University of Zurich[UZH]

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

Protocol Design and Implementation for a Fast and Reliable Mobile Bitcoin Payment System (MBPS) with two-way NFC

Jeton Memeti
Stalden VS, Switzerland
Student ID: 07-722-408

Supervisor: Andri Lareida, Dr. Thomas Bocek, Prof. Dr. Burkhard Stiller
Date of Submission: August 22, 2014

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

The cryptocurrency Bitcoin is gaining more and more popularity. However, everyday transactions such as to pay for a coffee are hard to achieve. The reason is that a Bitcoin transaction takes around one hour to be considered valid. The MBPS developed at the CSG group addressed this limitation by enabling instant payments over NFC. The evaluation of MBPS revealed reliability and performance issues.
The goal of this thesis is to make the payment process reliable with any NFC enabled Android 4.4 device. The time required to conduct a payment should as well be minimized, ideally below 1 second. Furthermore, a new use-case, where a user can send money, should be enabled.
The solution has been evaluated in different scenarios with different devices. Compared to the previous MBPS, much more devices are supported in the new version. The time required to conduct a payment could also be reduced significantly. Compared to the previous MBPS, a payment requires on the Nexus 10 32.7% and on the Nexus 5 40.8% less time.
Future versions can focus on increasing the user security by allowing a user to revoke a public key on the MBPS server. This is reasonable in the case a user loses his mobile device. Furthermore, a user should have the possibility to block a malicious user such that the server rejects payments between these two users. Finally, a future version can integrate Visa payWave into MBPS and enable payments with a credit card in addition to the prepaid approach applied in MBPS.

Zusammenfassung

Die Kryptowährung Bitcoin wird je länger je populärer. Alltägliche Zahlungen wie beispielsweise einen Kaffee bezahlen sind jedoch schwer machbar. Der Grund liegt darin, dass Bitcoin Transaktionen erst nach ungefähr einer Stunde als gültig betrachtet werden können. Das MBPS, entwickelt in der CSG Gruppe, hebt diese Einschränkung auf und ermöglicht unverzügliche Zahlungen über NFC. Die Evaluation des MBPS deckte jedoch Zuverlässigkeits- sowie Leistungsprobleme auf.
Das Ziel dieser Arbeit ist, Zahlungsvorgänge mit jedem NFC fähigen Android 4.4 Gerät zuverlässig zu unterstützen. Die Zeit, die ein Zahlungsvorgang benötigt, sollte ebenfalls minimiert werden und idealerweise unterhalb einer Sekunde liegen. Des Weiteren sollte ein weiterer Anwendungsfall unterstützt werden, wonach ein Benutzer Geld senden kann. Die Lösung wurde in unterschiedlichen Szenarien und mit verschiedenen Geräten evaluiert. Im Vergleich zum früheren MBPS werden in der neuen Version nun weit mehr Geräte unterstützt. Die Zeit, die benötigt wird, um einen Zahlungsvorgang abzuschliessen, konnte ebenfalls erheblich verkürzt werden. Im Vergleich zur älteren Version benötigen Zahlungen nun auf dem Nexus 10 32.7% und auf dem Nexus 5 40.8% weniger Zeit.
Zukünftige Versionen könnten die Sicherheit der Benutzer steigern, indem sie es den Benutzern ermöglichen, ihre privaten Schlüssel auf dem MBPS Server zu widerrufen. Dies ist dann sinnvoll, wenn ein Benutzer sein Mobilgerät verlieren sollte. Zudem sollte ein Benutzer die Möglichkeit haben, einen böswilligen Benutzer zu blockieren, so dass künftige Zahlungen zwischen diesen beiden Benutzern vom Server abgelehnt werden. Eine zukünftige Version könnte auch Visa payWave in das MBPS integrieren, um Zahlungen mit einer Kreditkarte, zusätzlich zum gewählten Ansatz basierend auf Vorauszahlungen, zu ermöglichen.

Acknowledgments

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.

Many thanks also go to Andri Lareida and Dr. Thomas Bocek for providing the idea of this thesis and for their great support during the design, implementation, and evaluation phases as well as concerning this written report.

Contents

Chapter 1

Introduction

Bitcoin [1] is an open, fully distributed Peer-to-Peer (P2P) digital currency that is gaining more and more popularity [2, 3, 4]. However, everyday transactions such as to pay 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 which is going to receive the bitcoins for a service or good – has to wait a given amount of time for the transaction to be confirmed. A Bitcoin transaction is confirmed when it is included in a block which is published to the network [5]. At this time, it takes around 10 minutes for a block to be created [6]. This means that Bitcoin transactions cannot 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, 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 verify that transaction [7]. This means that if the seller is not willing to risk losing bitcoins – even in the micro payments sector – he has to wait for 6 confirmations. Based on the time it takes currently to create one block, this requires both peers to wait about one hour before closing the deal. Hence, bitcoins in its current implementation and approach are not applicable when it comes to everyday transactions, e.g., paying in a market or in a restaurant.
However, there exists an approach to just broadcast a transaction without waiting for any confirmation. This so called fast payment takes around 10 seconds [8]. But there remains the risk of double spending. Therefore, this approach is not taken into consideration for this project in order to eliminate the double spending risk for the seller.

A group of three students addressed these limitations in their master project [9]. The developed Mobile Bitcoin Payment Solution (MBPS) [10] allows exchanging bitcoins in less than three seconds. The master project focused in addition on security aspects, two-way Near Field Communication (NFC), and reducing the number of Bitcoin transaction by introducing a clearing center. This system has been successfully tested in a one week test-run with the UZH Mensa Binzmühlestrasse [11].

1.1 Motivation

The evaluation of MBPS during the test-run revealed two major flaws. First, the users had difficulties on handling the client properly. Because of the limited range of the NFC chips, each user has to know where the NFC chip is on his mobile device in order to be able to establish a NFC connection. Furthermore, it often happened that a user took his device away too soon, so that not all necessary data could be streamed and as a result the payment failed. Second, payments failed when mobile devices other than the tested ones – i.e., Nexus 5, Nexus 7, Nexus 10, and ACR122u [12] – were used. It turned out that there are two NFC chip manufacturers for Android devices: NXP and Broadcom [13]. The current MBPS does not support devices with NXP chips.

The first flaw could be addressed just by improving the user interface. A progress bar could indicate that the communication is not finished yet and therefore force the user to keep the two communicating devices tapped together. However, making the message exchange faster avoids that the users get impatient and remove the devices too fast. Reducing the time two users have to keep their devices together also increases the usability.
Solving the second problem requires improving the communication protocol in order to handle the NFC connection interruptions appropriately. To not exclude a large number of users (or mobile devices), the MBPS should support the chips from both manufacturers.

Thus, the goal of this thesis is to improve the NFC payment protocol and to define a generic two-way payment protocol tailored to NFC communication channels. The current MBPS solution with its Android 4.4 client sets the baseline for the performance. The current MBPS protocol performance is considerably higher than the goal to complete a transaction below 1 second. Thus, the goal of this thesis is to improve the protocol, i.e., make it more resilient when devices from different vendors are used. The goal is a library that can reliably stream data over NFC with any kind of NFC chip (NXP, Broadcom, or external readers) with Android 4.4.

1.2 Description of Work

This thesis covers the design, implementation, and evaluation of an improved protocol for a mobile payment system using NFC. The existing MBPS is used as a baseline to keep track of the improvements. This thesis also discusses all the optimizations implemented and their impact on the different goals, i.e., reliability and speed. Furthermore, the protocol must be designed in a generic way to handle any kind of currency and not limit it to bitcoins (although this remains the main use-case in MBPS). The work is highly technical and explorative in finding the right optimizations.
The existing protocol has been designed with one use-case in mind: a seller who enters the amount he wants to receive and the payer who accepts (or rejects) the payment request. The new protocol must also support another use-case, where the payer decides what amount to pay and initiates the payment.

The work consists of several parts. The behavior of the different NFC chips needs to be analysed in order to identify the differing properties. These findings are then used to

design and implement a reliable protocol which works with devices from different vendors. The size of the messages exchanged between peers in order to conduct a payment has also to be analysed and reduced. One solution is already given and consists of using a binary format instead of Java Serialization [14]. Further options have to be explored, analysed, and implemented. The new use-case which has to be covered in the payment protocol needs as well to be designed and implemented. The performance improvements of the new payment protocol have then to be evaluated using the current MBPS as a baseline.

1.3 Thesis Goals

The goal of this thesis is to improve the existing MBPS protocol using NFC. The following goals have to be reached in order to complete this thesis successfully.

- Implement a retransmission scheme for devices that drop packets. Such a scheme should handle devices that send keep alive messages or randomly restart the communication with a NFC handshake.
- Implement session management in such way that a user can take away the device for a certain amount of time (e.g., for confirming the payment, or entering a pin) and then resume the payment.
- Use a binary format instead of Java serialization as in the current solution.
- Investigate the time to create signatures on mobile devices. If performance issues arise, evaluate different Public Key Infrastructure (PKI) schemes such as Elliptic Curve Cryptography (ECC).
- Implement early data sending, where data is sent in an optimistic way (e.g., when still polling).
- Source code needs to be open sourced, well documented, and readable.
- Evaluate the solution in different scenarios with different devices.
- Use JUnit where possible to test the protocol. Also try to cover losing packets and protocol restarts with these test cases.

1.4 Thesis Outline

The next chapters address the aforementioned goals by showing the design choices and decisions, the implementation of the payment protocol, and the evaluation showing the performance improvements based on the current MBPS payment protocol. Chapter 2 presents background information on the MBPS and the NFC capability in Android 4.4. It also covers related work in the field of mobile payment solutions. Chapter 3 describes the design choices and decisions, which are supported by findings based on benchmark tests

where suitable, as well as the implementation details concerning the new NFC payment protocol. Chapter 4 compares the performance of the current MBPS payment protocol with the new one developed in this thesis. Finally, Chapter 5 summarizes the key findings and draws a conclusion. It also shows open issues to be covered in future work.

Chapter 2

Background and Related Work

To offer a better understanding of the domain or the application area of this thesis, Section 2.1 provides background information. Section 2.2 describes related work with respect to payments with mobile devices.

2.1 Background

As already mentioned, this thesis aims to improve the payment protocol of the Mobile Bitcoin Payment System. Therefore, Section 2.1.1 gives an overview of MBPS and its key functionalities. Section 2.1.2 provides some information about NFC in Android mobile devices. Understanding the Android NFC operating mode helps to comprehend the design decisions of the payment protocol implemented in this thesis.

2.1.1 MBPS

The motivation of MBPS [9] is to overcome the time constraint of the Bitcoin core network – waiting for 6 confirmations results in a waiting time of approximately one hour before a transaction can be considered as confirmed – and therefore to develop a mobile Bitcoin payment method which allows exchanging bitcoins instantly. Furthermore, MBPS focuses in addition on security aspects, two-way NFC, and reducing the number of transaction by introducing a clearing center.
In order to overcome the time constraint, MBPS runs on a centralized system which manages all the user accounts. Furthermore, MBPS is based on a prepaid approach and therefore allows instant transactions. Users have to wait about one hour for a transaction to be confirmed only in two cases, i.e., transferring bitcoins to MBPS and transferring bitcoins out of MBPS. Transactions between two users within the system are performed immediately within seconds.
Another key feature of this approach is the implementation of a clearing center. For example there are two users, A and B, and A pays three times within a day a given amount to user B. To keep the number of core Bitcoin transactions low, instead of heaving three

separate transactions the system can initiate a core Bitcoin transaction at the end of the day. Or user B can define a given threshold, where the MBPS initiates a core Bitcoin transaction only when B's balance exceeds that threshold. In this way, transactions could be accumulated over a longer period. Since transaction fees will play a bigger role in the future [5, 15], decreasing the number of transactions which leave the MBPS system is desirable because it helps keeping the transaction fees low.
The only manner in which users can interact with the MBPS or conduct transactions between them is by means of an Android application. Using Android as operating system of the mobile client has been chosen because at the time MBPS was implemented Android made 81 percent of the market share (according to the data of IDC for Q3 of 2013) [16]. Furthermore, in contrast to Apple products, the recent generation of Android mobile devices support NFC.

Summarized, these are the key features of MBPS:

- By applying a centralized instance in combination with a prepaid approach, transactions between two users are performed immediately. This holds only for transactions within the system.
- To decrease the transaction fees, a clearing center is implemented which minimizes the number of transactions leaving the MBPS system.
- Users can interact with the MBPS or conduct transactions between them by means of an Android application. The message exchange between two MBPS users in order to conduct a transaction is covered by NFC.

2.1.2 NFC

NFC is a wireless technology (such as Bluetooth or Wi-Fi) which requires a distance of less than 4cm to connect two devices – or a device and a tag – and to allow the message exchange between them. It enables two-way communication between electronic devices by the simplicity of touch with a maximum communication speed of 424 kb/s [17].

According to [18], Android NFC devices support three modes of operation:

1. *Reader/Writer mode* allows NFC devices to read/write to passive NFC tags.
2. *Peer-to-Peer mode* allows two NFC devices to exchange data between them.
3. *Card Emulation mode* allows a NFC device to act as a NFC card in order to communicate with another NFC enabled device.

Based on the fact that two NFC devices need to communicate in order to conduct a payment, the operating mode 1 listed above is not applicable for the MBPS client, since there is a NFC tag involved. NFC tags act only as memory and are not able to carry out computations. However, this is needed in MBPS since the peers sign their payments to avoid fraud.

A promising alternative is the operating mode 2, i.e., the Peer-to-Peer mode. This mode is also applied in the Android Beam [19] functionality. However, this approach has a major drawback with respect to the usability. Once two devices are in the corresponding NFC range, your application receives an event from the Android operating system. In order to exchange a message, one of the two users has to click on the screen of his device (called *push to beam*). Exchanging messages automatically once two devices are in range is not possible with this approach. The device which, is clicked on, then sends a message to the other device and receives a response accordingly as defined in the custom protocol. The problem, however, is that if your protocol needs to exchange more than one message, the user would have to click the given amount of times. For the current MBPS payment protocol this means that the user would have to click twice on his device, since two message pairs are exchanged. This would have a negative impact on the usability.
The only operating mode which came into consideration for the current MBPS client and is also considered for the improvements developed in this thesis is the operating mode 3. The only drawback of this operating mode is that it requires to have Android 4.4 [20] installed – mobile devices with an older operating system version are not supported. The Card Emulation mode, also called Host-based Card Emulation (HCE) [21] in the Android environment, operates similar to the Peer-to-Peer mode. This means that messages cannot be exchanged arbitrarily but need to follow a request/response scheme. In the HCE mode one device acts as an initiator and sends a message to the other device acting as a responder, providing a response to that specific request. This means that the responder is not able to send any message to the initiator before having received a message. The huge advantage of the HCE mode, however, is that no user interaction is required to start the message exchange once a NFC connection has been established. This allows designing a protocol which automatically starts exchanging messages as soon as a connection between two devices has been established. Furthermore, any number of messages can be exchanged without waiting for a user interaction.

2.2 Related Work

MBPS is not unique in the area of enabling a money transfer between two entities by means of mobile devices such as smartphones or tablets. However, related systems differ in the chosen approach. First, not all related applications use NFC to exchange the payment information between the peers involved. Bitcoin Wallet [22, 23, 24] for example relies on QR codes [25] to send the payee's payment address to the payer. Second, the supported currency varies as well. Some systems (e.g., Bitcoin Wallet) allow exchanging just bitcoins, while others do only support Fiat money [26]. In the case of Bitcoin systems differ in the required confirmation time. Approaches which implement an online wallet based on prepaid enable instant transactions, while others rely on the core Bitcoin network and confirm payments after about one hour.
Below, the three most related mobile payment systems with respect to NFC or Bitcoin are described in detail. Section 2.2.1 describes Google Wallet. Section 2.2.2 depicts Tapit, a NFC payment solution introduced in Switzerland, and Section 2.2.3 presents the mobile payment solution Aircoin, which allows the exchange of bitcoins. A more extensive list of mobile payment systems can be found for example in [27].

2.2.1 Google Wallet

Google Wallet [28] is a digital wallet that combines all your credit and debit cards in one place. The key features are to pay in stores or online and to send or request money. Sending to or requesting money from a given person can be achieved trough the Google wallet application or Google's mail service Gmail. As one has to indicate the email address of the payer (in the requesting case) or of the payee (in the sending case), there are no information exchanged between the peers in order to initiate a payment. The entity which has received a payment or a payment request must, however, accept it in order to be processed. In contrast to MBPS, this has not to be synchronous but can be at a later point in time. Therefore, sending or requesting money follows a different approach as opposed to MBPS.
The Google Wallet feature which is related to MBPS is paying in stores. Both systems require NFC enabled Android devices. If a store accepts contactless payments, paying with Google Wallet is achieved by tapping the mobile phone to the point-of-sale (POS) terminal of the store. In contrast to MBPS, the two entities involved in a payment process are not two peers but one peer and one POS. In Google Wallet, the Peer-to-Peer payment is handled in the backend and does not involve any inter-peer communication. When paying in a store with the *tap and pay* feature, Google Wallet charges the selected debit or credit card for the purchase, so there is no prepaid needed as it is required in MBPS. One major drawback is that Google Wallet is only available in the United States at this time whereas MBPS can be used in the whole world, assumed one peer has an internet connection. Another drawback is that Android 4.4 devices with a NFC chip from NXP are not supported for the *tap and pay* functionality [29]. This thesis aims to enable NFC mobile payments in MBPS for all Android 4.4 devices, irrespective of the chip manufacturer.

2.2.2 Tapit

Tapit [30] has been developed in cooperation by the three major mobile network operators in Switzerland and has been launched on July 2014. Similar to Google Wallet, it acts as a virtual credit card and charges a registered card for purchases with the application. Tapit supports only one scenario, namely to pay with the mobile device on a POS. Peer-to-Peer payments, e.g., send money to a friend, are not provided. Just like Google Wallet, Tapit supports payments in Fiat money only. Cryptocurrencies such as Bitcoin are not supported at this time. The related feature to MBPS is the application of NFC.
The technology concerning the NFC is slightly different. To be able to use Tapit, a NFC enabled Android device is not enough. Additionally, the mobile device must be certified by Visa and MasterCard. The Google Nexus devices can for example not be used, since these are not certified. On the other hand, there is no restriction on the NFC chip manufacturer, so NXP and Broadcom devices can be used. Furthermore, a NFC enabled Subscriber Identity Module (SIM) card is needed. These NFC SIM cards are not capable of communicating over NFC but rather include a Secure Element which can directly communicate with the NFC controller of the Android device [31]. The encrypted credit card information are stored on this Secure Element. Finally, a credit card is needed

which can be registered in the application and which is charged for purchases with Tapit. Only credit cards from issuers who have an arrangement with Tapit can be used.
To be able to pay on a POS, neither the mobile device needs to be unlocked nor the Tapit application has to be started. For payments below 40 Swiss Francs (CHF) the mobile device can even be switched off. Amounts equals to or above CHF 40 must be accepted by entering the PIN number of the credit card, and this requires that the Tapit application is running. An advantage over Google Wallet is that Tapit can be used in any country where contactless payments are accepted and an appropriate POS terminal is available. However, outside of Switzerland only payments below CHF 40 are possible.

2.2.3 Aircoin

Aircoin [32, 33] differs from the two mobile payment systems described above in the sense that it does not accomplish a payment by means of NFC. Nevertheless, it is related to MBPS because it allows exchanging bitcoins instantly.
Unfortunately, they do not explain the approach or any technical detail. What can be seen on the demo video [33] is that Aircoin supports (at least) one use-case, namely sending money to another peer. The message exchange between the peers is accomplished over ad-hoc Wi-Fi or Bluetooth [34]. Furthermore, Bitcoin is used as currency for the payments. Payments are apparently also possible in USD, probably by converting the USD amount to Bitcoin (BTC) and charging the corresponding BTC amount.
It is not stated if Aircoin is based on a prepaid approach such as MBPS. Therefore, it is not possible to say if the money transfer is handled within Aircoin or if it is delivered to the Bitcoin core network. The demo video shows, however, that the money is transferred instantly. Therefore, Aircoin must be based on prepaid or the zero confirmation approach in Bitcoin core is applied.

Chapter 3

Design and Implementation

This Chapter covers the design choices and decisions as well as implementation details concerning the new NFC payment protocol. Based on the goals described in Section 1.3, the new NFC payment protocol has to meet the following requirements:

(R1) The new NFC payment protocol has to be an open source library which can be used by the community for other projects.

(R2) The library has to support two-way NFC payments with different currencies and not limit the usage only to bitcoins.

(R3) The lowest supported Android version has to be Android 4.4.

(R4) The library must be able to stream data over NFC with any kind of NFC chip such as NXP, Broadcom, or external readers (i.e., ACR122u). This means that streaming data should be possible irrespective of whether the device sends keep alive messages or restarts the communication with a NFC handshake.

(R5) The library must provide session management in such a way that a user can take away the device for a certain amount of time (e.g., for confirming the payment) and then resume the payment.

(R6) The whole payment should be conducted below 1 second. One step in making the message exchange faster is to use a binary format instead of Java Serialization.

(R7) The payment library must support in addition to the POS use-case (i.e., requesting money) also the Peer-to-Peer use-case (i.e., sending money).

These requirements led to the decision to split the new NFC payment protocol library into two different libraries. Section 3.1 describes the generic NFC Library, which can be used to stream data between two NFC enabled devices. The goal of this library is to make the NFC reliable and to allow streaming any kind of data and, therefore, not limit the usage only to mobile payment solutions. Section 3.2 describes the Payment Library which

is tailored to NFC payments in different currencies. It implements a custom payment protocol and uses the underlying NFC Library to handle the message exchange. This library aims to speed up the payment process among others by reducing the message size.

3.1 NFC Library

This library allows streaming data between two devices over NFC. A user of the NFC Library can use the library to stream any kind of data by implementing a custom protocol on top of it. This library has been designed and implemented in close collaboration with the supervisors.
Section 3.1.1 depicts the purpose of the NFC Library. The NFC based protocol is covered in Section 3.1.2. The structure of these NFC messages or packets is described in Section 3.1.3. Finally, Section 3.1.4 covers the implementation of the NFC Library.

3.1.1 Purpose

Having a dedicated library which is responsible just for streaming data over NFC is desirable because this is not straightforward in Android and no appropriate library exists. There are two reasons which complicate the NFC in Android concerning the Card Emulation mode.
First, message fragmentation is not provided internally, even if the Android documentation claims the contrary [35]. If one tries to stream a message which exceeds the chip's sending or receiving capacity, it will not work. Experience has shown that the maximum send/receive packet size varies between devices, or more precisely between NFC chips. Table 3.1 shows the maximum packet sizes which can be sent or received for three different devices. The first two rows report the benchmark conducted between the Nexus

Device	**Operating Mode**	**Max. Send (bytes)**	**Max. Receive (bytes)**
Samsung Nexus 10	Initiator	>253	261
Samsung Galaxy Note 3	Responder	>261	253
Samsung Nexus 10	Responder	255	255
Samsung Galaxy Note 3	Initiator	255	255
ACR122u	Initiator	54	255
ACR122u	Responder	54	255

Table 3.1: Maximum Packet Sizes

10 [36] in Initiator mode and the Galaxy Note 3 [37] in Responder mode. The Nexus 10 (Broadcom chip) can send packets larger than 253 bytes, but the Galaxy Note 3 (NXP chip) cannot process them. On the other hand, the Galaxy Note 3 can send packets larger than 261 bytes, but the the Nexus 10 cannot receive them. The next two rows show the benchmark for the same devices but in contrary modes. Here, the packet size limit lies at 255 bytes. Therefore, a packet size which can be processed by all built-in NFC chips of

the tested devices should not exceed 253 bytes. The external ACR122u NFC reader on the other hand can only write packets of at most 54 bytes. However, it can receive and process packets of at least 253 bytes.
Second, problems arise when devices with NFC chips other than Broadcom are used in Android's two-way NFC. While Broadcom chips send keep alive messages if a connection has been idle for a given time, NXP chips randomly close the NFC connection and automatically reconnect by establishing a NFC handshake. Having in mind that this is usually where the custom protocol started by sending the first message, a NFC handshake restarts the protocol or results in an error. This erratic behavior is also the reason why the previous version of MBPS failed with devices having NXP chips. The payment protocol had not been designed to handle NFC handshakes in between the message exchange.

The purpose of the NFC Library is to facilitate streaming data over NFC by offering message fragmentation/reassembly and handling the described NFC reconnections implicitly. Therefore, it targets the requirements R1, R3, and especially R4 mentioned on Page 11. This means that other software developers can use this library and design and implement their protocol rather than dealing with message fragmentation and reliability concerns. Regarding MBPS, this also addresses the reliability flaw described in Section 1.1 and aims to support all NFC enabled Android 4.4 devices, irrespective of the NFC chip manufacturer.

