Static Discharge

Bitcoin payments method over
Near Field Communication (NFC)

Michael Hawkins

November 29, 2019

1 Overview

Near Field Communication (NFC) is used for contactless payment in bank cards and more recently it has been integrated into mobile phones for use in apps such as google or apple pay.
This project provides a secure transport layer for point of sale lightning transactions by connecting devices over NFC and interfacing with existing lightning wallet implementations.

2 Security

Lightning point of sale transactions are normally conducted via QR Codes which provide a one-way interface, encoding a lightning invoice that is then transferred optically between devices. The invoice is encoded as plain text and cannot be encrypted using a PKI due to the inability to exchange keys. This means that the invoice can be intercepted and decoded by any QR Code reader with visibility of the genrated code.

NFC provides an interface for bi-directional communication between devices which enables the use of encryption via key exhange. This can be added to the transport of transaction data and greatly improve the security model of making a lightning payment through the range that the transaction data is visible and the ability to read the transaction data if it was intercepted.

3 Use Case

A real world example of a MFC point of sale system, we can look at systems that require a successful payment to gain access to transport. Transport For London use a system of NFC readers to allow travellers to pay for travel via NFC as an alternative to purchasing traditional paper tickets.

Users present their NFC device to a terminal that attempts to make a transaction that, if successful, unlocks the turnstile and allows them access to the train

platform or informs the bus driver that payment has been received.

4 objectives

From a design point of view we have a service (access to train platform) and 3 actors:

- **User** the entity that requests a service
- **Device** used by the **User** to interact with the service
- **Terminal** that provides the service

To break the system down into the simplest form requires the following:

- **Terminal** MUST generate invoice
- **Device** MUST read an invoice
- **Device** MUST crate a lightning transaction using the invoice
- **User** SHOULD be notified of the transaction
- **Terminal** SHOULD confirm the transaction

Constraints on the system are:

- communication between **Terminal** and **Device** MUST be over NFC
- lightning invoices MUST be unique
- communications SHOULD be encrypted
- transactions SHOULD be made without **User** input

5 Technical Specification

5.1 Terminal

5.1.1 NFC

The terminal is created using a Raspberry PI with a RC522 circuit. To connect the RC522, header pins are soldered onto the board and then wired to the PI GPIO pins. The RC522 board provides the interfaces listed in Table 1 on page 3.

We use python and the library MFCRF522 to interact with the circuit and provide the methods needed to read and write using NFC.

5.1.2 Bitcoin

The Raspberry Pi runs a bitcoin core node connected to the testnet3 network.

5.1.3 C-Lightning

An instance of c-lightning is running and connected to the local bitcoind instance.

Description	Label	Pin
Serial Data Signal	SDA	24
Serial Clock	SCK	23
Master Out Slave In	MOSI	19
Master In Slave Out	MISO	21
Interrupt Request	IRQ	None
Ground Power	GND	6
Reset-Circuit	RST	22
(3.3v Power In	3.3v	1

Table 1: RC522 to Raspberry Pi pin mapping

5.1.4 LND

An instance of https://github.com/lightningnetwork/lnd)LND is also running as a backend for the mobile wallet[1]

The LND node was configured to avoid conflicting ports with the c-lightning instance, using the bitcoind backend, connected to the testnet. It was also configured to listen on the LAN network address (10.6.5.14) for rpc commands as the zap wallet will be using it.

In order to link the Zap wallet to the LND node I also had to install lndconnect in order to generate the lnconnect url needed (sigh). Once installed I found that I couldn't generate a readable form on the command line so needed to run the following command and then generate a QR Code online

```
MACAROON_HOME=~/.lnd/data/chain/bitcoin/testnet
./bin/lndconnect -j \
        --lnddir ~/.lnd/ \
        --host 10.6.5.14 \
        --adminmacaroonpath=${MACAROON\_HOME}/admin.macaroon \
        --datadir=~/.lnd/data \
        --tlscertpath=~/.lnd/tls.cert
```

Listing 1: Query to generate lnconnect url using lndconnect in bash shell

5.1.5 Invoice Provider

We use python and the library pylighting to interact with the lightningd instance and provide the methods needed to generate an invoice.

5.2 Mobile Hardware

The mobile device used for development is a oneplus A5000 running Android 9. This has developer mode and NFC enabled:

> Settings ¿ Bluetooth and device connection ¿ Connection preferences ¿ NFC

[1]This was a comprimise as I had wanted a standalone app for the mobile device but the only wallet I could get running that I could easily develop with was the LN-Zap wallet

5.3 Mobile Software

5.3.1 Wallet Selection

To avoid having to create a lightning wallet, the functionality for communicating over NFC is built into the open source Zap wallet. The advantage that this wallet had over the other wallets I evaluated was simply that I could compile it!

I was suprised by the relatively small selection of mobile wallets available and even more so by the problems that I had in being able to set up a development environment to work on them.

- Eclair Wallet would have been my first choice as it is a full lightning node implementation in Java that I'm comfortable programming with. Unfortunately getting it to build in android studio was beyond me. I believe that this was due to eclair-core being written in Scala which I was able to compile successfully but when including the jar it looked like I was missing some of the standard library.
- Bitcoin Lightning Wallet is written in Scala so after my experience with eclair I was put off somewhat and having never writen in Scala before I thought that the learning curve would be too steep to get going with using it in the hackathon.
- Bluewallet is wallet written in Javascript (nodejs) that I wasn't able to build. I think that there was/is a problem with my development environment that stops me from running react-native applications.
- Lightning labs mobile wallet was written in Javascript and the build instructions lean heavily towards development using Apple Mac (I'm rocking linux). Once again not able to run this due to my dev environment.
- Spark is another wallet that was on my radar but never tried this as it wasn't a complete lightning node and is written in Javascript, both of which put me off.

5.3.2 Modifications

`SendPaymentActivity.java` is the class containing the methods for sending the lightning (and bitcoin) payments. `ScanActivity` is the class that is used as a template for creating the intent as it defines the QR interface. We'll be creating the NFC interface which is a stripped down version of this.

5.4 User

The **User** should have minimal input in the interaction. A constraint of NFC on Android is that the scrren needs to be on for NFC, this the user will do before presentig the **Device** to the **Terminal** .

The **Device** SHOULD show a notification to let the **User** know that an interactoin has taken place. The standard form of this is with a vibration but it would also be nice to have an onscreen display of the status of the interaction.