Network Working Group
Internet Draft
Expires April, 1999

L.A. Sanchez, BBN/GTEI
M.N. Condell, BBN/GTEI
November 18, 1998

Security Policy System
<draft-ietf-ipsec-sps-00.txt>

Status of this Memo

Abstract

This document describes a distributed system that provides the mechanisms needed for discovering, accessing and processing security policy information of hosts, subnets or networks of a security domain. In this system policy clients and servers exchange information using the Security Policy Protocol. The protocol defines how the policy information is exchanged, processed, and protected by clients and servers. The system accommodates topology changes, hence policy changes, rather easily without the scalability constraints imposed by static reconfiguration of each client. The protocol is extensible and flexible. It allows the exchange of complex policy objects between clients and servers.

Table of Contents

1. Introduction 3
1.1 Terms and Technical Definitions. 3
1.1.1 Requirements Terminology 3
1.1.2 Technical Definitions. 3
1.2 Requirements 4
2. Overview 5
3. Policy Master File 8
4. Policy Database. 9
4.1 Local Policy Database. 9
4.2 Cache Database 10
4.3 Security Domain Database 10
5. Security Policy Protocol (SPP) 11
5.1 SPP Message Format 12
5.2 SPP Payloads 14
5.2.1 Query Payload. 14
5.2.2 Record Payload 15
5.2.3 Signature Payload. 16
5.3 SPP Messages 17
5.3.1 Query Messages 17
5.3.2 Reply Messages 17
5.3.3 Policy Messages. 18
5.3.4 Policy Acknowledgment Messages 18
5.3.4 Transfer Messages. 18
5.3.5 KeepAlive Messages 19
6. Policy Attribute Encoding. 19
7. Policy Queries 21
7.1 Security Gateway Query 21
7.2 COMSEC Query 21
7.3 Certificate Query. 22
8. Policy Records 24
8.1 Security Gateway Record. 24
8.2 COMSEC Record. 25
8.3 Security Association Record. 26
8.4 Policy Server Record 28
8.5 Certificate Record 29
9. SPP Message Processing 30

9.1 General Message Processing 30
9.2 Query Message Processing 30
9.3 Reply Message Processing 33
9.4 Policy Message Processing. 35
9.5 Policy Acknowledgment Message Processing 37
9.6 KeepAlive Message Processing 38
10. Policy Decorrelation Process 39
10.1 Decorrelation Algorithm 40
11. Policy Resolution Process. 42
12. Usage of SPS with IPSec. 43
Acknowledgments. 46
References 46
Appendix A. DATA_TYPE Definitions. 47
Appendix B. Decorrelation Example. 64
Disclaimer 69
Author Information 69

1. Introduction

The Security Policy System (SPS) is a distributed database of security policy information. It provides the mechanisms needed for discovering, accessing and processing security policy information of hosts, subnets or networks of a security domain.

In SPS, each security domain has a master file that uniquely defines a security domain by its network resources (hosts, subnets, networks) and the policies to access them. Security policies and the entire domain definition is stored in the Master File.

These policies reside in a database local to the security domain. The Policy Server provides access to these policies to client applications requesting policy information for a particular host, subnet or network. SPS provides mechanisms to limit the access of policy records to authorized hosts and/or users. SPS also provides procedures to validate the authenticity and integrity of the policy information exchanged between security domains.

Policy Clients generate query databases of different security domains using the Security Policy Protocol. SPS provides a standard set of query types for policy information. New query types can be incorporated as needed. Policy Servers reply to these requests after determining the validity of the request.

Since security policies could be highly restrictive and relative to the intended communication between a source and destination, it is possible that the replies provided by a Policy Server would differ depending on the identity of the requestor, making caching of the policies a plausible but complicated process. SPS

defines an algorithm that makes the caching of security policies a feasible task.

SPS defines the format of the security policy records and the encoding of these before transmission. SPS also defines a standard and extensible message format. SPS specifies the message processing required at the Policy Servers and Policy Clients.

1.1 Terms and Technical Definitions

1.1.1 Requirements Terminology

Keywords "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT" and "MAY" that appear in this document are to be interpreted as described in [Bra97].

1.1.2 Technical Definitions

Security Gateway

A security gateway refers to an intermediate system that implements IPSec protocols. For example, a router or a firewall implementing IPSec is a security gateway.

Security Domain

A set of communicating entities and resources that share a common security policy enforced at one or more enforcement agents or at an individual host. The definition of security domain applies to networks protected by security gateways as well as to single hosts, since a host may be the enforcer of its own policies. Security domains could exist inside other security domains.

Security Association (SA)

A simplex "connection" that affords security services to the traffic it carries. Two types of security associations are defined: transport mode and tunnel mode. A transport mode SA is a security association between two hosts. A tunnel mode SA is essentially an SA applied to an IP tunnel. For transit traffic, whenever either end of a security

association is a security gateway, the SA MUST be tunnel mode.

Security Association Bundle

A group of security associations that are used to protect communications that share a common endpoint. For example, all the SAs that a particular host needs to use to communicate with another host, including any SAs that host itself needs with intermediate security gateways.

1.2 Requirements

The architectural requirements of the Security Policy System are as follows:

- Gateway Discovery

  SPS must be able to determine a set of necessary security gateways through which a message must travel to complete a communication on a single path between two hosts.

- Identity Verification

  SPS must allow hosts to verify the identities of security gateways and other hosts with which they are communicating. It must also be able to verify that a gateway that claims to represent a particular host actually does have the authority to represent that host.

- Require no changes to security protocols

  SPS must not require changes, additions or modifications to security protocols that use it.

- Key Management Protocol Independence

  SPS must be independent of any particular key management protocol.

- Minimal Exterior Infrastructure Dependency

  SPS SHOULD NOT depend upon an exterior infrastructure, although implementations may use an exterior infrastructure. For example, public keys may be distributed using the existing DNS infrastructure. SPS must not prohibit other means for distributing keys. Particular implementations may, however, rely on the DNS for key

distribution, though they may not be as robust as implementations that provide several key distribution mechanisms.

2. Overview

The Security Policy System is a distributed system which provides hosts and security gateways with the policy information required to establish a secure communication end-to-end through possibly multiple security gateways. The Security Policy System provides one or more automated mechanism for hosts to discover primary and secondary security gateways relevant in an end-to-end communication. Using the Security Policy System, hosts can validate the identity of security gateways and verify that the gateways in question are authorized to represent the source or destination host which they claim to represent.

SPS is comprised of Policy Servers (PS), Policy Clients (PC), Master Files and SPS Databases. Master Files contain local policies and other particular information about a security domain. Local policy information combined with non-local policies (policies outside the boundaries of the security domain) form the SPS Databases. Policy Servers receive request messages from policy clients and other policy servers, process them, and provide the appropriate policy information to the requestor based on the request and access control rules. The servers also maintain the SPS databases by loading local and non-local policy information received through SPS exchanges. Policy Clients generate requests for policy information and transform the replies into the appropriate format required by the application using SPS. Figure 1.0 depicts the SPS components.

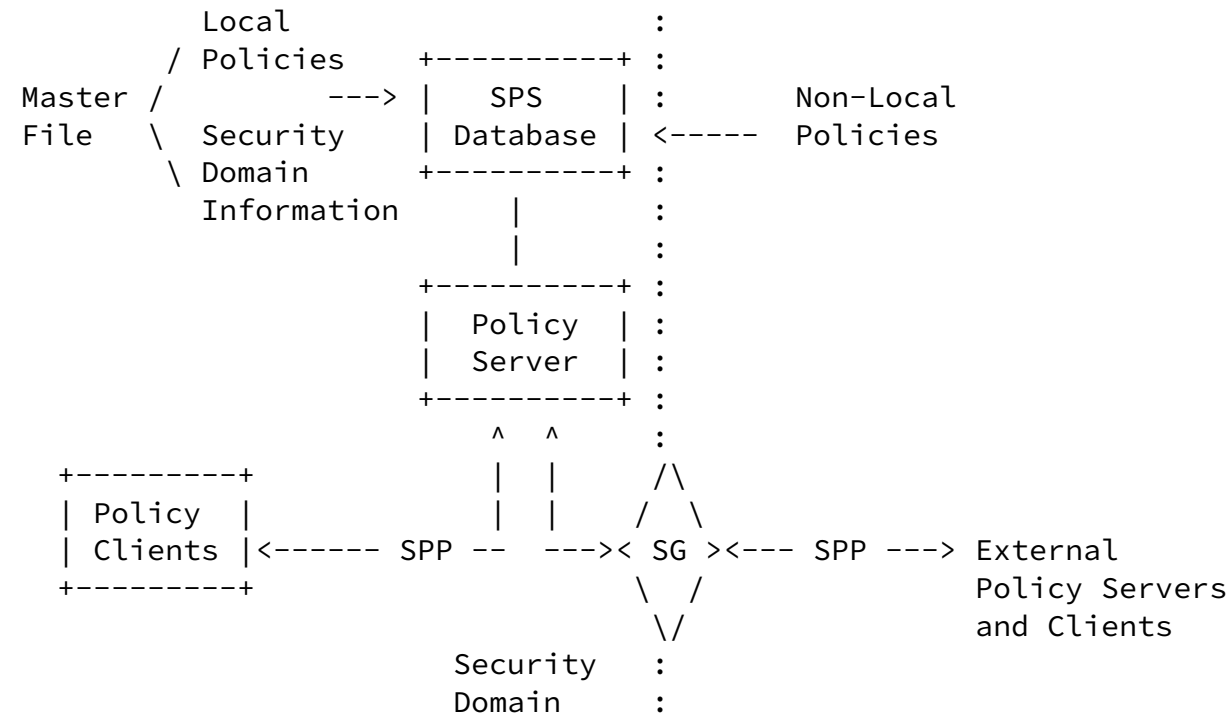


Figure 1 SPS components and interactions

Policy Servers and Clients use the Security Policy Protocol (SPP) to exchange policy information. SPP transports policy information from the SPS Database where this information resides to security gateways and hosts. This protocol provides hosts with the policy information needed to establish security associations with security gateways in the communication path to other hosts, without requiring a-priori knowledge of the identities of the security gateways.

Suppose Policy ClientA would like to establish an IPSec protected communication with Policy ClientB. Policy ClientA is unaware of the existence of Security GatewayB (SGB). Similarly, Policy ClientB is unaware of the existence of Security Gateway A (SGA). Furthermore, assumme that SGB and SGA both have policies stating that any inbound communication must be authenticated using ESP [KA98b]. Now, if Policy ClientA initiates negotiations for a security association with Policy ClientB it would fail since SGB requires any inbound packets to be authenticated using ESP and Policy ClientA doesn't have a security association with SGB.

SPS provides the mechanisms to resolve this problem. Imagine that the network administrators for Security Domain Foo and Foobar prepare a Master File containing the policies of their respective domains. These are the policies enforced by SGA and SGB do not include specific policies of any of the clients members of a security domain. A security domain could be as small as one host acting as its own Policy Client and Server and, enforcing its own policies, or as large as many networks with several Policy Servers and Security

Gateways.

Both Master Files are first parsed and verified to avoid syntax errors. Policies within the Master Files are decorrelated using the decorrelation process specified in section 10. Policy Server A and B load the policies from their respective security domains into their SPS Databases and listen for policy requests.

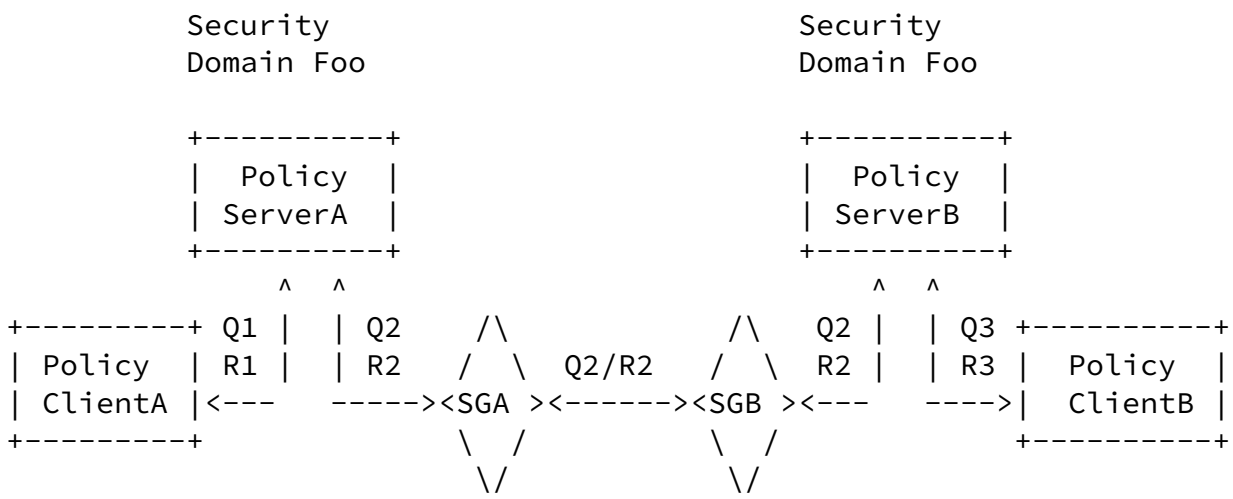


Figure 2 Example of SPS operation

Now suppose that Policy ClientA had a facility enabling it to request policy information to Policy Server A dynamically. The policy request message could be triggered by a message sent from the kernel to a Key Management Protocol. So, Policy ClientA sends a Query (Q1) to Policy ServerA. Policy ServerA looks in its cache for a policy record that matches the query. If it doesn't find one it sends a Query (Q2) containing the same policy request information destined to Policy ClientB. This message includes a signature that validates the authenticity and integrity of the query's content.

(Q2) is intercepted by SGB. SGB forwards the message (Q2) to Policy ServerB. Policy serverB first verifies that it can accept queries from Policy ServerA. It then validates the signature in Q2. It searches its database for the appropriate policy information after verifying that it is authoritative over Policy ClientB.

Policy ServerB first merges its local policy with the policy information in (Q2) and it then replies (R2) to Policy ServerA. The reply includes the original query information and all policy information needed to allow Policy ClientA to establish a secure communication with Policy ClientB. The merging or policy resolution process helps in determining any policy conflicts and ambiguities.

It is possible for Policy ServerB to query Policy ClientB for

its policy with respect to this particular communication. Policy ServerB could generate then a third query (Q3). Policy ClientB responds with its policy in (R3). Policy ServerB merges its policy for this communication and the policy in (R3) before replying to Policy ServerA. Policy ServerB also attached additional information to the reply asserting its authority over Policy ClientB. This chain of trust MUST be validated cryptographically. The merged policy returns in (R2) to Policy ServerA where it merges its local policy with the external policy received in (R2) and caches it for later use.

