E. Rescorla
RTFM, Inc.
INTERNET-DRAFT IAB
draft-iab-model-01.txt May 2004 (Expires November 2004)

Writing Protocol Models

Status of this Memo

Abstract

The IETF process depends on peer review. However, IETF documents are generally written to be useful for implementors, not for reviewers. In particular, while great care is generally taken to provide a complete description of the state machines and bits on the wire, this level of detail tends to get in the way of initial understanding. This document describes an approach for providing protocol "models" that allow reviewers to quickly grasp the essence of a system.

1. Introduction

The IETF process depends on peer review. However, in many cases, the documents submitted for publication are extremely difficult to review. Since reviewers have only limited amounts of time, this leads to extremely long review times, inadequate reviews, or both. In my view, a large part of the problem is that most documents fail to present an architectural model for how the protocol operated, opting instead to siply describe the protocol and let the reviewer figure it out.

This is acceptable when documenting a protocol for implementors, because they need to understand the protocol in any case, but dramatically increases the strain on reviewers. Reviewers necessarily need to get the big picture of the system and then focus on particular points. They simply do not have time to give the entire document the attention an implementor would.

One way to reduce this load is to present the reviewer with a MODEL--a short description of the system in overview form. This provides the reviewer with the context to identify the important or difficult pieces of the system and focus on them for review. As a side benefit, if the model is done first, it can be serve as an aid to the detailed protocol design and a focus for early review prior to protocol completion. The intention is that the model would either be the first section of the protocol document or be a separate document provided with the protocol.

2. The Purpose of a Protocol Model

A protocol model needs to answer three basic questions:

1. What problem is the protocol trying to achieve?
2. What messages are being transmitted and what do they mean?
3. What are the important but un-obvious features of the protocol?

The basic idea is to provide enough information that the reader could design a protocol which was roughly isomorphic to the protocol being described. This doesn't, of course, mean that the protocol would be identical, but merely that it would share most important features. For instance, the decision to use a KDC-based authentication model is an essential feature of Kerberos [KERBEROS]. By constrast, the use of ASN.1 is a simple implementation decision. S-expressions--or XML, had it existed at the time--would have served equally well.

3. Basic Principles

In this section we discuss basic principles that should guide your presentation.

3.1. Less is more

Humans are only capable of keeping a very small number of pieces of information in their head at once. Since we're interested in ensuring

that people get the big picture, we therefore have to dispense with a lot of detail. That's good, not bad. The simpler you can make things

the better.

3.2. Abstraction is good

A key technique for representing complex systems is to try to abstract away pieces. For instance, maps are better than photographs for finding out where you want to go because they provide an abstract, stylized, view of the information you're interested in. Don't be afraid to compress multiple protocol elements into a single abstract piece for pedagogical purposes.

3.3. A few well-chosen detail sometimes helps

The converse of the the previous principle is that sometimes details help to bring a description into focus. Many people work better when given examples. Thus, it's often a good approach to talk about the material in the abstract and then provide a concrete description of one specific piece to bring it into focus. Authors should focus on the normal path. Error cases and corner cases should only be discussed where they help illustrate some important point.

4. Writing Protocol Models

Our experience indicates that it's easiest to grasp protocol models when they're presented in visual form. We recommend a presentation format that is centered around a few key diagrams with explanatory text for each. These diagrams should be simple and typically consist of "boxes and arrows"--boxes representing the major components, arrows representing their relationships and labels indicating important features.

We recommend a presentation structured in three parts to match the three questions mentioned in the previous sections. Each part should contain 1-3 diagrams intended to illustrate the relevant points.

4.1. Describe the problem you're trying to solve

> First, figure out what you are trying to do (this is good advice under most circumstances, and it is especially apropos here.
> --NNTP Installation Guide

The absolutely most critical task that a protocol model must perform

is to explain what the protocol is trying to achieve. This provides crucial context for understanding how the protocol works and whether it meets its goals. Given the desired goals, in most cases an experienced reviewer will have an idea of how they would approach the problem and be able to compare that to the approach taken by the protocol

under review.

