# HushList Protocol Specification
Version

David Mercer[†]
Duke Leto[†]

December 22, 2017

**Abstract.**

**HushList** is a protocol for anonymous 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 512 byte 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).

**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 1

[†] Hush Core, Zcash Core, Bitcoin Core

# 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*.

# Notation

$\mathbb{B}$ means the type of bit values, i.e. $\{0, 1\}$.

$\mathbb{N}$ means the type of nonnegative integers. $\mathbb{N}^+$ means the type of positive integers. $\mathbb{Q}$ means the type of rationals.

$x \mathrel{⦂} T$ is used to specify that $x$ has type $T$. A cartesian product type is denoted by $S \times T$, and a function type by $S \to T$. An argument to a function can determine other argument or result types.

The type of a randomized algorithm is denoted by $S \xrightarrow{\mathrm{R}} T$. The domain of a randomized algorithm may be $()$, indicating that it requires no arguments. Given $f \mathrel{⦂} S \xrightarrow{\mathrm{R}} T$ and $s \mathrel{⦂} S$, sampling a variable $x \mathrel{⦂} T$ from the output of $f$ applied to $s$ is denoted by $x \xleftarrow{\mathrm{R}} f(s)$.

Initial arguments to a function or randomized algorithm may be written as subscripts, e.g. if $x \mathrel{⦂} X$, $y \mathrel{⦂} Y$, and $f \mathrel{⦂} X \times Y \to Z$, then an invocation of $f(x, y)$ can also be written $f_x(y)$.

$T^{[\ell]}$, where $T$ is a type and $\ell$ is an integer, means the type of sequences of length $\ell$ with elements in $T$. For example, $\mathbb{B}^{[\ell]}$ means the set of sequences of $\ell$ bits.

$\mathsf{length}(S)$ means the length of (number of elements in) $S$.

$T \subseteq U$ indicates that $T$ is an inclusive subset or subtype of $U$.

$\mathbb{B}^{[8 \cdot \mathbb{N}]}$ means the set of bit sequences constrained to be of length a multiple of 8 bits.

$\mathbf{0x}$ followed by a string of **boldface** hexadecimal digits means the corresponding integer converted from hexadecimal.

**"`...`"** means the given string represented as a sequence of bytes in US-ASCII. For example, **"`abc`"** represents the byte sequence $[\mathbf{0x61}, \mathbf{0x62}, \mathbf{0x63}]$.

$[0]^{\ell}$ means the sequence of $\ell$ zero bits.

$a..b$, used as a subscript, means the sequence of values with indices $a$ through $b$ inclusive. For example, $\mathsf{a}^{\mathsf{new}}_{\mathsf{pk},1..\mathrm{N}^{\mathsf{new}}}$ means the sequence $[\mathsf{a}^{\mathsf{new}}_{\mathsf{pk},1}, \mathsf{a}^{\mathsf{new}}_{\mathsf{pk},2}, ... \mathsf{a}^{\mathsf{new}}_{\mathsf{pk},\mathrm{N}^{\mathsf{new}}}]$. (For consistency with the notation in [BCG+2014] and in [BK2016], this specification uses 1-based indexing and inclusive ranges, notwithstanding the compelling arguments to the contrary made in [EWD-831].)

$\{a\,..\,b\}$ means the set or type of integers from $a$ through $b$ inclusive.

[ $f(x)$ for $x$ from $a$ up to $b$ ] means the sequence formed by evaluating $f$ on each integer from $a$ to $b$ inclusive, in ascending order. Similarly, [ $f(x)$ for $x$ from $a$ down to $b$ ] means the sequence formed by evaluating $f$ on each integer from $a$ to $b$ inclusive, in descending order.

$a \,||\, b$ means the concatenation of sequences $a$ then $b$.

$\mathsf{concat}_{\mathbb{B}}(S)$ means the sequence of bits obtained by concatenating the elements of $S$ viewed as bit sequences. If the elements of $S$ are byte sequences, they are converted to bit sequences with the ***most significant*** bit of each byte first.

$\mathsf{sorted}(S)$ means the sequence formed by sorting the elements of $S$.

$\mathbb{F}_n$ means the finite field with $n$ elements, and $\mathbb{F}^*_n$ means its group under multiplication. $\mathbb{F}_n[z]$ means the ring of polynomials over $z$ with coefficients in $\mathbb{F}_n$.

$a \cdot b$ means the result of multiplying $a$ and $b$. This may refer to multiplication of integers, rationals, or finite field elements according to context.

$a^b$, for $a$ an integer or finite field element and $b$ an integer, means the result of raising $a$ to the exponent $b$.

$a \bmod q$, for $a \,⦂\, \mathbb{N}$ and $q \,⦂\, \mathbb{N}^+$, means the remainder on dividing $a$ by $q$.

$a \oplus b$ means the bitwise-exclusive-or of $a$ and $b$, and $a \mathbin{\&} b$ means the bitwise-and of $a$ and $b$. These are defined either on integers or bit sequences according to context.

$\sum\limits_{i=1}^{\mathrm{N}} a_i$ means the sum of $a_{1..\mathrm{N}}$. $\bigoplus\limits_{i=1}^{\mathrm{N}} a_i$ means the bitwise exclusive-or of $a_{1..\mathrm{N}}$.

The binary relations $<$, $\leq$, $=$, $\geq$, and $>$ have their conventional meanings on integers and rationals, and are defined lexicographically on sequences of integers.

$\mathsf{floor}(x)$ means the largest integer $\leq x$. $\mathsf{ceiling}\,(x)$ means the smallest integer $\geq x$.

$\mathsf{bitlength}(x)$, for $x \,⦂\, \mathbb{N}$, means the smallest integer $\ell$ such that $2^{\ell} > x$.

The symbol $\bot$ is used to indicate unavailable information or a failed decryption.

# 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 $z_F \to t_L$ 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 $0.0$.

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

## 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.

## List Creation

...

## 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.

## 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.

## 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.

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

## 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.

# References

[BCG+2014] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. *Zerocash: Decentralized Anonymous Payments from Bitcoin (extended version)*. URL: `http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf` (visited on 2016-08-06). A condensed version appeared in *Proceedings of the IEEE Symposium on Security and Privacy (Oakland) 2014*, pages 459–474; IEEE, 2014. (↑ p4).

[Bitcoin-Protocol] *Protocol documentation — Bitcoin Wiki*. URL: `https://en.bitcoin.it/wiki/Protocol_documentation` (visited on 2016-10-02) (↑ p3).

[BK2016] Alex Biryukov and Dmitry Khovratovich. *Equihash: Asymmetric Proof-of-Work Based on the Generalized Birthday Problem (full version)*. Cryptology ePrint Archive: Report 2015/946. Last revised October 27, 2016. URL: `https://eprint.iacr.org/2015/946` (visited on 2016-10-30) (↑ p4).

[EWD-831] Edsger W. Dijkstra. *Why numbering should start at zero*. Manuscript. August 11, 1982. URL: `https://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html` (visited on 2016-08-09) (↑ p4).

[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) (↑ p3).