Freedom Network 1.0 Architecture and Protocols

Ian Goldberg, Adam Shostack
Zero-Knowledge Systems, Inc.
{ian,adam}@zeroknowledge.com

10th October 2001

Abstract

This white paper, targeted at cryptographers and security experts, offers a detailed look at the entities, protocols, and systems that make up the Freedom Network. In addition, it adds a great deal of mathematical and implementation detail, with the intent of allowing and encouraging external cryptanalysis of the system.

1 Introduction

1.1 This paper

The Freedom product line is designed to be the most integrated, strongest and easiest-to-use privacy system available. This white paper gives the technical reader a deep understanding of each component, and of the system as a whole. This paper can be read on its own, or in conjunction with "Freedom 1.0 Security Issues and Analysis".

This paper is intended to explain exactly how the Freedom Network works, and to contain sufficient information to construct a Freedom compatible client or server. This paper exists in two versions, one with the protocol details and the math ("Freedom Network 1.0 Architecture and Protocols"), and one without ("Freedom Network 1.0 Architecture"). We have published the paper in this way to better serve the intended readers of each version. This is the version with the protocol details.

We start by introducing the entities that make up the Freedom network. We then explain the databases that the various entities in the network use, and then how those entities communicate, starting with AIP to AIP communication, and building from there to client-AIP communication, the telescope encryption protocols, and the way IP, UDP and TCP are handled as they traverse the Internet. We conclude by examining the application layer handling for the protocols that Freedom supports.

This paper concentrates on the protocols as they exist today. There are a number of known issues which we will be addressing over time. Those issues are not always noted here. The current version of "Freedom 1.0 Security Issues and Analysis" will always list those issues that are known to exist from a security or privacy standpoint.

1.2 Freedom overview

The Freedom network is an overlay network which runs on top of the Internet. It uses layers of encryption to allow a Freedom end-user to engage in a wide variety of pseudonymous activity by hiding the user's real IP address, email address, and other identifying information from counter-parties, eavesdroppers, and active attempts to violate the user's privacy.

Users are encouraged to create psuedonyms for each area in which they want to preserve privacy. The nyms that someone uses cannot be tied together. Thus, it is not possible to say if `superman@freedom.net` and `clarkkent@freedom.net` are the same, or different people. Superman is happy with this situation because he doesn't want his supervillian enemies to know about his life. Similarly, when `jobseeker@freedom.net` browses a resume web site, his employer can't see that Clark isn't happy with the working conditions at the Daily Planet, and wants to jump into another line of work.

Freedom protects Clark's privacy by proxying the various supported protocols, and sending those proxied packets through a private network before they are deposited on the Internet for normal service. That private network, as a system, is operated by Zero Knowledge. Individual nodes in the network are operated by Zero Knowledge and our partners, so that no single operator has comprehensive knowledge of what data is flowing through the network.

Thus, the main components of the system are pseudonyms (or nyms) and Freedom Servers. In the next section, we offer precise definitions of these and other entities.

2 Entities

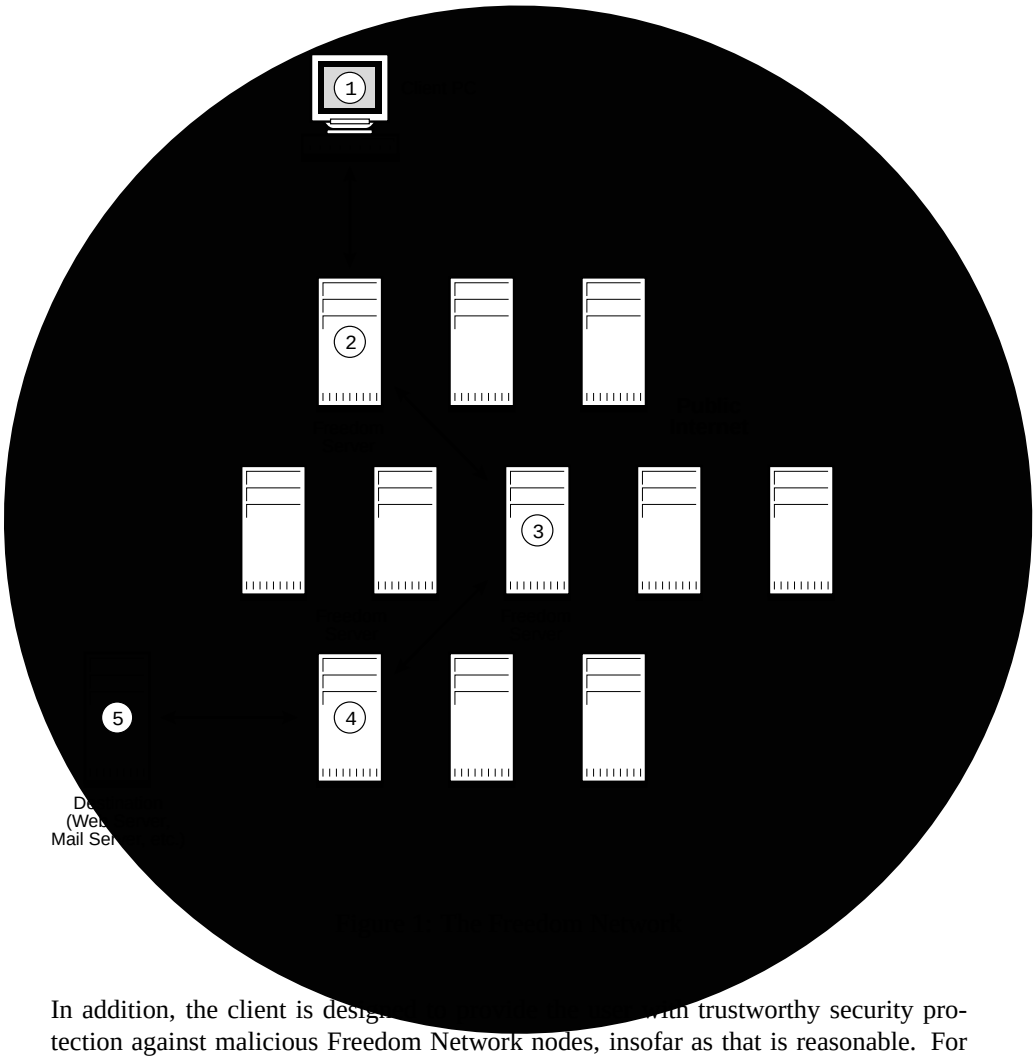
2.1 Nyms

Nyms are the identities that Freedom users assume on the Internet. A nym is defined by a unique email address at Freedom.net, and the associated digital signature key. Zero Knowledge certifies only the uniqueness of the email address. A nym has an email address, a signing key, an encryption key, and zero or more reply blocks. Reply blocks are defined in more depth in section 10, below. A nym is created when a user sends a nym creation token, a signature verification key and an encryption public key to the nym server. This process is detailed in the "Untraceable Nym Creation on the Freedom Network" white paper.