The "Problem" section of the model should start out with a short statement of the environments in which the protocol is expected to be used. This section should describe the relevant entities and the likely scenarios under which they participate in the protocol. The Problem section should feature a diagram showing the major communicating parties and their inter-relationships. It is particularly important to lay out the trust relationships between the various parties as these are often un-obvious.

4.1.1. Example: STUN (RFC 3489)

Network Address Translation (NAT) makes it difficult to run a number of classes of service from behind the NAT gateway. This is a particular problem when protocols need to advertise address/port pairs as part of the application layer protocol. Although the NAT can be configured to accept data destined for that port, address translation means that the address that the application knows about is not the same as the one that it is reachable on.

Consider the scenario represented in the figure below. A SIP client is initiating a session with a SIP server in which it wants the SIP server to send it some media. In its Session Description Protocol (SDP) [SDP] request it provides the IP and port on which it is listening. However, unbeknownst to the client, a NAT is in the way. It translates the IP address in the header, but unless it is SIP aware, it doesn't change the address in the request. The result is that the media goes into a black hole.

```
                      +-----------+
                      |    SIP    |
                      |  Server   |
                      |           |
                      +-----------+
                           ^
                           | [FROM: 198.203.2.1:8954]
                           | [MSG: SEND MEDIA TO 10.0.10.5:6791]
                           |
                           |
                      +-----------+
                      |           |
                      |    NAT    |
       ---------------+  Gateway  +----------------
                      |           |
                      +-----------+
                           ^
                           | [FROM: 10.0.10.5:6791]
                           | [MSG: SEND MEDIA TO 10.0.10.5:6791]
                           |
                        10.0.10.5
                      +-----------+
                      |    SIP    |
                      |  Client   |
                      |           |
                      +-----------+
```

The purpose of STUN [STUN] is to allow clients to detect this situation and determine the address mapping. They can then place the appropriate address in their application-level messages. This is done by making use of an external STUN server. That server is able to determine the translated address and tell the STUN client, as shown below.

```
                               +-----------+
                               |   STUN    |
                               |  Server   |
                               |           |
                               +-----------+
                                 ^      |
[IP HDR FROM: 198.203.2.1:8954]  |      | [IP HDR TO: 198.203.2.1:8954]
[MSG: WHAT IS MY ADDRESS?] |      | [MSG: YOU ARE 198.203.2.1:8954]
                                 |      v
                               +-----------+
                               |           |
                               |    NAT    |
                 --------------+  Gateway  +----------------
                               |           |
                               +-----------+
                                 ^      |
[IP HDR FROM: 10.0.10.5:6791]    |      | [IP HDR TO: 10.0.10.5:6791]
[MSG: WHAT IS MY ADDRESS?] |      | [MSG: YOU ARE 198.203.2.1:8954]
                                 |      v
                                 10.0.10.5
                               +-----------+
                               |    SIP    |
                               |  Client   |
                               |           |
                               +-----------+
```

4.2. Describe the protocol in broad overview

Once you've described the problem, the next task is to describe the protocol in broad overview. This means showing, either in "ladder

diagram" or "boxes and arrows" form, the protocol messages that flow between the various networking agents. This diagram should be accompied with explanatory text that describes the purpose of each message and the MAJOR data elements.

This section SHOULD NOT contain detailed descriptions of the protocol messages or of each data element. In particular, bit diagrams, ASN.1 modules and XML schema SHOULD NOT be shown. The purpose of this section is explicitly not to provide a complete description of the protocol. Instead, it is to provide enough of a map so that a person reading the full protocol document can see where each specific piece fits.

4.3. State Machines

In certain cases, it may be helpful to provide a state machine description of the behavior of network elements. However, such state machines should be kept as minimal as possible. Remember that the purpose is to promote high-level comprehension, not complete understanding.

