



Bern University
of Applied Sciences

Informatikseminar

Bitmessage – Communication Without Metadata

Author Christian Basler
Tutor Kai Brännler
Date May 6, 2015

Contents

1. Synopsis	3
2. Basics	3
3. Protocol	3
3.1. Nomenclature	3
3.2. Process Flow	4
3.3. Messages	4
3.3.1. version / verack	4
3.3.2. addr	4
3.3.3. inv	4
3.3.4. getdata	5
3.3.5. object	5
3.3.6. ping / pong	5
3.4. Addresses	5
3.5. Encryption	5
4. Issues	6
4.1. Scalability	6
4.1.1. Streams	6
4.1.2. Prefix Filtering	6
4.2. Forward Secrecy	6
5. Discussion	7
Appendix	7
A. TODO	7

1. Synopsis

TODO

2. Basics

While encryption technology like PGP or S/MIME provides a secure way to protect content from prying eyes, ever since Edward Snowdens whistleblowing we learned that metadata — most notably information about who communicates with whom — is equally interesting and much easier to analyze.

With e-mail, we can only prevent this by encrypting the connection to the server as well as between servers. Therefore we can only hope that both our and the recipient's e-mail provider are both trustworthy as well as competent.

With Bitmessage we send a message to a sufficiently large number of participants, with the intended recipient among them. Content is encrypted such as only the person in possession of the private key can decrypt it. All participants try to do this in order to find their messages.

3. Protocol

3.1. Nomenclature

There are a few terms that are easily mixed up. Here's a list of the most confusing ones:

<i>message</i>	is sent from one node to another, i.e. to announce new objects or to initialize the network connection.
<i>msg</i>	is the object payload containing the actual message written by a user. The term 'message' is never used to describe information exchange between users in this document. 'Content' is mostly used instead.
<i>payload</i>	There are two kinds of payload: message payload for message types, e.g. containing inventory vectors, and object payload, which is distributed throughout the network.
<i>object</i>	is a kind of message whose payload is distributed among all nodes. Sometimes just the payload is meant.

3.2. Process Flow

The newly started node **A** connects to a random node **B** from its node registry and sends a *version* message, announcing the latest supported protocol version. If **B** accepts the version¹, it responds with a *verack* message, followed by a *version* message announcing its own latest supported protocol version. Node **A** then decides whether it supports **B**'s version and sends its *verack* message.

If both nodes accept the connection, they both send an *addr* message containing up to 1000 of its known nodes, followed by one or more *inv* messages announcing all valid objects they are aware of. They then send *getobject* request for all objects still missing from their inventory.

Getobject requests are answered by *object* messages containing the requested objects.

If a user writes a new mail on node **A**, it is offered via *inv* to up to eight connected nodes. They will get the object and distribute it to up to eight of their connections, and so on.

3.3. Messages

The messages, objects and binary format are very well described in the Bitmessage wiki [1], the message description is therefore narrowed down to a description of what they do and when they're used.

3.3.1. version / verack

A *version* message contains the latest protocol version supported by a node, as well as the streams it is interested in and which features it supports. If the other node accepts, it acknowledges with a *verack* message. The connection is initialized when both nodes sent a *verack* message.

3.3.2. addr

Contains up to 1000 known nodes with their IP addresses, ports, streams and supported features.

3.3.3. inv

One *inv* message contains the hashes of up to 50000 valid objects. If your inventory is larger, several messages can be sent.

¹A version is accepted by default if it is higher or equal to a nodes latest supported version. Nodes supporting experimental protocol versions might accept older versions.

3.3.4. getdata

Can request up to 50000 objects by sending their hashes.

3.3.5. object

Contains one requested object, which might be one of:

<i>getpubkey</i>	A request for a public key, which is needed to encrypt a message to a specific user.
<i>pubkey</i>	A public key. See 3.4 Addresses
<i>msg</i>	Content intended to be received by one user.
<i>broadcast</i>	Content sent in a way that the Addresses public key can be used to decrypt it, allowing any subscriber who knows the address to receive the such a message

3.3.6. ping / pong

3.4. Addresses

BM-2cXxfcSetKnBHJX2Y85rSkaVpsdNUZ5q9h: Addresses start with "BM-" and are, like Bitcoin addresses, Base58 encoded².

<i>version</i>	Address version.
<i>stream</i>	Stream number.
<i>ripe</i>	Hash of both public signing and encryption key. Please note that the keys are sent without the leading 0x04 in pubkey objects, but for creating the ripe it must be prepended. This is also necessary for most other applications, so it's a good idea to do it by default. <code>ripemd160(sha512(pubSigKey + pubEncKey))</code>
<i>checksum</i>	First four bytes of a double SHA-512 hash of the above. <code>sha512(sha512(version + stream + ripe))</code>

3.5. Encryption

TODO

²Which avoids the easily confused characters l, I, 0 and O.

4. Issues

4.1. Scalability

Bitmessage doesn't really scale. If there are very few users, anonymity isn't given anymore, and with many users traffic and storage use grows quadratically.

4.1.1. Streams

The intended solution for this problem is splitting traffic³ into streams. When all active streams are full, a new one is created which should be used for new addresses. All users can send messages to any stream, but only listen to the streams belonging to their addresses. The unsolved problem is to determine when a stream is full. The other issue is the fact that, as the overall network grows, traffic on full streams still grows, as there are more users who might want to write someone on the full stream.

4.1.2. Prefix Filtering

TODO

4.2. Forward Secrecy

Obviously it's trivial for an attacker to collect all (encrypted) objects distributed through the Bitmessage network⁴. If this attacker can somehow get the private key of a user, they can decrypt all stored messages intended for that user, as well as impersonate said user⁵.

Plausible deniability can, in some scenarios, help against this. This action, called "nuking an address", is done by anonymously publishing the private keys somewhere publicly accessible⁶.

Perfect forward secrecy seems impractical to implement, as it requires to exchange messages prior to sending content. That would in turn need proof of work to protect the network, resulting in twice the work for the sender and three times longer to send — that is if both clients are online.

³Addresses, actually

⁴As long as disk space is not an issue.

⁵The latter might be more difficult if they got the key through a brute force attack.

⁶So <https://bitmessage.ch/nuked/> for an example.

5. Discussion

TODO

References

[1] Atheros and Jonathan Coe. Bitmessage wiki: Protocol specification, 2015.

Appendix

A. TODO