HushList Protocol Specification
Version

David Mercer[†]
Duke Leto[†]

December 23, 2017

Abstract.

HushList is a protocol for mailing lists using the encrypted memo field of the **Zcash** protocol. It supports anonymous and pseudonymous senders, receivers and Hushlist creators, as well as public and private lists. The HushList protocol can run on any fork of **Zcash** that has a compatible memo field, though certain advanced features might not be fully supported on all chains. HushList is developed and tested on the Hush and Zcash mainnets as well as testnets (TUSH and TAZ), next to be tested is Komodo (KMD).

In addition to the above properties, **HushList** provides users with censorship-resistant storage and retrieval, since every **Hush** full node will have an encrypted copy of every **HushList** memo. Furthermore, sending and receiving via one or more blockchains is a serious deviation from traditional server-client design which easily allows a Man-In-The-Middle Attack and Deep Packet Inspection (DPI). Network traffic monitoring and correlation is made much harder, because there is no longer a packet with a timestamp and "selectors" going from one unique IP to another unique IP via a very predictable network route.

Zcash is an implementation of the *Decentralized Anonymous Payment* scheme **Zerocash**, with security fixes and adjustments to terminology, functionality and performance. It bridges the existing ***transparent*** payment scheme used by **Bitcoin** with a ***shielded*** payment scheme secured by zero-knowledge succinct non-interactive arguments of knowledge (*zk-SNARKs*).

Hush is a fork of the **Zcash** codebase (1.0.9) which generated it's own genesis block and uses the Zcash Sprout proving key.

This specification defines the **HushList** communication protocol and explains how it builds on the foundation of **Zcash** and **Bitcoin**.

Keywords: anonymity, freedom of speech, cryptographic protocols, electronic commerce and payment, financial privacy, proof of work, zero knowledge.

Contents

[†] Hush Core Developers

Introduction

HushList is a protocol for anonymous mailing lists using the encrypted memo field of the zcash protocol.

Technical terms for concepts that play an important role in **HushList** are written in *slanted text*. ***Italics*** are used for emphasis and for references between sections of the document.

The key words **MUST**, **MUST NOT**, **SHOULD**, and **SHOULD NOT** in this document are to be interpreted as described in [RFC-2119] when they appear in **ALL CAPS**. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

This specification is structured as follows:

- Notation — definitions of notation used throughout the document;
- Concepts — the principal abstractions needed to understand the protocol;
- Abstract Protocol — a high-level description of the protocol in terms of ideal cryptographic components;
- Concrete Protocol — how the functions and encodings of the abstract protocol are instantiated;
- Implications

High-level Overview

The following overview is intended to give a concise summary of the ideas behind the protocol, for an audience already familiar with *block chain*-based cryptocurrencies such as **Bitcoin** or **Zcash**.

XXX

Value in **Hush** is either *transparent* or *shielded*. Transfers of *transparent* value work essentially as in **Bitcoin** and have the same privacy properties. *Shielded* value is carried by *notes*, which specify an amount and a *paying key*. The *paying key* is part of a *payment address*, which is a destination to which *notes* can be sent. As in **Bitcoin**, this is associated with a private key that can be used to spend *notes* sent to the address; in **Hush** this is called a *spending key*.

A *transaction* can contain *transparent* inputs, outputs, and scripts, which all work as in **Bitcoin** [Bitcoin-Protocol]. It also contains a sequence of zero or more *JoinSplit descriptions*. Each of these describes a *JoinSplit transfer* which takes in a *transparent* value and up to two input *notes*, and produces a *transparent* value and up to two output *notes*.

Each **HushListMUST** have a default blockchain that it is attached to, and the default chain **SHOULD** be HUSH. The user **MUST** be able to set their GLOBAL default chain (not implemented yet) as well as a default chain for each list.

Each list also has a tadd+zaddr dedicated to that list, so the user has dedicated addresses to send psuedo/anon messages, as well as default fee and amount. The default amount is 0.0 and the default fee is currently 0.0001 but these numbers are subject to change.