Nym signature keys are 1024 bit DSA keys. They are signed by the nym server signature key (see section 2.5, below). Nym encryption keys are 1024 bit El Gamal keys, which have been signed by the nym's signature key. (Unless otherwise noted, all signature keys are 1024 bit DSA, and all encryption keys are 1024 bit ElGamal.

2.2 Client

The client is a software package, currently provided by Zero Knowledge, which implements a variety of security activities on behalf of the user. It has, hardwired into it, a Master cryptographic key which is used to authenticate all components of the system.



In addition, the client is desig[illegible] with trustworthy security protection against malicious Freedom Network nodes, insofar as that is reasonable. For example, it is not reasonable to expect that the client can offer protection against a fully compromised network. However, much network information is delivered to the client so it can make decisions about which nodes in the network to use, rather than trusting the network to decide which nodes constitute an appropriate path. This decision is driven by a possible tension between definitions of "appropriate" used by the end user and the network operators.

The client has no private keys separate from those of the nyms which use it. It ships with the public parameters of the Freedom Master key hardcoded into it, as well as a pre-loaded cache of server public keys.

2.3 AIP

The Anonymous Internet Proxies (AIPs) are the core network privacy daemons that make up the Freedom network. They pass encapsulated network packets between them-

selves until they reach an exit node. The exit node has a "wormhole" which acts as a proxy, allowing packets to pass between the Internet and the Freedom Network.

The wormhole acts much like a traditional network address translator, with additional proxy functionality to only allow well-formed packets to acceptable server ports.

An AIP has a signature key which is certified by the Monthly key after an out-of-band confirmation process with the server operator. It also has an encryption key, which is signed by its signature key.

2.4 MAIP and FMG

The Mail AIP (MAIP) is a mail transport daemon that works with reply blocks to move mail through the Freedom Network. It uses the Anonymous Mail Transfer Protocol to move messages. AMTP may be described in a forthcoming white paper, or it may be replaced, and the replacement published.

When the FMG-MAIP receives a message that is destined for a non-Freedom address, it applies some security and spam-prevention techniques, such as ensuring that the message is properly signed by the nym, and that the nym is not trying to send out too many messages at once. It then formats the message as a MIME-encapsulated email, and sends it to the destination address.

Incoming mail from the Internet, destined for a nym, is received by a Freedom Mail Gateway (FMG-MAIP), which sits behind an SMTP server. The message is encrypted and chained using each of a nym's reply blocks (more than one reply block may be used to avoid reliability problems should a Freedom server crash). This means that multiple encrypted copies of a message may be delivered to a user's (real) mailbox, but the client software hides this fact from the user, and automatically deletes duplicates.

2.5 Databases

There are a number of support databases which help make the Freedom Network a complete operational system. These databases, collectively referred to as the core services, offer network status and crypto keys and certificates.

Network Information Query and Status Servers (NIQS, NISS) These servers hold the network topology, status, ratings information and some operator data. The NIQS is a read-only server which serves up the digested information. The NISS receives the status packets from the entities on the network and stores everything in the Network Information Database (NIDB). The information that is collected is described in section 11.

Nym Server This server holds all of the nym information and keeps track of all spent tokens. The Nym server keeps a copy of the current nym keys and submits these to the key update server. It also stores a hash of each spent token, in order to prevent token double-spending.

Key Update / Key query servers Key update server receives updates from the nym server. Key query server serves up all public keys on the network. Key update is write only, key query is read only.

Token Server This server keeps the list of active (unredeemed) serial numbers, and generates tokens in exchange for them.

FMG-MAIP The FMG-MAIP has two databases, FMG-Stat and Nym-block. FMG-Stat stores recipient count information which is used for spam control. The nym-block database contains requests that have come in to not allow a nym to send messages to a specific address or domain.

2.6 Master Signature Key hierarchy

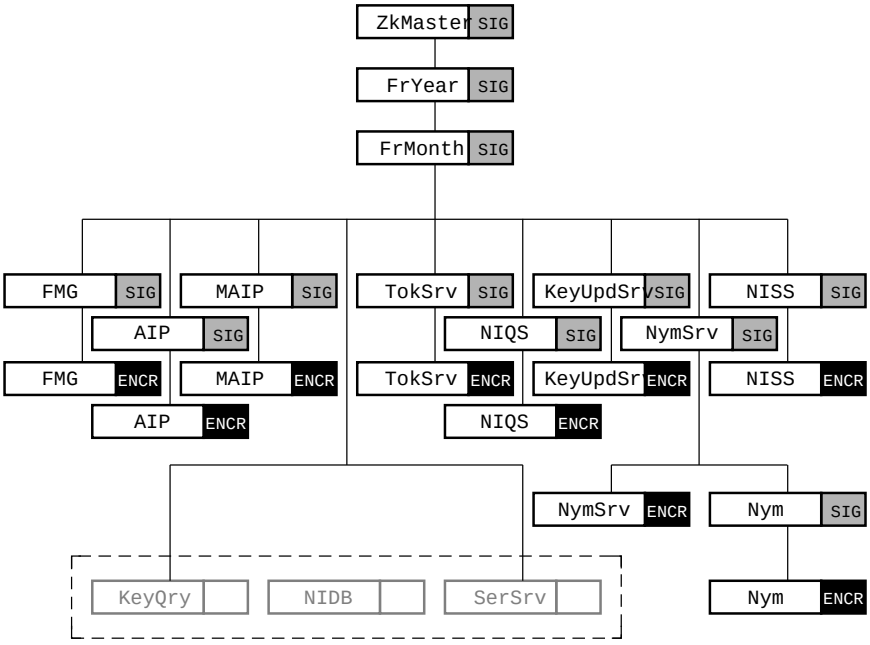


Figure 2: The key hierarchy

There is a set of keys at the top of the hierarchy which are used only for signing. These keys include the Master, Yearly, and Monthly keys. The Monthly key tends to sign a lot of other keys, including signature keys for AIPs, MAIPs, and the FMG.

