Confidant Mail spec v0.2
Mike Ingle <mike@confidantmail.org> d2b89e6f95e72e26e0c917d02d1847dfecfcd0c2
SMTP inglem@pobox.com

Confidant Mail is a non-SMTP cryptographic email system for TCP/IP. It has its own client, server, and protocol. It is not compatible with SMTP, and any attempt to build gateways would compromise its security. Confidant Mail is intended to be used in parallel with legacy email where its capabilities are needed. It uses GNU Privacy Guard (GPG) for content encryption and authentication, and TLS 1.2 with ephemeral keys for transport encryption.

In exchange for breaking backward compatibility, Confidant Mail provides these benefits:

Encryption and signatures: all messages are automatically encrypted and signed with GPG.
Automatic key management: keys are looked up as needed using DNS or Kademlia DHT.
Traffic analysis resistance: a passive snooper cannot see who is communicating with whom.
Unlimited attachment length: files over 4GB have been tested and work.
Piecemeal downloading: large messages are transferred in hash-checked blocks, like BitTorrent.
Automatic acknowledgment: sent messages show which recipients have received them.
Bulk mail protection: a user-configurable, Bitcoin-style proof of work makes junk mail costly.
Forward with signature: forwarding an email lets the recipient check the original signature.
Server independence: change servers without changing your email address.
Ease of running a home server: dynamic DNS and home broadband works well.
High availability: servers can be paired so that if either one fails, your email still works.
Optional anonymity: support for TOR and I2P is built into both server and client.
SMTP-compatible address format: keep your existing email address.

Blocks:

The Confidant Mail storage system is a key-value system using 160-bit binary hash keys. These are often represented by 40-character hexadecimal strings. This 160-bit key format matches both GPG key fingerprints and Kademlia/Entangled distributed hash tables. Each block in the system has a 160-bit hash key, and in most cases the key is determined by the block's contents using the SHA1 algorithm.

There are presently five types of blocks:

Type "key-announcement": represents a GPG public key along with the address and port of the key owner's mail server(s), and other information required to contact the key owner.
Type "address-claim": maps a SMTP-style user@domain email address to a GPG key.
Type "message-announcement": informs a recipient of an incoming email, and lists its data blocks.
Type "acknowledgment": informs a sender that his message has been received and decrypted.
Type "data": contains a message or portion thereof.

Sample "key-announcement" block:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

ProofOfWork: bd,0000004a8,0000017b4
Type: key-announcement
Version: 1
Date: 2014-11-08T09:23:00Z
Expires: 2014-11-15T09:23:00Z
Userid: William Wreck <wreck@pobox.com>
Transport: server=localhost:8081,localhost:8082
Mailboxes: 0,1,2,3
SenderProofOfWork: bd,24,2

- -----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQENBFRTTlIBCADCQEAJE9aDvof6+Emqu27K112FMsQjZlMLFU8laUaMIVVR2jDF
EH/XqtR6MdV/tYDOkxGlnr57jXjtdOJJ9hN23c9q2C6TOf2q1pr/AWxSHd5krfmG
TDPzR0oPjnwZeCoZkD9WYQ7u5pYXwIRF7DQrLEpX8W0zP7jOLtpDXTVBouTRRdM3
Nz3Ytk2pKq90uyheQtz/LfpbLQvDOKxE97S6F9poGm/peRaqmCZw73KTLBZEc8D7
e8MZZy0g//YuruuVry439+tX8ojuaQagb9M5SXlZIoLW8wP1hxxlefiDFdthccn/
/If4P13odlI6KObBuJkdNXom3mCuBmQsV0b3ABEBAAG0H1dpbGxpYW0gV3JlY2sg
PHdyZWNrQHBvYm94LmNvbT6JATcEEwECACECGy8CHgECF4AFAlRdneUGCwkIBwMC
BRUKCQgLBBYCAwEACgkQtPXEHOPlln0bHAf/aHG4EKtWwxskjXnB2muRUL490HoT
MXiaZsSLjfUHu4f2pg9pdrAzRQ+fb5ThxHsdJXtGqMk0ib5vneU5k4O//WqXwm2q
PCRRMB8sb5oGO62GfUZ6hLUC0vgWqhFhYPA8MreXJ3AVcRVnJCMoVPuel1y3RBG0
lRXOBGboiC7TVGDUlCECEk+agKDSELF1HSYFpxOCXFRwvVoNLLwp1Nc8x4XDeyhh
Tq/47cjv1KZb/O4oZnJwaOM6vPCHai0O4LoV/cahHXGgHpYDYl7KPAiNmDrMgXD1
j+QTMQQkrorBfOOUq4DkeGaED8eM4OlQtGwR17xdjsbY1XM/oK+Qq+E2SA==
=Q8HZ
- -----END PGP PUBLIC KEY BLOCK-----