4.4. Example: DCCP

Although DCCP [DCCP] is datagram oriented like UDP, it is stateful like TCP. Connections go through the following phases:
1. Initiation
2. Feature negotiation
3. Data transfer
4. Termination

4.4.1. Initiation

As with TCP, the initiation phase of DCCP involves a three-way handshake, shown in Figure 1.

```
Client                                            Server
------                                            ------
DCCP-Request            ->
[Ports, Service,
Features]
                        <-              DCCP-Response
                                           [Features,
                                              Cookie]
```

```
DCCP-Ack                ->
[Features,
Cookie]
```

FFiigguurree 11 DCCP 3-way handshake

In the DCCP-Request message, the client tells the server the name of the service it wants to talk to and the ports it wants to communicate on. Note that ports are not tightly bound to services the way they are in TCP or UDP common practice. It also starts feature negotiation. For pedagogical reasons, we will present feature negotiation separately in the next section. However, realize that the early phases of feature negotiation happen concurrently with initiation.

In the DCCP-Response message, the server tells the client that it is willing to accept the connection and continues feature negotiation. In order to prevent SYN-flood style DOS attacks, DCCP incorporates an IKE-style cookie exchange. The server can provide the client with a cookie that contains all the negotiation state. This cookie must be

echoed by the client in the DCCP-Ack, thus removing the need for the server to keep state.

In the DCCP-Ack message, the client acknowledges the DCCP-Response and returns the cookie to permit the server to compleye its side of the connection. As indicated in Figure 1, this message may also include feature negotiation messages.

4.4.2. Feature Negotiation

In DCCP, feature negotiation is performed by attaching options to other DCCP packets. Thus feature negotiation can be piggybacked on any other DCCP message. This allows feature negotiation during connection initiation as well as feature renegotiation during data flow.

Somewhat unusually, DCCP features are one-sided. Thus, it's possible to have a different congestion control regime for data sent from client to server than from server to client.

Feature negotiation is done with three options:

1. Change
2. Prefer
3. Confirm

A Change message says to the peer "change this option setting on your side". The peer can either respond with a Confirm, meaning "I've changed it" or a Prefer, containing a list of other settings that the peer would like. Multiple exchanges of Change and Prefer may occur as the peers attempt so sort out what options they have in common. Some sample exchanges (partly cribbed from the DCCP spec) follow:

```
Client                                            Server
------                                            ------
Change(CC,2)               ->
                           <-              Confirm(CC,2)
```

In this exchange, the peers agree to set CC equal to 2.

```
Client                                            Server
------                                            ------
Change(CC,3,4)              ->
                            <-          Prefer(CC,1,2,5)
Change(CC,5)                ->
                            <-              Confirm(CC,5)
```

In this exchange, the client requests CC values 3 and 4. Note that the client can offer multiple values. The server doesn't like any of

these and offers 1, 2, and 5. The client chooses 5 and the server agrees.

Since features are one-sided, if a party wants to change one of his own options, he must ask the peer to issue a Change. This is done using a Prefer, as shown below, where the client gets the server to request that the client change the value of CC to 3.

```
Client                                            Server
------                                            ------
Prefer(CC,3,4)              ->
                            <-              Change(CC,3)
Confirm(CC,3)               ->
```

4.4.3. Data Transfer

Rather than have a single congestion control regime as in TCP, DCCP offers a variety of negotiable congestion control regimes. The DCCP documents describe two congestion control regimes: additive increase, multiplicative decrease (CCID-2 [CCID2]) and TCP-friendly rate control (CCID-3 [CCID3]). CCID-2 is intended for applications which want

maximum throughput. CCID-3 is intended for real-time applications which want smooth response to congestion.

4.4.3.1. CCID-2

CCID-2's congestion control is extremely similar to that of TCP. The sender maintains a congestion window and sends packets until that window is full. Packets are Acked by the receiver. Dropped packets and ECN [ECN] are used to indicate congestion. The response to congestion is to halve the congestion window. One subtle diference between DCCP and TCP is that the Acks in DCCP must contain the sequence numbers of all received packets (within a given window) not just the highest sequence number as in TCP.

4.4.3.2. CCID-3

CCID-3 is an equation based form of rate control which is intended to provide smoother response to congestion than CCID-2. The sender maintains a "transmit rate". The receiver sends ACK packets which also contain information about the receiver's estimate of packet loss. The sender uses this information to update its transmit rate. Although CCID-3 behaves somewhat differently from TCP in its short term congestion response, it is designed to operate fairly with TCP over the long term.

