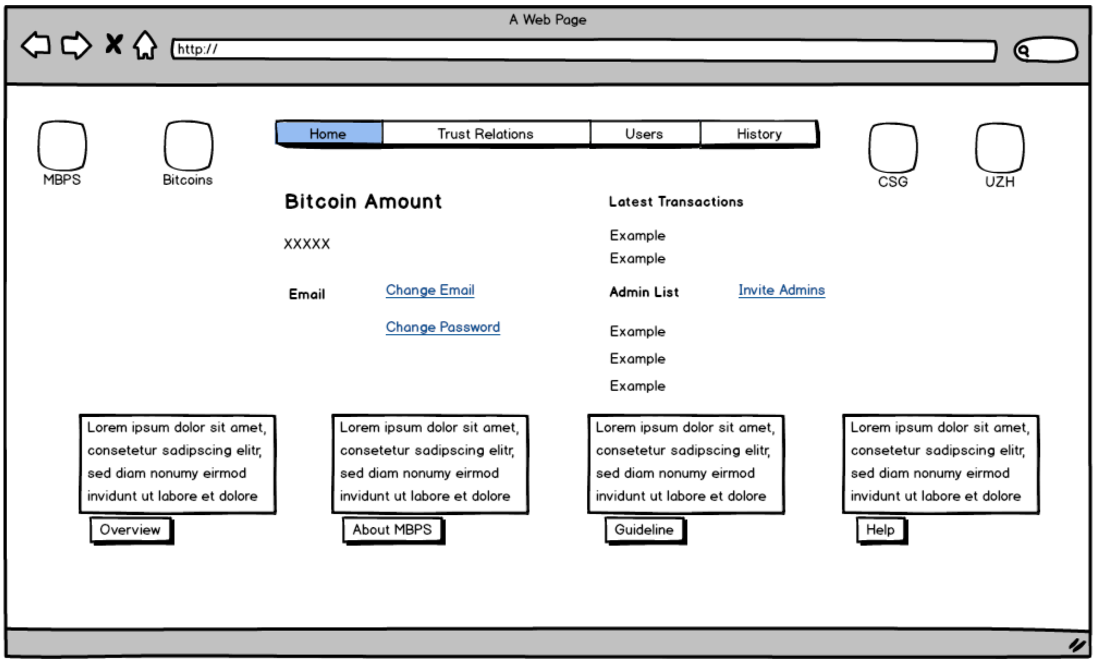
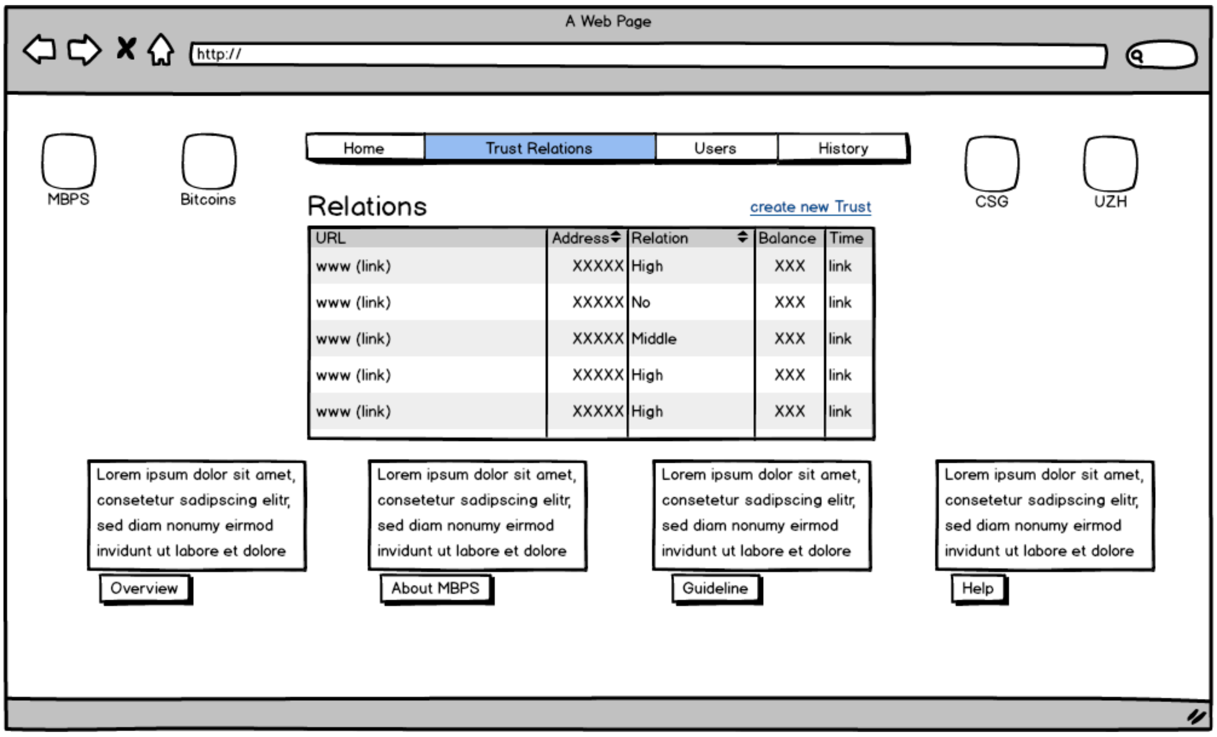

A Web Page
http://
MBPS
Bitcoins
Home
Trust Relations
Users
History
CSG
UZH
Bitcoin Amount
XXXXX
Latest Transactions
Example
Example
Email
Change Email
Change Password
Admin List
Invite Admins
Example
Example
Example
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore
Overview
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore
About MBPS
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore
Guideline
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore
Help

Figure D.2: Mock Home



URL	Address	Relation	Balance	Time
www (link)	XXXXX	High	XXX	link
www (link)	XXXXX	No	XXX	link
www (link)	XXXXX	Middle	XXX	link
www (link)	XXXXX	High	XXX	link
www (link)	XXXXX	High	XXX	link

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Figure D.3: Mock Relation

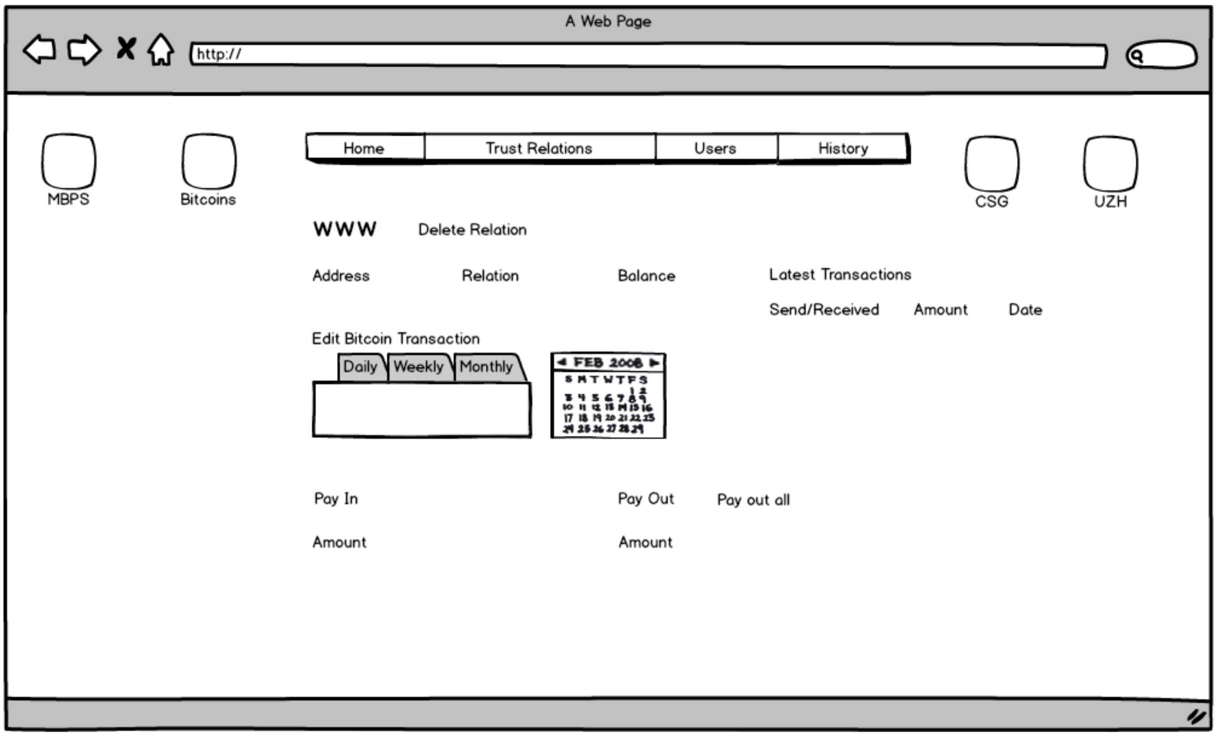
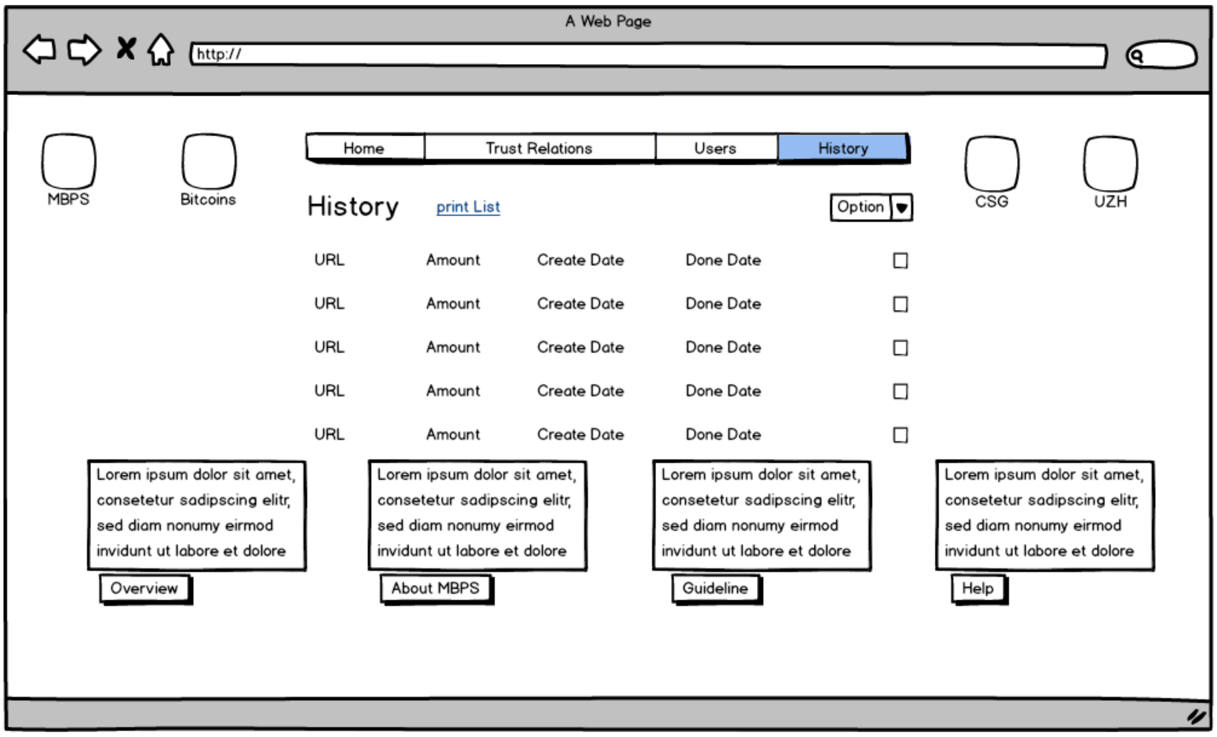