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1

iQEcBAEBCgAGBQJUXeD1AAoJELT1xBzj5ZZ9GAYH/0+kTQ9vq2hrJMI1V1ZbIgFM
sDysh1Xv4cXkPu2SxGIg8QXX3CqPYsufNqvRp2KEa93QUVqtYvM46uDy8Rt1fu1u
L+1Fv6RMfUJNsFGSxC5y+FNj5JslDhPrfxBITQtVoWfwRn6VpXx2pl/ggXIX4mgS
IE0TmN3/03A9sKTK2xxIsupXkvQoOgwi/cS9zPNAMa60lKbsLcF2ZHUgTpIg7ush
IhoduyH+5SRLd+zOaUjgq5Z4u1qFkJzk28gOOLWq8PXcJeG0ghDGUN+HkH4aM8Qy
ShrTnpg8x6fk4FJjjkiLYydlhZ8/v1v0noS6QpRJM2/FIkI6gE2z74y7KiAfX3I=
=QWx4
-----END PGP SIGNATURE-----
```

The key announcement block contains a GPG key, and is self-signed using the same key in clear-signature mode. The hash key of this block is C3202B03EB24B1A156DD2F2BB4F5C41CE3E5967D, the GPG fingerprint of the public key. For a key-announcement, the hash key of the block must be the fingerprint of the key being published.

In addition to the key and signature, some fields are included in "field: value" form:

ProofOfWork: this is a "payment" used to prevent bulk mail or abuse of the Kademlia DHT. It is an algorithm that is intentionally somewhat costly to compute. The algorithm will be explained later.

Type: identifies the block type.

Version: identifies the software and data format version. This will be incremented to allow newer software versions to communicate with older versions.

Date: Date and time the key was posted, in UTC form.

Expires: Date and time the key should be re-fetched by the client.

Userid: Username and SMTP-style email address for this key.

Transport: Network addresses of the user's email servers. This can also be "entangled" for users who receive email directly from the Entangled DHT. Note that the message lengths are limited and proof-of-work costs are intentionally high when using Entangled. When multiple servers are specified, the sender should contact them in random order, and stop when the message has been successfully delivered to any one server in the group.

Servers are specified as host:port, and the host part can be a DNS name or an IPv4 numeric address. IPv6 numeric addresses will be supported in the future. As a special case, DNS names ending in ".onion" are TOR hidden services, and DNS names ending in ".i2p" are I2P hidden services.

Mailboxes: this is a prefix used to compute the hash code for the sender's message-announcements. To prevent one block from becoming excessively large, one user can receive email at multiple hash keys. In this case, we take the ASCII character "0" and append the binary key hash C3202B03EB24B1A156DD2F2BB4F5C41CE3E5967D (21 bytes in total are being hashed), take the SHA1, and get the email hash key A47D8BADFF5E0458EFB4974006E0F49A5FE0AC5A. Prepending 1, 2, and 3 gives 63840CD939FE26C996119587BDE58434E5DDF6B4, 5FD8D56FCA0E583C6D27B2D69053B88C27AC3C37, and 2D3CAA730BAF6C6E846E02B3AA6EC509203FBEA5. A message-announcement posted to any of these hash keys will be addressed to this user. The sender should choose a mailbox at random.

SenderProofOfWork: this specifies the amount of work required of the sender who wants to message this user. The user can set the work requirement to limit the amount of bulk mail he receives. A future version will have a token mechanism to permit trusted senders to bypass the work requirement.

BypassTokenAccepted: 2015-04-14T20:48:00Z
New addition in v0.31; this indicates that the client accepts bypass tokens from known senders, and gives the date of the earliest incoming bypass token the client has. Senders can use any bypass token dated later than this in lieu of a proof of work.

The proof of work for the key-announcement is computed on this data, with all CR and LF characters removed. The ***BEGIN DATA and ***END DATA lines are not included.

```
***BEGIN DATA FOR PROOF OF WORK***
Type: key-announcement
Version: 1
Date: 2014-11-09T08:10:27Z
Expires: 2014-11-16T08:10:27Z
Userid: William Wreck <wreck@pobox.com>
Transport: server=localhost:8081
Mailboxes: 0,1,2,3
SenderProofOfWork: bd,24,2

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQENBFRTTlIBCADCQEAJE9aDvof6+Emqu27K112FMsQjZlMLFU8laUaMIVVR2jDF
EH/XqtR6MdV/tYDOkxGlnr57jXjtdOJJ9hN23c9q2C6TOf2q1pr/AWxSHd5krfmG
TDPzR0oPjnwZeCoZkD9WYQ7u5pYXwIRF7DQrLEpX8W0zP7jOLtpDXTVBouTRRdM3
Nz3Ytk2pKq90uyheQtz/LfpbLQvDOKxE97S6F9poGm/peRaqmCZw73KTLBZEc8D7
e8MZZy0g//YuruuVry439+tX8ojuaQagb9M5SXlZIoLW8wP1hxxlefiDFdthccn/
/If4P13odlI6KObBuJkdNXom3mCuBmQsV0b3ABEBAAG0H1dpbGxpYW0gV3JlY2sg
PHdyZWNrQHBvYm94LmNvbT6JATcEEwECACECGy8CHgECF4AFAlRdneUGCwkIBwMC
BRUKCQgLBBYCAwEACgkQtPXEHOPlln0bHAf/aHG4EKtWwxskjXnB2muRUL490HoT
MXiaZsSLjfUHu4f2pg9pdrAzRQ+fb5ThxHsdJXtGqMk0ib5vneU5k4O//WqXwm2q
PCRRMB8sb5oGO62GfUZ6hLUC0vgWqhFhYPA8MreXJ3AVcRVnJCMoVPuel1y3RBG0
lRXOBGboiC7TVGDUlCECEk+agKDSELF1HSYFpxOCXFRwvVoNLLwp1Nc8x4XDeyhh
Tq/47cjv1KZb/O4oZnJwaOM6vPCHai0O4LoV/cahHXGgHpYDYl7KPAiNmDrMgXD1
j+QTMQQkrorBfOOUq4DkeGaED8eM4OlQtGwR17xdjsbY1XM/oK+Qq+E2SA==
=Q8HZ
-----END PGP PUBLIC KEY BLOCK-----
***END DATA FOR PROOF OF WORK***
```

Sample "address-claim" block:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

Type: address-claim
Version: 1
Date: 2014-11-09T08:32:25Z
ProofOfWork: bd,000000957,0000013ba
Keyid: C3202B03EB24B1A156DD2F2BB4F5C41CE3E5967D
Address: wreck@pobox.com
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1

iQEcBAEBCgAGBQJUXyaaAAoJELT1xBzj5ZZ9npMIAIbXQVg7GvZvT4GDKfP6KQ3M
hDPlh9cHOmGXzDnQFGngnmrg+JKvMvt+zb5KHYKSADW/MBN3x0GA1KjzXsDcyCKo
QgUuyjOXnDYEWXS98bEz26xkXRZ4ZeIFNVzS9cBan/ryeYZcsDMV5qbEGlBfl2Gi
VDmH+/9QEIDXWpZRuxmo04BCcE5Dskm+lvSVcjr6Jr63JPPBBf09WDkxH3z0gxCO
zH0l8vGMo9Uws6w6qYqVkPBwrfLuNArg+M0QJjIq0N20qTBXIDKLGqtJ57KdLHR7
8Vz3tY3p4XEwilDb9c/yuLjkNlHGOCQYQuV7fx0HVLb8GueLQat08BpimTWxlLo=
=JAbk
-----END PGP SIGNATURE-----
```

