Freemail - Specification and Protocol Documentation

May 2011

This is a working draft of the Freemail specification. All parts of this document are subject to change.

1 Introduction

1.1 What is Freemail

Freemail is an email-like messaging system that transports all messages over Freenet 0.7 in order to achieve anonymity and censorship-resilience. Its protocol is designed to be as resistant as possible to attacks such as message floods and denial of service. Unlike traditional email, it makes it extremely difficult for others to discover what you have been communicating, who you have been communicating with, and even that you have been communicating at all.

Freemail uses IMAP and SMTP to interface with standard email clients, taking advantage of interfaces that people are already accustomed to.

2 Channel Setup

2.1 Mailsites

Before any communication occurs between a sender and a recipient (who, in accordance with cryptography tradition shall be called Alice and Bob, respectively), a channel is setup that is used between those two and only those two parties. This channel comprises a Freenet SSK keypair to which both Alice and Bob has the private key, permitting two-way communication between those two parties.

All Freemail users have an Freemail address, which one may give out to others in order to allow them to contact you. From the Freemail address it is possible to derive an identity in the web of trust that can be used to find the user's 'mailsite'.

A Freemail address comprises an arbitrary text string, followed by an '@' character. Following this is the base32 encoded representation of the hash of the public key (the first part of an SSK is this value base64 encoded). The URI must be base 32 encoded in order to make the address case insensitive to maintain compatibility with traditional email clients. The string '.freemail' is appended to the whole address. For example, the identity `D3MrAR-AVMqKJRjXnpKW2guW9z1mw5GZ9BB15mYVkVc` has the Freemail address <anything>@b5zswai7ybkmvcrfddlz5euw3ifzn5z5m3bzdgpucb26mzqvsflq.freemail

2.2 Mailpages

Once the full key for an identity has been obtained the mailpage can be located at USK@<key of the identity>/mailsite/<edition>/mailpage. The latest edition should be published through the Web of Trust plugin using the property Freemail.mailsite. The format of a mailpage is a 'Props File', which is used repeatedly in Freemail as a trivial format for storing short pieces of information. See section **??**. The following pieces of information are required in a mailpage:

asymkey.modulus The modulus of the owner's RSA encryption key, as an integer in base 32.

asymkey.pubexponent The public exponent of the owner's RSA encryption key, as an integer in base 32.

2.3 RTS Messages

Once Alice has retrieved the recipient's mailpage, she sends an RTS (Request To Send) message to Bob. This RTS message is, again, a props file, with the following keys:

messagetype This should be 'rts', to indicate that this message is an RTS.

mailsite The mailsite key of the sender

recipient The WoT identity of the recipient

privkey The private key of the channel

fetchslot The initial slot the recipient should use for fetching messages

sendslot The initial slot the recipient should use for sending messages

Following the last data item, there are two carriage-return-line-feeds, followed by Alice's signature. This is the SHA-256 hash of the message RSA encrypted with Alice's private key, included as raw bytes.

The final message comprises a randomly generated IV, followed by an AES session key, encrypted with Bob's public RSA key, followed by this message-signature combination encrypted with this session key. The encrypted session key must be precicely one RSA block of ciphertext. All bytes after this are part of the symmetrically encrypted message. The main message is encoded with a block size of 128 bits in CBC mode, with PKCS7 padding.

It is both parties responsibility to keep the private key private. Since only the two parties involved have the private key, any message that was not inserted by Alice can be assumed to be from Bob and vice versa.

The 'recipient' field is included to prevent surreptitious forwarding. That is, to prevent Bob from decrypting the message, leaving Alice's signature intact and encrypting it to someone else (say, Charlie), who would then be lead in to believing that Alice wished to communicate with him, which is fact not the case.

This RTS message is then inserted using the Web of Trust plugin to a key derived from the solution to a CAPTCHA the recipient inserted to Freenet and the sender solved.

Both Alice and Bob then regularly polls the initial slot they use for fetching messages for new messages or acknowledgements.

2.4 CTS Messages

When Bob receives an RTS message from Alice, he decrypts the message using his RSA private key. He then retrieves the mailsite advertised in the RTS message. Having done this, he reads the signature on the end and decrypts the signature with the public key he just retrieved from the mailsite. He then calculates a SHA-256 checksum of the message and checks that his checksum is identical to the one he has decrypted. If it is not, he must discard the message. This ensures that the message is really from Alice. He must then read the 'recipient' field and ensure that its value is identical to his mailsite URI. If it is not, he must discard the message. This ensures that he is the intended recipient of the message.

Bob then records the public and private key so that he can poll the key for new messages.

Before doing so, Bob inserts a props file containing messagetype=cts to the first availiable slot. This completes Bob's part of the channel setup procedure.

Alice should check periodically for the insertion of this CTS (Clear To Send) message. If it does not arrive, Alice should re-send the RTS message. The client may try several times before declaring the message undeliverable.