A protocol such as SPP accommodates topology changes, hence policy changes, rather easily without the scalability constraints imposed by static reconfiguration of each client. The protocol is extensible and flexible It allows the exchange of complex policy objects between clients and servers.

3. Policy Master File

In SPS, policy information of a particular security domain is stored in a Master File. SPS does not impose a particular format for the data store in a master file. SPSL [SPSL] defines a vendor independent language that can be used to define policy information for a security domain. Master Files however, must contain the information specified below. The order in which the policy information appears in the Master File is extremely important since most policy enforcers search for the first policy entry that matches the characteristics of a particular packet. Any other applicable policy found after the first match is ignored.

Master files require the following information:

certificate

This information points to one or more certificates to be referenced by the maintainer information. The private key corresponding to the public key found in this certificate is used to sign the information contained in the master file. The public key is used to verify the information's integrity and authenticity.

maintainer

This information defines the entities authorized to create, delete and modify the policy information in a particular

Master File.

policy-server

This information specifies the identity of the primary and secondary policy servers for a particular security domain.

nodes

This information identifies a set of interfaces that may have policies attached to them. There must be at least one node in a security domain.

gateway

This information identifies a set of interfaces associated with a host that enforces the policies of a particular security domain.

domain

The domain information defines a security domain in terms of the nodes, the security gateways and the policy servers that are part of it.

policy

An ordered set of policies. These policies cover the domain or domains describe by the preceding information.

The Master file contains the list of nodes that are part of the security domain, the list of security gateways protecting the security domain, a list of the policy rules enforced by the security gateways and possibly the policy rules enforced at the nodes.

The Master File MUST contain information indicating who is responsible for the security domain and a pointer to a public key that can be used to verify the integrity and authenticity of the information found in the master file.

4. Policy Database

In SPS, every security domain MUST maintain a database containing the policy information for that domain. Security domains could be as small as one host or as large as several networks. This database, called the SPS Database, is comprised of three logical databases: 1) the Local Policy Database, 2) the Cache Database and, 3) the Security Domain Database.

The Local Policy Database contains all the policies for the

security domain. It is populated with information coming from the Master File of the security domain. The Cache Database contains local and external policies received from other security domains via SPS exchanges. The policies are merged using the policy resolution process specified in section 11. The Security Domain Database contains a list of all hosts, security gateways, and policy servers that are part of the security domain.

Compliant SPS implementations of a policy client and server do not need to implement these three databases separately. However, the information contained in each one of them MUST exist. The Local Database and the Cache Database MUST keep a distinct sets of policies since it is not possible to revert cached policy information into Local Database policy information after the cached items expire.

While it is not necessary to standardize the format of the database used, the SPS database MUST contain a minimum set of information. The next three subsections describe each database in terms of its functionality and requirements.

4.1 Local Policy Database

Every policy client and every policy server MUST keep a database containing its local policy. In the client case, the policy is kept in some pre-configured form and loaded into the database. The database could be the same as the client's Security Policy Database (SPD) as defined in [kent98]. In the server's case the information found in the Local Policy Database applies to all the members of the security domain and therefore it MUST be kept separate from the server's own policy information found in its SPD.

The Local Policy Database MUST contain the policies expressed in the Master File for the security domain. Policies are divided into a clause section and an action section. The clause section describes a particular communication while the action clause indicates what should be done with the communication.

Each policy entry MUST contained a unique identifier, an expiration time, and a single policy clause followed by each action clause that applies to that policy clause. In terms of the Security

Policy Protocol, the clause portion of the policy is encoded in communication security records and the action portion (if IPSec related) is encoded in security association records.

Policies in this database MUST be decorrelated as defined in section 10. An algorithm for policy decorrelation is presented in section 10.1. Decorrelated policies allow for efficient reordering and organization of the policies without affecting the enforcement process. Policy decorrelation also facilitates the caching of external

policies.

4.2 Cache Database

In addition to the information stored in the Local Policy Database, every policy client and server MUST keep a cache of merged local and non-local policy data. Non-local policies are policies which do not exist in some pre-configured form in a policy client or server. These policies are learned through SPS exchanges.

Non-local policies are merged with Local Database policies using the policy resolution process discussed in section 11. The resultant merged policies MUST be kept logically separate from the local policies. The cached data then can be used to answer subsequent queries for the same policy information.

Each policy entry MUST contain a unique identifier, an expiration time, and typically a single policy clause followed by each action clause that applies to that policy clause. A policy entry MAY contain multiple policy clauses each followed by action clauses, if the policy must be enforced at multiple endpoints. Each policy entry MUST also contain any assertion information that could indicate the relationship between the policy server that provide the SPS message and the information in the policy exchange. It SHOULD also include any public keying material (e.g. certificates, etc.) that might be needed to validate the assertions made by policy servers. In terms of the Security Policy Protocol, policy server assertions, and certificates are encoded in policy server and certificates records, respectively.

Policy Clients and Servers MUST NOT cache policies indefinitely, since cached policies have a non-local component that may change without warning. Each policy entry MUST contain an expiration time that MUST be considered as the maximum amount of time that this policy MAY be cached. Longer cache expiry times should be associated with policies that are less likely to change, while shorter cache expiry times should be associated with policies that are likely to change. As in the Local Policy Database, policies in the Cache Database MUST be decorrelated as defined in section 10.

4.3 Security Domain Database

A Policy Server MUST also maintain information that describes the security domain for which it is authoritative. This information includes a list with the identities of each security gateway enforcing policies at the perimeter of the security domain and an entry indicating the identities of the nodes that are members of the security domain.

Policy servers use this information to determine that they are authoritative over the host for which they are providing policy information. It also allows policy servers to determine if they may participate in an SPP exchange without violating the chain of trust that the recipient of the information will require to verify the authenticity of the policy.

5. Security Policy Protocol (SPP)

Policy clients and servers exchange information using the Security Policy Protocol. The protocol defines how the policy information is exchanged, processed, and protected by clients and servers. The protocol also defines what policy information is exchanged and the format used to encode the information. The protocol specifies six different message types used to exchange policy information. An SPP message contains a message header section followed by zero or more SPP payloads, depending on the message type.

Figure 3 depicts the format of an SPP message. The header section is present in every message. It contains fields identifying the message, the type of message, the status of the message, the number of queries and/or record payloads, and the host requesting policy information. The header also includes a timestamp field that provides anti-replay protection. Following the header there might be zero or more SPP payloads. Currently, there are three payload types defined in SPP: Query, Record, and Signature payloads. See section 5.2 for encoding details.

SPP has six distinct message types. Query messages contain a specific request for policy information. Reply messages include policy records that answer specific policy queries. Policy messages include policy information and are utilized for up/downloading security policies to and from a policy server. Policy Acknowledgment messages are utilized to acknowledge corresponding Policy messages but do not themselves contain policy information. Transfer messages, which include policy information, are utilized by policy servers to exchange bulk policy information between servers. Finally, policy servers use keep alive messages to inform security gateways and/or other monitoring devices of the status of the server.

SPP messages MUST be authenticated either using IPSec [Kent98] or another security mechanism. SPP provides a basic security mechanism that can be used to provide authentication and integrity to its messages, especially when traversing heterogenous domains and the identity of the policy server authoritative for the destination is unknown. These services are provided using digital signatures.

SPP caries signatures in the signature payload. The signature is calculated over the entire SPP message. When this service is used, the entity (host, policy server or security gateway) verifying the signature must have access to the public key that corresponds to the private key used to sign the SPP message.

Certificate fetching is out of the scope of SPP. However, SPP provides a simple certificate fetching mechanism for entities that elect to use it as an alternative to other mechanisms. SPP suports several Public Key certificates formats.

SPP is modular and extensible. New policy queries and records can be defined and incorporated easily. This document defines a minimum set of queries and policy records required in a policy-based security management system.

5.1 SPP Message Format

An SPP message follows the format depicted in figure 3. It is comprised of a header and zero or more SPP payloads. This section defines the encoding for the SPP header. Sections 5.2 and 5.3 cover the encoding for the SPP payload and message types, respectively.

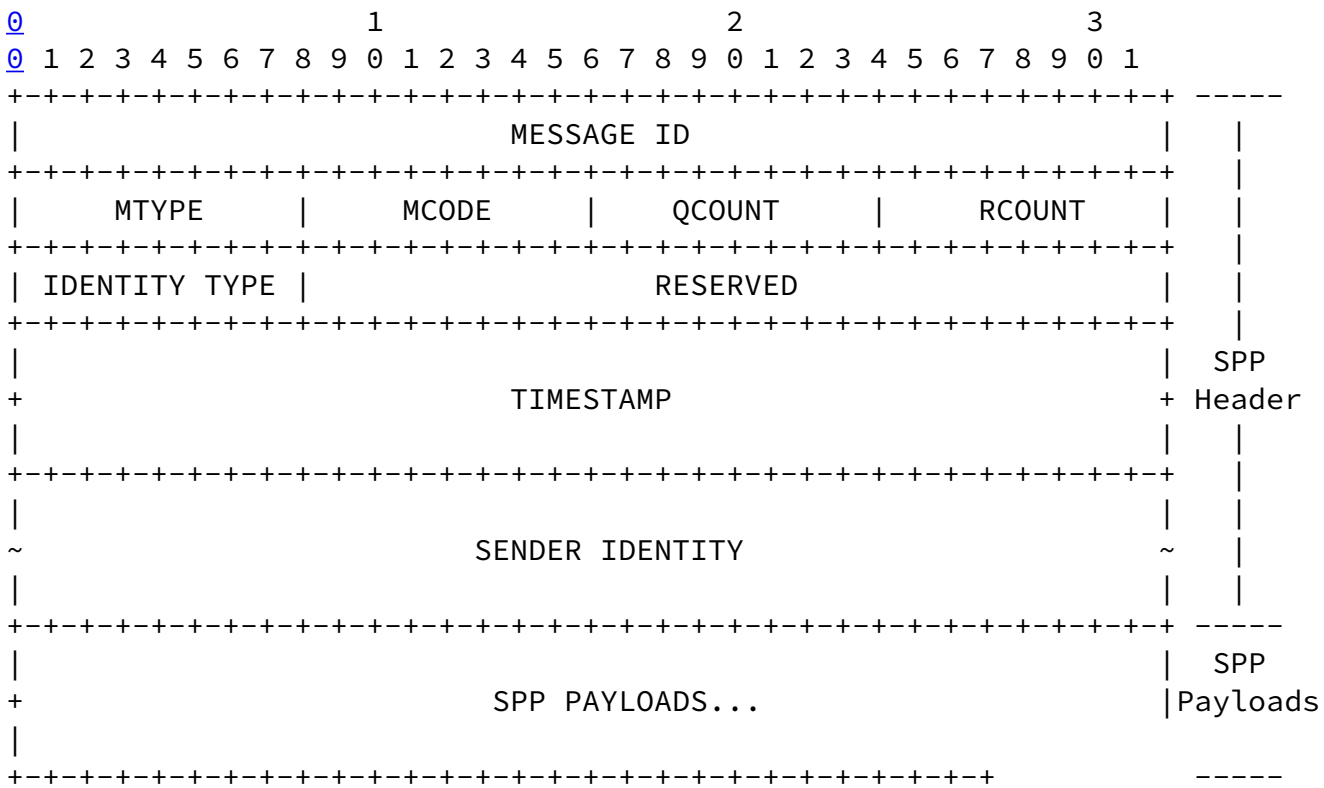


Figure 3: Format of an SPP Message

The SPP header includes the following fields:

MESSAGE ID

A 4 octet field used to match messages and their responses (e.g. queries to replies and policy to policy acknoledgement messages). This value starts at "zero" and MUST be incremented by (01)hex with every new message.

MTYPE

A 1-octet field indicating the SPP message type.
The currently defined values are:

Value	Message Type
00	Value Not Assigned
01	SPP-QUERY
02	SPP-REPLY
03	SPP-POL
04	SPP-POL_ACK
05	SPP-XFR
06	SPP-KEEP_ALIVE
07-255	Reserved

MCODE

A 1-octet field providing information about this message.
See section 5.3 for the codes applicable to each message type.

Code Field	Action Type
00	Value Not Assigned
01	message accepted
02	denied, administratively prohibited
03	denied, timestamp failed
04	denied, failed signature
05	denied, insufficient resources
06	denied, malformed message
07	denied, unspecified
08	partially available
09	unavailable

10	communication prohibited
11-255	reserved

QCOUNT

A 1 octet field indicating the number of Query payloads included in the message.

RCOUNT

A 1 octet field indicating the number of Record payloads included in the message.

IDENTITY TYPE

This 1 octet field indicates the type of indentity found in the Sender Identity field. Valid values are:

Value	Identity Type
00	Value Not Assigned
01	IPV4_ADDR
02	IPV6_ADDR
03	Host DNS Name
04-255	Reserved

RESERVED

A 3 octet field used for field 32-bit boundary alignment and reserved for future use. Set value to all zeros (00)hex.

TIMESTAMP

This 8-octet field contains a timestamp used to provide limited protection against replay attacks. The timestamp is formatted as specified by the Network Time Protocol [RFC1305].

SENDER IDENTITY

A variable length field containing the identity of the sender (host, security gateway or policy server) of the SPP message. The IDENTITY_TYPE field indicates the format of the content in this field. This field does not allow IP address ranges or wildcards.

5.2 SPP Payloads

5.2.1 Query Payload

The Query payload contains fields to express a particular request for policy information. Hosts, security gateways, or policy servers can generate and transmit Query payloads in SPP messages to policy servers. Figure 4 shows the format of the Query payload.

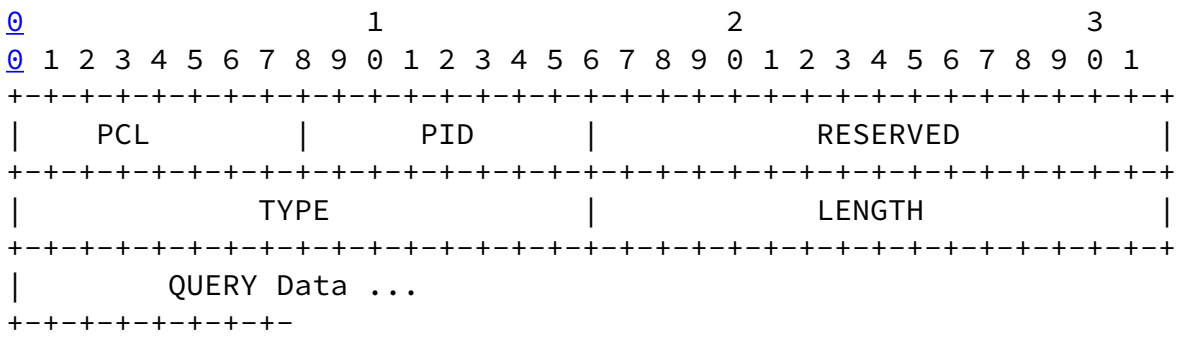


Figure 4: Format of Query Payload

The Query Payload fields are defined as follows:

PCL

A 1 octet field indicating the payload class. Query payloads MUST contain (01)hex in the PCL field.