This is a clear-signed object which maps an SMTP-style email address to a GPG key. It must be signed by the keyid specified. An address claim is just a claim. There is no mechanism to certify that this key is the rightful owner of the corresponding email address. The claim is equivalent to "the owner of the key X would like to be contacted using the email address Y".

The hash key of an address-claim block is the SHA1 of the email address in the Address: line. This block has the hash key CEC2D4AC1EFAD403ADD1453034EA9A231277195A. The email address is converted to lower case and Unicode characters are represented in UTF-8.

More than one key may claim the same email address. The system has no mechanism to resolve this, and the user must choose the correct key fingerprint. If there are multiple claims, the block will contain all of them concatenated with "NextClaim" on a line by itself in between claims.

```
...
-----END PGP SIGNATURE-----
NextClaim
-----BEGIN PGP SIGNED MESSAGE-----
...
```

The proof of work for the address-claim is computed on the date, binary Keyid, and address concatenated, with no newlines and without the quotes, like this:
"2014-11-09T08:32:25Z" + binary(C3202B03EB24B1A156DD2F2BB4F5C41CE3E5967D) + "wreck@pobox.com"

A future version may assign a basic level of trust based on DNS lookup and servers having commercial X.509 certificates. The server already uses SSL so this would not be difficult to add.

Sample "message-announcement" block:

```
Type: message-announcement
Date: 2014-11-09T07:49:41Z
Version: 1
Recipient: c3202b03eb24b1a156dd2f2bb4f5c41ce3e5967d
Mailbox: 1
ProofOfWork: bd,000000915,000002a05
DataBlock: 2bb16080c26c54867668ca8262cdc02c2d500641
DataBlock: 079117a94ece35366ebf36bd3334bf6200f617ec
DataBlock: d9857dd9a707d31ce7ced0cb00cd563e8de44936
DataBlock: 4bc3193fb6ad432be9d7ff7f84a2b27eade125bc
DataBlock: e887eab6c8b1d638dc8c7ee1f32171948d315497
DataBlock: f432a74e711d9c4ef3b784941943994c4d0999ed
DataBlock: 44f953cce64db9d74f0f99f4424e804cb032ab8e
DataBlock: 7acf9eb6c1a5c087d3e1dcc6f7e3155f0d844785
MessageHash: 565dec1ea7bf9897503e3b473454b4fa42c3ab70
NextMessage
Type: message-announcement
Date: 2014-11-08T05:25:22Z
Version: 1
Recipient: c3202b03eb24b1a156dd2f2bb4f5c41ce3e5967d
Mailbox: 1
ProofOfWork: bd,000000643,000001c77
DataBlock: 02a424f444d12c56916a24cd65907e11ab51ce53
MessageHash: 02a424f444d12c56916a24cd65907e11ab51ce53
```

The hash key of this block is 63840CD939FE26C996119587BDE58434E5DDF6B4, obtained by prepending the mailbox "1" to the binary recipient shown, and taking the SHA1. There are two messages with "NextMessage" between them. In in most cases there will be multiple messages.

The DataBlock lines give the hash keys of the data blocks for this message. There may be one or more DataBlock lines depending on the length of the message. The message is reassembled by concatenating the binary data of the data blocks in the order listed.

The MessageHash line is the hash of the entire binary data file, and will be identical to the DataBlock for a single-block message.

The proof of work for the message-announcement is computed on the concatenation of the binary recipient ID (20 bytes), the binary message hash (20 bytes), the date (20 bytes), and the concatenated binary data block IDs (20 bytes * n).

Note that the message-announcement does not specify the sender. This is intentional to prevent bulk traffic analysis by servers. The recipient must decrypt the message to see who it is from.

BypassToken: 5af81b78163126d3c7bff19ce28c4c1e78291ea5

The bypass token can optionally be added to the message-announcement, assuming the recipient advertised this capability with BypassTokenAccepted. The bypass token is computed based on a shared secret previously received from the intended recipient of this message. The shared secret is a 20-byte value. The bypass token is the HMAC_SHA1 with the shared secret as the key and the same data used for the proof of work (described above) as the data. This token should not reveal anything to observers who do not have the shared secrets.

If a bypass token is provided, the sender does not have to do the Sender Proof Of Work, but can instead provide the (likely much easier) proof of work for the server or Entangled. This permits a recipient to set a high threshold for strangers, while eliminating this cost for known senders. The recipient must search his list of shared secrets and test hash each one. If one matches, the message will be accepted without checking the proof of work.