3.1.2 Protocol

To understand the design of the NFC Library protocol, it is important to have understood the Android NFC mode of operation, in particular the HCE mode. As mentioned in Section 2.1.2, the message exchange in the HCE mode follows a request/response scheme. Each request must receive a response, before the next request can be transmitted. Furthermore, the two NFC connected devices act in different roles, i.e., Initiator and Responder. The Initiator sends a request whereas the Responder provides the appropriate response. The Responder cannot initiate the communication by sending the first message, because it needs a request from the Initiator. Put in other words, the Initiator acts as a NFC reader and the Responder acts as a emulated NFC card.
A central aspect of the HCE mode is the Application ID (AID) [21]. When an emulated card gets tapped to a NFC reader, the Android system needs to know to which application to forward the incoming message. This decision is taken based on the AID. Therefore, the Responder needs to register an AID with the Android system on the local device. Since MBPS is a new NFC reader infrastructure as opposed to existing ones such as payment networks (e.g., Visa), it has to define its own AID [21]. Once the emulated card is in NFC range, the NFC reader receives a tag discovered event fired by the Android system and the Initiator send the AID to the Responder. This procedure is not a design decision but specified by Android [21] and the ISO 7816-4 specification [38].

These characteristics had to be taken into consideration when designing the new payment protocol, especially the new NFC layer protocol. To meet requirement R4 and, therefore, to handle reconnections while two devices are tapped together, the following approach has been chosen. Since the Initiator receives the event when the Responder is in range

– which also happens in a reconnection case – the former saves the time of each send or receive activity. This approach allows to set a threshold and to decide based on it if a common connection or a reconnection occurred. If the Initiator for example was about to send a message at time T and a reconnection (i.e., the Initiator receives a tag discovered event) occurs at time $T+X$, it can compare the difference between these times to the threshold. If X is greater than the defined threshold the Initiator starts a new session, otherwise it resumes the old one. The threshold has to be chosen such that false negatives – i.e., it is a reconnection, but X is greater than the threshold – can be avoided. This requires that the threshold is defined as small as possible. False positives – i.e., it is a new connection, but X is not greater than the threshold – require more attention, because a session can be resumed with another device. To avoid false positives an additional feature has been designed and introduced. After the Initiator has sent the AID to the Responder and received the appropriate response, it decides if it is a reconnection case based on the threshold approach. If this is true, the Initiator sends a resume message together with its unique ID assigned by the application. Once the Responder receives that message, it checks if the Initiator has detected a resume or not. If the Initiator wants to resume the session, the Responder compares the user ID contained in the received message to the one it has stored from the last session. If these two user IDs match, then the Responder responds as well with a resume message. Therefore, the Initiator distinguishes between a regular connection and a reconnection based on the threshold and the Responder checks as well the unique device ID. This procedure is illustrated in Figure 3.1. False positives are still possible with this approach but only with the same devices. At worst, the message exchange, i.e., the payment, fails.
Below, the NFC Library handshake is described first. Afterwards, the protocol for a new session is described, followed by the protocol for a session resume.

Handshake The NFC handshake in Android's HCE mode consists of sending the AID from the Initiator to the Responder and receiving a response with a length as indicated in the AID message. In a first design and implementation cycle, the resume information and the user ID were added to the AID message in order to complete the handshake in one message pair. However, this approach had to be revised because Android failed in finding the appropriate application to forward the message to. Therefore, the NFC Library applies a custom handshake consisting of two message pairs which need to be transmitted. Figure 3.1 shows a sequence diagram of the NFC Library handshake. The starting point is the *Tag Discovered Event* fired by the Android system once the Initiator and the Responder are in NFC range (or if a reconnection occurs). The Initiator receives this event and sends the *AID* to the Responder. If the Responder has not registered the same AID with the Android system, the AID message cannot be forwarded to the Responder and the protocol terminates without any other message being exchanged (*AID not ok* case). Otherwise (*AID ok* case), the Responder returns a message indicating that the AID has been selected. This message also contains the protocol's version number, which is used to detect incompatibilities right at the beginning and for backward compatibility reasons. The Initiator then analyses the received protocol version number. If it cannot deal with the given version, it sends an appropriate error message to the Responder, which in turn responds with the same error message due to the request/response scheme, and the protocol terminates properly (*Protocol Version not ok* case). If the Initiator can handle

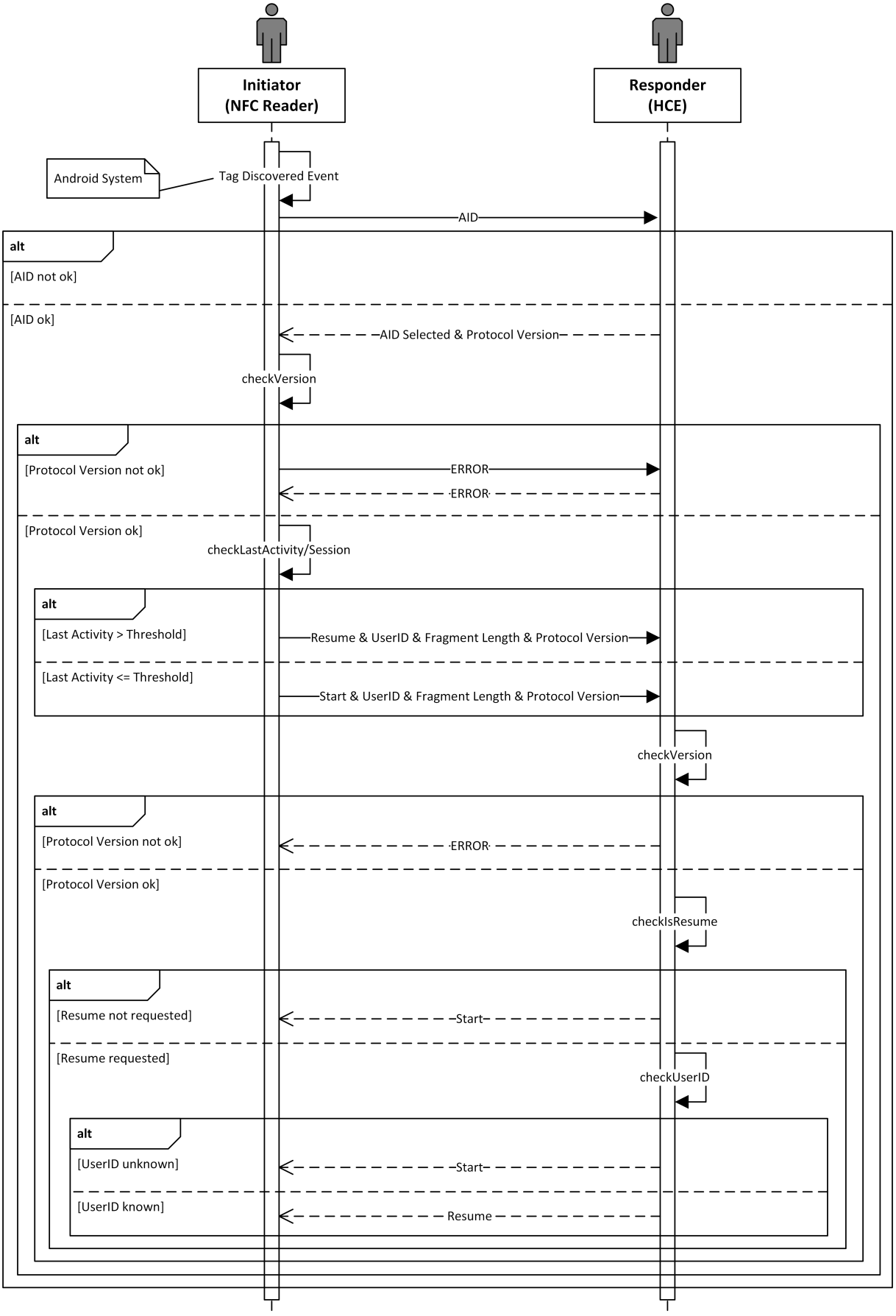

Figure 3.1: NFC Library Protocol - Handshake

packets of the indicated protocol version (*Protocol Version ok*), he checks whether this is a regular connection or a reconnection based on the threshold approach described above. Depending on the result, the Initiator sends a resume or a start message. That message also contains the Initiator's unique user ID, the maximum packet length his NFC chip can receive (see *Fragment Length*), as well as the highest protocol version number he understands. When the Responder receives this second message, he checks as well if he can handle messages of that protocol version. If this is not the case (*Protocol Version not ok*), it returns an appropriate error message and the protocol terminates. Otherwise (*Protocol Version ok*), the protocol continues. Given that the Initiator requested to start a new session (*Resume not requested*), the Responder returns a start message and the handshake completes. If the Initiator requested to resume the old session, the Responder compares the received user ID with the stored one from the last session. If the user IDs do not match (*UserID unknown*), the Responder returns a start message and the handshake completes. This means that the Initiator clears the old states and starts a new session, although he requested to resume the old session. If the user IDs match (*UserID known*), the Responder returns a resume message and the handshake completes.

New Session If the handshake revealed that the established NFC connection is a regular one and not a reconnection, a new session is initialized. The Initiator can then start sending messages to the Responder and receiving responses as defined in a custom protocol (such as the Payment Library). One purpose of the NFC Library is to offer message fragmentation and reassembly implicitly. A message should be transmitted irrespective of its size and the NFC chip capacity. The NFC Library meets this requirement by following the request/respone scheme. Besides of message fragmentation, the protocol also involves polling. The latter is needed to prevent the communication channel from being idle for several hundred milliseconds. The experience gained during the implementation of the previous MBPS client has shown that the Responder cannot wait several hundred milliseconds before returning a response, otherwise transmitting a response is not possible anymore. There were no such problems detected concerning the Initiator. Therefore, in a first design and implementation cycle, the communication channel has been idle until the Initiator was ready to send a message. Polling was only implemented and conducted when the Responder was not able to provide the response. However, the protocol design had to be revised and involve polling also when the Initiator is not ready to send a message. While testing the first implementation on real devices, it turned out that when the Initiator has a NXP chip, letting the communication channel idle for several hundred milliseconds leads to the same erratic behavior.
The part of the NFC Library protocol, which handles the transmission of messages between the Initiator and the Responder by including message fragmentation and polling is shown in Figure 3.2. After the NFC connection has been established and the handshake has completed, the Initiator must start sending the first message. If the Initiator is not ready to send the first message for several hundred milliseconds (e.g., waiting for user input or making a HTTP request), polling is applied. The Initiator sends a polling message to the Responder, indicating that the latter must wait for the message. The Responder returns as well a polling message. Once the Initiator is ready to send the first message, he stops sending polling messages. If the given message is too large to fit into one packet without exceeding the maximum supported length, it is fragmented into several packets.

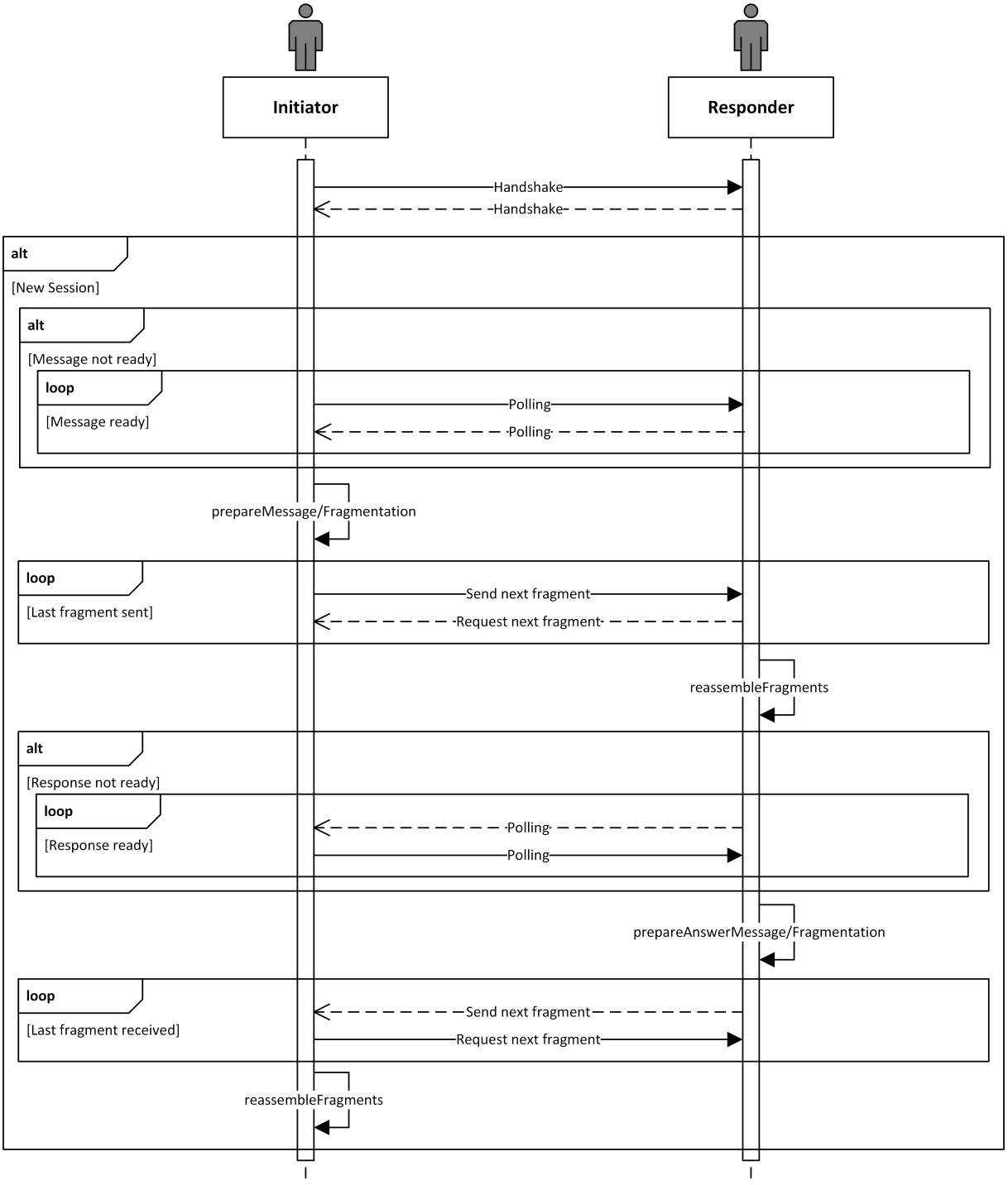

Figure 3.2: NFC Library Protocol - New Session

Each packet is then sent sequentially, until the last fragment has been sent. Based on the request/response scheme, the Responder must provide a response to each fragment. It does so by sending a packet which requests the next fragment from the Initiator. As soon as the Responder has received the last fragment, it reassembles all fragments and constructs the main message. As defined in the upper layer protocol, this message requests a response. If the Responder is not ready to return the response, it starts polling similar to the Initiator. Once the message is ready to be returned, it is fragmented if required and sent to the Initiator sequentially. The initiator can the construct the response message by reassembling the received fragments. At this time, one message has been sent to the Responder, and he has returned a response to the Initiator. If the upper layer protocol requests sending more messages, it follows the same procedure except of the initialization at the beginning.

Resume Session If the Initiator and the Responder agreed on resuming the old session based on the two factors described above, the protocol proceeds as shown in Figure 3.3. The chosen approach is Initiator oriented, since the Initiator sends a message and knows if he received the according response or if the connection aborted in the meantime. The Responder on the other side returns a response but does not know if it arrived at the Initiator. This behavior is based on the Android HCE mode of operation. So if a NFC Library handshake is conducted which results in a resume, the Initiator retransmits the last packet and the Responder returns the same packet as before. If the NFC connection has been interrupted while the Initiator was receiving fragments, he resumes this procedure by requesting the same fragment. The Responder then returns the requested fragment and the following ones until the last fragment has been delivered. The Initiator can construct the message by reassembling all fragments he received and the protocol completes. If the connection has been interrupted while the Initiator was sending fragments, the session is resumed similarly. The Initiator sends all fragments sequentially to the Responder, who reassembles them. If the Responder cannot provide the response immediately, polling is conducted in the meantime. Once the response is ready, it is fragmented if necessary and transmitted in parts. Once the Initiator receives the last fragment, he can construct the response message and the protocol completes. If the upper layer protocol requests sending more messages, this is done according to the protocol shown in Figure 3.2 except of the initialization at the beginning.

3.1.3 Packet Structure

The NFC Library cannot simply transmit messages which are defined in an upper layer protocol from one device to the other. As described above, message fragmentation and polling is applied where required. Therefore, there is a distinction between upper layer messages and NFC packets. A NFC packet is the data, which is sent from the Initiator to the Responder and vice versa. This means that any data sent between two devices over NFC according to the protocol described in Section 3.1.2 is called a NFC packet. These packets might contain only status information (such as polling or requesting the next fragment) or carry the upper layer message in the payload.

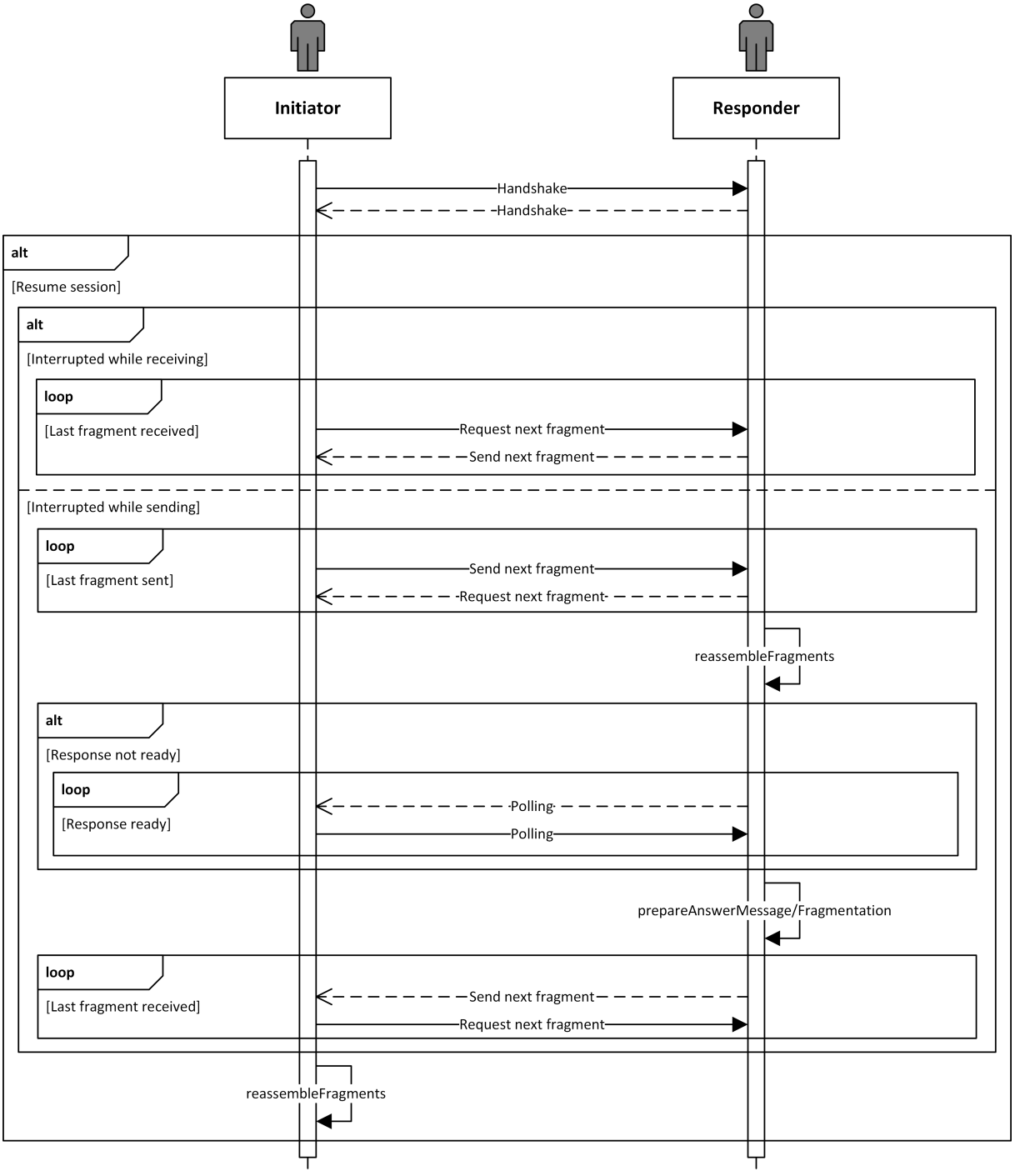

Figure 3.3: NFC Library Protocol - Resume Session

Figure 3.4 shows the structure of a NFC packet, which consists of a header and an optional payload. The header consists of two entries, called status and sequence number. Each of them uses one Byte, resulting in two bytes for the header. The payload length depends on the send/receive capacity of the underlying NFC chip. For built-in NFC chips, a packet size of 245 bytes has been chosen which works reliably for all tested devices (see Table 3.1). This means that a NFC packet sent with the internal NFC chip can contain up to 243 bytes of payload (Table 3.2 shows which message types contain a payload).

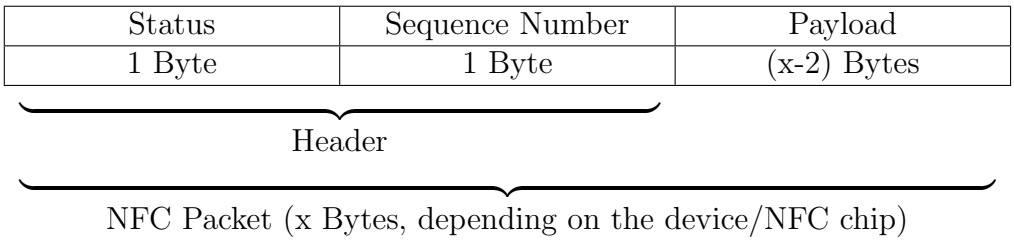

Figure 3.4: NFC Packet Structure

The design and purpose of both header entries is described below.

Status

The first Byte of the header is called status and contains the version of the packet, its type, and optional flags. The combination of various information into one Byte has the purpose to keep the overhead as small as possible, and therefore, contribute to meet requirement R6 (see Page 11). Figure 3.5 shows how much bits each of these three entries consumes.

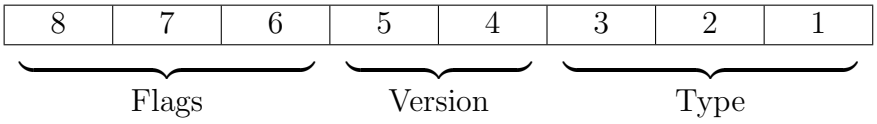

Figure 3.5: Status Header in NFC Packets

Type The type of the NFC packet is encoded in the first three bits. With three bits, 8 different types can be represented. Providing a type to a NFC packet allows distinguishing the received packets according to the protocol. Receiving an `ERROR` type packet for example shows that the sending device has encountered an error and requires to terminate the communication. Table 3.2 shows all different types and describes them. The description shows in which direction a message can be sent (e.g., `AID` is only sent from the Responder to the Initiator, while others can be sent from both modes) and provides additional information about the specific type, i.e., when it is sent and what it does contain in the payload.
The types `AID` and `USER_ID` belong to the handshake. All other types are exchanged after the handshake completes. Currently, there are only 7 types defined. One type is not used and can be defined in future modifications of the packet structure.
There are two exceptions in the NFC Library protocol concerning NFC packets. The first

packet, which is sent from the Initiator to the Responder, is the Application ID. This is not a NFC packet as defined here and, therefore, does not contain a header. Furthermore, the `READ_BINARY` type is only used to match an incoming byte sequence to a Android system command. This type does not apply to the NFC packet structure and has no header as defined above.

Version As mentioned in Section 3.1.2, the packets exchanged between the Initiator and the Responder have a version number, which assures interoperability with future versions or rather backward compatibility. The version is encoded in the status Byte of the header. By using two bits, this packet structure allows to have four different versions. If more modifications and releases are necessary, the fourth version could then easily add one more Byte to the header which then would contain the version number.

Flags The bits 6 to 8 of the status Byte are reserved for different flags which provide additional information compared to the type. Each Bit has a dedicated meaning and can be set to true or false respectively. Any reasonable permutation is possible. While each NFC packet can have exactly one type, having multiple flags is valid. The following three flags can be set:

- `RESUME` (Bit 6): If this flag is set, i.e., the given Bit is set to 1, it indicates that the sending device is requesting to resume the old session. This flag is considered only during the NFC Library handshake. If this flag is not set, it means that a new session must be started.
- `REQUEST` (Bit 7): This flag – if set – announces that the NFC packet is a request. The usual mode of operation is that the Initiator sends requests and the Responder provides appropriate responses. However, in cases where the Responder is not ready to provide a response instantly, it starts polling with this flag set to 1. The Initiator then sees that it must provide a response to this request. A response has this flag not set.
- `HAS_MORE_FRAGMENTS` (Bit 8): If this flag is set, it indicates that the given packet is part of a set of packets, and there are more fragments to be sent in order for the whole upper layer message to be transmitted. Otherwise, it is the last fragment or the message fit into one packet. This flag is only used in packets with the `DEFAULT` type.