PID

A 1 octet field containing the ID number that identifies a particular Query payload within an SPP message. Since one SPP message can contain multiple Query payloads, each one MUST be uniquely identified. This number MUST be unique among the Query payloads within an SPP message.

RESERVED

A 2 octet field reserved for future use. Set value to all zeros (00)hex.

TYPE

A 2 octet field that specifies the type of query contained in the QUERY Data fields. The currently defined queries are:

Value	Query Payload Type
00	Value Not Assigned
01	Security Gateway Query
02	Communication Security Query
03	Certificate Query
04-65536	Reserved

LENGTH

A 2 octet field indicating the length in octets of the query data field.

QUERY Data

A variable length field containing a single policy query. See section 7 for encoding format.

5.2.2 Record Payload

The Record payload contains fields that assert policy information. Hosts, security gateways, or policy servers can generate and transmit Record payloads in SPP messages. Figure 5 shows the format of the Record payload.

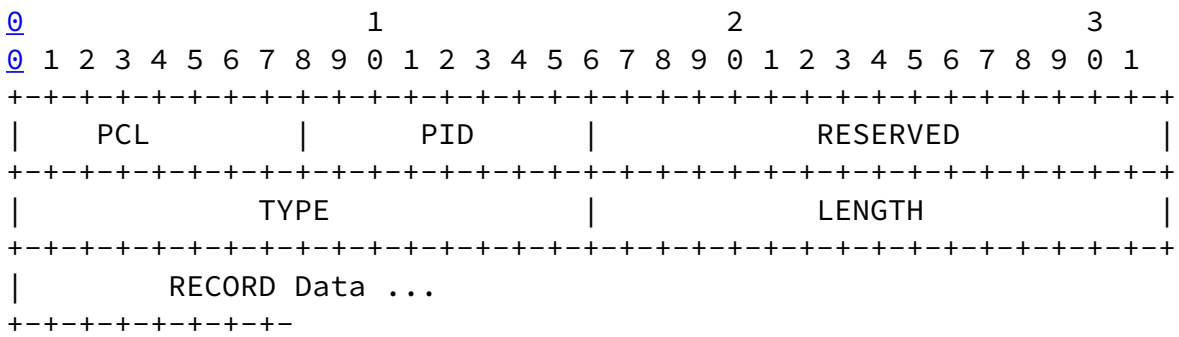


Figure 5: Format of Record Payload

The Record Payload fields are defined as follows:

PCL

A 1 octet field indicating the payload class. Record payloads MUST contain (02)hex in the PCL field.

PID

This field is used to match queries to Record payloads. If the record is a reply to a query, then the value for this field MUST match the correspondent Query payload PID. If it is not a reply to a query, the value SHOULD be set to zero.

RESERVED

A 2 octet field reserved for future use. Set value to all zeros (00)hex.

TYPE

A 2 octet field that specifies the type of Record. The currently defined records are:

Value	Record Type
00	Value Not Assigned
01	Security Gateway Record
02	Communication Security Record
03	Security Association Record
04	Certificate Record
05	Policy Server Record
06-65536	Reserved

LENGTH

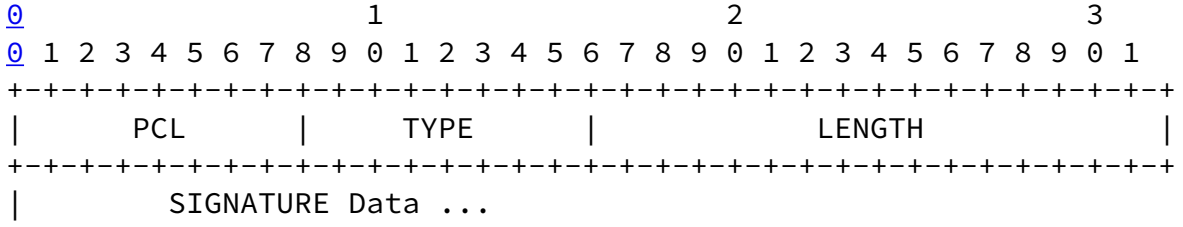
A 2 octet field indicating the length in octets of the RECORD data field.

RECORD Data

A variable length field containing a single policy record. See section 8 for encoding format.

5.2.3 Signature Payload

The Signature Payload contains data generated by the digital signature function (selected by the originator), over the entire SPP message, except for part of the Signature payload. This payload is used to verify the integrity of the data in the SPP message. Figure 6 shows the format of the Signature payload.



```
+-+-+-+-+-+-+-+-
```

Figure 6: Signature Payload Format

The Signature payload fields are defined as follows:

PCL

A 1 octet field indicating the payload class. Signature payloads MUST contain (03)hex in the PCL field.

TYPE

A 1 octet field that specifies the signature algorithm employed. The currently defined signature types are:

Value	Algorithm Type
00	Value Not Assigned
01	RSA
02	DSA
03-255	Reserved

LENGTH

A 2 octet field indicating the length in octets of the SIGNATURE Data field.

SIGNATURE Data

A variable length field that contains the results from applying the digital signature function to the entire SPP message (including the PCL, TYPE, and LENGTH fields of the Signature payload), except for the Signature Data field of the Signature payload.

5.3 SPP Messages

5.3.1 Query Message

An SPP-QUERY message is comprised of an SPP header, one or more Query payloads, zero or more Record payloads, and a Signature payload, if one is required. Query messages MUST always contain a Query payload. Record payloads may optionally be included to pass policy information along with the query. If the Signature payload is employed it MUST be the last payload in the message. The Query message MTYPE value is (01)hex. The MCODE field must be set to zero (00)hex.

5.3.2 Reply Message

An SPP-REPLY message is comprised of an SPP header, one or more Query payloads, zero or more Record payloads which answer the corresponding Query payload, and a Signature payload, if one is required. Reply messages MUST contain a Query payload. Reply messages MUST include a Record payload unless the reply contains an error code of values (02-08). If the Signature payload is employed it MUST be the last payload in the message. The MTYPE value for a Reply message is (02)hex. The following MCODE values may be used for Reply messages:

Code Field	Action Type
01	message accepted
02	denied, administratively prohibited
03	denied, timestamp failed
04	denied, failed signature
05	denied, insufficient resources
06	denied, malformed message
07	denied, unspecified
08	partially available
09	unavailable
10	communication prohibited

5.3.3 Policy Message

An SPP-POL message is comprised of an SPP header, one or more Record payloads, and a Signature payload, if one is required. Policy messages MUST NOT include Query payloads. If the Signature payload is employed it MUST be the last payload in the message. The MTYPE value for a Policy message is (03)hex. The MCODE field must be set to zero (00)hex.

5.3.4 Policy Acknowledgement Message

An SPP-POL_ACK message is comprised of an SPP header and a Signature payload, if one is required. These messages MUST NOT contain Query or Record payloads. The status of the associated Policy message

is expressed within the MCODE field. If the Signature payload is employed it MUST be the only payload in the message. The MTYPE value for a Policy Acknowledgement message is (04)hex. The following MCODE values may be used for Policy Acknowledgement messages:

Code Field	Action Type
01	message accepted
02	denied, administratively prohibited
03	denied, timestamp failed
04	denied, failed signature
05	denied, insufficient resources
06	denied, malformed message
07	denied, unspecified

5.3.5 Transfer Message

An SPP-XFR message is comprised of an SPP header, one or more Record payloads, and a Signature payload, if one is required. Transfer messages MUST NOT include Query payloads. If the Signature payload is employed it MUST be the last payload in the message. The MTYPE value for a Transfer message is (05)hex. The MCODE field must be set to zero (00)hex.

5.3.6 KeepAlive Message

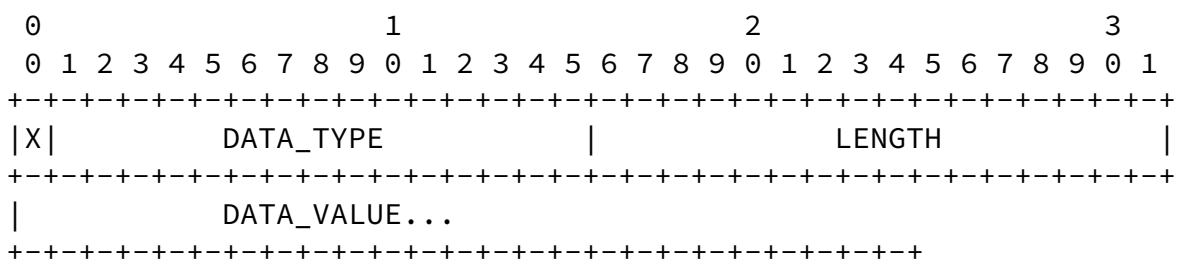
An SPP-KEEP_ALIVE message is comprised of an SPP header and a Signature payload, if one is required. These messages MUST NOT contain Query or Record payloads. If the Signature payload is employed it MUST be the only payload in the message. The MTYPE value for a KeepAlive message is (06)hex. The MCODE field must be set to zero (00)hex.

6.0 Policy Attribute Encoding

Query and Record payloads include several different selector types and SA attributes with their associated values. These data are encoded following a Type/Length/Value (TLV) format to provide flexibility for representing different kinds of data within a payload. Certain Data_Types with values of length equal to 2 octets follow the Type/Value (T/V) format. The first bit of the DATA_TYPE field is used to distinguished between the two formats. A value of (0) indicates a TLV format while a value of (1) indicates TV format. This

generic encoding format is depicted in figure 7.

X = 0:



X = 1:

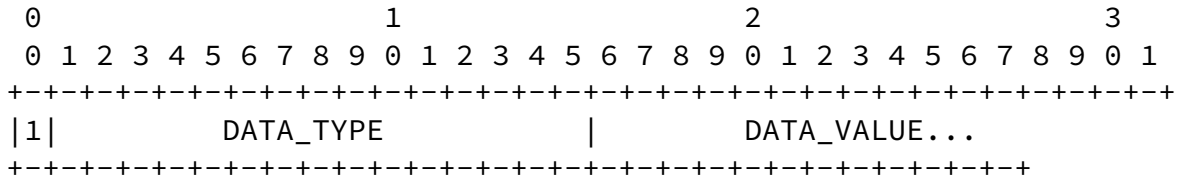


Figure 7: Generic Data Attribute Formats

The generic data attribute fields are defined as follows:

X

This bit indicates if the DATA_TYPE follows the TLV(0) or the TV(1) format.

DATA_TYPE

A 2 octet field indicating the selector type. The currently defined values are:

DATA	DATA_TYPE	X
IPV4_ADDR	1	0
IPV6_ADDR	2	0
SRC_IPV4_ADDR	3	0
SRC_IPV4_ADDR_SUBNET	4	0
SRC_IPV4_ADDR_RANGE	5	0
DST_IPV4_ADDR	6	0

DST_IPV4_ADDR_SUBNET	7	0
DST_IPV4_ADDR_RANGE	8	0
SRC_IPV6_ADDR	9	0
SRC_IPV6_ADDR_SUBNET	10	0
SRC_IPV6_ADDR_RANGE	11	0
DST_IPV6_ADDR	12	0
DST_IPV6_ADDR_SUBNET	13	0
DST_IPV6_ADDR_RANGE	14	0
DIRECTION	15	1
USER_NAME	16	0
SYSTEM_NAME	17	0
XPORT_PROTOCOL	18	0
SRC_PORT	19	0
SRC_PORT_DYNAMIC	20	0
DST_PORT	21	0
DST_PORT_DYNAMIC	22	0
SEC_LABELS	23	0
V6CLASS	24	1
V6FLOW	25	0
V4TOS	26	1
ACTION	27	1
SRC_PORT_RANGE	28	0
DST_PORT_RANGE	29	0
IPSEC_ACTION	50	0
ISAKMP_ACTION	51	0
RESERVED	30-49, 52-32767	

LENGTH

A 2 octet field indicating the length of the selector value in octets, not including any trailing padding added to the DATA_VALUE field. The padding length is implicit.

DATA_VALUE

A variable length field containing the value of the selector specified by DATA_TYPE. If the Selector value is not aligned at the 4 octet boundary the field MUST be padded on the right with (00)hex to align on the next 32-bit boundary.

7. Policy Queries

7.1 Security Gateway Query

This basic query provides a dynamic mechanism to determine which relevant security gateways, both primary and backup, are in the path to a particular destination address. Since the answer to a request for information could depend on the identity of the requestor, the host address of the source of the intended communicaton is included in the query. Figure 8 shows the format of the Security Gateway Query.

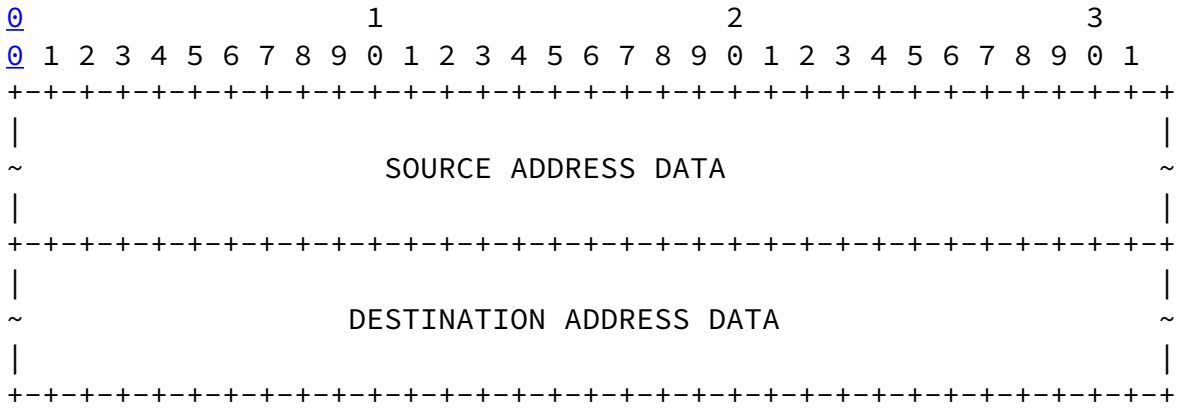


Figure 8: Security Gateway Query Format

The Security Gateway Query fields are defined as follows:

SOURCE ADDRESS DATA

This variable length field contains a single IP address (unicast) either in IPv4 or IPv6 format. The encoding format is specified in section 6. The acceptable DATA_TYPE values are 3 and 9.

