RAP Working Group
Internet Draft
Expires December 2001

R. Hess
Intel

June 2001

Cryptographic Authentication for RSVP POLICY_DATA Objects

draft-ietf-rap-auth-policy-data-00.txt

Status of this Memo

Copyright Notice

Abstract

This document describes the format and use of the INTEGRITY option within RSVP's POLICY_DATA object to provide integrity and authentication of POLICY_DATA objects within RSVP messages.

1. Introduction

The Resource ReSerVation Protocol (RSVP) [1] is a protocol for setting up distributed state in routers and hosts, and in particular for reserving resources to implement integrated service. RSVP allows particular users to obtain preferential access to network resources, under the control of an admission control mechanism. Permission to make a reservation will depend both upon the availability of the requested resources along the path of the data, and upon satisfaction of policy rules.

Policy based admission control will occur at Policy Enforcement Points (PEPs); for the purposes of this document these nodes are policy aware RSVP systems. Policy data are distributed among PEPs using POLICY_DATA objects in RSVP messages. Initially, the enforcement of policy rules may concentrate on border nodes between autonomous systems. As such, POLICY_DATA objects may traverse policy ignorant RSVP systems (PINs) whose capabilities are limited to default policy handling [2].

To ensure the integrity of this policy based admission control mechanism, PEPs require the ability to protect their POLICY_DATA objects against corruption and spoofing. The RSVP integrity mechanism [3] works hop-by-hop, which, unfortunately, is insufficient for our needs as it places trust with the POLICY_DATA object in PINs. What is required is an integrity mechanism analogous to RSVP's, but one what works PEP peer to PEP peer. This document defines such a mechanism. The proposed scheme transmits an authenticating digest of the POLICY_DATA object, computed using a secret Authentication Key and a keyed-hash algorithm. This scheme provides protection against forgery or object modification. The INTEGRITY option of each POLICY_DATA object is tagged with a one-time-use sequence number. This allows the message receiver to identify playbacks and hence to thwart replay attacks. The proposed mechanism does not afford confidentiality, since messages stay in the clear; however, the mechanism is also exportable from most countries, which would be impossible were a privacy algorithm to be used. Note: this document uses the terms "sender" and "receiver" differently from [3]. They are used here to refer to policy aware RSVP systems (a.k.a. PEPs) that face each other either across an RSVP hop or through one or more PINs, the "sender" being the system generating POLICY_DATA objects.

The message replay prevention algorithm is quite simple. The sender generates packets with monotonically increasing sequence numbers. In turn, the receiver only accepts packets that have a larger sequence number than the previous packet. To start this process, a receiver handshakes with the sender to get an initial sequence number. This memo discusses ways to relax the strictness of the in-order delivery of messages as well as techniques to generate monotonically increasing sequence numbers that are robust across sender failures and restarts.

The proposed mechanism is independent of a specific cryptographic algorithm, but this document describes the use of Keyed-Hashing for Message Authentication using HMAC-MD5 [4]. As noted in [4], there exist stronger hashes, such as HMAC-SHA1; where warranted, implementations will do well to make them available. However, in the general case, [4] suggests that HMAC-MD5 is adequate to the purpose at hand and has preferable performance characteristics. [4] also offers source code and test vectors for this algorithm, a boon to those who would test for interoperability. HMAC-MD5 is required as a baseline to be universally included in policy aware RSVP implementations providing cryptographic authentication, with other proposals optional (see Section 6 on Conformance Requirements).

1.1. Conventions used in this Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [5].

1.2. Why not use the Standard IPsec Authentication Header?

One obvious question is why, since there exists a standard authentication mechanism, IPsec [6,7], we would choose not to use it. The use of IPsec was rejected for the following reasons.

The security associations in IPsec are based on destination address. It is not clear that POLICY_DATA objects are well defined for either source or destination based security associations, as a router must forward PATH and PATH TEAR messages using the same source address as the sender listed in the SENDER TEMPLATE. RSVP traffic may otherwise not follow exactly the same path as data traffic. Using either source or destination based associations would require opening a new security association among the routers for which a reservation

traverses.

In addition, it was noted that neighbor relationships between PEPs are not limited to those that face one another across a communication channel. POLICY_DATA objects may traverse PINs, which are not necessarily visible to the sending system. These arguments suggest the use of a key management strategy based on PEP to PEP associations instead of IPsec.