Sequence Number

The second Byte in the NFC packet header contains the sequence number of a given NFC packet. Assigning incrementing sequence numbers to packets allows determining if a packet has been lost. However, this does only occur if the NFC connection is aborted. If a packet has been sent either it arrives or the connection aborts. In the case of an automatic reconnection, the sequence number helps to distinguish whether a packet has

Name	Description	
AID	From:	Responder
	To:	Initiator
	Info:	This type is only considered in the NFC Library handshake. It is the response to the Application ID received from the Initiator and does not contain any data in the payload. The Initiator does not send a AID type packet, but the Application ID, which is a special byte sequence and does not apply to the NFC packet structure.
USER_ID	From:	Initiator/Responder
	To:	Responder/Initiator
	Info:	This type is also considered in the NFC Library handshake. When sent from the Initiator to the Responder, it contains the user ID in the payload.
DEFAULT	From:	Initiator/Responder
	To:	Responder/Initiator
	Info:	This is the default type which is used to transmit upper layer messages. The message to be transmitted is carried in the payload.
ERROR	From:	Initiator/Responder
	To:	Responder/Initiator
	Info:	This type indicates that an error has encountered/been reported and that the communication should be terminated. The cause of error is contained in the payload.
READ_BINARY	From:	Initiator
	To:	Responder
	Info:	This type is sent from the Android system if the connection has been idle for a given time. Returning an arbitrary byte indicates that the Responder is still there. This packet is a special byte sequence and does not apply to the NFC packet structure.
GET_NEXT_FRAGMENT	From:	Initiator/Responder
	To:	Responder/Initiator
	Info:	This type indicates that the device, which sent this packet, is requesting the next fragment. This packet type does not contain any data in the payload.
POLLING	From:	Initiator/Responder
	To:	Responder/Initiator
	Info:	This type is sent if polling has to be started or as a response to a polling packet. It does not contain any data in the payload.

Table 3.2: NFC Packet Types

been processed and can be dropped or if the connection aborted before this packet has successfully been transmitted. The NFC Library handshake packets do not contain a sequence number.
By using one Byte, it is possible to have 256 different sequence numbers (from 0 to 255). This limit is only reached when transmitting very large messages or when polling. However, after reaching 255, the counter can simply be reset to 0.

3.1.4 Implementation

The NFC Library has been implemented in close collaboration with the supervisors. Once the first protocol draft was implemented, the NFC Library has been tested on physical mobile devices. These tests revealed that not all devices behave equal. The NXP chip devices for example do not allow to let the NFC channel idle for several hundred milliseconds. However, this holds only if the NXP device is in the Initiator mode – the Responder mode is not affected. This problem has not been encountered at all with Broadcom chips. These findings led to a redesign of the NFC Library and required to adapt the implementation. The design and implementation phases were therefore iterative rather than sequential. It has to be mentioned that most adaptations of the protocol were due to the differing device – or more precisely NFC chip – behaviors.
The stable protocol and the corresponding packets have been described above. The NFC Library has been tested and works reliably on the following devices: Google Nexus 5, Google Nexus 7, Google Nexus 10, Samsung Galaxy Note 3, LG G3 [39], and the ACR122u USB NFC reader. Except for the ACR122u, which provides specific device drivers, the NFC Library requires that all mobile devices run on at least Android 4.4. The ACR122u however is only able to operate in the Initiator mode – the Responder mode is not supported.

The implementation of the handshake protocol requires to define a threshold which is used to distinguish a regular connection from a reconnection. Since these reconnections occur arbitrarily, the ideal threshold cannot be specified based on benchmarks and would probably vary between devices. Instead, it is empirically determined. The current implementation uses a threshold of 500 ms, which works smoothly with all tested devices.

3.2 Payment Library

The Payment Library is an Android library tailored to two-way NFC mobile payments. It implements a protocol which allows two devices to conduct a payment. The Payment Library operates one layer above the NFC Library and uses the latter to transmit messages between NFC enabled devices. Therefore, the Payment Library has not to handle the fragmentation of large messages or NFC reconnections.
Section 3.2.1 describes the purpose of this library. The protocol defining the messages which are exchanged between two devices is covered in Section 3.2.2. These messages are called payment packets. Their structure is described in Section 3.2.3. Section 3.2.4

discusses security aspects concerning the Payment Library. It shows what security mechanisms the Payment Library applies. Section 3.2.5 covers the improvements of the Payment Library compared to the previous MBPS, which aim to speed up the payment process. Finally, Section 3.2.6 covers the implementation details of the Payment Library.

3.2.1 Purpose

The purpose of the Payment Library is to implement a secure and fast mobile payment protocol with regard to two-way NFC. Since this code needs as well to be open source (requirement R1 on Page 11), it is a bigger contribution to the community to distribute it as a library which can be used by others for similar projects rather than hard-code the payment protocol into the MBPS. In contrast to the NFC Library, this library is very specific, since it is tailored to two-way NFC payments. However, it is generic in the sense that it supports payments in different currencies (R2). By relying on the NFC Library for the message exchange, this library runs only on Android mobile devices with Android 4.4 (R3). Similar to the previous MBPS, the new payment protocol must as well provide session management and allow a user to take away the device for a certain amount of time (e.g., for confirming the payment) and then resume the payment (requirement R5).
There are two requirements concerning the Payment Library which are more important. First, R6 demands that the payment process must become faster compared to the previous MBPS, which takes around 2 seconds. Ideally, a payment should be conducted below 1 second in order to increase the usability. Since two devices need to stay tapped together to exchange the payment messages, requiring less time for the physical contact is more comfortable for the users. Second, the new payment protocol must also support the P2P (send money) use-case in addition to the POS use-case (requesting money). This is stated in requirement R7. The previous MBPS only supported requesting money. In the POS use-case, a user initializes the payment by requesting money from another user. This is similar to the scenario on a POS terminal when paying for groceries in the supermarket for example. In the P2P use-case, a user initializes the payment by sending money to another user. In the POS use-case, the payee determines the amount he wants to receive. In the P2P use-case on the other hand, the payer decides how much he wants to pay.

3.2.2 Protocol

The design of the Payment Library protocol has to consider the mode of operation of the underlying NFC Library. As mentioned above, one device must act in the Initiator mode and the other one in the Responder mode. The exchange of NFC packets is only possible with this configuration. Furthermore, the Initiator has to start sending the first upper layer message which has to be followed by a response from the Responder. The Payment Library protocol cannot be designed such that the Responder sends the first message. The previous MBPS already met these requirements. The corresponding payment protocol can be found in Appendix A. The new payment protocol differs from the previous one in the sense that it is implemented as a reusable library rather than hard-coded in the MBPS client. Furthermore, it supports payments in different currencies and is not limited to

bitcoins. And as mentioned, the new payment protocol also enables the send money use-case. Besides of these high-level differences, both versions also differ in the data they exchange. Below, the protocol is described first for the POS use-case and afterwards for the P2P use-case.

POS (Request Money) Use-Case

For the POS use-case, the protocol operates similar to the previous version. The same amount of messages is exchanged. However, the message contents differ. Figure 3.6 depicts the new protocol for this use-case. The procedure can be described as follows:

(POS-1) The payee (or the seller) enters the amount and the currency he wants to receive. Both devices can be tapped together now.

(POS-2) The payee sends a request payment message to the payer. This message contains the payee's username (as registered in the MBPS system), the currency of the payment request, and the requested amount.

(POS-3) Given that the payer (or the buyer) is willing to pay the indicated amount, he creates a payment request, signs it with his private key, and returns it to the payee. The payment request contains the payee's username, the payer's username, the currency, the amount, the signature algorithm the payer used to sign the request, the payer's key pair number (as registered in the MBPS system), as well as the payer's timestamp.

(POS-4) The payee creates a similar payment request. The payer's username and his timestamp are copied from the payer's payment request. All the other fields are provided by the payee. The payee then signs his payment request with his private key and sends both signed payment requests to the server.

(POS-5) The server processes both payment requests. He verifies the signatures, compares if the payment requests are equal (same username for payer and payee, same currency/amount/timestamp), and verifies that the transaction does not result in a negative balance for the payer. The server also assures that the same payment has not been conducted before (the first five fields of the payment request). If all these conditions are satisfied, the server accepts the payment request. Otherwise, the payment is refused.

- If the server accepts the payment, he signs the response and returns it to the payee. The first five fields of the response are equals to the payment request. In addition, the payment response contains the status of the response (accepted or rejected), the signature algorithm the server used to sign the response, and the server's key pair number. The payment is then also persisted and the balances are updated accordingly.
- If the server rejects the payment, the response is similar to the response returned when a payment is accepted. In this case, the status field is different. In addition, the server provides a reason why the payment

request has been refused. This payment response is as well signed and returned to the payee.

(POS-6) Once the payee receives the payment response from the server, he forwards the response to the payer. The payee also verifies the signature to assure that the response came from the server.

(POS-7) The payer verifies the signature as well to assure that the response indeed came from the server and was not falsified by the payee for example. Finally, the payer returns an acknowledgement message to terminate the protocol.

Polling If the payee cannot return the payment request immediately in step POS-3 – e.g., because the user has been prompted if he accepts the payment – polling is applied in the lower level protocol to keep the connection up. This has also been applied in the previous MBPS. The new payment protocol also involves polling in step POS-4 before the sever request is launched until the response arrives. This is mandatory for the NXP devices in order to keep the NFC connection alive.

Session Management In step POS-3, the payer might abort the physical contact between the two devices and take his device away in order to accept the payment by clicking on a button in the user interface for example. Once he has accepted the payment and tapped the devices together again, the payee restarts the protocol by sending the first message. The payer then recognizes that the given payment request was already accepted by the user and returns the response immediately. This assures a session management as required in R5 on Page 11 and the payment can be resumed.

Message Modifications As mentioned above, the messages exchanged between the payee and the payer differ between the previous and the new payment protocol. These are the four modifications:

- The message in POS-2 as well as the signed messages contain the currency of the payment. This is due to the requirement to support different currencies.
- The transaction numbers (see Appendix A) have been replaced by a timestamp generated by the payer. In the previous MBPS, the transaction number has been introduced to avoid multiple unintended payments. It might for example happen that a payment request has been forwarded to the server and the server accepted this payment (and updated the payer's and payee's user accounts accordingly), but the payee did not receive the server response. In this case, both participants might decide to retry the same payment. This would then result in two payments accepted by the server where only one should have been conducted. The transaction number helps to solve this problem by increasing the transaction number only when the server response arrives and the payment was accepted. With this approach, the sever would reject the second payment request because it uses the same transaction numbers on the payer's and/or the payee's side. The timestamp solves the

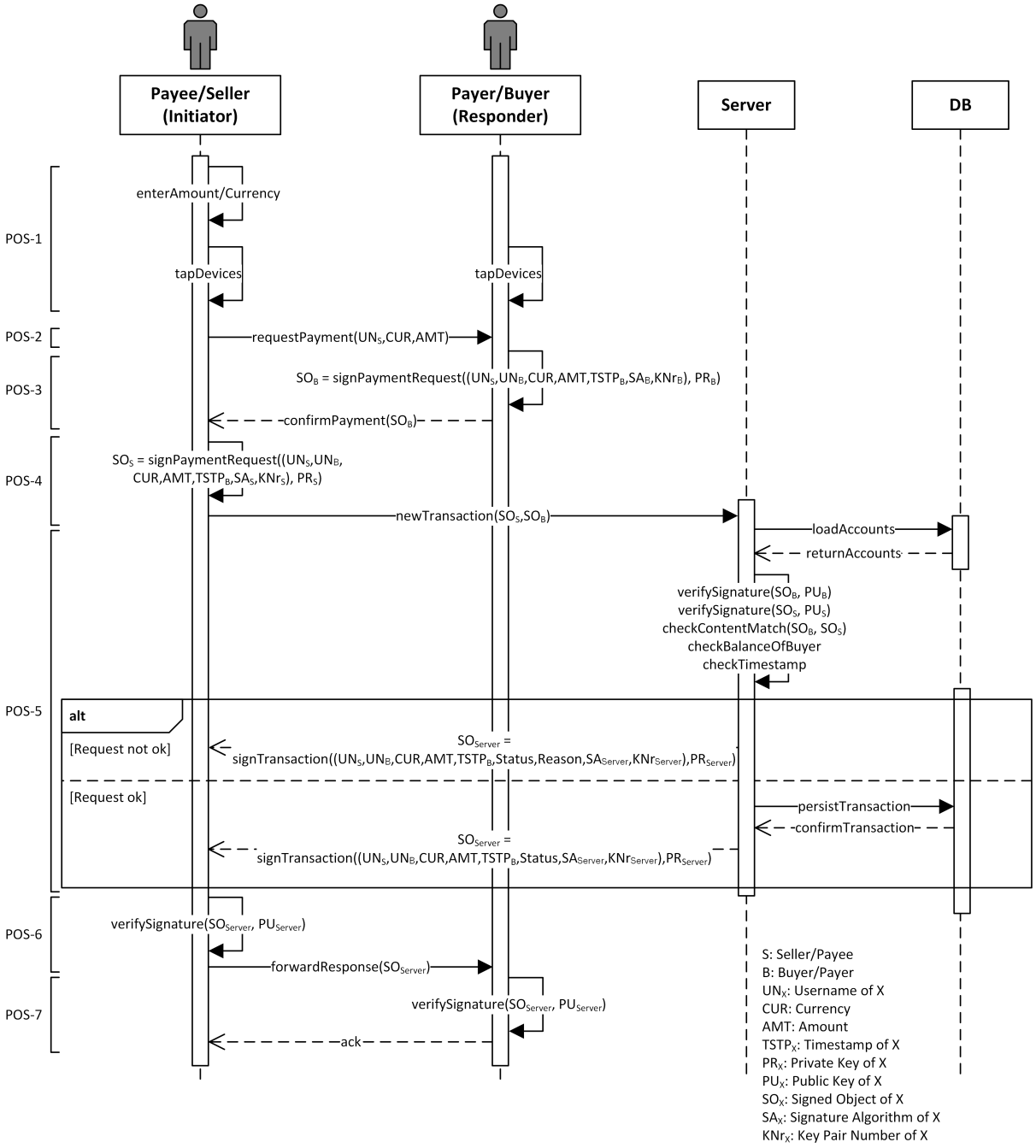

Figure 3.6: Payment Library Protocol - POS Use-Case

same challenge. Since the risk is on the payer's side to pay twice, only the payer's timestamp is included in the payment request. This results in less data exchanged over NFC, particularly in step POS-2. So if a server response does not arrive to the payer, he uses the same timestamp for future identical payments, i.e., same amount, currency, and payee. Furthermore, this approach does not prevent the user to conduct payments with other users in the meantime.

- The previous MBPS generated the key pairs for a user on the server, which stored the user's public as well as the private key. Based on the approach of MBPS, the server holds all the bitcoins the users have paid in. If a server has been compromised or is untrustworthy, all the money is lost anyway, irrespective of where the keys are stored. However, having the users generate their key pairs and store only their public keys on the server allows to guard against users who claim that they did not authorize a payment. The new MBPS and payment protocol implement this new approach. If a user is using MBPS on several devices (tablet and smartphone), he would automatically generate multiple key pairs. The key pair number entry in the signed message (see Figure 3.6) indicates which private key has been used to sign a message. This allows the receiving entity to chose the appropriate public key to verify the signature.

- The last modification concerns as well the signatures. Once an entity signs a message, it indicates in the message which signature algorithm has been used. This allows the receiving entity to use the same signature algorithm to verify the signature. By now, only one signature algorithm is used in MBPS. In the future, however, several signature algorithms might be used in parallel (e.g., transition period when the MBPS applies a new signature algorithm).

Server Response Based on the message modifications described above, the server response has changed in the new payment protocol. For example, the server has as well to indicate the signature algorithm and key number he used to sign a response. In contrast to the previous MBPS, the server signs his response also when he rejects a payment. This implies that the payer can verify the signature of any server response and assure that the given response was not compromised by a malicious payee or someone else. With the previous MBPS, a malicious payee could have forwarded a faked error message to the payer instead of the server confirmation. The payee could then suggest to give up the payment and take his leave. The payer would not be aware of that the server has booked the payment, until he checks his balance or his history. However, this might be too late. Since the new payment protocol implies signed server responses, this malicious behavior is not possible. If the payer does not receive a server response within a given time, the Payment Library notifies the user to check his balance immediately, since the server might have booked the payment.

P2P (Send Money) Use-Case

This use-case targets requirement R7. The protocol handling the P2P use-case is designed similar to the POS use-case concerning the amount of messages which is exchanged.

However, the content of these messages differ. Furthermore, the payer acts as Initiator and the payee as Responder. Figure 3.7 presents the protocol for this use-case. The procedure can be described as follows:

(P2P-1) The payer (or the buyer) enters the amount and the currency he wants to send. Both devices can be tapped together now.

(P2P-2) The payer sends a message to the payee to request the username (as registered in the MBPS system) of the latter.

(P2P-3) The payee returns his username.

(P2P-4) The payer creates a payment request similar to step POS-4. The payment request contains the payee's username, the payer's username, the currency, the amount, the signature algorithm the payer used to sign the request, the payer's key pair number, as well as the payer's timestamp. The payer then signs the payment request with his private key and sends the signed payment request to the server.

(P2P-5) The server processes the payment request. He verifies the signature, assures that the authenticated user is requesting to send money, and verifies that the transaction does not result in a negative balance for the payer. The server also assures that the same payment has not been conducted before (the first five fields of the payment request). If all these conditions are satisfied, the server accepts the payment request. Otherwise, the payment is refused.

- If the server accepts the payment, he signs the response and returns it to the payer. The first five fields of the response are equals to the payment request. In addition, the payment response contains the status of the response (accepted or rejected), the signature algorithm the server used to sign the response, and the server's key pair number. The payment is then also persisted and the balances are updated accordingly.
- If the server rejects the payment, the response is similar to the response returned when a payment is accepted. In this case, the status field is different. In addition, the server provides a reason why the payment request has been refused. This payment response is as well signed and returned to the payer.

(P2P-6) Once the payer receives the payment response from the server, he forwards the response to the payer. The payer also verifies the signature to assure that the response came from the server.

(P2P-7) The payee verifies the signature as well to assure that the response indeed came from the server and was not falsified by the payer for example. Finally, the payee returns an acknowledgement message to terminate the protocol.

Polling In the P2P use-case, polling is applied only in step P2P-4 before the sever request is launched until the response arrives. The reason is the same as in step POS-4.

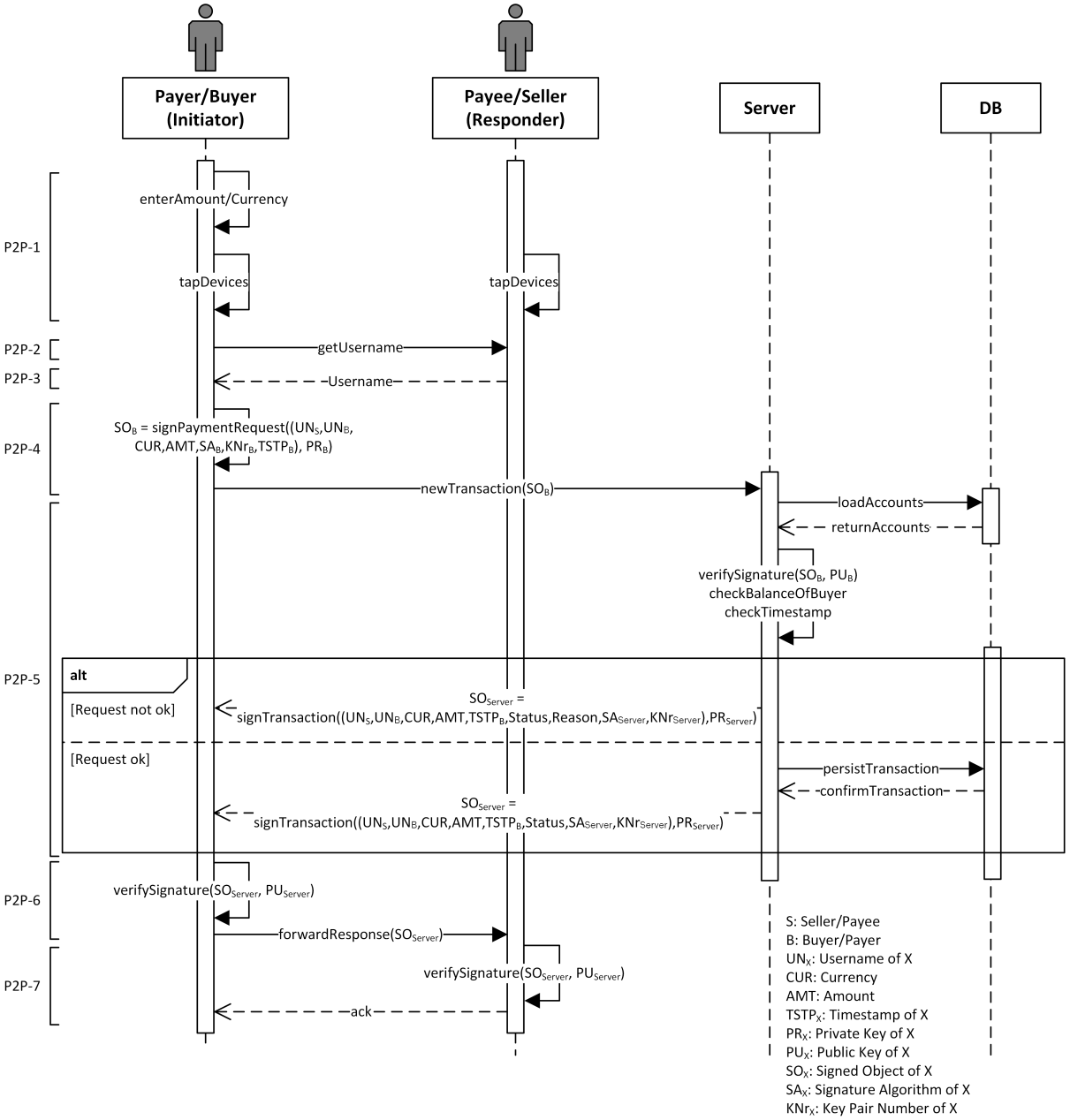

Figure 3.7: Payment Library Protocol - P2P Use-Case

Session Management The P2P use-case does not allow to abort the physical contact between the two devices by removing one device. If the payer enters the amount and taps the devices, he is willing to pay and does not need to accept the payment once again. This use-case does therefore not implement a session management. If the NFC connection is aborted on purpose, the protocol starts from scratch if the devices are tapped together again.

3.2.3 Packet Structure

As described above, in both use-cases the protocol specifies that two message pairs have to be exchanged over NFC between two devices. These messages are called payment packets. Whenever a payment packet is transmitted to the NFC counterpart, it is included in the payload of a NFC packet (see Section 3.1.3). If a payment packet is too large, it is fragmented and distributed among several NFC packets which are sent sequentially and then reassembled on the receiving side once all fragments have arrived.
Similar to the NFC packets, a payment packet also consists of a header and a payload. Figure 3.8 shows the structure of a payment packet. A payment packet or rather the payload can be arbitrary long. However, the longer the message is, the longer the users have to wait with the devices tapped together.

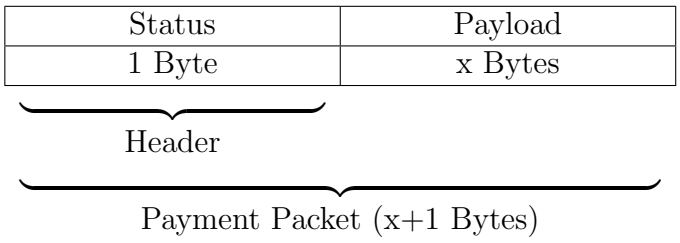

Figure 3.8: Payment Packet Structure

In contrast to the NFC packets, the payment packet header is only 1 Byte long. The status entry combines the version and flags into one single Byte in order to save space. Figure 3.9 shows which bits of the status entry are assigned to the version and to the flags respectively.

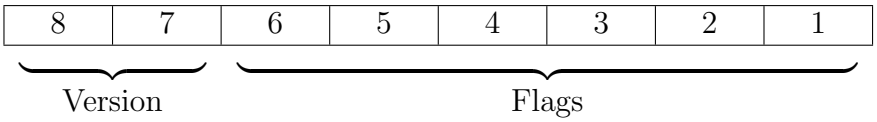

Figure 3.9: Status Header in Payment Packets

Version Two bits of the header are used to store the version of a given payment packet. The reason why a payment packet has a version number is to assure backward compatibility for the case that the packet structure is going to change in the future, e.g., by extending the header. If there will be more than three versions, the fourth one would have to add an additional Byte to the header which would then contain the packet's version number.

Flags The remaining 6 bits of the header are reserved for flags. As can be seen, there is no type information encoded in the payment packet header. Using types in the NFC packets is reasonable and necessary because different kinds of packets are exchanged, such as `POLLING`, `GET_NEXT_FRAGMENT`, or `USER_ID`. This does not apply to payment packets because the protocol is far simpler. At most, there are two different types, namely `DEFAULT` and `ERROR`. However, these two types are described as flags in the payment packet structure. Similar to the flags in the NFC packets, each flag has a dedicated position or Bit in the status header and can be set to 0 (disabled) or 1 (enabled). Furthermore, multiple flags can be enabled or disabled for a given packet. The bits 3 to 6 of the status header Byte are not used and free for future modifications of the payment packet structure and protocol. The payment packet structure has the following two flags:

- `ERROR` (Bit 1): If this flag is set, i.e., the given Bit is set to 1, it indicates that the sending device has encountered a problem and requires to abort the payment protocol. The payload of the payment packet stores the cause of error. If this flag is not set, it means that this is a regular message whose payload should be processed by the receiving device.
- `PAYER` (Bit 2): This flag indicates which role the device who sent the packet has. In both use-cases (POS and P2P) one device acts as a payer and the other device acts as a payee. Therefore, this flag helps to distinguish which use-case is being executed, i.e., how messages have to be processed and what has to be returned to the counterpart. If this flag is enabled, it means that the payer sends the given message. Otherwise, the message comes from the payee. The payload depends on the first flag and, therefore, might be a cause of error or data to be processed according to the payment protocol.

