

# Anon-IP version 2

Adam Back  
Zero-Knowledge Systems Inc  
888 de Maisonneuve E., suite 600,  
Montréal (Québec), Canada, H2L 4S8  
e-mail: adamb@zeroknowledge.com

11th June 2000

## Abstract

The anon-IP v1 protocol as used in freedom version 1.0 and 1.1 has a number of short-falls in security, reliability, efficiency and functionality. Here we describe a revision of the anon-IP protocol: anon-IP v2, which includes a new route create mechsism, new data packet, new link layer and new telescope layer which addresses these issues.

## 1 Introduction

The anon-IP protocol in freedom version 1.0 has a number of short-falls in security, reliability, efficiency and functionality. Here we describe a revision of the anon-IP protocol: anon-IP v2. which includes a new route create mechsism, new data packet, new link layer and new telescope layer which addresses these issues.

The problems are that:

- *security* the route create packets are of different size to data packets, this makes traffic analysis easy as there is little cover traffic for route create packets.
- *efficiency* forward-secrecy is provided by a separate link layer encryption with the result that every byte passed through an AIP having to be processed using block cipher encrypt or decrypt three times. (One decrypt for incoming link layer, one encrypt data, and one encrypt for outgoing link layer). This is inefficient.
- *bandwidth efficiency* the packets have a high overhead: 45 bytes of anon-IP overhead for every 252 bytes of data sent.
- *security* forward-secrecy is only hop-by-hop and not end-to-end. Any single rogue node could strip forward-secrecy by: removing the link layer, and presenting the contained telescope encryption to each node in turn using a subpoena attack. End-to-end negotiation of forward-secrect keys would provide more secure forward-secrecy.
- *reliability* there is no way to distinguish where failures occur as the route create protocol is non-interactive. (This is the case with our current implementation, but is this an unavoidable consequence of the design, or an implementation short-fall?)
- *functionality* the size of the route create packets imposes a limit on the maximal length of routes which can be created. The limit is 5 hops in the current implementation.

- *reliability* in the implementation of the anon-IP v1 protocol, the route create packets are sent via UDP, and re-try semantics are written into the client. The use of UDP rather than TCP is an implementation decision rather than an inherent design problem. With UDP route create packet transport, the size of the route create packets (1200 bytes) also causes network reliability problems because the maximum UDP packet size that can be relied upon on the Internet is 576 bytes. Some users have been unable to connect for this reason.

## 2 Route Create Protocol

Here we describe a replacement route create protocol which addresses the issues described in section 1.

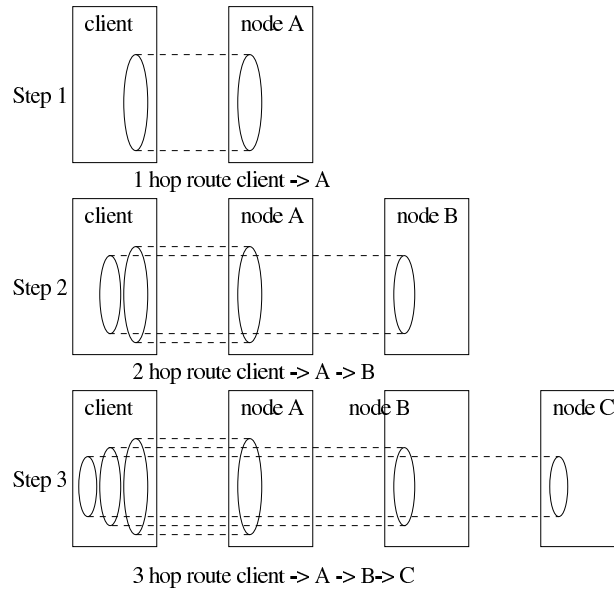


Figure 1: Anon-IP v2 route create mechanism

The version 2 route create process is shown in figure 1, and the steps in the process can be described as follows:

1. Create a 1 hop route between the client and node A.
2. Create a 2 hop route between the client and node B, by tunneling a route create session between client and node B over the 1 hop route between client and node A.
3. Create a 3 hop route between the client and node C, by tunneling a route create session between client and node C over the 2 hop route between client and node B.

