Abstract.

HushList is a protocol for mailing lists using the encrypted memo field of the Zcash protocol. It supports anonymous and pseudonymous senders, anonymous 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 mainnet and testnets and is designed to run on any ZEC code fork including but not limited to HUSH, KMD, ZCL, ZEN, VOT, BTCZ and LTZ. HushList is also compatible with Bitcoin Hush BTCH, which is a KMD asset chain.

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 [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 [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

1 Introduction 3

1.1 High-level Overview 3

1.2 Types Of Transactions 3
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 blockchain-based cryptocurrencies such as **Bitcoin** or **Zcash**.

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**.

**Types Of Transactions**

All Zcash forks have what we will classify into **FOUR categories of transactions**.

Let \( t \rightarrow t \) be called a **transparent** transaction, which is identical to a **Bitcoin** transaction, consisting entirely of transparent inputs and outputs. **HushList** protocol implementations **MUST NOT** create **transparent** transactions, they do not protect metadata in any way and lack memo fields. These transactions can be done with traditional wallet software and does not have any part in **HushList** protocol.

Let \( t \rightarrow z \) be called **shielding** transactions, which takes transparent value stored in UTXOs and transforms them to **Shielded** value stored inside of **notes** which are protected by zk-SNARKs.

Let \( z \rightarrow z \) be called **Shielded** transactions, which take **Shielded** value from one **Shielded** address to another, and is fully protected by zk-SNARKs. **HushList** implementations **MUST** support these transactions, and additionally **SHOULD** educate users that they are the most private type of transaction, which minimized metadata leakage.

Let \( z \rightarrow t \) be called **deshielding** transactions, which take **Shielded** value stored in **notes** in \( z \) and transfer them to UTXOs stored in \( t \). **HushList** **MUST NOT** create **deshielding** transactions, as they leak metadata and can potentially link a previously **Shielded** address \( z \) to a transparent address \( t \). **HushList** implementation **SHOULD** attempt to prevent, at all costs, accidentally sending to a \( t \) address via the \( z\_sendmany \)RPC command.
An easy way to summarize the support transactions of HushList is to say: All receivers must by *shielded* addresses, senders can be either *transparent* or *shielded* addresses.

Each HushListMUST have a default blockchain and network that it is attached to, and the default chain **SHOULD** be HUSH. The default network **MUST** be assumed to be “mainnet” if not specified, similar to how the *master* branch is assumed in many Git commands if not specified. 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 taddr+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. HushList does not impose file size limits and network fees and CPU/RAM costs provide natural incentives for spammers to find cheaper and easier-to-access dumping grounds.

### 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 the coin being used. In this case, cryptocurrencies are much less portable than Perl, so Perl will not be the limiting factor.

The reference implementation 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 to an ElectrumX backend server, which natively supports Hush as of the upcoming 1.2.1 release.

### Reference Implementation

The reference implementation is developed as Free Software under the GNU Public License Version 3 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. This is the bleeding edge, folks.

The current reference implementation can send and receive memos, including files on disk or simple strings of text, as long as they are up to 512 bytes.

Multipart HushList memos (file attachments) and public HushLists are still in development.
Account Funding

On first run, HushList creates a new shielded zaddress $z_F$ to fund transparent addresses for pseudonymous sending via the $z\_sendmany$ RPC method.

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, HushList SHOULD 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.

If one has the addresses for a set of contacts on multiple chains that are supported, say ZEC, HUSH and KMD, then a user may send a memo to members across multiple blockchains to ensure delivery and subvert censorship of a single chain.

HushList Creation

Private HushLists

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

Multi-Chain Private HushLists

A user may choose to send a HushList memo via multiple coins as long as there is a valid address for each Hush Contact on for each coin. For example, if you have addresses for three of your friends on each of the HUSH, KMD and ZEC chains, then you may choose to redundantly send a memo on all of the chains. This provides a backup of the data on the other chains should one of them be blocked (such as dropping any packets for certain peer-to-peer ports), filtered or temporarily inaccessible.
Additionally a user may choose to send day-to-day memos on an inexpensive chain such as HUSH which has lower network difficulty and for things that need to have Bitcoin-level security, an archive copy to KMD can be sent. KMD uses the delayed-Proof-Of-Work [dPOW] algorithm ensuring that once the information is engraved on the Bitcoin blockchain, it would be required both blockchains in question to be compromised to prevent accessing the data.