4.4.4. Termination

Connection termination in DCCP is initiated by sending a Close message. Either side can send a Close message. The peer then responds with a Reset message, at which point the connection is closed. The side that sent the Close message must quietly preserve the socket in TIMEWAIT state for 2MSL.

```
Client                                      Server
------                                      ------
Close                    ->
                         <-                  Reset
[Remains in TIMEWAIT]
```

Note that the server may wish to close the connection but not remain in TIMEWAIT (e.g., due to a desire to minimize server-side state.) In order to accomplish this, the server can elicit a Close from the client by sending a CloseReq message and thus keeping the TIMEWAIT

state on the client.

5. Describe any important protocol features

The final section (if there is one) should contain an explanation of any important protocol features which are not obvious from the previous sections. In the best case, all the important features of the protocol would be obvious from the message flow. However, this isn't always the case. This section is an opportunity for the author to explicate those features. Authors should think carefully before writing this section. If there are no important points to be made they should not populate this section.

Examples of the kind of feature that belongs in this section include: high-level security considerations, congestion control information and overviews of the algorithms that the network elements are intended to follow. For instance, if you have a routing protocol you might use this section to sketch out the algorithm that the router uses to determine the appropriate routes from protocol messages.

5.1. Example: WebDAV COPY and MOVE

WebDAV [WEBDAV] includes both a COPY method and a MOVE method. While a MOVE can be thought of as a COPY followed by DELETE, COPY+DELETE and MOVE aren't entirely equivalent.

The use of COPY+DELETE as a MOVE substitute is problematic because of the creation of the intermediate file. Consider the case where the user is approaching some quota boundary. A COPY+DELETE should be forbidden because it would temporarily exceed the quota. However, a

simple rename should work in this situation.

The second issue is permissions. The WebDAV permissions model allows the server to grant users permission to rename files but not to create new ones--this is unusual in ordinary filesystems but nothing prevents it in WebDAV. This is clearly not possible if a client uses COPY+DELETE to do a MOVE.

Finally, a COPY+DELETE does not produce the same logical result as would be expected with a MOVE. Because COPY creates a new resource, it is permitted (but not required) to use the time of new file creation as the creation date property. By contrast, the expectation for move is that the renamed file will have the same properties as the original.

6. Formatting Issues

The requirement that Internet-Drafts and RFCs be renderable in ASCII is a significant obstacle when writing the sort of graphics-heavy document being described here. Authors may find it more convenient to do a separate protocol model document in Postscript or PDF and simply make it available at review time--though an archival version would certainly be handy.

7. A Complete Example: Internet Key Exchange (IKE)

7.1. Operating Environment

Internet key Exchange (IKE) [IKE] is a key establishment and parameter negotiation protocol for Internet protocols. Its primary application is for establishing security associations (SAs) [IPSEC] for IPsec AH [AH] and ESP [ESP].


```
+---------------------+                       +---------------------+
|                     |                       |                     |
|   +-------------+   |                       |   +-------------+   |
|   |     Key     |   |          IKE          |   |     Key     |   |
|   | Management  | <-+-----------------------+-> | Management  |   |
|   |   Process   |   |                       |   |   Process   |   |
|   +-------------+   |                       |   +-------------+   |
|          ^          |                       |          ^          |
|          |          |                       |          |          |
|          v          |                       |          v          |
|   +-------------+   |                       |   +-------------+   |
```

```
   |    |   IPsec     |   |          AH/ESP         |   |   IPsec     |   |
   |    |   Stack     | <-+-------------------------+-> |   Stack     |   |
   |    |             |   |                         |   |             |   |
   |    +-------------+   |                         |   +-------------+   |
   |                      |                         |                     |
   |                      |                         |                     |
   |     Initiator        |                         |     Responder       |
   +----------------------+                         +---------------------+
```

The general deployment model for IKE is shown in Figure 2. The IPsec engines and IKE engines typically are separate modules. When a packet needs to be processed (either sent or received) for which no security association exists, the IPsec engine contacts the IKE engine and asks it to establish an appropriate SA. The IKE engine contacts the appropriate peer and uses IKE to establish the SA. Once the IKE handshake is finished it registers the SA with the IPsec engine.