### 2.1 Networking Architecture

The key negotiation messages are sent using a reliable transmission protocol (TCP). The encrypted tunnel packets are sent using an unreliable transmission protocol (UDP). A single hop connection is shown in figure 2.

Two hop routes are created by tunnelling a TCP connection from the client to the 2nd server (C) through the tunnel between the client (A) and the first hop server (B) as shown in figure 3.

This process can be repeated indefinitely to create arbitrary length routes.

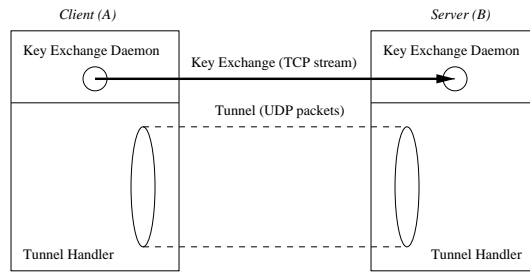


Figure 2: Anon-IP v2 One-Hop Route

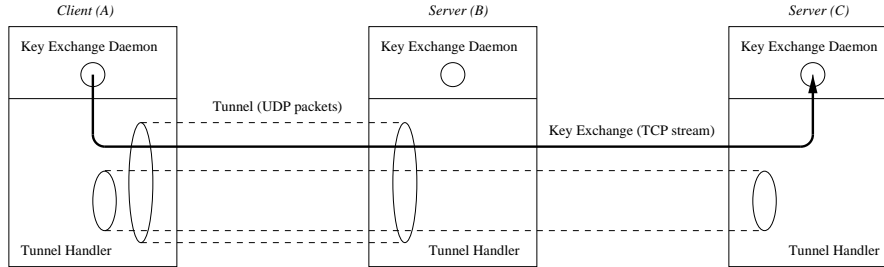


Figure 3: Anon-IP v2 Two-Hop Route

### 3 Key negotiation

We use a variant of the *Diffie-Hellman Station-to-Station protocol* (STS) for key negotiation.

We refer the entity establishing the connection as the *client*, and the entity to which the connection is being established as the *server*. There are two route-create variants:

- *anonymous* (or more strictly *unauthenticated anonymous*) client unauthenticated, server authenticated.
- *authenticated* client authenticated. Two authentication mechanisms are possible:
  - *pseudonymous authentication* client identifies the users pseudonym to the server using a signature and the pseudonym’s certificate.
  - *anonymous authentication* client proves it has the right to obtain service using a *blind credential* show protocol.

In the following discussion we use the term *authenticated connection* to encompass both *anonymously authenticated* and *pseudonymously authenticated* connections.

We remark that the term *anonymous* is triply overloaded here:

- *Anonymous Diffie-Hellman* (ADH) is often used in the literature to refer to Diffie-Hellman without any authentication.
- *anonymous STS* is the term used to refer to hiding the client’s identity from attackers in STS [?]. We call this STS property *identity privacy* to disambiguate.
- In addition in the context of ZKS freedom product an *anonymous connection* refers to the client being *anonymous*, either by virtue of being *unauthenticated*, or by using an *anonymous authentication* mechanism.

The variant of STS we use is modified in the following ways:

- We use static keys (traditionally STS uses *ephemeral* keys).
- We allow client authentication to be optional, and accordingly only provide key confirmation when the client does choose to authenticate itself.
- As with authenticated STS, the static keys are authenticated using a signature function, and the signature public keys are certified by one or more other entities, such as a trusted certification authority. The trust metric used to decide whether a certificate is valid is outside the scope of this paper.
- We use Kocher et al's generalised approach to authentication as used in SSL3 [?]. The approach is to derive keys from negotiated secrets and all prior messages, and similarly to make per connection authentication tokens depend on all prior messages. This is a robust generic defence against subtle message modification and deletion attacks, such as the *version roll-back attack* and *supported ciphersuite modification attack* which SSL2 [?] is vulnerable to.
- We use a key derivation function KDF() to avoid any biases in the raw Diffie-Hellman negotiated parameter which may arise from the fact that only a subset of the bits of a negotiated DH parameter have been proven to be secure (see [?] for details).