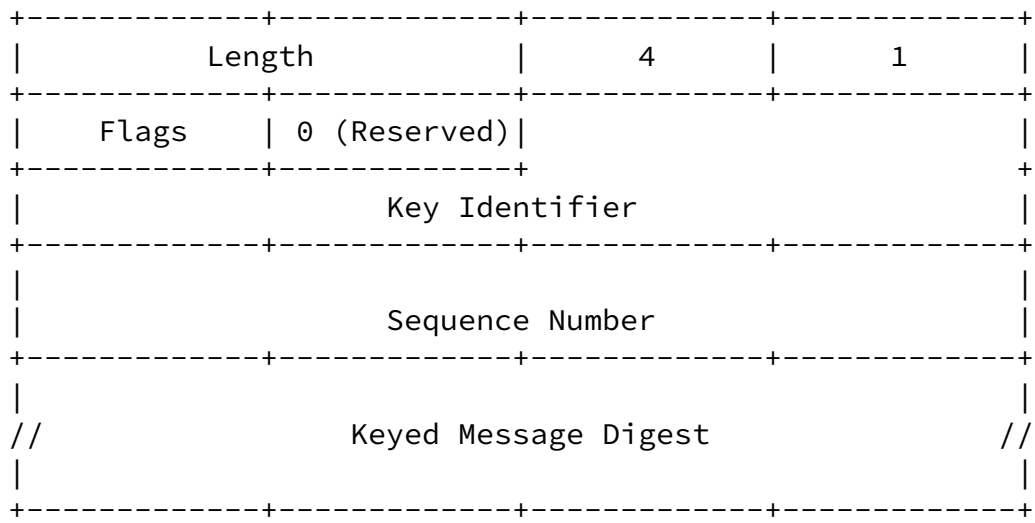
2. Data Structures

2.1. INTEGRITY Option Format

The Options List of a POLICY_DATA object consists of a sequence of "objects," which are type-length-value encoded fields having specific purposes. The information required for PEP peer to PEP peer integrity checking is carried in an INTEGRITY option. The same INTEGRITY option type is used for both IPv4 and IPv6.

The INTEGRITY Option format is defined to be identical to RSVP's INTEGRITY object as defined in [8], Section 2.1. For clarity, the format is reproduced below.

o Keyed Message Digest INTEGRITY Option: Class = 4, C-Type = 1

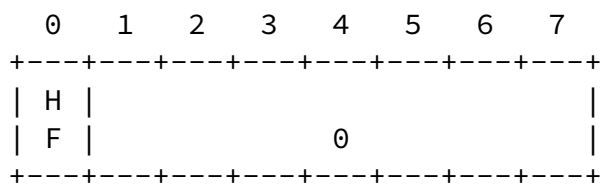


Length: 16 bits

The total length of the INTEGRITY Option in octets. Must always be a multiple of 4.

Flags: An 8-bit field with the following format:

Flags



Currently only one flag (HF) is defined. The remaining flags are reserved for future use and MUST be set to 0.

- Bit 0: Handshake Flag (HF) concerns the integrity handshake mechanism (Section 4.3). POLICY_DATA object senders willing to respond to integrity handshake messages SHOULD set this flag to 1 whereas those that will reject integrity handshake messages SHOULD set this to 0.

Reserved: 8 bits

Unused at this time. This field MUST be set to 0.

Key Identifier: 48 bits

An unsigned 48-bit number that MUST be unique for a given sender. Locally unique Key Identifiers can be generated using some combination of the address (IP or MAC or LIH) of the sending interface and the key number. The combination of the Key Identifier and the sending system's IP address uniquely identifies the security association (Section 2.2).

Sequence Number: 64 bits

An unsigned monotonically increasing, unique sequence number.

Sequence Number values may be any monotonically increasing sequence that provides the INTEGRITY option (of each POLICY_DATA object) with a tag that is unique for the associated key's lifetime. Details on sequence number generation are presented in Section 3.

Keyed Message Digest: Variable length

The digest MUST be a multiple of 4 octets long. For HMAC-MD5, it will be 16 octets long.

2.2. Security Association

The sending and receiving systems maintain a security association for each authentication key that they share. This security association includes the following parameters:

- Authentication algorithm and algorithm mode being used.
- Key used with the authentication algorithm.
- Lifetime of the key.
- Associated sending interface and other security association selection criteria [REQUIRED at Sending System].
- Source Address of the sending system [REQUIRED at Receiving System].
- Latest sending sequence number used with this key identifier [REQUIRED at Sending System].
- List of last N sequence numbers received with this key identifier [REQUIRED at Receiving System].