HushList supports file attachments and embedding arbitrary binary data, it is not limited to ASCII.

Design of HushList

The design of **HushList** is inspired by Git. The reference implementation is a command-line program which is a very thin wrapper around an API, which is implemented as a various Perl modules. **HushList** uses many of the same subcommands as Git which have intuitive meanings, which provide "easy-onramps" to learn how to use the CLI.

This document specifies a protocol and the authors provide a reference implementation of this protocol in cross-platform Perl which can be easily installed on millions of computers around the world via CPAN and other methods.

HushList should work across any platform that supports Perl and Hush (or your other coin).

The reference implemenation is written in a maintainable and testable way such that it can easily evolve as the Protocol evolves.

It is hoped that in the future there will be many implementations of **HushList**, running on various blockchains and using various software stacks. The design of **HushList** is compatible with Simple Payment Verification (SPV) light wallets and a future version of **HushList** will learn to speak an ElectrumX backend server.

Reference Implementation

The reference implementation is developed as Free Software on Github at the following URL:

https://github.com/leto/hushlist

This code is still in active development, consider it EXPERIMENTAL and ONLY FOR DEVELOPERS at this point pending a security review.

Account Funding

On first run, **HushList** creates a new shielded zaddress z_F to fund transparent addresses for pseudonymous sending.

It may be funded by the user from any taddr or zaddr with no loss of privacy.

For each pseudonym the user sends from (may be globally used or per-list), a taddr t_L is created and a de-shielding transaction is done from $z_F \rightarrow t_L$ which will allow the user to send memos to the given **HushList** on behalf of the t_L pseudonym. Since **HushList** memos have, by default, an amount of 0.0, all the costs associated with using **HushList** are network costs. Users may additionally add a non-zero amount to a **HushList** memo.

For each **HushList** the user wants to be part of, **HushList** will create a brand new zaddress z_L (it **MUST NOT** reuse an existing address) and fund that address via a shielded $z \rightarrow z$ transaction between $z_F \rightarrow z_L$.

If there are no taddr or zaddr funds in the entire wallet, **HushListSHOULD** present the user a taddr + zaddr which can be used to "top up" the current **HushList** wallet from another wallet/exchange/etc.

HushList Contacts

HushList maintains a database of contacts which use the address as the unique ID and additional metadata. Since **HushList** supports multiple blockchains, it **MUST** have a contact database for each chain. Each chain **MUST** have it's own contact namespace, so you can have Bob on Hush and Bob and Zcash and they will not conflict.

HushList internally associates lists to Contacts, not the address of a contact. This allows the user to update the address of a contact in one place and things work correctly the next time the address of that contact is looked up. Lists contain Contacts and Contacts have addresses.

A **HushList** contact may only have ONE address, either taddr or zaddr, but not both.

To have a taddr and zaddr for a person, you can simply create two contacts, such as tBob and zBob. In terms of the metadata that is revealed when communicating with tBob or zBob, they are quite different, and it is healthy for metadata minimization to consider them as two different contacts.

List Creation

A private **HushList** is simply a list of contacts stored locally and costs nothing. The **Zcash** protocol itself has a max of 54 recipients currently, so **HushList** implementations should not allow lists with more than 54 recipients at this time.

A public **HushList** means publishing the PRIVATE KEY of a taddr (or potentially a zaddr) such that this address is no longer owned by a single individual. By intentially publishing the PRIVATE KEY in a public place, the owner has put all FUNDS and more importantly, the metadata of all transactions to that address, in the public domain.

By default, **HushListMUST** refuse to publicize the PRIVATE KEY of an address that has non-zero balance. **HushList** implementations **SHOULD** protect users from accidental monetary loss in every way possible. Even so, a user could accidentally send funds to an address that has been publicized and this very real confusion is still looking for good answers.