3.2.4 Security Aspects

The International Telecommunication Union (ITU) defines different security attacks against information and communication system. The Recommendation X.800 classifies those attacks in passive and active attacks [40]. In passive attacks, an opponent reads the data exchanged between two entities without modifying the data. In active attacks, an opponent modifies the data which has been sent by a sender and forwards the modified data to the receiver. Furthermore, an opponent also might create false data and forward them to the receiver. Stallings subdivides active attacks into four categories: masquerade, replay, modification of messages, and denial of service [41].
Active as well as passive attacks constitute a security risk for the MBPS and in particular the Payment Library. Below, the protection against passive attacks is described first. Afterwards, the countermeasures against active attacks are analysed. Finally, the nonrepudiation of payments is discussed.

Protection Against Passive Attacks

Passive attacks can occur on two different places in the Payment Library. First, the messages exchanged over NFC between two devices can be eavesdropped by an opponent.

As mentioned in Section 2.1.2, NFC requires a physical distance of less than 4cm between two devices in order to work. Based on this requirement, it can be excluded that the attendance of a third entity or opponent remains unnoticed by the payer or/and the payee. Therefore, the Payment Library provides neither authentication (the receiver does not know who sent the message) nor confidentiality (the message is sent in plaintext) for the messages exchanged over NFC. Second, the server request and response sent over HTTP are also vulnerable to passive attacks by default. To prevent that a payment request is eavesdropped, the user of the Payment Library has to apply SSL for the HTTP calls. MBPS also utilises SSL and assures that the data sent between the server and the client is confidential (i.e., encrypted) as well as authentic (i.e., the client can be sure he communicates with the server he intended to do).

Protection Against Active Attacks

Below, countermeasures against the first three categories of active attacks as categorised by Stallings are analysed. Denial of service attacks are neglected in this analysis because they can prevent the normal usage of MBPS but no user can lose his money on this way.

Masquerade In such an attack, an entity pretends to be a different one. For example, user X is requesting money from user Y (POS use-case). Y could create a payment request which contains user B's username as payer and return the payment request to X. Once X receives the payment request, it is forwarded to the server. If this attack would succeed, Y could conduct payments at the expense of B. However, once the server receives a payment request, he verifies the signature with the stored public key of the payer, in this case with B's public key. Assuming that Y has not stolen B's private key, the signature verification will fail and the server aborts the payment. Even if Y does not communicate with the server in this use-case, authentication is provided based on the signature. In the POS use-case, X is not interested in performing a masquerade attack, since he is the one who receives money. Nevertheless, he has to sign his payment request as well, which also serves as authentication. Furthermore, the sever can verify that both parties agreed on the same conditions when both payment requests are sent to the server.
This approach is only secure if the process of committing public keys to the server is secure. To prevent that a masquerade attack is performed while committing public keys, the client has to authenticate against the server by providing a valid username and the corresponding password. Since this concerns the MBPS server rather than the Payment Library, it is not contained in the protocol described above.
A masquerade attack is also possible in the P2P use-case. If user E wants to send money to user F, the latter could return the username of another user. However, since F is the one who is going to receive money, this attack is very unlikely to happen. User E on the other hand cannot pretend to be someone else, since he must sign the payment request. The server verifies the signature with the private key associated with the payer's username, same as in the POS use-case.

Replay In a replay attack, an opponent would capture a message and retransmit it later. So if user X is requesting money from user Y (POS use-case), X could for example

store the signed payment request he received from Y and send it later again to the server in order to conduct a payment. If the server has already booked the payment, a replay of the same payment request does not work due to the payer's timestamp. If the payer, payee, currency, and amount are equals to a previous payment, the server only accepts a payment with a newer timestamp. X however is not able to change any field without invalidating the signature. But X could not forward the payment request to the server in the first place and pretend a failed server request. After Y has left the POS, X could forward the payment request to the server. This payment would be accepted by the server. However, this is not a real replay attack. Furthermore, the payer is notified that he has signed and sent a payment request and that he has to check his transaction history to verify that the payment was not booked.

Modification of Messages In such an attack, an opponent changes the message content. Considering again the POS use-case where X is requesting money from Y, a malicious payee would try to increase the amount. While Y has agreed on paying 0.1 BTC, X could try to change the payment request to contain 1.0 BTC instead. However, modifying the signed payment request invalidates the signature, and the server will refuse that payment request.

Nonrepudiation of Payments

In contrast to the previous MBPS, the new version is protected against users who might claim they did not authorize a payment. Therefore, all users create their key pairs on their own device and commit their public keys to the server. The key pairs are encrypted on the device with the user's password he uses to authenticate against the server. The nonrepudiation of payments is achieved by persisting on the server also the payer's signature of a payment request. Payments not authorized by the given user are still possible, but in this case an opponent has stolen or cracked his username and password or he has used the victim's device. However, MBPS cannot be blamed for having stolen the victim's money.
This security service is not related to the Payment Library but rather to the MBPS server. Nevertheless, it is mentioned for the sake of completeness.

3.2.5 Speeding up the Payment Process

According to requirement R6 (see Page 11), the new Payment Library should conduct a payment below 1 second. The previous MBPS took around 2 seconds to conduct a payment (see Chapter 4 for more details). Since the Android NFC bandwidth cannot be influenced, there remain two other approaches to achieve this goal. The first option consists of decreasing the size of the messages exchanged between two devices over NFC. The second option comprises the improvement of the execution time, i.e., how fast an incoming message is processed and how fast the response is generated. The focus lies on the time to create and verify signatures on mobile devices. These two options are covered below.

Decreasing the Message Sizes

A way to decrease the size of the messages is already given in requirement R6 and consists of replacing Java Serialization by a custom binary format. The payment packets – and the messages they contain in the payload – need to be serialized before they can be passed to the NFC Library in order to be transmitted to another device over NFC. When serializing an object, the state of this object, consisting of primitive data types (e.g., Integer, Long) and/or classes, is converted to a byte stream. The previous MBPS used the Java Serializable [14] interface for this purpose. However, Java Serialization has a drawback concerning the produced byte stream length. In addition to the state of an object, Java also writes metadata of the class associated with the object instance (e.g., class name, package name) and metadata of all fields (e.g., type) to the serialization output [42]. The metadata increase the length of the produced byte stream enormously.

In the new Payment Library, Java Serialization has been replaced by a custom binary format. In contrast to Java Serialization, the custom serialization does not serialize the class name, package name, and so forth. According to the Payment Library protocol, the receiver knows what kind of data he is receiving and can deserialize the byte stream accordingly. Furthermore, the fields (such as the payer's username or the amount) are not named. They are specified by their distinct order in the serialized byte stream. Fields with a variable length (such as String or Byte array) are serialized with a preceding number indicating how many bytes are used for a given field. Fields with a fixed length (such as Byte or Long) are serialized without a preceding number. This convention assures that a receiver can deserialize a byte stream and recreate the message. The custom serialization applies to all messages exchanged over NFC between two devices. All serialized messages also contain a version number to assure backward compatibility between two devices with a differing version.

To illustrate the custom serialization, Table 3.3 shows the content of a payment request, which is sent from the payee to the payer in the POS use-case. The column on the left shows the payment request of the previous MBPS (see step 3 in Appendix A). The column on the right shows the payment request of the new Payment Library (see step POS-3 on Page 25). The raw signature is not considered in this table. The results shown in the last row are variable and depend on the payer's username length, the payee's username lenght, and the amount (only in the previous MBPS variable). To be able to compare the length of the serialized byte streams, in both columns the payer's and payee's username have been chosen to be 5 characters long. The amount is set to 1 mBTC (= 0.001 BTC). As shown, the new payment request contains more data, such as the payer's signature algorithm or his key pair number used to sign the payment request. Nevertheless, the length of the serialized byte stream is only 33 bytes long in the new Payment Library. The according payment request in the previous MBPS has a length of 485 bytes, which is almost 15 times longer compared to the new Payment Library.

Evaluating the Time for Creating and Verifying Signatures

The time to create and verify signatures on mobile devices has also been analysed. For example, performance issues might arise with the chosen signature algorithm used to sign

Previous MBPS	New Payment Library
usernamePayee: String	version: byte
usernamePayer: String	usernamePayee: String
transactionNrPayer: int	usernamePayer: String
transactionNrPayee: int	currency: byte
amount: BigDecimal	amount: long
	timestampPayer: long
	signatureAlgorithmPayer: byte
	keyNrPayer: byte
Total: 485 Bytes	Total: 33 Bytes

Table 3.3: Size of Previous and New Payment Request

a payment request or to verify a payment response. Other signature algorithms might as well have an impact on the message size.

As agreed with the supervisors, the signature algorithm used in the previous MBPS should be changed anyway to enhance the security. The previous MBPS used RSA keys with a key length of 1024 bits and SHA-1 as digest algorithm to create digital signatures. Instead, the MBPS should use RSA with a key length of 2048 bits and SHA-256. To chose the best digital signature algorithm with respect to time and output length for the Payment Library, RSA as well as Elliptic Curve DSA (ECDSA) have been evaluated. For ECDSA, only elliptic curves with a comparable security level to RSA-2048 have been taken into consideration, which is achieved by a key length between 200 and 250 bits for ECC according to [43]. The key length to be used in ECC varies between the recommendations of different national agencies and institutions such as NIST [44], BSI [45], and ANSSI [46]. Three named curves have been chosen based on these recommendations and based on what Bouncycastle [47], the security provider used in the Payment Library, provides. Based on the benchmarks shown in Appendix C, ECDSA has been chosen as signature algorithm instead of RSA. Both, ECDSA as well as RSA, us SHA-256 as digest algorithm. Based on these benchmarks and the recommendation of BSI [48], the named curve brainpoolP256r1 [49] is used to generate the PKI key pairs.

As shown in Appendix C, signing a message with ECDSA and brainpoolP256r1 (9.1ms $\pm$ 1.04ms) is faster as compared to RSA-2048 (16.9ms $\pm$ 0.3ms). Verifying a signature on the other hand takes longer with ECDSA-brainpoolP256r1 (5ms $\pm$ 0ms) as compared to RSA-2048 (0.4ms $\pm$ 0.49ms). However, since the payer needs to sign his payment request and verifies the server response in both use-cases, the longer time in verifying a signature equates the shorter time in signing for the ECDSA algorithm. So there is only a small advantage of 3.2ms in average of ECDSA-brainpoolP256r1 over RSA-2048.
A bigger advantage of ECDSA concerns the signature length. With ECDSA and brainpoolP256r1 the signature length is substantially shorter (70-72 bytes) as compared to RSA-2048 (256 bytes). The variable signature length with ECDSA is due to the ANS.1 [50] encoding. Using elliptic curves not only shortens the time used to sign messages and verify signatures, but also helps to decrease the message sizes.
Table 3.4 compares a payment request of the previous MBPS and the new Payment Library. In contrast to Table 3.3, it also contains the payer's signature. The previous MBPS

signature on the left has been created with RSA-2048. The signature of the Payment Library on the right has been created with ECDSA-brainpoolP256r1. As shown, the size of the payment request in the previous MBPS is almost 9 times greater as compared to the Payment Library, which applies a custom serialization.

Previous MBPS	New Payment Library
usernamePayee: String	version: byte
usernamePayer: String	usernamePayee: String
transactionNrPayer: int	usernamePayer: String
transactionNrPayee: int	currency: byte
amount: BigDecimal	amount: long
signaturePayer: SignedObject	timestampPayer: long
	signatureAlgorithmPayer: byte
	keyNrPayer: byte
	signaturePayer: byte[]
Total: 905 Bytes	Total: 104 Bytes

Table 3.4: Size of Previous and New Payment Request Including Signatures

Appendix B contains the benchmarks for creating PKI key pairs using RSA as well as ECC on a mobile device. However, a key pair is usually created only when the application is used for the first time or when a user uses MBPS on a new device. Furthermore, a key pair is created when the application is launched and not while the payment is in process. Therefore, these benchmarks have not been taken into consideration to chose an appropriate signature algorithm.

3.2.6 Implementation

The Payment Library has been designed and implemented on top of the NFC Library. While the latter handles message fragmentation and NFC reconnections, the Payment Library is concerned about enabling secure and fast mobile payments with regard to two-way NFC. Similar to the NFC Library, the Payment Library has been tested and works reliably on the following devices: Google Nexus 5, Google Nexus 7, Google Nexus 10, Samsung Galaxy Note 3, LG G3, and the ACR122u USB NFC reader. Since the NFC Library is used on the lower layer to transmit NFC messages, the Payment Library as well requires Android 4.4 on the mobile devices. However, this requirement is only due to the NFC Library.

The Payment Library is designed and implemented in such a way that it allows to adapt the protocol as well as the custom serialization in future versions while providing backward compatibility. This is assured by including a version number in the protocol as well as in the serialized messages. Furthermore, the signature algorithm currently used is not hard-coded. If any vulnerability concerning ECDSA and/or brainpoolP256r1 is exposed in the future, the Payment Library allows to change the signature algorithm easily.

The Payment Library is based on the concept of MBPS and, therefore, requires a server or centralized instance which accepts or rejects payment requests based on different factors such as the payer's balance. The server also keeps track of the transaction or payment history and allows users to commit their public keys in order to verify the payment request signatures.

Chapter 4

Evaluation

The evaluation focuses on the performance improvements concerning the time required to complete a payment in the previous and the new payment protocol. Therefore, this concerns only the Payment Library.
The NFC Library cannot be compared to the previous MBPS, since the older version did not support Android mobile devices with NXP chips. All payments with these devices failed. Devices with a Broadcom chip on the other hand were always successful as reported in the Master Project [9], assuming the users did not remove their devices too soon. The improvements designed and implemented in the NFC Library have solved the reliability problem with NXP chips. The NFC Library works reliably on all tested devices, irrespective of the chip manufacturer.

Below, Section 4.1 describes the setup of the performance measurements. Section 4.2 presents the performance measurements of the previous payments. Section 4.3 shows the performance measurements of the payments containing the improvements mentioned in the Payment Library. Finally, Section 4.4 compares the performance of the previous MBPS to the new Payment Library.

4.1 Setup

The performance measurements of the previous MBPS and the new Payment Library have the same setup. The measurements have been conducted with two different devices: the Google Nexus 5 and the Google Nexus 10. One device acted in the payer role and the other one in the payee role. For each device, the measurement has been repeated 10 times as payer and 10 times as payee, with the other device in the contrary role. The time has been measured for different atomic steps involved in the according role. The total time required to complete a payment has also been measured. It is important to mention, that the total time is not the sum of all atomic steps. Minor objects accesses and manipulations have not been considered in the measurements.

There is one important remark concerning the Android's HCE mode of operation. The Initiator acts as a transceiver, which means that it sends a message and receives a response

in the same turn. The Android API does not offer a functionality to detect when the Initiator is writing and when he is reading data. The Responder on the other hand only knows when he has received a message from the Initiator. Based on the Android API, it is not possible to measure how long the Responder has been reading data. Furthermore, Android also does not provide a functionality to detect when the Responder is writing the response.
This means that for the Initiator, one can start measuring before passing data to the NFC channel and stop measuring when the response arrives. For the Responder on the other hand, one can start measuring after a message has arrived and before the response is passed to the NFC channel. This affects all measurements concerning the messages exchanged over NFC.
Nevertheless, it is possible to differentiate between the time required to send and to receive a message. The Initiator tells when he started to send a message, and the Responder tells at which time he received the data. The difference between these two times is the time required to write the given data. The same approach can be chosen to measure the time the Responder needs to write a response. However, this approach requires that both devices have synchronized system times, which would complicate the measurements enormously. As agreed with the supervisors, this goes beyond the scope of this evaluation. The goal of this evaluation, which is to show if there are improvements and to quantify them, can be reached nevertheless. Since both versions are affected in the same way, the measurements can be compared and possible improvements can be quantified.

4.2 Performance of Previous MBPS

The results shown below are based on the payment protocol of the previous MBPS (see Appendix A). The results apply to the POS use-case, which is the only supported use-case in the previous MBPS. The payee role is described first, followed by the payer role.

Payee Role

Figure 4.1 shows the performance measurements for the atomic steps applying to the payee (or seller) role. The last bar is not an atomic step but the total time required to complete the whole payment on the payee's side. This figure shows the arithmetic average and the standard deviation for the total time and for each atomic step and device, based on the measurements shown in Table D.1 for the Nexus 5 and in Table D.3 for the Nexus 10. The first step, i.e., *Init NFC*, measures the NFC handshake, from the time a device has been discovered until the handshake completes by a message returned by the payer. The payee then creates a payment request (step *Create Payment Request*) and serializes it (step *Serialize Payment Request*) in order to send it over NFC. In this measurement, a payment request containing usernames with the length of 5 characters has been chosen. Using Java Serialization, the resulting byte array has a length of 511 bytes. Based on the fragment length of 245 bytes applied in the previous MBPS, this message is split into 3 fragments, which results in a total of 518 bytes to be sent including the packet headers (6 bytes for NFC layer and 1 Byte for upper layer). The step *Send Payment Request*

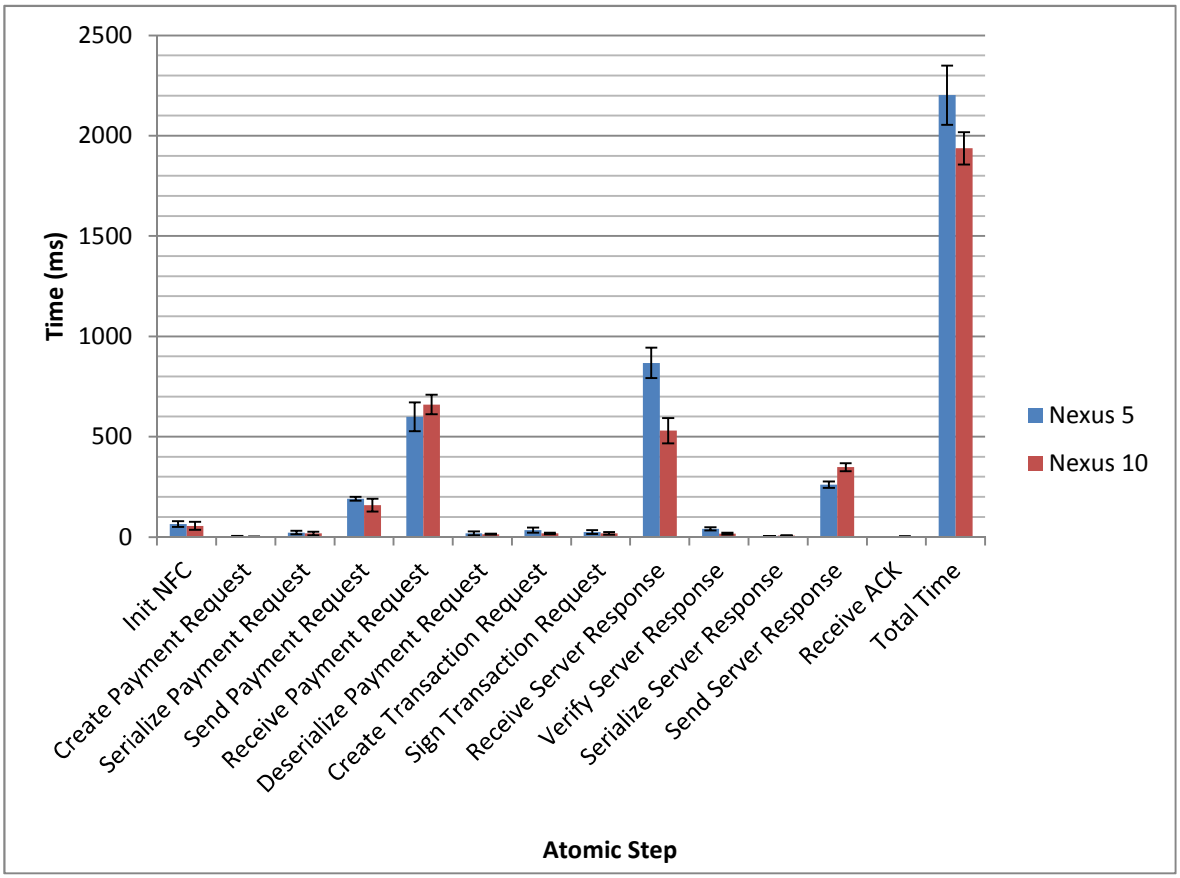

Figure 4.1: Mean and Standard Deviation Previous MBPS – Payee

indicates how long it takes to transmit these bytes to the payer. As mentioned above, it cannot be measured how long the Initiator has been writing and how long reading data. Therefore, this step also contains the first fragment of the response, i.e., the payer's signed payment request. The *Send Payment Request* step not only measures the time to send 518 bytes, but also measures the time the payer processes the message and returns 1 fragment (245 bytes). The *Receive Payment Request* step indicates the time required to receive the remaining 3 fragments from the payer. Including the headers, the complete message consisting of 4 fragments is 817 bytes long in the measured case. The payee then needs to deserialize the received data using Java Serialization, which is covered in step *Deserialize Payment Request*. Using the information received from the payer, the payee as well creates a payment request containing the complete information (step *Create Transaction Request*) and signs that payment request (step *Sign Transaction Request*). Both signed payment requests are then sent to the server over HTTP. The step *Receive Server Response* measures the time when the server request was about to be launched until the server response arrives. The payee then verifies the signature (step *Verify Server Response*) and serializes the server response (step *Serialize Server Response*) in order to forward it to the payer. Including the packet header, the message has a length of 910 bytes in the measured case. These bytes are then sent to the payer. Similar to the *Send Payment Request* step, the *Send Server Response* step not only measures the time required to send the message, but also measures the time until the first fragment is returned. Since the

ACK message fits into one fragment (3 bytes including the headers), this step measure as well the processing on the payer's side and receiving the data from the payer. The *Receive ACK* step only measure the time between the ACK message has arrived until it is processed and the protocol terminates.

There are two steps which consume the most time. The server call, measured in step *Receive Server Response*, takes 530ms (± 62.8ms) on the Nexus 10. This time is high when considering that around 2 kB are send to the server and around 900 bytes are returned. In the case of the Nexus 5, the server call takes even more (867.7ms ± 75.6ms). Since both devices were connected to the same WLAN, the difference seems to be device specific. The *Receive Payment Request* step also consumes a lot of time compared to the other steps. As mentioned above, this step measures the time required to receive 3 fragments from the payer. The first fragment of the response is already measured in the *Send Payment Request* step. This means that it takes 598.9ms (± 71.9ms) on the Nexus 5 and 660.4ms (± 48ms) on the Nexus 10 to receive 572 bytes (817 bytes less 245 bytes of the first fragment). Compared to the *Send Server Response* step, where 910 bytes are sent, this leads to the conclusion that sending messages from the Initiator to the Responder is much faster than vice versa. In the measurements, the Nexus 10 outperforms the Nexus 5 except of two steps. In the *Receive Payment Request* step, it takes longer when the Nexus 10 is receiving data from the Nexus 5 as compared to when the Nexus 5 is receiving data from the Nexus 10. It also takes longer when the Nexus 10 is sending data to the Nexus 5 (step *Send Server Response*).