Figure D.4: Mock Edit



URL	Amount	Create Date	Done Date	☐
URL	Amount	Create Date	Done Date	☐
URL	Amount	Create Date	Done Date	☐
URL	Amount	Create Date	Done Date	☐
URL	Amount	Create Date	Done Date	☐

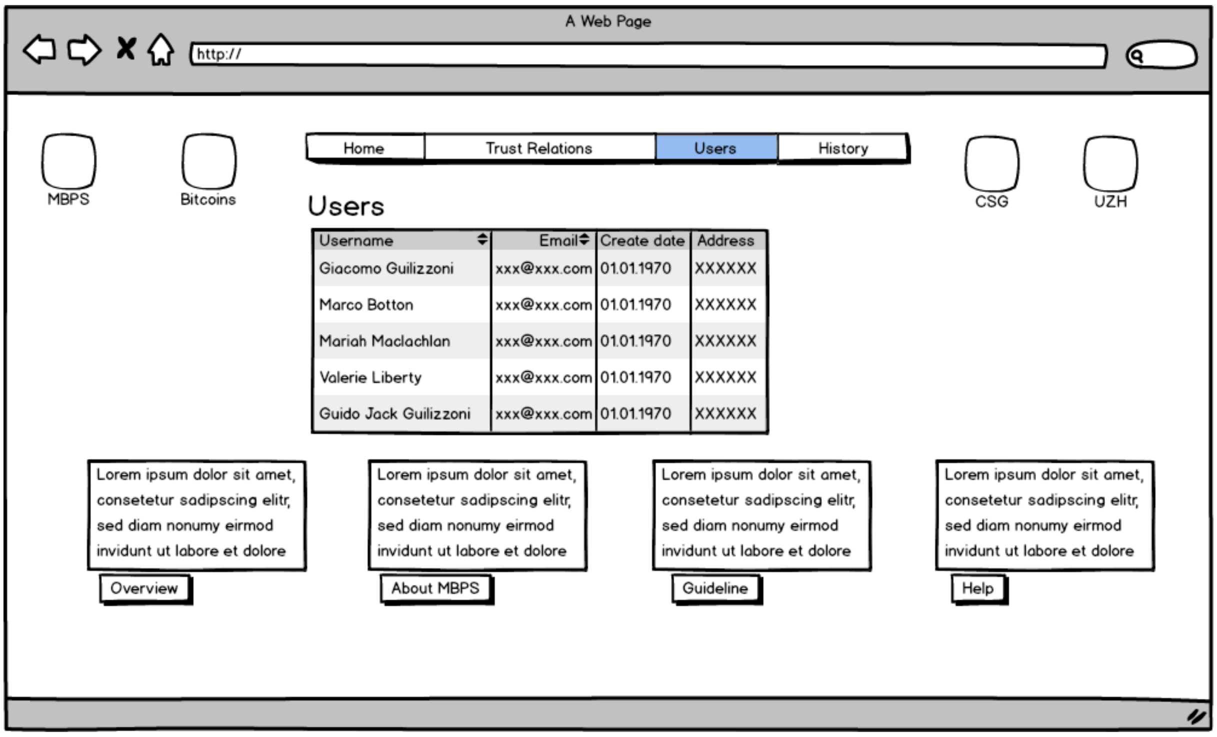
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Figure D.5: Mock History



Username	Email	Create date	Address
Giacomo Guilizzoni	xxx@xxx.com	01.01.1970	XXXXXX
Marco Botton	xxx@xxx.com	01.01.1970	XXXXXX
Mariah Maclachlan	xxx@xxx.com	01.01.1970	XXXXXX
Valerie Liberty	xxx@xxx.com	01.01.1970	XXXXXX
Guido Jack Guilizzoni	xxx@xxx.com	01.01.1970	XXXXXX

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod invidunt ut labore et dolore

Figure D.6: Mock Users

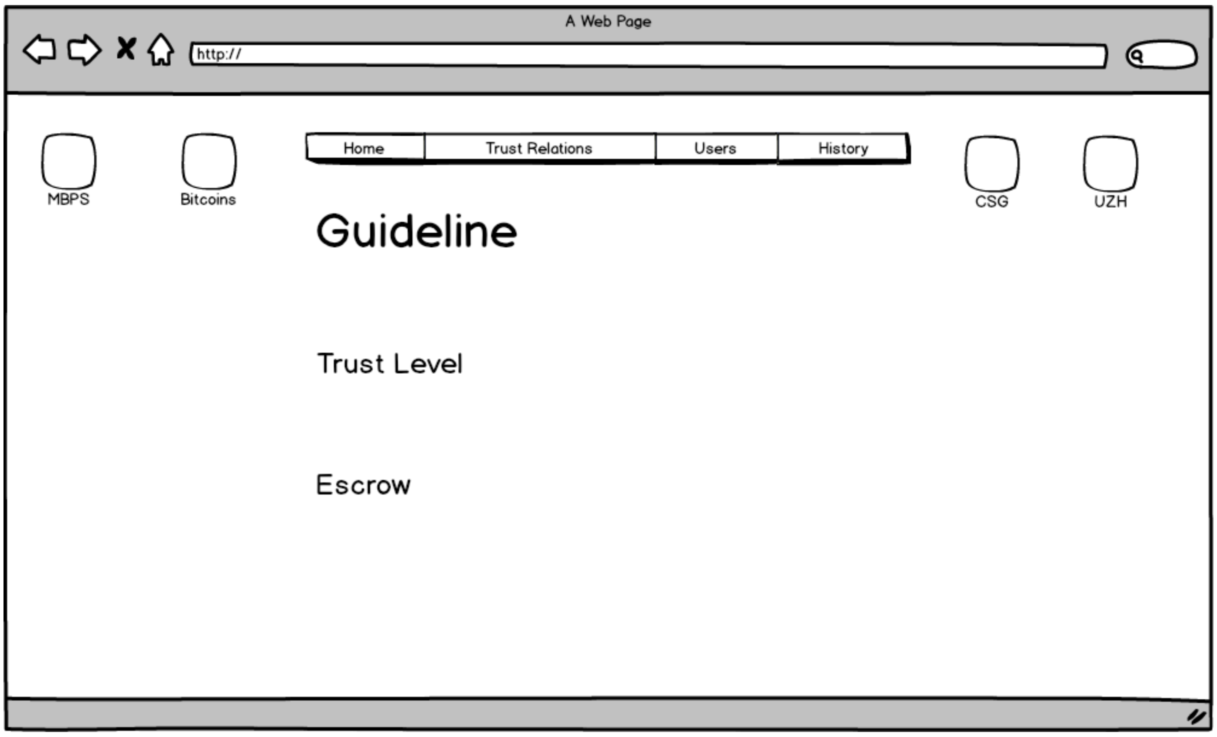


Figure D.7: Mock Overview

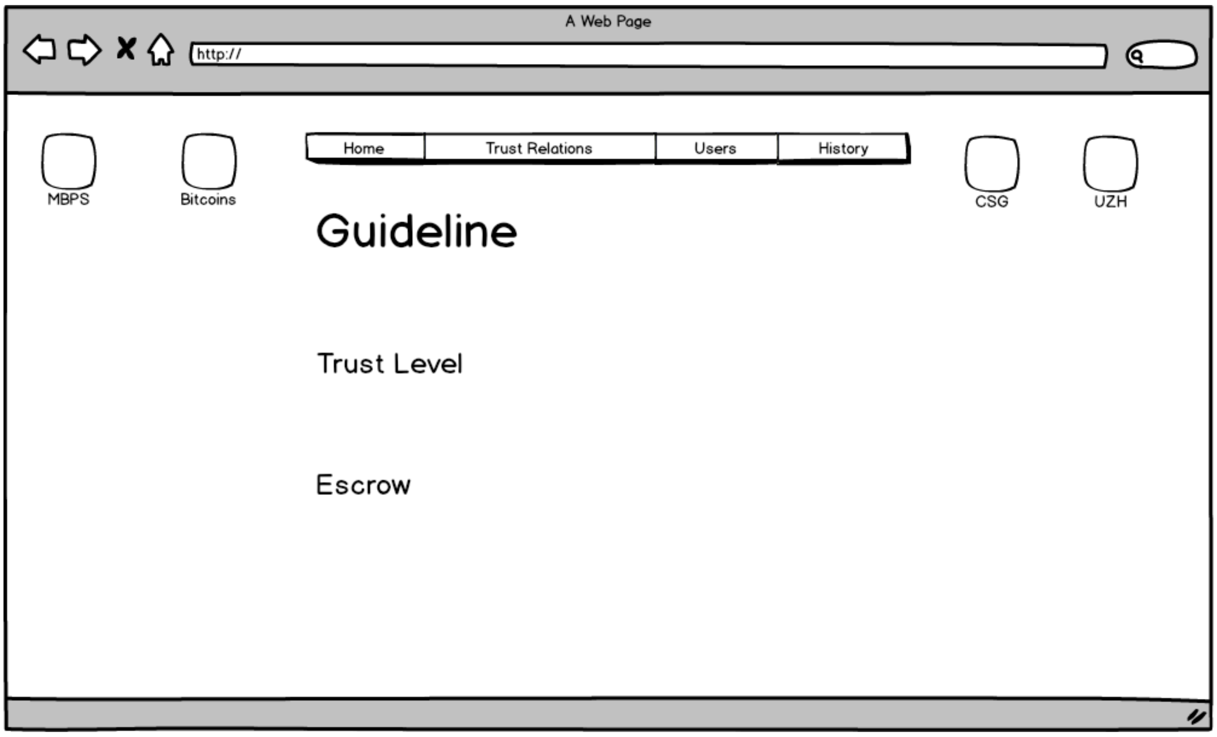


Figure D.8: Mock Guideline

Appendix E

Web-Site Screenshots

This Appendix shows the Screenshots of the web interface. The main page described in Section 4.5.

