

Network Working Group
Request for Comments: 2522
Category: Experimental

P. Karn
Qualcomm
W. Simpson
DayDreamer
March 1999

Photuris: Session-Key Management Protocol

Status of this Memo

This document defines an Experimental Protocol for the Internet community. It does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (1999). Copyright (C) Philip Karn and William Allen Simpson (1994-1999). All Rights Reserved.

Abstract

Photuris is a session-key management protocol intended for use with the IP Security Protocols (AH and ESP). This document defines the basic protocol mechanisms.

Table of Contents

| | | |
|-------|-----------------------------------|----|
| 1. | Introduction | 1 |
| 1.1 | Terminology | 1 |
| 1.2 | Protocol Overview | 3 |
| 1.3 | Security Parameters | 5 |
| 1.4 | LifeTimes | 6 |
| 1.4.1 | Exchange LifeTimes | 6 |
| 1.4.2 | SPI LifeTimes | 7 |
| 1.5 | Random Number Generation | 8 |
| 2. | Protocol Details | 9 |
| 2.1 | UDP | 9 |
| 2.2 | Header Format | 10 |
| 2.3 | Variable Precision Integers | 11 |
| 2.4 | Exchange-Schemes | 13 |
| 2.5 | Attributes | 13 |
| 3. | Cookie Exchange | 14 |
| 3.0.1 | Send Cookie_Request | 14 |
| 3.0.2 | Receive Cookie_Request | 15 |
| 3.0.3 | Send Cookie_Response | 15 |
| 3.0.4 | Receive Cookie_Response | 16 |
| 3.1 | Cookie_Request | 17 |
| 3.2 | Cookie_Response | 18 |
| 3.3 | Cookie Generation | 19 |
| 3.3.1 | Initiator Cookie | 19 |
| 3.3.2 | Responder Cookie | 20 |
| 4. | Value Exchange | 21 |
| 4.0.1 | Send Value_Request | 21 |
| 4.0.2 | Receive Value_Request | 22 |
| 4.0.3 | Send Value_Response | 22 |
| 4.0.4 | Receive Value_Response | 23 |
| 4.1 | Value_Request | 24 |
| 4.2 | Value_Response | 25 |
| 4.3 | Offered Attribute List | 26 |
| 5. | Identification Exchange | 28 |
| 5.0.1 | Send Identity_Request | 29 |
| 5.0.2 | Receive Identity_Request | 29 |
| 5.0.3 | Send Identity_Response | 30 |
| 5.0.4 | Receive Identity_Response | 30 |
| 5.1 | Identity_Messages | 31 |
| 5.2 | Attribute Choices List | 33 |
| 5.3 | Shared-Secret | 34 |
| 5.4 | Identity Verification | 34 |

| | | |
|--------|-------------------------------------|----|
| 5.5 | Privacy-Key Computation | 36 |
| 5.6 | Session-Key Computation | 37 |
| 6. | SPI Messages | 38 |
| 6.0.1 | Send SPI_Needed | 38 |
| 6.0.2 | Receive SPI_Needed | 39 |
| 6.0.3 | Send SPI_Update | 39 |
| 6.0.4 | Receive SPI_Update | 39 |
| 6.0.5 | Automated SPI_Updates | 40 |
| 6.1 | SPI_Needed | 41 |
| 6.2 | SPI_Update | 43 |
| 6.2.1 | Creation | 44 |
| 6.2.2 | Deletion | 45 |
| 6.2.3 | Modification | 45 |
| 6.3 | Validity Verification | 45 |
| 7. | Error Messages | 46 |
| 7.1 | Bad_Cookie | 47 |
| 7.2 | Resource_Limit | 47 |
| 7.3 | Verification_Failure | 48 |
| 7.4 | Message_Reject | 49 |
| 8. | Public Value Exchanges | 50 |
| 8.1 | Modular Exponentiation Groups | 50 |
| 8.2 | Moduli Selection | 50 |
| 8.2.1 | Bootstrap Moduli | 51 |
| 8.2.2 | Learning Moduli | 51 |
| 8.3 | Generator Selection | 51 |
| 8.4 | Exponent Selection | 52 |
| 8.5 | Defective Exchange Values | 53 |
| 9. | Basic Exchange-Schemes | 54 |
| 10. | Basic Key-Generation-Function | 55 |
| 10.1 | MD5 Hash | 55 |
| 11. | Basic Privacy-Method | 55 |
| 11.1 | Simple Masking | 55 |
| 12. | Basic Validity-Method | 55 |
| 12.1 | MD5-IPMAC Check | 55 |
| 13. | Basic Attributes | 56 |
| 13.1 | Padding | 56 |
| 13.2 | AH-Attributes | 57 |
| 13.3 | ESP-Attributes | 57 |
| 13.4 | MD5-IPMAC | 58 |
| 13.4.1 | Symmetric Identification | 58 |

| | | |
|----------------------------------|---|----|
| 13.4.2 | Authentication | 59 |
| 13.5 | Organizational | 60 |
| APPENDICES | | 61 |
| A. | Automaton | 61 |
| A.1 | State Transition Table | 62 |
| A.2 | States | 65 |
| A.2.1 | Initial | 65 |
| A.2.2 | Cookie | 66 |
| A.2.3 | Value | 66 |
| A.2.4 | Identity | 66 |
| A.2.5 | Ready | 66 |
| A.2.6 | Update | 66 |
| B. | Use of Identification and Secrets | 67 |
| B.1 | Identification | 67 |
| B.2 | Group Identity With Group Secret | 67 |
| B.3 | Multiple Identities With Group Secrets | 68 |
| B.4 | Multiple Identities With Multiple Secrets | 69 |
| OPERATIONAL CONSIDERATIONS | | 70 |
| SECURITY CONSIDERATIONS | | 70 |
| HISTORY | | 71 |
| ACKNOWLEDGEMENTS | | 72 |
| REFERENCES | | 73 |
| CONTACTS | | 75 |
| COPYRIGHT | | 76 |

1. Introduction

Photuris [Firefly] establishes short-lived session-keys between two parties, without passing the session-keys across the Internet. These session-keys directly replace the long-lived secret-keys (such as passwords and passphrases) that have been historically configured for security purposes.

The basic Photuris protocol utilizes these existing previously configured secret-keys for identification of the parties. This is intended to speed deployment and reduce administrative configuration changes.

This document is primarily intended for implementing the Photuris protocol. It does not detail service and application interface definitions, although it does mention some basic policy areas required for the proper implementation and operation of the protocol mechanisms.

Since the basic Photuris protocol is extensible, new data types and protocol behaviour should be expected. The implementor is especially cautioned not to depend on values that appear in examples to be current or complete, since their purpose is primarily pedagogical.

1.1. Terminology

In this document, the key words "MAY", "MUST", "MUST NOT", "optional", "recommended", "SHOULD", and "SHOULD NOT", are to be interpreted as described in [RFC-2119].

| | |
|----------------|--|
| byte | An 8-bit quantity; also known as "octet" in standardese. |
| exchange-value | The publically distributable value used to calculate a shared-secret. As used in this document, refers to a Diffie-Hellman exchange, not the public part of a public/private key-pair. |
| private-key | A value that is kept secret, and is part of an asymmetric public/private key-pair. |
| public-key | A publically distributable value that is part of an asymmetric public/private key-pair. |
| secret-key | A symmetric key that is not publically distributable. As used in this document, this is distinguished from an asymmetric public/private |

key-pair. An example is a user password.

Security Association (SA)

A collection of parameters describing the security relationship between two nodes. These parameters include the identities of the parties, the transform (including algorithm and algorithm mode), the key(s) (such as a session-key, secret-key, or appropriate public/private key-pair), and possibly other information such as sensitivity labelling.

Security Parameters Index (SPI)

A number that indicates a particular set of unidirectional attributes used under a Security Association, such as transform(s) and session-key(s). The number is relative to the IP Destination, which is the SPI Owner, and is unique per IP (Next Header) Protocol. That is, the same value MAY be used by multiple protocols to concurrently indicate different Security Association parameters.

session-key A key that is independently derived from a shared-secret by the parties, and used for keying one direction of traffic. This key is changed frequently.

shared-secret As used in this document, the calculated result of the Photuris exchange.

SPI Owner The party that corresponds to the IP Destination; the intended recipient of a protected datagram.

SPI User The party that corresponds to the IP Source; the sender of a protected datagram.

transform A cryptographic manipulation of a particular set of data. As used in this document, refers to certain well-specified methods (defined elsewhere). For example, AH-MD5 [RFC-1828] transforms an IP datagram into a cryptographic hash, and ESP-DES-CBC [RFC-1829] transforms plaintext to ciphertext and back again.

Many of these terms are hierarchically related:

- Security Association (bi-directional)
 - one or more lists of Security Parameters (uni-directional)
 - one or more Attributes
 - may have a key
 - may indicate a transform

Implementors will find details of cryptographic hashing (such as MD5), encryption algorithms and modes (such as DES), digital signatures (such as DSS), and other algorithms in [Schneier95].

1.2. Protocol Overview

The Photuris protocol consists of several simple phases:

1. A "Cookie" Exchange guards against simple flooding attacks sent with bogus IP Sources or UDP Ports. Each party passes a "cookie" to the other.

In return, a list of supported Exchange-Schemes are offered by the Responder for calculating a shared-secret.

2. A Value Exchange establishes a shared-secret between the parties. Each party passes an Exchange-Value to the other. These values are used to calculate a shared-secret. The Responder remains stateless until a shared-secret has been created.

In addition, supported attributes are offered by each party for use in establishing new Security Parameters.

3. An Identification Exchange identifies the parties to each other, and verifies the integrity of values sent in phases 1 and 2.

In addition, the shared-secret provides a basis to generate separate session-keys in each direction, which are in turn used for conventional authentication or encryption. Additional security attributes are also exchanged as needed.

This exchange is masked for party privacy protection using a message privacy-key based on the shared-secret. This protects the identities of the parties, hides the Security Parameter attribute values, and improves security for the exchange protocol and security transforms.

4. Additional messages may be exchanged to periodically change the session-keys, and to establish new or revised Security Parameters.

These exchanges are also masked for party privacy protection in the same fashion as above.

The sequence of message types and their purposes are summarized in the diagram below. The first three phases (cookie, exchange, and identification) must be carried out in their entirety before any Security Association can be used.

| Initiator | | Responder |
|--|----|--|
| ===== | | ===== |
| Cookie_Request | -> | |
| | <- | Cookie_Response offer schemes |
| Value_Request | -> | |
| pick scheme | | |
| offer value | | |
| offer attributes | | |
| | <- | Value_Response offer value offer attributes |
| [generate shared-secret from exchanged values] | | |
| Identity_Request | -> | |
| make SPI | | |
| pick SPI attribute(s) | | |
| identify self | | |
| authenticate | | |
| make privacy key(s) | | |
| mask/encrypt message | | |
| | <- | Identity_Response make SPI pick SPI attribute(s) identify self authenticate make privacy key(s) mask/encrypt message |
| [make SPI session-keys in each direction] | | |

| | | |
|--|----|---|
| SPI User ===== SPI_Needed list SPI attribute(s) make validity key authenticate make privacy key(s) mask/encrypt message | -> | SPI Owner ===== <- SPI_Update make SPI pick SPI attribute(s) make SPI session-key(s) make validity key authenticate make privacy key(s) mask/encrypt message |
|--|----|---|

Either party may initiate an exchange at any time. For example, the Initiator need not be a "caller" in a telephony link.

The Initiator is responsible for recovering from all message losses by retransmission.

1.3. Security Parameters

A Photuris exchange between two parties results in a pair of SPI values (one in each direction). Each SPI is used in creating separate session-key(s) in each direction.

The SPI is assigned by the entity controlling the IP Destination: the SPI Owner (receiver). The parties use the combination of IP Destination, IP (Next Header) Protocol, and SPI to distinguish the correct Security Association.

When both parties initiate Photuris exchanges concurrently, or one party initiates more than one Photuris exchange, the Initiator Cookies (and UDP Ports) keep the exchanges separate. This results in more than one initial SPI for each Destination.

To create multiple SPIs with different parameters, the parties may also send SPI_Updates.

There is no requirement that all such outstanding SPIs be used. The SPI User (sender) selects an appropriate SPI for each datagram transmission.

Implementation Notes:

The method used for SPI assignment is implementation dependent. The only requirement is that the SPI be unique for the IP Destination and IP (Next Header) Protocol.

However, selection of a cryptographically random SPI value can help prevent attacks that depend on a predictable sequence of values. The implementor MUST NOT expect SPI values to have a particular order or range.

1.4. LifeTimes

The Photuris exchange results in two kinds of state, each with separate LifeTimes.

- 1) The Exchange LifeTime of the small amount of state associated with the Photuris exchange itself. This state may be viewed as between Internet nodes.
- 2) The SPI LifeTimes of the individual SPIs that are established. This state may be viewed as between users and nodes.

The SPI LifeTimes may be shorter or longer than the Exchange LifeTime. These LifeTimes are not required to be related to each other.

When an Exchange-Value expires (or is replaced by a newer value), any unexpired derived SPIs are not affected. This is important to allow traffic to continue without interruption during new Photuris exchanges.

1.4.1. Exchange LifeTimes

All retained exchange state of both parties has an associated Exchange LifeTime (ELT), and is subject to periodic expiration. This depends on the physical and logistical security of the machine, and is typically in the range of 10 minutes to one day (default 30 minutes).

In addition, during a Photuris exchange, an Exchange TimeOut (ETO) limits the wait for the exchange to complete. This timeout includes the packet round trips, and the time for completing the Identification Exchange calculations. The time is bounded by both the maximum amount of calculation delay expected for the processing power of an unknown peer, and the minimum user expectation for

results (default 30 seconds).

These Exchange LifeTimes and TimeOuts are implementation dependent and are not disclosed in any Photuris message. The paranoid operator will have a fairly short Exchange LifeTime, but it MUST NOT be less than twice the ETO.

To prevent synchronization between Photuris exchanges, the implementation SHOULD randomly vary each Exchange LifeTime within twice the range of seconds that are required to calculate a new Exchange-Value. For example, when the Responder uses a base ELT of 30 minutes, and takes 10 seconds to calculate the new Exchange-Value, the equation might be (in milliseconds):

$$1790000 + \text{urandom}(20000)$$

The Exchange-Scheme, Exchange-Values, and resulting shared-secret MAY be cached in short-term storage for the Exchange LifeTime. When repetitive Photuris exchanges occur between the same parties, and the Exchange-Values are discovered to be unchanged, the previously calculated shared-secret can be used to rapidly generate new session-keys.