Payer Role

Figure 4.2 shows the performance measurements for the atomic steps applying to the payer (or buyer) role. The last bar indicates the total time required to complete the payment on the payer's side. The arithmetic average and the standard deviation for the total time and for each atomic step and device are based on the measurements shown in Table D.2 for the Nexus 5 and in Table D.4 for the Nexus 10. The first step, i.e., *Init NFC*, measures the NFC handshake, from the time the Android system has received the complete AID message from the Initiator until the handshake message is about to be returned. This measurement neither considers the time the Responder has been reading data nor the time required to return the handshake message. The following *Receive Payment Request* step measures the time beginning when the handshake message was ready to be returned until the payee's payment request has completely been received. This step also includes the time required by the payee to serialize the payment request for example. Once the payer has received the payment request, the next atomic step consists of deserializing the message (step *Deserialize Payment Request*). Until the payer accepts the payment request, polling is applied to keep the NFC channel open. For the performance measurements, the payments are automatically accepted using the according feature of MBPS. The step *Polling* measures the time after the payment request has been deserialized until the user has accepted the payment and polling is terminated. The payer then signs the payment request he accepted (step *Sign Payment Request*) and serializes it (step *Serialize Payment Request*) in order to transmit the message over NFC. The next step, i.e., *Send Payment Request*, consists of sending the signed payment request to the

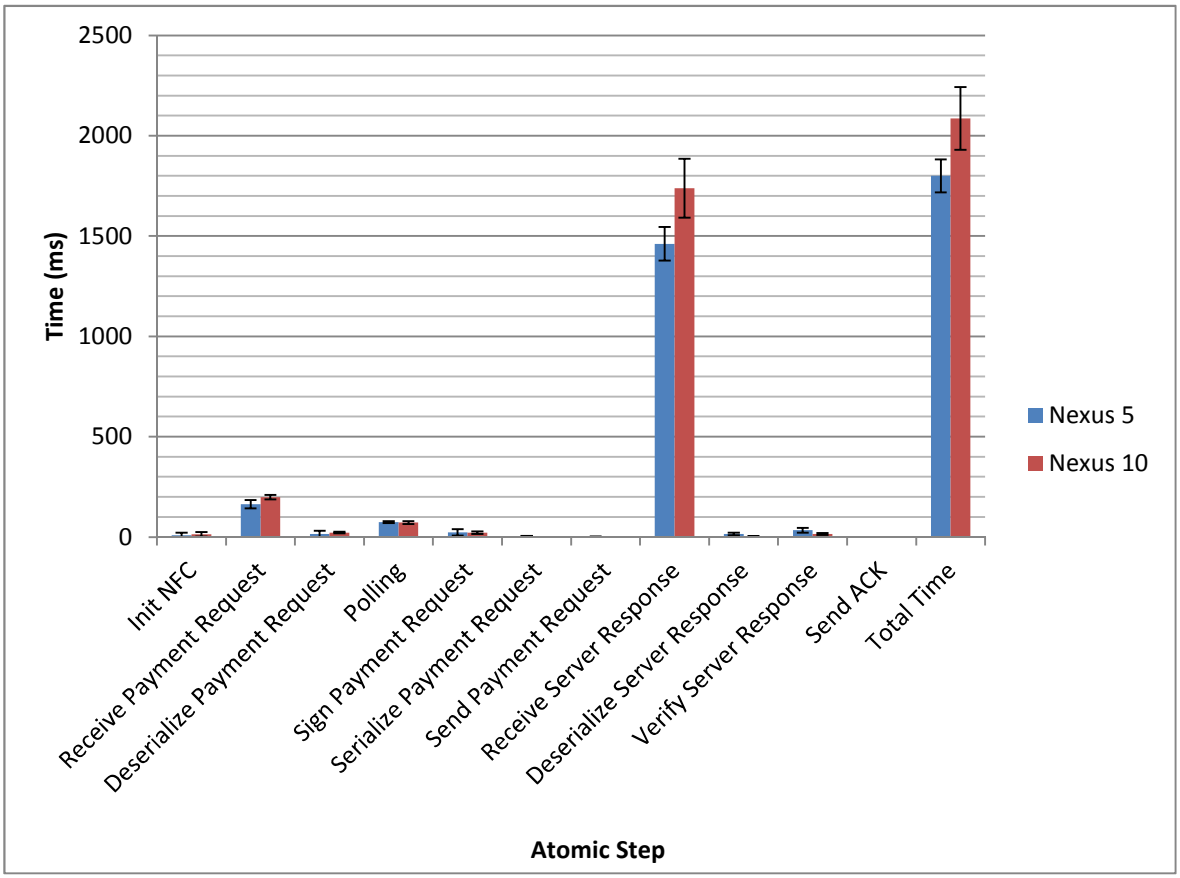

Figure 4.2: Mean and Standard Deviation Previous MBPS – Payer

payer. This step only measures the time required after the payment request has been serialized until the message is ready to be returned to the payee. The time required to write the data is not considered in this measurement but in the *Receive Server Response* step. The latter indicates the time elapsed since the signed payment request was about to be sent until the complete server response arrives. This comprises the steps 5 to 11 and a portion of the steps 4 and 12 shown in Figure 4.1. The payer then deserializes the server response (step *Deserialize Server Response*) and verifies the signature (step *Verify Server Response*). The time indicated in the step *Send ACK* reports the time elapsed after the signature has been verified until the acknowledgement message can be returned and the protocol terminates on the payer's side. The time required to write that data is not considered in the measurement.

Since the measurements of the payer role reported in Figure 4.2 do not contain the time to send or receive data, Figure 4.1 is more significant. However, there are as well measurement errors, since a proper separation between reading and writing data is not possible. When comparing the total times reported in Figure 4.1 and 4.2, it stands out that the time reported in the payee role is higher as compared to the payer role. The measurements conducted with the Nexus 5 in the payee role and the Nexus 10 in the payer role resulted in a total time of 2202.2ms ($\pm$ 147.6ms) for the payee and 2085.8ms ($\pm$ 156ms) for the payer. The contrary case with the Nexus 10 in the payee role and the Nexus 5 in the

payer role resulted in a total time of 1937.1ms (± 80.5ms) for the payee and 1799.9ms (± 82ms) for the payer. The difference between the payee and payer is due to the fact that the payee measures the time before the NFC handshake is conducted until the ACK message has been received. The payer on the other hand starts measuring the total time after the AID message has been received and stops before the ACK message is returned.

4.3 Performance of New Payment Library

The results shown below are based on the new payment protocol implemented in the Payment Library (see Section 3.2.2). The results apply to the POS use-case of the new Payment Library. This use-case is the same as discussed above for the previous MBPS. The performance measurements of the P2P use-case can be found in Appendix E. Below, the performance measurements of the payee role are discussed first, followed by the payer role.

Payee Role

Figure 4.3 shows the performance measurements for the atomic steps applying to the payee (or seller) role as well as the total time. This figure shows the arithmetic average and the standard deviation for the total time and for each atomic step and device, based on the measurements shown in Table D.5 for the Nexus 5 and in Table D.7 for the Nexus 10. The atomic steps involved in the payee role are the same as in the previous MBPS described above. Therefore, only the differences concerning the message length or measurement are discussed below:

- Step *Init NFC*: The NFC handshake in the new Payment Library exchanges two message pairs instead of one (see Figure 3.1).
- Step *Serialize Payment Request*: The custom serialization format is used instead of Java Serialization. The serialized message to be transmitted to the payer has a length of 19 bytes (including the 1 Byte Payment Library header and the 2 bytes NFC Library header) as compared to 518 bytes in the previous MBPS.
- Step *Send Payment Request*: In the measured case, the payer's signed payment request has a length of 107 bytes (including 3 bytes for headers). Since the NFC Library supports a fragment length of 245 bytes, the whole response fits into one fragment. This step not only measures the time required to send 19 bytes, but also measures the time the payer processes the message and returns 107 bytes.
- Step *Receive Payment Request*: The measurement reports the time elapsed until the payer's payment request is passed to the Payment Library and is ready to be processed. The time required to receive the data is covered in the previous step, since the response consists of only one fragment.

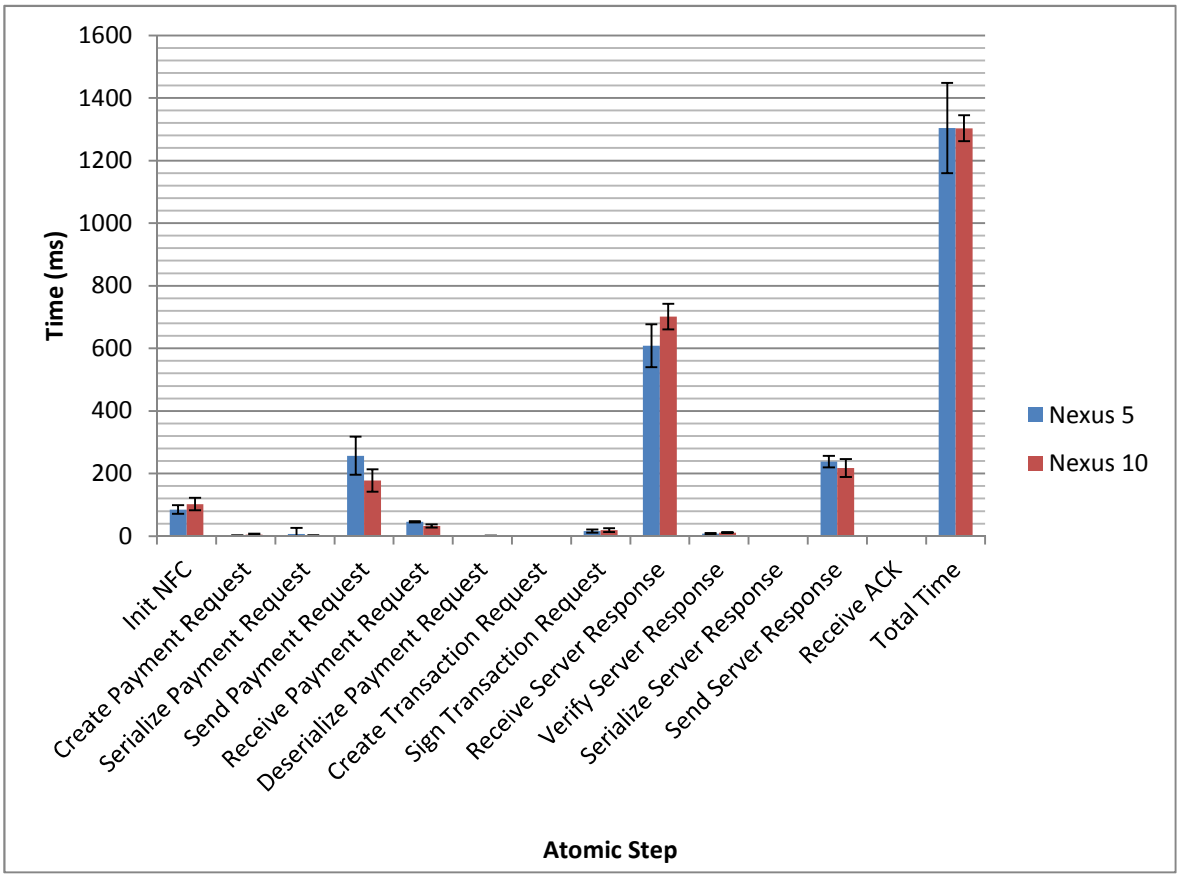

Figure 4.3: Mean and Standard Deviation Payment Library – Payee

- Step *Receive Server Response*: Based on the custom serialization, less data is sent to the server and less data is returned by approximately a factor of 8.
- Step *Send Server Response*: Based on the custom serialization format, only 107 bytes (including 3 bytes for the headers) need to be transmitted here. The ACK message has 4 bytes including the 3 bytes for the headers, as opposed to the empty message consisting only of headers which has been sent in the previous MBPS.

In these performance measurements, the server call consumes about half of the time required for a payment. The server call takes on average 608.5ms (± 68.8ms) on the Nexus 5 and 701.4ms (± 41.3ms) on the Nexus 10. Having in mind, that in the new Payment Library less data is sent to the server and less data is returned, the server call time is still very high. In contrast to the measurements of the previous MBPS, the Nexus 10 takes longer than the Nexus 5 for the server call. This is due to network issues encountered with the Nexus 10 during the measurements. The Nexus 10 often lost the WLAN connection or ran into a timeout when doing a server call. Other major differences between the two devices could not be observed. In some atomic steps, the Nexus 5 performs better than the Nexus 10 (e.g., *Init NFC*). In other steps, the Nexus 10 outperforms the Nexus 5 (e.g., *Send Payment Request*). However, the total time required to conduct a payment is equal for both devices. Assuming the same time for the server call, the Nexus 10 would nevertheless outperform the Nexus 5.

Payer Role

Figure 4.4 shows the performance measurements for the atomic steps applying to the payee (or seller) role as well as the total time. This figure shows the arithmetic average and the standard deviation for the total time and for each atomic step and device, based on the measurements shown in Table D.6 for the Nexus 5 and in Table D.8 for the Nexus 10. The atomic steps involved in the payer role are as well the same as in the previous

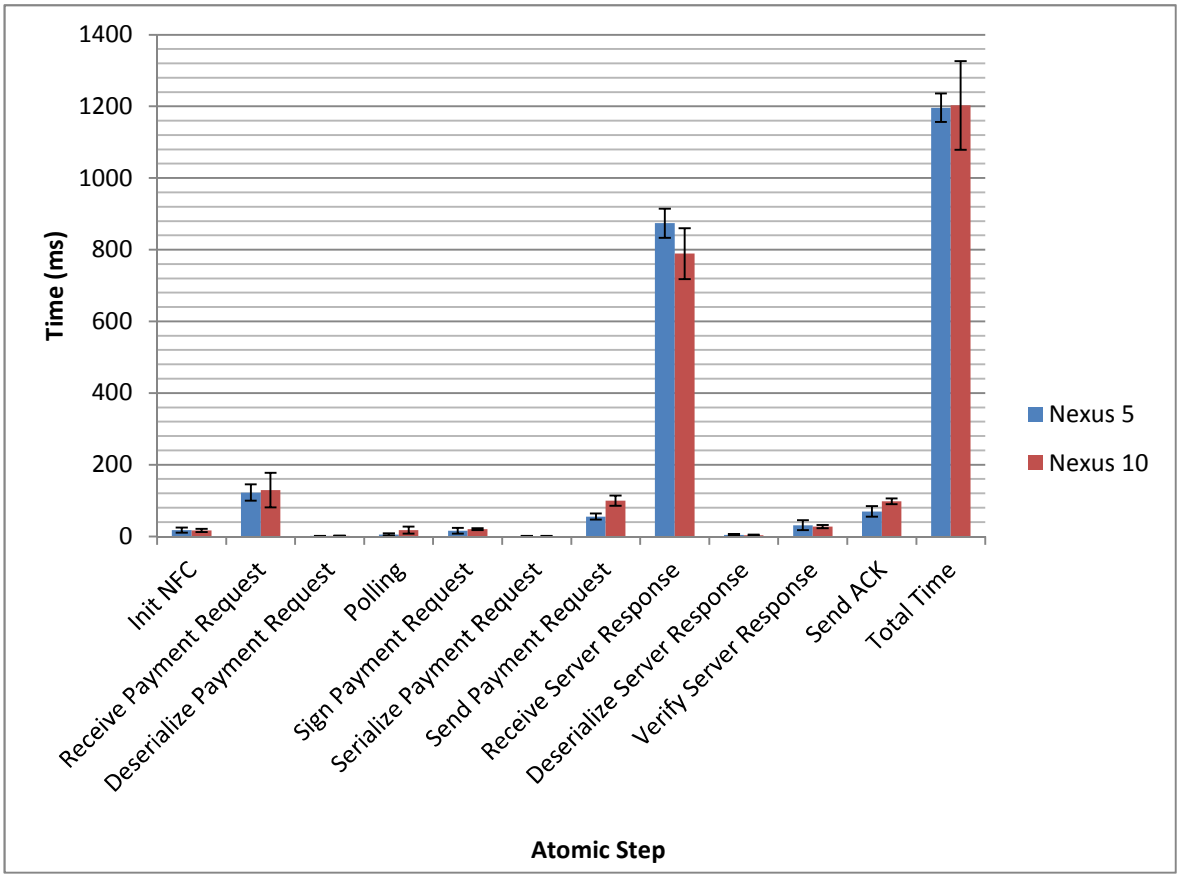

Figure 4.4: Mean and Standard Deviation Payment Library – Payer

MBPS described above. Therefore, only important differences concerning the message length or measurement are discussed below:

- Step *Init NFC*: This step measures the time elapsed from the time the Android system has received the complete AID message from the Initiator until the second handshake message is about to be returned. This step differs from the previous MBPS in the sense that the time to send the first handshake message and to receive and process the second one from the Initiator are considered in the measurement. The time required to write the second handshake message is not considered.

- Step *Receive Payment Request*: The start and end point for the measurement are equal. What differs is the message size. In the new Payment Library only 19

bytes are received as opposed to the 518 bytes of the previous MBPS (including the headers).

- Step *Send Payment Request*: What differs in this step compared to the previous MBPS is the length of the message returned. In the new Payment Library 107 bytes are returned, while in the previous MBPS 817 bytes are returned (including the headers).
- Step *Receive Server Response*: Here as well the message length differs. 107 bytes are received in the new Payment Library as opposed to 910 bytes in the previous MBPS (including the headers).
- Step *Send ACK*: A minor difference is given in the message length. The previous MBPS sent 3 bytes as acknowledgement. In the new Payment Library 4 bytes are sent (including the headers).

In the performance measurements of the new Payment Library, the results reported in Figure 4.3 are as well more significant, since they contain the time required to send or receive data, which is not represented in Figure 4.4. The total times reported in both figures also vary between the payee and payer role. The measurements conducted with the Nexus 5 in the payee role and the Nexus 10 in the payer role resulted in a total time of 1303.8ms (± 144.ms) for the payee and 1202.7ms (± 123.5ms) for the payer. The contrary case with the Nexus 10 in the payee role and the Nexus 5 in the payer role resulted in a total time of 1303.4ms (± 41.9ms) for the payee and 1196.1ms (± 39.7ms) for the payer. The reason why there is a difference between the total time for the payer and the payee is the same as discussed above for the previous MBPS.

4.4 Comparison Between Previous MBPS and Payment Library

The improvements of the Payment Library compared to the previous MBPS were conducted with the goal to speed up the payment process. By replacing Java Serialization with a custom serialization format, the length of the messages exchanged over NFC could be decreased enormously. Using ECDSA instead of RSA for the digital signatures helped to decrease the message length further. Figure 4.5 shows the performance of the previous MBPS and the new Payment Library for the payee role for the Nexus 5 as well as for the Nexus 10. The following important performance differences have been identified:

- The *Init NFC* step takes longer in the new version. However, the new handshake exchanges two message pairs, whereas the previous MBPS conducted the handshake with only one message pair.
- The *Serialize Payment Request* step is conducted faster in the new version. This is due to the custom serialization. The Nexus 10 for example serializes the payment request in the Payment Library in only 10% of the time which is required in the

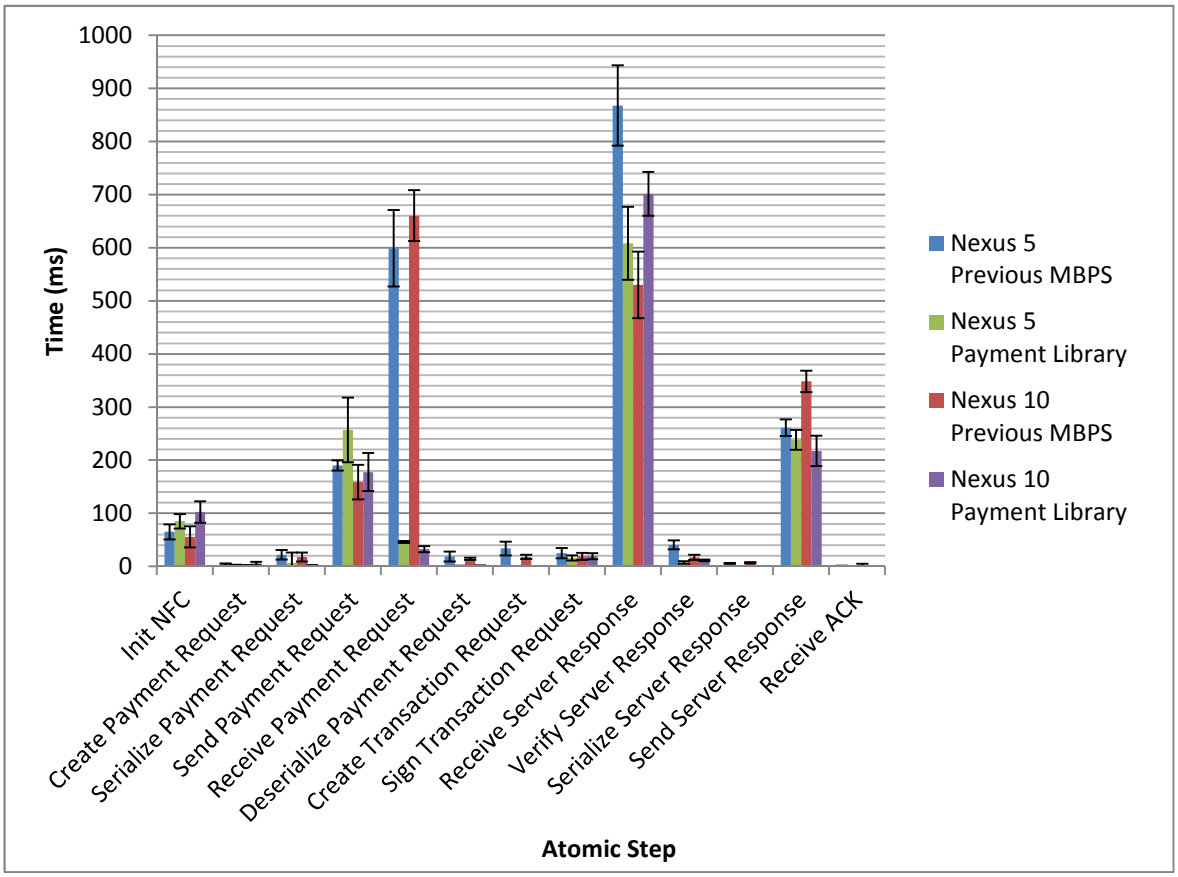

Figure 4.5: Performance Comparison Payee Role

previous MBPS. The *Deserialize Payment Request* and *Serialize Server Response* steps are also conducted much faster in the new version.

- The *Send Payment Request* step surprisingly takes longer in the new Payment Library, even if less data is sent and returned over NFC in the same step. The same holds as well for the *Send Server Response* step. Probably, this can be attributed to a slower implementation. The implementation of two libraries which are imported in the MBPS client might have a negative impact concerning the performance.

- A tremendous performance improvement is given in step *Receive Payment Request*. Due to the custom serialization and the new signature algorithm, much less data is returned from the Responder to the Initiator. However, the time required to read the response is covered in the previous step. The time measured in this step indicates the time required after the data has been received until it is processed. This indicates, that the implementation in this specific step is slowing down the protocol.

- Using ECDSA as signature algorithm in the new version has, besides the shorter signature length, not a significant performance impact concerning the Nexus 10. The steps *Sign Transaction Request* and *Verify Server Response* take a similar time. On the Nexus 5 however, both steps are conducted faster.

- The *Receive Server Response* step is significantly faster in the new version, in particular for the Nexus 5. However, it can not be concluded that this is due to the improvements in the Payment Library, since the server call depends on the server performance and on other factors, such as network latency.

Figure 4.6 compares the performance of the previous MBPS and the new Payment Library for the payer role. Again, the performance measurements of the Nexus 5 as well as for the Nexus 10 are shown in the same bar chart. Concerning the payer role, the following

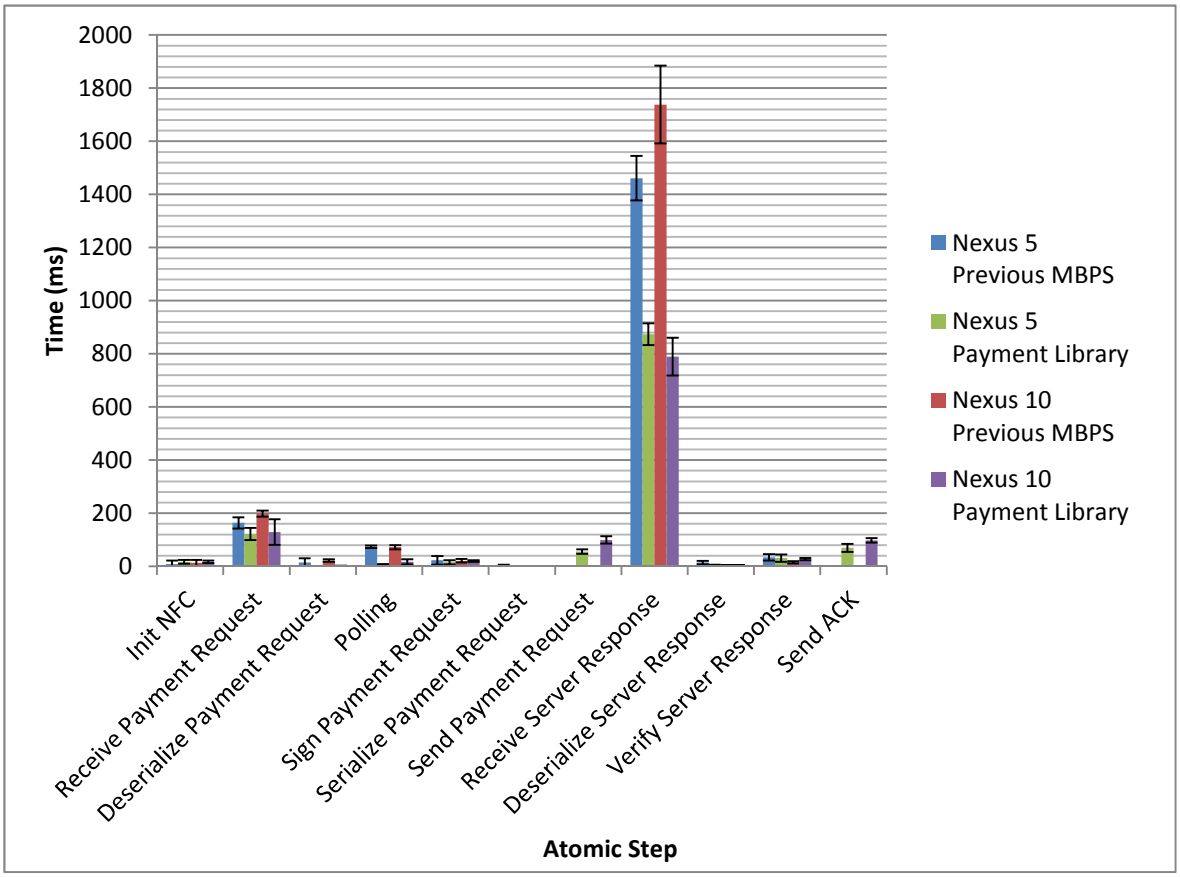

Figure 4.6: Performance Comparison Payer Role

important performance differences have been identified:

- The *Init NFC* step in the new version takes also in the payer role slightly longer for the same reason as mentioned for the payee role.
- Applying a custom serialization had also on the payer's side a positive impact. The steps *Deserialize Payment Request*, *Serialize Payment request*, and *Deserialize Server Response* are all conducted faster in the new version on both devices.
- The *Polling* step is executed much faster in the new version. The performance impact is due to a faster implementation, since messages with the same length are send in both versions.