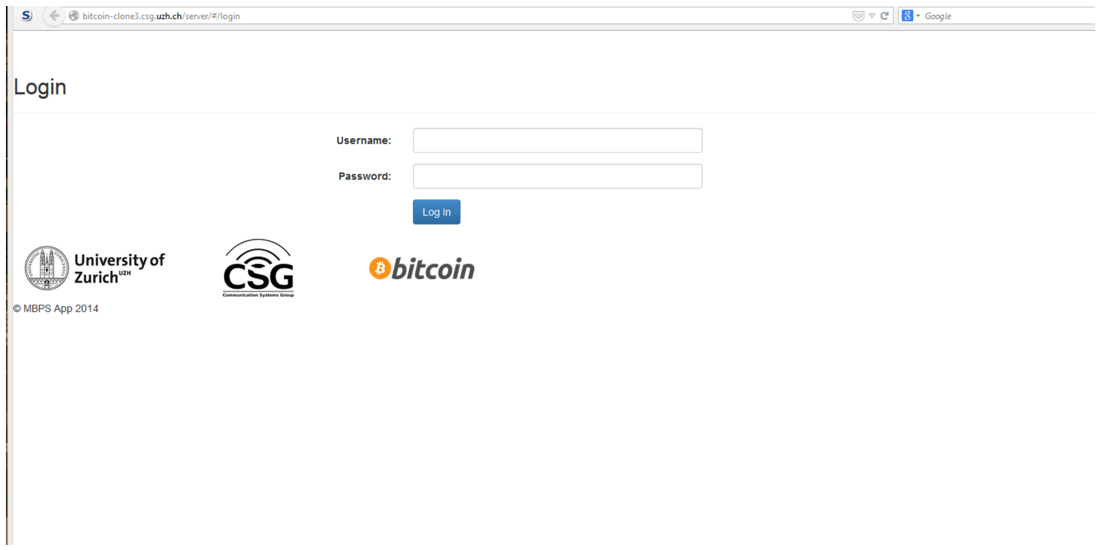
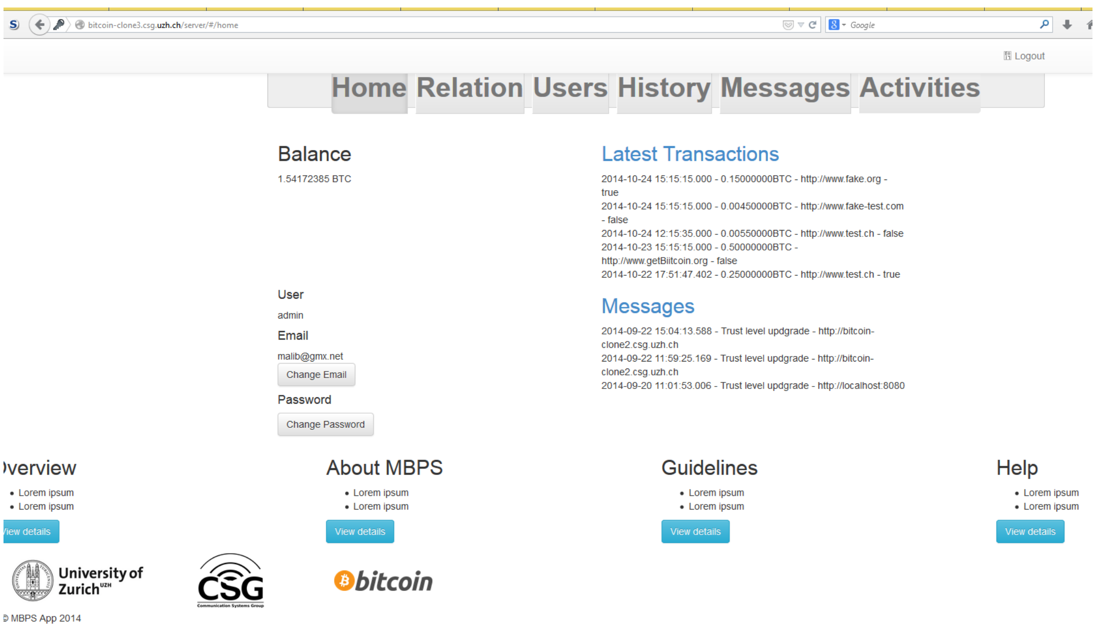


Figure E.1: Web Login



bitcoin-clone3.csg.uzh.ch/server/#/home

Logout

Home Relation Users History Messages Activities

Balance

1.54172385 BTC

User

admin

Email

malib@gmx.net

Change Email

Password

Change Password

Latest Transactions

2014-10-24 15:15:15.000 - 0.15000000BTC - http://www.fake.org - true
2014-10-24 15:15:15.000 - 0.00450000BTC - http://www.fake-test.com - false
2014-10-24 12:15:35.000 - 0.00550000BTC - http://www.test.ch - false
2014-10-23 15:15:15.000 - 0.50000000BTC - http://www.getBiitcoin.org - false
2014-10-22 17:51:47.402 - 0.25000000BTC - http://www.test.ch - true

Messages

2014-09-22 15:04:13.588 - Trust level updgrade - http://bitcoin-clone2.csg.uzh.ch
2014-09-22 11:59:25.169 - Trust level updgrade - http://bitcoin-clone2.csg.uzh.ch
2014-09-20 11:01:53.006 - Trust level updgrade - http://localhost:8080

Overview
- Lorem ipsum
- Lorem ipsum

View details

About MBPS
- Lorem ipsum
- Lorem ipsum

View details

Guidelines
- Lorem ipsum
- Lorem ipsum

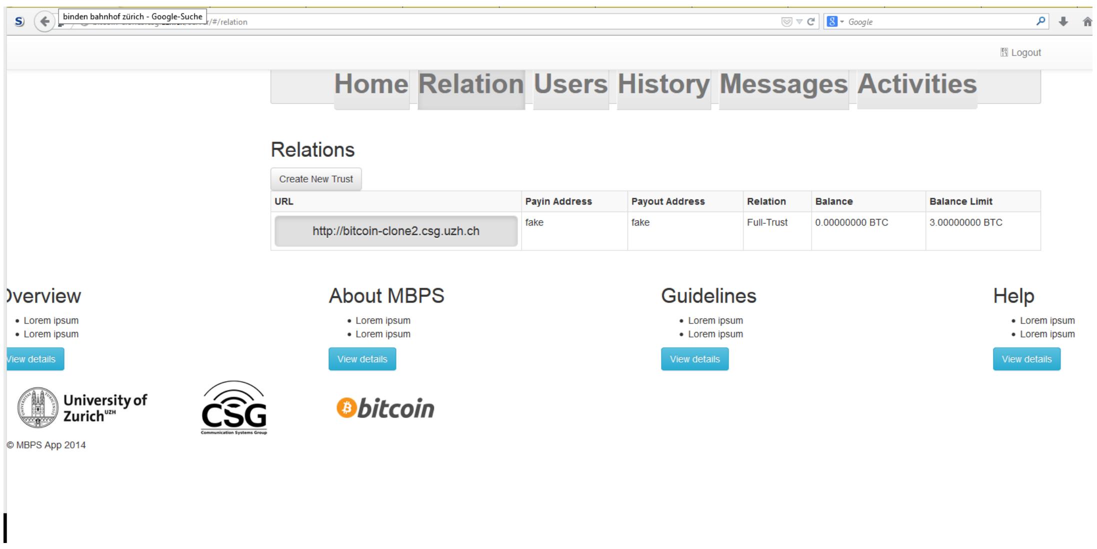
View details

Help
- Lorem ipsum
- Lorem ipsum

View details

© MBPS App 2014

Figure E.2: Web Login



binden bahnhof zürich - Google-Suche

/#/relation

Logout

Home Relation Users History Messages Activities

Relations

Create New Trust

URL	Payin Address	Payout Address	Relation	Balance	Balance Limit
http://bitcoin-clone2.csg.uzh.ch	fake	fake	Full-Trust	0.00000000 BTC	3.00000000 BTC

Overview
- Lorem ipsum
- Lorem ipsum

View details

About MBPS
- Lorem ipsum
- Lorem ipsum

View details

Guidelines
- Lorem ipsum
- Lorem ipsum

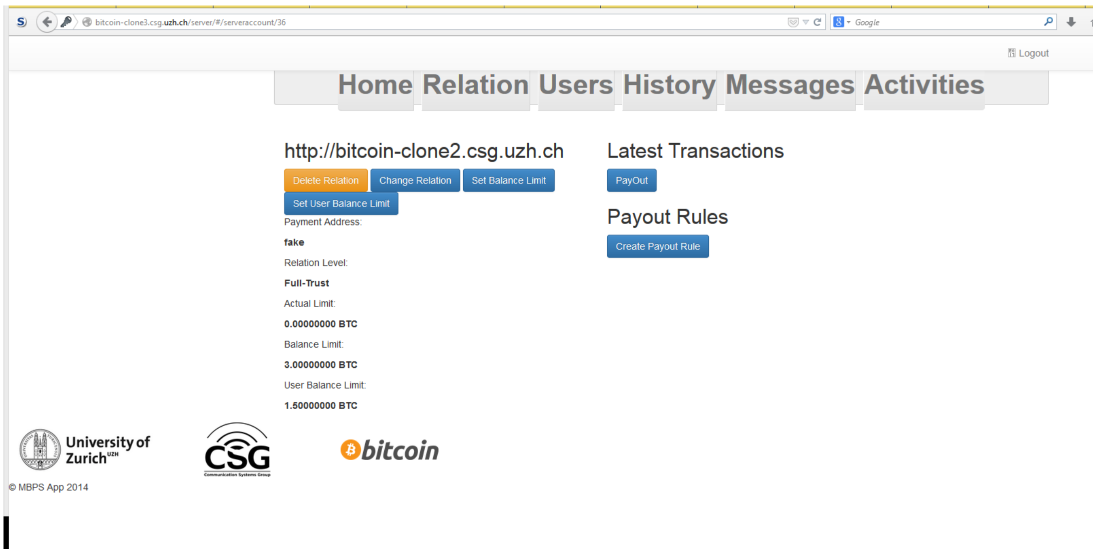
View details

Help
- Lorem ipsum
- Lorem ipsum

View details

© MBPS App 2014

Figure E.3: Web Relation



Home Relation Users History Messages Activities

http://bitcoin-clone2.csg.uzh.ch

Payment Address:

fake

Relation Level:

Full-Trust

Actual Limit:

0.00000000 BTC

Balance Limit:

3.00000000 BTC

User Balance Limit:

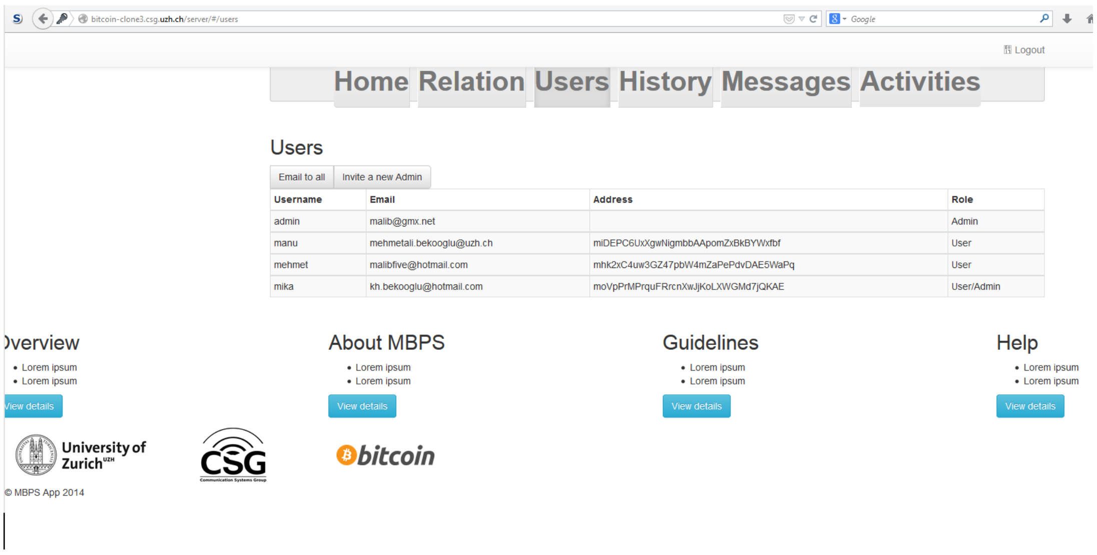
1.50000000 BTC

Latest Transactions

Payout Rules

© MBPS App 2014

Figure E.4: Web Server Acounts



Home Relation Users History Messages Activities

Users

Username	Email	Address	Role
admin	malib@gmx.net		Admin
manu	mehmetali.bekooglu@uzh.ch	miDEPC6UxXgwNigmbbAApomZxBkBYWxfbf	User
mehmet	malibfive@hotmail.com	mhk2xC4uw3GZ47pbW4mZaPePdvDAE5WaPq	User
mika	kh.bekooglu@hotmail.com	moVpPrMPrquFRrcnXwJjKoLXWGMd7jQKAE	User/Admin

Overview
- Lorem ipsum
- Lorem ipsum

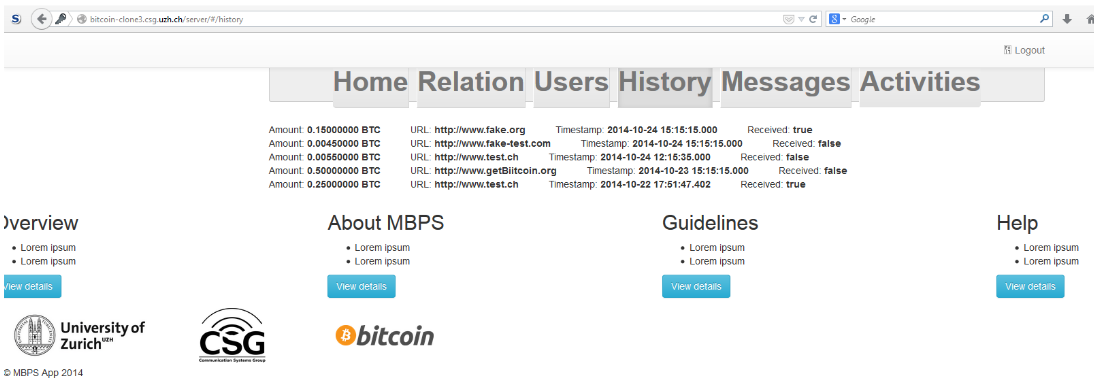
About MBPS
- Lorem ipsum
- Lorem ipsum

Guidelines
- Lorem ipsum
- Lorem ipsum

Help
- Lorem ipsum
- Lorem ipsum

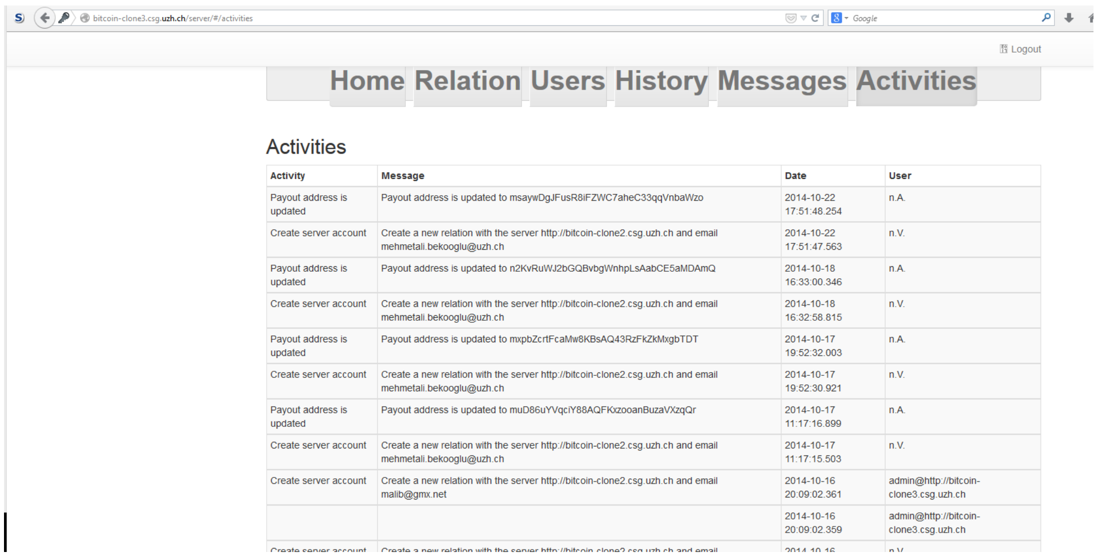
© MBPS App 2014

Figure E.5: Web Users



Amount: **0.15000000 BTC** URL: **http://www.fake.org** Timestamp: **2014-10-24 15:15:15.000** Received: **true**
Amount: **0.00450000 BTC** URL: **http://www.fake-test.com** Timestamp: **2014-10-24 15:15:15.000** Received: **false**
Amount: **0.00550000 BTC** URL: **http://www.test.ch** Timestamp: **2014-10-24 12:15:35.000** Received: **false**
Amount: **0.50000000 BTC** URL: **http://www.getBiitcoin.org** Timestamp: **2014-10-23 15:15:15.000** Received: **false**
Amount: **0.25000000 BTC** URL: **http://www.test.ch** Timestamp: **2014-10-22 17:51:47.402** Received: **true**

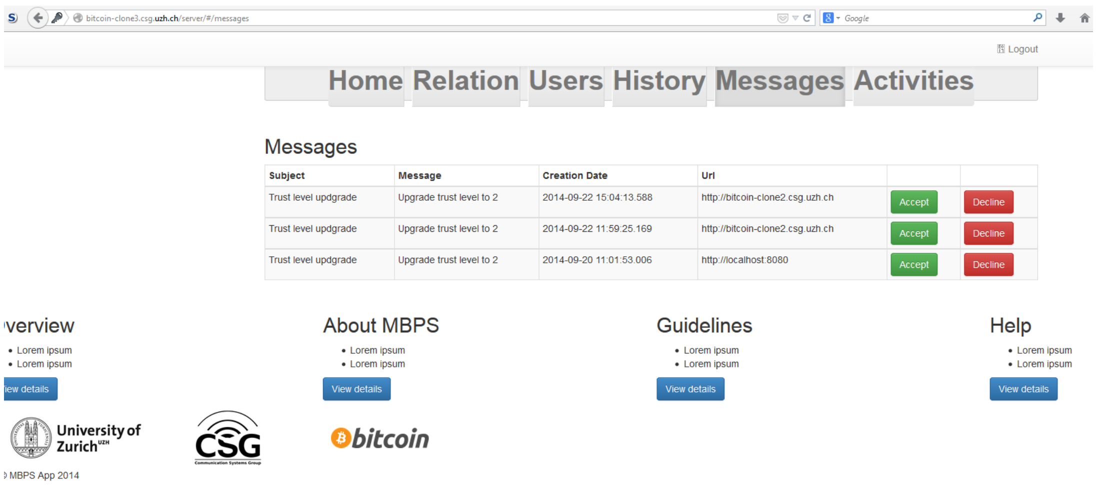
Figure E.6: Web History



Activities

Activity	Message	Date	User
Payout address is updated	Payout address is updated to msaywDgJFusR8iFZWC7aheC33qqVnbaWzo	2014-10-22 17:51:48.254	n.A.
Create server account	Create a new relation with the server http://bitcoin-clone2.csg.uzh.ch and email mehmetali.bekooglu@uzh.ch	2014-10-22 17:51:47.563	n.V.
Payout address is updated	Payout address is updated to n2KvRuWJ2bGQBvbgWnhpLsAabCE5aMDAmQ	2014-10-18 16:33:00.346	n.A.
Create server account	Create a new relation with the server http://bitcoin-clone2.csg.uzh.ch and email mehmetali.bekooglu@uzh.ch	2014-10-18 16:32:58.815	n.V.
Payout address is updated	Payout address is updated to mxpbZcrtFcaMw8KBsAQ43RzFkZkMxgbTDT	2014-10-17 19:52:32.003	n.A.
Create server account	Create a new relation with the server http://bitcoin-clone2.csg.uzh.ch and email mehmetali.bekooglu@uzh.ch	2014-10-17 19:52:30.921	n.V.
Payout address is updated	Payout address is updated to muD86uYVqciY88AQFKxzooanBuzaVXzqQr	2014-10-17 11:17:16.899	n.A.
Create server account	Create a new relation with the server http://bitcoin-clone2.csg.uzh.ch and email mehmetali.bekooglu@uzh.ch	2014-10-17 11:17:15.503	n.V.
Create server account	Create a new relation with the server http://bitcoin-clone2.csg.uzh.ch and email malib@gmx.net	2014-10-16 20:09:02.361	admin@http://bitcoin-clone3.csg.uzh.ch
		2014-10-16 20:09:02.359	admin@http://bitcoin-clone3.csg.uzh.ch
Create server account	Create a new relation with the server http://bitcoin-clone2.csg.uzh.ch and email	2014-10-16	n.V.