- STS can provide authentication whilst retaining *identity privacy* with respect to active and passive attackers. We retain the *identity privacy* option, but for the client only.

STS *identity privacy* is achieved by transferring the clients identifying authentication token to the server in encrypted form, so that only the server can read it. The server learns who the client is, but passive attackers watching the communications, and even active attackers modifying and replaying packets must be unable to discover the clients identity.

The anonymous protocol must have the following security properties:

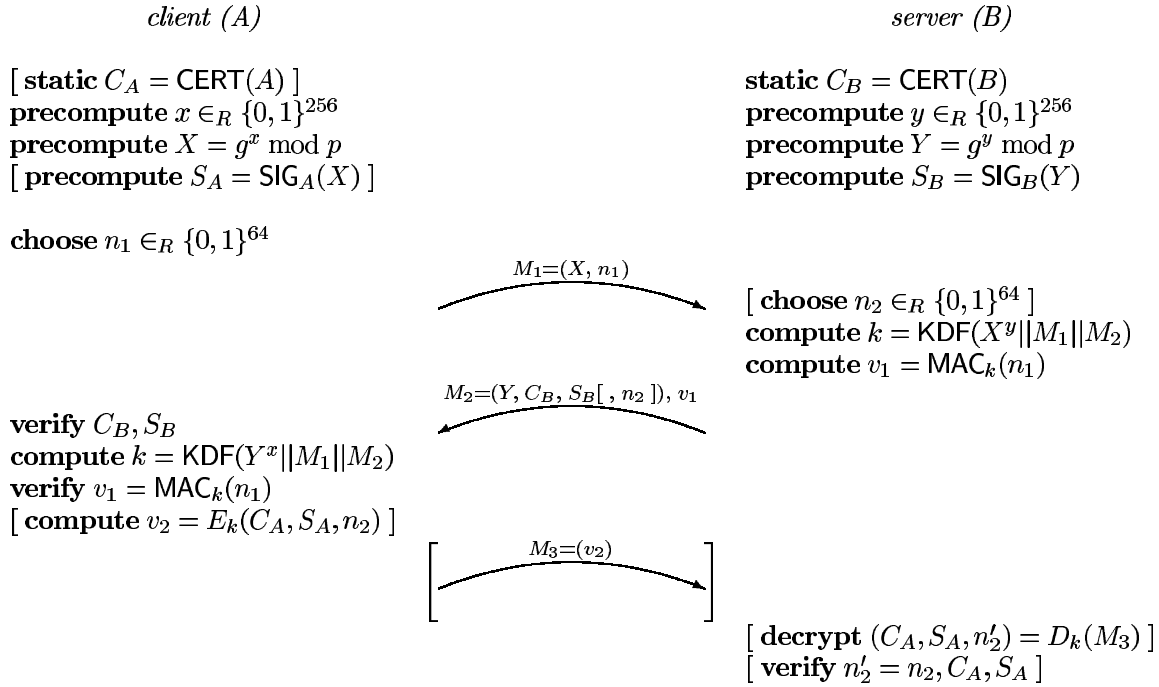
- The client must receive *key confirmation* from the server (provides protection from server message *replay* against the client).
- The protocol does *not* need to prevent the attacker replaying client messages to the server, because replay attacks do not make sense for anonymous connections – anyone can connect. (There is a separate re-key protocol, which is described in the next section, so replay is not an issue for *connection hi-jack attacks*.)
- Both parties have *static keys*, this allows trade-offs to be made between immediacy of *forward-secrecy* provided against CPU resources expended generating replacement *ephemeral* keys. Certificates are sent within the protocol, so either party can change static keys at any frequency it chooses (up to a new key per connection, which corresponds to *ephemeral*, or *immediately forward-secret*.) The use of static keys makes nonces necessary for key confirmation.
- With STS the server can not have *identity privacy* from active attackers. (The protocols *could* be modified to provide server *identity privacy* against passive observers, but this would cost an extra round, and is of limited value, as any passive attacker with the same ability as other clients can establish a connection to the server and obtain it's identity).