Very recent developments in **Zcash** might allow the potential to use "viewing keys" in the fture, but as this feature has not been fully merged to master at this time and lacks a RPC interface, **HushList** chooses to use PRIVATE KEYS which are core **Zcash** protocol that is well-supported in all forks. If "viewing keys" are one day to be used, that feature will need to be merged into multiple **Zcash** forks, which does not seem likely in the near-term.

Since creating a private **HushList** requires making a transaction on the network to store data in the memo-field, it has a cost. This cost will be the fee of the transaction, most likely around 0.0001 but each chain is different and fees obviously change as blockchains get more active.

List Subscription

When the private key for a list is imported into HushList, either from the blockchain, URI or manual entry, the private key is added to the user's wallet, along with a user entered or approved name and description for the list (if provided in on-chain or uri encoded metadata). HushList creates a unique taddr + zaddr for each list so that the user may choose to send each message to the list psuedonymously or anonymously or a mixture of both. There is no loss of privacy to send memos to the same **HushList** with a psuedonym tAlice and an anon handle zBob if the user so chooses.

Subscribing to a **HushList** is completely free, it is simply the act of importing data to your local wallet.

Sending To A List

One may send to a **HushList** from a taddr (pen name, psuedonym) or zaddr (anonymous shielded address) which is implemented in the client via the z_sendmany RPC command. Up to 54 recepients may be in a single shielded transaction. v1 of HushList only supports HushLists of this size, but v2 may implement larger HushLists by breaking large recipient lists into multiple sends.

One may send a string of text via the *send* subcommand or send the contents of a file via the *send-file* sub-command. If one sends a string of text, there is no metadata related to that at all, locally. It only exists encrypted in a memo field on the chain. If one uses the *send-file* command, it may be prudent to securely delete the file from the filesystem after it is sent, depending on the needs of the user.

Each HushList has a dedicated default chain that it is attached to. When looking up **HushList** contacts for a given list, their address on that chain will be retreived.

A unique feature of **HushList** is that speech=money, so you may always attach a non-zero amount of **HUSH**,**ZEC**,**KMD**/etc to each memo to a **HushList**. Currently you must send each member of a **HushList** the same amount in one memo, but you may send different amounts in different memos.

Receiving Messages

At any time later, after the transaction has entered the blockchain, memos sent to a given address can be downloaded and viewed by those parties who have valid private keys or viewing keys.

Clients can poll the local full node periodically at a user specifiable default interval OR, by default, the same as the average block time for the chain in question. For the **Hush** chain, this is 2.5 minutes.

If for any reason a **HushList** user wants to PROVE with cryptographic certaintity that they knew certain information at a certain time, all they would need to do is publish the PRIVATE KEY of an address which made the transaction that contains the information.

This is the so-called "investigative journalist" or "whistle-blower" use case. An individual can send themselves **HushList** memos "just in case" they need to prove something in the future. This can be considered "data as insurance".

Costs

Sending **HushList** memos requires making a financial transaction and by default, **HushList** sends the recipient a transaction for 0.0 **HUSH** (or **ZEC** etc) with the default network fee (currently 0.0001 for **ZEC** +**HUSH**). The fee amount **MUST** be configurable by the user. In the reference implementation of **HushList** it be changed via the HUSHLIST_FEE environment variable. Additionally, every **HushList** has it's own configurable fee declared in the configuration file for that list. The user may set a higher fee on some lists to ensure faster delivery while using lower fees on other lists which are not as time sensitive.

References

[Bitcoin-Protocol] *Protocol documentation — Bitcoin Wiki*. URL: `https://en.bitcoin.it/wiki/Protocol_documentation` (visited on 2016-10-02) (↑ p 3).

[RFC-2119] Scott Bradner. *Request for Comments 7693: Key words for use in RFCs to Indicate Requirement Levels*. Internet Engineering Task Force (IETF). March 1997. URL: `https://tools.ietf.org/html/rfc2119` (visited on 2016-09-14) (↑ p 3).