1.4.2. SPI LifeTimes

Each SPI has an associated LifeTime, specified by the SPI owner (receiver). This SPI LifeTime (SPILT) is usually related to the speed of the link (typically 2 to 30 minutes), but it MUST NOT be less than thrice the ETO.

The SPI can also be deleted by the SPI Owner using the SPI_Update. Once the SPI has expired or been deleted, the parties cease using the SPI.

To prevent synchronization between multiple Photuris exchanges, the implementation SHOULD randomly vary each SPI LifeTime. For example, when the Responder uses a base SPILT of 5 minutes, and 30 seconds for the ETO, the equation might be (in milliseconds):

$$285000 + \text{urandom}(30000)$$

There is no requirement that a long LifeTime be accepted by the SPI User. The SPI User might never use an established SPI, or cease using the SPI at any time.

When more than one unexpired SPI is available to the SPI User for the same function, a common implementation technique is to select the SPI

with the greatest remaining LifeTime. However, selecting randomly among a large number of SPIs might provide some defense against traffic analysis.

To prevent resurrection of deleted or expired SPIs, SPI Owners SHOULD remember those SPIs, but mark them as unusable until the Photuris exchange shared-secret used to create them also expires and purges the associated state.

When the SPI Owner detects an incoming SPI that has recently expired, but the associated exchange state has not yet been purged, the implementation MAY accept the SPI. The length of time allowed is highly dependent on clock drift and variable packet round trip time, and is therefore implementation dependent.

1.5. Random Number Generation

The security of Photuris critically depends on the quality of the secret random numbers generated by each party. A poor random number generator at either party will compromise the shared-secret produced by the algorithm.

Generating cryptographic quality random numbers on a general purpose computer without hardware assistance is a very tricky problem. In general, this requires using a cryptographic hashing function to "distill" the entropy from a large number of semi-random external events, such as the timing of key strokes. An excellent discussion can be found in [RFC-1750].

2. Protocol Details

The Initiator begins a Photuris exchange under several circumstances:

- The Initiator has a datagram that it wishes to send with confidentiality, and has no current Photuris exchange state with the IP Destination. This datagram is discarded, and a Cookie_Request is sent instead.
- The Initiator has received the ICMP message [RFC-1812] Destination Unreachable: Communication Administratively Prohibited (Type 3, Code 13), and has no current Photuris exchange state with the ICMP Source.
- The Initiator has received the ICMP message [RFC-2521] Security Failures: Bad SPI (Type 40, Code 0), that matches current Photuris exchange state with the ICMP Source.
- The Initiator has received the ICMP message [RFC-2521] Security Failures: Need Authentication (Type 40, Code 4), and has no current Photuris exchange state with the ICMP Source.
- The Initiator has received the ICMP message [RFC-2521] Security Failures: Need Authorization (Type 40, Code 5), that matches current Photuris exchange state with the ICMP Source.

When the event is an ICMP message, special care MUST be taken that the ICMP message actually includes information that matches a previously sent IP datagram. Otherwise, this could provide an opportunity for a clogging attack, by stimulating a new Photuris Exchange.

2.1. UDP

All Photuris messages use the User Datagram Protocol header [RFC-768]. The Initiator sends to UDP Destination Port 468.

When replying to the Initiator, the Responder swaps the IP Source and Destination, and the UDP Source and Destination Ports.

The UDP checksum MUST be correctly calculated when sent. When a message is received with an incorrect UDP checksum, it is silently discarded.

Implementation Notes:

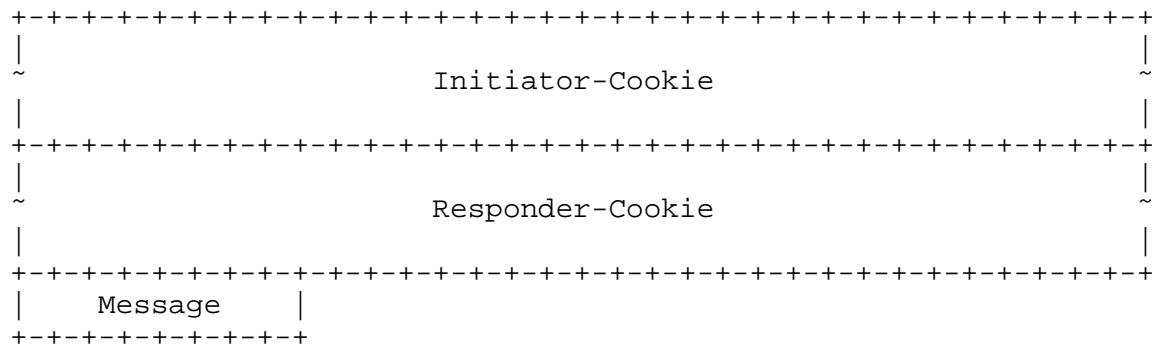
It is expected that installation of Photuris will ensure that UDP checksum calculations are enabled for the computer operating system and later disabling by operators is prevented.

Internet Protocol version 4 [RFC-791] restricts the maximum reassembled datagram to 576 bytes.

When processing datagrams containing variable size values, the length must be checked against the overall datagram length. An invalid size (too long or short) that causes a poorly coded receiver to abort could be used as a denial of service attack.

2.2. Header Format

All of the messages have a format similar to the following, as transmitted left to right in network order (most significant to least significant):



Initiator-Cookie 16 bytes.

Responder-Cookie 16 bytes.

Message 1 byte. Each message type has a unique value.
Initial values are assigned as follows:

- 0 Cookie_Request
- 1 Cookie_Response
- 2 Value_Request
- 3 Value_Response
- 4 Identity_Request
- 5 Secret_Response (optional)
- 6 Secret_Request (optional)
- 7 Identity_Response
- 8 SPI_Needed
- 9 SPI_Update
- 10 Bad_Cookie
- 11 Resource_Limit
- 12 Verification_Failure
- 13 Message_Reject

Further details and differences are elaborated in the individual messages.

2.3. Variable Precision Integers

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|               Size               |               Value ...
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

Size 2, 4, or 8 bytes. The number of significant bits used in the Value field. Always transmitted most significant byte first.

When the Size is zero, no Value field is present; there are no significant bits. This means "missing" or "null". It should not be confused with the value zero, which includes an indication of the number of significant bits.

When the most significant byte is in the range 0 through 254 (0xfe), the field is 2 bytes. Both bytes are used to indicate the size of the Value field, which ranges from 1 to 65,279 significant bits (in 1 to 8,160 bytes).

When the most significant byte is 255 (0xff), the field is 4 bytes. The remaining 3 bytes are added to 65,280 to indicate the size of the Value field, which is limited to 16,776,959 significant bits (in

2,097,120 bytes).

When the most significant 2 bytes are 65,535 (0xffff), the field is 8 bytes. The remaining 6 bytes are added to 16,776,960 to indicate the size of the Value field.

Value 0 or more bytes. Always transmitted most significant byte first.

The bits used are right justified within byte boundaries; that is, any unused bits are in the most significant byte. When there are no unused bits, or unused bits are zero filled, the value is assumed to be an unsigned positive integer.

When the leading unused bits are ones filled, the number is assumed to be a two's-complement negative integer. A negative integer will always have at least one unused leading sign bit in the most significant byte.

Shortened forms SHOULD NOT be used when the Value includes a number of leading zero significant bits. The Size SHOULD indicate the correct number of significant bits.

Implementation Notes:

Negative integers are not required to be supported, but are included for completeness.

No more than 65,279 significant bits are required to be supported. Other ranges are vastly too long for these UDP messages, but are included for completeness.

2.4. Exchange-Schemes

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|           Scheme           |           Size           |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|           Value ...
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

Scheme 2 bytes. A unique value indicating the Exchange-Scheme. See the "Basic Exchange-Schemes" for details.

Size 2 bytes, ranging from 0 to 65,279. See "Variable Precision Integer".

Value 0 or more bytes. See "Variable Precision Integer".

The Size MUST NOT be assumed to be constant for a particular Scheme. Multiple kinds of the same Scheme with varying Sizes MAY be present in any list of schemes.

However, only one of each Scheme and Size combination will be present in any list of schemes.

2.5. Attributes

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Attribute | Length | Value(s) ...
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

Attribute 1 byte. A unique value indicating the kind of attribute. See the "Basic Attributes" for details.

When the value is zero (padding), no Length field is present (always zero).

Length 1 byte. The size of the Value(s) field in bytes.

When the Length is zero, no Value(s) field is present.

Value(s) 0 or more bytes. See the "Basic Attributes" for details.

The Length MUST NOT be assumed to be constant for a particular

Attribute. Multiple kinds of the same Attribute with varying Lengths MAY be present in any list of attributes.

3. Cookie Exchange

| | | |
|----------------|----|-----------------|
| Initiator | | Responder |
| ===== | | ===== |
| Cookie_Request | -> | |
| | <- | Cookie_Response |
| | | offer schemes |

3.0.1. Send Cookie_Request

The Initiator initializes local state, and generates a unique "cookie". The Initiator-Cookie MUST be different in each new Cookie_Request between the same parties. See "Cookie Generation" for details.

- If any previous exchange between the peer IP nodes has not expired in which this party was the Initiator, this Responder-Cookie is set to the most recent Responder-Cookie, and this Counter is set to the corresponding Counter.

For example, a new Virtual Private Network (VPN) tunnel is about to be established to an existing partner. The Counter is the same value received in the prior Cookie_Response, the Responder-Cookie remains the same, and a new Initiator-Cookie is generated.

- If the new Cookie_Request is in response to a message of a previous exchange in which this party was the Responder, this Responder-Cookie is set to the previous Initiator-Cookie, and this Counter is set to zero.

For example, a Bad_Cookie message was received from the previous Initiator in response to SPI_Needed. The Responder-Cookie is replaced with the Initiator-Cookie, and a new Initiator-Cookie is generated. This provides bookkeeping to detect bogus Bad_Cookie messages.

Also, can be used for bi-directional User, Transport, and Process oriented keying. Such mechanisms are outside the scope of this document.

- Otherwise, this Responder-Cookie and Counter are both set to zero.

By default, the Initiator operates in the same manner as when all of its previous exchange state has expired. The Responder will send a Resource_Limit when its own exchange state has not expired.

The Initiator also starts a retransmission timer. If no valid Cookie_Response arrives within the time limit, the same Cookie_Request is retransmitted for the remaining number of Retransmissions. The Initiator-Cookie value MUST be the same in each such retransmission to the same IP Destination and UDP Port.

When Retransmissions have been exceeded, if a Resource_Limit message has been received during the exchange, the Initiator SHOULD begin the Photuris exchange again by sending a new Cookie_Request with updated values.

3.0.2. Receive Cookie_Request

On receipt of a Cookie_Request, the Responder determines whether there are sufficient resources to begin another Photuris exchange.

- When too many SPI values are already in use for this particular peer, or too many concurrent exchanges are in progress, or some other resource limit is reached, a Resource_Limit message is sent.
- When any previous exchange initiated by this particular peer has not exceeded the Exchange Timeout, and the Responder-Cookie does not specify one of these previous exchanges, a Resource_Limit message is sent.

Otherwise, the Responder returns a Cookie_Response.

Note that the Responder creates no additional state at this time.

3.0.3. Send Cookie_Response

The IP Source for the Initiator is examined. If any previous exchange between the peer IP nodes has not expired, the response Counter is set to the most recent exchange Counter plus one (allowing for out of order retransmissions). Otherwise, the response Counter is set to the request Counter plus one.

If (through rollover of the Counter) the new Counter value is zero (modulo 256), the value is set to one.

If this new Counter value matches some previous exchange initiated by this particular peer that has not yet exceeded the Exchange Timeout,

the Counter is incremented again, until a unique Counter value is reached.

Nota Bene:

No more than 254 concurrent exchanges between the same two peers are supported.

The Responder generates a unique cookie. The Responder-Cookie value in each successive response SHOULD be different. See "Cookie Generation" for details.

The Exchange-Schemes available between the peers are listed in the Offered-Schemes.

3.0.4. Receive Cookie_Response

The Initiator validates the Initiator-Cookie, and the Offered-Schemes.

- When an invalid/expired Initiator-Cookie is detected, the message is silently discarded.
- When the variable length Offered-Schemes do not match the UDP Length, or all Offered-Schemes are obviously defective and/or insufficient for the purposes intended, the message is silently discarded; the implementation SHOULD log the occurrence, and notify an operator as appropriate.
- Once a valid message has been received, later Cookie_Responses with matching Initiator-Cookies are also silently discarded, until a new Cookie_Request is sent.

When the message is valid, an Exchange-Scheme is chosen from the list of Offered-Schemes.

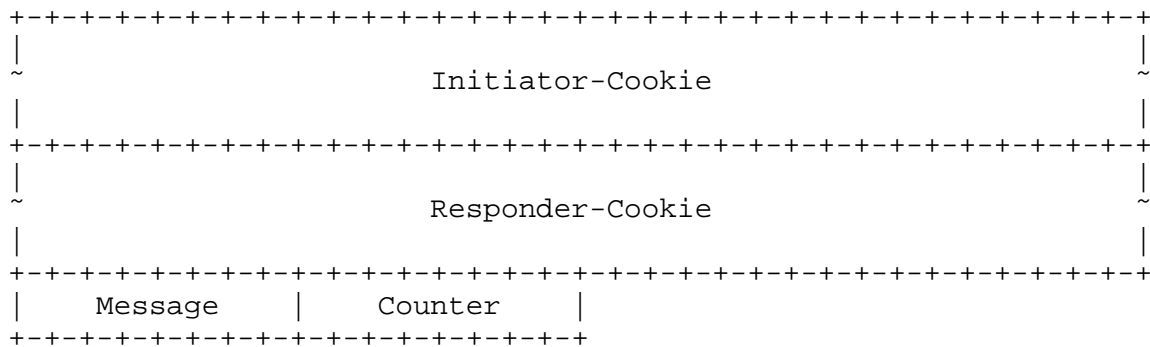
This Scheme-Choice may affect the next Photuris message sent. By default, the next Photuris message is a Value_Request.

Implementation Notes:

Only the Initiator-Cookie is used to identify the exchange. The Counter and Responder-Cookie will both be different from the Cookie_Request.

Various proposals for extensions utilize the Scheme-Choice to indicate a different message sequence. Such mechanisms are outside the scope of this document.

3.1. Cookie_Request



Initiator-Cookie 16 bytes. A randomized value that identifies the exchange. The value **MUST NOT** be zero. See "Cookie Generation" for details.

Responder-Cookie 16 bytes. Identifies a specific previous exchange. Copied from a previous **Cookie_Response**.