Shared secrets are received as part of the acknowledgment secret ACK_*.PGP, and are sent along with each message, so once you have received a message from someone, you have a shared secret and can skip the proof of work.

If the ACK_*.PGP decrypts to a 20-byte binary string, this is the acknowledgment hash and no bypass secret has been provided. If it decrypts to an ASCII string longer than 20 bytes, split the string with pipe (|) as a separator. The string "bt=<hex secret>,<create date>,<expire date>" is the bypass secret, while "ack=<hex value>" is the acknowledgment secret. Any unknown values should be skipped for forward compatibility. The bypass secret can be used to generate bypass tokens between the create and expire dates, provided the create date is later than the date advertised in BypassTokenAccepted. Expire date may be "never" and dates are otherwise YYYY-MM-DDTHH:MM:SSZ.

The present client saves these secrets in a file called bypass_tokens.txt, uses all non-expiring secrets, and has a "Full Get" option to get any messages with invalid proof of work and unknown bypass tokens. If the user is forced to restore from a backup or otherwise loses bypass tokens, the whole bypass_tokens.txt file should be deleted, and the user should do Full Gets for a couple of weeks until all recipients have refetched the key announcement.

Sample "data" block:

```
Type: data
Date: 2014-11-09T07:49:41Z
Version: 1
ProofOfWork: bd,0000000ca,00000018b
Data: 8388608
<8,388,608 binary bytes>
```

The data block contains the message content. The hash key of the data block is the SHA1 of the data. Data blocks are stored and transmitted in binary. After the "Data: 8388608" is a CR, LF, and the next byte is the first binary data byte.

The proof of work is computed on the 20-byte date followed by the binary data. This makes the cost proportional to the size, and discourages filling up the DHT with large files.

To reassemble the message, the data blocks are concatenated in the order specified in the message-announcement, and the result is a GPG-encrypted binary file.

Sample "acknowledgment" block:

```
Type: acknowledgment
Date: 2014-10-31T09:02:22Z
Version: 1
ProofOfWork: bd,00000014f,0000001f5
Hash: 15f8b83f6380c2a25a0249cc615ec7e58a9dcf71
```

The hash key of this block is DA458F9588E1B4F9F85EE79E8DEB4DD85B448092, which is obtained by taking the SHA1 of the binary Hash: field. Each message header (explained later) has a public (visible to all recipients) acknowledgment value for each user, in this case DA458F9588E1B4F9F85EE79E8DEB4DD85B448092.

Each recipient in a multi-recipient message receives an encrypted message revealing the Hash value, in this case, 15f8b83f6380c2a25a0249cc615ec7e58a9dcf71. By publishing this hash value, the recipient proves to the sender that he decrypted the message successfully.

This is a "weak" proof in the sense that the sender could not prove anything in court. The sender knows the target hash value already, and could fake the acknowledgment. However, the sender can be sure that the message was received by the intended recipient. Note that other recipients could, if so motivated, check to see which of their co-recipients have acknowledged the message. They could not fake acknowledgments for other recipients.

The Date field is the date of receipt. This is not cryptographically protected and could be forged. The proof of work is computed on the 20-byte date plus the 20-byte binary Hash.

Proof of work algorithm:

The goal of the proof of work is to make bulk advertising expensive for the sender, and to prevent the abuse of the Kademlia DHT. The proof of work algorithm should have adjustable difficulty and should be computationally easy to verify.

Bitcoin uses a simple of proof of work which requires adding a nonce to the message and searching for a nonce value that produces a SHA1 hash with a large number of leading zero bits. This is easy to do in hardware, and has produced a wasteful arms race of custom hardware to perform Bitcoin mining. Ordinary users with computers have been shut out of this process.

The Confidant Mail proof of work uses SHA256, and is based on the Birthday Paradox. This refers to the fact that if you want two strings whose hashes have the identical first N bits, you must on average test only 2^(N/2) strings. The string:
SenderProofOfWork: bd,24,2
requires two nonces which, prepended to the string being hashed, have the same first 24 bits of hash value. This should take about 2^12 or 4096 hashes to find.

For example, take the string "This is a sample string".

>>> print proofofwork.generate_proof_of_work("This is a sample string",48,2)
(just under a minute passes)
bd,00022ab21,0003b3307

The SHA256 of "00022ab21This is a sample string" is:
e98f1825d48f299173a8e37d2682a7e7d1de02b401f589e4778485a478e3c20b

The SHA256 of "0003b3307This is a sample string" is:
e98f1825d48f3eabeec8fe2163bdf800461e19464fa995b55330d6c937168f62

The first 48 bits are the same, so this is a valid proof of work of level 48. It can be verified with only two SHA256 operations and some Boolean operations.

The efficient algorithm to find these is to generate and store hash values and nonces in an associative memory until a collision is found. This requires dynamic allocation of a fair amount of memory, and is therefore difficult to perform on a GPU, FPGA, or custom hardware. Hopefully this will prevent bulk mailers from using GPUs to brute-force the algorithm.

The proof-of-work generator will probably need to be rewritten in compiled code so as to keep up with the commercial bulk mailers using optimized implementations.

As of version 0.31, there is an opaque BypassToken that can be included in the message-announcement so that trusted users can bypass the proof of work. This allows much higher thresholds of work to accept mail from a stranger than are currently allowed. The token is designed so that it does not reveal the sender's identity to outsiders.

Wire protocol:

The Confidant Mail wire protocol is a stream protocol running over TCP. It is a text-based, line-oriented protocol, with the exception of DATA blocks which are transmitted in binary form. Here S> refers to messages from the server and C> refers to messages from the client. The S> and C> do not actually appear on the wire. Lines are terminated by the CR,LF (0x0D,0x0A) combination.