In addition where the client is authenticated, the following requirements are added:

- the client must send the server *key confirmation* (the server may be trying to maintain one active connection between each of it's network neighbours, it therefore needs to know whether a connection was successful).
- The client must *authenticate* itself to the server, but must retain *identity privacy* with respect to observers, and active attackers. (The client authenticates itself pseudonymously using it's pseudonymous certificate and corresponding asymmetric key, or using a one show blind token proving right to access service).

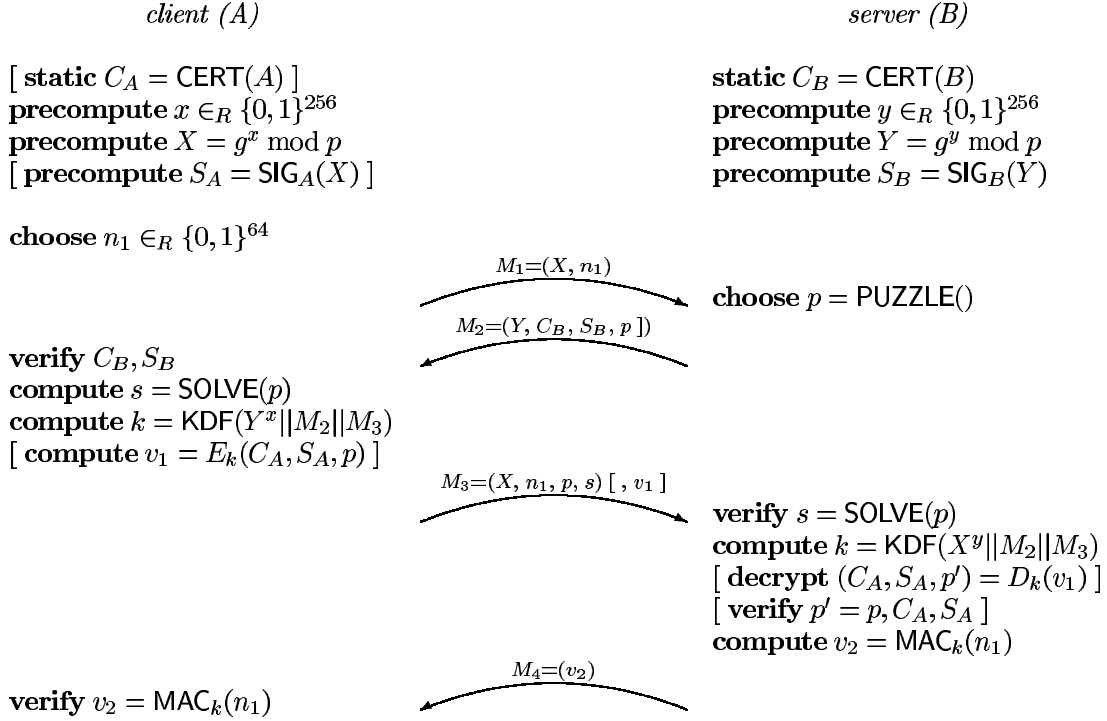
### 3.1 Without DoS protection

Computation steps and messages shown in square brackets are optional, and only used when the client wishes to identify itself.



### 3.2 With DoS protection

When server is under attack this protocol is used instead. Note the first client message  $M_1$  is the same as without DoS protection, as the client does not know a priori that the server is under attack, it discovers this by the alternate message  $M_2$  returned by the server containing *client puzzle*  $p$ . The client must present the server with the solution  $s$  to the client puzzle before the server will be willing to proceed with the steps of the protocol which involve significant CPU expenditure.