DESTINATION ADDRESS DATA

This variable length field contains a single IP address (unicast) either in IPv4 or IPv6 format. The encoding format is specified in section 6. The acceptable DATA_TYPE values are 6 and 12.

7.2 COMSEC Query

The Communication Security Query (or COMSEC query) provides a dynamic mechanism for a host or security gateway to inquire if a communication having a particular set of characteristics is allowed. The communication is described in terms of source and

destination addresses, protocols, source port, destination port, and other parameters as defined in section 6. These parameters are known as selectors in the IPSec context and are primarily the contents of the IP and TCP headers. Figure 9 shows the format of the COMSEC Query.

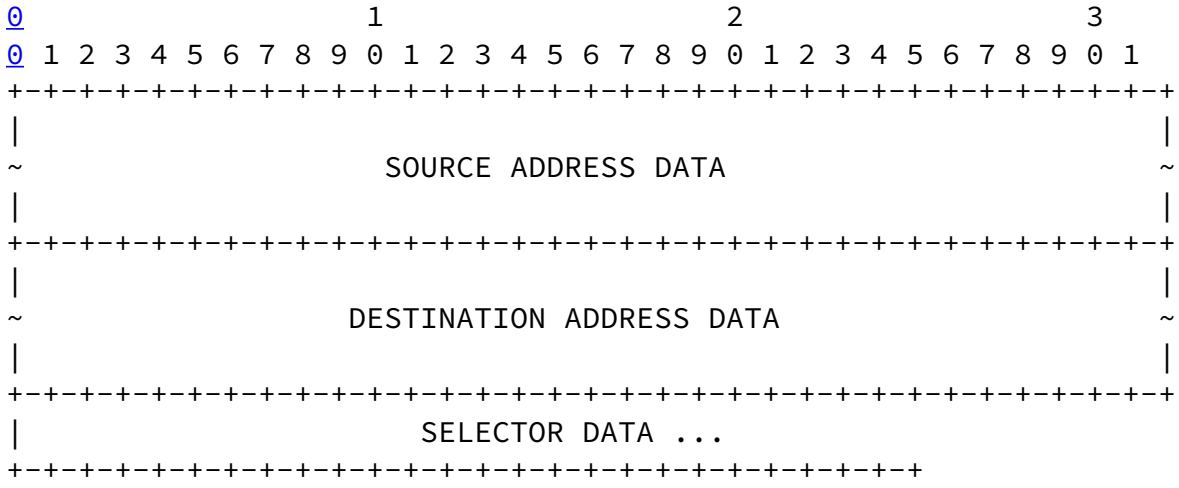


Figure 9: COMSEC Query Format

The COMSEC Query fields are defined as follows:

SOURCE ADDRESS DATA

This variable length field contains a single IP address (unicast) either in IPv4 or IPv6 format. The encoding format is specified in section 6. The acceptable DATA_TYPE values are 3 and 9.

DESTINATION ADDRESS DATA

This variable length field contains a single IP address (unicast) either in IPv4 or IPv6 format. The encoding format is specified in section 6. The acceptable DATA_TYPE values are 6 and 12.

SELECTOR DATA

This includes one or more fields following the encoding format specified in section 6. The acceptable DATA_TYPE values are 15-29, inclusive.

7.3 CERT Query

Mechanisms to dispatch and fetch public-key certificates are not part of SPP. However, in the absence of external request/dispatch mechanisms, SPP provides for a certificate request query that allows a host, security gateway, or server to solicit a certificate. Figure 9 shows the format of the CERT Query.

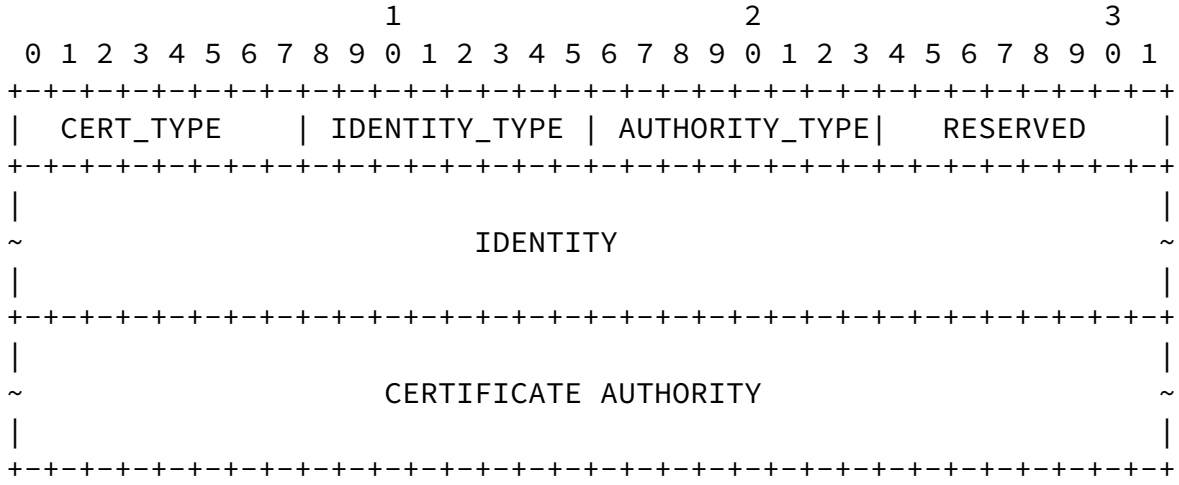


Figure 9: Certificate Query Format

The Certificate query fields are defined as follows:

CERT_TYPE

A 1 octet field that contains an encoding of the type of certificate requested. Acceptable values are listed below:

Certificate Type	Value
Value Not Assigned	0
PKCS #7 wrapped X.509 certificate	1
PGP Certificate	2

DNS Signed Key	3
X.509 Certificate - Signature	4
X.509 Certificate - Key Exchange	5
Kerberos Tokens	6
SPKI Certificate	7
RESERVED	8 - 255

IDENTITY_TYPE

This 1 octet field indicates the type of indentity found in the Identity field. Valid values are listed below:

Value	Identity Type
0	Value Not Assigned
1	IPV4_ADDR
2	IPV6_ADDR
3	DNS Name
4	X.500 Distinguished Name
5-255	Reserved

AUTHORITY_TYPE

This 1 octet field indicates the type of authority found in the Certificate Authority field. Valid values are the same as IDENTITY_TYPE.

IDENTITY

This variable length field contains the identity of the principal by which the certificate should be located. The value MUST be of the type stated in IDENTITY_TYPE.

CERTIFICATE AUTHORITY

A variable length field containing an encoding of an acceptable certificate authority for the type of certificate requested. The value MUST be of the type stated in AUTHORITY_TYPE.

8. Policy Records

8.1 Security Gateway Record

This record contains information that indicates the IP

addresses of the interfaces for the the primary and secondary security gateways protecting a host or group of hosts. The record contains the primary and secondary gateways at one point in the path that an IP datagram originated from the source address listed in the Security Gateway query will have to traverse to reach the destination address listed in that query. If the IP datagram must traverse multiple gateways, a Security Gateway Record must be included for each gateway. The list of secondary security gateways is optional. Figure 10 shows the format of the Security Gateway Record.

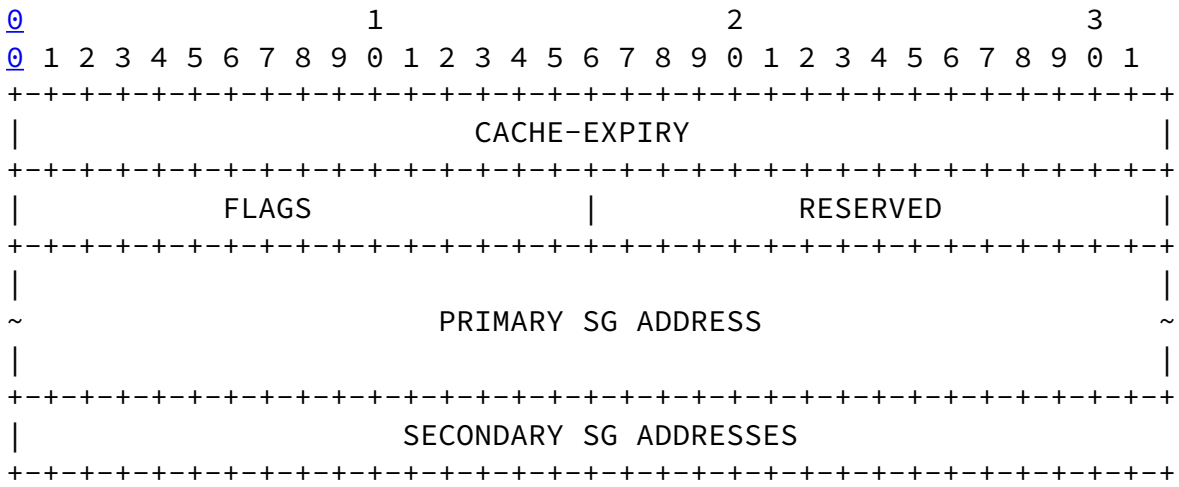


Figure 10: Security Gateway Record Format

The Security Gateway Record fields are defined as follows:

CACHE-EXPIRY

A 4 octet field indicating the maximum amount of time, in seconds, this policy record MAY be cached.

FLAGS

A 2 octet field indicating different options to aid in interpreting the security gateway data. If not in use, set value to all zeros (00)hex. Currently, no flag values are defined so this field MUST be set to (00)hex.

RESERVED

A 2 octet field reserved for future use. If not in use, set value to all "zeros".

PRIMARY SG ADDRESS

A variable length field containing the IP address of the primary security gateway for protecting a particular host. This variable length field contains a single unicast IP address. The encoding format is specified in section 6. The acceptable DATA_TYPE values are 1 and 2.

SECONDARY SG ADDRESSES

This variable length field contains the IP addresses of secondary security gateways protecting a particular host. This field may contain a list of single unicast IP addresses. The encoding format is specified in section 6. The acceptable DATA_TYPE values are 1 and 2.

8.2 COMSEC Record

The COMSEC record indicates if a communication having a particular set of characteristics is allowed or not. The communication is described in terms of source and destination addresses, protocols, source ports, destination ports, and other attributes defined in section 6. Figure 11 shows the format of the COMSEC Record.

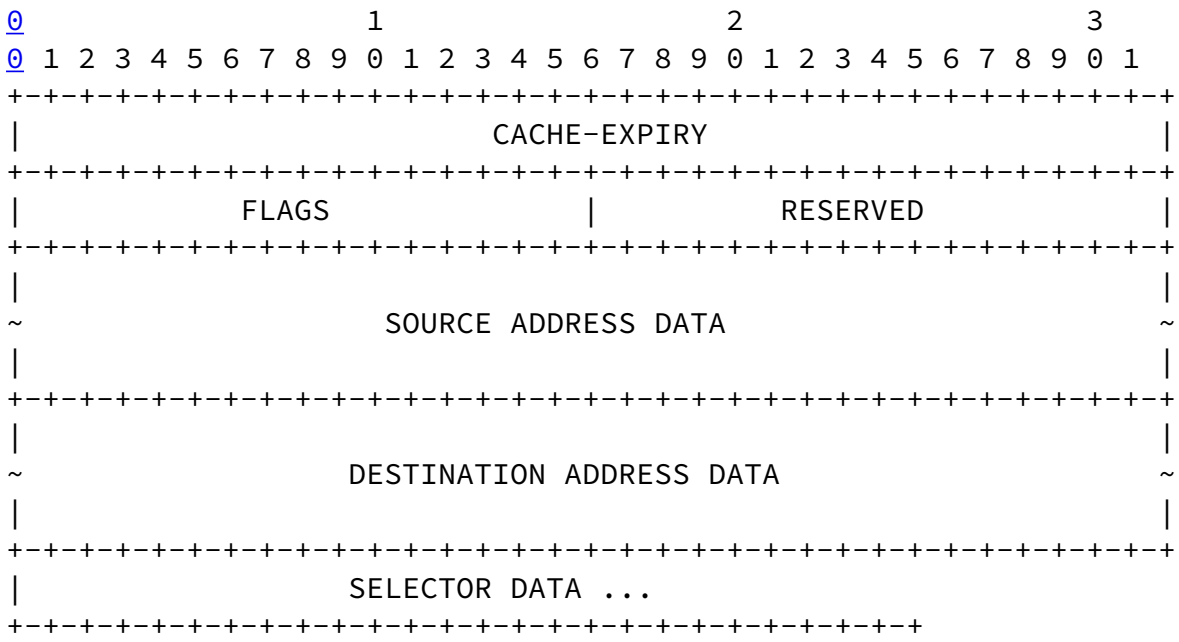


Figure 11: COMSEC Record Format

The COMSEC Record fields are defined as follows:

CACHE-EXPIRY

A 4 octet field indicating the maximum amount of time, in seconds, this policy record MAY be cached.

FLAGS

A 2 octet field indicating different options to aid in interpreting the selector data. If not in use, set value to all zeros (00)hex. Currently, no flag values are defined so this field MUST be set to (00)hex.

RESERVED

A 2 octet field reserved for future use. If not in use, set value to all zeros (00)hex.

SOURCE ADDRESS DATA

This variable length field contains a single IP address (unicast, anycast, broadcast (IPv4 only), or multicast group), range of addresses (low and high values, inclusive), address + mask, or a wildcard address. The encoding format is specified in section 6. The acceptable DATA_TYPE values are 3-5 and 9-11, inclusive.

DESTINATION ADDRESS DATA

This variable length field contains a single IP address (unicast, anycast, broadcast (IPv4 only), or multicast group), range of addresses (low and high values, inclusive), address + mask, or a wildcard address. The encoding format is specified in section 6. The acceptable DATA_TYPE values are 6-8 and 12-14, inclusive.

SELECTOR DATA

This includes one or more fields following the encoding format specified in section 6. The acceptable DATA_TYPE values are 15-29, inclusive.

8.3 Security Association Record

Security Association Records contain selectors and security association attributes (APPLIERS) that characterize a particular Security Association between the source and destination addresses

listed in the record. This record contains data types as defined in the section 6. Figure 12 shows the format of the SA Record.