Upon accepting a connection, the server responds with:
CONFIDANT MAIL SERVER PROTOCOL 1 READY
The protocol version will be incremented in the future if features are added. The present client accepts any integer protocol version. The server may also respond with something like:
CONFIDANT MAIL SERVER PROTOCOL 1 BUSY
and immediately close the connection. Any sequence of words other than READY at the end of the header line means the server is unavailable, and can be logged or displayed as an error message. The present server reports BUSY if it is over its connection count limit.

The client sends commands and the server replies. The protocol is used for client-to-server and server-to-server communication. In the server-to-server case, the server originating the session is acting as a client and will be referred to as the client.

Command STARTTLS: sets up TLS encryption.

```
C> STARTTLS
S> PROCEED
<TLS handshake takes place>
S> ENCRYPTED
```

The server should send PROCEED before calling server-side startTLS(). The server should send ENCRYPTED after startTLS() returns. The client should wait for PROCEED and then call client-side startTLS(). The client should wait for ENCRYPTED before sending another command.

The present client sends STARTTLS immediately after receiving the READY header. Clients should always use TLS, and LOGIN should never be sent before setting up TLS. The present client records the server's X.509 certificate, and logs a system message if it changes, indicating a possible man-in-the-middle attack. The present server also logs certificate changes when acting as a client.

Future clients may give additional trust to GPG keys received from a server where the server certificate matches the email domain of the key, and where the server certificate has a valid chain to a public certificate authority.

The protocol uses TLS 1.2 with ephemeral keys. This means the traffic encryption keys are obtained by Diffie-Hellman exchange rather than the server's private key, so obtaining the private key does not expose previous traffic. Clients and servers should not support or accept older versions of TLS. Since this is a new protocol, there is no reason to support obsolete and broken security standards.

Command QUIT: ends the connection.

```
C> QUIT
S> GOODBYE
<disconnect>
```

Command LOGIN: authenticates with the server.
LOGIN userid authkey
where userid is normally the 40-character keyid of the user, and authkey is a string specific to that user. Both userid and authkey are 8 to 40 characters, 0-9 and A-Z only. The authkey needs to be unique to each user, and is assigned by the server administrator.

In addition to the normal keyids, there are two special userids, "administrator" and "replication", which have special powers. The "replication" user can use the REPLICATE and REPLOGIN commands. The "administrator" can perform functions such as adding and deleting users, and shutting down the server.

Users must be logged in to post key-announcement and address-claim records, and the userid must match the keyid. Users must also be logged in to retrieve data, message-announcement, and acknowledgment records. This prevents abuse of the server as a file-sharing system.

This is not a high-security authentication system. It is intended to prevent abuse of the server, not to protect user privacy. The login is done after the connection is TLS encrypted, so a passive snooper should not be able to get the user's authkey.

```
C> LOGIN C3202B03EB24B1A156DD2F2BB4F5C41CE3E5967D E00D395B35C6A2F40AE545AD40B000B831147AAF
S> DONE
```

or

```
C> LOGIN C3202B03EB24B1A156DD2F2BB4F5C41CE3E5967D E00D395B35C6A2F40AE545AD40B000B831147AAF
S> FAILED
```

Command GET SERVER fetches a block from the server.

```
C> GET SERVER E00D395B35C6A2F40AE545AD40B000B831147AAF
S> NOT FOUND

C> GET SERVER E00D395B35C6A2F40AE545AD40B000B831147AAF SINCE 2014-10-31T09:02:22Z
S> NOT FOUND

C> GET SERVER DA458F9588E1B4F9F85EE79E8DEB4DD85B448092
S> Type: acknowledgment
S> Date: 2014-10-31T09:02:22Z
S> Version: 1
S> ProofOfWork: bd,00000014f,0000001f5
S> Hash: 15f8b83f6380c2a25a0249cc615ec7e58a9dcf71
S> EndBlock

C> GET SERVER 02A424F444D12C56916A24CD65907E11AB51CE53
S> Type: data
S> Date: 2014-11-08T05:25:22Z
S> Version: 1
S> ProofOfWork: bd,000000027,000000035
S> Data: 1535
S> <1535 binary bytes after CR,LF>
```

GET SERVER fetches a block from the server's local storage. The user must be logged in to fetch data, message-announcement, and acknowledgment blocks. Anyone can fetch key-announcement and address-claim blocks. The client sends the request followed by CR,LF. The server returns either NOT

FOUND, or the block.

All block types except data are returned as a series of text lines terminated by EndBlock on a line of its own. EndBlock is not actually part of the block; it is just an end marker.

Data blocks are returned as a series of lines, followed by the Data: line specifying the number of bytes, CR, LF, and then the binary data. There is no end marker; the client must count bytes.

The SINCE keyword affects only message-announcements, and fetches only those announcements with Date fields equal to or later than the specified date. This saves bandwidth in checking for new mail.

Command STORE SERVER posts a block to the server.

```
C> STORE SERVER DA458F9588E1B4F9F85EE79E8DEB4DD85B448092
C> Type: acknowledgment
C> Date: 2014-10-31T09:02:22Z
C> Version: 1
C> ProofOfWork: bd,00000014f,0000001f5
C> Hash: 15f8b83f6380c2a25a0249cc615ec7e58a9dcf71
C> EndBlock
S> DONE

C> STORE SERVER 02A424F444D12C56916A24CD65907E11AB51CE53
C> Type: data
C> Date: 2014-11-08T05:25:22Z
C> Version: 1
C> ProofOfWork: bd,000000027,000000035
C> Data: 1535
C> <1535 binary bytes after CR,LF>
S> DONE
```

STORE SERVER writes a data block to the server's local storage. The client sends the command followed by the block. Blocks are terminated by EndBlock or, in the case of a data block, by the appropriate number of bytes. The server responds with DONE when it has finished saving the block and the client can send another command.