Public HushLists

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 intentionally 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 future, 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.

To facilitate applications being able to uniquely identify public HushLists we introduce a new URL scheme where there username is the currency symbol of the cryptocoins and the password field is the network, i.e.

hushlist://COIN:NETWORK@K

COIN can be the currency symbol of a compatible cryptocoins such as HUSH (Hush), KMD (Komodo), ZEC (Zcash), ZCL (ZClassic), ZEN (ZenCash) or BTCH (Bitcoin Hush).

NETWORK is will often be “mainnet” but this schema allows for the very real use case of developers iterating through various testnets and supports “sidenets” for those that want to isolate data from mainnet.

K is the base58-formatted PRIVATE KEY as returned by the dumpprivkey RPC method of the associated coin.

When COIN and NETWORK are omitted, they default to HUSH and “mainnet” respectively, so

hushlist://K

is equivalent to

hushlist://HUSH:mainnet@K
Additionally, URL parameters can be used to specify a nickname and fee for the list to be imported:

```
  hushlist://HUSH:mainnet@K?n=nickname&amp;fee=X&amp;height=N
```

The nickname and fee are just suggestions and the user **MUST** be able to modify them before importing the list. The height is actually for performance reasons, and helps the local avoid scanning the entire history of the blockchain for transactions.

The first public HushList can be uniquely identified by the following URL:

```
  hushlist://SKxqPjNKvcfpmbpR8daQHNj4DoMfKmaPiVcT3A3YPynZNYXoDoaq
```

For performance reasons, we can help each node skip over 200,000 blocks filled with transactions by specifying the minimum block height for `z_importkey` to look in. This more performant URL is:

```
  hushlist://SKxqPjNKvcfpmbpR8daQHNj4DoMfKmaPiVcT3A3YPynZNYXoDoaq?height=215683
```

This HushList contains the first HushList memo, described in the sections below.

### Subscribing To A Public HushList

The URL above serves as the main way for people to subscribe to a public HushList, since it can be embedded just like a https:// link in any other text document or website. Depending on the browser that one uses, the hushlist:// protocol may already be linkified on a web page. Protocol helper applications can and will be developed so the appropriate action can be taken when a user clicks a hushlist:// link.

In the reference implementation, the `subscribe` subcommand can be used:

```
  hushlist subscribe
  hushlist://SKxqPjNKvcfpmbpR8daQHNj4DoMfKmaPiVcT3A3YPynZNYXoDoaq?height=215683
```

### Sending To A HushList

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 method. Up to 54 recipients may be in a single shielded transaction, which is an upstream ZEC limitation which exists in all ZEC forks. v1 of HushList only supports HushLists of this size, but v2 may implement larger HushLists by breaking large recipient lists into multiple sends, at the expense of atomicity.

One may send a string of text via the *send* subcommand or send the contents of a file via the *send-file* subcommand. 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 retrieved.

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 certainty 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.

Examples

The first HushList memo was a $t \rightarrow z$ transaction which also included a non-zero amount of 0.055555 HUSH. It is viewable on the Hush block explorer here: https://explorer.myhush.org/tx/30a38c7ba0929efb7cd54d3b724d9eb1d9cb03f35381a94d889bc4cfb0593bf

One may note that the zaddr associated with this transaction does not appear anywhere in the explorer, because shielded addresses never show up directly in the public blockchain. Network transaction analysis is not possible on zaddrs. The explorer only shows that a JoinSplit occurred and that change was given to a taddr.

Nevertheless, the follow text is forever embedded in the 512 byte memo field of the above transaction:

A beginning is the time for taking the most delicate care that the balances are correct.
- "Manual of Muad'Dib" by the Princess Irulan

Once men turned their thinking over to machines in the hope that this would set them free. But that only permitted other men with machines to enslave them.
- Reverend Mother Gaius Helen Mohiam

Polish comes from the cities; wisdom from the desert.
- Arrakeen villager saying

Be prepared to appreciate what you meet.
- Fremen proverb

Note that the transaction does leak the metadata of the amount, since it was a de-shielding transaction, from $t \rightarrow z$. All HushList memos have amount=0.0 by default so this is not normally a concern.
Metadata Analysis

The biggest concern for metadata leakage in HushList is in de-shielding $t \rightarrow z$ transactions which leak amount metadata.