3. Generating Sequence Numbers

In this section we describe methods that could be chosen to generate

the sequence numbers used in the INTEGRITY option of a POLICY_DATA object in a RSVP message. As previous stated, there are two important properties that MUST be satisfied by the generation procedure. The first property is that the sequence numbers are unique, or one-time, for the lifetime of the integrity key that is in current use. A receiver can use this property to unambiguously distinguish between a new or a replayed object. The second property is that the sequence numbers are generated in monotonically increasing order, modulo 2^64. This is required to greatly reduce the amount of saved state, since a receiver only needs to save the value of the highest sequence number seen to avoid a replay attack. Since the starting sequence number might be arbitrarily large, the modulo operation is required to accommodate sequence number roll-over within some key's lifetime. This solution draws from TCP's approach [9].

The sequence number field is chosen to be a 64-bit unsigned quantity. This is large enough to avoid exhaustion over the key lifetime. For example, if a key lifetime was conservatively defined as one year, there would be enough sequence number values to send POLICY_DATA objects at an average rate of about 585 gigaObjects per second. A 32-bit sequence number would limit this average rate to about 136 objects per second.

The ability to generate unique monotonically increasing sequence numbers across a failure and restart implies some form of stable storage, either local to the device or remotely over the network. Three sequence number generation procedures are described below.

3.1. Simple Sequence Numbers

The most straightforward approach is to generate a unique sequence number using an object counter. Each time a POLICY_DATA object is transmitted for a given key, the sequence number counter is incremented. The current value of this counter is continually or periodically saved to stable storage. After a restart, the counter is recovered using this stable storage. If the counter was saved periodically to stable storage, the count should be recovered by increasing the saved value to be larger than any possible value of the counter at the time of the failure. This can be computed, knowing the interval at which the counter was saved to stable storage and incrementing the stored value by that amount.

3.2. Sequence Numbers Based on a Real Time Clock

Most devices will probably not have the capability to save sequence number counters to stable storage for each key. A more universal solution is to base sequence numbers on the stable storage of a real time clock. Many computing devices have a real time clock module

that includes stable storage of the clock. These modules generally include some form of nonvolatile memory to retain clock information in the event of a power failure.

In this approach, we could use an NTP based timestamp value as the sequence number. The roll-over period of a NTP timestamp is about 136 years, much longer than any reasonable lifetime of a key. In addition, the granularity of the NTP timestamp is fine enough to allow the generation of a POLICY_DATA object every 200 picoseconds for a given key. Many real time clock modules do not have the resolution of an NTP timestamp. In these cases, the least significant bits of the timestamp can be generated using an object counter, which is reset every clock tick. For example, when the real time clock provides a resolution of 1 second, the 32 least significant bits of the sequence number can be generated using an object counter. The remaining 32 bits are filled with the 32 least significant bits of the timestamp. Assuming that the recovery time after failure takes longer than one tick of the real time clock, the object counter for the low order bits can be safely reset to zero after a restart.

3.3. Sequence Numbers Based on a Network Recovered Clock

If the device does not contain any stable storage of sequence number counters or of a real time clock, it could recover the real time clock from the network using NTP. Once the clock has been recovered following a restart, the sequence number generation procedure would be identical to the procedure described above.

4. POLICY_DATA Object Processing

Implementations SHOULD allow specification of interfaces that are to be secured, for either sending objects, or receiving them, or both. The sender must ensure that all POLICY_DATA objects sent on secured sending interfaces include an INTEGRITY option, generated using the appropriate Key. Receivers verify whether POLICY_DATA objects, except of the type "Integrity Challenge" (Section 4.3), arriving on a secured receiving interface contain the INTEGRITY option. If the INTEGRITY option is absent, the receiver discards the object.

Security associations are simplex - the keys that a sending system uses to sign its objects may be different from the keys that its receivers use to sign theirs. Hence, each association is associated

with a unique sending system and (possibly) multiple receiving systems.

Each sender SHOULD have distinct security associations (and keys) per secured sending interface (or LIH). While administrators may configure all the routers and hosts on a subnet (or for that matter, in their network) using a single security association, implementations MUST assume that each sender may send using a distinct security association on each secured interface. At the

sender, security association selection is based on the interface through which the object is sent. This selection MAY include additional criteria, such as the destination address (when sending the object unicast, over a broadcast LAN with a large number of hosts) or user identities at the sender or receivers [10]. Finally, all intended object recipients should participate in this security association. Route flaps in a non RSVP cloud might cause objects for the same receiver to be sent on different interfaces at different times. In such cases, the receivers should participate in all possible security associations that may be selected for the interfaces through which the object might be sent.