- The tremendous performance impact in the *Receive Server Response* step (on average 948.9ms on the Nexus 10 and 587.1ms on the Nexus 5) is due to the fact that the payee has received the signed payment request from the payer much faster, since the given message is shorter.
- The steps *Send Payment Request* and *Send ACK* are significantly slower in the new version. Since writing (or returning) the data is not measured in these steps, the performance loss is due to a slower implementation in the new version.

The most important measurement, which clearly shows the performance of both versions and therefore allows a better comparison, is the total time required to conduct a payment. Figure 4.7 shows the results for the previous MBPS as well as for the new Payment Library. The results are reported for each device and each role. On the payee's device, the payment

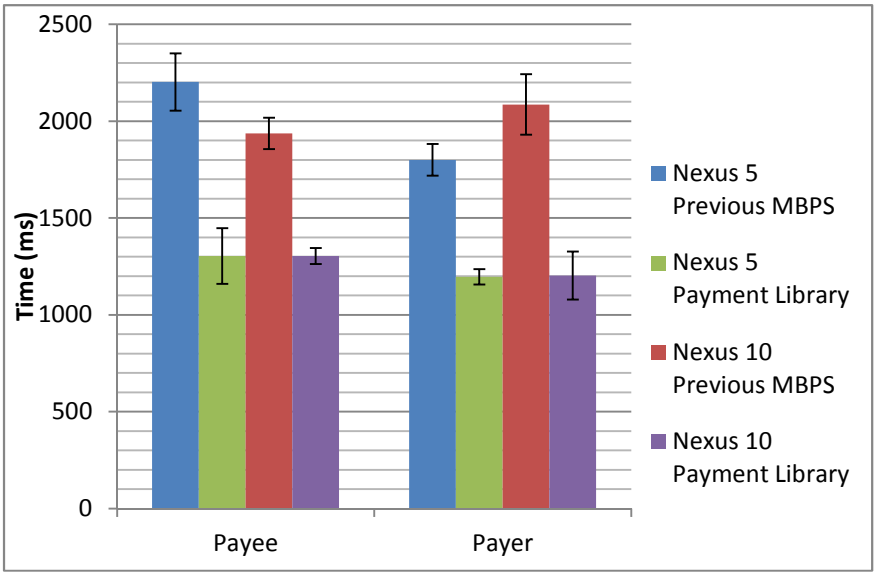

Figure 4.7: Performance Comparison Total Times Payee/Payer

in the new version is completed after 1303.8 ms (± 144.5ms) on the Nexus 5 and 1303.4ms (± 41.9ms) on the Nexus 10. Compared to the previous MBPS, where a payment took 2202.2ms (± 147.6ms) on the Nexus 5 and 1937.1ms (± 80.5ms) on the Nexus 10, this results in a significant performance improvement. On the Nexus 10, a payment requires on average 32.7% less time as compared to the previous MBPS. For the Nexus 5, the improvement is even higher. In the new version, a payment requires on average 40.8% less time.
On the payer's device, the payment in the new version is completed after 1196.1ms (± 39.7ms) on the Nexus 5 and 1202.7ms (± 123.5ms) on the Nexus 10. Compared to the previous MBPS, this results in a performance improvement of 33.5% for the Nexus 5 and 42.3% for the Nexus 10 on average.

Chapter 5

Summary and Conclusions

The evaluation of the previous MBPS during the test-run revealed two major flaws. First, a payment process was not conducted as fast as the users expected. Many users expected the payments to be conducted instantly after having tapped their devices together and, therefore, aborted the NFC contact too soon. As a result, the payments failed. Second, payments failed when mobile devices other than the tested ones were used. The evaluation of this issue revealed that NFC chips from NXP behave differently. Therefore, Android 4.4 devices with NXP chips are not supported in the previous MBPS.
The goal of this thesis is to improve the payment protocol, i.e., make it more resilient when devices, or more precisely NFC chips, from different vendors are used. Furthermore, a payment should be conducted faster with the goal to complete a transaction below 1 second. Finally, the new protocol has also to support the P2P (send money) use-case in addition to the POS (request money) use-case. The performance of the new payment protocol has to be evaluated based on the previous MBPS. To attain these goals, two different Android libraries have been designed, developed, and tested. The first library, i.e., the NFC Library, can reliably stream data between two NFC enabled Android 4.4 devices, irrespective of the chip manufacturer. It facilitates streaming data over NFC by offering message fragmentation and reassembly as well as the proper handling of NFC chip-specific behaviour implicitly. The application area of this library is not limited to mobile payments but applies to any application using two-way NFC. The second library, i.e., the Payment Library, implements a two-way payment protocol tailored to NFC communication channels. The Payment Library is built one layer above the NFC Library and uses the latter to handle the message exchange over NFC. While the NFC Library focuses on reliability, the Payment Library aims to be secure and fast.

Both libraries have been evaluated in different scenarios with different devices and work reliably. The following devices are supported: Google Nexus 5, Nexus 7, Nexus 10, Samsung Galaxy Note 3, LG G3, and ACR122u USB NFC reader.
The Payment Library has further been evaluated with respect to the performance using the previous MBPS as a baseline. The improvements of the Payment Library compared to the previous MBPS were conducted with the goal to speed up the payment process. By replacing Java Serialization with a custom serialization format, the length of the messages exchanged over NFC could be decreased enormously. Using ECDSA instead of RSA for the digital signatures helped to decrease the message length further. The performance

improvement is mainly due to the shorter messages exchanged over NFC. In the new Payment Library, a payment requires on the Nexus 10 32.7% and on the Nexus 5 40.8% less time as compared to the previous MBPS. Nevertheless, the total time required to complete a payment is still above 1 second (on average 1303.8 ms ± 144.5ms on the Nexus 5 and 1303.4ms ± 41.9ms on the Nexus 10). Therefore, this is the only goal not attained. The most time consuming step in the new payment process is the server call. In the performance measurements, this step on average amounts to 46.7% of the total time for the Nexus 5 and 53.8% for the Nexus 10. The reason for the long duration could not be identified. Most likely it is an issue concerning the MBPS server implementation. Performance issues might as well arise with the implementation of the server call on the MBPS client.

To support as well other devices besides of the mentioned ones, it is mandatory to test them first. As observed during the implementation, almost each device has a specific and unpredictable behavior. In Initiator mode, the Galaxy Note 3 for example restarted the protocol from scratch once the protocol terminated. Soft enabling and disabling the NFC feature had to be introduced just for the Galaxy Note 3.

Besides of testing the Payment Library with other devices and improving the server call performance, there are also other improvement opportunities which can be addressed in future work. If the payer removes his device to accept a payment by clicking on a button in the POS use-case, the payment protocol restarts by sending the first message. This can be improved in the sense that a session resume can be recognized. The payee can then send a shorter message to request the payer's signed payment request.
Another improvement opportunity consists of increasing the user security. Since the users generate their PKI key pairs on their own devices in the new version (see Section 3.2.2), they should also have the possibility to revoke a public key on the server even though the key pair is encrypted on the device. This might be required if a user loses his mobile device. Another improvement opportunity concerns as well the user security. If a malicious payee has received a signed payment request from the payer, the payee can modify the open source Payment Library in the sense that the payment request is not forwarded to the server (see discussion of Replay Attack in Section 3.2.4). Even if the payer is notified that the payment request might be accepted by the server and he has to check his transaction history to assure that no payment has been conducted, the payer should have the possibility to block such a malicious user. This assures, that the payee can not send the payment request to the server later and conduct a payment. Instead of blocking a user, the timestamp introduced in the Payment Library can also have a defined temporary validity. If a payment request arrives too late, the server can refuse that request. This approach would require to synchronize the time between the MBPS server and a client.
Finally, a future version can add additional value to the Payment Library or MBPS in general by integrating Visa payWave [51]. This would enable payments with a credit card between MBPS users and not limit the usage to a prepaid approach.

Bibliography

[1] "Bitcoin." http://bitcoin.org/en. Accessed: August 2014.

[2] "The New York Times, Dell Begins Accepting Bitcoin." http://dealbook.nytimes.com/2014/07/18/dell-begins-accepting-bitcoin. Accessed: August 2014.

[3] "Neue Zürcher Zeitung, Winklevoss-Zwillinge planen Bitcoin-Fonds." http://www.nzz.ch/aktuell/digital/tyler-und-cameron-winklevoss-fonds-bitcoin-1.18109996. Accessed: August 2014.

[4] "20 Minuten, Bitcoins sind der neuste Scheidungstrick." http://www.20min.ch/finance/news/story/16616768. Accessed: August 2014.

[5] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System." https://bitcoin.org/bitcoin.pdf. Accessed: August 2014.

[6] "Blockchain." http://blockchain.info/en/stats. Accessed: August 2014.

[7] "Bitcoin Wiki, Confirmation." https://en.bitcoin.it/wiki/Confirmation. Accessed: August 2014.

[8] 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.

[9] "Mobile Bitcoin Payment Solution, Master Project Report." https://files.ifi.uzh.ch/CSG/staff/tsiaras/Extern/Theses/MP_Bitcoin_in_Practice_Memeti_Bekooglu_Kaeser.pdf. Accessed: August 2014.

[10] "Mobile Bitcoin Payment Solution." http://bitcoin.csg.uzh.ch. Accessed: August 2014.

[11] "Mensa UZH Binzmühle." http://www.mensa.uzh.ch/standorte/mensa-uzh-binzmuehle.html. Accessed: August 2014.

[12] "Advanced Card Systems Ltd., ACR122U USB NFC Reader." http://www.acs.com.hk/en/products/3/acr122u-usb-nfc-reader. Accessed: August 2014.

[13] "Embedded Linux Wiki, NFC driver notes." http://elinux.org/NFC_driver_notes. Accessed: August 2014.

[14] "Java Platform SE 7, Serializable." http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html. Accessed: August 2014.

[15] "Bitcoin Wiki, Transaction Fees." https://en.bitcoin.it/wiki/Transaction_fees. Accessed: August 2014.

[16] "IDC. Android Pushes Past 80% Market Share." http://www.idc.com/getdoc.jsp?containerId=prUS24442013. Accessed: August 2014.

[17] "NFC Forum, About the Technology." http://nfc-forum.org/what-is-nfc/about-the-technology/. Accessed: August 2014.

[18] "Android Developers, Near Field Communication." http://developer.android.com/guide/topics/connectivity/nfc/index.html. Accessed: August 2014.

[19] "Android Developers, NFC Basics." http://developer.android.com/guide/topics/connectivity/nfc/nfc.html#p2p. Accessed: August 2014.

[20] "Android 4.4." http://www.android.com/versions/kit-kat-4-4/. Accessed: August 2014.

[21] "Android Developers, Host-based Card Emulation." http://developer.android.com/guide/topics/connectivity/nfc/hce.html. Accessed: August 2014.

[22] A. Schildbach, "Bitcoin Wallet." https://play.google.com/store/apps/details?id=de.schildbach.wallet. Accessed: August 2014.

[23] A. Schildbach, "Bitcoin Wallet." https://code.google.com/p/bitcoin-wallet. Accessed: August 2014.

[24] A. Schildbach, "Bitcoin Wallet." https://github.com/schildbach/bitcoin-wallet/tree/master/wallet. Accessed: August 2014.

[25] "Denso Wave, QR Code." http://www.qrcode.com/en/. Accessed: August 2014.

[26] "Wikipedia, Fiat money." http://en.wikipedia.org/wiki/Fiat_money. Accessed: August 2014.

[27] "Hongkiat." http://www.hongkiat.com/blog/digital-wallets/. Accessed: August 2014.

[28] "Google Wallet." http://www.google.com/wallet. Accessed: August 2014.

[29] "Google Wallet Help, Eligible devices for use with Google Wallet." https://support.google.com/wallet/answer/1347934?hl=en. Accessed: August 2014.

[30] "Tapit." https://www.tapit.ch. Accessed: August 2014.

[31] "Android Developers, Host-based Card Emulation with a Secure Element." https://developer.android.com/guide/topics/connectivity/nfc/hce.html#SecureElement. Accessed: August 2014.

[32] "Coinbase Apps, Aircoin." https://coinbase.com/apps/5343b2120487fb03c0000061. Accessed: August 2014.

[33] "Vimeo, Aircoin." http://vimeo.com/90285603. Accessed: August 2014.

[34] "Twitter, Aircoin." https://twitter.com/aircoinapp. Accessed: August 2014.

[35] "Android Developers, IsoDep." http://developer.android.com/reference/android/nfc/tech/IsoDep.html#transceive(byte[]). Accessed: August 2014.

[36] "Google, Nexus 10." https://www.google.com/nexus/10/. Accessed: August 2014.

[37] "Samsung, Galaxy Note 3." http://www.samsung.com/ch/consumer/mobile-phone/mobile-phone/smartphones/SM-N9005ZKEAUT?subsubtype=galaxy. Accessed: August 2014.

[38] "CardWerk, ISO 7816-4 Smart Card Standard." http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816-4.aspx. Accessed: August 2014.

[39] "LG, G3." http://www.lg.com/de/handy/lg-G3. Accessed: August 2014.

[40] International Telecommunication Union Telecommunication Standardization Sector, "X.800: Security architecture for Open Systems Interconnection for CCITT applications." http://www.itu.int/rec/T-REC-X.800-199103-I/en. Accessed: August 2014.

[41] W. Stallings, *Network Security Essentials, Applications and Standards*. Pearson, 4th ed., 2011. 17-30.

[42] "JavaWorld, The Java serialization algorithm revealed." http://www.javaworld.com/article/2072752/the-java-serialization-algorithm-revealed.html. Accessed: August 2014.

[43] "BlueKrypt, Cryptographic Key Length Recommendation." http://www.keylength.com/en/compare/. Accessed: August 2014.

[44] "National Institute of Standards and Technology." http://www.nist.gov/. Accessed: August 2014.

[45] "Bundesamt für Sicherheit in der Informationstechnik." https://www.bsi.bund.de/DE/Home/home_node.html. Accessed: August 2014.

[46] "Agence nationale de la sécurité des systèmes d'information." http://www.ssi.gouv.fr/en/. Accessed: August 2014.

[47] "Bouncycastle." http://www.bouncycastle.org/. Accessed: August 2014.

[48] Bundesamt für Sicherheit in der Informationstechnik, "Kryptographische Vorgaben für Projekte der Bundesregierung." https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116-3.pdf?__blob=publicationFile, 2014. Accessed: August 2014.

[49] Lochter, M. and Merkle, J., "Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation." IETF. RFC 5639, 2010.

[50] International Telecommunication Union Telecommunication Standardization Sector, "Introduction to ASN.1." `http://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspx`. Accessed: August 2014.

[51] "Visa payWave for Mobile." `https://developer.visa.com/paywavemobile`. Accessed: August 2014.

Abbreviations

AID	Application ID
BTC	Bitcoin
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve DSA
HCE	Host-based Card Emulation
ITU	International Telecommunication Union
MBPS	Mobile Bitcoin Payment Solution
NFC	Near Field Communication
P2P	Peer-to-Peer
PKI	Public Key Infrastructure
POS	Point-of-sale
SIM	Subscriber Identity Module

List of Figures

List of Tables

Appendix A

Payment Protocol Previous MBPS

Figure A.1 shows the payment protocol of the previous MBPS. It shows the POS use-case (i.e., requesting money), since it is the only use-case supported in the previous MBPS. This protocol operates as follows:

1. The payee enters the amount he wants to receive. Both devices can be tapped together now.

2. The payee sends his username (registered in the MBPS system), the payment amount (in Bitcoin), and his transaction number to the payer.

3. Given that the payer is willing to pay the indicated amount, he creates a payment request, signs it with his private key, and returns it to the payee. The payment request contains the payee's username, the payer's username, the bitcoin amount, the payee's transaction number, and the payer's transaction number.

4. The payee creates a similar payment request. The payer's username and his transaction number are copied from the payer's payment request. All the other fields are provided by the payee. The payee then signs his payment request with his private key and sends both signed payment requests to the server.

5. The server processes the payment request. He verifies the signatures, compares if the payment requests are equals (same username for payer and payee, same amount), and verifies that the transaction does not result in a negative balance for the payer. The server also assures that the indicated transaction numbers are by one greater than the persisted ones on the server. If all these conditions are satisfied, the server accepts the payment request. Otherwise, the payment is refused.

    - If the server accepts the payment, he signs the response and returns it to the payee. The response contains the same information as the request. The payment is then also persisted and the balances are updated accordingly.
    - If the server refuses the payment, he returns an appropriate cause of error to the payee.

6. Once the payee receives the server response, he forwards it to the payer. If the transaction was successful, the payee verifies the signature to assure that the response came from the server.

7. If the transaction was successful, the payer verifies the signature as well to assure that the response indeed came from the server and was not falsified by the payee for example. Finally, the payer returns an acknowledgement message to terminate the protocol.

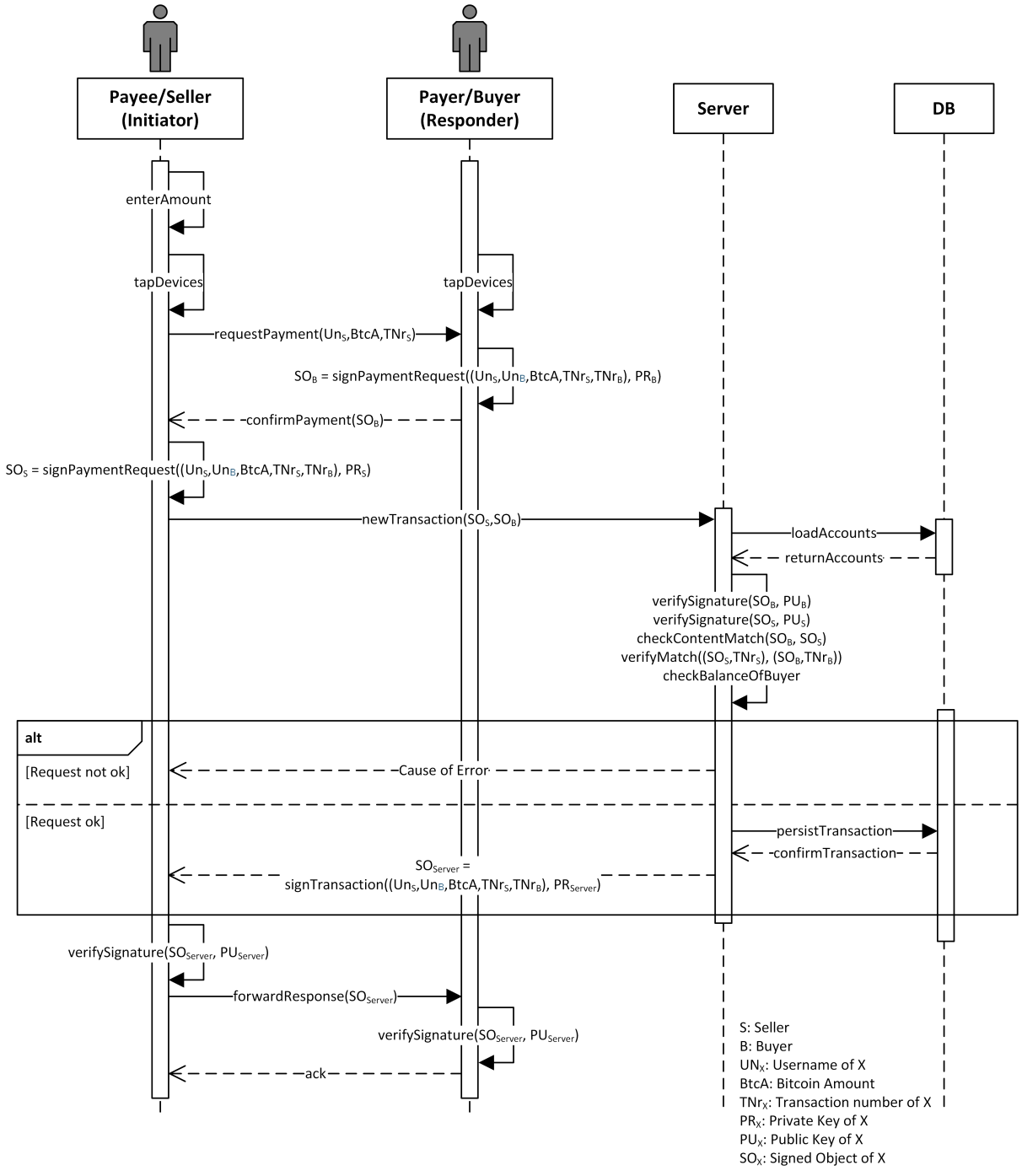

Figure A.1: Payment Protocol of the Previous MBPS

Appendix B

Benchmarks PKI Key Pairs

Table B.1 shows the benchmarks for generating PKI key pairs on the Google Nexus 5. It contains the alternatives considered for the MBPS client. The PKI algorithm is displayed on the left. The columns 1 to 10 contain the time measurements for each corresponding run or repetition. The time is indicated in milliseconds for the corresponding algorithm/run. The last two columns report the arithmetic average and the standard deviation for each algorithm in milliseconds. Figure B.1 depicts the arithmetic average and the standard deviation in a graphical representation.

Algorithm	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
RSA-2048	1586	607	726	849	1482	3205	3510	3373	3746	3696	2278.00	1268.75
ECC brainpoolP224r1	238	205	202	229	213	203	205	201	201	230	212.70	13.44
ECC brainpoolP256r1	244	247	231	172	184	208	264	255	250	250	230.50	30.05
ECC brainpoolP384r1	413	407	415	278	334	391	412	416	420	384	387.00	43.74

Table B.1: Mean and Standard Deviation of PKI Benchmarks

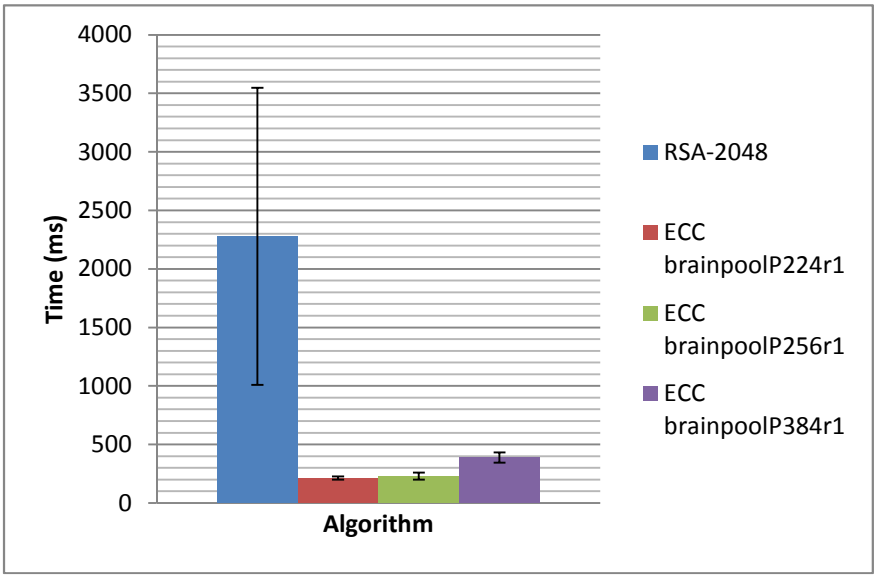

Figure B.1: Mean and Standard Deviation of PKI Benchmarks

Appendix C

Benchmarks Digital Signatures

To chose the best digital signature algorithm with respect to time and output length for the Payment Library, different algorithms have been tested. For the benchmarks measuring the signing performance, an arbitrary byte sequence of length 58 bytes has been chosen. This byte length is the maximum length a payment request can have in the Payment Library, when limiting the usernames to at most 20 characters. All benchmarks have been conducted on a Google Nexus 5 smartphone. Section C.1 shows the RSA benchmarks. Section C.2 shows the benchmarks for Elliptic Curve DSA. Section C.3 compares the signature lengths of the different signature algorithms mentioned below.