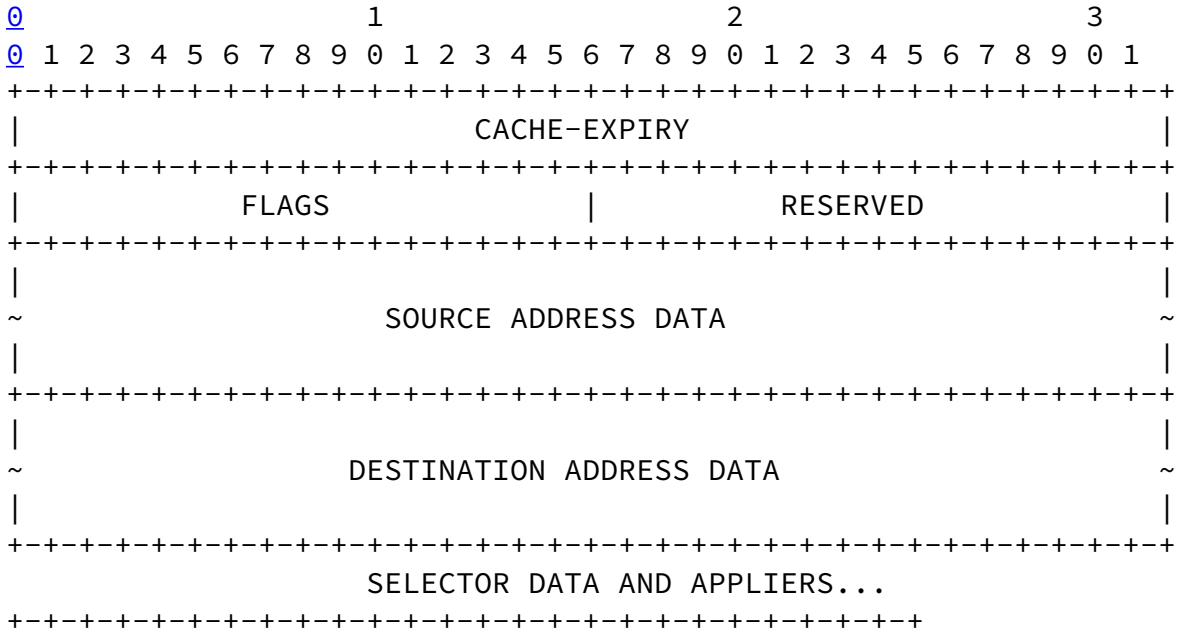


Figure 12: SA Record Format

The SA record fields are defined as follows:

CACHE-EXPIRY

A 4 octet field indicating the maximum amount of time, in seconds, this policy record MAY be cached.

FLAGS

A 2 octet field indicating different options to aid in interpreting the selector data. If not in use, set value to all zeros (00)hex. Currently, no flag values are defined so this field MUST be set to (00)hex.

RESERVED

A 2 octet field reserved for future use. If not in use, set value to all zeros (00)hex.

SOURCE ADDRESS DATA

This variable length field contains a single IP address (unicast, anycast, broadcast (IPv4 only), or multicast group), range of addresses (low and high values, inclusive), address + mask, or a wildcard address. The encoding format is specified in section 6. The acceptable DATA_TYPE values are 3-5 and 9-11, inclusive.

DESTINATION ADDRESS DATA

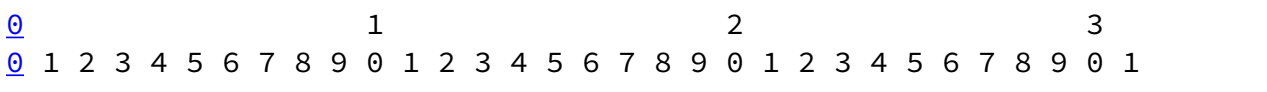
This variable length field contains a single IP address (unicast, anycast, broadcast (IPv4 only), or multicast group), range of addresses (low and high values, inclusive), address + mask, or a wildcard address. The encoding format is specified in section 6. The acceptable DATA_TYPE values are 6-8 and 12-14, inclusive.

SELECTOR DATA AND APPLIERS

This includes one or more fields following the encoding format specified in section 6. The acceptable DATA_TYPE values are 15-29 and 50-51, inclusive.

8.4 Policy Server Record

The Policy Server record indicates the host, security gateway, or policy server for which a particular policy server is authoritative. It represents an assertion, typically made by a policy server, with repect to a member of a security domain that the server represents. The record includes the Identity of the policy server and the identity of a node (host, security gateway, another server, etc.). Figure 13 shows the format of the Policy Server Record.



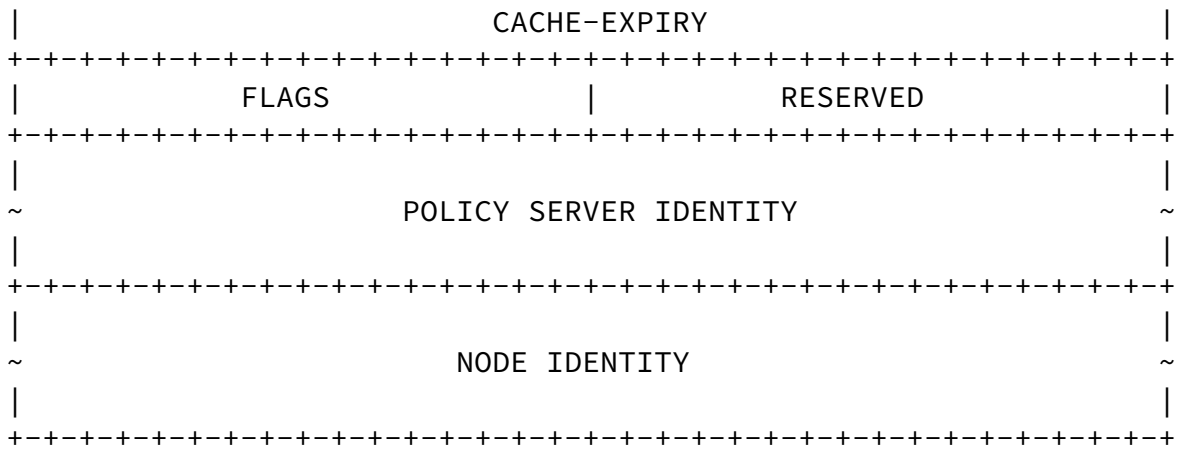


Figure 13: Policy Server record format

The Policy Server Record fields are defined as follows:

CACHE-EXPIRY

A 4 octet field indicating the maximum amount of time, in seconds, this policy record MAY be cached.

FLAGS

A 2 octet field indicating different options to aid in interpreting the server and node data. If not in use, set value to all zeros (00)hex. Currently, no flag values are defined so this field MUST be set to (00)hex.

RESERVED

A 2 octet field reserved for future use. If not in use, set value to all zeros (00)hex.

POLICY SERVER IDENTITY

This variable length field contains the identity of the policy server. It may contain an IP address (unicast) either in IPv4 or IPv6 format. The encoding format is specified in section 6 The acceptable DATA_TYPE values

are 1 and 2.

NODE IDENTITY

This variable length field contains the identity of a node for which the policy server is authoritative. It may contain an IP address (unicast) either in IPv4 or IPv6 format. The encoding format is specified in section 6 The acceptable DATA_TYPE values are 1 and 2.

8.5 CERT Record

The CERT record contains one public key certificate. This record is provided in SPP as an alternate mechanism for certificate dispatching. Figure 14 shows the format of the CERT Record.

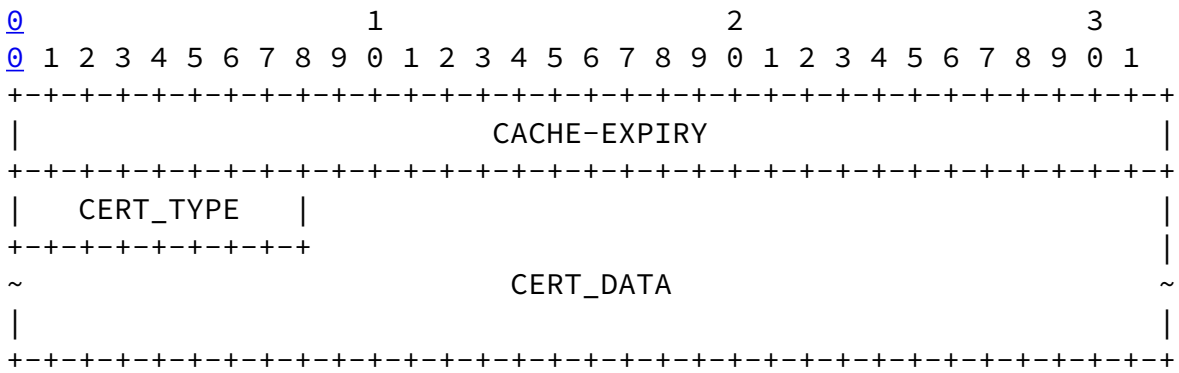


Figure 14: Certificate Record Format

CACHE-EXPIRY

A 4 octet field indicating the maximum amount of time, in seconds, this policy record MAY be cached.

CERT_TYPE

This 1 octet field indicates the type of certificate or certificate-related information contained in the Certificate Data field. The values for this field are described in Section 7.3.

CERT_DATA

This variable length field contains the actual encoding of certificate data. The type of certificate is indicated by the Certificate Type field.

9. SPP Message Processing

SPP messages use UDP or TCP as their transport protocol. Messages carried by UDP are restricted to 512 bytes (not counting the IP or UDP headers). Fragmentation is allowed for messages containing more than 512 bytes. The SPP-XFR message SHOULD use TCP to transfer the contents of the SPS Database from a primary to secondary policy server. SPP uses UDP and TCP ports 501.

9.1 General Message processing

For an SPP-Query or SPP-Pol message, the transmitting entity MUST do the following:

1. Set a timer and initialize a retry counter.
2. If an SPP-Reply or SPP-Pol_Ack message corresponding to the appropriate SPP-Query or SPP-Pol message is received within the time interval, or before the retry counter reaches 0, the transmitting entity continues normal operation.
3. If an SPP-Reply or SPP-Pol_Ack message corresponding to the appropriate SPP-Query or SPP-Pol message is not received within the time interval, and a secondary policy server is available, the message is sent to the secondary server. If a secondary server does not exist the message is resent to the primary and the retry counter is decremented.
4. If the retry counter reaches zero (0) the event should be logged in the appropriate system audit file.
5. At this point, the transmitting entity will clear state for this peer and revert to using conventional security mechanisms.

9.2 Query Message (SPP-Query) Processing

When creating a SPP-Query message, the entity (host, security gateway, or policy server) MUST do the following:

1. Generate the Message ID value. This value starts at "zero" and MUST be incremented by (01)hex with every new message.
2. Set the value of the MTYPE field to 01
3. Set the MCODE value to (00)hex.
4. Set the QCOUNT and RCOUNT fields. All fields MUST be set to (00)hex when their corresponding payloads do no exist.

5. Set the flag bits accordingly and set the RESERVED field to (00)hex.
6. Calculate and place the timestamp value used for anti-replay attack protection.
7. Set the IDENTITY_TYPE and IDENTITY of the Sender of the SPP message.
8. Construct the SPP data payloads. A Query payload MUST be present in this message.
9. Local policy information related to the query MAY be included as Record payloads following the Query payloads.
10. If the Signature payload is required for message integrity and authentication, calculate a signature over the SPP header, SPP payloads, the PCL, TYPE, and LENGTH fields of the Signature payload. If required, the Signature payload MUST be the last payload in the message.

When a policy server receives an SPP-Query message it MUST do the following:

1. Check for SPP access control. If enabled, read the IP address in the Sender's field of the SPP header.

   - Verify whether or not the message is allowed. If the message is not allowed then:

     o send an SPP_Reply message with the error code 02
     o discard message and log EVENT if configured to do so

   - If the message is allowed, continue.

2. Check timestamp field for anti-replay protection. If a replayed message is detected:

   - send an SPP_Reply message with the error code 03
   - discard message and log EVENT if configured to do so

3. If the message requires signature validation.

   - Fetch certificate or key corresponding to the IP address found in the sender field of the SPP header.

   - If a certificate or key is not available the entity MAY, depending on configuration:

     o choose to abort validation process, send SPP-Reply message with error code 05, discard the message, and MAY log the EVENT.

     o send an SPP-Query message to the source of the IP address

found in the sender field of the SPP header with a CERT Query payload. Keep the SPP-Query message until the SPP_Reply returns. Extract certificate or key, validate it and proceed.

- Once a validated certificate or key is available then validate signature.

  o If validation fails, send SPP-Reply message with error code 05, discard the message, and the EVENT MAY be logged.

4. Parse the Query record. Check the Destination Address Data field to determine if the message received was intended for a node that is a member of the server's security domain.

5. If the message is for a node that is a member of the server's security domain then:

- Using src, dst, and any other applicable fields found in the Query Payload, search the SPS database for an appropriate policy.

  o If a policy is found then construct a SPP-Reply message per section 9.3 with appropriate payloads

  o If a policy is not found then construct a SPP-Reply message per section 9.3 with appropriate payloads and error code.

6. If the message is for a node that is not part of the server's security domain then:

- Check the Cache database for any relevant policies that apply to this communication.

  o If a policy is found then construct a SPP-Reply message per section 9.3 with appropriate payloads

  o If a policy is not found then continue.

- Check the Local database for any relevant policies that apply to this communication.

- If a matching policy is found:

- o Generate a new SPP-Query message repeating all payloads and applicable fields. The Sender Address will be the address of the server. The destination for this message MUST be the one in the original SPP-Query received.

- o Keep state. Upon reception of the corresponding SPP-Reply the policy server MUST send an SPP-Reply addressed to sender of the original SPP-Query.

- If a matching policy is not found then:

- o The policy server MUST send the SPP-Query message unchanged. The server MAY validate the authenticity of the message, if neccessary, before forwarding it.

9.3 Reply Message (SPP-Reply) Processing

When creating a SPP-Reply message, the policy server MUST do the following:

1. Copy the Message ID value from the corresponding SPP-Query message into the Message ID field.
2. Set the value of the MTYPE field to 02
3. Set the MCODE value.
4. Set the QCOUNT and RCOUNT fields. All fields MUST be set to (00)hex when their corresponding payloads do no exist.
5. Set the flag bits accordingly and set the RESERVED field to (0).
6. Calculate and place the timestamp value used for anti-replay attack protection.
7. Set the IDENTITY_TYPE and IDENTITY of the Sender of the SPP message.
8. Copy the Query payloads from the SPP-Query message to the SPP-Reply message.
9. Construct the SPP data payload. Unless there is an error, at least one record corresponding to each Query payload MUST be present.
10. A policy server record and a CERT record MUST be added to the SPP-Reply message if the server is not authoritative for the source of the query (e.g., the address in the Source Address Data field of the COMSEC Query payload) and,

    1) The policy server is authoritative for the address found in the Destination Address Data field of the query, or

2) The policy server is authoritative for the host in the Sender ID of the previous reply, or
3) The policy server is authoritative for the host in the Sender ID of the query to which this is a reply.