To minimise the state the server must keep during connection establishment prior to being provided with a solution to the issued client puzzle, the client resends all nonces and parameters in message  $M_3$ .

Two extra rounds in the protocol are incurred in the case of client anonymous connections because the client requires *key confirmation*, and key confirmation requires CPU resources which the server is not willing to commit to prior to receiving the puzzle solution, which doesn't happen until round 3. Anonymous connections therefore take an extra round beyond the number which might be expected.

It is desirable to arrange that the server does not need to keep state prior to accepting the connection, otherwise memory depletion attacks may become relevant. An approach to defending against connection depletion suggested by Juels and Brainard in [?] is to derive the puzzle from a server secret and the TCP connection parameters<sup>1</sup>. However this approach doesn't work as a defense against CPU resource DoS, because there are no TCP connection parameters which can be guaranteed to be unique and fairly chosen (the server IP address is fixed, and the server port also typically fixed, and the client IP address and port are under the control of the client's software). Were one to do this client puzzles could be double spent by reusing connection parameters (the client can free the parameters ready for reuse by closing the connection).

The server must therefore issue a random puzzle, and retain it for the duration of the key negotiation.

## 4 Forward-Secrecy and Re-keying

In order to provide forward-secrecy a mechanism to select new keys for a connection is required. We give specifications for two types of re-key: symmetric re-key, and public re-key. The symmetric re-key achieves *forward-secrecy*, and the public re-key achieves both *forward-secrecy* and *session key compromise recovery*.

To avoid connection hi-jacking on unauthenticated routes, and to hide re-keying activity from earlier hops in the route, re-key requests are sent to the key exchange daemon on the machine at the other of the tunnel being re-keyed through the tunnel terminating at that machine as shown in figure 4.

<sup>1</sup>the TCP connection parameters are the quintuple formed by the protocol class, client IP, client port, server IP and server port

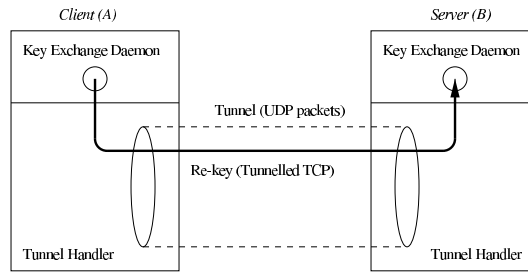
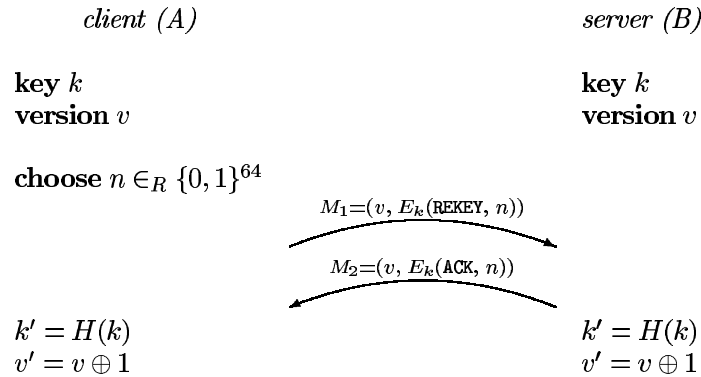


Figure 4: Anon-IP v2 Re-key mechanism

## 4.1 Symmetric Re-key

As an authenticated secure tunnel is already available at this point, forward-secrecy can be achieved using symmetric key constructs. A reliable TCP connection will be established through the tunnel, as depicted in figure 4, and the tunnel has replay protection, and so only two messages are needed.



To avoid race conditions we define that only the client can initiate a re-key.

## 4.2 Public key re-key

The public key re-key protocol is to execute the same key negotiation protocol used to create the tunnel, but through the tunnel as depicted in figure 4. Doing this ensures that even unauthenticated connections can not be hi-jacked by injected or replayed encrypted re-key packets. This works because the keys derived for the initial unauthenticated connection are used to authenticate the re-key.