3 Message Exchange

3.1 The Messages

Messages are inserted to a key derived from the 'privkey' field and one of the initial slots. The party that sends the RTS, the initiator, uses the fetchslot for inserting messages and the sendslot for receiving messages, while the recipient uses the slots for the opposite purpose. The full key used for inserting messages is then SSK@<privkey>/<role>-<slot>, where role is i for the initiator and r for the recipient. The slot is initially the value indicated in the RTS. Each subsequent slot is the SHA-256 digest of the previous slot, forming a seqence of message slots. This hash sequence gives forward security, provided that clients destroy values of the slots, and the inital slot once they have been used. Alice should insert a new message to the first slot to which inserting does not causes a collision. Formulaically, Bob first polls the key:

SSK@<SSK key base>/i-<fetchslot>

Once Bob has successfully retrieved this key, he begins to periodically request the key:

SSK@<SSK key base>/i-<H(fetchslot)>

It is recommended that clients poll several messages ahead rather than just the immediately next message, since simulations suggest that it is possible for single keys not to be retrievable in this kind of circumstance. So for example, once Bob has sent his CTS messages, he should start polling for the keys:

SSK@9GXtGxN4CEJD 8a307V6yzyhl8Gx5UYbWVDTEyUXH6o,gDWfr2CqVm-DAeJurKF2iieM5AkjXstOl2V5jyuTHeo4,AQABAAE/i-<fetch slot>

SSK@9GXtGxN4CEJD 8a307V6yzyhl8Gx5UYbWVDTEyUXH6o,gDWfr2CqVm-DAeJurKF2iieM5AkjXstOl2V5jyuTHeo4,AQABAAE/i-<H(fetch slot)>

and

SSK@9GXtGxN4CEJD 8a307V6yzyhl8Gx5UYbWVDTEyUXH6o,gDWfr2CqVm-DAeJurKF2iieM5AkjXstOl2V5jyuTHeo4,AQABAAE/i-<H(H(fetch slot))>

Alice can insert a message to the lowest numbered key of this pattern that does not cause a collision whenever she chooses. These comprise a number of properties (in the same way as a props file) followed by a double carriage-return-line-feed. Following this is the standard MIME mail messages. No signing or encryption is used here, since at this stage it is achieved inside Freenet by virtue of only Alice and Bob knowing the SSK upon which they communicate.

The mandatory properties of the props file are 'id' and 'messagetype=message'. 'id' is a string that must uniquely identify the message from any other past or future message transported using the same channel. Bob must check this value and ensure that he has not already received this message id. If he has, he discards the message, but still acknowledges his receipt of it. All clients must ignore any unknown keys and begin reading the message only at the double line break in order that extra properties can be added in the future.

3.2 Message Acknowledgements

When Bobs client receives a message, it reads it and passes it onto Bob. He then inserts a propsfile to the next availiable slot with 'messagetype' set to ack, and 'id' set to a comma separated list of the 'id' values of the messages that should be acked. For example, if Bob has just fetched a message from key:

SSK@9GXtGxN4CEJD 8a307V6yzyhl8Gx5UYbWVDTEyUXH6o,gDWfr2CqVm-DAeJurKF2iieM5AkjXstOl2V5jyuTHeo4,AQABAAE/<role>-<fetch slot>

With the contents:

```
messagetype=message
id=657488664753

To: Bob Burton <bob@longkey.freemail>
From: Alice Andrews <alice@anotherlongkey.freemail>
Subject: Eve

I think Eve from down the road might be trying to spy on us. I've never liked
the looked of her, you know. It's always the quiet ones.
```

Then he might insert an acknowledgement to the key:

SSK@AJoZbUvGkXlAJwIjdbu9BLPhpIXBu6w6nGwKYBnMfNLi,ACEgE1uUIzJdC-Xcsz1yjgW45uAz-KuMrXBFYGU8maqc/<role>-<send slot>

With the content:

```
messagetype=ack
id=657488664753
```

If Alice has, for whatever reason, not received a CTS message from Bob, her receipt of a message ack should additionally be treated as receipt of a CTS message.

A Props Files

A props file is a sequence of keys and values. Keys and values are separated by a single equals sign ('=') and lines are separated by a carriage return and line feed (\r\n), with the exception that if the propsfile will only be read locally, it is permissable to use the line separator native to the local machine. For example, for props files that are never transmitted over the network, it is permissable to use just a line feed (\n) to separate lines. It is recommended for simplicity, though not required, that the keys be lowercase and contain only alphanumeric characters. The keys must not contain the equals sign, as there is no mechanism for escaping equals signs. The value may contain equals signs and therefore parsers of this format must treat and equals signs after the first on any line as part of the value text.

An example of a propsfile is below:

```
name=Bob Burton
age=39
occupation=Builder
Pet's Name=Stevie the Sycophantic Squirrel
```