When zero, no previous exchange is specified.

When non-zero, and the Counter is zero, contains the Initiator-Cookie of a previous exchange. The specified party is requested to be the Responder in this exchange, to retain previous party pairings.

When non-zero, and the Counter is also non-zero, contains the Responder-Cookie of a previous exchange. The specified party is requested to be the Responder in this exchange, to retain previous party pairings.

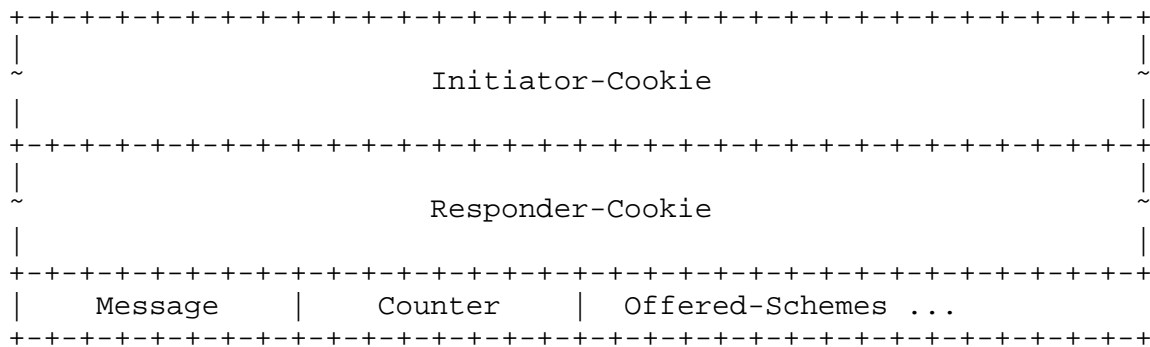
Message 0

Counter 1 byte. Indicates the number of previous exchanges.

When zero, the Responder-Cookie indicates the Initiator of a previous exchange, or no previous exchange is specified.

When non-zero, the Responder-Cookie indicates the Responder to a previous exchange. This value is set to the Counter from the corresponding **Cookie_Response** or from a **Resource_Limit**.

3.2. Cookie_Response



Initiator-Cookie 16 bytes. Copied from the Cookie_Request.

Responder-Cookie 16 bytes. A randomized value that identifies the exchange. The value MUST NOT be zero. See "Cookie Generation" for details.

Message 1

Counter 1 byte. Indicates the number of the current exchange. Must be greater than zero.

Offered-Schemes 4 or more bytes. A list of one or more Exchange-Schemes supported by the Responder, ordered from most to least preferable. See the "Basic Exchange-Schemes" for details.

Only one Scheme (#2) is required to be supported, and SHOULD be present in every Offered-Schemes list.

More than one of each kind of Scheme may be offered, but each is distinguished by its Size. The end of the list is indicated by the UDP Length.

3.3. Cookie Generation

The exact technique by which a Photuris party generates a cookie is implementation dependent. The method chosen must satisfy some basic requirements:

1. The cookie MUST depend on the specific parties. This prevents an attacker from obtaining a cookie using a real IP address and UDP port, and then using it to swamp the victim with requests from randomly chosen IP addresses or ports.
2. It MUST NOT be possible for anyone other than the issuing entity to generate cookies that will be accepted by that entity. This implies that the issuing entity will use local secret information in the generation and subsequent verification of a cookie. It must not be possible to deduce this secret information from any particular cookie.
3. The cookie generation and verification methods MUST be fast to thwart attacks intended to sabotage CPU resources.

A recommended technique is to use a cryptographic hashing function (such as MD5).

An incoming cookie can be verified at any time by regenerating it locally from values contained in the incoming datagram and the local secret random value.

3.3.1. Initiator Cookie

The Initiator secret value that affects its cookie SHOULD change for each new Photuris exchange, and is thereafter internally cached on a per Responder basis. This provides improved synchronization and protection against replay attacks.

An alternative is to cache the cookie instead of the secret value. Incoming cookies can be compared directly without the computational cost of regeneration.

It is recommended that the cookie be calculated over the secret value, the IP Source and Destination addresses, and the UDP Source and Destination ports.

Implementation Notes:

Although the recommendation includes the UDP Source port, this is very implementation specific. For example, it might not be included when the value is constant.

However, it is important that the implementation protect mutually suspicious users of the same machine from generating the same cookie.

3.3.2. Responder Cookie

The Responder secret value that affects its cookies MAY remain the same for many different Initiators. However, this secret SHOULD be changed periodically to limit the time for use of its cookies (typically each 60 seconds).

The Responder-Cookie SHOULD include the Initiator-Cookie. The Responder-Cookie MUST include the Counter (that is returned in the Cookie_Response). This provides improved synchronization and protection against replay attacks.

It is recommended that the cookie be calculated over the secret value, the IP Source and Destination addresses, its own UDP Destination port, the Counter, the Initiator-Cookie, and the currently Offered-Schemes.

The cookie is not cached per Initiator to avoid saving state during the initial Cookie Exchange. On receipt of a Value_Request (described later), the Responder regenerates its cookie for validation.

Once the Value_Response is sent (also described later), both Initiator and Responder cookies are cached to identify the exchange.

Implementation Notes:

Although the recommendation does not include the UDP Source port, this is very implementation specific. It might be successfully included in some variants.

However, it is important that the UDP Source port not be included when matching existing Photuris exchanges for determining the appropriate Counter.

The recommendation includes the Offered-Schemes to detect a dynamic change of scheme value between the Cookie_Response and

Value_Response.

Some mechanism MAY be needed to detect a dynamic change of pre-calculated Responder Exchange-Value between the Value_Response and Identity_Response. For example, change the secret value to render the cookie invalid, or explicitly mark the Photuris exchange state as expired.

4. Value Exchange

| | | |
|--|----|------------------|
| Initiator | | Responder |
| ===== | | ===== |
| Value_Request | -> | |
| pick scheme | | |
| offer value | | |
| offer attributes | | |
| | <- | Value_Response |
| | | offer value |
| | | offer attributes |
| [generate shared-secret from exchanged values] | | |

4.0.1. Send Value_Request

The Initiator generates an appropriate Exchange-Value for the Scheme-Choice. This Exchange-Value may be pre-calculated and used for multiple Responders.

The IP Destination for the Responder is examined, and the attributes available between the parties are listed in the Offered-Attributes.

The Initiator also starts a retransmission timer. If no valid Value_Response arrives within the time limit, the same Value_Request is retransmitted for the remaining number of Retransmissions.

When Retransmissions have been exceeded, if a Bad_Cookie or Resource_Limit message has been received during the exchange, the Initiator SHOULD begin the Photuris exchange again by sending a new Cookie_Request.

4.0.2. Receive Value_Request

The Responder validates the Responder-Cookie, the Counter, the Scheme-Choice, the Exchange-Value, and the Offered-Attributes.

- When an invalid/expired Responder-Cookie is detected, a Bad_Cookie message is sent.
- When too many SPI values are already in use for this particular peer, or too many concurrent exchanges are in progress, or some other resource limit is reached, a Resource_Limit message is sent.
- When an invalid Scheme-Choice is detected, or the Exchange-Value is obviously defective, or the variable length Offered-Attributes do not match the UDP Length, the message is silently discarded; the implementation SHOULD log the occurrence, and notify an operator as appropriate.

When the message is valid, the Responder sets its Exchange timer to the Exchange Timeout, and returns a Value_Response.

The Responder keeps a copy of the incoming Value_Request cookie pair, and its Value_Response. If a duplicate Value_Request is received, it merely resends its previous Value_Response, and takes no further action.

4.0.3. Send Value_Response

The Responder generates an appropriate Exchange-Value for the Scheme-Choice. This Exchange-Value may be pre-calculated and used for multiple Initiators.

The IP Source for the Initiator is examined, and the attributes available between the parties are listed in the Offered-Attributes.

Implementation Notes:

At this time, the Responder begins calculation of the shared-secret. Calculation of the shared-secret is executed in parallel to minimize delay.

This may take a substantial amount of time. The implementor should ensure that retransmission is not blocked by this calculation. This is not usually a problem, as retransmission timeouts typically exceed calculation time.

4.0.4. Receive Value_Response

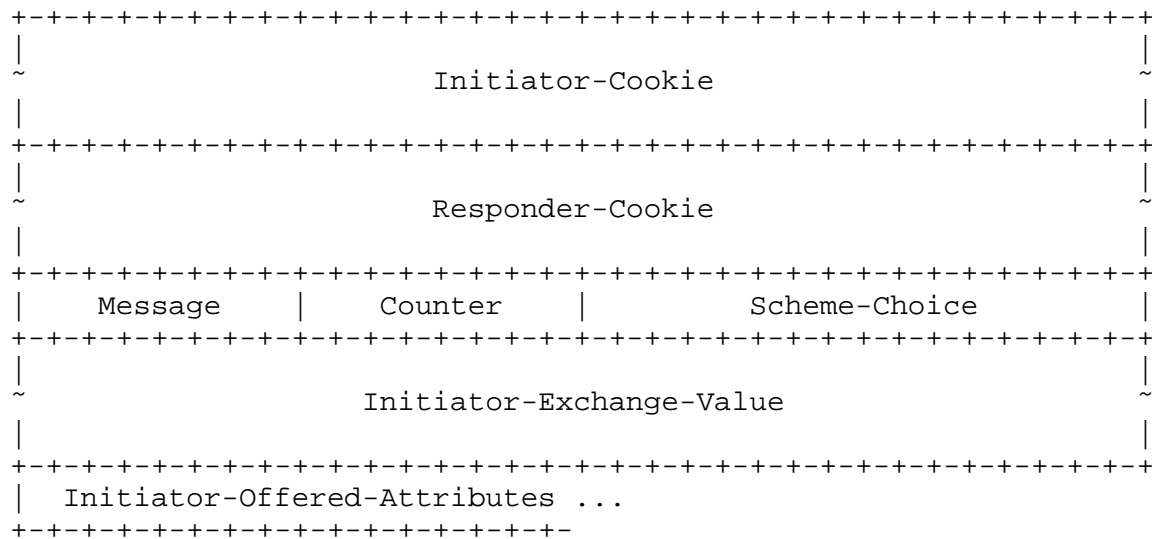
The Initiator validates the pair of Cookies, the Exchange-Value, and the Offered-Attributes.

- When an invalid/expired cookie is detected, the message is silently discarded.
- When the Exchange-Value is obviously defective, or the variable length Offered-Attributes do not match the UDP Length, the message is silently discarded; the implementation SHOULD log the occurrence, and notify an operator as appropriate.
- Once a valid message has been received, later Value_Responses with both matching cookies are also silently discarded, until a new Cookie_Request is sent.

When the message is valid, the Initiator begins its parallel computation of the shared-secret.

When the Initiator completes computation, it sends an Identity_Request to the Responder.

4.1. Value_Request



Initiator-Cookie 16 bytes. Copied from the Cookie_Response.

Responder-Cookie 16 bytes. Copied from the Cookie_Response.

Message 2

Counter 1 byte. Copied from the Cookie_Response.

Scheme-Choice 2 bytes. A value selected by the Initiator from the list of Offered-Schemes in the Cookie_Response.

Only the Scheme is specified; the Size will match the Initiator-Exchange-Value, and the Value(s) are implicit.

Initiator-Exchange-Value

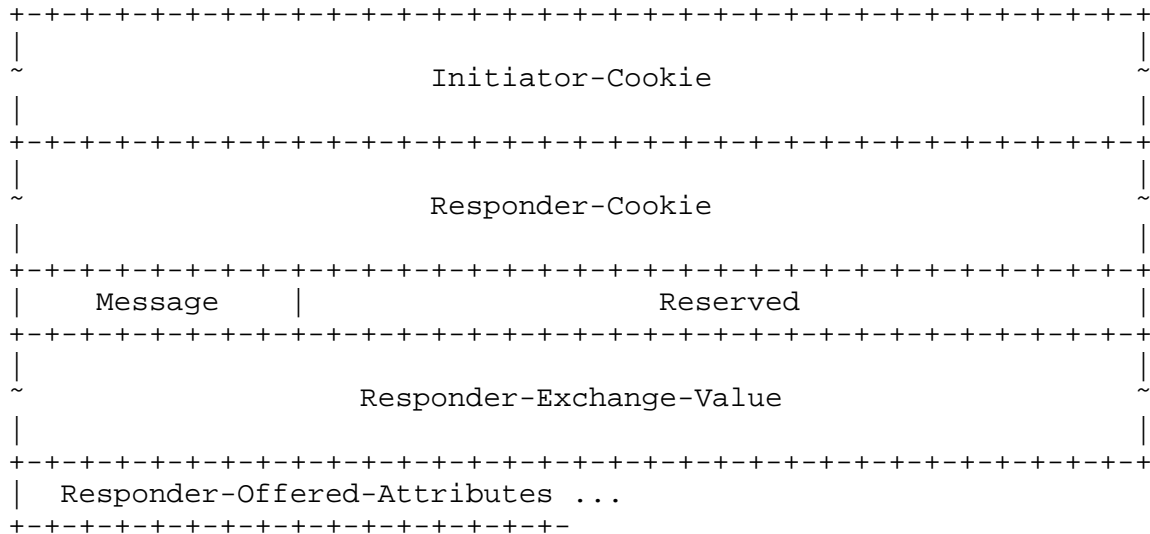
Variable Precision Integer. Provided by the Initiator for calculating a shared-secret between the parties. The Value format is indicated by the Scheme-Choice.

The field may be any integral number of bytes in length, as indicated by its Size field. It does not require any particular alignment. The 32-bit alignment shown is for convenience in the illustration.

Initiator-Offered-Attributes

4 or more bytes. A list of Security Parameter attributes supported by the Initiator.

The contents and usage of this list are further described in "Offered Attributes List". The end of the list is indicated by the UDP Length.

4.2. Value_Response

Initiator-Cookie 16 bytes. Copied from the Value_Request.

Responder-Cookie 16 bytes. Copied from the Value_Request.

Message 3

Reserved 3 bytes. For future use; MUST be set to zero when transmitted, and MUST be ignored when received.

Responder-Exchange-Value

Variable Precision Integer. Provided by the Responder for calculating a shared-secret between the parties. The Value format is indicated by the current Scheme-Choice specified in the Value_Request.

The field may be any integral number of bytes in

length, as indicated by its Size field. It does not require any particular alignment. The 32-bit alignment shown is for convenience in the illustration.

Responder-Offered-Attributes

4 or more bytes. A list of Security Parameter attributes supported by the Responder.