11. If the Signature payload is required for message integrity and authentication, calculate a signature over the SPP header, SPP payloads, the PCL, TYPE, and LENGTH fields of the Signature payload. If present, the Signature payload MUST be the last payload in the message.
12. The SPP-Reply message is sent to the host that is listed in the Sender ID field of the SPP-Query to which this is a reply.

When a host or security gateway receives an SPP-Reply message it MUST do the following:

1. Read the Message ID field. If the value does not match the value of an outstanding SPP-Query message from a policy server then:
 - silently discard the message and the event MAY be logged.

2. If Message ID matches, Check the timestamp field for anti-replay protection. If a replayed message is detected the message is silently discarded and the event MAY be logged.

3. Establish if the message requires signature validation by seraching for a Signature payload:
 - if signature validation is required proceed with step 4.
 - if signature validation is not required proceed with step 6.

4. Validate the signature on the message.

 - Fetch certificate or key corresponding to the IP address found in the sender field of the SPP header.

 - If a certificate or key is not available the entity MAY:

  o choose, depending on configuration, to abort validation process, discard the message and MAY log the EVENT.

  o send an SPP-query message to the source of the IP address found in the sender field of the SPP header with a CERT query payload. Keep SPP-Reply message until the corresponding

SPP_Reply returns. Extract certificate or key, validate it and proceed.

5. Once a validated certificate or key is available then validate signature.

   If validation fails:
   - the message is silently discarded and the EVENT MAY be logged

   If validation succeeds, continue processing.

6. For Policy Servers, validate the chain of trust:

   - For each Policy Server record, verify that the Policy Server is authoritative over the Node. This may be done using information contained in certificates.

   - Use the Policy Server records to Create a chain of hosts from the destination host to this policy server where two records are linked if the Node in one is the Policy Server in another.

   - If the chain has no breaks, then this policy server MUST be authoritative over the sender of the reply, otherwise drop the message and stop processing it.

   - If the chain has one break, then the last policy server on the chain MUST be the sender of the reply, otherwise drop the message and stop processing it.

   - If the chain has two or more breaks, then there is an error in the chain so drop the message and stop processing it.

   - Verify that the Policy Server that is authoritative over the destination host is authoritative over ALL destination hosts in the reply.

7. If MCODE value is 2-10:

   For hosts or security gateways:
   - clear all the state and stop processing
   For policy servers:
   - create an SPP-Reply message using the same MCODE value.

8. If the reply received correponds to a Cert query and the MCODE is

either (01)hex, (08)hex (accept or partially unavailable) process message that was held up waiting for the cert.

9. For Policy Servers:

- Search the Local Policy Database for a policy entry that matches the policy expressed in Query payload.

- Merge the local and non-local policy information using the policy resolution proccess outlined in section 11.

- If the merge fails send an SPP-Reply message with MCODE 10, Communication Prohibited

- If the merge succeeds:

  o cache policy information. This includes all Query and Record payloads.
  o send an SPP-Reply message with the resulting policy records

10. For hosts or security gateways:

- verify that the information in the Record payload corresponds to the information in the Query payload.
- extract the records and create a policy entry in the SPD according to local format. The policy is entered in the SPD ONLY if local configuration allows.

9.4 Policy Message (SPP-Pol) Processing

When creating a SPP-Pol message, the entity (host, security gateway, or policy server) MUST do the following:

1. Generate the Message ID value. This value starts at "zero" and MUST be incremented by (01)hex with every new message.
2. Set the value of the MTYPE field to 03.
3. Set the MCODE value to (00)hex.
4. Set the RCOUNT field. The QCOUNT field MUST be set to (00)hex since no query payloads exist.
5. Set the flag bits accordingly and set the RESERVED field to (0).
6. Calculate and place the timestamp value used for anti-replay attack protection.
7. Set the IDENTITY_TYPE and IDENTITY of the Sender of the SPP message.
8. Construct the SPP data payloads. A Record payload MUST be present in this message. Query payloads MUST NOT be present.

9. If the Signature payload is required for message integrity and authentication, calculate a signature over the SPP header, SPP payloads, the PCL, TYPE, and LENGTH fields of the Signature payload. If required, the Signature payload MUST be the last payload in the message.

When a policy server receives an SPP-Pol message it MUST do the following:

1. Check for SPP access control. If enabled, read the IP address in the Sender's field of the SPP header.

   - Verify whether or not the message is allowed. If the message is not allowed then:

     o send an SPP-Pol_Ack message with the error code 02
     o discard message and log EVENT if configured to do so

   - If the message is allowed, continue.

2. Check timestamp field for anti-replay protection. If a replayed message is detected:

   - send an SPP_Reply message with the error code 03
   - discard message and log EVENT if configured to do so

3. If the message requires signature validation.

   - Fetch certificate or key corresponding to the IP address found in the sender field of the SPP header.

   - If a certificate or key is not available the entity MAY, depending on configuration:

     o choose to abort validation process, send SPP-Pol_Ack message with error code 05, discard the message and MAY log the EVENT.

     o send an SPP-Query message to the source of the IP address found in the sender field of the SPP header with a CERT Query payload. Keep SPP-Pol message until the SPP_Reply returns. Extract certificate or key, validate it and proceed.

   - Once a validated certificate or key is available then validate signature.

     o If validation fails the message is silently discarded and the

EVENT MAY be logged

4. If signature was not required or upon successful validation of a signature parse the payloads.

5. For hosts and security gateways:

- extract the records and create a policy entry in the cache database. The policy MAY be entered in the SPD, also, ONLY if cofiguration allows.

6. For Policy Servers:

- extract the records and create a policy entry in the cache database.

9.5 Policy Acknowledgement Message (SPP-Pol_Ack) Processing

When creating a SPP-Pol_Ack message, the policy server MUST do the following:

1. Copy the Message ID value from the corresponding SPP-Pol message into the Message ID field.
2. Set the value of the MTYPE field to 04
3. Set the MCODE value.
4. Set the QCOUNT and RCOUNT fields. All fields MUST be set to (00)hex.
5. Set the flag bits accordingly and set the RESERVED field to (0).
6. Calculate and place the timestamp value used for anti-replay attack protection.
7. Set the IDENTITY_TYPE and IDENTITY of the Sender of the SPP message.
8. Query or Record payloads MUST NOT be present.
9. If the Signature payload is required for message integrity and authentication, calculate a signature over the SPP header, the PCL, TYPE, and LENGTH fields of the Signature payload.

When a host, security gateway or policy server receives an SPP-Pol_Ack message the entity MUST do the following:

1. Read the Message ID field. If the value does not match the value of an outstanding SPP-Pol message from a policy server then:
   - silently discard the message and the EVENT MAY be logged.

2. If Message ID matches then check the timestamp field for anti-replay

protection. If a replayed message is detected the message is silently discarded and the EVENT MAY be logged.

3. If the message is original (not replayed) and the message requires signature validation then:

   - Fetch certificate or key corresponding to the IP address found in the sender field of the SPP header.

   - If a certificate or key is not available the entity MAY:

     o choose, depending on configuration, to abort validation process, discard the message and MAY log the EVENT.

     o send an SPP-query message to the source of the IP address found in the sender field of the SPP header with a CERT Query payload. Keep SPP-Pol_ack message until the SPP_Reply returns. Extract certificate or key, validate it and proceed.

4. Once a validated certificate or key is available then validate the signature.
   If validation fails:
   - the message is silently discarded and the event MAY be logged
   If validation succeeds:
   - read the MCODE field and proceed accordingly. If no errors, clear all the state for this message and proceed.

9.6 KeepAlive Message (SPP-KEEP_ALIVE) Processing

When creating an SPP-Keep_Alive message, the policy server MUST do the following:

1. Generate the Message ID value. This value starts at "zero" and MUST be incremented by (01)hex with every new message.
2. Set the value of the MTYPE field to (06)hex.
3. Set the MCODE value to (00)hex.
4. Set the QCOUNT and RCOUNT fields. All fields MUST be set to (00)hex.
5. Set the flag bits accordingly and set the RESERVED field to (0).
6. Calculate and place the timestamp value used for anti-replay attack protection.

7. Set the IDENTITY_TYPE and IDENTITY of the Sender of the SPP message.
8. Query or Record payloads MUST NOT be present.
9. If the Signature payload is required for message integrity and authentication, calculate a signature over the SPP header, the PCL, TYPE, and LENGTH fields of the Signature payload.

When a host or security gateway receives an SPP-Keep_Alive message it MUST do the following:

1. Check for SPP access control. If enabled, read the IP address in the Sender's field of the SPP header.

   - Verify whether or not the message is allowed. If the message is not allowed then discard message and log EVENT if configured to do so.

2. Check timestamp field for anti-replay protection. If a replayed message is detected discard message and log EVENT if configured to do so.

3. If the message requires signature validation then:

   - Fetch certificate or key corresponding to the IP address found in the sender field of the SPP header.

   - If a certificate or key is not available the entity MAY depending on configuration:

     o choose to abort validation process, discard the message and perhaps logged the event.

     o send an SPP-Query message to the source of the IP address found in the sender field of the SPP header with a CERT Query payload. Keep SPP-Keep_Alive message until the SPP-Reply returns. Extract certificate or key, validate it and proceed.

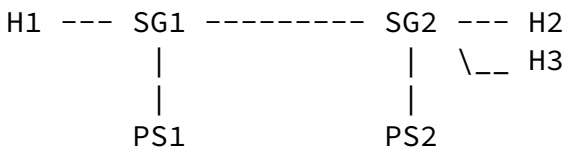
   - Once a validated certificate or key is available then validate signature.

     o If validation fails the message is silently discarded and the event MAY be logged

4. If signature validation was not required or upon successful validation of a signature, the EVENT MAY be logged.

10. Policy Decorrelation Process

It is not possible for a Policy Server to use policies as they are written in the SPS master file, since those policies are likely to be correlated. Two policies are correlated if there is a non-nil intersection between the values of each of their selectors. Caching correlated policies can lead to incorrect policy implementation based on those cached policies, as the following example shows.



PS2 contains the following policies in its master file:

	src	dst	proto	direction	action
1)	*	H2	*	inbound	permit
2)	*	*	*	inbound	deny

These two policies are correlated since all the selectors (src, dst, proto, and direction) overlap. The following SPP exchanges occur:

1) PS1 requests policy for H3.
2) PS2 returns policy #2 to PS1 which then caches policy #2.
3) PS1 now looks up the policy for H2 and discovers that it already has a matching policy (policy #2) and uses that.

This is clearly wrong, since policy #2 indicates that the communication to H2 should be denied, though PS2's policy actually indicates that it should be allowed.

The solution is to remove the ambiguities that may exist in the master file. The policies of the master file MUST be decorrelated before they are entered into the Local Policy Database. That is, the policies must be rewritten so that for every pair of policies there exists a selector for which there is a nil intersection between the values in both of the policies.

The policies in the above example could be decorrelated as follows:

```
        src     dst     proto   direction       action
   1')  *       H2      *       inbound         permit
   2')  *       not H2  *       inbound         deny
```

Now the exchange is a bit different:

1) PS1 requests policy for H3.
2) PS2 returns policy #2' to PS1 which then caches policy #2'.
3) PS1 now looks up the policy for H2, doesn't have a matching policy, so it requests a policy for H2.
4) PS2 returns policy #1' to PS1 which then caches policy #1', which is the correct policy for H2.

All SPAs and SPRs, therefore, MUST decorrelate their master files before using those policies for SPP. Once the policies are decorrelated, there is no longer any ordering requirement on the policies, since only one policy will match any requested communication. The next section describes decorrelation in more detail and presents an algorithm that may be used to implement decorrelation.

10.1 Decorrelation Algorithm

The basic decorrelation algorithm takes each policy in a correlated set of policies and divides it up into a set of policies using a tree structure. Those of the resulting policies that are decorrelated with the uncorrelated set of policies are then added to that decorrelated set.

The basic algorithm does not guarantee an optimal set of decorrelated policies. That is, the policies may be broken up into smaller sets than is necessary, though they will still provide all the necessary policy information. Some extensions to the basic algorithm are described later to improve this and improve the performance of the algorithm.

```
C  A set of ordered, correlated policies
Ci The ith policy in C.
U  The set of uncorrelated policies being built from C
Ui The ith policy in U.
```

A policy P may be expressed as a mapping of selector values to actions:

```
Pi = Si1 x Si2 x ... x Sik -> Ai
```

1) Put C1 in set U as U1

For each policy Cj (j > 1) in C

2) If Cj is uncorrelated with every policy in U, then add it to U.

3) If Cj is correlated with one or more policies in U, create a tree rooted at the policy Cj that partitions Cj into a set of uncorrelated policies. The algorithm starts with a root node where no selectors have yet been chosen.

A) Choose a selector in Cj, Scjn, that has not yet been chosen when traversing the tree from the root to this node. If there are no selectors not yet used, continue to the next unfinished branch until all branches have been completeded. When the tree is completed, go to step D.

T is the set of policies in U that are correlated with the policy to this node.

The policy at this node is the policy formed by the selector values of each of the branches between the root and this node. Any selector values that are not yet represented by brances assume the corresponding selector value in Cj, since the values in Cj represent the maximum value for each selector.

B) Add a branch to the tree for each value of the selector Scjn that appears in any of the policies in T. (If the value is a superset of the value of Scjn in Cj, then use the value in Cj, since that value represents the universal set.) Also add a branch for the compliment of the union of all the values of the selector Scjn in T. When taking the compliment, remember that the universal set is the value of Scjn in Cj. A branch need not be created for the nil set.

C) Repeat A and B until the tree is completed.

D) The policy to each leaf now represents a policy that is a subset of Cj. The policies at the leaves completely partition Cj in such a way that each policy is either completely overriden by a policy in U, or is uncorrelated with the policies in U.

Add all the uncorrelated policies at the leaves of the tree to U.

5) Get next Cj and goto 2.

6) When all policies in C have been processed, then U will contain an uncorrelated version of C.

There are several optimizations that can be made to this algorithm. A few of them are presented here.

It is possible to optimize, or at least improve, the amount of branching that occurs by carefully choosing the order of the selectors used for the next branch. For example, if a selector Scjn can be chosen so that all the values for that selector in T are equal to or a superset of the value of Scjn in Cj, then only a single branch need to be created (since the compliment will be nil).

Branches of the tree do not have to procede with the entire decorrelation algorithm. For example, if a node represents a policy that is uncorrelated with all the policies in U, then there is no reason to continue decorrelating that branch. Also, if a branch is completely overridden by a policy in U, then there is no reason to continue decorrelating the branch.

An additional optimization is to check to see if a branch is overridden by one of the CORRELATED policies in set C that has alread been decorrelated. That is, if the branch is part of decorrelating Cj, then check to see if it was overridden by a policy Cm, m < j. This is a valid check, since all the policies Cm are already expressed in U.