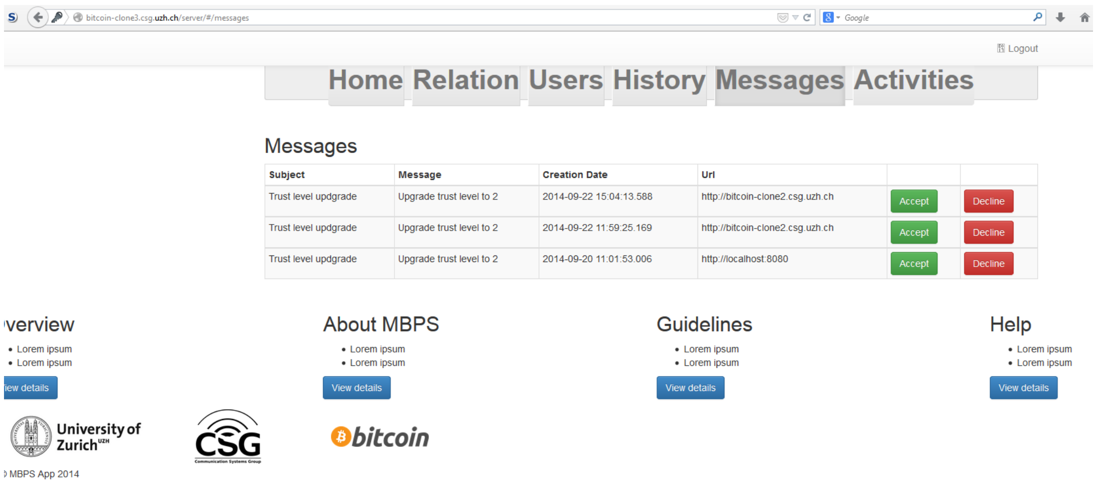
Figure E.7: Web Activities



Messages

Subject	Message	Creation Date	Url		
Trust level updgrade	Upgrade trust level to 2	2014-09-22 15:04:13.588	http://bitcoin-clone2.csg.uzh.ch	Accept	Decline
Trust level updgrade	Upgrade trust level to 2	2014-09-22 11:59:25.169	http://bitcoin-clone2.csg.uzh.ch	Accept	Decline
Trust level updgrade	Upgrade trust level to 2	2014-09-20 11:01:53.006	http://localhost:8080	Accept	Decline

Figure E.8: Web Messages



Messages

Subject	Message	Creation Date	Url		
Trust level updgrade	Upgrade trust level to 2	2014-09-22 15:04:13.588	http://bitcoin-clone2.csg.uzh.ch	Accept	Decline
Trust level updgrade	Upgrade trust level to 2	2014-09-22 11:59:25.169	http://bitcoin-clone2.csg.uzh.ch	Accept	Decline
Trust level updgrade	Upgrade trust level to 2	2014-09-20 11:01:53.006	http://localhost:8080	Accept	Decline

Figure E.9: Web Messages

Appendix F

Sequence Diagram

This appendix shows the sequence diagrams which have been modelled during the analysis. The Full-Trust as Initiator as payee and the create Account are discussed in Chapter 5 Section 5.2.