The contents and usage of this list are further described in "Offered Attributes List". The end of the list is indicated by the UDP Length.

4.3. Offered Attribute List

This list includes those attributes supported by the party that are available to the other party. The attribute formats are specified in the "Basic Attributes".

The list is composed of two or three sections: Identification-Attributes, Authentication-Attributes, and (optional) Encapsulation-Attributes. Within each section, the attributes are ordered from most to least preferable.

The first section of the list includes methods of identification. An Identity-Choice is selected from this list.

The second section of the list begins with "AH-Attributes" (#1). It includes methods of authentication, and other operational types.

The third section of the list begins with "ESP-Attributes" (#2). It includes methods of authentication, compression, encryption, and other operational types. When no Encapsulation-Attributes are offered, the "ESP-Attributes" attribute itself is omitted from the list.

Attribute-Choices are selected from the latter two sections of the list.

Support is required for the "MD5-IPMAC" (#5) attribute for both "Symmetric Identification" and "Authentication" and they SHOULD be present in every Offered-Attributes list.

Implementation Notes:

For example,

"MD5-IPMAC" (Symmetric Identification),
"AH-Attributes",
"MD5-IPMAC" (Authentication).

Since the offer is made by the prospective SPI User (sender), order of preference likely reflects the capabilities and engineering tradeoffs of a particular implementation.

However, the critical processing bottlenecks are frequently in the receiver. The SPI Owner (receiver) may express its needs by choosing a less preferable attribute.

The order may also be affected by operational policy and requested services for an application. Such considerations are outside the scope of this document.

The list may be divided into additional sections. These sections will always follow the ESP-Attributes section, and are indistinguishable from unrecognized attributes.

The authentication, compression, encryption and identification mechanisms chosen, as well as the encapsulation modes (if any), need not be the same in both directions.

5. Identification Exchange

| | | |
|-----------------------|----|-----------------------|
| Initiator | | Responder |
| ===== | | ===== |
| Identity_Request | -> | |
| make SPI | | |
| pick SPI attribute(s) | | |
| identify self | | |
| authenticate | | |
| make privacy key(s) | | |
| mask/encrypt message | | |
| | <- | Identity_Response |
| | | make SPI |
| | | pick SPI attribute(s) |
| | | identify self |
| | | authenticate |
| | | make privacy key(s) |
| | | mask/encrypt message |

[make SPI session-keys in each direction]

The exchange of messages is ordered, although the formats and meanings of the messages are identical in each direction. The messages are easily distinguished by the parties themselves, by examining the Message and Identification fields.

Implementation Notes:

The amount of time for the calculation may be dependent on the value of particular bits in secret values used in generating the shared-secret or identity verification. To prevent analysis of these secret bits by recording the time for calculation, sending of the Identity_Messages SHOULD be delayed until the time expected for the longest calculation. This will be different for different processor speeds, different algorithms, and different length variables. Therefore, the method for estimating time is implementation dependent.

Any authenticated and/or encrypted user datagrams received before the completion of identity verification can be placed on a queue pending completion of this step. If verification succeeds, the queue is processed as though the datagrams had arrived subsequent to the verification. If verification fails, the queue is discarded.

5.0.1. Send Identity_Request

The Initiator chooses an appropriate Identification, the SPI and SPILT, a set of Attributes for the SPI, calculates the Verification, and masks the message using the Privacy-Method indicated by the current Scheme-Choice.

The Initiator also starts a retransmission timer. If no valid Identity_Response arrives within the time limit, its previous Identity_Request is retransmitted for the remaining number of Retransmissions.

When Retransmissions have been exceeded, if a Bad_Cookie message has been received during the exchange, the Initiator SHOULD begin the Photuris exchange again by sending a new Cookie_Request.

5.0.2. Receive Identity_Request

The Responder validates the pair of Cookies, the Padding, the Identification, the Verification, and the Attribute-Choices.

- When an invalid/expired cookie is detected, a Bad_Cookie message is sent.
- After unmasking, when invalid Padding is detected, the variable length Attribute-Choices do not match the UDP Length, or an attribute was not in the Offered-Attributes, the message is silently discarded.
- When an invalid Identification is detected, or the message verification fails, a Verification_Failure message is sent.
- Whenever such a problem is detected, the Security Association is not established; the implementation SHOULD log the occurrence, and notify an operator as appropriate.

When the message is valid, the Responder sets its Exchange timer to the Exchange LifeTime (if this has not already been done for a previous exchange). When its parallel computation of the shared-secret is complete, the Responder returns an Identity_Response.

The Responder keeps a copy of the incoming Identity_Request values, and its Identity_Response. If a duplicate Identity_Request is received, it merely resends its previous Identity_Response, and takes no further action.

5.0.3. Send Identity_Response

The Responder chooses an appropriate Identification, the SPI and SPILT, a set of Attributes for the SPI, calculates the Verification, and masks the message using the Privacy-Method indicated by the current Scheme-Choice.

The Responder calculates the SPI session-keys in both directions.

At this time, the Responder begins the authentication and/or encryption of user datagrams.

5.0.4. Receive Identity_Response

The Initiator validates the pair of Cookies, the Padding, the Identification, the Verification, and the Attribute-Choices.

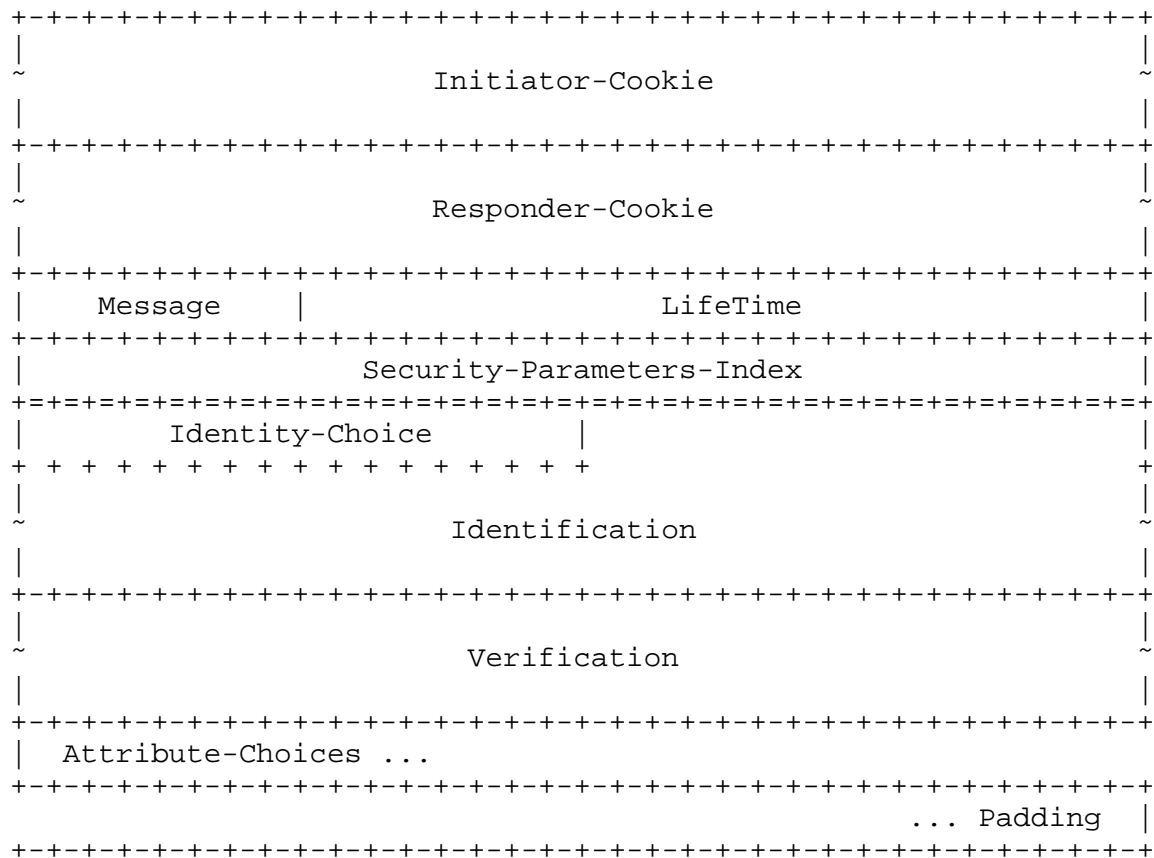
- When an invalid/expired cookie is detected, the message is silently discarded.
- After unmasking, when invalid Padding is detected, the variable length Attribute-Choices do not match the UDP Length, or an attribute was not in the Offered-Attributes, the message is silently discarded.
- When an invalid Identification is detected, or the message verification fails, a Verification_Failure message is sent.
- Whenever such a problem is detected, the Security Association is not established; the implementation SHOULD log the occurrence, and notify an operator as appropriate.
- Once a valid message has been received, later Identity_Responses with both matching cookies are also silently discarded, until a new Cookie_Request is sent.

When the message is valid, the Initiator sets its Exchange timer to the Exchange LifeTime (if this has not already been done for a previous exchange).

The Initiator calculates the SPI session-keys in both directions.

At this time, the Initiator begins the authentication and/or encryption of user datagrams.

5.1. Identity_Messages



Initiator-Cookie 16 bytes. Copied from the Value_Request.

Responder-Cookie 16 bytes. Copied from the Value_Request.

Message 4 (Request) or 7 (Response)

LifeTime 3 bytes. The number of seconds remaining before the indicated SPI expires.

When the SPI is zero, this field MUST be filled with a random non-zero value.

```
Security-Parameters-Index (SPI)
    4 bytes.  The SPI to be used for incoming
    communications.
```

When zero, indicates that no SPI is created in this

direction.

Identity-Choice 2 or more bytes. An identity attribute is selected from the list of Offered-Attributes sent by the peer, and is used to calculate the Verification.

The field may be any integral number of bytes in length, as indicated by its Length field. It does not require any particular alignment. The 16-bit alignment shown is for convenience in the illustration.

Identification Variable Precision Integer, or alternative format indicated by the Identity-Choice. See the "Basic Attributes" for details.

The field may be any integral number of bytes in length. It does not require any particular alignment. The 32-bit alignment shown is for convenience in the illustration.

Verification Variable Precision Integer, or alternative format indicated by the Identity-Choice. The calculation of the value is described in "Identity Verification".

The field may be any integral number of bytes in length. It does not require any particular alignment. The 32-bit alignment shown is for convenience in the illustration.

Attribute-Choices

0 or more bytes. When the SPI is non-zero, a list of attributes selected from the list of Offered-Attributes supported by the peer.

The contents and usage of this list are further described in "Attribute Choices List". The end of the list is indicated by the UDP Length after removing the Padding (UDP Length - last Padding value).

Padding 8 to 255 bytes. This field is filled up to at least a 128 byte boundary, measured from the beginning of the message. The number of pad bytes are chosen randomly.

In addition, when a Privacy-Method indicated by the

current Scheme-Choice requires the plaintext to be a multiple of some number of bytes (the block size of a block cipher), this field is adjusted as necessary to the size required by the algorithm.

Self-Describing-Padding begins with the value 1. Each byte contains the index of that byte. Thus, the final pad byte indicates the number of pad bytes to remove. For example, when the unpadded message length is 120 bytes, the padding values might be 1, 2, 3, 4, 5, 6, 7, and 8.

The portion of the message after the SPI field is masked using the Privacy-Method indicated by the current Scheme-Choice.

The fields following the SPI are opaque. That is, the values are set prior to masking (and optional encryption), and examined only after unmasking (and optional decryption).

5.2. Attribute Choices List

This list specifies the attributes of the SPI. The attribute formats are specified in the "Basic Attributes".

The list is composed of one or two sections: Authentication-Attributes, and/or Encapsulation-Attributes.

When sending from the SPI User to the SPI Owner, the attributes are processed in the order listed. For example,

```
"ESP-Attributes",  
"Deflate" (Compression),  
"XOR" (Encryption),  
"DES-CBC" (Encryption),  
"XOR" (Encryption),  
"AH-Attributes",  
"AH-Sequence",  
"MD5-IPMAC" (Authentication),
```

would result in ESP with compression and triple encryption (inside), and then AH authentication with sequence numbers (outside) of the ESP payload.

The SPI Owner will naturally process the datagram in the reverse order.

This ordering also affects the order of key generation. Both SPI

Owner and SPI User generate the keys in the order listed.

Implementation Notes:

When choices are made from the list of Offered-Attributes, it is not required that any Security Association include every kind of offered attribute in any single SPI, or that a separate SPI be created for every offered attribute.

Some kinds of attributes may be included more than once in a single SPI. The set of allowable combinations of attributes are dependent on implementation and operational policy. Such considerations are outside the scope of this document.

The list may be divided into additional sections. This can occur only when both parties recognize the affected attributes.

The authentication, compression, encryption and identification mechanisms chosen, as well as the encapsulation modes (if any), need not be the same in both directions.

5.3. Shared-Secret

A shared-secret is used in a number of calculations. Regardless of the internal representation of the shared-secret, when used in calculations it is in the same form as the Value part of a Variable Precision Integer:

- most significant byte first.
- bits used are right justified within byte boundaries.
- any unused bits are in the most significant byte.
- unused bits are zero filled.

The shared-secret does not include a Size field.

5.4. Identity Verification

These messages are authenticated using the Identity-Choice. The Verification value is calculated prior to masking (and optional encryption), and verified after unmasking (and optional decryption).

The Identity-Choice authentication function is supplied with two input values:

- the sender (SPI Owner) verification-key,
- the data to be verified (as a concatenated sequence of bytes).

The resulting output value is stored in the Verification field.

The Identity-Choice verification data consists of the following concatenated values:

- + the Initiator Cookie,
- + the Responder Cookie,
- + the Message, LifeTime and SPI fields,
- + the Identity-Choice and Identification,
- + the SPI User Identity Verification (response only),
- + the Attribute-Choices following the Verification field,
- + the Padding,
- + the SPI Owner TBV,
- + the SPI Owner Exchange-Value,
- + the SPI Owner Offered-Attributes,
- + the SPI User TBV,
- + the SPI User Exchange-Value,
- + the SPI User Offered-Attributes,
- + the Responder Offered-Schemes.

The TBV (Three Byte Value) consists of the Counter and Scheme-Choice fields from the Value_Request, or the Reserved field from the Value_Response, immediately preceding the associated Exchange-Value.

Note that the order of the Exchange-Value and Offered-Attributes fields is different in each direction, and the Identification and SPI fields are also likely to be different in each direction. Note also that the SPI User Identity Verification (from the Identity_Request) is present only in the Identity_Response.