There is a Master key, whose public parameters are encoded into all Freedom software for reference. It is a 2048 bit DSA key, using the same prime parameters as are used in PGP.

The Master Key signs a Yearly key. [1] The Yearly key is 1024 bit DSA; it is used to sign the Monthly key and client software updates. The Monthly key is used to sign a signature key for each of the FMG, NymServ, TokSrv, KeyUpdSrv, NIQS, and NISS. It is also used to sign the AIP and MAIP signature key for each AIP and MAIP. Each of

[1] The original intent was to rotate all the keys, except for the master, on a reasonable time scale. That code has not been extensively tested, and as such, we may not invoke it in this version of the fielded system.

these keys signs an encryption key for the entity. The NymSrv key also signs the Nym signature keys. The Monthly key, and those keys which it signs, are stored online.

The master key DSA parameters are:

```
p =
    f64257b7087f081772a2bad6a942f305
    e8f95311394fb6f16eb94b3820da01a7
    56a314e98f4055f3d007c6cb43a994ad
    f74c648649f80c83bd65e917d4a1d350
    f8f5595fdc76524f3d3d8ddbce99e157
    9259cdfdb8ae744fc5fc76bc83c54730
    61ce7cc966ff15f9bbfd915ec701aad3
    5b9e8da0a5723ad41af0bf4600582be5
    f488fd584e49dbcd20b49de49107366b
    336c380d451d0f7c88b31c7c5b2d8ef6
    f3c923c043f0a55b188d8ebb558cb85d
    38d334fd7c175743a31d186cde33212c
    b52aff3ce1b1294018118d7c84a70a72
    d686c40319c807297aca950cd9969fab
    d00a51b84fcf7ec96a96fa9e4edc9f93
    3721131de3dd3de07d1d6bce098311b5

q =
    01fe46d3083d66a45d419f9c7cbd894b
    221926baaba25ec355e927c2cf

g =
    6bb9042546a5a9c771e08d6c71ef2bd5
    c6b5b006cd5d56b54483192b44de36ab
    aedfc1924676ec21709685f9615152ad
    07dbed807510bc45911535f91d0f632b
    7efdffaea78588a1ad0f94191486f210
    5ad53e15bf88a5dfe90c32999aed6274
    42071a67498635fa17cc7303da44de39
    5e95bfefa47d801689bc716cc186d8db
    8a5580780a2756918a91ec7c84306450
    098dad020de0118bdd702065f5094a74
    a53b1495b5b3df19cc0902bd705beb59
    00b88a0067eead9f98a38519524555d3
    0de83a1efd1b820d226a48d67e5972bf
    c67d147108eb457f9cdd6bd054fe6de2
    4927d64f6c1117df90944e79cbecf237
    044d169fc4187cbdca5d29beedbbafdd

y =
    8874fff053627d67a6928f310cd95233
```

```
6ee01b967485c721a442458c83fb68c7
3fbc274e8fc6886e4a0dcb43aa1753d3
c743b2e8927e61ca4a40e6df980232fa
b7ba8ec96dc29591b47638f10db8c1ba
2dc9700f4c28d8df0f88729a34a6782e
88dbf6176de88d3b08b5356b34fa283e
11bb1c9e4da97f13bf579e9943c491f0
b3ee9e551ccccb8fe0e5efbcf8f8b9c6
5398d62ab1206575af4a824c0656c84c
1ab25895fa159ad5cd9966d86aee036a
9150b91a4af3f54dc9be8df152b73466
634bc674f48f7d9a36920870d7ff031a
ec873342437b106e0a984035cf879587
c60a173244c0e535efd41dd14eaaa8ed
8d623e88023d3be1a2efb59d0c4edfc4
```

These parameters have the properties that p and q are prime, q divides $p-1$, and g is an element of Z_p^* of order q.

2.7 Procedures for generating and using important keys

The Yearly and Master private keys are stored on a non-networked computer in a physically secure location. There is two-person access control for the locks controlling access to the machine, the passwords needed to log in, and the passphrases for the keys. The process for using them has strong procedural and audit controls.

2.8 Secret-sharing

The Master private key is backed up using secret sharing code by Hal Finney and updated by Ian Goldberg. The shares are 3DES encrypted and stored in the care of certain managers of the company.

3 Database queries

The databases listed in section 2.5 form the core services of the Freedom Network. Various entities in the network query these databases at various times:

NIQS: The NIQS is queried by clients, AIPs and MAIPs in order to determine the current network topology. The clients use the information in order to create routes through the network (as described in 6.1, below); AIPs use the information to determine to which other AIPs they should establish secure links.

Key query server: A client will obtain public keys of other nyms from the key query server in the event that it wishes to send encrypted email to them. In order to protect the identity of the client, the request is made over the Freedom network.

In the event that the NIQS reports a new AIP being available, the client will query the Key Query server in the clear. Most servers, including at least AIPs, MAIPs, and FMG-MAIP have reasons to query this server, including authorization and authentication.

Nym server: The Nym Server is queried by the FMG and FMG-MAIP in order to get reply-blocks, check authorizations, and perform spam control.

token server: The Token Server is only queried during the nym generation process.

FMG database: The FMG database is queried by the FMG, to keep track of outgoing mail quotas, as described in section 9.2, and to apply blocking rules.

Queries to the databases are not encrypted or signed by the requestor, and generally, the responses are not encrypted or signed by the database. The NIQS and Nym Server sign their data, but that does not include negative responses, which are generated on the fly. However, the *contents* of the responses have enough information to ensure their accuracy (for example, the key query server will return signed certificates).

4 Inter-AIP link encryption

AIPs on the Freedom network talk to each other with secure links. These links are set up between specific pairs of AIPs that are chosen to minimize network latency and delay; we do not create inter-AIP links that allow you to traverse three continents, as the latency of such a link would make it unusable.

Each AIP has a number of "neighbours"; i.e. the other AIPs to which it has a secure link. As mentioned above, it is not the case that every AIP is a neighbour of every other AIP.

There is no automatic configuration of the graph of which AIPs are neighbours of which other AIPs (the "AIP graph"). The secure links are configured manually with a tool, via the NIQS. An AIP will periodically query the NIQS to get a list of the neighbours it is supposed to have, and will bring its secure links up and down accordingly.