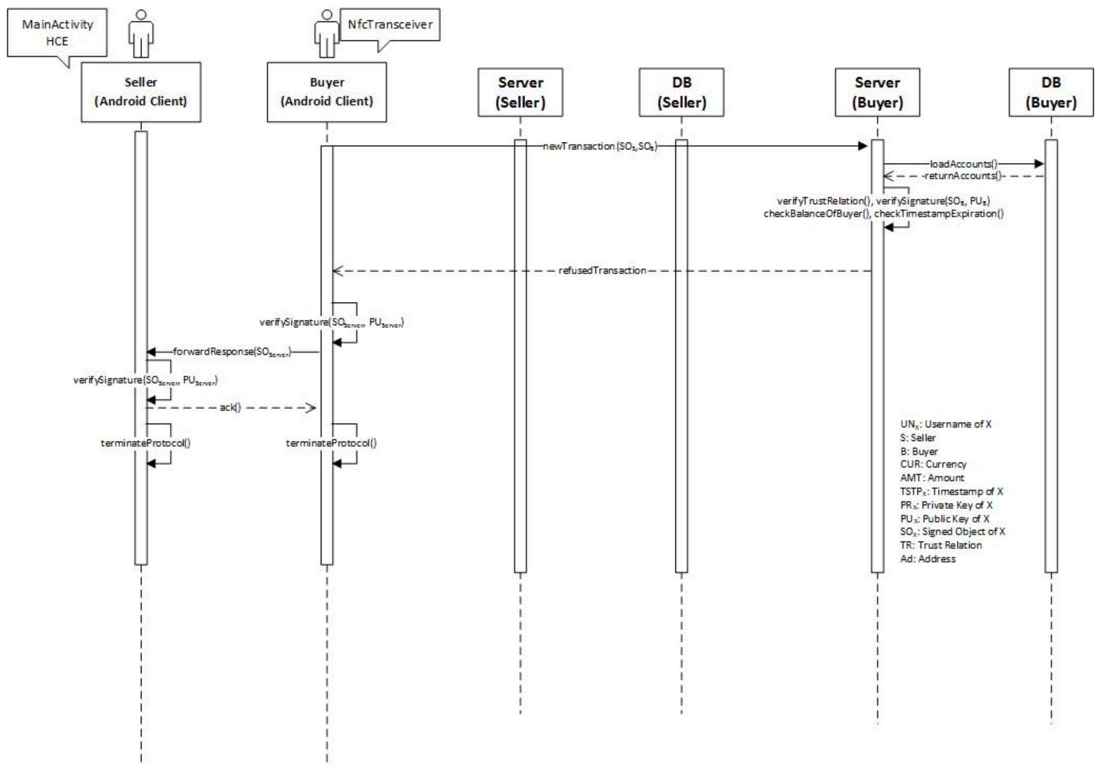


Figure F.1: No-Trust Buyer as Initiator

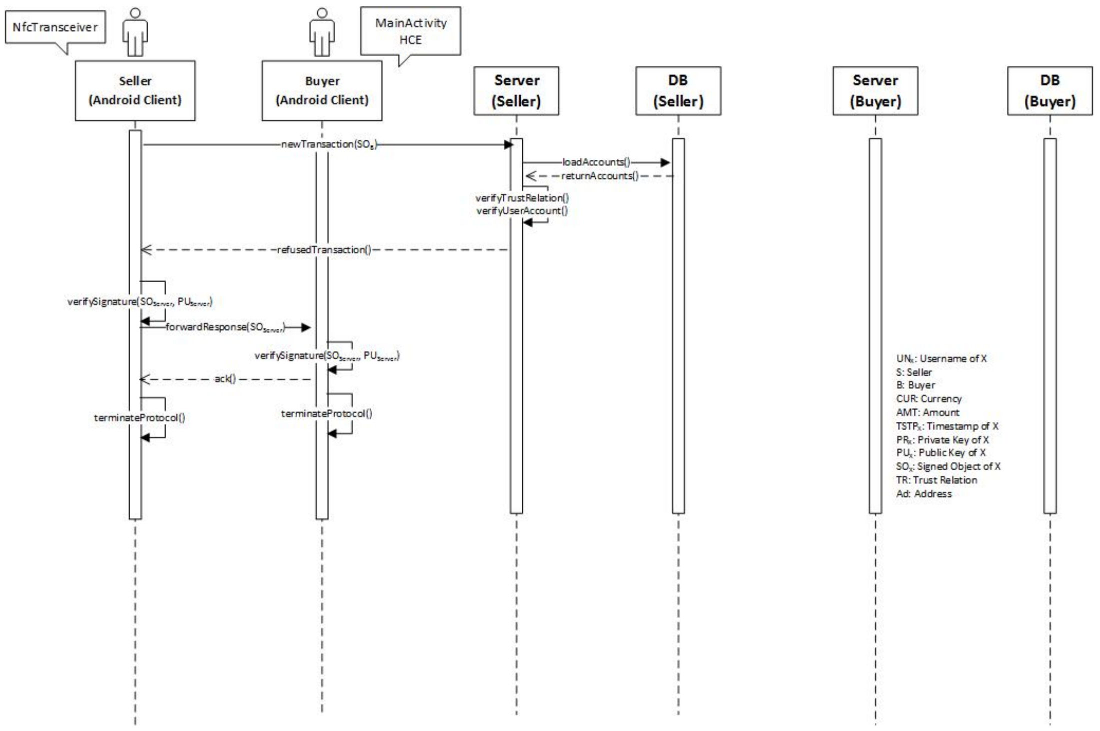


Figure F.2: No-Trust Seller as Initiator

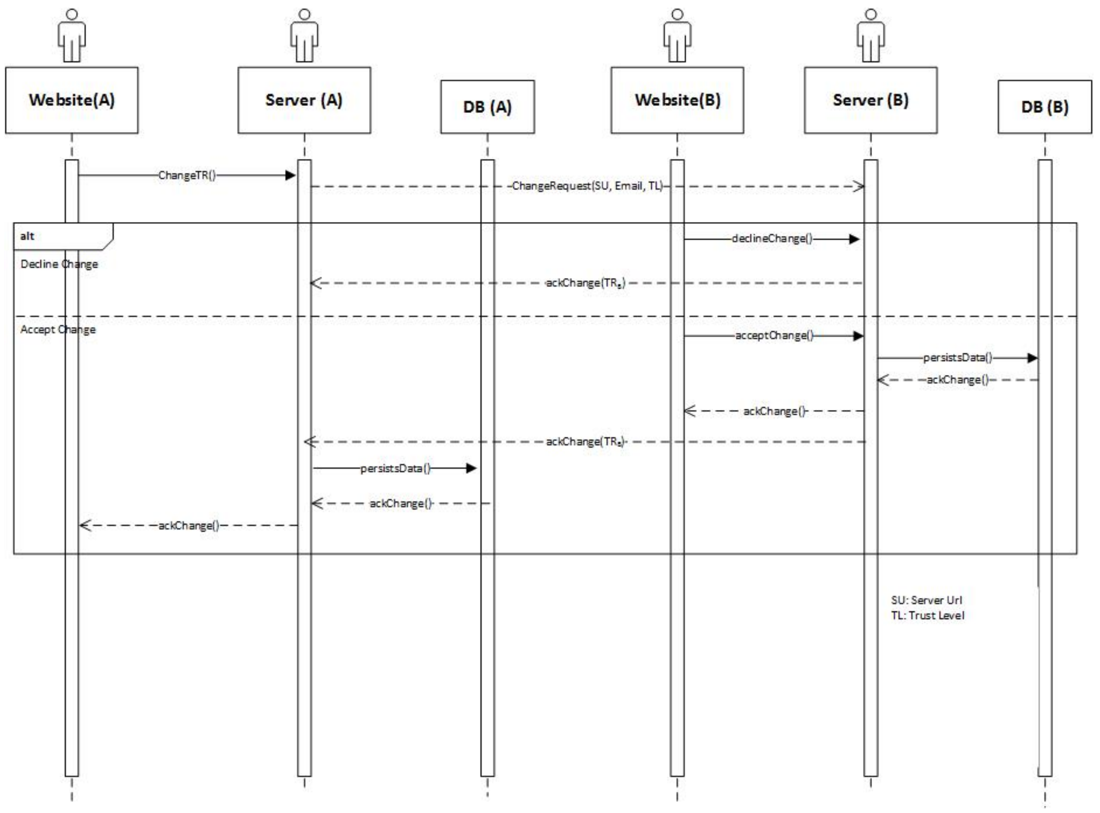


Figure F.3: Upgrade Trust Level

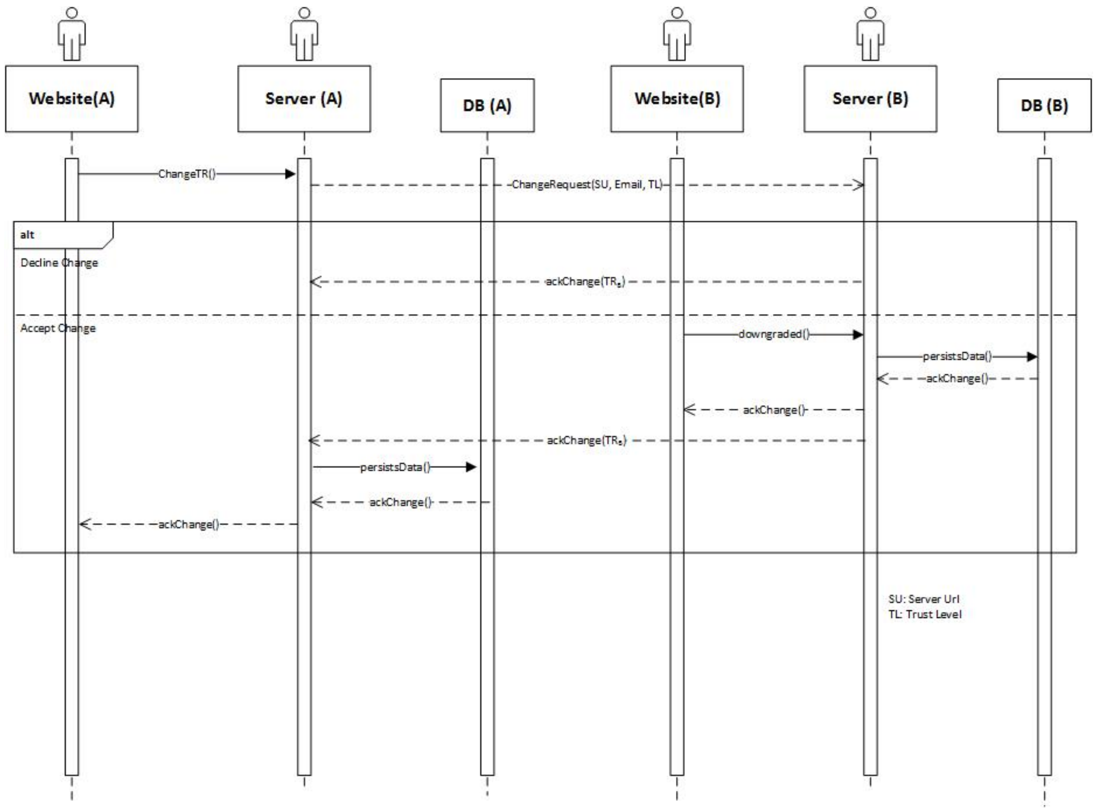


Figure F.4: Downgrade Trust Level

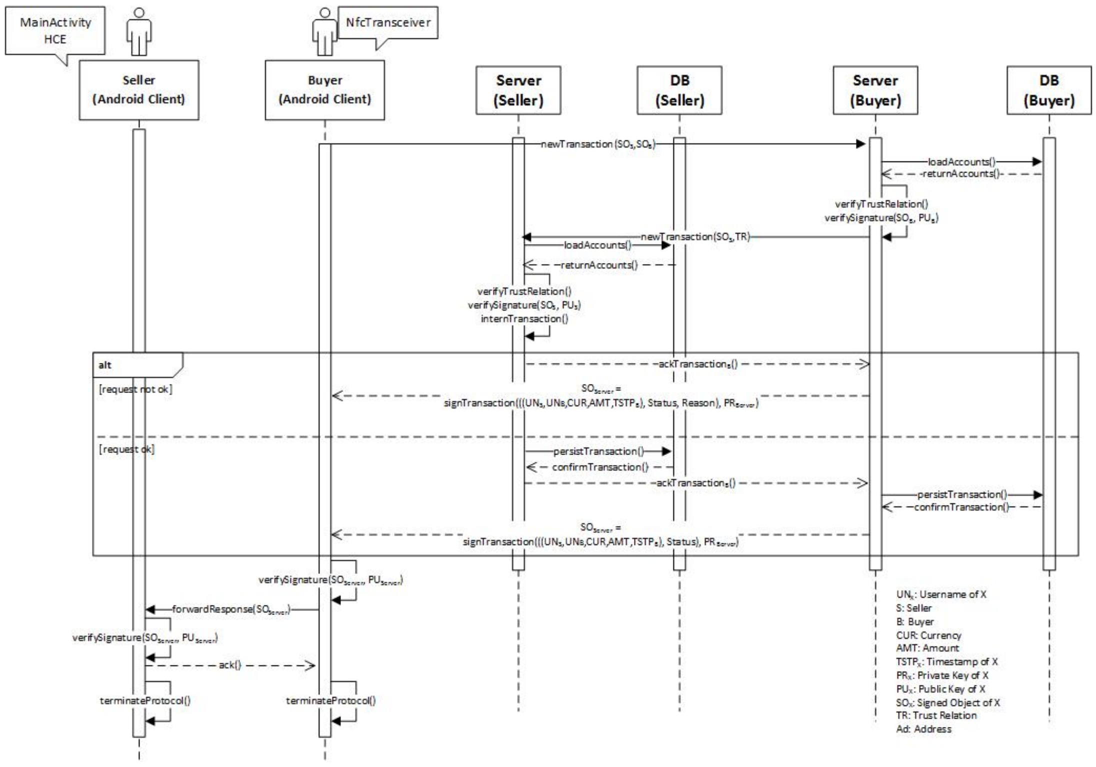


Figure F.5: Full-Trust Cross-Server NFC Transactions (Buyer)

Appendix G

Performance Measurement Results

This Appendix contains the performance measurements of the NFC payment with and without cross-server payment. A detailed analysis is provided in Chapter 6.

The tables in this Appendix shows the performance measurement for a NFC payment between two devices. The performance measurement are split into steps. The steps between a NFC payment of two user from the same server and the NFC payment between two users from trust related servers differ in their steps. The values in all tables are indicated in milliseconds. The measurements have been repeated 20 times for each role payer and payee. The repetitions are represented by the columns 1 to 20. The last two columns of every table shows the arithmetic average and the standard deviation of the given step. The last row of each table contains the total time required for a NFC payment to complete. The sum of all steps are not the total time required for a payment. Minor manipulations have been not considered.
A second evaluation has been conducted. Only the server site of the payment over NFC has been addressed. The Section G.4 and G.5 shows the results of the evaluation. For each device the role has been repeated 10 times.
The measurements have been conducted with the devices Google Nexus 7 with a Broadcom NFC chip and LG G3 with a NXP NFC chip. Section G.1 contains the performance measurements for the NFC payments from users of the same server. Section G.2 contains the performance measurements for the NFC payments from users of different servers.

G.1 Results Same Server

Tables G.1 and G.2 show the performance measurement for the Nexus 7 in the payee role. The next Tables G.3 and G.4 show the performance measurement for the Nexus 7 in the payer role. The same for the Tables G.5 and G.6 in payee role and tables G.7 and G.8 in payer role. The last four tables have been measured with the LG G3.