The only time HushList does a de-shielding transaction is when the local wallet has 0 shielded value and it must transfer value from a taddr OR when the user chooses to send from a pseudonymous taddr to a HushList.

The first case we call a "shielded top-up" and happens rarely but we would not want to always have the same default amount to "top-up" because that amount can be searched for on the public chain. For this reason, we add some noise to the exact amount of our topups. For instance, if the user wanted to move up to 1 HUSH, we would generate a random number between 0.9 and 1.0 and then subtract it from the top-up amount. Then all HushList users would have slightly different top-up amount instead of a few easily searchable amounts.

In the second case, normal transactions will have amount=0 which will stand out and network transaction analysis is possible. If these pseudonyms choose to actually send non-zero amounts, network analysis can be made harder since most HushList messages use amount=0.

User Stories

This section contains various "User Stories" of how potential users can use the various features of the HushList protocol to meet their needs.

"Pen Name" user story - Amanda

Let Amanda have a transparent address $t_A$ and let there be a PUBLIC HushList with shielded address $z_L$.

Amanda sends HushList memos from $t_A$ to a PUBLIC HushList with a de-shielding transaction, ie.

$t_A \rightarrow z_L$.

Any person who is subscribed to this public HushList will be able to see Amanda’s memos, yet Amanda’s identity is "pseudonymous", i.e. everybody knows that every message from $t_A$ is the same person, but her identity remains unknown. If at any time in the future, Amanda would like to *cryptographically prove* that she is the identity behind $t_A$, all she must do is create a signed message with her private key, which proves her ownership of it.

A more "nuclear" option is to publish the PRIVATE KEY of $t_A$. If any transparent value resides in $t_A$, it can simply be moved to another address before publication. This option "burns" the identity somewhat, as no messages after the publishing of the PRIVATE KEY can be known as the original authors or any other person who learned the key.

Of course Amanda is free to never reveal her identity and remain a pseudonym indefinitely.

Amanda needs to be concerned about her IP address being tied to $t_A$ by a passive network attacker who records the Internet and is encouraged to use a proxy, Tor or other means depending on risk and operational security needs.

Last Will And Testament User Story - Xerxes

Xerxes would like to store a copy of their Last Will And Testament in multiple secure locations, where they cannot be lost nor destroyed by parties that would benefit from the destruction of the Will.

Xerxes can use HushList protocol to store their will in many different blockchains, in the hopes that at least one will survive longer than him, and to prevent censorship if he only stored the data on one chain. Xerxes can choose to additionally make the will public initially, or after some time period, or only leave instructions for retrieving the will with executors of their Estate.
This use case also supports the continual updating of a Will, and provides a record of all the changes to a will, with timestamps and cryptographic certainty. This record can be verified by any and all executors, with or without making the records public. Indeed, a public HushList can be used to provide instructions and the actual Will, and newer memos to that list are public proof that the person has changed their Will.

"Oppressed Minority" user story - Francesca and Nicalau

Francesca and Nicalau live in a place where their local religion/government/organization is oppressed by a larger religion/government/organization that controls everything around them, yet they still want to safely communicate.

"Security Researcher" user story - Gordon

Dana wants to communicate 0-day exploits about nation-state infrastructure to the people that run this critical infrastructure, without anybody else listening in on this very sensitive information.

"Whistleblower" user story - Martha

Martha has data about something that must be transported from internal-only systems, to external places, preferably many, while knowing that the data is not tampered with or even viewed until the appropriate time.

"Censored Journalist" user story - Billy

This is an extension of the “Pen Name” User Story. Let’s say that for some reason a journalist Billy is already known publicly, but is censored from all media locally in various places. Billy can use HushList to publish his writing (and also source data, encrypted or not) to multiple blockchains to make it permanently mirrored across thousands of servers and very hard to censor.

Special Thanks

A special thanks to Daira Hopwood for an inspiring Zcash Protocol document and for making the Zcash infrastructure open source, which was used to make this document. HushList is built on the shoulders of giants and to all the people that have made the Bitcoin and Zcash ecosystems what they are, thank you.

Additionally, a special thanks to the Komodo Platform, which has embraced Hush as one of the first cryptocoins to be added to their BarterDEX atomic swap platform and continues to support the Hush Community in various ways.

References