Some operations such as checking key-announcements and merging message-announcements can take a while to complete. Therefore, the server writes incoming blocks to temporary storage, returns DONE, and processes the incoming block in a background thread. Receipt of DONE does not mean the block has passed verification. It could still be rejected if it is invalid or the client is not logged in.

The user must be logged in as the appropriate userid to post key-announcement and message-announcement blocks. The other block types can be posted by anyone.

Command REPLICATE is a variant of STORE SERVER for server-to-server replication:

```
C> LOGIN replication repl-authkey
S> DONE
...
C> REPLICATE DA458F9588E1B4F9F85EE79E8DEB4DD85B448092 none
C> Type: acknowledgment
C> Date: 2014-10-31T09:02:22Z
C> Version: 1
C> ProofOfWork: bd,00000014f,0000001f5
C> Hash: 15f8b83f6380c2a25a0249cc615ec7e58a9dcf71
C> EndBlock
S> DONE
...
C> REPLICATE CEC2D4AC1EFAD403ADD1453034EA9A231277195A C3202B03EB24B1A156DD2F2BB4F5C41CE3E5967D
C> -----BEGIN PGP SIGNED MESSAGE-----
C> Hash: SHA512
C>
C> Type: address-claim
C> Version: 1
C> ...
C> EndBlock
S> DONE

REPLICATE hashkey sender-userid
```

The REPLICATE command is a modified STORE SERVER for server-to-server replication. The sender must specify the userid of the posting user. Unlike STORE SERVER, the block will not be queued for replication to the partner. The sender must be logged in as the replication user.

When a server configured for replication receives a STORE SERVER, it queues the block for replication to the partner as well as storing it locally. The block is then sent to the partner using the REPLICATE command, specifying the userid of the user doing the STORE SERVER, or None if the sender was not logged in. This allows the partner to evaluate the block using the same logic as the original receiving server.

If one server is down, blocks will remain queued on the live server until they can be replicated. In this way, brief downtime of either server does not affect the user experience, and a hard failure of one server results in relatively little loss of data. The replication is peer-to-peer; there is no master server. A future server might implement hard real time replication (sending to the partner before DONE is returned to the client) to guarantee no loss of data.

Command STORE PROXY allows a client to request a server to forward blocks for it. The client can send outgoing mail to his own server and have his server forward it to the recipient's server. A message can be sent to the server once, and the server handles the work of posting it to multiple recipient servers without tying up the client machine. This also permits users without TOR/I2P clients to send messages to servers that are only accessible behind TOR/I2P. The client must be logged in to use STORE PROXY.

```
C> STORE PROXY 02A424F444D12C56916A24CD65907E11AB51CE53
C> Post-To: server=recipient1-kms.com:8081,recipient1-kms.com:8082
C> Post-To: server=recipient2-kms.com:8081,recipient2-kms.com:8082
C> Type: data
C> Date: 2014-11-08T05:25:22Z
C> Version: 1
C> ProofOfWork: bd,000000027,000000035
C> Data: 1535
C> <1535 binary bytes after CR,LF>
S> DONE
```

```
C> STORE PROXY 63840CD939FE26C996119587BDE58434E5DDF6B4 AFTER 2014-11-08T10:25:22Z
C> Post-To: server=recipient-kms.com:8081,recipient-kms.com:8082
C> Type: message-announcement
C> Date: 2014-11-08T05:25:22Z
C> Version: 1
C> Recipient: c3202b03eb24b1a156dd2f2bb4f5c41ce3e5967d
C> Mailbox: 1
C> ProofOfWork: bd,000000643,000001c77
C> DataBlock: 02a424f444d12c56916a24cd65907e11ab51ce53
C> MessageHash: 02a424f444d12c56916a24cd65907e11ab51ce53
C> EndBlock
S> DONE
```

The Post-To: header specifies the recipients. Multiple servers on one line are replication peers; only one will receive the block. To send to multiple recipient servers, use multiple Post-To lines.

The optional AFTER time delays delivery until a specified time in the future. This might improve traffic analysis resistance for an anonymous user. It can also be used whenever a sender wants a message to be delivered after a particular time. The delay should be limited to three days, to avoid filling up the server's outgoing queue. Post-To: entangled is also allowed to embargo Entangled messages.

Commands STORE ENTANGLED and GET ENTANGLED:

These commands write to, and read from, the shared Kademlia distributed hash table. The syntax is the same as STORE SERVER and GET SERVER. The present server allows all users to post to and read from Entangled. Entangled is freely accessible via UDP, so there is no reason to restrict access via TCP. This permits anonymous users behind TOR to use Entangled freely.

The Entangled storage pool is effectively a global hash table shared among all servers that participate. It is intended mostly for key distribution, but users can post and receive short messages through Entangled. The proof of work requirement is intentionally set high to discourage abuse.

Command DNS TXT:

Clients need to do DNS TXT record lookups to find keys. If a client is anonymous behind TOR/I2P, and does TXT lookups directly, he compromises his anonymity. To prevent this, the server has the ability to do TXT lookups on behalf of the client. The server returns each TXT record on a separate line prefixed by TXT:<space>, or NOT FOUND if no record exists. The present server permits anyone to perform TXT lookups.

```
C> DNS TXT testtxt.testdomain.org
S> NOT FOUND
C> DNS TXT txttest.testdomain.org
S> TXT: server=kmsvr1.testdomain.org:8081,kmsvr2.testdomain.org:8081
S> EndBlock
```

Command SHUTDOWN:

Causes the server process to exit. Requires administrator login.

```
C> LOGIN administrator password
S> DONE
C> SHUTDOWN
S> GOODBYE
<disconnect>
```

Commands ADDLOGIN, RMLOGIN, GENLOGIN, REPLOGIN:

These commands create and delete user accounts. All except REPLOGIN require administrator login. REPLOGIN requires replication login.