## 5 Tunnel Layer

In anon-IP v1 protocol, the link layer protocol provides forward-secrecy. Each pair of nodes in the network maintains a forward-secret key with each of it's neighbours in the network.

The keys are negotiated using the authenticated Diffie-Hellman protocol. New (forward-secret) link layer keys are negotiated every half hour, and the re-key may be initiated by either node.

With the anon-IP v2 route create mechanism, the payload no longer needs to be encrypted at the link layer, because link layer encryption was only to provide forward-secrecy, and content forward-secrecy is now provided directly by the keys negotiated end-to-end during the anon-IP v2 route create protocol. However, the ACI (Anonymous Circuit Identifier) field and supporting fields (Sequence No, Type, Version, Priority) must be encrypted at the link layer, because they can not be encrypted by the telescope layer, and sending them in the clear would remove anonymity. In addition this information should be encrypted using keys which are forward-secret, because a log of inter-AIP traffic together with recovery of the link layer keys would remove privacy.

As a result the link layer remains conceptually, but only encrypts this header information, so is no longer a true *layer*, and might be more accurately called a *collar* to depict a tunnel with gaps in it with clear text showing through. This is shown in figure 5. The encrypted header information is small, and we restrict it to be under one cipher block size (8 bytes) so we can optimise and avoid needing a separate IV.

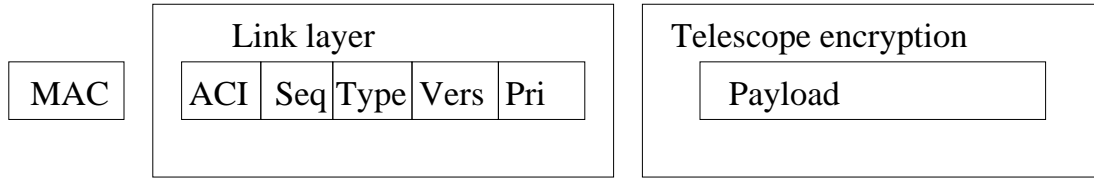


Figure 5: Anon-IP v2 packet

The link layer in anon-IP v2 uses the authenticated version of the STS variant specified in section 3. This protocol provides client *identity privacy*, which is not generally required as servers are generally publicly identified, but *identity privacy* is efficient.

Anon-IP v1 has an unauthenticated link layer between the client and the first node, as well as authenticated pair-wise link layers between nodes. This is necessary to prevent packet correlation between packets going over different routes which share the same first hop.

Anon-IP v2 can dispense with the link layer between the client and the first node, because the *authenticated connection* protocols for clients provide *identity privacy*, and because packets for different routes which share the same first hop are sent down the same tunnel from the client to the first hop. The anon-IP v2 tunneling directly prevents packet correlation at the first hop, because all packets to that node are routed through the same tunnel. This is depicted in figure 6.