In addition, IKE traffic between the peers can be used to refresh keying material or adjust operating parameters such as algorithms.

7.1.1. Initiator and Responder

Although IPsec is basically symmetrical, IKE is not. The party who sends the first message is called the INITIATOR. The other party is called the RESPONDER. In the case of TCP connections the INITIATOR will typically be the peer doing the active open (i.e. the client).

7.1.2. Perfect Forward Secrecy

One of the major concerns in IKE design was that traffic be protected even if they keying material of the nodes was later compromised, provided that the session in question had terminated and so the session-specific keying material was gone. This property is often called PERFECT FORWARD SECRECY (PFS) or BACK TRAFFIC PROTECTION.

7.1.3. Denial of Service Resistance

Since IKE allows arbitrary peers to initiate computationally expensive cryptographic operations, it potentially allows resource consumption denial of service attacks to be mounted against the IKE engine. IKE includes countermeasures designed to minimize this risk.

7.1.4. Keying Assumptions

Because Security Associations are essentially symmetric, both sides must in general be authenticated. Because IKE needs to be able to establish SAs between a broad range of peers with various kinds of prior relationships, IKE supports a very flexible keying model. Peers can authenticate via shared keys, digital signatures (typically from keys vouched for by certificates), or encryption keys.

7.1.5. Identity Protection

Although IKE requires the peers to authenticate to each other, it was considered desirable by the working group to provide some identity protection for the communicating peers. In particular, the peers should be able to hide their identity from passive observers and one peer should be able to require the author to authenticate before they self-identity. In this case, the designers chose to make the party who speaks first (the INITIATOR) identify first.

7.2. Protocol Overview

At a very high level, there are two kinds of IKE handshake:
  (1) Those which establish an IKE security association.
  (2) Those which establish an AH or ESP security association.

When two peers which have never communicated before need to establish an AH/ESH SA, they must first establish an IKE SA. This allows them to exchange an arbitrary amount of protected IKE traffic. They can then use that SA to do a second handshake to establish SAs for AH and ESP. This process is shown in schematic form below. The notation E(SA,XXXX) is used to indicate that traffic is encrypted under a given SA.


```
Initiator                                   Responder
---------                                   ---------

Handshake MSG            ->                           \
```

```
                          <-          Handshake MSG   \ Establish IKE
                                                      / SA (IKEsa)
                         [...] /


   E(IKEsa, Handshake MSG)  ->                       \  Establish AH/ESP
                            <-  E(IKEsa, Handshake MSG)  /  SA
```

IKE terminology is somewhat confusing, referring under different circumstances to "phases" and "modes". For maximal clarity we will refer to the the Establishment of the IKE SA as "Stage 1" and the Establishment of AH/ESP SAs as "Stage 2". Note that it's quite possible for there to be more than one Stage 2 handshake, once Stage 1 has been finished. This might be useful if you wanted to establish multiple AH/ESP SAs with different cryptographic properties.

The Stage 1 and Stage 2 handshakes are actually rather different, because the Stage 2 handshake can of course assume that its traffic is being protected with an IKE SA. Accordingly, we will first discuss Stage 1 and then Stage 2.

7.2.1. Stage 1

There are a large number of variants of the IKE Stage 1 handshake, necessitated by use of different authentication mechanisms. However, broadly speaking they fall into one of two basic categories: MAIN MODE, which provides identity protection and DoS resistance, and AGGRESSIVE MODE, which does not. We will cover MAIN MODE first.

7.2.1.1. Main Mode

Main Mode is a six message (3 round trip) handshake which offers identity protection and DoS resistance. An overview of the handshake is below.


```
   Initiator                                       Responder
```

```
---------                                    ---------
CookieI, Algorithms      ->                          \  Parameter
                         <-       CookieR, Algorithms /  Establishment

CookieR,
Nonce, Key Exchange      ->
                         <-        Nonce, Key Exchange\  Establish
                                                      /  Shared key

E(IKEsa, Auth Data)      ->
                         <-        E(IKEsa, Auth data)\  Authenticate
                                                      /      Peers
```

In the first round trip, the Initiator offers a set of algorithms and parameters. The Responder picks out the single set that it likes and responds with that set. It also provides CookieR, which will be used to prevent DoS attacks. At this point, there is no secure association but the peers have tentatively agreed upon parameters. These parameters include a Diffie-Hellman group, which will be used in the second round trip.