If the verification fails, the users are notified, and a Verification_Failure message is sent, without adding any SPI. On success, normal operation begins with the authentication and/or encryption of user datagrams.

Implementation Notes:

This is distinct from any authentication method specified for the SPI.

The exact details of the Identification and verification-key included in the Verification calculation are dependent on the Identity-Choice, as described in the "Basic Attributes".

Each party may wish to keep their own trusted databases, such as the Pretty Good Privacy (PGP) web of trust, and accept only those identities found there. Failure to find the Identification in either an internal or external database results in the same

Verification_Failure message as failure of the verification computation.

The Exchange-Value data includes both the Size and Value fields. The Offered-Attributes and Attribute-Choices data includes the Attribute, Length and Value fields.

5.5. Privacy-Key Computation

Identification Exchange messages are masked using the Privacy-Method indicated by the current Scheme-Choice. Masking begins with the next field after the SPI, and continues to the end of the data indicated by the UDP Length, including the Padding.

The Scheme-Choice specified Key-Generation-Function is used to create a special privacy-key for each message. This function is calculated over the following concatenated values:

- + the SPI Owner Exchange-Value,
- + the SPI User Exchange-Value,
- + the Initiator Cookie,
- + the Responder Cookie,
- + the Message, LifeTime and SPI (or Reserved) fields,
- + the computed shared-secret.

Since the order of the Exchange-Value fields is different in each direction, and the Message, LifeTime and SPI fields are also different in each direction, the resulting privacy-key will usually be different in each direction.

When a larger number of keying-bits are needed than are available from one iteration of the specified Key-Generation-Function, more keying-bits are generated by duplicating the trailing shared-secret, and recalculating the function. That is, the first iteration will have one trailing copy of the shared-secret, the second iteration will have two trailing copies of the shared-secret, and so forth.

Implementation Notes:

This is distinct from any encryption method specified for the SPI.

The length of the Padding, and other details, are dependent on the Privacy-Method. See the "Basic Privacy-Method" list for details.

To avoid keeping the Exchange-Values in memory after the initial verification, it is often possible to pre-compute the function over the initial bytes of the concatenated data values for each

direction, and append the trailing copies of the shared-secret.

The Exchange-Value data includes both the Size and Value fields.

5.6. Session-Key Computation

Each SPI has one or more session-keys. These keys are generated based on the attributes of the SPI. See the "Basic Attributes" for details.

The Scheme-Choice specified Key-Generation-Function is used to create the SPI session-key for that particular attribute. This function is calculated over the following concatenated values:

- + the Initiator Cookie,
- + the Responder Cookie,
- + the SPI Owner generation-key,
- + the SPI User generation-key,
- + the message Verification field,
- + the computed shared-secret.

Since the order of the generation-keys is different in each direction, and the Verification field is also likely to be different in each direction, the resulting session-key will usually be different in each direction.

When a larger number of keying-bits are needed than are available from one iteration of the specified Key-Generation-Function, more keying-bits are generated by duplicating the trailing shared-secret, and recalculating the function. That is, the first iteration will have one trailing copy of the shared-secret, the second iteration will have two trailing copies of the shared-secret, and so forth.

Implementation Notes:

This is distinct from any privacy-key generated for the Photuris exchange. Different initialization data is used, and iterations are maintained separately.

The exact details of the Verification field and generation-keys that are included in the session-key calculation are dependent on the Identity-Choices, as described in the "Basic Attributes".

To avoid keeping the generation-keys in memory after the initial verification, it is often possible to pre-compute the function over the initial bytes of the concatenated data values for each direction, and append the trailing copies of the shared-secret.

When both authentication and encryption attributes are used for the same SPI, there may be multiple session-keys associated with the same SPI. These session-keys are generated in the order of the Attribute-Choices list.

6. SPI Messages

| | | |
|--|--|--|
| SPI User ===== SPI_Needed list SPI attribute(s) make validity key authenticate make privacy key(s) mask/encrypt message | -> <- | SPI Owner ===== SPI_Update make SPI pick SPI attribute(s) make SPI session-key(s) make validity key authenticate make privacy key(s) mask/encrypt message |
|--|--|--|

The exchange of messages is not related to the Initiator and Responder. Instead, either party may send one of these messages at any time. The messages are easily distinguished by the parties.

6.0.1. Send SPI_Needed

At any time after completion of the Identification Exchange, either party can send SPI_Needed. This message is sent when a prospective SPI User needs particular attributes for a datagram (such as confidentiality), and no current SPI has those attributes.

The prospective SPI User selects from the intersection of attributes that both parties have previously offered, calculates the Verification, and masks the message using the Privacy-Method indicated by the current Scheme-Choice.

6.0.2. Receive SPI_Needed

The potential SPI Owner validates the pair of Cookies, the Padding, the Verification, and the Attributes-Needed.

- When an invalid/expired cookie is detected, a Bad_Cookie message is sent.
- When too many SPI values are already in use for this particular peer, or some other resource limit is reached, a Resource_Limit message is sent.
- After unmasking, when invalid Padding is detected, the variable length Attributes-Needed do not match the UDP Length, or an attribute was not in the Offered-Attributes, the message is silently discarded.
- When the message verification fails, a Verification_Failure message is sent.
- Whenever such a problem is detected, the SPI is not established; the implementation SHOULD log the occurrence, and notify an operator as appropriate.

When the message is valid, the party SHOULD send SPI_Update with the necessary attributes.

If an existing SPI has those attributes, that SPI is returned in the SPI_Update with the remaining SPILT.

6.0.3. Send SPI_Update

At any time after completion of the Identification Exchange, either party can send SPI_Update. This message has effect in only one direction, from the SPI Owner to the SPI User.

The SPI Owner chooses the SPI and SPILT, a set of Attributes for the SPI, calculates the Verification, and masks the message using the Privacy-Method indicated by the current Scheme-Choice.

6.0.4. Receive SPI_Update

The prospective SPI User validates the pair of Cookies, the Padding, the Verification, and the Attributes-Needed.

- When an invalid/expired cookie is detected, a Bad_Cookie message

is sent.

- After unmasking, when invalid Padding is detected, the variable length Attribute-Choices do not match the UDP Length, an attribute was not in the Offered-Attributes, or the message modifies an existing SPI, the message is silently discarded.
- When the message verification fails, a Verification_Failure message is sent.
- Whenever such a problem is detected, the SPI is not established; the implementation SHOULD log the occurrence, and notify an operator as appropriate.

When the message is valid, further actions are dependent on the value of the LifeTime field, as described later.

6.0.5. Automated SPI_Updates

Each SPI requires replacement under several circumstances:

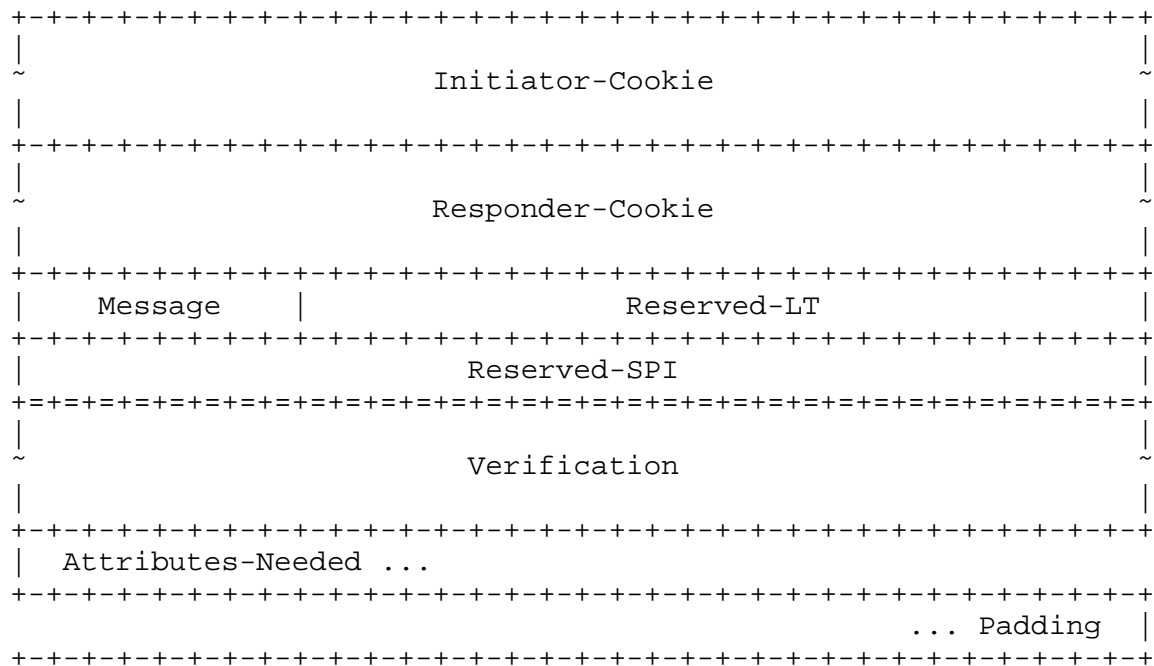
- the volume of data processed (inhibiting probability cryptanalysis),
- exhaustion of available anti-replay Sequence Numbers,
- and expiration of the LifeTime.

In general, a determination is made upon receipt of a datagram. If the transform specific processing finds that refreshment is needed, an automated SPI_Update is triggered.

In addition, automated SPI_Updates allow rapid SPI refreshment for high bandwidth applications in a high delay environment. The update messages flow in the opposite direction from the primary traffic, conserving bandwidth and avoiding service interruption.

When creating each SPI, the implementation MAY optionally set an Update Timeout (UTO); by default, to half the value of the LifeTime (SPILT/2). This time is highly dynamic, and adjustable to provide an automated SPI_Update long before transform specific processing. If no new Photuris exchange occurs within the time limit, and the current exchange state has not expired, an automated SPI_Update is sent.

6.1. SPI_Needed



Initiator-Cookie 16 bytes. Copied from the Value_Request.

Responder-Cookie 16 bytes. Copied from the Value_Request.

Message 8

Reserved-LT 3 bytes. For future use; MUST be filled with a random non-zero value when transmitted, and MUST be ignored when received.

Reserved-SPI 4 bytes. For future use; MUST be set to zero when transmitted, and MUST be ignored when received.

Verification Variable Precision Integer, or other format indicated by the current Scheme-Choice. The calculation of the value is described in "Validity Verification".

The field may be any integral number of bytes in length. It does not require any particular alignment. The 32-bit alignment shown is for convenience in the illustration.

Attributes-Needed

4 or more bytes. A list of two or more attributes, selected from the list of Offered-Attributes supported by the peer.

The contents and usage of this list are as previously described in "Attribute Choices List". The end of the list is indicated by the UDP Length after removing the Padding (UDP Length - last Padding value).

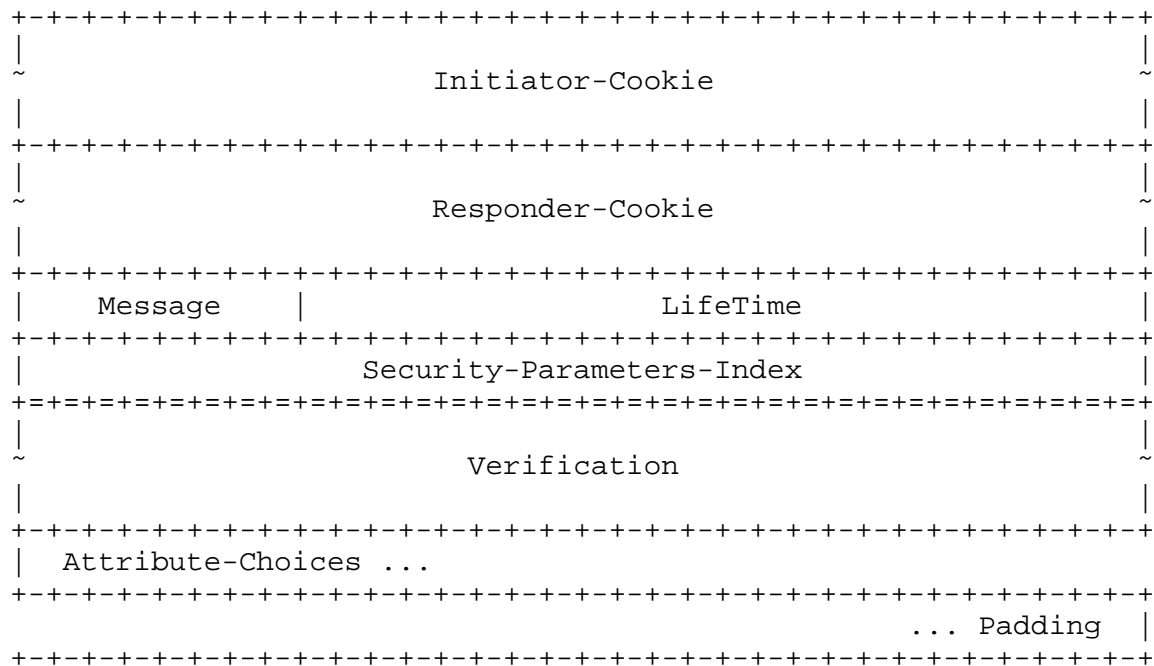
Padding

8 or more bytes. The message is padded in the same fashion specified for Identification Exchange messages.

The portion of the message after the SPI field is masked using the Privacy-Method indicated by the current Scheme-Choice.

The fields following the SPI are opaque. That is, the values are set prior to masking (and optional encryption), and examined only after unmasking (and optional decryption).

6.2. SPI_Update



Initiator-Cookie 16 bytes. Copied from the Value_Request.

Responder-Cookie 16 bytes. Copied from the Value_Request.

Message 9

LifeTime 3 bytes. The number of seconds remaining before the indicated SPI expires. The value zero indicates deletion of the indicated SPI.

Security-Parameters-Index (SPI)
4 bytes. The SPI to be used for incoming communications.

This may be a new SPI value (for creation), or an existing SPI value (for deletion). The value zero indicates special processing.

Verification Variable Precision Integer, or other format indicated by the current Scheme-Choice. The calculation of the value is described in "Validity Verification".

The field may be any integral number of bytes in length. It does not require any particular alignment. The 32-bit alignment shown is for convenience in the illustration.

Attribute-Choices

0 or more bytes. When the SPI and SPILT are non-zero, a list of attributes selected from the list of Offered-Attributes supported by the peer.

The contents and usage of this list are as previously described in "Attribute Choices List". The end of the list is indicated by the UDP Length after removing the Padding (UDP Length - last Padding value).