4.1 Setup: Authenticated D-H key agreement

When two AIPs are configured to talk to each other, both will query the key server to get the key for the other AIP. Call the AIPs Alice and Bob. Alice gets Bob's public key certificate from the key query server, and validates it by climbing the key hierarchy until reaching a key that she trusts. Bob mirrors this activity to ensure the exchange is mutually authenticated. Alice and Bob then undergo an authenticated Diffie-Hellman key agreement protocol to derive the encryption key to be used on the secure link. This exchange is done once per hour; old link encryption keys are discarded to ensure perfect forward secrecy; that is, if an adversary records the encrypted traffic, and wants to force Alice or Bob to decrypt it for him, he only has until the hour is up. After Alice and Bob forget the link encryption key, there is no way to recover that traffic.

4.1.1 Protocol information

The side initiating the link establishes a TCP connection to the other host, on its Diffie-Hellman port, which is usually TCP port 51102.

Each side sends a packet of the following format to the other (in this, and all protocol information descriptions, multibyte values are sent in network (i.e. big-endian) byteorder):

```
uint8_t vers;
uint8_t type;
uint16_t pktlen;
uint8_t symAlg;
uint8_t lekv;
uint16_t port;
int64_t absTime;
byte halfsec[256];
FrEntIdExp ent;
fr_sig_pk sig;
```

vers (1 byte): The constant DH_VERSION_VALUE = 1

type (1 byte): The constant FRPKTTYPE_DIFHEL = 6

pktlen (2 bytes): The total length of the packet = 402

symAlg (1 byte): The symmetric algorithm to use for encryption. Currently always set to FRCRYPT_SYMALG_BLOWFISH = 3, indicating 128-bit Blowfish. Other valid values are FRCRYPT_SYMALG_DES3 = 4 and FRCRYPT_SYMALG_DESX = 5 indicating 3-key 3DES, and DESX, respectively.

lekv (1 byte): The link encryption key version. Alternates between 0 and 1 for successive D-H agreements on the same link (so that packets encrypted with the old key can arrive on the network while a new key is being negotiated; the key is discarded when the second subsequent key nogotiation is performed).

port (2 bytes): The local port number for the AIP (the port number from which packets from this AIP will be sent). Used to distinguish multiple AIPs running on the same IP address.

absTime (8 bytes): The current time of the AIP, in seconds since the UNIX epoch. (The first 4 bytes of this will be 0 for a while...)

halfsec (256 bytes): The 2048-bit value of $g^h \bmod p$. g and p are given below. h is a random number, currently 256 bits in size.

ent (65 bytes): The entity signing the packet, in this case, an AIP. This field has the following subfields:

ent.type (1 byte): The entity type of the AIP signing the packet, which will be FRENT_TYPE_AIP = 1.

ent.name (64 bytes): The entity name of the AIP signing the packet.

sig (65 bytes): The DSA signature of the previous data (all bytes in the packet except for the sig element itself). This field has the following subfields:

sig.r (29 bytes): The DSA r value of the signature.

sig.s (29 bytes): The DSA s value of the signature.

sig.keyVersion (1 byte): The version number of the key used to sign the message.

sig.keySize (2 bytes): The size of the public key (in bits) used to sign the message.

sig.dataSize (4 bytes): The number of bytes of signed data (in this case, 337).

The two hosts receive each other's packets, and close the TCP connection. The signatures are validated, and a shared encryption key is derived as shown below.

4.1.2 Diffie-Hellman parameters

The following are the values for p and g used in the Diffie-Hellman calculation:

```
p =
    f64257b7087f081772a2bad6a942f305
    e8f95311394fb6f16eb94b3820da01a7
    56a314e98f4055f3d007c6cb43a994ad
    f74c648649f80c83bd65e917d4a1d350
    f8f5595fdc76524f3d3d8ddbce99e157
    9259cdfdb8ae744fc5fc76bc83c54730
    61ce7cc966ff15f9bbfd915ec701aad3
    5b9e8da0a5723ad41af0bf4600582be5
    f488fd584e49dbcd20b49de49107366b
    336c380d451d0f7c88b31c7c5b2d8ef6
    f3c923c043f0a55b188d8ebb558cb85d
    38d334fd7c175743a31d186cde33212c
    b52aff3ce1b1294018118d7c84a70a72
    d686c40319c807297aca950cd9969fab
    d00a509b0246d3083d66a45d419f9c7c
    bd894b221926baaba25ec355e9320b3b

g =
    02
```

They have the properties that p is prime, $(p-1)/2$ is prime, and g is an element of Z_p^* of order $p-1$. These are the same values that PGP supplies for ElGamal keys.

4.1.3 Computing the shared secret

A 2048-bit shared value S is computed as:

$$S = g^{(h_{\text{local}} * h_{\text{remote}})} \bmod p = (\text{halfsec}_{\text{received}})^{h_{\text{local}}} \bmod p$$

4.1.4 Deriving the encryption key

S is expressed in a 256-byte buffer secBuf.

The first and last bytes of this buffer are discarded. The remaining 254 bytes are passed to the following key generation routine:

```
data = secBuf[1..254]
buffer[0..19] = SHA1(data);
buffer[20..39] = SHA1(buffer[0..19] || data);
buffer[40..59] = SHA1(buffer[20..39] || data);
etc. (where || denotes concatenation)
```

The first bits of the result are taken as an encryption key for the symmetric algorithm specified as symAlg in 4.2.1. The number of bits so taken depend on which algorithm is used.

For FRCRYPT_SYMALG_BLOWFISH, 128 bits are used as the Blowfish key.

For FRCRYPT_SYMALG_DES3, 192 bits are used as follows: the first 64 bits become k1, the next 64 bits become k2, the next 64 bits become k3 (parity bits are ignored, leaving 168 bits of key material). The cipher is $E_{k3}(D_{k2}(E_{k1}(M)))$.

For FRCRYPT_SYMALG_DESX, 192 bits are used as follows: the first 64 bits become the DES key k (parity bits are ignored, leaving 184 bits of key material), the next 64 bits become the input whitening inw, the next 64 bits become the output whiteneing outw. The cipher is $E_k(M \oplus inw) \oplus outw$, where $\oplus$ denotes XOR.

4.2 Link encryption

Link encryption is applied between node-pairs in order to hide the nature and characteristics of the traffic between them. Data is sent between AIPs in UDP datagrams, typically to port 51101/udp; all but one byte of the body of the datagram is encrypted using a key derived from the Diffie-Hellman key agreement. One byte is sent in the clear, which is merely a flag indicating which key to use to decrypt the packet (this is useful during the transition from one key to the next).

The algorithm to use is specified by the AIPs as part of the key agreement protocol. The default is 128-bit Blowfish; other possibilities are 168-bit 3DES and 184-bit DESX.