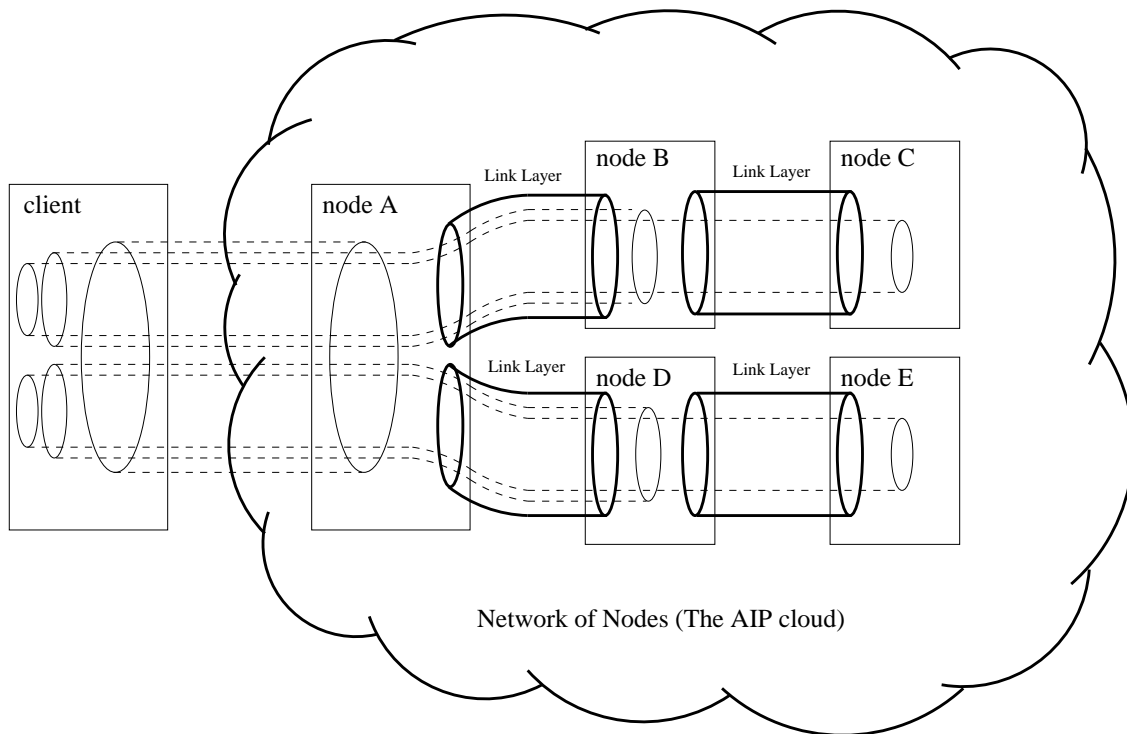


Figure 6: Anon-IP v2 Showing no link-layer from client

Avoiding the link layer from the client to the first node in addition means that in a network supporting

only one hop routes the nodes do not need to maintain link layers. Also, in a network with a threat model which allows ACIs to be sent in the clear up to it's supported hop-length, the nodes do not need to maintain link layers.

## 6 Packet Spec

packet	bytes	description
MAC	8	SHA1-HMAC-64
...		

Table 1: anon-IP v2 packet

...

## 7 Open Questions

A collection of open questions and notes:

- Will we need to mix different priority packets over the same connection?
- Using nested headers, so on outgoing packets each hop decrypts to reveal headers for next hop, and on incoming packets each hop adds headers for next hop and encrypts doesn't work properly, because on incoming packets the node has to do the lookup by ACI for the next hop. So you end up with lookup on the incoming direction and not the outgoing. If you're going to use lookup in one direction you may as well use it in both directions, and save the packet overhead of nesting. Additional problem with nesting is that the exit node learns how many hops the connection is, based on the remaining space left for payload.
- We should use a table to lookup connection information by ACI. Then we can store IVs, and telescope keys for each direction, and avoid packet bloat.
- For security you really need a MAC for each hop, using end-to-end negotiated keys between the client and each hop. This takes us back to the nested problem of leaking the size to the exit node, and it takes lots of space too. Is there a way around this?
- How much security does link layer MACs only provide? (Vulnerable to Wei Dai's active attack, at minimum). Or link layer MACs plus one end-to-end MAC between the client and the exit node? (Ugly to have to differentiate between exit node and middle node, as the tunnel doesn't otherwise know at the telescope layer. (It clearly knows at the key negotiation layer, and during key route creation).)
- Will we need to support unauthenticated AIPs in the long term? (For example if clients are AIPs, perhaps some AIPs don't have certified public keys).
- Will we want *identity privacy* against passive attackers for servers? (If everyone is an AIP, perhaps we don't want to leak the servers identity in the clear).
- Is *identity privacy* for both client and server possible against active attackers? (Perhaps a simultaneous authentication protocol, where the parties are authenticated only if both parties played fairly. If one party cheats, the other party doesn't reveal his identity.) Do we care enough... it sounds like a solution would be expensive and interactive.
- How many rounds of communication will it take to show a blind credential? Are there advantages to the interactive show protocols? Or can we use the non-interactive show protocols?