Receivers select keys based on the Key Identifier and the sending system's IP address. The Key Identifier is included in the INTEGRITY option. The sending system's address can be obtained from the Originating RSVP_HOP option. Since the Key Identifier is unique for a sender, this method uniquely identifies the key.

The integrity mechanism slightly modifies the processing rules for POLICY_DATA objects, both when including the INTEGRITY option in a policy object sent over a secured sending interface and when accepting a policy object received on a secured receiving interface. These modifications are detailed below.

4.1. INTEGRITY Generation

For a POLICY_DATA object sent over a secured sending interface, the object is created as follows:

(1) The INTEGRITY option is inserted in the appropriate place, and its location in the POLICY_DATA object is remembered for later use.

(2) The sending interface and other appropriate criteria (as mentioned above) are used to determine the Authentication Key and the hash algorithm to be used.

(3) The unused flags and the reserved field in the INTEGRITY option MUST be set to 0. The Handshake Flag (HF) should be set according to rules specified in Section 2.1.

(4) The sending sequence number MUST be updated to ensure a unique, monotonically increasing number. It is then placed in the Sequence Number field of the INTEGRITY option.

(5) The Keyed Message Digest field is set to zero.

(6) The Key Identifier is placed into the INTEGRITY option.

4.2. INTEGRITY Reception

(7) A copy of the RSVP SESSION object is temporarily appended to the end of the POLICY_DATA object (for computational purposes only, without changing the length of the POLICY_DATA object). The flags field of the SESSION object is set to 0. This concatenation is considered as the message for which a digest is to be computed.

(8) An authenticating digest of the object is computed using the Authentication Key in conjunction with the keyed-hash algorithm. When the HMAC-MD5 algorithm is used, the hash calculation is described in [4]. Note: When the computation is complete, the SESSION object is ignored and is not part of the POLICY_DATA object.

(9) The digest is written into the Cryptographic Digest field of the INTEGRITY option.

When the policy object is received on a secured receiving interface, and is not of the type "Integrity Challenge", it is processed in the following manner:

(1) The Cryptographic Digest field of the INTEGRITY option is saved and the field is subsequently set to zero.

(2) A copy of the RSVP SESSION object is temporarily appended to the end of the POLICY_DATA object (for computational purposes only, without changing the length of the POLICY_DATA object). The flags field of the SESSION object is set to 0. This concatenation is considered as the message for which a digest is to be computed.

(3) The Key Identifier field and the sending system address are used to uniquely determine the Authentication Key and the hash algorithm to be used. Processing of this packet might be delayed when the Key Management System (Appendix 1) is queried for this information.

(4) A new keyed-digest is calculated using the indicated algorithm and the Authentication Key. Note: When the computation is complete, the SESSION object is ignored and is not part of the POLICY_DATA object.

(5) If the calculated digest does not match the received digest, the policy object is discarded without further processing.

(6) If the policy object is of type "Integrity Response", verify that the CHALLENGE option identically matches the originated challenge. If it matches, save the sequence number in the

INTEGRITY option as the largest sequence number received to date.

Otherwise, for all other policy objects, the sequence number is validated to prevent replay attacks, and messages with invalid sequence numbers are ignored by the receiver.

When a policy object is accepted, the sequence number of that object could update a stored value corresponding to the largest sequence number received to date. Each subsequent object must then have a larger (modulo 2^64) sequence number to be accepted. This simple processing rule prevents message replay attacks, but it must be modified to tolerate limited

out-of-order message delivery. For example, if several messages were sent in a burst (in a periodic refresh generated by a router, or as a result of a tear down function), they might get reordered and then the sequence numbers would not be received in an increasing order.

An implementation SHOULD allow administrative configuration that sets the receiver's tolerance to out-of-order message delivery. A simple approach would allow administrators to specify a message window corresponding to the worst case reordering behavior. For example, one might specify that packets reordered within a 32 message window would be accepted. If no reordering can occur, the window is set to one.