4.2.1 Protocol information

To transmit data of size n bytes (which must be a multiple of 8, and should start with 8 bytes of randomness), a $n + 9$ byte packet is constructed of the following form:

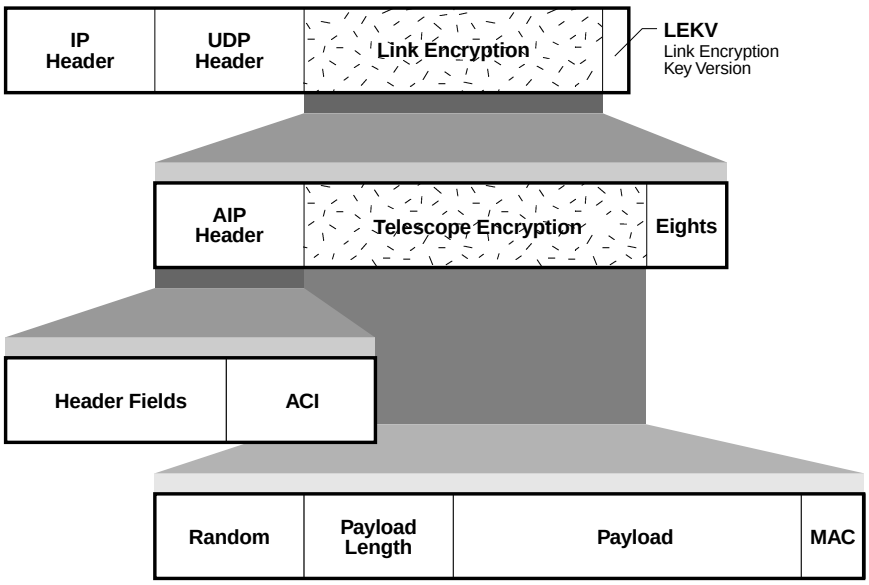


Figure 3: The encapsulation of a data packet in telescope- and link-layer encryption

```
unsigned char encdata[n+8];
uint8_t lekv;
```

encdata ($n + 8$ **bytes):** The encryption, using the algorithm specified as symAlg in 4.1.1, the key derived in 4.1.4, and an all-zeros IV, of the following data:

data (n **bytes):** the data to encrypt. n must be a multiple of 8, and should start with 8 bytes of random data. Note that

$$E_{\text{CBC}}(0, \text{RAND}||M) = \text{IV}||E_{\text{CBC}}(\text{IV}, M)$$

where RAND is 8 bytes long, IV $= E(\text{RAND})$, and $E_{\text{CBC}}(I, M)$ denotes the CBC encryption of M with IV I.

eights (8 bytes): 8 bytes of value 0x08. Used to differentiate valid packets from random dummy packets.

lekv (1 byte): The value of the lekv field in 4.1.1.

5 Client-to-AIP link encryption

A client, upon starting up, will create secure links between itself and one or more AIPs. These secure links are in most ways identical to the Inter-AIP secure links described in section 4. The differences are that:

- they are short-lived (they only exist while the client is running)

- they are under the control of the client (inter-AIP links are under the control of the NIDB)
- the client does *not* sign its Diffie-Hellman parameters, as it has no key it can use to do so privately (the AIP with which it is communicating *does* sign its parameters)

5.1 Protocol information

The only difference between this protocol and the one described in section 4 is in the contents of the fields of the Diffie-Hellman packet sent by the client to the AIP. The fields that *differ* from those described in 4.1.1 are as follows:

port (2 bytes): The local port number for the client (the port number from which packets from this client will be sent). This value will probably be ignored by the peer AIP; the source port in the UDP packets that arrive at the AIP will be used instead. This is because of IP Masquerading.

absTime (8 bytes): The current time of the client, in seconds since the UNIX epoch. (The first 4 bytes of this will be 0 for a while...)

ent (65 bytes): The entity signing the packet, in this case, none. The client's nyms do not sign the D-H connection to the first AIP; the first AIP is assumed to know the true identity of the client, and so it must not be given a signature created with one of the client's nym's keys. This field has the following subfields:

ent.type (1 byte): The entity type of the anonymous client, which will be FRENT_TYPE_ANON = 12.

ent.name (64 bytes): 64 bytes of 0x00.

sig (65 bytes): 65 bytes of 0x00.

6 Telescope Encryption

Telescope encryption is the set of encryption layers that is designed to provide client-to-wormhole confidentiality. It is called telescope encryption because the layers can be visualized as an old fashioned telescope which collapses in on itself, leading to a set of concentric tubes. Each tube is the layer of encryption that is removed as a packet travels from the client, and added as the the packet travels to the client. In this analogy, the ends of the tube are each connected to an AIP. We use the terms "telescope" and "route" somewhat interchangeably.

There are two types of telescope encryption used in the system, authenticated and anonymous. Authenticated telescopes use a signed ROUTE CREATE request, and create a route that can be used for arbitrary destination hosts. Anonymous telescopes can only be used to connect to certain sets of defined hosts, such as the database servers. They are useful for a set of initialization purposes, when there is no appropriate certificate available with which to create routes. We discuss authenticated telescopes first.

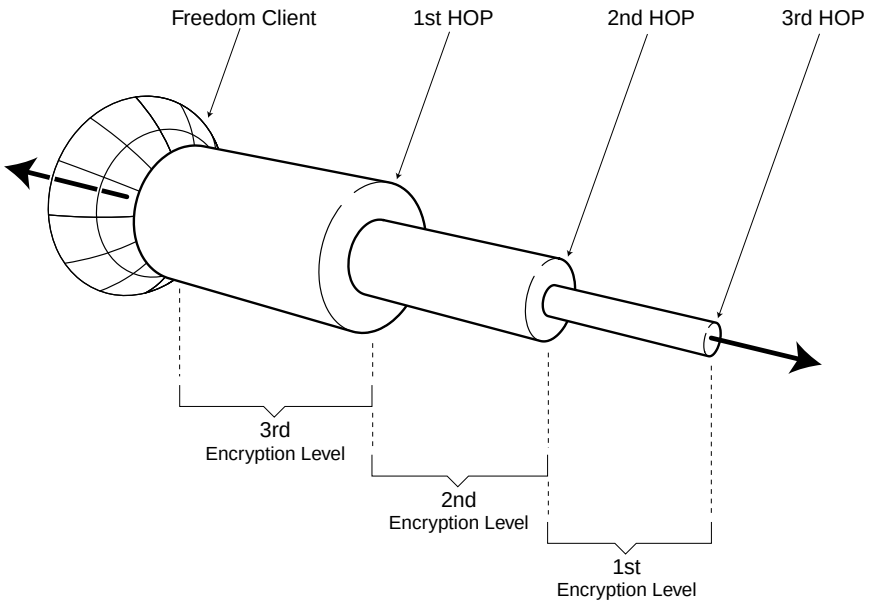


Figure 4: Telescope encryption

An authenticated telescope is one created with a ROUTE CREATE packet signed by a valid nym. Only a nym can create an authenticated telescope.