Along with checking if a policy is already uncorrelated in step 2, check if Cj is overriden by any policy in U. If it is, skip it since it is not relevant. A policy x is overriden by another policy y if every selector in x is equal to or a subset of the corresponding selector in policy y. Appendix B shows an example of applying the algorithm to a set of correlated policies.

11. Policy Resolution Process

When a policy server receives a reply (Section 9.3), it merges its local policy for the communication with any non-local policies contained in the reply. The merging process creates a new policy that is the intersection of the local and remote policies. It then uses the merged policy as its reply to the queryand caches it. The policy resolution process is as follows:

1. Get local and remote policies for the requested communication.

2. Verify that the remote policy answer the query. This may be accomplished by intersecting the query with each of the answers to the query and verify that they have a non-nil intersection.

3. For each pair of endpoints described by the policy:

- Find the intersection of the policies between these endpoints.

  o If the intersection is nil, then the policies do not permit the communication and an error should be returned. It is not necessary to continue processing other endpoints.

  o If the intersection is not nil, then the intersection should be added to the reply policy.

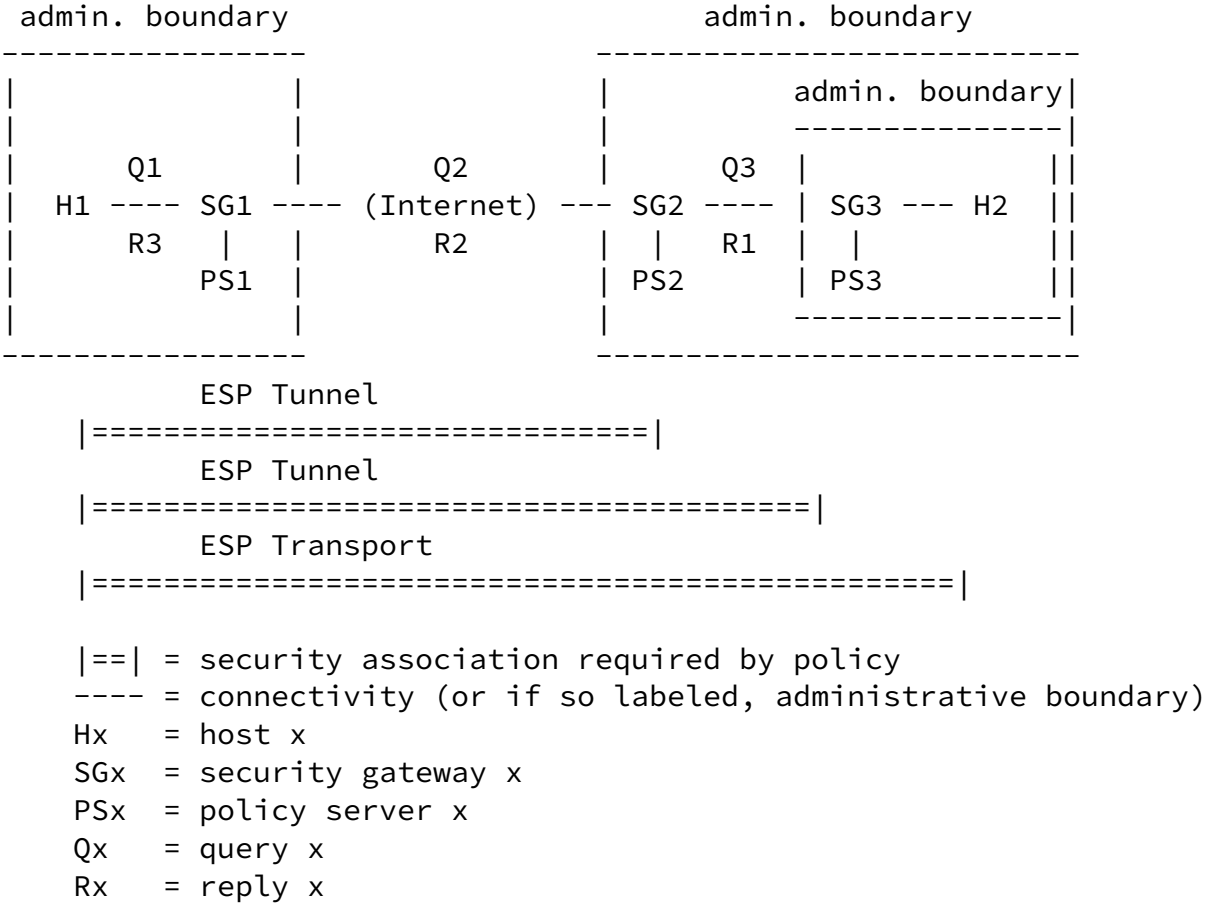
4. The policy created by the intersections is the policy that should be cached and used as a reply to the query.

Step 3 requires that the policy server must be able to determine all the sets of endpoints described by the policy. The endpoint information comes from two places: the source and destination addresses in the query (which is possibly more specific than those fields in the policies) and the location information in the ipsec_action attribute.

The location information may offer the policy server some flexibility in how it interprets endpoints for the communication. For example, if the policy indicates a tunnel must be established with any host or gateway in the source or destination host's domain, the policy server can choose the endpoint within the bounds of the policy. This choice can be made randomly, using a set policy (e.g., always choose the outermost gateway permitted by the policy), or using additional information the server may maintain for this purpose. For example, the server may keep track of previous policy decisions it made and use those as hints to which security associations may already exist. It can then try to make decisions that will allow these security associations to be reused.

12. Usage of SPS with IPSec

This section illustrates how the Security Policy System operates and interacts with IPSec. It describes, step-by-step, the process from the inital application call to the establishment of the necessary security associations.



1. User application attempts to send a message from H1 to H2 (e.g., finger somebody@H2)

2. IPSec on H1 gets the packet and finds a policy for it in the Security Policy Database (SPD).

3. H1 does not have a security association for the communication and asks the Key Management Protocol to establish the needed security association.

4. The policy client in H1 is queried for the policy governing the communication between H1 and H2.

5. H1's policy client creates an SPP query, Q1, and sends it to PS1, its configured policy server.

6. PS1 receives the query. Its security domain database indicates that it is not authoritative over H2 so it checks its cache to see if it has a cached answer. For this example, it does not, so it creates a new SPP query, Q2, with the query and sends it to H2.

7. SG2 intercepts Q2 and passes it to PS2.

8. PS2 receives the query. Its security domain database indicates that it is not authoritative over H2 and PS2 determines it wants to be involved in the communication. It checks its cache to see if it has a cached answer. For this example, it does not, so it creates a new SPP query, Q3, with the query and sends it to H2.

9. SG3 intercepts Q3 and passes it to PS3.

10. PS3 receives the query. It checks its security domain database and determines that it is authoritative over H2 so it will send a reply. It checks its cache to see if it has a cached answer. For this example, it does have one cached from previous information sent to it by H2. The cached policy indicates ESP transport must be done with H2 and ESP tunnel must be done with SG3.

11. PS3 creates two messages. One is an SPP reply message, R1, with the policy indicating the required security associations, a security server record indicating PS3 is authoritative over H2, and PS3's certificate in a certificate record. The reply is sent to PS2.

    The second message is an SPP policy message to signal SG3 that it will need a security association with H1.

12. PS2 receives the R1. It verifies that PS3 is authoritative over H2. It looks up its local policy and resolves it with the policy in the reply. It caches the resolved policy and creates two messages. One is the reply to PS1, R2, that contains the resolved policy, PS3's security server and certificate records and a security server record that states that PS2 is authoritative over PS3 and PS2's certificate.

    The second message is an SPP policy message to signal SG2 that it will need a security association with H1.

13. PS1 receives the R2. It verifies that PS2 is authoritative over PS3

and PS3 is authoritative over H2, forming a valid chain of trust. It looks up its local policy and resolves it with the policy in the reply. It caches the resolved policy and creates R3, a reply to H1. R3 contains the resolved policy (the security server and certificate records are no longer needed).

14. The policy client receives the R3 and returns it to the application that queried for it. The policy indicates that the three security associations must be established and they must be established in a particular order.

15. The KMP is given this information and first creates a security association with PS2 which it can use to set up the security association with PS3. They both can be used to set up the security association with H2.

16. Finally, the original message from the application can proceed using the security associations.

If H2 wanted to get directly involved in the communication, PS3 would not be authoritative over H2, H2 would be authoritative over itself. There would then be another pair of messages where PS3 sends a query to H2 which H2 receives and sends a reply back to PS3 and the processing continues as outlined above.

Acknowledgments

The authors thank Charles Lynn, Steve Kent and John Zao for their participation in requirements discussions for the Security Policy System. Our gratitude to Charlie Lynn, Matt Fredette, Alden Jackson, Dave Mankins, Marla Shepard and Pam Helinek for the contributions to this document. We thank Mary Hendrix (INS Corp.) for reviewing this document. We thank Isidro Castineyra for his contributions to the early parts of this work.

References

[Bra97] Bradner, S., "Key Words for use in RFCs to indicate Requirement Levels", RFC2119, March 1997.

[Kent98] S. Kent, R. Atkinson, "Security Architecture for the Internet Protocol", Internet Draft draft-ietf-ipsec-arch-sec-07, July 1998.

[KA98b] S. Kent, R. Atkinson, "IP Encapsulating Security Payload (ESP)", Internet Draft, July 1998.

[isakmp] D. Maughan, M. Schertler, M. Schneider, J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)", Internet Draft draft-ietf-ipsec-isakmp-10, July 3, 1998

[ALX98] A. Vopilov, "The Locating an IPSec Gateway Algorithm", Working Draft, February 1998.

[RFC1305] Mills, D., "Network Time Protocol (Version 3): Specification, Implementation and Analysis", RFC 1305, March 1992.

[RFC2230] R. Atkinson, "Key Exchange Delegation Record for the DNS", RFC 2230, The Internet Society, November 1997

[PKIXP1] R. Housley, W. Ford, W. Polk, D. Solo, "Internet Public Key Infrastructure: X.509 Certificate and CRL Profile". Internet Draft draft-ietf-pkix-ipki-part1-10, September 1998.

[PolMod] R. Pereira, P. Bhattacharya, "IPSec Policy Data Model", Internet Draft draft-ietf-ipsec-policy-model-00, February 1998

[Harkins98] D. Harkins, D. Carrel, "The Internet Key Exchange (IKE)", Internet Draft draft-ietf-ipsec-isakmp-oakley-08, June 1998.

[Piper98] D. Piper, "The Internet IP Security Domain of Interpretation for ISAKMP", Internet Draft draft-ietf-ipsec-ipsec-doi-10.txt, July 1998

[SPSL] M. Condell, C. Lynn, J. Zao "Security Policy Specification Language", Internet Draft draft-ietf-ipsec-spsl-00.txt, November 1998

APPENDIX A

DATA_TYPE Definitions

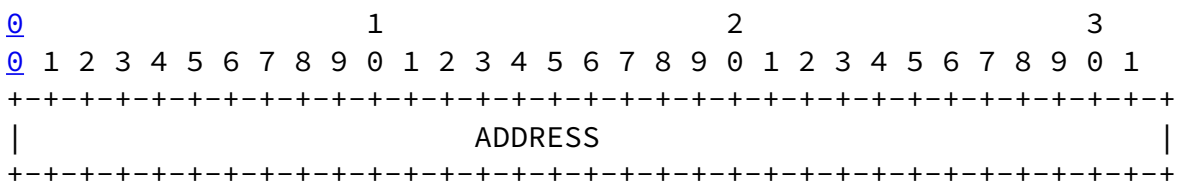
The encoding of each selector and SA attribute is decribed here. Attributes generally encode "any" in one of two ways. If using the TLV format (X = 0) then the length is set to 0 to indicate any. If the TV format (X = 1) is used, then the value is set to 0;

Lists are generally expressed by setting the length of the value to a multiple of the length of a single data value. The multiple used is the number of elements in the list. If the selector or attribute may express a list, each element is expressed the same as the element described in DATA_VALUE.

This section describes the values and DATA_VALUE encoding for each selector and SA attribute.

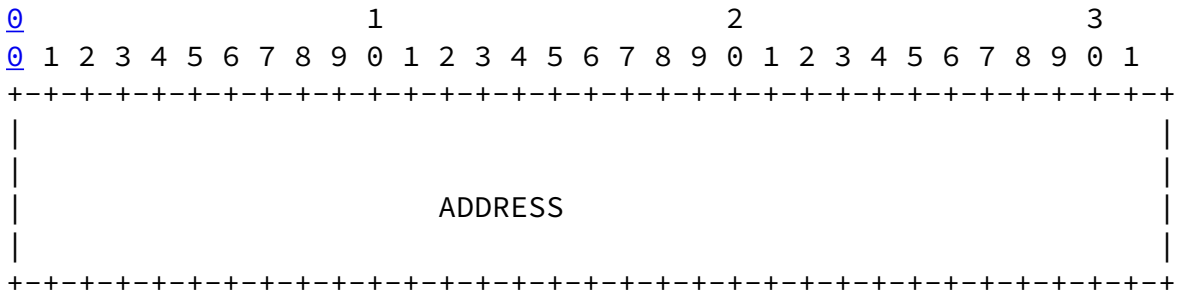
A.1 IPV4_ADDR

X	0
DATA_TYPE	1
LENGTH	4 if an IP address is present, 0 if no IP address is present.
list	Yes
DATA_VALUE	



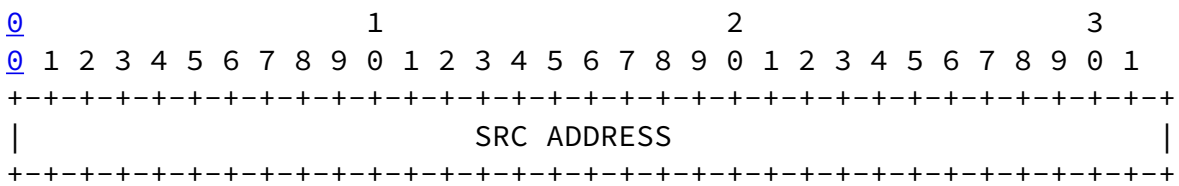
A.2 IPV6_ADDR

X	0
DATA_TYPE	2
LENGTH	16 if an IP address is present, 0 if no IP address is present.
list	Yes
DATA_VALUE	