The receiver must store a list of all sequence numbers seen within the reordering window. A received sequence number is valid if (a) it is greater than the maximum sequence number received or (b) it is a past sequence number lying within the reordering window and not recorded in the list. Acceptance of a sequence number implies adding it to the list and removing a number from the lower end of the list. Policy objects received with sequence numbers lying below the lower end of the list or marked seen in the list are discarded.

When an "Integrity Challenge" policy object is received on a secured sending interface it is processed in the following manner:

(1) An "Integrity Response" policy object is formed using the Challenge option received in the challenge policy object.

(2) The response object is sent back to the receiver, based on the source IP address of the challenge policy object, using the "INTEGRITY Generation" steps outlined above. The selection of the Authentication Key and the hash algorithm to be used is determined by the key identifier supplied in the challenge policy object.

4.3. Integrity Handshake at Restart or Initialization of the Receiver

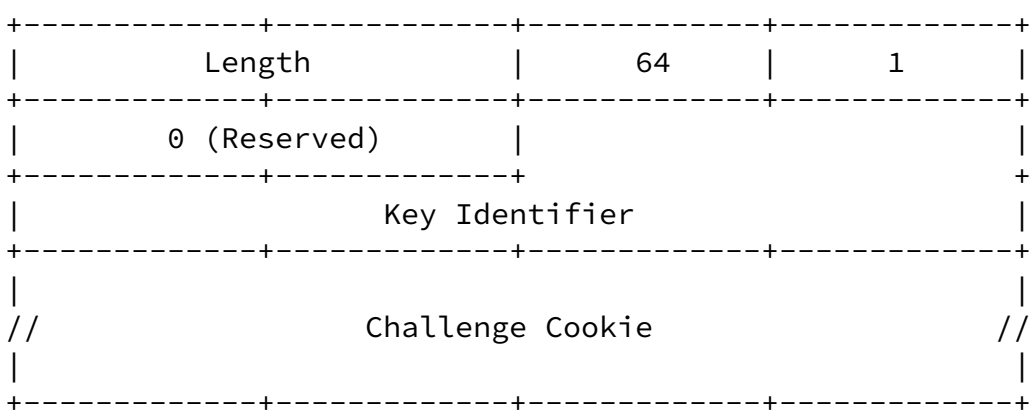
To obtain the starting sequence number for a live Authentication Key,

the receiver MAY initiate an integrity handshake with the sender. This handshake consists of a receiver's Challenge and the sender's Response, and may be either initiated during restart or postponed until a message signed with that key arrives.

Once the receiver has decided to initiate an integrity handshake for a particular Authentication Key, it identifies the sender using the sending system's address configured in the corresponding security association. The receiver then sends an Integrity Challenge, that is, a POLICY_DATA object with a CHALLENGE Option to the sender. This option contains the Key Identifier to identify the sender's key and MUST have a unique challenge cookie that is based on a local secret to prevent guessing (see Section 2.5.3 of [11]). It is suggested that the cookie be an MD5 hash of a local secret and a timestamp to provide uniqueness (see Section 9).

A CHALLENGE Option format is defined to be identical to RSVP's CHALLENGE object as defined in [8], Section 4.3. For clarity, the format is reproduced below.

o CHALLENGE option: Class = 64, C-Type = 1



Length: 16 bits

The total length of the CHALLENGE Option in octets. Must always be a multiple of 4.

Reserved: 16 bits

Unused at this time. This field MUST be set to 0.

Key Identifier: 48 bits

Challenge Cookie: Variable length

The cookie MUST be a multiple of 4 octets long.

The sender accepts the "Integrity Challenge" without doing an integrity check. It returns an "Integrity Response," that is, a POLICY_DATA object that contains the original CHALLENGE option. It also includes an INTEGRITY option, signed with the key specified by the Key Identifier included in the "Integrity Challenge".

The "Integrity Response" message is accepted by the receiver (challenger) only if the returned CHALLENGE option matches the one sent in the "Integrity Challenge" message. This prevents replay of old "Integrity Response" messages. If the match is successful, the receiver saves the Sequence Number from the INTEGRITY option as the latest sequence number received with the key identifier included in the CHALLENGE.

If a response is not received within a given period of time, the challenge is repeated. When the integrity handshake successfully completes, the receiver begins accepting normal POLICY_DATA objects from that sender and ignores any other "Integrity Response" messages.

The Handshake Flag (HF) is used to allow implementations the flexibility of not including the integrity handshake mechanism. By setting this flag to 1, message senders that implement the integrity handshake distinguish themselves from those that do not. Receivers SHOULD NOT attempt to handshake with senders whose INTEGRITY option has HF = 0.