For an anonymous telescope, the signature and identity fields are left blank (all zeros). Clients and AIPs can make anonymous telescopes.

6.1 Administration

The client selects a chain of AIPs to use to build the telescope. The first AIP must be one to which the client has a secure link, as described in section 5. As well, each pair of consecutive AIPs in the chain must be a pair that has a secure link between them, as described in section 4. The client queries the NIQS to find the set of all AIP pairs with secure links between them (so as to avoid leaking information about which AIP pairs the client is interested in). The last AIP in the chain is called the “last-hop” or “exit” AIP.

6.2 Route Setup

The client constructs a ROUTE CREATE packet which contains secrets to be shared one with each AIP in the chosen chain. Nested El Gamal encryption with the AIPs’ encryption keys is used to securely transmit the secrets (and to ensure that no AIP knows more than who are the previous and next AIPs in the chain).

6.2.1 Protocol Information

A ROUTE CREATE packet has the following format:

```
byte rand[8];
uint8_t vers;
uint8_t type;
uint8_t pri;
byte pad[3];
uint16_t aci;
RouteHeader rtehead;
byte routeCrypt[1088];
```

rand (8 bytes): 8 bytes of random data

vers (1 byte): The constant FRPKT_VERSION_CURRENT = 1

type (1 byte): The constant FRPKTTYPE_AIP_ROUTE_CREATE = 2

pri (1 byte): The priority of the packet (currently ignored). [Eventually, nym certificates will indicate which priority packets they are allowed to use.]

pad (3 bytes): 3 bytes of 0x05 (sic)

aci (2 bytes): The incoming Anonymous Circuit Identifier.

rtehead (136 bytes): The header for the route, consisting of the following fields:

- **rtehead.keySeed (128 bytes):** The 128-byte result of $g^{\text{encrand}} \bmod p$, where encrand is a random 1024-bit number, and g and p are the parameters of the destination AIP's El Gamal encryption key.
- **rtehead.keyAlg (1 byte):** The encryption algorithm used to encrypt the rest of the packet, below. The allowed vaules here are the same as in 4.1.1.
- **rtehead.cryptKeyVers (1 byte):** The version number of the public key used to derive keySeed, above.
- **rtehead.pad (6 bytes):** 6 bytes of 0x06

routeCrypt (1088 bytes): An encrypted buffer. The algorithm used is that specified in keyAlg, above. The key is derived as follows: the value of encrand chosen above and the destination AIP's public El Gamal Encryption key (g, p, y) are used to calculate the 128-byte value $y^{\text{encrand}} \bmod p$. The first two bytes of this 128-byte value are discarded. The next 24 bytes are passed to the key generation routine described in 4.1.4. The next 8 bytes are used as an IV for the algorithm. When decrypted, the buffer should contain the following:

- **bckKeyAlg (1 byte):** The symmetric encryption algorithm to use to encrypt data heading along this route from the Internet to the client. The key to use is generated from bytes 34 to 57 (inclusive, starting from 0) of the 128-byte value $y^{\text{encrand}} \bmod p$ calculated above.

fwdKeyAlg (1 byte): The symmetric encryption algorithm to use to decrypt data heading along this route from the client to the Internet. The key to use is generated from bytes 58 to 81 (inclusive, starting from 0) of the 128-byte value $y^{\text{encrand}} \bmod p$ calculated above.

flags (1 byte): If bit 0 (the lsb) of this byte is set, this is the last hop. Otherwise, this is an intermediate hop.

If bit 1 is set, an ACI will be present in specAci field, below. This is only valid if bit 0 is also set.

nextEnt (65 byte): The entity specifying the next hop in the route (as described in 4.1.1), if this is not the last hop, or 65 bytes of 0x00 if this is the last hop.

specAci (2 bytes): If bit 1 of flags is set, this field contains the ACI to use at the last hop (this is used when a client wishes create a new route with the same exit information, so that existing TCP connections, for example, stay up). Otherwise, this field contains 0x0202.

expTime (4 bytes): The requested absolute expiry time of the route created by this packet, in seconds since the Unix epoch. We note that this is a 4-byte field, whereas most of the absolute timestamps in these protocols are 8 bytes; this is indeed a Y2106 problem.

pad (6 bytes): 6 bytes of 0x06

After the above, the buffer will contain one of the following, depending on whether the flags byte indicates that this is the last hop:

If bit 0 of flags is 0 (this is not the last hop), what will follow is another RouteHeader and corresponding routeCrypt, expect that the size of routeCrypt will be 216 bytes shorter than the one in which it is contained.

If bit 0 of flags is 1 (this is the last hop), what will follow is an authentication section (144 bytes), followed by random data to the end of the buffer. The authentication section is as follows:

nonce (8 bytes): A random 8 byte nonce used to detect replay attacks.

type (1 byte): The type of the entity which signed this route creation packet. Possible values are FRENT_TYPE_ANON = 12 and FRENT_TYPE_NYM = 6.

name (64 bytes): The name of the entity which signed this route creation packet.

sig (65 bytes): The DSA signature (as in 4.1.1) of the SHA1 hash of the following data:

- The 128-byte value $y^{\text{encrand}} \bmod p$ calculated above (128 bytes)
- rtehead.keyAlg, rtehead.cryptKeyVers, rtehead.pad (8 bytes)
- bckKeyAlg, fwdKeyAlg, flags, nextEnt, specAci, expTime, pad (80 bytes)
- nonce, type, name (73 bytes)

If the type is FRENT_TYPE_ANON, this should be 65 bytes of 0x00.

pad (6 bytes): 6 bytes of 0x06

Also, if this is the last hop, a MAC key (shared between the last hop and the client) will be generated from bytes 82 to 101 (inclusive, starting from 0) of the 128-byte value $y^{\text{encrand}} \bmod p$ calculated above.