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
NFC Exchange	361	335	382	416	445	394	420	376	321	363	371.90	9.90
Server Call	256	424	281	352	288	334	438	246	354	300	311.75	23.33
Server Process	169	158	163	249	167	167	256	141	224	178	176.35	22.63
NFC Notification	91	135	107	87	93	167	98	154	129	133	120.95	11.31
Total Time	709	897	776	859	827	897	959	778	807	798	807.45	46.67

Table G.1: Performance Measurements Payment Nexus 7 as Payee (1-10)

Step	11	12	13	14	15	16	17	18	19	20	Mean	Std. Dev.
NFC Exchange	303	318	445	354	328	346	394	391	371	375	371.90	9.90
Server Call	250	251	327	303	325	347	294	277	299	289	311.75	23.33
Server Process	132	135	212	195	197	216	149	138	144	137	176.35	22.63
NFC Notification	93	151	133	103	151	100	117	103	167	107	120.95	11.31
Total Time	709	897	776	859	827	897	959	778	807	798	807.45	46.67

Table G.2: Performance Measurements Payment Nexus 7 as Payee (11-20)

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
NFC Exchange	272	166	214	225	321	253	160	171	214	214	252.00	41.17
Server Call	323	230	232	234	286	248	213	264	256	279	289.50	26.42
Server Process	217	125	123	131	199	132	124	159	124	167	161.00	28.78
NFC Notification	221	208	151	211	174	182	165	170	167	200	177.90	35.36
Total Time	818	606	599	673	782	685	539	607	639	695	744.00	63.37

Table G.3: Performance Measurements Payment Nexus 7 as Payer (1-10)

Step	11	12	13	14	15	16	17	18	19	20	Mean	Std. Dev.
NFC Exchange	271	235	161	265	221	219	200	212	204	232	252.00	41.17
Server Call	227	271	268	280	284	245	286	278	268	256	289.50	26.42
Server Process	124	151	158	162	157	132	171	150	109	105	161.00	28.78
NFC Notification	165	175	172	177	164	189	171	216	175	180	177.90	35.36
Total Time	665	683	602	724	670	654	659	708	648	670	744.00	63.37

Table G.4: Performance Measurements Payment Nexus 7 as Payer (11-20)

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
NFC Exchange	405	387	483	448	462	408	457	467	317	271	423.80	33.94
Server Call	339	333	363	357	211	370	363	360	364	476	368.50	67.18
Server Process	162	166	152	157	76	216	167	169	168	237	158.85	15.56
NFC Notification	124	164	163	176	160	213	146	173	208	182	200.50	19.48
Total Time	875	888	1016	989	836	994	973	1006	893	936	976.35	134.35

Table G.5: Performance Measurements Payment LG G3 as Payee (1-10)

Step	11	12	13	14	15	16	17	18	19	20	Mean	Std. Dev.
NFC Exchange	455	412	473	295	408	493	446	475	461	453	423.80	33.94
Server Call	260	316	263	391	415	516	428	408	403	434	368.50	67.18
Server Process	60	152	152	227	146	192	140	149	149	140	158.85	15.56
NFC Notification	149	257	161	199	151	178	182	221	177	174	200.50	19.48
Total Time	869	994	899	903	979	1197	1062	1108	1045	1065	976.35	134.35

Table G.6: Performance Measurements Payment LG G3 as Payee (11-20)

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
NFC Exchange	239	395	350	331	356	325	380	364	363	298	302.00	89.10
Server Call	300	254	343	227	248	213	363	268	242	296	297.50	3.54
Server Process	101	130	140	124	129	116	147	123	108	143	127.50	37.48
NFC Notification	229	281	303	167	239	190	244	201	272	232	243.00	19.80
Total Time	771	936	1002	727	847	730	996	836	884	832	845.00	104.65

Table G.7: Performance Measurements Payment LG G3 as Payer (1-10)

Step	11	12	13	14	15	16	17	18	19	20	Mean	Std. Dev.
NFC Exchange	363	392	374	348	393	384	363	261	372	365	302.00	89.10
Server Call	296	307	306	296	281	236	297	370	352	295	297.50	3.54
Server Process	140	138	129	116	113	117	106	124	140	154	127.50	37.48
NFC Notification	184	346	331	279	228	326	299	294	282	257	243.00	19.80
Total Time	846	1047	1015	925	917	960	963	928	1010	919	845.00	104.65

Table G.8: Performance Measurements Payment LG G3 as Payer (11-20)

G.2 Results Different Server

The Tables G.9, G.10, G.12 and G.12 have been conducted by the Nexus 7. The first two Tables G.9 and G.10 show the performance measurement in the payee role. Tables G.11 and G.12 show the performance measurement in the payer role. The next Tables G.13, G.14, G.15 and G.16 show the performance measurement conducted with the LG G3. The Tables G.13 and G.14 have been in the payee role and the Tables G.15 and G.16 have been in the payer role.

G.3 Results Atomic Steps One Server

This are the atomic steps of a payment on the server. The evaluation has been tested for the payee and payer. But the scenario for the test was a NFC payment on the same server. This means both users are registered on the same server.

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
NFC Exchange	389	280	419	355	353	301	358	292	432	418	339,00	70,71
Server Call	774	829	774	731	389	480	752	698	355	657	699,00	106,07
Server Process	595	599	640	532	277	352	197	419	252	469	477,50	166,17
Trust Relation Process	366	437	568	338	210	252	143	289	128	341	304,00	87,68
Trusted Server Call	292	374	150	249	183	216	109	188	99	291	246,50	64,35
Trusted Server Process	246	321	118	202	167	196	92	158	85	247	205,50	57,28
NFC Notification	144	103	154	97	122	100	97	95	138	103	153	12.7
Total Time	1309	1214	1349	1184	866	884	1210	1087	927	1180	1193,00	164,05

Table G.9: Performance Measurements Nexus 7 as Payee (1-10)

Step	11	12	13	14	15	16	17	18	19	20	Mean	Std. Dev.
NFC Exchange	374	366	378	435	460	372	354	357	427	289	339,00	70,71
Server Call	634	770	678	693	791	799	720	605	727	624	699,00	106,07
Server Process	488	586	450	467	545	546	462	411	455	360	477,50	166,17
Trust Relation Process	312	471	281	325	411	393	309	276	333	242	304,00	87,68
Trusted Server Call	255	408	233	248	341	311	259	185	269	201	246,50	64,35
Trusted Server Process	210	364	193	202	305	279	221	160	219	165	205,50	57,28
NFC Notification	102	102	132	142	103	111	142	131	140	162	153	12.7
Total Time	1111	1239	1190	1271	1356	1285	1218	1098	1298	1077	1193,00	164,05

Table G.10: Performance Measurements Nexus 7 as Payee (11-20)

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
NFC Exchange	264	151	253	164	214	207	214	261	357	231	228,90	31,82
Server Call	757	703	585	634	460	643	670	750	474	500	593,95	70,71
Server Process	637	387	390	412	352	387	458	438	345	347	400,45	150,61
Trust Relation Process	445	228	259	298	248	272	311	225	198	221	267,35	88,39
Trusted Server Call	344	174	180	250	201	204	238	167	151	168	210,90	95,46
Trusted Server Process	237	144	140	222	152	171	203	137	125	136	175,85	42,43
NFC Notification	225	180	226	174	185	176	312	183	160	163	189	12
Total Time	1247	1036	1066	974	867	1032	1198	1195	992	895	1016,95	144,96

Table G.11: Performance Measurements Nexus 7 as Payer (1-10)

Step	11	12	13	14	15	16	17	18	19	20	Mean	Std. Dev.
NFC Exchange	242	158	238	212	225	297	236	247	188	219	228,90	31,82
Server Call	445	607	544	572	725	616	568	335	634	657	593,95	70,71
Server Process	315	419	388	397	514	381	398	181	439	424	400,45	150,61
Trust Relation Process	192	290	279	205	402	255	243	119	337	320	267,35	88,39
Trusted Server Call	147	245	238	151	361	204	195	97	294	209	210,90	95,46
Trusted Server Process	121	217	211	128	334	167	164	74	257	177	175,85	42,43
NFC Notification	186	161	173	213	180	175	221	176	205	165	189	12
Total Time	875	928	957	999	1132	1089	1027	759	1029	1042	1016,95	144,96

Table G.12: Performance Measurements Nexus 7 as Payer (11-20)

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
NFC Exchange	557	453	421	479	321	489	456	403	451	444	505,00	73,54
Server Call	817	699	689	742	638	634	775	781	559	659	759,00	82,02
Server Process	520	423	482	400	428	433	436	492	420	504	469,00	72,12
Trust Relation Process	425	242	318	261	289	285	277	274	273	331	349,50	106,77
Trusted Server Call	386	192	212	207	235	225	211	217	216	255	304,00	115,97
Trusted Server Process	363	154	173	174	190	181	174	173	178	212	269,00	132,94
NFC Notification	197	175	180	190	172	119	163	209	175	182	191.95	42.43
Total Time	1580	1330	1292	1414	1142	1246	1411	1396	1188	1289	1460,00	169,71

Table G.13: Performance Measurements LG3 as Payee (1-10)

Step	11	12	13	14	15	16	17	18	19	20	Mean	Std. Dev.
NFC Exchange	392	471	456	430	480	284	406	413	292	453	505,00	73,54
Server Call	695	833	596	780	793	757	771	619	650	701	759,00	82,02
Server Process	452	537	420	516	468	445	448	374	393	418	469,00	72,12
Trust Relation Process	305	373	272	373	263	303	288	222	254	274	349,50	106,77
Trusted Server Call	248	257	218	321	210	250	214	167	202	222	304,00	115,97
Trusted Server Process	199	192	170	174	167	206	184	137	170	175	269,00	132,94
NFC Notification	181	165	181	179	210	255	189	161	185	180	191.95	42.43
Total Time	1272	1471	1238	1403	1500	1299	1372	1196	1132	1340	1460,00	169,71

Table G.14: Performance Measurements LG3 as Payee (11-20)

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
NFC Exchange	324	408	143	335	363	375	0	321	201	253	345,00	29,70
Server Call	617	718	471	605	623	501	583	582	625	612	562,50	77,07
Server Process	418	537	383	460	392	289	410	405	528	419	377,50	57,28
Trust Relation Process	269	350	244	269	245	155	271	268	387	265	245,50	33,23
Trusted Server Call	214	263	189	210	186	111	216	211	303	203	183,00	43,84
Trusted Server Process	169	197	156	167	151	94	177	167	251	158	143,00	36,77
NFC Notification	221	179	245	179	286	303	176	285	173	326	245.5	34.65
Total Time	1172	1308	860	1126	1282	1182	761	1190	1000	1194	1159,00	18,38

Table G.15: Performance Measurements LG3 as Payer (1-10)

Step	11	12	13	14	15	16	17	18	19	20	Mean	Std. Dev.
NFC Exchange	262	119	157	374	374	278	374	280	362	366	345,00	29,70
Server Call	703	484	781	547	370	729	619	596	609	508	562,50	77,07
Server Process	515	394	208	260	166	434	388	368	375	337	377,50	57,28
Trust Relation Process	324	258	133	154	112	312	250	240	217	222	245,50	33,23
Trusted Server Call	266	193	99	115	0	261	197	180	170	152	183,00	43,84
Trusted Server Call	266	193	99	115	0	261	197	180	170	152	183,00	43,84
Trusted Server Process	219	169	83	102	87	219	156	153	145	117	143,00	36,77
NFC Notification	263	134	123	344	313	349	287	393	279	270	245.5	34.65
Total Time	1232	738	1070	1269	1066	1373	1292	1279	1254	1146	1159,00	18,38

Table G.16: Performance Measurements LG3 as Payer (11-20)

Step	11	12	13	14	15	16	17	18	19	20	Mean	Std. Dev.
NFC Exchange	262	119	157	374	374	278	374	280	362	366	345,00	29,70
Server Call	703	484	781	547	370	729	619	596	609	508	562,50	77,07
Server Process	515	394	208	260	166	434	388	368	375	337	377,50	57,28
Trust Relation Process	324	258	133	154	112	312	250	240	217	222	245,50	33,23
Trusted Server Call	266	193	99	115	0	261	197	180	170	152	183,00	43,84
Trusted Server Call	266	193	99	115	0	261	197	180	170	152	183,00	43,84
Trusted Server Process	219	169	83	102	87	219	156	153	145	117	143,00	36,77
Total Time	1232	738	1070	1269	1066	1373	1292	1279	1254	1146	1159,00	18,38

Table G.17: Performance Measurements Server Nexus 7 Payee

G.4 Results Atomic Steps One Server

This are the result for a cross-server payment. Only the steps for the server has been evaluated. In this scenario two server are involved in the payment.