An integrity handshake may not be necessary in all environments. A common use of POLICY_DATA integrity will be between peering PEPs, which are likely to be processing a steady stream of policy objects due to aggregation effects. When a PEP restarts after a crash, valid policy objects from peering senders will probably arrive within a short time. Assuming that replay objects are injected into the stream of valid policy objects, there may be only a small window of opportunity for a replay attack before a valid object is processed. This valid object will set the largest sequence number seen to a value greater than any number that had been stored prior to the crash, preventing any further replays.

On the other hand, not using an integrity handshake could allow exposure to replay attacks if there is a long period of silence from a given sender following a restart of a receiver. Hence, it SHOULD be an administrative decision whether or not the receiver performs an integrity handshake with senders that are willing to respond to

"Integrity Challenge" messages, and whether it accepts any messages from senders that refuse to do so. These decisions will be based on assumptions related to a particular network environment.

5. Key Management

It is likely that the IETF will define a standard key management protocol. It is strongly desirable to use that key management protocol to distribute POLICY_DATA Authentication Keys among

communicating policy aware RSVP implementations. Such a protocol would provide scalability and significantly reduce the human administrative burden. The Key Identifier can be used as a hook between PEPs and such a future protocol. Key management protocols have a long history of subtle flaws that are often discovered long after the protocol was first described in public. To avoid having to change all PEP implementations should such a flaw be discovered, integrated key management protocol techniques were deliberately omitted from this specification.

5.1. Key Management Procedures

Each key has a lifetime associated with it that is recorded in all systems (sender and receivers) configured with that key. The concept of a "key lifetime" merely requires that the earliest (KeyStartValid) and latest (KeyEndValid) times that the key is valid be programmable in a way the system understands. Certain key generation mechanisms, such as Kerberos or some public key schemes, may directly produce ephemeral keys. In this case, the lifetime of the key is implicitly defined as part of the key.

In general, no key is ever used outside its lifetime (but see Section 5.3). Possible mechanisms for managing key lifetime include the Network Time Protocol and hardware time-of-day clocks.

To maintain security, it is advisable to change the POLICY_DATA Authentication Key on a regular basis. It should be possible to switch the POLICY_DATA Authentication Key without loss of RSVP state or denial of reservation service, and without requiring people to change all the keys at once. This requires a PEP implementation to support the storage and use of more than one active POLICY_DATA Authentication Key at the same time. Hence both the sender and receivers might have multiple active keys for a given security

association.

Since keys are shared between a sender and (possibly) multiple receivers, there is a region of uncertainty around the time of key switch-over during which some systems may still be using the old key and others might have switched to the new key. The size of this uncertainty region is related to clock synchrony of the systems. Administrators should configure the overlap between the expiration time of the old key (KeyEndValid) and the validity of the new key (KeyStartValid) to be at least twice the size of this uncertainty interval. This will allow the sender to make the key switch-over at the midpoint of this interval and be confident that all receivers are now accepting the new key. For the duration of the overlap in key lifetimes, a receiver must be prepared to authenticate messages using either key.

During a key switch-over, it will be necessary for each receiver to handshake with the sender using the new key. As stated before, a receiver has the choice of initiating a handshake during the

switchover or postponing the handshake until the receipt of a message using that key.

5.2. Key Management Requirements

Requirements on an implementation are as follows:

- It is strongly desirable that a hypothetical security breach in one Internet protocol not automatically compromise other Internet protocols. The Authentication Key of this specification SHOULD NOT be stored using protocols or algorithms that have known flaws.

- An implementation MUST support the storage and use of more than one key at the same time, for both sending and receiving systems.

- An implementation MUST associate a specific lifetime (i.e., KeyStartValid and KeyEndValid) with each key and the corresponding Key Identifier.

- An implementation MUST support manual key distribution (e.g., the privileged user manually typing in the key, key lifetime,

and key identifier on the console). The lifetime may be infinite.

- If more than one algorithm is supported, then the implementation MUST require that the algorithm be specified for each key at the time the other key information is entered.
- Keys that are out of date MAY be automatically deleted by the implementation.
- Manual deletion of active keys MUST also be supported.
- Key storage SHOULD persist across a system restart, warm or cold, to ease operational usage.

