Lelantus Light Wallet Design Ideas
Draft Version 0.0.6 (WIP)

www.firo.org

March 2021

Abstract

This note discusses light-wallet design ideas for Firo supporting both shielded and transparent transactions.

1 Introduction

In order to receive coins or verify transactions the blockchain wallet needs to download the full ledger. This is not suitable for low capacity devices (mobile wallets).

A possible approach for supporting light-wallet functionality is the reliance on third-party [Electrum] services which can maintain the full ledger on behalf of the end users (wallets) and provide and easy to use API to the wallets for accessing or filtering the transactions data. The drawback of this approach is this third-party service will get important information about the users transactions (its spending habits etc), so this is not acceptable for privacy reasons.

Our light-wallet design enables the wallets to receive new coins or spend their coins by downloading only the anonymity set data along with other small block metadata and without relying on any semi-trusted third party services. This design leverages Firo's master nodes network and does not leak any spending or other transaction information to the nodes

2 Lelantus V1 Anonymity Sets

3 Lelantus V2 Anonymity Sets

With Lelantus V2, the anonymity sets are comprised of pair values $[C, Q]$, where each output coin C is explicitly associated with the corresponding recipient address Q.

Each wallet can generate as many receiving addresses Q as he wants and the wallet can receive coins to the same address Q or to different addresses. Importantly, these receiving addresses can be generated through HDMint-like functionality from the user master secret so the user can re-generate all his previous addresses on a new device with his master secret key.

The output coins and associated receiving addresses are stored in block headers so all information necessary for composing the anonymity set can be easily extracted by scanning only the block headers and not the whole block information. Assuming at the given moment the full anonymity set is comprised of N coins as follows

$$\text{Anonymity Set - } \Big([C_1, Q_1], [C_2, Q_2], \ldots, [C_N, Q_N] \Big)$$

, the light-Wallets connect with master nodes and requires this anonymity set information which in turn **should be signed** by the quorum of master nodes. An access to the full anonymity set is necessary for both identifying the received coins and also for spending a particular coin.

3.1 Anonymity Set Attestation

Each transaction's output coins data is stored in block headers along with the corresponding receiving addresses, so block headers information is enough to fully compose the anonymity set.

As Firo already leverages the master nodes network and uses Chainlocks already, this existing infrastructure can be used to efficiently power light wallet functionality.

Master nodes can cache the anonymity set information and each new added block's header contributing to the anonymity set in turn can signed separately by the LLMQ (long-living masternode quorum) Assuming each block B_i contains b_i new output coins

$$[C_1^i, Q_1^i], \ldots, [C_{b_i}^i, Q_{b_i}^i]$$

. We require the LLMQ sign the hash of all output coins contained in the block concatenated with the signed material of the previous block. So for each block B_i, the following signatures are computed should the signed by the LLMQ

- $t_1 = \text{CSHF}(""||[C_1^1, Q_1^1], \ldots, [C_{b_1}^1, Q_{b_1}^1])$, $sig_1 = BLS_{sign}(t1)$
- $t_2 = \text{CSHF}(t_1||[C_1^2, Q_1^2], \ldots, [C_{b_1}^2, Q_{b_1}^2])$, $sig_2 = BLS_{sign}(t_2)$
- $t_3 = \text{CSHF}(t_2||[C_1^3, Q_1^3], \ldots, [C_{b_3}^3, Q_{b_3}^3])$, $sig_3 = BLS_{sign}(t_3)$
- $\cdots$
- $t_N = \text{CSHF}(t_{N-1}||[C_1^N, Q_1^N], \ldots, [C_{b_N}^N, Q_{b_N}^N])$, $sig_N = BLS_{sign}(t_N)$

The anonymity set information cached by the masternodes and provided to the light-wallet upon the download request is formed as a chain of output coin blocks paired with corresponding signatures

$$\left(([C_1^1, Q_1^1, \ldots, C_{b_1}^1, Q_{b_1}^1], t_1, sig_1); ([C_1^2, Q_1^2, \ldots, C_{b_2}^2, Q_{b_2}^2], t_2, sig_2); \ldots, ([C_1^N, Q_1^N, \ldots, C_{b_N}^N, Q_{b_N}^N], t_N, sig_N)\right)$$