ADDLOGIN userid authkey
ADDLOGIN authkey
RMLOGIN authkey
REPLOGIN userid authkey
GENLOGIN number-of-accounts

ADDLOGIN creates an account on the server. Userid is set to undefined if not specified. Accounts are replicated to the replication partner using REPLOGIN. REPLOGIN is equivalent to ADDLOGIN except the change is not replicated to the partner, and login must be replication instead of administrator.

RMLOGIN removes an account and replicates the deletion to the partner. GENLOGIN followed by an integer creates several accounts with random authkeys, and returns the authkeys. The accounts are replicated to the partner.

```
C> LOGIN administrator password
S> DONE
C> ADDLOGIN FC4F5D43A561BC05DAE16A0131FF192F6600223D
S> DONE
C> RMLOGIN FC4F5D43A561BC05DAE16A0131FF192F6600223D
S> DONE
C> GENLOGIN 2
S> 365673C54165B9B1EDA097E40BD83C7B532C0277
S> 329ACA9781A38E8FFB153864271406F2DDB37BB3
S> EndBlock
```

Key lookup operation

There are three ways to look up a key: specific server, Entangled, and DNS.

For the specific server case, the user (or existing knowledge such as a message header) has provided a server hostname and port from which the key can be fetched.

For the Entangled case, the client looks up the key in the shared Entangled DHT.

For the DNS case, the client prepends cmsvr. to the domain portion of the email address (everything after @), looks up the result as a TXT record in DNS, and obtains a server string giving the appropriate servers for that email address.

If the keyid hash is known, the client can directly look up the key-announcement using the hash value. If only the email address is known, the email address is converted to lower-case UTF-8 and hashed. The client looks up that SHA1 value and obtains an address-claim block, which provides the hashes of all the keys that claim that email address. The user must select the appropriate key. Without some central authority, there is no mechanism to determine the rightful owner of an email address.

In the future, the client may give preference to keys fetched from a server which is specified in DNS as the owner of the domain (TXT record), and which have a server certificate, with a valid chain, matching the server's hostname. Businesses using Confidant Mail could buy commercial certificates for their mail servers, and this would provide good-enough key validation for most purposes.

Message structure

A message, as seen on the server, consists of one or more data blocks, pointed to by one message-announcement per recipient. The client fetches the data blocks in the order specified in the announcement, and concatenates the binary data. The client should check the MessageHash against the concatenated data.

The concatenated message is a GPG ciphertext encrypted to all the message recipients. The client submits the message to GPG for decryption using the user's private key, obtaining a plaintext block.

One design goal of Confidant Mail is to be able to forward a message with its signature intact so a third party can check the signature of the original sender. Therefore, the message uses the GPG detached signature format. The plaintext block begins with a two-byte binary signature length (most significant byte first), then the detached signature, and then the message itself. The client separates the plaintext block into two files ending in .DTS and .ZIP respectively, then calls GPG to check the signature, and saves the signature status into a file ending in .SIG.

If the client gets a message from an unknown sender, it has to stop and fetch the sender's key before attempting to check the signature. The HEADER.TXT provides information which tells the client where to get the key from.

The message is a ZIP file, containing some predefined files as well as any attachments. File names within the ZIP file are:

HEADER.TXT - the message headers, the contents of which are explained below.
BODY.TXT - the message body in plain text format.
BODY.HTML - the message body in HTML format.
BODY.XML - the messge body in the XML format used by the wx.RichTextCtrl class.
ACK_<keyid>.BIN - acknowledgment hash for a particular recipient, which may or may not be encrypted. There will be one per recipient. If encrypted, it will end in .PGP rather than .BIN
<messageid>.DTS and <messageid>.ZIP - the signature and contents of the original message, if this is a forwarded message. Only one DTS/ZIP pair will be present.
Attachments, if any, will have their filenames preceded by an underscore (_). The underscore should be removed from the filename when saving the attachment.

The body is provided in three formats, because at the moment the RichText class can emit HTML but not parse HTML. I would like to get rid of the XML and change to a pure HTML format in the future. This is not possible with the current RichText class, so HTML is included for forward compatibility.

The XML is displayed by default. The user can view the text-only format in case he is worried about an exploit in the XML. If the client gets HTML but no XML, it will display the HTML. If the client gets only text, it will display the text.

If the message is addressed to one recipient, the ACK_<keyid>.BIN file is the 20-byte acknowledgment value. If the message is addressed to more than one recipient, each ACK file is a GPG encrypted message, encrypted to that particular user only, containing the 20-byte acknowledgment value. This prevents one user from forging an acknowledgment from another user. [v0.31] this file can also contain the bypass secret; see the description above under BypassToken.