5.3. Pathological Case

It is possible that the last key for a given security association has expired. When this happens, it is unacceptable to revert to an unauthenticated condition, and not advisable to disrupt current reservations. Therefore, the system should send a "last authentication key expiration" notification to the network manager and treat the key as having an infinite lifetime until the lifetime is extended, the key is deleted by network management, or a new key is configured.

6. Conformance Requirements

To conform to this specification, an implementation MUST support all of its aspects. The HMAC-MD5 authentication algorithm defined in [4] MUST be implemented by all conforming implementations. A conforming implementation MAY also support other authentication algorithms such as NIST's Secure Hash Algorithm (SHA). Manual key distribution as described above MUST be supported by all conforming implementations. All implementations MUST support the smooth key roll over described under "Key Management Procedures."

Implementations SHOULD support a standard key management protocol for secure distribution of POLICY_DATA Authentication Keys once such a

key management protocol is standardized by the IETF.

7. Kerberos Generation of POLICY_DATA Authentication Keys

Kerberos [12] MAY be used to generate the POLICY_DATA Authentication key used in generating a signature in the Integrity Option sent from a PEP sender to a receiver. Kerberos key generation avoids the use of shared keys between PEP senders and receivers such as hosts and routers. Kerberos allows for the use of trusted third party keying relationships between security principals (PEP sender and receivers) where the Kerberos key distribution center (KDC) establishes an ephemeral session key that is subsequently shared between PEP sender and receivers. In the multicast case all receivers of a multicast POLICY_DATA object MUST share a single key with the KDC (e.g. the receivers are in effect the same security principal with respect to Kerberos).

The Key information determined by the sender MAY specify the use of Kerberos in place of configured shared keys as the mechanism for establishing a key between the sender and receiver. The Kerberos identity of the receiver is established as part of the sender's interface configuration or it can be established through other mechanisms. When generating the first Integrity Option for a specific key identifier the sender requests a Kerberos service ticket and gets back an ephemeral session key and a Kerberos ticket from the KDC. The sender encapsulates the ticket and the identity of the sender in an Identity Option of the POLICY_DATA object [10]. The session key is then used by the sender as the POLICY_DATA Authentication key in section 4.1 step (2) and is stored as Key information associated with the key identifier.

Upon policy object reception, the receiver retrieves the Kerberos Ticket from the Identity Option, decrypts the ticket and retrieves the session key from the ticket. The session key is the same key as used by the sender and is used as the key in section 4.2 step (3). The receiver stores the key for use in processing subsequent policy objects.

Kerberos tickets have lifetimes and the sender MUST NOT use tickets

that have expired. A new ticket MUST be requested and used by the sender for the receiver prior to the ticket expiring.

7.1. Optimization when using Kerberos Based Authentication

Kerberos tickets are relatively long (> 500 bytes) and it is not necessary to send a ticket in every POLICY_DATA object. The ephemeral session key can be cached by the sender and receiver and can be used for the lifetime of the Kerberos ticket. In this case, the sender only needs to include the Kerberos ticket in the first POLICY_DATA object generated. Subsequent messages use the key identifier to retrieve the cached key (and optionally other identity information) instead of passing tickets from sender to receiver in each POLICY_DATA object.

A receiver may not have cached key state with an associated Key Identifier due to reboot or route changes. If the receiver's policy indicates the use of Kerberos keys for integrity checking, the receiver can send an integrity Challenge message back to the sender. Upon receiving an integrity Challenge message a sender MUST send an Identity option that includes the Kerberos ticket in the integrity Response message, thereby allowing the receiver to retrieve and store the session key from the Kerberos ticket for subsequent Integrity checking.

8. Acknowledgements

This document is derived directly from similar work done for RSVP by Fred Baker, Bob Lindell and Mohit Talwar in [8].

9. References

[1] Braden, R., Zhang, L., Berson, S., Herzog, S. and S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.

[2] Hess, R., Ed., Herzog, S., "RSVP Extensions for Policy Control", work in progress, draft-ietf-rap-new-rsvp-ext-00.txt, June 2001.

[3] Baker, F., Lindell, B. and Talwar, M., "RSVP Cryptographic Authentication", RFC 2747, January 2000.

[4] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, March 1996.

[5] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

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

[8] Baker, F., Lindell, B. and Talwar, M., "RSVP Cryptographic Authentication", RFC 2747, January 2000.

[9] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[10] Yadav, S., et al., "Identity Representation for RSVP", RFC 2752, January 2000.

[11] Maughan, D., Schertler, M., Schneider, M. and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, November 1998.