In the second round trip, the Initiator sends the key exchange information. This generally consists of the Initiator's Diffie-Hellman public share (Yi). He also supplies CookieR, which was provided by the responder. The Responder replies with his own DH share (Yr). At this point, both Initiator and Responder can compute the shared DH key (ZZ). However, there has been no authentication and so they don't know with any certainty that the connection hasn't been attacked. Note that as long as the peers generate fresh DH shares for each handshake than PFS will be provided.

Before we move on, let's take a look at the cookie exchange. The basic anti-DoS measure used by IKE is to force the peer to demonstrate that they can receive traffic from you. This foils blind attacks like SYN floods [SYNFLOOD] and also makes it somewhat easier to track down attackers. The cookie exchange serves this role in IKE. The Responder can verify that the Initiator supplied a valid CookieR before doing the expensive DH key agreement. This does not totally eliminate DoS attacks, since an attacker who was willing to reveal his location could still consume server resources, but it does protect against a certain class of blind attack.

In the final round trip, the peers establish their identities. Since they share an (unauthenticated) key, they can send their identities encrypted, thus providing identity protection from eavesdroppers. The exact method of proving identity depends on what form of credential is being used (signing key, encryption key, shared secret, etc.), but

in general you can think of it as a signature over some subset of the handshake messages. So, each side would supply its certificate and then sign using the key associated with that certificate. If shared keys are used, the authentication data would be a key id and a MAC. Authentication using public key encryption follows similar principles but is more complicated. Refer to the IKE document for more details.

At the end of the Main Mode handshake, the peers share:
(1) A set of algorithms for encryption of further IKE traffic.
(2) Traffic encryption and authentication keys.
(3) Mutual knowledge of the peer's identity.

7.2.1.2. Aggressive Mode

Although IKE Main Mode provides the required services, there was concern that the large number of round trips required added excessive latency. Accordingly, an Aggressive Mode was defined. Aggressive mode packs more data into fewer messages and thus reduces latency. However, it does not provide protection against DoS or identity protection.

```
Initiator                                          Responder
---------                                          ---------
Algorithms, Nonce,
Key Exchange,                ->
                             <-          Algorithms, Nonce,
                                   Key Exchange, Auth Data
Auth Data                    ->
```

After the first round trip, the peers have all the required properties except that the Initiator has not authenticated to the Responder. The third message closes the loop by authenticating the Initiator. Note that since the authentication data is sent in the clear, no identity protection is provided and since the Responder does the DH key agreement without a round trip to the Initiator, there is no DoS protection

7.2.2. Stage 2

Stage 1 on its own isn't very useful. The purpose of IKE, after all, is to establish associations to be used to protect other traffic, not just to establish IKE SAs. Stage 2 (what IKE calls "Quick Mode") is used for this purpose. The basic Stage 2 handshake is shown below.

```
      Initiator                                      Responder
      ---------                                      ---------
      AH/ESP parameters,
      Algorithms, Nonce,
      Handshake Hash           ->

                                <-           AH/ESP parameters,
                                             Algorithms, Nonce,
                                                 Handshake Hash
      Handshake Hash            ->
```

As with quick mode, the first two messages establish the algorithms and parameters while the final message is a check over the previous messages. In this case, the parameters also include the transforms to be applied to the traffic (AH or ESP) and the kinds of traffic which are to be protected. Note that there is no key exchange information shown in these messages.

In this version of Quick Mode, the peers use the pre-existing Stage 1 keying material to derive fresh keying material for traffic protection (with the nonces to ensure freshness). Quick mode also allows for a new Diffie-Hellman handshake for per-traffic key PFS. In that case, the first two messages shown above would also include Key Exchange payloads, as shown below.

```
      Initiator                                      Responder
      ---------                                      ---------
      AH/ESP parameters,
      Algorithms, Nonce,
      Key Exchange,             ->
      Handshake Hash

                                <-           AH/ESP parameters,
                                             Algorithms, Nonce,
                                                  Key Exchange,
                                                 Handshake Hash
      Handshake Hash            ->
```

7.3. Other Considerations

There are a number of features of IKE that deserve special consideration. These are discussed here.