Padding 8 or more bytes. The message is padded in the same fashion specified for Identification Exchange messages.

The portion of the message after the SPI field is masked using the Privacy-Method indicated by the current Scheme-Choice.

The fields following the SPI are opaque. That is, the values are set prior to masking (and optional encryption), and examined only after unmasking (and optional decryption).

6.2.1. Creation

When the LifeTime is non-zero, and the SPI is also non-zero, the SPI_Update can be used to create a new SPI. When the SPI is zero, the SPI_Update is silently discarded.

The new session-keys are calculated in the same fashion as the Identity_Messages. Since the SPI value is always different than any previous SPI during the Exchange LifeTime of the shared-secret, the resulting session-keys will necessarily be different from all others used in the same direction.

No retransmission timer is necessary. Success is indicated by the peer use of the new SPI.

Should all creation attempts fail, eventually the peer will find that all existing SPIs have expired, and will begin the Photuris exchange again by sending a new Cookie_Request. When appropriate, this Cookie_Request MAY include a Responder-Cookie to retain previous party pairings.

6.2.2. Deletion

When the LifeTime is zero, the SPI_Update can be used to delete a single existing SPI. When the SPI is also zero, the SPI_Update will delete all existing SPIs related to this Security Association, and mark the Photuris exchange state as expired. This is especially useful when the application that needed them terminates.

No retransmission timer is necessary. This message is advisory, to reduce the number of ICMP Security Failures messages.

Should any deletion attempts fail, the peer will learn that the deleted SPIs are invalid through the normal ICMP Security Failures messages, and will initiate a Photuris exchange by sending a new Cookie_Request.

6.2.3. Modification

The SPI_Update cannot be used to modify existing SPIs, such as lengthen an existing SPI LifeTime, resurrect an expired SPI, or add/remove an Attribute-Choice.

On receipt, such an otherwise valid message is silently discarded.

6.3. Validity Verification

These messages are authenticated using the Validity-Method indicated by the current Scheme-Choice. The Verification value is calculated prior to masking (and optional encryption), and verified after unmasking (and optional decryption).

The Validity-Method authentication function is supplied with two input values:

- the sender (SPI Owner) verification-key,
- the data to be verified (as a concatenated sequence of bytes).

The resulting output value is stored in the Verification field.

The Validity-Method verification data consists of the following concatenated values:

- + the Initiator Cookie,
- + the Responder Cookie,
- + the Message, LifeTime and SPI (or Reserved) fields,
- + the SPI Owner Identity Verification,
- + the SPI User Identity Verification,
- + the Attribute-Choices following the Verification field,
- + the Padding.

Note that the order of the Identity Verification fields (from the Identity_Messages) is different in each direction, and the Message, LifeTime and SPI fields are also likely to be different in each direction.

If the verification fails, the users are notified, and a Verification_Failure message is sent, without adding or deleting any SPIs. On success, normal operation begins with the authentication and/or encryption of user datagrams.

Implementation Notes:

This is distinct from any authentication method specified for the SPI.

The Identity Verification data includes both the Size and Value fields. The Attribute-Choices data includes the Attribute, Length and Value fields.

7. Error Messages

These messages are issued in response to Photuris state loss or other problems. A message has effect in only one direction. No retransmission timer is necessary.

These messages are not masked.

The receiver checks the Cookies for validity. Special care MUST be taken that the Cookie pair in the Error Message actually match a pair currently in use, and that the protocol is currently in a state where such an Error Message might be expected. Otherwise, these messages could provide an opportunity for a denial of service attack. Invalid messages are silently discarded.

7.1. Bad_Cookie

For the format of the 33 byte message, see "Header Format". There are no additional fields.

Initiator-Cookie 16 bytes. Copied from the offending message.

Responder-Cookie 16 bytes. Copied from the offending message.

Message 10

This error message is sent when a Value_Request, Identity_Request, SPI_Needed, or SPI_Update is received, and the receiver specific Cookie is invalid or the associated exchange state has expired.

During the Photuris exchange, when this error message is received, it has no immediate effect on the operation of the protocol phases. Later, when Retransmissions have been exceeded, and this error message has been received, the Initiator SHOULD begin the Photuris exchange again by sending a new Cookie_Request with the Responder-Cookie and Counter updated appropriately.

When this error message is received in response to SPI_Needed, the exchange state SHOULD NOT be marked as expired, but the party SHOULD initiate a Photuris exchange by sending a new Cookie_Request.

When this error message is received in response to SPI_Update, the exchange state SHOULD NOT be marked as expired, and no further action is taken. A new exchange will be initiated later when needed by the peer to send authenticated and/or encrypted data.

Existing SPIs are not deleted. They expire normally, and are purged sometime later.

7.2. Resource_Limit

For the format of the 34 byte message, see "Cookie_Request". There are no additional fields.

Initiator-Cookie 16 bytes. Copied from the offending message.

Responder-Cookie 16 bytes. Copied from the offending message.

Special processing is applied to a Cookie_Request. When the offending message Responder-Cookie and Counter were both zero, and an existing exchange has not yet been purged, this field is replaced with the

Responder-Cookie from the existing exchange.

Message 11

Counter 1 byte. Copied from the offending message.

When zero, the Responder-Cookie indicates the Initiator of a previous exchange, or no previous exchange is specified.

When non-zero, the Responder-Cookie indicates the Responder to a previous exchange. This value is set to the Counter from the corresponding Cookie_Response.

This error message is sent when a Cookie_Request, Value_Request or SPI_Needed is received, and too many SPI values are already in use for that peer, or some other Photuris resource is unavailable.

During the Photuris exchange, when this error message is received in response to a Cookie_Request or Value_Request, the implementation SHOULD double the retransmission timeout (as usual) for sending another Cookie_Request or Value_Request. Otherwise, it has no immediate effect on the operation of the protocol phases. Later, when Retransmissions have been exceeded, and this error message has been received, the Initiator SHOULD begin the Photuris exchange again by sending a new Cookie_Request with the Responder-Cookie and Counter updated appropriately.

When this error message is received in response to SPI_Needed, the implementation SHOULD NOT send another SPI_Needed until one of the existing SPIs associated with this exchange is deleted or has expired.

7.3. Verification_Failure

For the format of the 33 byte message, see "Header Format". There are no additional fields.

Initiator-Cookie 16 bytes. Copied from the offending message.

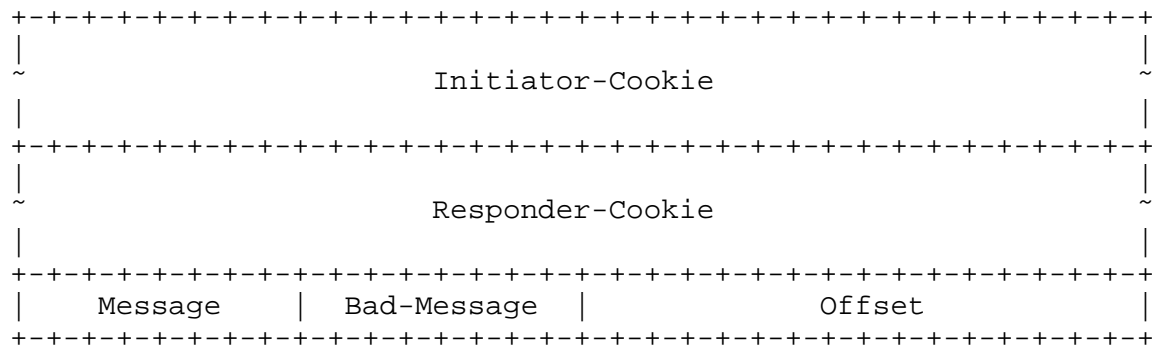
Responder-Cookie 16 bytes. Copied from the offending message.

Message 12

This error message is sent when an Identity_Message, SPI_Needed or SPI_Update is received, and verification fails.

When this error message is received, the implementation SHOULD log the occurrence, and notify an operator as appropriate. However, receipt has no effect on the operation of the protocol.

7.4. Message_Reject



Initiator-Cookie 16 bytes. Copied from the offending message.

Responder-Cookie 16 bytes. Copied from the offending message.

Message 13

| | |
|-------------|--|
| Bad-Message | 1 byte. Indicates the Message number of the offending message. |
|-------------|--|

Offset 2 bytes. The number of bytes from the beginning of the offending message where the unrecognized field starts. The minimum value is 32.

This error message is sent when an optional Message type is received that is not supported, or an optional format of a supported Message is not recognized.

When this error message is received, the implementation SHOULD log the occurrence, and notify an operator as appropriate. However, receipt has no effect on the operation of the protocol.

8. Public Value Exchanges

Photuris is based in principle on public-key cryptography, specifically Diffie-Hellman key exchange. Exchange of public D-H Exchange-Values based on private-secret values results in a mutual shared-secret between the parties. This shared-secret can be used on its own, or to generate a series of session-keys for authentication and encryption of subsequent traffic.

This document assumes familiarity with the Diffie-Hellman public-key algorithm. A good description can be found in [Schneier95].

8.1. Modular Exponentiation Groups

The original Diffie-Hellman technique [DH76] specified modular exponentiation. A public-value is generated using a generator (g), raised to a private-secret exponent (x), modulo a prime (p):

$$(g^{**x}) \bmod p.$$

When these public-values are exchanged between parties, the parties can calculate a shared-secret value between themselves:

$$(g^{**xy}) \bmod p.$$

The generator (g) and modulus (p) are established by the Scheme-Choice (see the "Basic Exchange-Schemes" for details). They are offered in the Cookie_Response, and one pair is chosen in the Value_Request.

The private exponents (x) and (y) are kept secret by the parties. Only the public-value result of the modular exponentiation with (x) or (y) is sent as the Initiator and Responder Exchange-Value.

These public-values are represented in single Variable Precision Integers. The Size of these Exchange-Values will match the Size of the modulus (p).

8.2. Moduli Selection

Each implementation proposes one or more moduli in its Offered-Schemes. Every implementation MUST support up to 1024-bit moduli.

For any particular Photuris node, these moduli need not change for significant periods of time; likely days or weeks. A background process can periodically generate new moduli.

For 512-bit moduli, current estimates would provide 64 (pessimistic) bit-equivalents of cryptographic strength.

For 1024-bit moduli, current estimates would range from 80 (pessimistic) through 98 (optimistic) bit-equivalents of cryptographic strength.

These estimates are used when choosing moduli that are appropriate for the desired Security Parameter attributes.

8.2.1. Bootstrap Moduli

Each implementation is likely to use a fixed modulus during its bootstrap, until it can generate another modulus in the background. As the bootstrap modulus will be widely distributed, and reused whenever the machine reinitializes, it SHOULD be a "safe" prime ($p = 2q+1$) to provide the greatest long-term protection.

Implementors are encouraged to generate their own bootstrap moduli, and to change bootstrap moduli in successive implementation releases.

8.2.2. Learning Moduli

As Photuris exchanges are initiated, new moduli will be learned from the Responder Offered-Schemes. The Initiator MAY cache these moduli for its own use.

Before offering any learned modulus, the implementation MUST perform at least one iteration of probable primality verification. In this fashion, many processors will perform verification in parallel as moduli are passed around.

When primality verification failures are found, the failed moduli SHOULD be retained for some (implementation dependent) period of time, to avoid re-learning and re-testing after subsequent exchanges.

8.3. Generator Selection

The generator (g) should be chosen such that the private-secret exponents will generate all possible public-values, evenly distributed throughout the range of the modulus (p), without cycling through a smaller subset. Such a generator is called a "primitive root" (which is trivial to find when p is "safe").

Only one generator (2) is required to be supported.

Implementation Notes:

One useful technique is to select the generator, and then limit the modulus selection sieve to primes with that generator:

- 2 when $p \pmod{24} = 11$.
- 3 when $p \pmod{12} = 5$.
- 5 when $p \pmod{10} = 3$ or 7 .

The required generator (2) improves efficiency in multiplication performance. It is usable even when it is not a primitive root, as it still covers half of the space of possible residues.

8.4. Exponent Selection

Each implementation generates a separate random private-secret exponent for each different modulus. Then, a D-H Exchange-Value is calculated for the given modulus, generator, and exponent.

This specification recommends that the exponent length be at least twice the desired cryptographic strength of the longest session-key needed by the strongest offered-attribute.

Based on the estimates in "Moduli Selection" (above):

For 512-bit moduli, exponent lengths of 128 bits (or more) are recommended.

For 1024-bit moduli, exponent lengths of 160 to 256 bits (or more) are recommended.

Although the same exponent and Exchange-Value may be used with several parties whenever the same modulus and generator are used, the exponent SHOULD be changed at random intervals. A background process can periodically destroy the old values, generate a new random private-secret exponent, and recalculate the Exchange-Value.

Implementation Notes:

The size of the exponent is entirely implementation dependent, is unknown to the other party, and can be easily changed.

Since these operations involve several time-consuming modular exponentiations, moving them to the "background" substantially improves the apparent execution speed of the Photuris protocol. It also reduces CPU loading sufficiently to allow a single public/private key-pair to be used in several closely spaced

Photuris executions, when creating Security Associations with several different nodes over a short period of time.

Other pre-computation suggestions are described in [BGMW93, LL94, Rooij94].

8.5. Defective Exchange Values

Some exponents do not qualify as secret. The exponent 0 will generate the Exchange-Value 1, and the exponent 1 will generate the Exchange-Value g. Small exponents will be easily visible and SHOULD be avoided where:

$$g^{**x} < p.$$

Depending on the structure of the moduli, certain exponents can be used for sub-group confinement attacks. For "safe" primes ($p = 2q+1$), these exponents are $p-1$ and $(p-1)/2$, which will generate the Exchange-Values 1 and $p-1$ respectively.

When an implementation chooses a random exponent, the resulting Exchange-Value is examined. If the Exchange-Value is represented in less than half the number of significant bits in the modulus, then a new random exponent MUST be chosen.

For 512-bit moduli, Exchange-Values of 2^{**256} or greater are required.

For 1024-bit moduli, Exchange-Values of 2^{**512} or greater are required.

In addition, if the resulting Exchange-Value is $p-1$, then a new random exponent MUST be chosen.

Upon receipt of an Exchange-Value that fails to meet the requirements, the Value Exchange message is silently discarded.

Implementation Notes:

Avoidance of small exponents can be assured by setting at least one bit in the most significant half of the exponent.

9. Basic Exchange-Schemes

Initial values are assigned as follows:

- (0) Reserved.
- (1) Reserved.
- (2) Implementation Required. Any modulus (p) with a recommended generator (g) of 2. When the Exchange-Scheme Size is non-zero, the modulus is contained in the Exchange-Scheme Value field in the list of Offered-Schemes.