Example HEADER.TXT files:

```
From: Amy Alien <alien@pobox.com> 62209a154b731d747be15e545e9b328e6ce255b3
To: Amy Alien <alien@pobox.com> 62209a154b731d747be15e545e9b328e6ce255b3
KeyTransport-62209a154b731d747be15e545e9b328e6ce255b3: server=localhost:8082
Ack-62209a154b731d747be15e545e9b328e6ce255b3: 5ba98e9a8f46bc903b1bac94ca93ac937b2dce8c
To: William Wreck <wreck@pobox.com> c3202b03eb24b1a156dd2f2bb4f5c41ce3e5967d
KeyTransport-c3202b03eb24b1a156dd2f2bb4f5c41ce3e5967d: server=192.168.4.102:8081
Ack-c3202b03eb24b1a156dd2f2bb4f5c41ce3e5967d: 3ecd496f6b893fae5583e07861e6b55ef192437c
MessageUniqueId: 031cf398b44be44b6d5dc08d22ddb35e0b51a0ba
ReplyThreadId: c0ca9efd86d54a8a22ca47e40a5fd6182e891c18
Subject: RE: test unique id
Date: 2014-11-20T06:25:23Z

From: William Wreck <wreck@pobox.com> c3202b03eb24b1a156dd2f2bb4f5c41ce3e5967d
To: Amy Alien <alien@pobox.com> 62209a154b731d747be15e545e9b328e6ce255b3
KeyTransport-62209a154b731d747be15e545e9b328e6ce255b3: server=localhost:8082
Ack-62209a154b731d747be15e545e9b328e6ce255b3: b11e5f837f229f32d8f21d1bc9b257bd6e5152e5
MessageUniqueId: 128293e5b6d2290d7201f1521b424436c57bdf9d
ForwardedMessageId: 749b112cdc5dc9af9e481387900d6687991373ca
Subject: FW: RE: test unique id
Date: 2014-11-20T06:29:42Z
KeyTransport-c3202b03eb24b1a156dd2f2bb4f5c41ce3e5967d: server=localhost:8081
```

Lines are as follows:

From: email address and keyid of the sender.
To: email address and keyid of each To recipient.
Cc: email address and keyid of each Cc recipient.
Bcc: email address and keyid of each Bcc recipient (not shown in header/not replied to)
Key-Transport-<keyid>: provides the sender's server path (or "entangled") for each keyid mentioned. The client can try to fetch from this server as well as from Entangled or DNS.
MessageUniqueId: an identifier made up by the sender, which is unique to this message.
ReplyThreadId: if this message is a reply, the MessageUniqueId of the original message it is a reply to. This allows for threaded display in the future.
ForwardedMessageId: if this message is a forward with original, the filename of the ZIP and DTS files included in this message ZIP. This does not correspond to any MessageUniqueId.
Subject: the subject line of the message.
Date: the date and time the message was sent.
Ack-<keyid>: the acknowledgment ID of the message for this particular recipient. This is the public ID. The 20-byte value in the ACK file is the precursor: hashing the private value in the file with SHA1 should give this value.

Message posting operation

To send a message to a user, assuming you have the user's key, the sender uses the Transport header in the key-announcement block to determine the destination. A message consists of one or more data blocks, and one message-announcement per user. The message-announcement must be posted to one of the recipient hashes for the recipient, obtained by prepending a Mailbox value to the user's keyid and hashing. If a message has multiple recipients on one server, the same data blocks are used for all recipients.

Message fetching operation

To check email, the client connects to his server, logs in, and fetches each of his recipient hashes. The SINCE operator can be used to limit the length of the reply. The client then checks to see if any messages are announced which he does not have, and if so, fetches the corresponding data blocks and attempts to reassemble and decrypt the message. If all the data blocks are not present, the client keeps the ones received and continues to look for the others in subsequent fetching operations.

Acknowledgment operation

After retrieving and decrypting a message, the recipient posts an acknowldgment to sender's transport. The recipient takes the ACK file for his userid, decrypts it if necessary, and composes and posts an Acknowledgment block on the sender's server (or to Entangled if the sender's transport so indicates.)

The sender periodically checks for Acknowledgments for all the messages in his Ack Pending category, and when all the Acknowledgments have come in, removes the message from that category.

Key posting operation

Each user periodically posts his key to his server (if any) and to Entangled. He can also post to an old server (if in the process of moving servers) or to a "Pub Server" if he has moved his server away from the server that his domain's DNS points to.

Anonymity models

Confidant Mail supports anonymous users via the TOR and I2P anonymous networks. Both client-to-server and server-to-server connections can run through TOR and I2P, allowing for several anonymity models. Public and anonymous users can exchange messages with each other.

TOR supports both hidden services and exit node connections to public services, while I2P supports only hidden services. Use of exit nodes should be avoided where possible, because they can and sometimes do tamper with traffic. If an exit node performs a man-in-the-middle attack, this will produce a "certificate has changed" log entry in the client's System Messages or sending server's log. Such an attack should not compromise message content, but it would allow for traffic analysis.

If the server provides both a public port and a TOR/I2P port, anonymous users connect via the hidden service and public users connect via the public port, and they can exchange emails even if the public user does not have a TOR/I2P client.

If the server provides only a public port, anonymous users can either send to it via a TOR exit node, or send via proxy, assuming the anonymous user's home server has TOR/I2P capability.

If the server provides only a TOR/I2P port (covert server), then public users can send to it only by installing TOR/I2P clients, or by proxy if the public user's home server has TOR/I2P capability.

Commercial and public-use servers should offer public, TOR, and I2P ports. Some people running home servers will probably not bother to set up TOR/I2P. Clandestine organizations may run covert servers. Since a covert server does not participate in the Entangled DHT, the only way to fetch keys for its users is to manually enter the server's hidden-service name and port into the address lookup. This can be a feature rather than a bug for organizations which do not want to advertise their existence.

The Confidant Mail client and server originate TOR/I2P connections using SOCKS protocol. The server receives TOR/I2P connections on its TCP port from TOR and I2P clients which must be installed on the server machine, or on another machine on the server's internal network. The user must install TOR and I2P client software, and in the case of the server, configure the hidden service forwarding.