REMARK: It may be possible to skip block-related signatures from being included in the list of provided data and only include the last signature **TODO Discuss with Sarang.**

3.1.1 Master-Node Side Anonymity Set Filters

Assuming the number of anonymity sets grows, it may not be rational for the light-wallet to download all anonymity sets as he needs only downloading the relevant ones. For example new created wallets are unlikely to have coins in the historical anonymity sets. Master nodes can support efficient and privacy-preserving filtering of all relevant anonymity sets for the light wallets by providing anonymity-set filters.

- They compute special anonymity set filters with respect to the output coins' shielded addresses.
- The wallet first downloads all filters corresponding to all anonymity sets which are much smaller than the anonymity sets themselves.
- The wallet identifies all relevant anonymity sets during this filtering process and then requests the full sets. This only leaks the fact that a certain wallet has coins within the selected anonymity sets, which should be ok from privacy perspectives.

The anonymity set filters should be generated and periodically updated by the master nodes. These filters should be certified through similar methods described above. Either Bloom filters or Golomb-Rice space efficient coding can be used for creating the filters.

3.2 Wallet Receiving Coins

After downloading the recent anonymity set information, the wallet can already identify all received coins by scanning the anonymity set and filtering it by its own generated addresses. Assuming the wallet has t different receiving addresses $Q_1^w, Q_2^w, \ldots, Q_t^w$, it can scan the full set to identify all coins and their indexes within the anonymity set which are sent to either of these addresses. The position of each received coin within the anonymity set is an important private parameter for spending it later.

3.3 Wallet Spending Coins

Having the full anonymity set $[C_1, Q_1], [C_2, Q_2], \ldots, [C_N, Q_N]$ information and also the spendable coin's $[C^w, Q^w]$ private data (l, S^w, V^w, R^w), the light wallet can already generate the Spend transaction and post it to blockchain. Full nodes can verify the validity of this transaction.

4 Client-Side Block Filtering for Transparent Layer Transactions

In order to support light wallet functionality for transparent transactions we should look for other well-known techniques already implemented for bitcoin wallets.

An interesting candidate is the Neutrino scheme, a privacy preserving light-wallet client protocol.It works as follows:

- Full nodes compose special filters for each block. These filters are much smaller compared to the original blocks. Filters are similar to Bloom filters, but are generated by using the Golomb-Rice coding methods. [More details on this coding below].
- The light wallet downloads the filters to check if it contains any transaction related to the wallet(e.g. sending to one of the wallet's addresses)
- When a match is detected, the light wallet can ask the full block for verifying the transactions.

A full specification of Neutrino is provided by BIP0157 [**BIP0157**] and BIP0158 [**BIP0158**]
A short summary of Neutino is cited below:

"Neutrino is a protocol to verify payments, except this time, the bulk of the work is done on the client side. Instead of the server filtering transactions for the client, now all the transactions belonging to a block (technically ScriptPubKeys corresponding to each input and output (except the OP-RETURN outputs) are compressed and sent to the client. It's now the client's job to figure out if any of the transactions are ones it has transacted in. If any of the transactions are relevant to the wallet, the client then requests the full block to verify the transactions

It turns out that the compression can be pretty impressive. A normal block is around 1.4MB, but by compressing it (technically, hashing each ScriptPubKey to 64 bits), each block produces about 20KB of super-compressed data per block. Since this super-compressed block is the same for every light client, this removes the denial-of-service vulnerability for the server. This also means that the server gets no special information about the light client other than what blocks it wants to look at, meaning that there are much fewer privacy leaks."

References

[1] https://github.com/bitcoin/bips/blob/master/bip-0157.mediawiki

[2] https://github.com/bitcoin/bips/blob/master/bip-0158.mediawiki