When the last-hop AIP receives a ROUTE CREATE packet that is correctly signed, it will return a ROUTE CREATE ACK packet, indicating success. A ROUTE CREATE ACK packet is just a data packet (as described in 6.3) with the special type FRPKTTYPE_AIP_ROUTE_CRTACK = 5, and with payload the 2-byte exit ACI selected by the last-hop AIP. This value can be later used by the client to create a new route, while maintaining the same exit imformation, as described in 6.2.1.

If any AIP receives a ROUTE CREATE packet that is for some reason malformed (for example, the signature is bad), it will send a ROUTE DESTROY packet back towards the client. The format of the ROUTE DESTROY packet is described in 6.4, below.

6.3 Telescope encryption

The majority of the packets on the network are DATA packets. These, like all packets on the Freedom network, are sent as the payload of a link-encrypted packet, as described in 4.2. These DATA packets are of a fixed size in order to avoid attacks based on size correlations of packets.

6.3.1 Protocol Information

DATA packets have the following format:

rand (8 bytes): 8 bytes of random data

vers (1 byte): The constant FRPKT_VERSION_CURRENT = 1

type (1 byte): The type of the packet. Normally, the constant FRPKTTYPE_AIP_DATA = 1, but possibly different for other packet types (see 6.3.2 above for an example of using FRPKTTYPE_AIP_ROUTE_CRTACK = 5).

pri (1 byte): The priority of the packet (currently ignored).

pad (3 bytes): 3 bytes of 0x05 (sic)

aci (2 bytes): The incoming Anonymous Circuit Identifier.

payload (272 bytes): The (possibly multiply-)encrypted data. The plaintext is the following:

payrand (8 bytes): 8 bytes of random data

paylen (2 bytes): The length, len, of the plaintext data, at most 252 bytes.

plaintext (252 bytes): The plaintext data, of length len, followed by 252-len bytes of value c, where $c = 8 - (\text{len} \bmod 8)$.

mac (10 bytes): The SHA1-HMAC-80 (using the key derived in 6.2.1) over the above 262 bytes, if such a key is available. If for some reason it is not (for example, with some packets of type different from FRPKTTYPE_AIP_DATA), this field should contain 10 bytes of 0x06.

The 272-byte plaintext is encrypted in the following way:

For packets travelling from the client to the Internet, the client will encrypt the data first with the forward encryption algorithm and key it specified in the section of the ROUTE CREATE packet that was read by the *last* AIP in the route chain; then with the forward encryption algorithm and key that was read by the next-to-last AIP in the route chain, and so on. (See fwdKeyAlg in 6.2.1.)

As the packet travels along the route, each AIP will decrypt the payload. After the last AIP decrypts the payload, it will be the plaintext, and the MAC will be checked.

For packets travelling from the Internet to the client, each AIP in turn (starting with the last AIP in the route) will encrypt the data with the backwards encryption algorithm and key the client specified in the section of the ROUTE_CREATE packet. (See bckKeyAlg in 6.2.1.)

When the packet arrives at the client, the client will successively decrypt the (now multiply-encrypted) payload, first with the algorithm and key shared with the *first* AIP, then with the algorith, and key shared with the second AIP, and so on. When the plaintext is recovered, the MAC will be checked.

6.4 Teardown

If for any reason, an entity needs to tear down an existing route (the entity must be one of the AIPs involved in the route, or the client which created the route), it will send a ROUTE DESTROY packet along the route. If the entity wishing to destroy the route is an AIP in the middle of the route, it should send two ROUTE DESTROY packets: one towards the client, and one towards the Internet.

The format of a ROUTE DESTROY packet is almost the same as a DATA packet, with the following exceptions:

type (1 byte): The constant FRPKTTYPE_AIP_ROUTE_DESTROY = 3

payload (272 bytes): 272 bytes of random data

7 IP layer handling

The client sits at the NDIS level on the Win95 stack. It removes certain identifying data from a packet before encrypting and sending it. This data is the IP source address, and the IP and UDP or TCP checksum. The checksums are removed to avoid brute force attacks on the missing IP address data. The MAC in the route data (telescope) packet ensures that the data has not been corrupted in transit. The wormhole proxy adds in

the appropriate source IP, changes the port, and constructs a new set of IP and TCP (or UDP) checksums.

There is a list of acceptable TCP and UDP ports which is enforced by both the client and the wormhole. That set of ports are those needed to allow the supported protocols to run over the Freedom Network. The restriction exists to minimize possibilities for abusive/hacking behavior over the network, and confirms to the principle of that which is not explicitly permitted is denied. Raw IP is explicitly not allowed through the network.

All packets are fragmented to 252 bytes before sending. The wormhole convinces the client that the path MTU is 252 bytes, so the client's standard IP stack will deal with fragmentation for us. The wormhole actually framgents inbound data to this size, and the client's normal stack does the reconstruction. The packet size is intended as a compromise, and will probably be replaced by a different compromise involving multiple packet sizes in the future.

The 252 byte fragment is telescope encrypted for each Freedom Server along the path, link encrypted for the first hop, and then placed in a UDP packet for sending to port 51101.

The data travels over the network, being link decrypted and telescope unwrapped at each point. The data is routed to the next hop by use of an Anonymous Circuit ID (ACI) mapping table; data coming in over a given ACI is encrypted or decrypted with a key that maps to that ACI, and then link encrypted and sent on its way. If the ACI indicates that packets from that host are sent to the wormhole, then they are. The wormhole will map the ACI into a local TCP (or UDP) port, and map its source port to the source port on which the client is expecting to receive a response. The wormhole will then insert its IP address into the packet, calculate IP and TCP header checksums, and insert the packet onto the Internet.

When a TCP response comes back from an Internet server to the wormhole, the wormhole will break the stream into chunks, add a cryptographic authenticator to the each, and then pass them to the AIP for encryption. The AIP only applys a sinlge layer of encryption. This may be counter-intuitive, if you expect the AIP to add layers, and see them stripped off as the packet travels through the network. This is not done, as the AIP must not know the set of keys it would need to add all the layers of encryption.

The information is passed along the route indicated by the ACIs until it reaches the client, where the software removes the several[2] layers of encryption, inserts the appropriate source address and port back in, recalculates the checksum, and pops the packet back into the IP stack.

8 Nym Creation

The nym creation process is described in detail in the "Untraceable Nym Creation on the Freedom Network" white paper. The process of paying for a nym is intentionally separated from the process of creating a nym, so that we can build a wall which payment identity information does not cross.

[2] This is 4 layers of encryption for a three-hop route; the three telescope and a link layer.

