

Storj Protocol Specification

Version 2 (May 23, 2017)

Gordon Hall (gordonh@member.fsf.org)

Braydon Fuller (braydon@storj.io)

Ryan Foran (ryan@storj.io)

0: License

Copyright (C) 2017 Storj Labs, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the “LICENSE” file.

1: Introduction

This specification documents the Storj network protocol in its entirety for the purpose of enabling its implementation in other languages. Described here, is the protocol **base** - the minimum specification for compatibility with the Storj network. Additional optional extensions to this work are defined as Storj Improvement Proposals (or “SIPs”), some of which have been folded into the base protocol since Version 1.

2: Identities

Every node (host computer speaking the Storj protocol) on the network possesses a unique cryptographic identity. This identity is used to derive a special 160 bit identifier for the purpose of organizing the overlay structure and routing messages (*3.1: Kademlia*). In order for a node to join the network it must generate an identity.

Identities are described as **hierarchically deterministic** and serve the purpose of running a cluster of nodes that can all share the same contracts and act on behalf of each other in the network. The specification extends Bitcoin ECDSA derivation standard BIP32 and BIP43.

Key derivation must match the specification of Bitcoin Hierarchical Deterministic Wallets (BIP32) with the purpose field described in Bitcoin Purpose Field for Deterministic Wallets (BIP43).

We define the following levels in BIP32 path:

`m / purpose' / group_index' / node_index`

The apostrophe in the path indicates that BIP32 hardened derivation is used. Purpose is a constant set to 3000, so as to not collide with any bitcoin related proposals which recommends to use the BIP number.

`m / 3000' / group_index' / node_index`

The `group_index` for most purposes will be 0. However is reserved for a future use to be able to increment in the case that the contracts should be updated with a new key. The `node_index` can be a number from 0 through $2^{31} - 1$, so that it's using a non-hardened paths and it's always possible to derive the public key for a node using the `m / 3000' / group_index'` derived extended public key. This gives a total of 2.147 billion possible nodes to run in a group cluster.

As noted in BIP32, a compromised private key at the `node_index` level in combination with the extended public key at the `group_index` level will compromise all descending private keys derived from the `group_index` level, this is the rationale for a hardened path for the `group_index`.

In every message exchanged on the network, each party will include a tuple structure which includes enough information to locate and authenticate each party.

```
[ "<node_id>", { /* <contact_hash_map> */ } ]
```

2.1: Node ID Generation

Once a HD identity has been generated, a child identity should be derived and used for a single node. The resulting public key from that child identity is used to derive the Node ID. The node's identifier is the `RIPEMD160(SHA256(CHILD_PUBLIC_KEY))` encoded in hexadecimal. This value is inserted as the first item in the identity tuple.

```
[ "705e93f855e60847fda4c48adff0dc1b1f7c40ef", { /* <contact_hash_map> */ } ]
```

2.2: Contact Hash Map

The second entry in the identity tuple contains additional information specific to addressing the node on the network. This includes:

```
{
  "hostname": "ip.address.or.domain.name",
  "port": 8443,
  "protocol": "https:",
  "xpub": "<child_identity_public_extended_key>",
```

```
    "index": "<child_identity_derivation_index>"
  }
```

Additional properties may be included based on individual use cases within the network, however the properties above are **required**.

3: Network Structure

TODO

3.1: Kademlia

TODO

3.2: Quasar

TODO

3.3: Transport

TODO

4: Remote Procedure Calls

TODO

4.1: Structure and Authentication

TODO

4.2: PROBE

TODO

4.3: PING

TODO

4.4: FIND_NODE

TODO

4.5: FIND_VALUE

TODO

4.6: STORE

TODO

4.7: SUBSCRIBE

TODO

4.8: UPDATE

TODO

4.9: PUBLISH

TODO

4.10: OFFER

TODO

4.11: CONSIGN

TODO

4.12: AUDIT

TODO

4.13: MIRROR

TODO

4.14: RETRIEVE

TODO

4.15: RENEW

TODO

4.16: ALLOCATE

TODO

4.17: CLAIM

TODO

4.18: TRIGGER

TODO

5: Data Transfer Endpoints

TODO

5.1: Uploading

TODO

5.2: Downloading

TODO

6: Storage Contracts

TODO

6.1: Descriptor Schema

TODO

6.2: Topic Codes

TODO

7: Retrieval Proofs

TODO