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Get Users	9	9	9	14	12	11	9	9	9	8	10	1.9
Verify Signed User	42	37	58	64	56	48	32	40	38	36	45	10.8
Persists DB	33	32	27	27	47	36	27	29	29	32	33	6.1
Create Payment Response	24	12	12	12	22	13	11	14	13	54	19	13.2
Server Process	155	123	147	156	167	148	112	134	122	182	155	21.9

Table G.18: Performance Measurements Nexus 7 Same Server Payee

G.5 Results Atomic Steps Two Server

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Get Users	10	10	8	9	11	18	9	7	10	13	11	3.8
Verify Signed Users	22	21	19	23	19	44	57	14	22	28	27	13.4
Persists DB	38	29	26	44	29	36	38	25	37	48	34	19.6
Create Payment Response	15	12	11	11	12	22	25	14	16	10	15	4.2
Server Process	123	108	111	120	106	156	169	113	132	140	128	30.2

Table G.19: Performance Measurements Nexus 7 Same Server Payer

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Get Users	10	11	12	8	9	8	8	9	17	11	10	2.8
Verify Signed Users	20	20	77	17	18	17	18	20	34	20	26	18.6
Persists DB	33	36	50	27	27	27	27	30	53	30	34	9.7
Create Payment Response	13	12	11	11	11	11	11	12	13	12	12	0.8
Server Process	114	121	190	112	105	112	105	125	160	125	127	27.3

Table G.20: Performance Measurements LG G3 Same Server Payer

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Get Users	10	10	8	9	11	18	9	7	10	13	11	3.8
Verify Signed Users	22	21	19	23	19	44	57	14	22	28	27	13.4
Persists DB	38	29	26	44	29	36	38	25	37	48	34	19.6
Create Payment Response	15	12	11	11	12	22	25	14	16	10	15	4.2
Server Process	123	108	111	120	106	156	169	113	132	140	128	30.2

Table G.21: Performance Measurements LG G3 Same Server Payee

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Get users	14	11	17	15	14	17	13	18	13	12	14	2.3
Check Server Acccount	22	10	12	21	22	13	11	12	11	11	15	5.0
Sign server	43	36	12	24	14	32	10	13	13	13	21	11.9
TS Get user	18	12	14	14	12	19	19	25	17	18	17	4.0
TS Verify Signed Server	22	21	18	19	18	48	23	41	21	22	25	10.4
TS Check Server Account	13	12	12	17	11	28	12	20	12	13	15	5.4
TS Verify Signed Users	41	53	30	56	18	21	20	29	34	20	32	13.8
TS Persists DB	81	48	49	82	37	42	45	43	41	44	51	16.3
Trusted Server Process	225	255	167	266	136	232	171	225	175	168	202	43.8
Server Call	279	290	204	314	164	295	214	267	210	204	244	50.6
Decode Verify Server	22	18	19	19	22	22	17	26	47	20	23	8.8
Trust Relation Process	357	357	248	382	212	364	253	321	281	250	303	60.7
Verify Signed Users	20	17	31	18	19	19	17	22	36	21	22	6.4
Persists DB	39	38	48	35	39	41	35	45	53	40	41	5.8
Server Process	481	469	398	504	340	499	367	463	437	407	437	56.5

Table G.22: Performance Measurements Nexus 7 Different Server Payee

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Get users	13	13	11	17	15	12	18	14	14	15	14	2.1
Check Server Acccount	11	12	11	13	22	11	12	11	11	18	13	3.8
Sign server	15	13	14	62	14	14	13	14	14	55	23	18.9
TS Get user	11	13	23	13	13	13	14	14	20	11	15	3.9
TS Verify Signed Server	23	23	28	22	21	20	21	22	41	37	26	7.3
TS Check Server Accounts	12	11	12	12	12	11	12	12	22	16	13	3.4
TS Verify Signed Users	0	0	0	0	0	0	0	0	0	0	0	0.0
TS Persists DB	45	44	39	38	40	37	42	41	44	43	41	2.8
Trusted Server Process	142	146	170	165	136	162	141	181	179	151	157	16.3
Server Call	177	177	214	212	164	188	178	214	206	200	193	18.5
Decode Verify Server	24	21	29	20	21	35	21	24	45	19	26	8.3
Trust Relation Process	231	225	259	310	222	250	225	263	278	293	256	30.8
Verify Signed Users	21	21	18	19	21	31	20	20	31	16	22	5.1
Persists DB	47	39	35	40	38	41	41	39	39	33	39	3.7
Server Process	434	348	365	434	343	391	378	402	411	402	391	32.2

Table G.23: Performance Measurements Nexus 7 Different Server Payer

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Get users	19	15	15	21	18	11	19	18	13	19	17	3.2
Check Server Acccount	12	11	11	15	12	11	12	12	11	12	12	1.2
Sign server	14	14	36	21	36	13	14	45	13	21	23	11.9
TS Get user	14	17	14	13	16	13	18	14	16	13	15	1.8
TS Verify Signed Server	24	23	17	18	16	27	23	22	22	21	21	3.4
TS Check Server Accounts	12	12	13	12	12	16	14	12	12	12	13	1.3
TS Verify Signed Users	22	33	16	18	15	16	20	22	21	21	20	5.1
TS Persists DB	42	41	38	36	42	36	43	39	81	59	46	14.0
Trusted Server Process	179	168	155	141	163	151	174	164	220	184	170	21.9
Server Call	217	203	217	207	210	184	220	223	254	238	217	19.1
Decode Verify Server	51	21	21	49	22	22	21	22	22	24	28	11.9
Trust Relation Process	296	250	286	295	283	230	269	303	302	296	281	24.3
Verify Signed Users	46	19	19	15	21	20	29	21	21	20	23	8.8
Persists DB	55	41	38	35	43	37	43	46	41	44	42	5.6
Server Process	471	375	414	428	422	345	407	445	431	443	418	36.2

Table G.24: Performance Measurements LG G3 Different Server Payee

Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Get users	34	18	18	15	19	12	20	17	11	36	14	8.4
Check Server Acccount	19	13	12	11	14	11	12	12	11	12	13	2.4
Sign server	16	15	16	14	16	29	14	14	12	14	23	4.7
TS Get user	15	15	15	19	19	12	11	14	14	12	15	2.7
TS Verify Signed Server	23	23	52	35	23	21	21	22	39	22	26	10.5
TS Check Server Accounts	14	12	16	11	13	12	13	12	12	12	13	1.4
TS Verify Signed Users	0	0	0	0	0	0	0	0	0	0	0	0.0
TS Persists DB	45	41	39	37	44	39	36	41	39	41	41	2.8
Trusted Server Process	172	156	190	174	164	138	162	150	161	169	157	14.2
Server Call	214	195	242	207	209	187	201	189	197	200	193	15.8
Decode Verify Server	27	23	45	26	24	17	25	37	17	43	26	9.9
Trust Relation Process	277	248	318	260	264	246	255	254	238	269	256	22.5
Verify Signed Users	28	22	31	22	22	16	21	36	17	32	22	6.7
Persists DB	44	43	41	45	47	35	60	70	35	42	39	10.9
Server Process	434	387	458	392	409	350	417	437	341	456	391	40.8

Table G.25: Performance Measurements LG G3 Different Server Payer

Appendix H

Installation Guidelines

H.1 Client Installation

1. Consider hardware requirements: Android 4.4 device with NFC
2. Download and install MBPS application from:
   - Google Play Store: `https://play.google.com/store/apps/details?id=ch.uzh.csg.mbps.client`
   - MBPS Homepage: `http://bitcoin.csg.uzh.ch/downloads`
   - Source: `https://github.com/MBPS-Project/mbps_client`
3. Register your MBPS account
4. Verify your email address
5. Login to MBPS with your username and password

H.2 Server Installation

MBPS is developed and tested for the software versions mentioned in the listing. MBPS should also run with newer versions, but there is no guarantee as not tested. Only up to date and stable software versions should be used to run MBPS.

1. Set up a Linux server (tested with Ubuntu 12.04) with Java 1.7 (tested with version 1.7.0_51)
2. Install and set up Apache Tomcat 7 on your server (tested with version 7.0.26)
3. Install Postgresql 9 (tested with version 9.1.12)
4. Install official Bitcoin Client *bitcoind* (tested with version 0.8.6)

5. Configure bitcoind configuration file (default: `/home/username/.bitcoin/bitcoin.conf`) to allow local RPC connections, define RPC username and password
6. Run bitcoind, encrypt your wallet and wait for successful download and verification of blockchain
7. Checkout MBPS source code from Github and import to Eclipse (make sure Maven is installed)
    - Server: `https://github.com/mbps-project/mbps_server`
    - Shared-Resources: `https://github.com/mbps-project/mbps-shared-ressources`
    - Custom Serialization: `https://github.com/mbps-project/custom-serialization`
8. In the server package adapt `Config.java`, `SecurityConfig.java` and `hibernate.cfg.xml` to match your configuration (server ports, usernames, passwords etc.)
9. Export server application as `.war` file (e.g., `Server.war`)
10. Deploy and run `.war` file on Apache Tomcat server

Appendix I

Contents of the CD

The CD-ROM contains the following files:

- **Abstract.txt**
  English version of the abstract
- **Zusfsg.txt**
  German version of the abstract
- **mbps_client.zip**
  Source code of the Android mobile client application
- **mbps_server.zip**
  Source code of the server application
- **mbps-shared-resources.zip**
  Source code of the shared resources. This library contains resources which are used by the client and the server, e.g., model classes.
- **Report.zip**
  Figures and LaTeX source files of this report
- **Report.pdf**
  PDF version of this report
- **Presentation.pptx**
  The slides for the final presentation
- **Presentation.pdf**
  The slides for the final presentation in PDF format