9 Application layer handling

9.1 Client-side application proxies

Client-side application proxies are designed to remove identifying information from application streams before it reaches the Internet. This is in contrast to the AIP-side packet-filters, which aim to prevent hacking attempts by the client, and assume that the data stream has already been sanitized. The location and assignment of responsibilities is designed to reduce the need for trust in the system.

Outgoing text in a variety of protocols is scanned by the text scanner. The text scanner looks for strings that match those entered into the client's "Word Scanning" dialog box in a case-sensitive manner. Any matches will result in a warning dialog box being displayed to the user. The data is not sent until the user has approved the sending.

9.1.1 DNS

DNS packets are intercepted and sent over the Freedom network (otherwise, a collaboration between your local DNS server and the wormhole could reveal a lot). There are no modifications made to the request.

9.1.2 SSL

SSL packets arrive at the Freedom client already encrypted by the web browser. As such, there is nothing we can do to reliably remove or edit what data they send.

9.1.3 HTTP

HTTP GET and POST messages are sent through the text scanner. Each nym is allocated a seperate cookie jar. The "Referer:" header is left intact.

9.1.4 SMTP/AMTP

Outbound SMTP messages are intercepted by the client-side SMTP proxy. The proxy sanitizes the headers of the message, including replacing the user's real email address with that of a nym. It then checks if all the recipients are Freedom users. If so, it fetches their keys from the Nym Server, and encrypts the multiple times, once for each nym. If there are non-nym recipients, a cleartext copy of the message is kept. The encrypted (and plaintext, if needed) messages are delivered over an anonymous connection to the FMG-MAIP, which applies its processing logic (signature processing, spam and abuse control), and send them either through the MAIP cloud for nym recipients or SMTP to a non-Freedom recipient.

9.1.5 POP3

The POP3 proxy intercepts POP3 requests to the user's pop server. The proxy will collect all mail after authenticating, and keep the user's mail client waiting. The POP3 proxy will correlate messages to prevent the mail client from seeing the redundant

versions of messages created by Reply Blocks. As such, the mail client (Eg, Netscape Mail or Outlook) authenticates itself to a local POP3 proxy, that proxy authenticates itself to the POP server, the proxy downloads the mail, decrypts it, and only then will the mail client see a number of messages for reading.

Note that the connection from the POP3 proxy to the user's POP3 server is *not* made through the Freedom Network, but rather is a regular TCP connection over the Internet. This is because the expected POP mailbox is the users standard POP mailbox, and we don't want to associate a nym with that mailbox. (All inbound mail is encrypted so that there is no way for an observer to distinguish to which nym it is addressed.)

9.1.6 NNTP

The body of outgoing news postings is passed through the text scanner. The message is encapsulated by the client news proxy in a mail message, and sent (un)encrypted to a mail2news gateway. Reading Usenet news is accomplished via a web-based news reading site.

9.1.7 Telnet/ssh

Any telnet/ssh based protocol can be passed over the Freedom network to acceptable target ports. The information is run through the text scanner. Uses for this include private contributions to source trees via CVS over SSH, access to MUDs, and other telnet based information services.

9.2 AIP-side application proxies

The server side packet filter in the wormhole is used to offer a level of protection for the Internet from abuse by our customers, by ensuring that outbound packets only reach certain target ports. Our goal is to make it difficult to use the Freedom network for hacking activity, however, since a vulnerability that exists on a target machine may be exploited without the Freedom Network, we simply attempt to be a good neighbor. Ultimately, defending your hosts and networks is your responsibility.

The FMG-MAIP is responsible for outbound spam control for the Freedom network. It does this by placing programmable limits on the volume of mail a given user may send. The limit is initially set based on the class of customer they are: Paid users have a higher quota than trial users. Either type of user's mail quota may be changed by the Zero Knowledge Abuse Center [3] in response to complaints or other issues. The exact quotas are not published so that spammers can not send a number of messages just under the quota. For the same reasons, quotas may vary slightly on a per user basis. The FMG-MAIP also implements the abuse blocking controls, where people can request that they not recieve email from given nyms.

[3] `http://www.freedom.net/support/abuse.html`

10 Incoming mail handling

Incoming mail for Nyms is received by a Freedom Mail Gateway. The gateway ensures that the mail is for a valid nym which is able to receive mail. (There may be valid nyms which do not have any reply blocks, or for other reasons are unable to receive mail.) If the mail is accepted, a copy of the message is encrypted with each reply block that the nym has created. Each copy of the message will be routed through a series of MAIPs using the AMTP protocol. Each inter-MAIP connection is done over a non-anonymous TCP connection to port 51112.

A reply block contains one or more layers of key, next-hop, message-block tuples. The key is used to encrypt the message before sending the message to the next-hop, which may be a MAIP or a POP mailbox.

Eventually, a number of (encrypted) copies of the message will arrive in the user's POP box (one for each reply block he has set up, unless some AIPs are down). As described above, the POP3 proxy will automatically decrypt the Freedom messages and remove the duplicate copies before passing them on to the user's POP3 client.

11 Collecting Network Information

Periodically, the Network Information Status Server (NISS) collects various data about the availability and quality of service of the Freedom Network. The exact details of what information is gathered will be given in a future version of this white paper.

12 Conclusion

Every effort has been made to make Freedom the most integrated, strongest and easiest-to-use privacy system available, and we believe we have achieved this goal. No system is completely infallable, however, but this white paper, in conjunction with "Freedom 1.0 Security Issues and Analysis", will show the reader the extent to which the system is secure under ordinary, and even extraordinary circumstances. We maintain a policy of full disclosure of the system's workings and weaknesses in an effort to be up-front and honest to the community of Freedom users and interested parties.

A Change History

November 29, 1999 Made the following changes:

2.3 AIP keys are signed by the monthly key, not nym server key

2.4 Corrected identity of actor to be FMG-MAIP in para 2

2.5 FMG-MAIP added 2nd database

3.x Added users of various databases

3.key-query added times when clients make cleartext, non-anon queries

3.nym-server expanded nym server, seperated token server to match reality

3.fmg-maip added block rules

4.2 fixed port for data packets

7 the last paragraph of section 7 was substantially cleaned up for clarity in what software performs which actions

9.1 noted that the user can prevent data xfer in dialog box.

9.1.3 added note about cookie jars

9.1.4 mail encryption was wrong, pop handling was insufficiently explained

9.1.6 expanded nntp handling

9.2 fmg-maip abuse control

A Fixed Y2K bug in Change History section

November 23, '99 Released Initial Version.