7.3.1. Cookie Generation

As mentioned previously, IKE uses cookies as a partial defense

against DoS attacks. When the responder receives Main Mode message 3 containing the Key Exchange data and the cookie, it verifies that the

cookie is correct. However, this verification must not involve having a list of valid cookies. Otherwise, an attacker could potentially consume arbitrary amounts of memory by repeatedly requesting cookies from a responder. The recommended way to generate a cookie, suggested by Phil Karn, is by having a single master key and compute a hash of the secret and the initiator's address information. This cookie can be verified by recomputing the cookie value based on information in the third message and seeing if it matches.

7.3.2. Endpoint Identities

So far we have been rather vague about what sorts of endpoint identities are used. In principle, there are three ways a peer might be identified: by a shared key, a pre-configured public key, and a certificate.

7.3.2.1. Shared Key

In a shared key scheme, the peers share some symmetric key. This key is associated with a key identifier which is known to both parties. It is assumed that the party verifying that identity also has some sort of table that indicates what sorts of traffic (e.g. what addresses) that identity is allowed to negotiate SAs for.

7.3.2.2. Pre-configured public key

A pre-configured public key scheme is the same as a shared key scheme except that the verifying party has the authenticating party's public key instead of a shared key.

7.3.2.3. Certificate

In a certificate scheme, authenticating party presents a certificate containing their public key. It's straightforward to establish that that certificate matches the authentication data provided by the peer. What's less straightforward is to determine whether a given peer is entitled to negotiate for a given class of traffic. In theory, one might be able to determine this from the name in the certificate (e.g. the subject name contains an IP address that matches the ostensible IP address). In practice, this is not clearly specified in IKE and therefore not really interoperable. The more likely case at the moment is that there is a configuration table map-

ping certificates to policies, as with the other two authentication schemes.

References

[AH] Kent, S., and Atkinson, R., "IP Authentication Header", RFC 2402, November 1998.

[CCID2] Floyd, S., Kohler, E., "Profile for DCCP Congestion Control ID 2: TCP-like Congestion Control", draft-ietf-dccp-ccid2-04.txt, October 2003.

[CCID3] Floyd, S., Kohler, E., Padhye, J. "Profile for DCCP Congestion Control ID 3: TFRC Congestion Control", draft-ietf-dccp-ccid3-05.txt, February 2004.

[DCCP] Kohler, E., Handley, M., Floyd, S., "Datagram Congestion Control Protocol (DCCP)", draft-ietf-dccp-spec-06.txt, February 2004.

[ECN] Ramakrishnan, K. Floyd, S., Black D., "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[ESP] Kent, S., and Atkinson, R., "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[IKE] Harkins, D., Carrel, D., "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[IPSEC] Kent, S., Atkinson, R., "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[KERBEROS] Kohl, J., Neuman, C., "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

[SDP] Handley, M., Jacobson, V., "SDP: Session Description Protocol" RFC 2327, April 1998.

[STUN] Rosenberg, J., Weinberger, J., Huitema, C., Mahy, R., "STUN - Simple Traversal of User Datagram Protocol (UDP)", RFC 3489, March 2003.

[WEBDAV] Goland, Y., Whitehead, E., Faizi, A., Carter, S., Jensen, D. "HTTP Extensions for Distributed Authoring -- WEBDAV", RFC 2518, February 1999.

Security Considerations

This document does not define any protocols and therefore has no security considerations.

Author's Address
Eric Rescorla <ekr@rtfm.com>
RTFM, Inc.

2064 Edgewood Drive
Palo Alto, CA 94303
Phone: (650)-320-8549

Internet Architecture Board <iab@iab.org>
IAB

Appendix A. IAB Members at the time of this writing

Bernard Aboba
Harald Alvestrand
Rob Austein
Leslie Daigle
Patrik Falstrom
Sally Floyd
Jun-ichiro Itojun Hagino
Mark Handley
Bob Hinden
Geoff Huston
Eric Rescorla
Pete Resnick
Jonathon Rosenberg