C.1 Benchmarks RSA

In this benchmarks, the RSA signature algorithm is evaluated. The RSA signature algorithm requires RSA key pairs. Table C.1 shows the time measurements for signing a 58 byte long byte array. The first row shows the performance of the signature algorithm of the previous MBPS, which used RSA key pairs with a key size of 1024 bits and the SHA-1 hash algorithm. To increase the security level, RSA key pairs with a key size of at least 2048 bits should be used. Furthermore, SHA-256 should be used as hash algorithm instead of SHA-1. The second row shows the performance of this setting. The measurements have been repeated 10 times. The time is indicated in milliseconds for the corresponding algorithm/run. The last two columns report the arithmetic average and the standard deviation for each setting in milliseconds. Figure C.1 depicts the arithmetic average and the standard deviation in a graphical representation.

Table C.2 shows the time measurements for verifying the byte array signed above. The first row shows the performance of RSA-1024 in combination with SHA-1. The second row shows the performance of RSA-2048 in combination with SHA-256. The measurements have been repeated again 10 times. The time is indicated in milliseconds for the corresponding algorithm/run. The last two columns report the arithmetic average and the standard deviation for each setting in milliseconds. Figure C.2 depicts the arithmetic average and the standard deviation in a graphical representation.

Algorithm	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
SHA1withRSA RSA-1024	4	3	2	3	2	3	3	2	3	3	2.80	0.60
SHA256withRSA RSA-2048	17	17	16	17	17	17	17	17	17	17	16.90	0.30

Table C.1: Mean and Standard Deviation of RSA Signature Benchmarks (Signing)

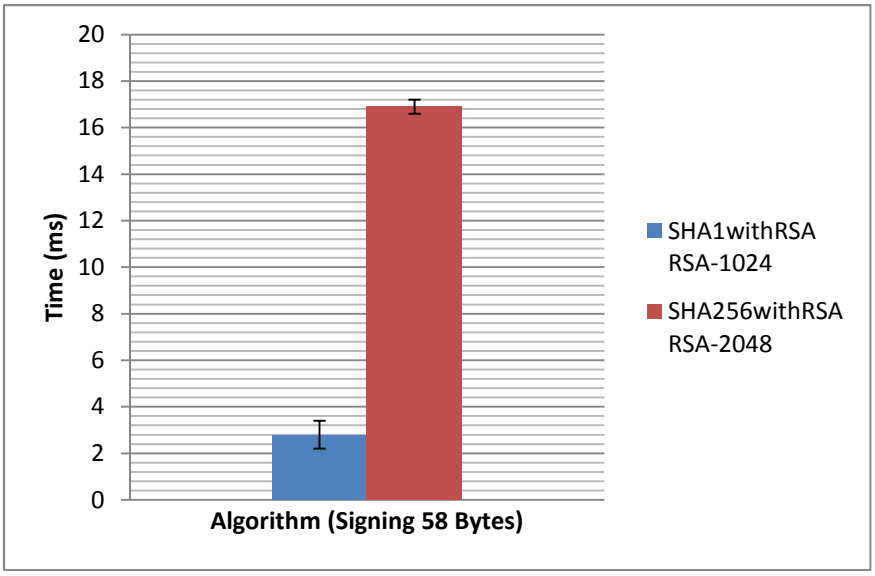

Figure C.1: Mean and Standard Deviation of RSA Signature Benchmarks (Signing)

Algorithm	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
SHA1withRSA RSA-1024	1	0	0	0	0	0	1	0	0	0	0.20	0.40
SHA256withRSA RSA-2048	0	1	0	1	0	1	0	1	0	0	0.40	0.49

Table C.2: Mean and Standard Deviation of RSA Signature Benchmarks (Verifying)

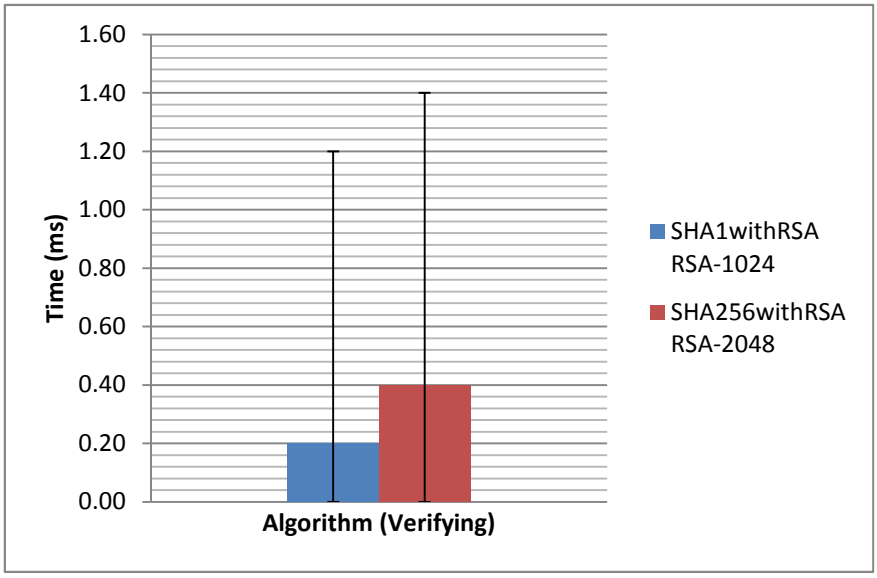

Figure C.2: Mean and Standard Deviation of RSA Signature Benchmarks (Verifying)

C.2 Benchmarks Elliptic Curve DSA

In this benchmarks, the ECDSA signature algorithm is evaluated. ECDSA require ECC key pairs. These benchmarks measure the performance of elliptic curves which provide a comparable security level to RSA-2048. According to [43], this is achieved by a key length between 200 and 250 bits for ECC. The key length of ECC varies between the recommendations of different national agencies and institutions such as NIST [44], BSI [45], and ANSSI [46]. Three named curves have been chosen based on these recommendations and based on what Bouncycastle [47] provides.

Table C.3 shows the time measurements for signing a 58 byte long byte array. For all brainpool ECC algorithms, SHA-256 is used as hash algorithm for the signature. The measurements have been repeated 10 times. The time is indicated in milliseconds for the corresponding algorithm/run. The last two columns report the arithmetic average and the standard deviation for each setting in milliseconds. Figure C.3 depicts the arithmetic average and the standard deviation in a graphical representation.

Algorithm	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
SHA256withECDSA brainpoolP224r1	8	7	7	6	7	7	6	7	6	7	6.80	0.60
SHA256withECDSA brainpoolP256r1	12	8	9	9	9	8	9	9	9	9	9.10	1.04
SHA256withECDSA brainpoolP384r1	26	27	25	25	26	26	25	26	25	26	25.70	0.64

Table C.3: Mean and Standard Deviation of ECDSA Signature Benchmarks (Signing)

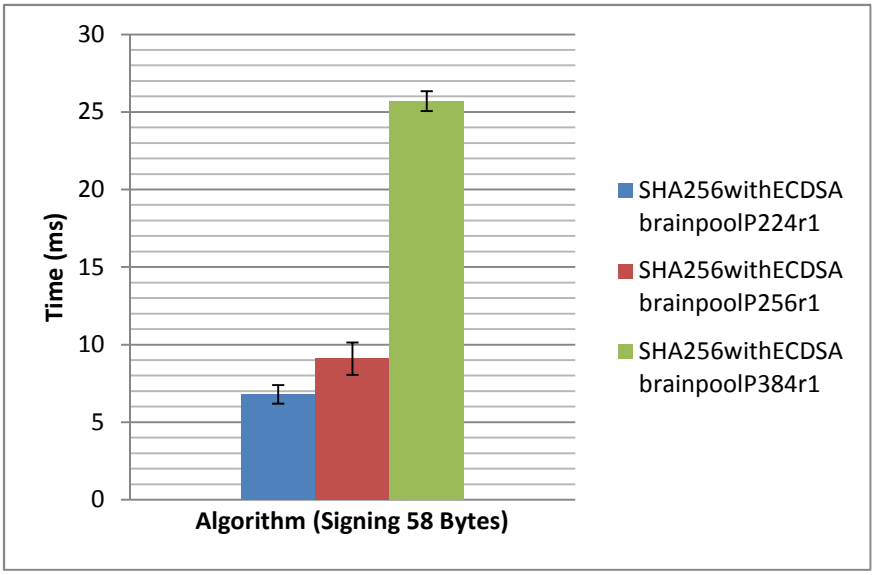

Figure C.3: Mean and Standard Deviation of ECDSA Signature Benchmarks (Signing)

Table C.4 shows the time measurements for verifying the byte array signed above. The measurements have been repeated again 10 times. The time is indicated in milliseconds for the corresponding algorithm/run. The last two columns report the arithmetic average and the standard deviation for each setting in milliseconds. Figure C.4 depicts the arithmetic average and the standard deviation in a graphical representation.

Algorithm	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
SHA256withECDSA brainpoolP224r1	4	3	4	4	4	4	3	4	4	7	4.10	1.04
SHA256withECDSA brainpoolP256r1	5	5	5	5	5	5	5	5	5	5	5.00	0.00
SHA256withECDSA brainpoolP384r1	14	14	14	15	14	20	14	15	14	14	14.80	1.78

Table C.4: Mean and Standard Deviation of ECDSA Signature Benchmarks (Verifying)

C.3 Comparison of Signature Lengths

Table C.5 shows the signature lengths of the signature algorithms RSA and ECDSA in different settings. The variable signature length with ECDSA is due to the ANS.1 [50] encoding.

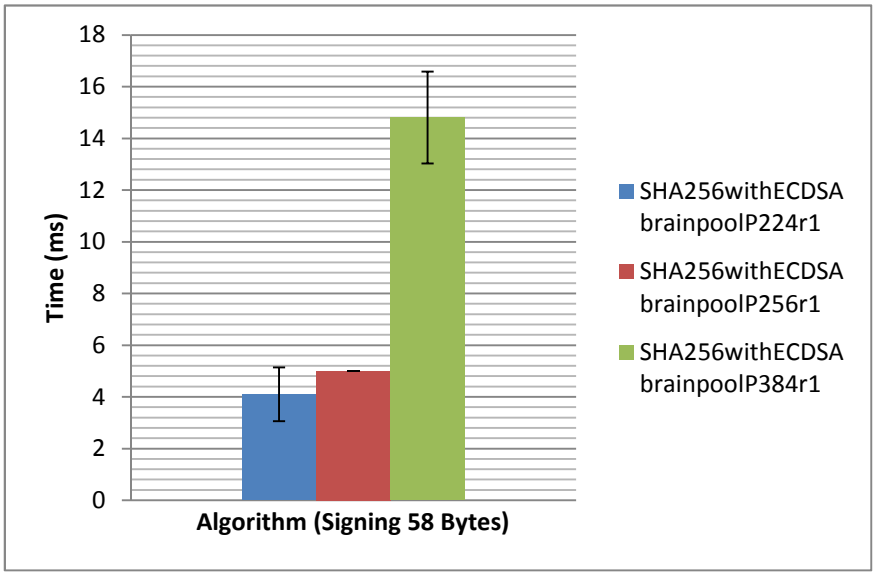

Figure C.4: Mean and Standard Deviation of ECDSA Signature Benchmarks (Verifying)

Algorithm	Signature Length (Bytes)
SHA1withRSA RSA-1024	128
SHA256withRSA RSA-2048	256
SHA256withECDSA brainpoolP224r1	62-64
SHA256withECDSA brainpoolP256r1	70-72
SHA256withECDSA brainpoolP384r1	102-103

Table C.5: Signature Length with RSA and ECDSA

Appendix D

Performance Measurement Results

This appendix contains the performance measurements of the previous MBPS and the new Payment Library. A detailed analysis is provided in Chapter 4.

Each table in this appendix shows the time of performance of every atomic step involved in the given payment protocol. The values in all tables are indicated in milliseconds. The payee (or seller) role involves other steps than the payer (or buyer) role. The measurements have been repeated 10 times, which is represented by the columns 1 to 10 respectively. The last two columns of every table in this appendix show the arithmetic average and the standard deviation of the given atomic step. The last rows contain the total time required for a payment to complete. It is important to mention that the total time is not the sum of all atomic steps. Minor object accesses and manipulations have not been considered. However, the total time indicates the time required from the start to the end of a payment process.
The measurements have been conducted with two different devices: Google Nexus 5 and Google Nexus 10. For the payer role measurements with the Nexus 5, the Nexus 10 has been used and measured in the payee role and vice versa.

Section D.1 contains the performance measurements of the previous MBPS payments. Section D.2 contains the performance measurements of the payments in the new Payment Library, in particular the POS use-case. The P2P use-case is only supported in the new Payment Library. The according performance measurements are shown in section D.3.

D.1 Results Previous MBPS

Table D.1 shows the performance measurements for the Nexus 5 in the payee role. Table D.2 shows the performance measurements for the Nexus 5 in the payer role. Table D.3 shows the performance measurements for the Nexus 10 in the payee role. Table D.4 shows the performance measurements for the Nexus 10 in the payer role.

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	80	83	88	67	62	63	52	52	61	42	65.00	14.06
Create Payment Request	2	4	5	2	2	2	2	10	3	3	3.50	2.38
Serialize Payment Request	17	21	19	15	23	14	20	23	20	48	22.00	9.13
Send Payment Request	184	188	173	199	175	196	193	204	192	196	190.00	9.57
Receive Payment Request	584	601	487	668	667	655	675	587	456	609	598.90	71.95
Deserialize Payment Request	8	22	14	23	18	20	20	40	3	20	18.80	9.34
Create Transaction Request	21	49	25	35	33	43	36	36	7	53	33.80	12.87
Sign Transaction Request	39	16	30	34	15	30	37	16	15	15	24.70	9.65
Receive Server Response	822	900	727	995	943	897	870	920	797	806	867.70	75.64
Verify Server Response	44	59	38	32	48	41	45	30	37	31	40.50	8.50
Serialize Server Response	5	6	6	6	5	5	6	5	5	7	5.60	0.66
Send Server Response	269	266	273	282	273	251	222	261	254	260	261.10	15.78
Receive ACK	0	0	1	0	1	0	1	1	0	3	0.70	0.90
Total Time	2172	2262	1955	2421	2347	2266	2241	2235	1925	2198	2202.20	147.63

Table D.1: Performance Measurements Previous MBPS – Payee (Seller) – Nexus 5

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	47	4	5	4	5	3	2	6	4	4	8.40	12.91
Receive Payment Request	180	155	146	211	154	167	166	141	178	137	163.50	21.06
Deserialize Payment Request	4	5	8	35	6	55	9	7	4	12	14.50	16.06
Polling	70	70	73	73	77	87	75	74	68	73	74.00	5.00
Sign Payment Request	11	12	19	11	46	17	62	22	11	18	22.90	16.37
Serialize Payment Request	3	5	8	2	8	2	3	5	1	3	4.00	2.32
Send Payment Request	0	2	2	1	1	1	1	1	0	2	1.10	0.70
Receive Server Response	1270	1514	1533	1518	1429	1583	1456	1381	1446	1481	1461.10	84.07
Deserialize Server Response	15	22	9	2	24	15	18	7	15	17	14.40	6.39
Verify Server Response	53	25	38	18	14	29	43	45	33	42	34.00	11.86
Send Ack	0	0	0	1	0	0	1	1	1	1	0.50	0.50
Total	1666	1814	1841	1877	1765	1959	1836	1690	1761	1790	1799.90	82.03

Table D.2: Performance Measurements Previous MBPS – Payer (Buyer) – Nexus 5

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	103	65	40	66	52	43	32	47	69	40	55.70	19.78
Create Payment Request	3	3	0	2	2	1	2	1	3	0	1.70	1.10
Serialize Payment Request	34	16	9	26	18	10	20	10	27	8	17.80	8.47
Send Payment Request	143	135	148	219	135	227	153	142	143	144	158.90	32.49
Receive Payment Request	556	676	738	638	637	682	678	692	692	615	660.40	48.01
Deserialize Payment Request	13	19	14	16	11	11	16	15	14	14	14.30	2.28
Create Transaction Request	20	12	17	18	15	27	18	13	20	18	17.80	3.99
Sign Transaction Request	24	11	14	24	25	13	25	11	28	12	18.70	6.63
Receive Server Response	433	550	535	605	541	614	533	435	465	589	530.00	62.75
Verify Server Response	14	16	11	13	21	16	20	19	13	28	17.10	4.78
Serialize Server Response	8	6	6	6	6	9	8	7	4	8	6.80	1.40
Send Server Response	388	362	345	304	346	349	361	339	340	349	348.30	20.15
Receive ACK	8	3	1	3	4	3	1	1	1	1	2.60	2.11
Total Time	1820	1958	1964	2041	1892	2075	1980	1815	1903	1923	1937.10	80.52

Table D.3: Performance Measurements Previous MBPS – Payee (Seller) – Nexus 10

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	40	30	12	6	9	11	10	7	5	3	13.30	11.38
Receive Payment Request	189	192	188	203	186	202	190	213	200	222	198.50	11.26
Deserialize Payment Request	16	19	18	31	17	21	25	23	26	27	22.30	4.67
Polling	72	85	60	77	71	78	72	78	63	63	71.90	7.57
Sign Payment Request	21	32	15	32	16	23	20	14	14	24	21.10	6.44
Serialize Payment Request	1	2	1	1	1	2	1	1	1	1	1.20	0.40
Send Payment Request	0	0	0	0	1	0	0	0	1	1	0.30	0.46
Receive Server Response	1674	1782	1498	1944	1889	1823	1816	1780	1476	1697	1737.90	146.38
Deserialize Server Response	4	5	4	4	5	2	1	4	3	2	3.40	1.28
Verify Server Response	16	16	17	18	16	16	3	19	18	11	15.00	4.49
Send Ack	2	0	2	0	1	0	1	1	1	1	0.90	0.70
Total	2035	2163	1815	2316	2212	2178	2139	2140	1808	2052	2085.80	156.00

Table D.4: Performance Measurements Previous MBPS – Payer (Buyer) – Nexus 10

D.2 Results Payment Library – POS Use-Case

Table D.5 shows the performance measurements for the Nexus 5 in the payee role. Table D.6 shows the performance measurements for the Nexus 5 in the payer role. Table D.7 shows the performance measurements for the Nexus 10 in the payee role. Table D.8 shows the performance measurements for the Nexus 10 in the payer role.

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	102	74	95	84	88	84	82	72	61	109	85.10	13.60
Create Payment Request	0	1	2	1	5	0	0	0	1	4	1.40	1.69
Serialize Payment Request	65	0	1	0	0	1	1	1	0	1	7.00	19.34
Send Payment Request	247	225	343	242	198	233	218	348	173	343	257.00	60.87
Receive Payment Request	46	45	45	48	46	44	45	45	45	51	46.00	1.95
Deserialize Payment Request	1	1	1	1	1	0	1	0	1	1	0.80	0.40
Create Transaction Request	0	0	0	0	0	1	0	0	1	0	0.20	0.40
Sign Transaction Request	11	10	26	16	10	15	18	20	13	20	15.90	4.93
Receive Server Response	581	574	665	575	525	579	560	649	598	779	608.50	68.76
Verify Server Response	6	5	8	6	6	6	11	5	13	9	7.50	2.58
Serialize Server Response	0	0	0	0	0	0	0	0	0	0	0.00	0.00
Send Server Response	218	234	216	253	215	264	268	226	249	239	238.20	18.70
Receive ACK	0	0	0	0	0	0	0	0	0	0	0.00	0.00
Total Time	1294	1201	1439	1245	1112	1249	1276	1426	1171	1625	1303.80	144.47

Table D.5: Performance Measurements Payment Library – POS Use-Case – Payee (Seller) – Nexus 5

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	0	15	25	14	16	17	19	29	22	16	17.30	7.35
Receive Payment Request	97	140	92	115	153	129	101	161	106	129	122.30	22.75
Deserialize Payment Request	1	1	1	1	2	0	1	1	1	1	1.00	0.45
Polling	4	3	3	5	5	5	4	3	7	14	5.30	3.13
Sign Payment Request	14	13	13	12	11	14	14	14	11	39	15.50	7.92
Serialize Payment Request	0	0	0	0	0	0	1	0	2	0	0.30	0.64
Send Payment Request	50	50	51	67	46	52	54	53	54	76	55.30	8.64
Receive Server Response	870	860	940	860	890	930	890	870	790	840	874.00	40.79
Deserialize Server Response	8	6	1	8	3	2	3	4	5	3	4.30	2.28
Verify Server Response	44	46	7	47	23	41	13	22	43	26	31.20	14.01
Send Ack	62	63	61	66	74	64	113	67	62	64	69.60	14.89
Total	1151	1202	1192	1198	1221	1257	1208	1221	1104	1207	1196.10	39.73

Table D.6: Performance Measurements Payment Library – POS Use-Case – Payer (Buyer) – Nexus 5

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	127	89	101	84	90	82	121	145	92	94	102.50	20.09
Create Payment Request	3	10	1	3	4	9	3	8	9	3	5.30	3.13
Serialize Payment Request	1	1	4	2	1	2	2	1	1	2	1.70	0.90
Send Payment Request	181	154	150	192	163	160	160	190	150	276	177.60	36.01
Receive Payment Request	28	32	28	47	28	36	33	34	31	28	32.50	5.55
Deserialize Payment Request	0	1	0	2	1	2	2	2	0	1	1.10	0.83
Create Transaction Request	0	0	0	1	1	1	0	0	0	0	0.30	0.46
Sign Transaction Request	12	19	16	18	24	22	29	27	12	13	19.20	5.84
Receive Server Response	711	712	789	674	724	716	714	693	646	635	701.40	41.30
Verify Server Response	10	10	11	15	12	11	12	11	11	9	11.20	1.54
Serialize Server Response	0	0	0	0	0	0	0	0	0	0	0.00	0.00
Send Server Response	209	225	151	222	219	272	241	212	210	213	217.40	28.64
Receive ACK	0	0	1	1	0	1	0	0	0	1	0.40	0.49
Total Time	1303	1299	1279	1295	1299	1350	1354	1343	1201	1311	1303.40	41.89

Table D.7: Performance Measurements Payment Library – POS Use-Case – Payee (Seller) – Nexus 10

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	11	14	27	16	18	18	18	16	11	20	16.90	4.41
Receive Payment Request	154	102	171	99	93	96	83	207	78	207	129.00	48.26
Deserialize Payment Request	1	0	2	1	1	1	0	2	1	3	1.20	0.87
Polling	28	10	34	16	8	7	12	29	6	23	17.30	9.83
Sign Payment Request	19	18	25	19	21	19	18	18	17	26	20.00	2.93
Serialize Payment Request	0	1	1	1	0	0	0	1	0	1	0.50	0.50
Send Payment Request	103	96	114	112	90	119	110	94	70	87	99.50	14.24
Receive Server Response	750	745	850	755	670	780	780	800	805	955	789.00	71.13
Deserialize Server Response	3	6	1	3	3	3	4	5	2	4	3.40	1.36
Verify Server Response	22	28	19	34	32	30	23	30	26	28	27.20	4.47
Send Ack	89	93	95	112	94	98	94	108	88	107	97.80	7.90
Total	1181	1111	1338	1166	1033	1171	1143	1314	1105	1465	1202.70	123.55

Table D.8: Performance Measurements Payment Library – POS Use-Case – Payer (Buyer) – Nexus 10

D.3 Results Payment Library – P2P Use-Case

Table D.9 shows the performance measurements for the Nexus 5 in the payer role. Table D.10 shows the performance measurements for the Nexus 5 in the payee role. Table D.11 shows the performance measurements for the Nexus 10 in the payer role. Table D.12 shows the performance measurements for the Nexus 10 in the payee role.