An Exchange-Scheme Size of zero is invalid.

| | |
|-------------------------|-------------------|
| Key-Generation-Function | "MD5 Hash" |
| Privacy-Method | "Simple Masking" |
| Validity-Method | "MD5-IPMAC Check" |

This combination of features requires a modulus with at least 64-bits of cryptographic strength.

- (3) Exchange-Schemes 3 to 255 are intended for future well-known published schemes.
- (256) Exchange-Schemes 256 to 32767 are intended for vendor-specific unpublished schemes. Implementors wishing a number MUST request the number from the authors.
- (32768) Exchange-Schemes 32768 to 65535 are available for cooperating parties to indicate private schemes, regardless of vendor implementation. These numbers are not reserved, and are subject to duplication. Other criteria, such as the IP Source and Destination of the Cookie_Request, are used to differentiate the particular Exchange-Schemes available.

10. Basic Key-Generation-Function

10.1. MD5 Hash

MD5 [RFC-1321] is used as a pseudo-random-function for generating the key(s). The key(s) begin with the most significant bits of the hash. MD5 is iterated as needed to generate the requisite length of key material.

When an individual key does not use all 128-bits of the last hash, any remaining unused (least significant) bits of the last hash are discarded. When combined with other uses of key generation for the same purpose, the next key will begin with a new hash iteration.

11. Basic Privacy-Method

11.1. Simple Masking

As described in "Privacy-Key Computation", sufficient privacy-key material is generated to match the message length, beginning with the next field after the SPI, and including the Padding. The message is masked by XOR with the privacy-key.

12. Basic Validity-Method

12.1. MD5-IPMAC Check

As described in "Validity Verification", the Verification field value is the MD5 [RFC-1321] hash over the concatenation of

MD5(key, keyfill, data, datafill, key, md5fill)

where the key is the computed verification-key.

The keyfill and datafill use the same pad-with-length technique defined for md5fill. This padding and length is implicit, and does not appear in the datagram.

The resulting Verification field is a 128-bit Variable Precision Integer (18 bytes including Size). When used in calculations, the Verification data includes both the Size and Value fields.

13. Basic Attributes

Implementors wishing a number MUST request the number from the authors. Initial values are assigned as follows:

| Use | Type |
|------|--|
| - | 0* padding |
| - | 1* AH-Attributes |
| - | 2+ ESP-Attributes |
| AEI | 5* MD5-IPMAC |
| AEIX | 255+ Organizational |
| A | AH Attribute-Choice |
| E | ESP Attribute-Choice |
| I | Identity-Choice |
| X | dependent on list location |
| + | feature must be recognized even when not supported |
| * | feature must be supported (mandatory) |

Other attributes are specified in companion documents.

13.1. Padding

```

+---+---+---+---+---+
|   Attribute   |
+---+---+---+---+---+

```

Attribute 0

Each attribute may have value fields that are multiple bytes. To facilitate processing efficiency, these fields are aligned on integral modulo 8 byte (64-bit) boundaries.

Padding is accomplished by insertion of 1 to 7 Attribute 0 padding bytes before the attribute that needs alignment.

No padding is used after the final attribute in a list.

13.2. AH-Attributes

```

+-----+-----+
| Attribute | Length |
+-----+-----+

```

Attribute 1

Length 0

When a list of Attributes is specified, this Attribute begins the section of the list which applies to the Authentication Header (AH).

13.3. ESP-Attributes

```

+-----+-----+-----+
| Attribute | Length | PayloadType |
+-----+-----+-----+

```

Attribute 2

Length 1

PayloadType 1 byte. Indicates the contents of the ESP Transform Data field, using the IP Next Header (Protocol) value. Up-to-date values of the IP Next Header (Protocol) are specified in the most recent "Assigned Numbers" [RFC-1700].

For example, when encrypting an entire IP datagram, this field will contain the value 4, indicating IP-in-IP encapsulation.

When a list of Attributes is specified, this Attribute begins the section of the list which applies to the Encapsulating Security Payload (ESP).

When listed as an Offered-Attribute, the PayloadType is set to 255.

When selected as an Attribute-Choice, the PayloadType is set to the actual value to be used.

13.4. MD5-IPMAC

```

+---+---+---+---+---+---+---+---+---+---+
|  Attribute   |  Length   |
+---+---+---+---+---+---+---+---+---+---+

```

Attribute 5

Length 0

13.4.1. Symmetric Identification

When selected as an Identity-Choice, the immediately following Identification field contains an unstructured Variable Precision Integer. Valid Identifications and symmetric secret-keys are preconfigured by the parties.

There is no required format or content for the Identification value. The value may be a number or string of any kind. See "Use of Identification and Secrets" for details.

The symmetric secret-key (as specified) is selected based on the contents of the Identification field. All implementations **MUST** support at least 62 bytes. The selected symmetric secret-key **SHOULD** provide at least 64-bits of cryptographic strength.

As described in "Identity Verification", the Verification field value is the MD5 [RFC-1321] hash over the concatenation of:

```
MD5( key, keyfill, data, datafill, key, md5fill )
```

where the key is the computed verification-key.

The keyfill and datafill use the same pad-with-length technique defined for md5fill. This padding and length is implicit, and does not appear in the datagram.

The resulting Verification field is a 128-bit Variable Precision Integer (18 bytes including Size). When used in calculations, the Verification data includes both the Size and Value fields.

For both "Identity Verification" and "Validity Verification", the verification-key is the MD5 [RFC-1321] hash of the following concatenated values:

- + the symmetric secret-key,
- + the computed shared-secret.

For "Session-Key Computation", the symmetric secret-key is used directly as the generation-key.

Regardless of the internal representation of the symmetric secret-key, when used in calculations it is in the same form as the Value part of a Variable Precision Integer:

- most significant byte first.
- bits used are right justified within byte boundaries.
- any unused bits are in the most significant byte.
- unused bits are zero filled.

The symmetric secret-key does not include a Size field.

13.4.2. Authentication

May be selected as an AH or ESP Attribute-Choice, pursuant to [RFC-1828] et sequitur. The selected Exchange-Scheme SHOULD provide at least 64-bits of cryptographic strength.

As described in "Session-Key Computation", the most significant 384-bits (48 bytes) of the Key-Generation-Function iterations are used for the key.

Profile:

When negotiated with Photuris, the transform differs slightly from [RFC-1828].

The form of the authenticated message is:

```
MD5( key, keyfill, datagram, datafill, key, md5fill )
```

where the key is the SPI session-key.

The additional datafill protects against the (impractical) attack described in [P096]. The keyfill and datafill use the same pad-with-length technique defined for md5fill. This padding and length is implicit, and does not appear in the datagram.

13.5. Organizational

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Attribute | Length | OUI |
+-----+-----+-----+-----+-----+-----+-----+-----+
| ... | Kind | Value(s) ... |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Attribute 255

Length >= 4

When the Length is four, no Value(s) field is present.

OUI 3 bytes. The vendor's Organizationally Unique Identifier, assigned by IEEE 802 or IANA (see [RFC-1700] for contact details). The bits within the byte are in canonical order, and the most significant byte is transmitted first.

Kind 1 byte. Indicates a sub-type for the OUI. There is no standardization for this field. Each OUI implements its own values.

Value(s) 0 or more bytes. The details are implementation specific.

Some implementors might not need nor want to publish their proprietary algorithms and attributes. This OUI mechanism is available to specify these without encumbering the authors with proprietary number requests.

A. Automaton

An example automaton is provided to illustrate the operation of the protocol. It is incomplete and non-deterministic; many of the Good/Bad semantic decisions are policy-based or too difficult to represent in tabular form. Where conflicts appear between this example and the text, the text takes precedence.

The finite-state automaton is defined by events, actions and state transitions. Events include reception of external commands such as expiration of a timer, and reception of datagrams from a peer. Actions include the starting of timers and transmission of datagrams to the peer.

Events

DUI3 = Communication Administratively Prohibited

SF0 = Bad SPI

SF4 = Need Authentication

SF5 = Need Authorization

WC = Want Confidentiality

RCQ+ = Receive Cookie_Request (Good)

RCQ- = Receive Cookie_Request (Bad)

RCR+ = Receive Cookie_Response (Good)

RCR- = Receive Cookie_Response (Bad)

RVQ+ = Receive Value_Request (Good)

RVQ- = Receive Value_Request (Bad)

RVR+ = Receive Value_Response (Good)

RVR- = Receive Value_Response (Bad)

RIQ+ = Receive Identity_Request (Good)

RIQ- = Receive Identity_Request (Bad)

RIR+ = Receive Identity_Response (Good)

RIR- = Receive Identity_Response (Bad)

RUN+ = Receive SPI_Needed (Good)

RUN- = Receive SPI_Needed (Bad)

RUM+ = Receive SPI_Update (Good)

RUM- = Receive SPI_Update (Bad)

RBC = Receive Bad Cookie

RRL = Receive Resource Limit

RVF = Receive Verification Failure

RMR = Receive Message Reject

TO+ = Timeout with counter > 0

TO- = Timeout with counter expired
UTO = Update TimeOut
XTO = Exchange TimeOut

Actions

scq = Send Cookie_Request
scr = Send Cookie_Response

svq = Send Value_Request
svr = Send Value_Response

siq = Send Identity_Request
sir = Send Identity_Response

sum = Send SPI_Update

se* = Send error message (see text)
sbc = Send Bad Cookie
srl = Send Resource Limit
svf = Send Verification Failure

brto = Backoff Retransmission TimeOut
buto = Backoff Update TimeOut
rto = Set Retransmission TimeOut
uto = Set Update TimeOut
xto = Set Exchange TimeOut

log = log operator message

A.1. State Transition Table

States are indicated horizontally, and events are read vertically. State transitions and actions are represented in the form action/new-state. Multiple actions are separated by commas, and may continue on succeeding lines as space requires; multiple actions may be implemented in any convenient order. The state may be followed by a letter, which indicates an explanatory footnote. The dash ('-') indicates an illegal transition.

Initiator

| | 0 | 1 | 2 | 3 | 4 |
|------|-----------|-----------|-----------|-----------|-----------|
| | Initial | Cookie | CookieBad | Value | ValueBad |
| DU13 | rto,scq/1 | rto,scq/1 | rto,scq/1 | 3 | 4 |
| SF0 | rto,scq/1 | 1 | 2 | 3 | 4 |
| SF4 | rto,scq/1 | 1 | 2 | 3 | 4 |
| SF5 | rto,scq/1 | 1 | 2 | 3 | 4 |
| WC | rto,scq/1 | 1 | 2 | 3 | 4 |
| RCR+ | - | rto,svq/3 | rto,svq/3 | 3 | 4 |
| RCR- | 0 | 1 | 2 | 3 | 4 |
| RVR+ | - | - | - | rto,siq/5 | rto,siq/5 |
| RVR- | 0 | 1 | 2 | 3 | 4 |
| RIR+ | - | - | - | - | - |
| RIR- | 0 | 1 | 2 | 3 | 4 |
| RUN+ | - | - | - | - | - |
| RUN- | sbc/0 | sbc/1 | sbc/2 | sbc/3 | sbc/4 |
| RUM+ | - | - | - | - | - |
| RUM- | sbc/0 | sbc/1 | sbc/2 | sbc/3 | sbc/4 |
| RBC | - | - | - | 4 | 4 |
| RRL | - | brto/2 | brto/2 | brto/4 | brto/4 |
| RVF | - | - | - | - | - |
| RMR | - | - | - | - | - |
| TO+ | - | scq/1 | scq/2 | svq/3 | svq/4 |
| TO- | - | 0 | scq/1 | 0 | scq/1 |
| UTO | - | - | - | - | - |
| XTO | - | 0 | 0 | 0 | 0 |

Initiator

| | 5 | 6 | 8 |
|------|----------|-------------|-----------|
| | Identity | IdentityBad | Update |
| DU13 | 5 | 6 | 8 |
| SF0 | 5 | 6 | rto,scq/1 |
| SF4 | 5 | 6 | rto,scq/1 |
| SF5 | 5 | 6 | rto,scq/1 |
| WC | 5 | 6 | sun/8 |
| RCR+ | 5 | 6 | 8 |
| RCR- | 5 | 6 | 8 |
| RVR+ | 5 | 6 | 8 |
| RVR- | 5 | 6 | 8 |
| RIR+ | uto/8 | uto/8 | 8 |
| RIR- | svf/5 | svf/6 | 8 |
| RUN+ | - | - | sum/8 |
| RUN- | sbc/5 | sbc/6 | se*/8 |
| RUM+ | - | - | 8 |
| RUM- | sbc/5 | sbc/6 | se*/8 |
| RBC | 6 | 6 | rto,scq/1 |
| RRL | 5 | 6 | buto/8 |
| RVF | log/5 | log/6 | log/8 |
| RMR | log/5 | log/6 | log/8 |
| TO+ | sim/5 | sim/6 | - |
| TO- | 0 | scq/1 | - |
| UTO | - | - | sum/8 |
| XTO | 0 | 0 | 0 |

Responder

| | 0 Initial | 7 Ready | 8 Update |
|------|--------------|------------|-------------|
| WC | - | 7 | sun/8 |
| RCQ+ | scr/0 | scr/7 | scr/8 |
| RCQ- | srl/0 | srl/7 | srl/8 |
| RVQ+ | xto,svr/7 | svr/7 | svr/8 |
| RVQ- | sbc/0 | sbc/7 | sbc/8 |
| RIQ+ | - | uto,sir/8 | sir/8 |
| RIQ- | sbc/0 | se*/7 | se*/8 |
| RUN+ | - | - | sum/8 |
| RUN- | sbc/0 | sbc/7 | se*/8 |
| RUM+ | - | - | 8 |
| RUM- | sbc/0 | sbc/7 | se*/8 |
| RBC | - | 7 | rto,scq/1 |
| RRL | - | - | buto/8 |
| RVF | - | - | log/8 |
| RMR | - | - | log/8 |
| UTO | - | - | sum/8 |
| XTO | - | 0 | 0 |

A.2. States

Following is a more detailed description of each automaton state.

The "Bad" version of a state is to indicate that the Bad_Cookie or Resource_Limit message has been received.

A.2.1. Initial

The Initial state is fictional, in that there is no state between the parties.

A.2.2. Cookie

In the Cookie state, the Initiator has sent a Cookie_Request, and is waiting for a Cookie_Response. Both the Restart and Exchange timers are running.

Note that the Responder has no Cookie state.

A.2.3. Value

In the Value state, the Initiator has sent its Exchange-Value, and is waiting for an Identity_Message. Both the Restart and Exchange timers are running.