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

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

10. Security Considerations

This entire memo describes and specifies an authentication mechanism for RSVP POLICY_DATA objects that is believed to be secure against active and passive attacks.

The quality of the security provided by this mechanism depends on the strength of the implemented authentication algorithms, the strength of the key being used, and the correct implementation of the security mechanism in all communicating policy aware RSVP implementations. This mechanism also depends on the POLICY_DATA Authentication Keys being kept confidential by all parties. If any of these assumptions are incorrect or procedures are insufficiently secure, then no real security will be provided to the users of this mechanism.

While the handshake "Integrity Response" message is integrity-checked, the handshake "Integrity Challenge" message is not. This was done intentionally to avoid the case when both peering routers do not have a starting sequence number for each other's key. Consequently, they will each keep sending handshake "Integrity Challenge" messages that will be dropped by the other end. Moreover,

requiring only the response to be integrity-checked eliminates a dependency on an security association in the opposite direction.

This, however, lets an intruder generate fake handshaking challenges with a certain challenge cookie. It could then save the response and attempt to play it against a receiver that is in recovery. If it was lucky enough to have guessed the challenge cookie used by the receiver at recovery time it could use the saved response. This response would be accepted, since it is properly signed, and would

have a smaller sequence number for the sender because it was an old message. This opens the receiver up to replays. Still, it seems very difficult to exploit. It requires not only guessing the challenge cookie (which is based on a locally known secret) in advance, but also being able to masquerade as the receiver to generate a handshake "Integrity Challenge" with the proper IP address and not being caught.

Confidentiality is not provided by this mechanism. If confidentiality is required, IPsec ESP [13] may be the best approach, although it is subject to the same criticisms as IPsec Authentication, and therefore would be applicable only in specific environments. Protection against traffic analysis is also not provided. Mechanisms such as bulk link encryption might be used when protection against traffic analysis is required.

11. Author's Address

Rodney Hess
Intel Corp, BD1
28 Crosby Dr
Bedford, MA 01730

EMail: rodney.hess@intel.com

Appendix A: Key Management Interface

This appendix describes a generic interface to Key Management. This description is at an abstract level realizing that implementations may need to introduce small variations to the actual interface.

At the start of execution, a policy aware RSVP system would use this interface to obtain the current set of relevant keys for sending and receiving POLICY_DATA objects. During execution, it can query for specific keys given a Key Identifier and Source Address, discover newly created keys, and be informed of those keys that have been deleted. The interface provides both a polling and asynchronous upcall style for wider applicability.

A.1. Data Structures

Information about keys is returned using the following KeyInfo data structure:

```
  KeyInfo {
          Key Type (Send or Receive)
          KeyIdentifier
          Key
          Authentication Algorithm Type and Mode
          KeyStartValid
```

```
            KeyEndValid
            Status (Active or Deleted)
            Outgoing Interface (for Send only)
            Other Outgoing Security Association Selection Criteria
                    (for Send only, optional)
            Sending System Address (for Receive Only)
    }
```

A.2. Default Key Table

This function returns a list of KeyInfo data structures corresponding to all of the keys that are configured for sending and receiving POLICY_DATA objects and have an Active Status. This function is usually called at the start of execution but there is no limit on the number of times that it may be called.

```
KM_DefaultKeyTable() -> KeyInfoList
```

A.3. Querying for Unknown Receive Keys

When a message arrives with an unknown Key Identifier and Sending System Address pair, PEP can use this function to query the Key Management System for the appropriate key. The status of the element returned, if any, must be Active.

```
KM_GetRecvKey( INTEGRITY Object, SrcAddress ) -> KeyInfo
```


A.4. Polling for Updates

This function returns a list of KeyInfo data structures corresponding to any incremental changes that have been made to the default key table or requested keys since the last call to either KM_KeyTablePoll, KM_DefaultKeyTable, or KM_GetRecvKey. The status of some elements in the returned list may be set to Deleted.

```
KM_KeyTablePoll() -> KeyInfoList
```

A.5. Asynchronous Upcall Interface

Rather than repeatedly calling the KM_KeyTablePoll(), an implementation may choose to use an asynchronous event model. This function registers interest to key changes for a given Key Identifier

or for all keys if no Key Identifier is specified. The upcall function is called each time a change is made to a key.

KM_KeyUpdate (Function [, KeyIdentifier])

where the upcall function is parameterized as follows:

Function (KeyInfo)

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.