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	108	378	101	94	82	67	114	117	131	131	132.30	84.13
Send Request Username	77	181	156	58	71	61	201	65	75	64	100.90	52.62
Receive Username	0	1	0	0	0	1	0	1	0	0	0.30	0.46
Deserialize Username	0	0	0	1	0	0	0	0	0	0	0.10	0.30
Create Transaction Request	79	35	36	36	38	84	31	48	44	40	47.10	17.82
Sign Transaction Request	11	9	10	10	12	21	9	12	11	10	11.50	3.32
Receive Server Response	527	585	538	517	575	626	542	525	515	1225	617.50	205.27
Verify Server Response	6	8	6	6	8	5	6	6	6	7	6.40	0.92
Serialize Server Response	0	0	0	0	0	0	0	0	0	0	0.00	0.00
Send Server Response	138	126	209	222	146	239	179	193	225	221	189.80	38.52
Receive ACK	0	1	0	0	0	0	0	0	0	0	0.10	0.30
Total Time	1030	1382	1174	1004	995	1166	1143	1033	1075	1766	1176.80	224.95

Table D.9: Performance Measurements Payment Library – P2P Use-Case – Payer – Nexus 5

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	19	18	17	15	19	20	17	22	22	18	18.70	2.10
Receive Request Username	115	212	106	96	89	115	119	130	127	137	124.60	32.36
Serialize Username	1	0	1	0	0	0	0	0	1	0	0.30	0.46
Send Username	4	0	0	0	1	0	1	0	0	0	0.60	1.20
Receive Server Response	887	853	926	942	891	1013	823	869	917	920	904.10	50.08
Deserialize Server Response	6	9	11	5	3	1	11	6	9	9	7.00	3.19
Verify Server Response	49	49	53	42	50	7	52	17	47	49	41.50	15.18
Send Ack	36	33	24	23	24	25	34	22	25	30	27.60	4.88
Total	1117	1174	1138	1123	1077	1181	1057	1066	1148	1163	1124.40	42.61

Table D.10: Performance Measurements Payment Library – P2P Use-Case – Payee – Nexus 5

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	107	168	82	53	75	82	89	103	88	87	93.40	28.57
Send Request Username	89	115	76	82	66	81	91	120	102	103	92.50	16.41
Receive Username	0	1	0	0	0	0	0	0	0	0	0.10	0.30
Deserialize Username	0	0	0	0	1	0	0	0	0	0	0.10	0.30
Create Transaction Request	61	69	67	54	63	73	60	66	94	92	69.90	12.57
Sign Transaction Request	11	19	11	17	16	13	10	19	13	18	14.70	3.32
Receive Server Response	649	592	711	709	668	732	615	587	660	646	656.90	47.38
Verify Server Response	10	10	10	10	9	12	10	11	11	13	10.60	1.11
Serialize Server Response	0	0	0	0	0	0	0	0	0	0	0.00	0.00
Send Server Response	188	180	155	151	157	146	189	156	154	188	166.40	16.62
Receive ACK	1	1	1	1	1	1	1	0	1	1	0.90	0.30
Total Time	1239	1258	1196	1182	1156	1234	1157	1141	1224	1258	1204.50	41.69

Table D.11: Performance Measurements Payment Library – P2P Use-Case – Payer – Nexus 10

Atomic Step	1	2	3	4	5	6	7	8	9	10	Mean	Std. Dev.
Init NFC	12	50	40	19	18	17	18	16	21	21	23.20	11.39
Receive Request Username	102	444	141	95	91	82	246	119	143	131	159.40	104.72
Serialize Username	0	1	0	0	1	0	4	0	0	0	0.60	1.20
Send Username	0	0	1	0	0	0	0	0	0	1	0.20	0.40
Receive Server Response	738	765	774	711	740	888	715	714	714	1412	817.10	204.61
Deserialize Server Response	6	2	10	1	4	3	6	4	7	5	4.80	2.48
Verify Server Response	29	26	19	10	10	28	54	54	71	79	38.00	23.53
Send Ack	35	35	76	96	72	86	34	35	37	37	54.30	23.78
Total	922	1323	1061	932	936	1104	1077	942	993	1687	1097.70	227.93

Table D.12: Performance Measurements Payment Library – P2P Use-Case – Payee – Nexus 10

Appendix E

Performance Payment Library (P2P Use-Case)

The results shown below are based on the new payment protocol implemented in the Payment Library (see Section 3.2.2). The results apply to the P2P use-case of the new Payment Library. The performance measurements of the POS use-case are already covered in Chapter 4. Below, the performance measurements of the payer role are discussed first, followed by the payee role. The setup is the same as described in Section 4.1.

Payer Role

Figure E.1 shows the performance measurements for the atomic steps applying to the payer role. The last bar is not an atomic step but the total time required to complete the whole payment. This figure shows the arithmetic average and the standard deviation for the total time and for each atomic step and device, based on the measurements shown in Table D.9 for the Nexus 5 and in Table D.11 for the Nexus 10. The first step, i.e., *Init NFC*, measures the NFC handshake, from the time a device has been discovered until the handshake completes by the second handshake message returned by the payee. The payer then requests the payee's username by sending a message of 4 bytes (including the 2 bytes NFC Library headers and the 1 Byte Payment Library header). As mentioned in Chapter 4, it cannot be measured how long the Initiator has been writing and how long reading data. Therefore, the step *Send Request Username* not only measures the time required to send these 4 bytes, but also measures the time the payee processes the message and returns his username. In these measurements, the payer's and payee's username have a length of 5 characters. Therefore, the payee returns 8 bytes (including the headers) in this step. The *Receive Username* step measures the time starting when the payer has received the payee's username until this message is processed. The payer then deserializes the payee's username (step *Deserialize Username*). Using the payee's username and the amount the payer wants to transfer, he creates a payment request (step *Create Transaction Request*) and signs it (step *Sign Transaction Request*). The signed request is then sent to the server over HTTP. The step *Receive Server Response* measures the time when the server request was about to be launched until the server response arrives. The payer then verifies the

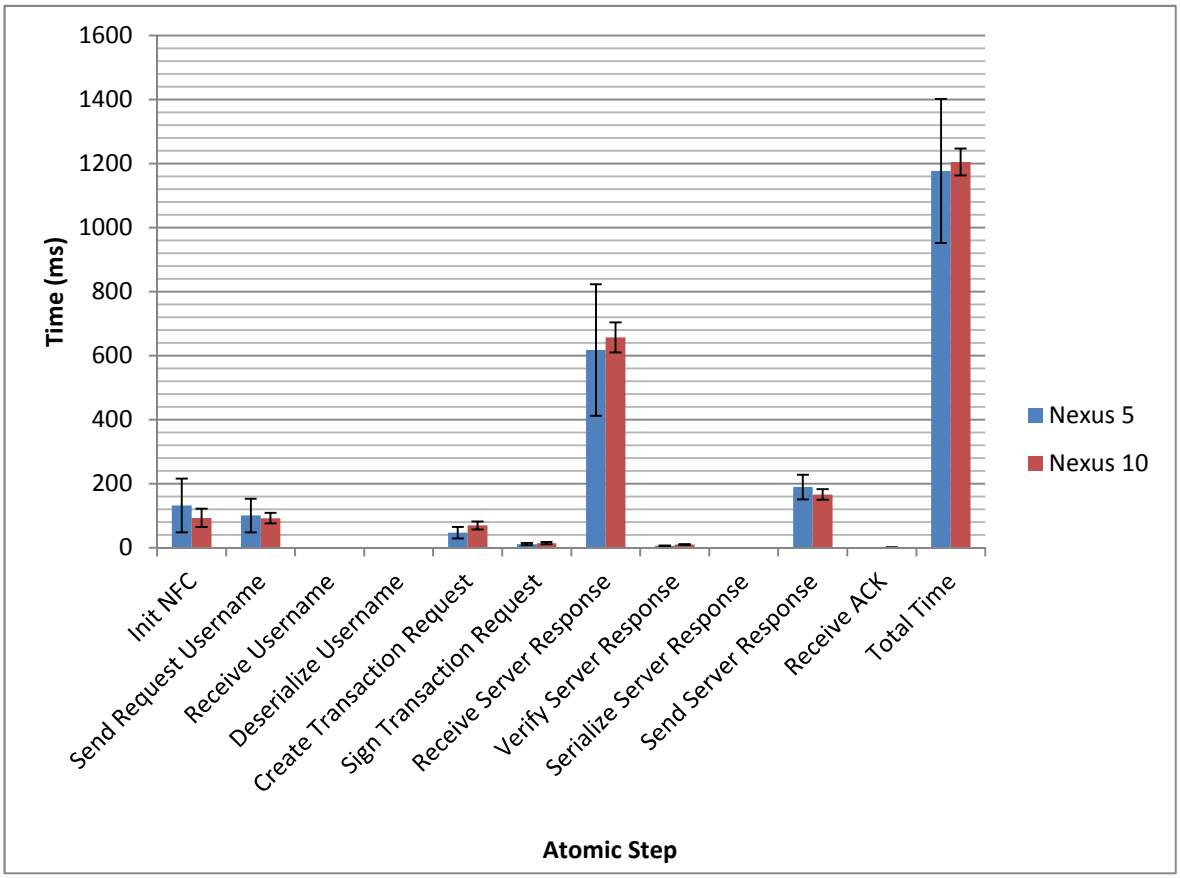

Figure E.1: Mean and Standard Deviation Payment Library – P2P use-case – Payer

signature (step *Verify Server Response*) and serializes the server response (step *Serialize Server Response*) in order to forward it to the payee. Including the packet header, the message has a length of 106 bytes in the measured case. These bytes are then sent to the payee. Similar to the *Send Username Request* step, the *Send Server Response* step not only measures the time required to send the message, but also measures the time until the response is returned. Since the ACK message fits into one fragment (4 bytes including the headers), this step measure as well the processing on the payee's side and receiving the data from the payee. The *Receive ACK* step only measure the time between the ACK message has arrived until it is processed and the protocol terminates.

The most time consuming step is by far the server call. This holds also for the measurements of the POS use-case described in Chapter 4.

Payee Role

Figure E.2 shows the performance measurements for the atomic steps applying to the payee role. The arithmetic average and the standard deviation for the total time and for each atomic step and device are based on the measurements shown in Table D.10 for the Nexus 5 and in Table D.12 for the Nexus 10. The first step, i.e., *Init NFC*, measures

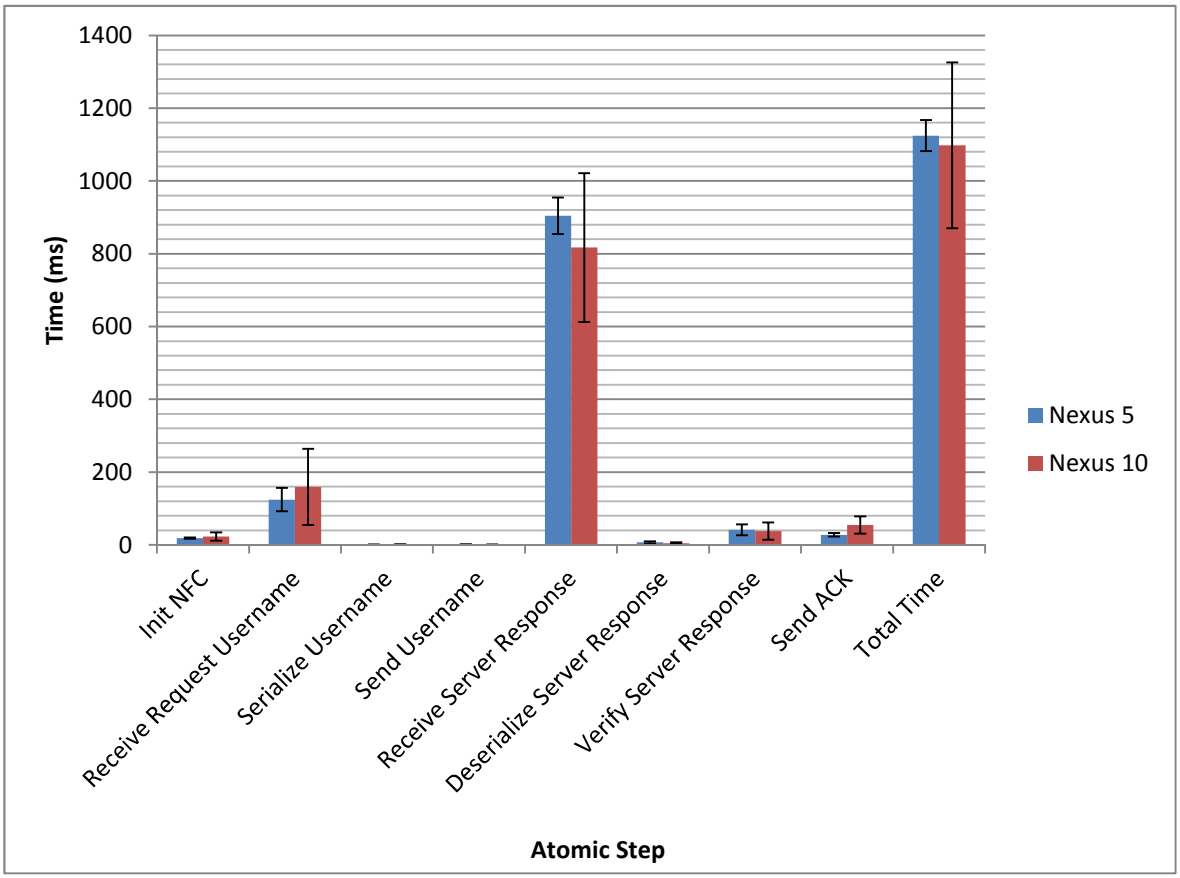

Figure E.2: Mean and Standard Deviation Payment Library – P2P use-case – Payee

the NFC handshake, from the time the Android system has received the complete AID message from the Initiator until the second handshake message is about to be returned. This measurement neither considers the time the Responder has been reading the first handshake message (i.e., the AID) nor the time required to return the second handshake message. The following *Receive Request Username* step measures the time beginning when the handshake message was ready to be returned until the payer's request for the username has completely been received. The payee then serializes his username (step *Serialize Username*) and returns it to the payer (step *Send Username*). This step only measures the time required after the username has been serialized until the message is ready to be returned to the payer. The time required to write the data is not considered in this measurement but in the *Receive Server Response* step. The latter indicates the time elapsed since the serialized username was about to be sent until the complete server response arrives. This comprises the steps 3 to 9 and a portion of steps 2 and 10 shown in Figure E.1. The payee then deserializes the server response (step *Deserialize Server Response*) and verifies the signature (step *Verify Server Response*). The time indicated in the step *Send ACK* reports the time elapsed after the signature has been verified until the acknowledgement message can be returned and the protocol terminates on the payee's side. The time required to write that data is not considered in the measurement.

Since the measurements of the payee role reported in Figure E.2 do not contain the

time to send or receive data, Figure E.1 is more significant. However, there are as well measurement errors, since a proper separation between reading and writing data is not possible.
When comparing the total times reported in Figure E.1 and E.2, it stands out that the time reported in the payer role is higher as compared to the payee role.

The measurements conducted with the Nexus 5 in the payer role and the Nexus 10 in the payee role resulted in a total time of 1176.8ms (± 225ms) for the payer and 1097.7ms (± 228ms) for the payee. The contrary case with the Nexus 10 in the payer role and the Nexus 5 in the payee role resulted in a total time of 1204.5ms (± 41.7ms) for the payer and 1124.4ms (± 42.6ms) for the payee. The difference between the payer and payee is due to the fact that the payer measures the time before the NFC handshake is conducted until the ACK message has been received. The payee on the other hand starts measuring the total time after the AID message has been received and stops before the ACK message is returned.

Compared to the POS use-case performance measurements of the new Payment Library (see Section 4.3), the P2P use-case is faster by approximately 100ms with regard to the total time. The reason is that in the POS use-case, the first message pair exchanged between the Initiator and the Responder after the handshake is larger (sending 19 bytes and receiving 107 bytes) as compared to the P2P use-case (sending 4 bytes and receiving 8 bytes). Furthermore, in the POS use-case the Responder needs to deserialize the message and sign his response, which also consumes time. These steps are not applied in the P2P use-case.

Appendix F

Installation Guidelines

F.1 Custom Serialization

Prerequisites

- Java JDK 7 (`http://www.oracle.com/technetwork/java/javase/downloads/index.html`)
- Maven 3.2.1 (`http://maven.apache.org/download.cgi`)

Installation Guidelines

1. Extract the archive *CustomSerialization.zip* (see Appendix H) to a directory on your hard drive. (Alternatively, you can clone the GitHub repository `https://github.com/jetonmemeti/custom-serialization`).
2. Open the Command Line Interface of your Operating System (e.g., Terminal) and navigate to the root folder on your hard drive where you extracted the archive or cloned the GitHub repository to.
3. Run `mvn install` to install this library to your local maven repository.

F.2 Android KitKat NFC Library

Prerequisites

- Java JDK 7 (`http://www.oracle.com/technetwork/java/javase/downloads/index.html`)
- Eclipse 4.3.2 (`http://www.eclipse.org/downloads/`)
- Maven 3.2.1 (`http://maven.apache.org/download.cgi`)
- Android SDK (`http://developer.android.com/sdk/index.html`). In the *Android SDK Manager* you need to install at least the *Android SDK Tools*, the *Android SDK Platform-tools*, and *Android 4.4.2 (API 19)*. For the API 19 the following items are mandatory:
  - SDK Platform
  - Google APIs
  - Glass Development Kit Preview
- Having set up Eclipse according to `http://developer.android.com/sdk/installing/index.html`

Installation Guidelines

1. Install two third-party libraries to your local maven repository:
   (a) You need the *maven-android-sdk-deployer* (`https://github.com/mosabua/maven-android-sdk-deployer`). Follow the instructions there - only platform 4.4 needs to be installed. (This library requires the *properties-maven-plugin* (`http://search.maven.org/#artifactdetails/org.codehaus.mojo/properties-maven-plugin/1.0-alpha-2/maven-plugin`) to be installed to your local maven repository. Otherwise, the installation of *maven-android-sdk-deployer* will fail.)
   (b) You also need the Android Library for the ACR122u, even if you do not plan to use it in your application. You can download the JAR from `http://www.acs.com.hk/download-driver-unified/5102/ACS-LIB-Android-112-A.zip`. You can also find this jar in the *AndroidKitKat-NFCLibrary.zip*.
2. Extract the archive *AndroidKitKatNFCLibrary.zip* (see Appendix H) to a directory on your hard drive. (Alternatively, you can clone the GitHub repository `https://github.com/jetonmemeti/android-kitkat-nfc-library`.)
3. In Eclipse go to `File -> Import -> Maven -> Existing Maven Project`. In the appearing dialog under `Root Directory` enter the path to the extracted archive.

4. Select the project `AndroidKitKatNFCLibrary` and click on `Finish`.

5. Run `mvn install` on the root folder of this project to install it to your local maven repository. Now you can use this library in your project by adding its *groupId*, *artifactId*, and *version* (see this project's pom.xml) to the POM of your project.

F.3 Android NFC Payment Library

Prerequisites

- See the Prerequisites section of Appendix F.2
- Having successfully installed the *Custom Serialization* library
- Having successfully installed the *Android KitKat NFC Library*

Installation Guidelines

- The installation of this library is similar to the installation of the *Android KitKat NFC Library* (see Appendix F.2). Use the archive *AndroidNFCPaymentLibrary.zip* (see Appendix H) instead. (Alternatively, you can clone the GitHub repository `https://github.com/jetonmemeti/android-nfc-payment-library`.)

If you want to use this library in your own Android project, there are three important things you need to add to your project in order for the NFC to work properly (besides of adding the dependency to this library in your POM).

- Copy the file *apduservice.xml* (which can be found in *AndroidKitKatNFCLibrary.zip*, see Appendix H) to `<project root folder>\res\xml\`.
- In `<project root folder>\res\values\strings.xml` add the following lines:

```
<!-- APDU SERVICE -->
<string name="aiddescription">ch.uzh.csg.nfclib</string>
<string name="servicedesc">Android KitKat NFC Library</string>
```

- In the `AndroidManifest.xml` add the following:

```
<uses-sdk android:minSdkVersion="19" android:targetSdkVersion="19"
   />
<uses-feature android:name="android.hardware.nfc" android:required=
   "true" />
<uses-permission android:name="android.permission.NFC" />
```

  Inside the `<application>` tag add:

```
<service android:name="ch.uzh.csg.nfclib.HostApduServiceNfcLib"
    android:exported="true" android:permission="android.permission.
    BIND_NFC_SERVICE">
         <intent-filter>
                 <action android:name="android.nfc.cardemulation.
                    action.HOST_APDU_SERVICE" />
         </intent-filter>
         <meta-data android:name="android.nfc.cardemulation.
            host_apdu_service" android:resource="@xml/apduservice" /
            >
</service>
```

Once this is done, you can use this library in your project by adding its *groupId*, *artifactId*, and *version* (see this project's pom.xml) to the POM of your project.

F.4 Sample Payment Project

Prerequisites

- See the Prerequisites section of Appendix F.2
- Having successfully installed the *Android NFC Payment Library*

Installation Guidelines

1. Extract the archive *SamplePaymentProject.zip* (see Appendix H) to a directory on your hard drive. (Alternatively, you can clone the GitHub repository `https://github.com/jetonmemeti/SamplePaymentProject`.)
2. In Eclipse go to `File -> Import -> Android -> Existing Android Code Into Workspace`. In the appearing dialog under `Root Directory` enter the path to the extracted archive.
3. Under `Project to Import` select the `SapmplePaymentProject`. Consider renaming the project (see `New Project Name`), otherwise it will be called `MainActivity`.
4. Click on `Finish`.
5. Right click on the project and select `Properties -> Configure -> Convert to Maven Project`.
6. Right click on the imported project and select `Run as -> Android Project`.
7. Select a plugged in Android device to run the application on or alternatively create an Android Virtual Device (see `http://developer.android.com/tools/devices/index.html`).

F.5 Signature Benchmarks

Prerequisites:

- Java JDK 7 (`http://www.oracle.com/technetwork/java/javase/downloads/index.html`)
- Eclipse 4.3.2 (`http://www.eclipse.org/downloads/`)
- Android SDK (`http://developer.android.com/sdk/index.html`)
- Having set up Eclipse according to `http://developer.android.com/sdk/installing/index.html`

How to run the benchmarks:

1. Extract the archive *AndroidSignaturesTests.zip* (see Appendix H) to a directory on your hard drive. (Alternatively, you can clone the GitHub repository `https://github.com/jetonmemeti/android-signature-tests`).
2. In Eclipse, go to `File -> Import -> Android -> Existing Android Code Into Workspace`. In the appearing dialog under `Root Directory` enter the path to the extracted archive.
3. Under `Project to Import` select the `AndroidSignaturesTests`. Consider renaming the project (see `New Project Name`), otherwise it will be called `MainActivity`.
4. Click on `Finish`.
5. Right click on the project and select `Properties`. In the prompt window select `Java Build Path`. Select `Add External JARs...` in the `Libraries` tab. Now navigate to the `libs` directory in the root directory of the imported project and select these two JARs: *sc-light-jdk15on-1.47.0.2.jar* and *scprov-jdk15on-1.47.0.2.jar*.
6. Right click on the imported project and select `Run as -> Android Project`.
7. Select a plugged in Android device to run the application on or alternatively create an Android Virtual Device (see `http://developer.android.com/tools/devices/index.html`).

Appendix G

Used Libraries

The following third-party libraries have been used for the implementation of the AndroidKitKatNFCLibrary:

- **JUnit 4.11**
  The JUnit test framework has been used to test the code. (URL: `http://mvnrepository.com/artifact/junit/junit/4.11`)
- **ACS Android Library 1.1.2**
  The ACS Android Library has been used as a driver for the ACR122u USB NFC Reader. (URL: `http://www.acs.com.hk/download-driver-unified/5102/ACS-LIB-Android-112-A.zip`)
- **Powermock Module JUnit 4 1.5.5** and **Powermock Api Mockito 1.5.5**
  These two libraries have been used to test the code by mocking NFC data. (URL: `http://mvnrepository.com/artifact/org.powermock/powermock-module-junit4/1.5.5` and `http://mvnrepository.com/artifact/org.powermock/powermock-api-mockito/1.5.5`)
- **Maven Android SDK Deployer**
  This library has been used to implement the AndroidNFCPaymentLibrary as a regular Java Project instead of a Android Project. This simplifies testing the library. (URL: `https://github.com/mosabua/maven-android-sdk-deployer`)

The following third-party libraries have been used for the implementation of the AndroidNFCPaymentLibrary:

- **JUnit 4.11**
- **Powermock Module JUnit 4 1.5.5** and **Powermock Api Mockito 1.5.5**
- **Maven Android SDK Deployer**

The following third-party libraries have been used for the implementation of the CustomSerialization:

- **JUnit 4.11**
- **Apache Commons Codec 1.8**
  The Base64 encoder of this library is used to convert private and public keys to regular strings in order to persist them. (URL: `http://mvnrepository.com/artifact/commons-codec/commons-codec/1.8`)
- **Bouncy Castle Provider 1.51**
  This library is used to generate ECC key pairs in a Java Runtime Environment (i.e., the MBPS server). The CustomSerialization also handles the ECC key pair generation. (URL: `http://mvnrepository.com/artifact/org.bouncycastle/bcprov-jdk15on`)

The following third-party libraries have been used for the implementation of the SamplePaymentProject:

- **Spongy Castle Core 1.50.0.0** and **Spongy Castle Provider 1.50.0.0**
  These libraries are used to generate ECC key pairs on Android devices. Android includes an older Bouncy Castle Provider and does not allow to import a newer version. Spongy Castle overcomes this problem by renaming the package names of Bouncy Castle. (URL: `http://mvnrepository.com/artifact/com.madgag.spongycastle/core/1.50.0.0` and `http://mvnrepository.com/artifact/com.madgag.spongycastle/prov/1.50.0.0`)

Appendix H

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
- **Masterarbeit.pdf**
  PDF version of this report
- **Masterarbeit.zip**
  Figures and LaTeX source files of this report
- **Presentation.pptx**
  The slides for the presentation/defense
- **Presentation.pdf**
  The slides for the presentation/defense in PDF format
- **Source/**
  - **AndroidKitKatNFCLibrary.zip**
    The Android KitKat NFC Library is responsible for exchanging messages between two devices over NFC. This archive contains the source code as well as the test classes.
  - **AndroidNFCPaymentLibrary.zip**
    The Android NFC Payment Library implements a generic payment protocol to be used on top of the Android NFC Payment Library. This archive contains the source code as well as the test classes.
  - **CustomSerialization.zip**
    The source code of the library handling the custom serialization on byte level. This archive contains the source code as well as the test classes. This is a shared resource used in the Android NFC Payment Library and in the MBPS client and server.

- **Miscellaneous/**
  - **AndroidSignaturesTests.zip**
    The source code for the signature algorithms benchmarks.
  - **SamplePaymentProject.zip**
    This is a minimalistic project which shows how the Android NFC Payment Library has to be used. This is a simple Android application which does not contain test classes.