A.2.4. Identity

In the Identity state, the Initiator has sent an Identity_Request, and is waiting for an Identity_Response in reply. Both the Restart and Exchange timers are running.

A.2.5. Ready

In the Ready state, the Responder has sent its Exchange-Value, and is waiting for an Identity_Message. The Exchange timer is running.

A.2.6. Update

In the Update state, each party has concluded the Photuris exchange, and is unilaterally updating expiring SPIs until the Exchange LifeTime expires. Both the Update and Exchange timers are running.

B. Use of Identification and Secrets

Implementation of the base protocol requires support for operator configuration of participant identities and associated symmetric secret-keys.

The form of the Identification and Secret fields is not constrained to be a readable string. In addition to a simpler quoted string configuration, an implementation MUST allow configuration of an arbitrary stream of bytes.

B.1. Identification

Typically, the Identification is a user name, a site name, a Fully Qualified Domain Name, or an email address which contains a user name and a domain name. Examples include:

```
user
node.site.
user@node.site.
rcmd@node.site.
"Mundane Name" <user@node.site>
```

There is no requirement that the domain name match any of the particular IP addresses in use by the parties.

B.2. Group Identity With Group Secret

A simple configuration approach could use a single Identity and Secret, distributed to all the participants in the trusted group. This might be appropriate between routers under a single administration comprising a Virtual Private Network over the Internet.

Nota Bene:

The passwords used in these examples do not meet the "MD5-IPMAC Symmetric Identification" recommendation for at least 64-bits of cryptographic strength.

The administrator configures each router with the same username and password:

```
identity local "Tiny VPN 1995 November" "abracadabra"
identity remote "Tiny VPN 1995 November" "abracadabra"
```

When the Initiator sends its Identity_Request, the SPI Owner

Identification field is "Tiny VPN 1995 November" and the SPI Owner secret-key is "abracadabra".

When the Responder sends its Identity_Response, the SPI Owner Identification field is "Tiny VPN 1995 November" and the SPI Owner secret-key is "abracadabra". The SPI User Identification is "Tiny VPN 1995 November" (taken from the request), and the SPI User secret-key is "abracadabra".

Note that even in the face of implementations with very poor random number generation yielding the same random numbers for both parties at every step, and with this completely identical configuration, the addition of the SPI User Verification field in the response calculation is highly likely to produce a different Verification value (see "Identity Verification"). In turn, the different Verification values affect the calculation of SPI session-keys that are highly likely to be different in each direction (see "Session-Key Computation").

B.3. Multiple Identities With Group Secrets

A more robust configuration approach could use a separate Identity and Secret for each party, distributed to the participants in the trusted group. This might be appropriate for authenticated firewall traversal.

An administrator has one or more networks, and a number of mobile users. It is desirable to restrict access to authorized external users. The example boundary router is 10.0.0.1.

The administrator gives each user a different username and password, together with a group username and password for the router.

The administrator configures (in part):

```
identity local "199511@router.site" "FalDaRah"
identity remote "Happy_Wanderer@router.site" "FalDaRee"
```

Each mobile user adds commands to tunnel and authenticate.

```
route addprivate 10.0.0.0/8 tunnel 10.0.0.1
secure 10.0.0.1 authenticate-only
identity local "Happy_Wanderer@router.site" "FalDaRee"
identity remote "199511@router.site" "FalDaRah"
identity remote "199512@router.site" "FalDaHaHaHaHaHaHa"
```

When the mobile Initiator sends its Identity_Request, the SPI Owner

Identification field is "Happy_Wanderer@router.site" and the SPI Owner secret-key is "FalDaRee".

When the firewall Responder sends its Identity_Response, the SPI Owner Identification field is "199511@router.site" and the SPI Owner secret-key is "FalDaRah". The SPI User Identification field is "Happy_Wanderer@router.site" (taken from the request), and the SPI User secret-key is "FalDaRee".

In this example, the mobile user is already prepared for a monthly password changeover, where the router might identify itself as "199512@router.site".

B.4. Multiple Identities With Multiple Secrets

Greater security might be achieved through configuration of a pair of secrets between each party. As before, one secret is used for initial contact to any member of the group, but another secret is used between specific parties. Compromise of one secret or pair of secrets does not affect any other member of the group. This might be appropriate between the routers forming a boundary between cooperating Virtual Private Networks that establish local policy for each VPN member access.

One administrator configures:

```
identity local "Apple" "all for one"
identity local "Apple-Baker" "Apple to Baker" "Baker"
identity remote "Baker" "one for all"
identity remote "Baker-Apple" "Baker to Apple"
```

Another configures:

```
identity local "Baker" "one for all"
identity local "Baker-Apple" "Baker to Apple" "Apple"
identity remote "Apple" "all for one"
identity remote "Apple-Baker" "Apple to Baker"
```

When the Initiator sends its Identity_Request, the SPI Owner Identification field is "Apple" and the SPI Owner secret-key is "all for one".

When the Responder sends its Identity_Response, finding that the special pairing exists for "Apple" (in this example, indicated by a third field), the SPI Owner Identification field is "Baker-Apple" and the SPI Owner secret-key is "Baker to Apple". The SPI User Identification is "Apple" (taken from the request), and the SPI User

secret-key is "all for one".

Operational Considerations

The specification provides only a few configurable parameters, with defaults that should satisfy most situations.

Retransmissions

Default: 3.

Initial Retransmission TimeOut (IRTO)

Default: 5 seconds.

Exchange TimeOut (ETO)

Default: 30 seconds. Minimum: Retransmissions * IRTO.

Exchange LifeTime (ELT)

Default: 30 minutes. Minimum: 2 * ETO.

SPI LifeTime (SPILT)

Default: 5 minutes. Minimum: 3 * ETO.

Each party configures a list of known identities and symmetric secret-keys.

In addition, each party configures local policy that determines what access (if any) is granted to the holder of a particular identity. For example, the party might allow anonymous FTP, but prohibit Telnet. Such considerations are outside the scope of this document.

Security Considerations

Photuris was based on currently available tools, by experienced network protocol designers with an interest in cryptography, rather than by cryptographers with an interest in network protocols. This specification is intended to be readily implementable without requiring an extensive background in cryptology.

Therefore, only minimal background cryptologic discussion and rationale is included in this document. Although some review has been provided by the general cryptologic community, it is anticipated that design decisions and tradeoffs will be thoroughly analysed in subsequent dissertations and debated for many years to come.

Cryptologic details are reserved for separate documents that may be more readily and timely updated with new analysis.

History

The initial specification of Photuris, now called version 1 (December 1994 to March 1995), was based on a short list of design requirements, and simple experimental code by Phil Karn. Only one modular exponentiation form was used, with a single byte index of pre-specified group parameters. The transform attributes were selected during the public value exchange. Party privacy was protected in the identification signature exchange with standard ESP transforms.

Upon submission for review by the IP Security Working Group, a large number of features were demanded. A mere 254 future group choices were not deemed enough; it was expanded to two bytes (and renamed schemes), and was expanded again to carry variable parameters. The transform attributes were made variable length to accomodate optional parameters. Every other possible parameter was made negotiable. Some participants were unable to switch modes on the UDP sockets to use standard ESP transforms for only some messages, and party privacy was integrated into the protocol. The message headers were reorganized, and selection of transform attributes was delayed until the identification exchange. An additional update message phase was added.

Version 2 (July 1995 to December 1995) specification stability was achieved in November 1995 by moving most parameters into separate documents for later discussion, and leaving only a few mandatory features in the base specification. Within a month, multiple interoperable implementations were produced.

Unfortunately, in a fit of demagoguery, the IP Security Working Group decided in a straw poll to remove party privacy protection, and the Working Group chair terminated the meeting without allowing further discussion. Because the identification exchange messages required privacy to function correctly, the messages were reorganized again. Party privacy and other optional schemes were split into a separate document.

The implementors established a separate discussion group. Version 3 (April 1996 to June 1997) enjoyed a long period of specification stability and multiple implementations on half a dozen platforms.

Meanwhile, the IP Security Working Group has developed a competing specification with large numbers of negotiable parameters. Also, the PPP Extensions Working Group has deployed link security transforms.

Version 4 (July 1997 onward) attempts to maintain a semblance of interface compatibility with these other efforts. Minor changes are

specified in transform padding format and key generation. More than one value is permitted per scheme, giving greater latitude in choice for future extensions. The opportunity is taken to return party privacy to the base document, and make small semantic changes in automated updates and error recovery. All ESP transform attributes are moved to separate documents, to (hopefully) avoid future incompatible changes to the base document.

Acknowledgements

Thou shalt make no law restricting the size of integers that may be multiplied together, nor the number of times that an integer may be multiplied by itself, nor the modulus by which an integer may be reduced. [Prime Commandment]

Phil Karn was principally responsible for the design of the protocol phases, particularly the "cookie" anti-clogging defense, developed the initial testing implementation, and provided much of the design rationale text (now removed to a separate document).

William Simpson was responsible for the packet formats and attributes, additional message types, editing and formatting. All such mistakes are his responsibility.

This protocol was later discovered to have many elements in common with the Station-To-Station authentication protocol [DOW92].

Angelos Keromytis developed the first completely independent implementation (circa October 1995). Also, he suggested the cookie exchange rate limitation counter.

Paul C van Oorschot suggested signing both the public exponents and the shared-secret, to provide an authentication-only version of identity verification. Also, he provided text regarding moduli, generator, and exponent selection (now removed to a separate document).

Hilarie Orman suggested adding secret "nonces" to session-key generation for asymmetric public/private-key identity methods (now removed to a separate document), and provided extensive review of the protocol details.

Bart Preneel and Paul C van Oorschot in [PO96] recommended padding between the data and trailing key when hashing for authentication.

Niels Provos developed another independent implementation (circa May 1997), ported to AIX, Linux, OpenBSD, and Solaris. Also, he made

suggestions regarding automated update, and listing multiple moduli per scheme.

Bill Sommerfeld suggested including the authentication symmetric secret-keys in the session-key generation, and using the Cookie values on successive exchanges to provide bi-directional user-oriented keying (now removed to a separate document).

Oliver Spatscheck developed the second independent implementation (circa December 1995) for the Xkernel.

International interoperability testing between early implementors provided the impetus for many of the implementation notes herein, and numerous refinements in the semantics of the protocol messages.

Randall Atkinson, Steven Bellovin, Wataru Hamada, James Hughes, Brian LaMacchia, Cheryl Madson, Lewis McCarthy, Perry Metzger, Bob Quinn, Ron Rivest, Rich Schroepel, and Norman Shulman provided useful critiques of earlier versions of this document.

Special thanks to the Center for Information Technology Integration (CITI) for providing computing resources.

References

- [BGMW93] E. Brickell, D. Gordon, K. McCurley, and D. Wilson, "Fast Exponentiation with Precomputation (Extended Abstract)", *Advances in Cryptology -- Eurocrypt '92*, Lecture Notes in Computer Science 658 (1993), Springer-Verlag, 200-207.
- Also U.S. Patent #5,299,262, E.F. Brickell, D.M. Gordon, K.S. McCurley, "Method for exponentiating in cryptographic systems", 29 Mar 1994.
- [DH76] Diffie, W., and Hellman, H.E., "New Directions in Cryptography", *IEEE Transactions on Information Theory*, v IT-22 n 6 pp 644-654, November 1976.
- [DOW92] Whitfield Diffie, Paul C van Oorshot, and Michael J Wiener, "Authentication and Authenticated Key Exchanges", *Designs, Codes and Cryptography*, v 2 pp 107-125, Kluwer Academic Publishers, 1992.
- [Firefly] "Photuris" is the latin name for the firefly. "Firefly" is in turn the name for the USA National Security Administration's (classified) key exchange protocol for the STU-III secure telephone. Informed speculation has

it that Firefly is based on very similar design principles.

- [LL94] Lim, C.H., Lee, P.J., "More flexible exponentiation with precomputation", Advances in Cryptology -- Crypto '94, Lecture Notes in Computer Science 839 (1994), Springer-Verlag, pages 95-107.
- [Prime Commandment] A derivation of an apocryphal quote from the usenet list sci.crypt.
- [PO96] Bart Preneel, and Paul C van Oorshot, "On the security of two MAC algorithms", Advances in Cryptology -- Eurocrypt '96, Lecture Notes in Computer Science 1070 (May 1996), Springer-Verlag, pages 19-32.
- [RFC-768] Postel, J., "User Datagram Protocol", STD 6, USC/Information Sciences Institute, August 1980.
- [RFC-791] Postel, J., "Internet Protocol", STD 5, USC/Information Sciences Institute, September 1981.
- [RFC-1321] Rivest, R., "The MD5 Message-Digest Algorithm", MIT Laboratory for Computer Science, April 1992.
- [RFC-1700] Reynolds, J., and Postel, J., "Assigned Numbers", STD 2, USC/Information Sciences Institute, October 1994.
- [RFC-1812] Baker, F., Editor, "Requirements for IP Version 4 Routers", Cisco Systems, June 1995.
- [RFC-1828] Metzger, P., Simpson, W., "IP Authentication using Keyed MD5", July 1995.
- [RFC-1829] Karn, P., Metzger, P., Simpson, W., "The ESP DES-CBC Transform", July 1995.
- [RFC-2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, Harvard University, March 1997.
- [RFC-2521] Karn, P., and Simpson, W., "ICMP Security Failures Messages", March 1999.
- [Rooij94] P. de Rooij, "Efficient exponentiation using precomputation and vector addition chains", Advances in Cryptology -- Eurocrypt '94, Lecture Notes in Computer

Science, Springer-Verlag, pages 403-415.

[Schneier95]

Schneier, B., "Applied Cryptography Second Edition", John Wiley & Sons, New York, NY, 1995. ISBN 0-471-12845-7.

Contacts

Comments about this document should be discussed on the photuris@adk.gr mailing list.

Questions about this document can also be directed to:

Phil Karn
Qualcomm, Inc.
6455 Lusk Blvd.
San Diego, California 92121-2779

karn@qualcomm.com
karn@unix.ka9q.ampr.org (preferred)

William Allen Simpson
DayDreamer
Computer Systems Consulting Services
1384 Fontaine
Madison Heights, Michigan 48071

wsimpson@UMich.edu
wsimpson@GreenDragon.com (preferred)

Full Copyright Statement

Copyright (C) The Internet Society (1999). Copyright (C) Philip Karn and William Allen Simpson (1994-1999). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards (in which case the procedures for copyrights defined in the Internet Standards process must be followed), or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING (BUT NOT LIMITED TO) ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