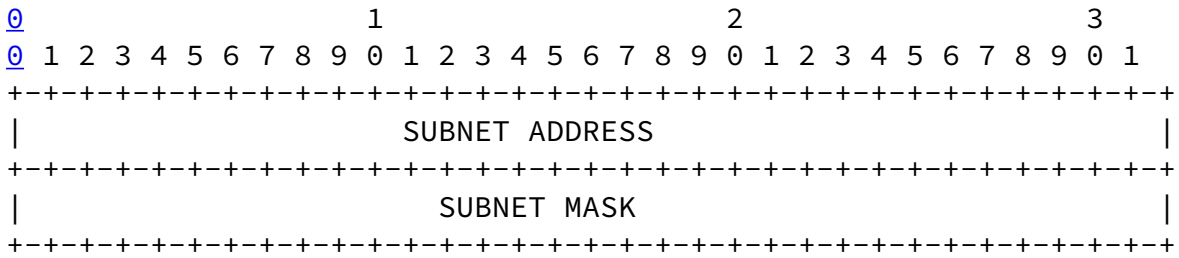
A.3 SRC_IPV4_ADDR

X	0
DATA_TYPE	3
LENGTH	4 times the number of addresses in the list. A length of 0 indicates any address.
list	Yes
DATA_VALUE	



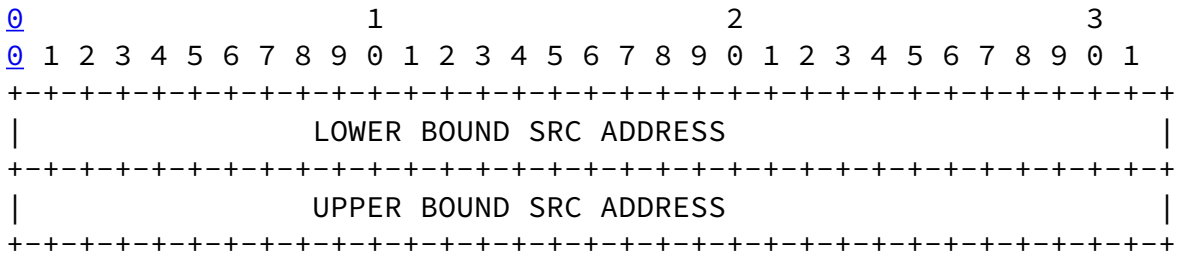
A.4 SRC_IPV4_ADDR_SUBNET

X	0
DATA_TYPE	4
LENGTH	8 times the number of subnets in the list. A length of 0 indicates any subnet.
list	Yes
DATA_VALUE	



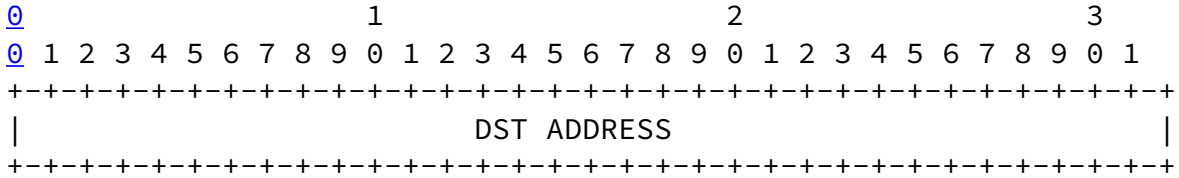
A.5 SRC_IPV4_ADDR_RANGE

X	0
DATA_TYPE	5
LENGTH	8 times the number of address ranges in the list. A length of 0 indicates any address.
list	Yes
DATA_VALUE	



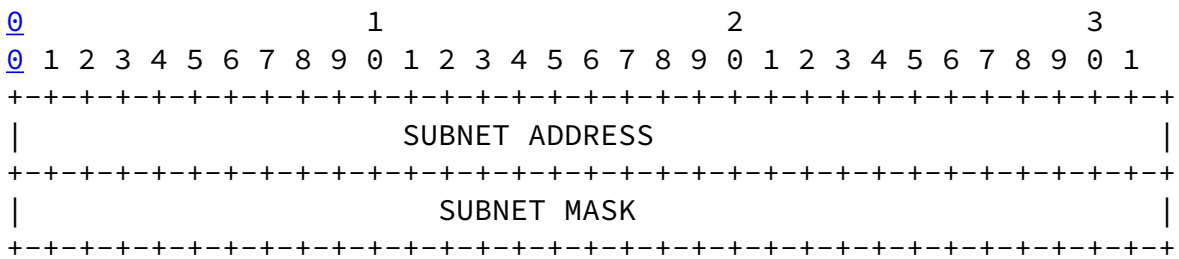
A.6 DST_IPV4_ADDR

```
X               0
DATA_TYPE       6
LENGTH          4 times the number of addresses in the list.
                A length of 0 indicates any address.
list            Yes
DATA_VALUE
```



A.7 DST_IPV4_ADDR_SUBNET

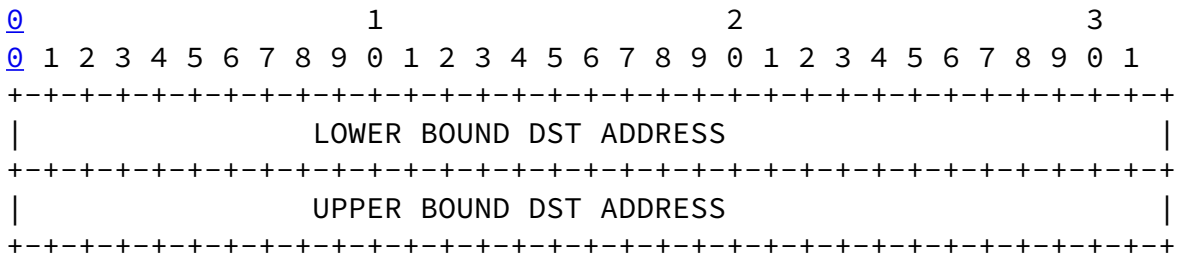
```
X               0
DATA_TYPE       7
LENGTH          8 times the number of subnets in the list.
                A length of 0 indicates any subnet.
list            Yes
DATA_VALUE
```



A.8 DST_IPV4_ADDR_RANGE

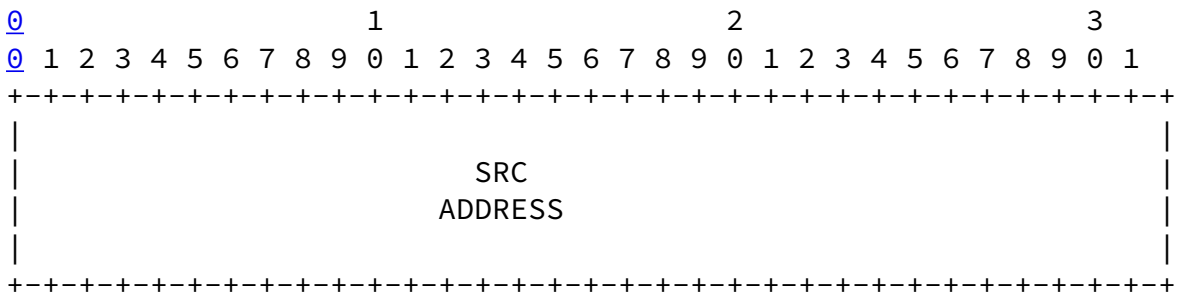
```
X               0
DATA_TYPE       8
LENGTH          8 times the number of address ranges in the list.
                A length of 0 indicates any address.
```

list Yes
DATA_VALUE



A.9 SRC_IPV6_ADDR

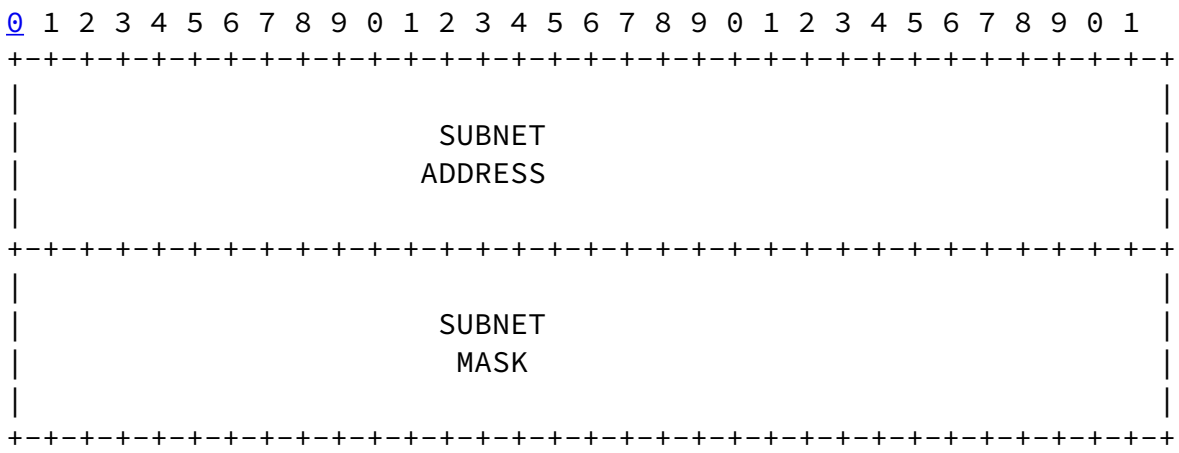
X 0
DATA_TYPE 9
LENGTH 16 times the number of addresses in the list.
A length of 0 indicates any address.
list Yes
DATA_VALUE



A.10 SRC_IPV6_ADDR_SUBNET

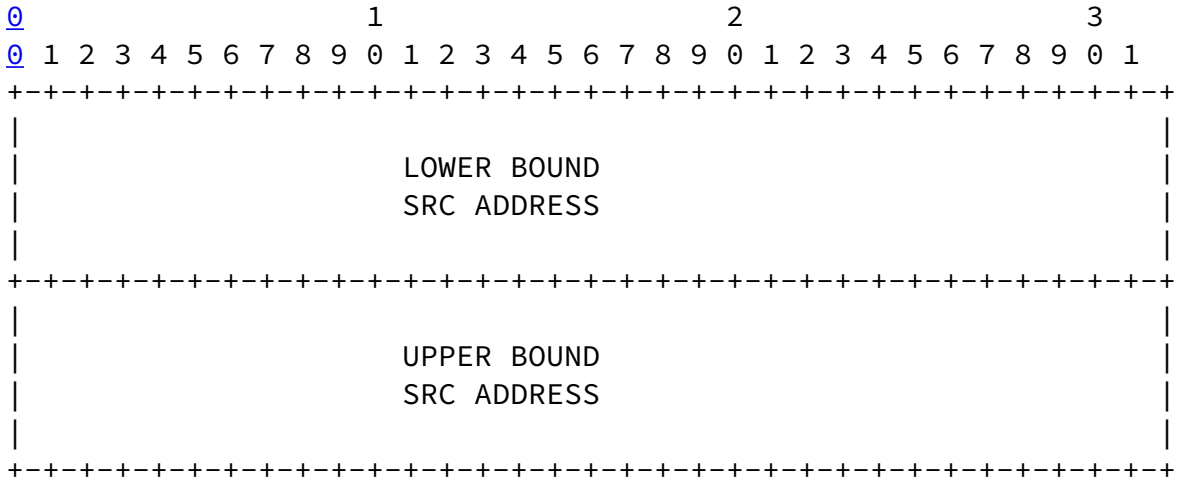
X 0
DATA_TYPE 10
LENGTH 32 times the number of subnets in the list.
A length of 0 indicates any subnet.
list Yes
DATA_VALUE

0 1 2 3



A.11 SRC_IPV6_ADDR_RANGE

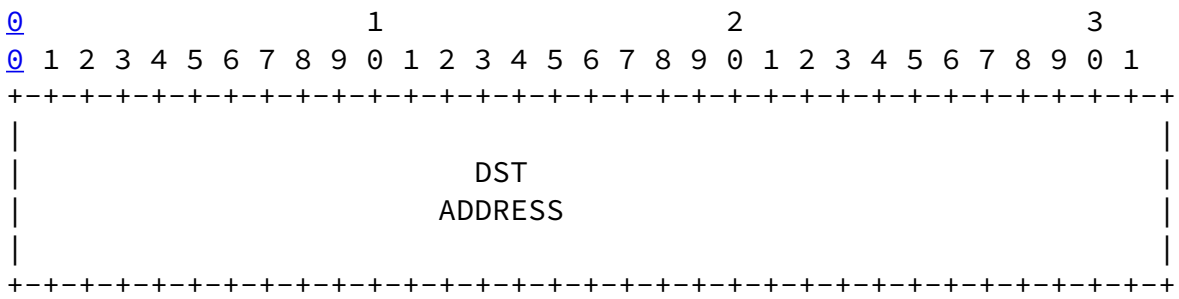
X	0
DATA_TYPE	11
LENGTH	32 times the number of address ranges in the list. A length of 0 indicates any address.
list	Yes
DATA_VALUE	



A.12 DST_IPV6_ADDR

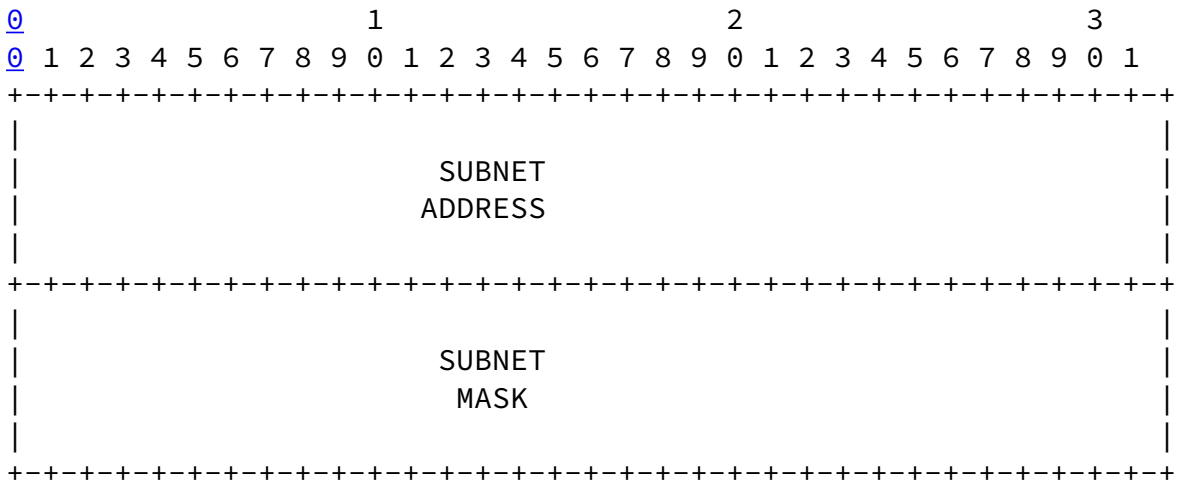
X	0
DATA_TYPE	12

LENGTH   16 times the number of addresses in the list.
                A length of 0 indicates any address.
list   Yes
DATA_VALUE



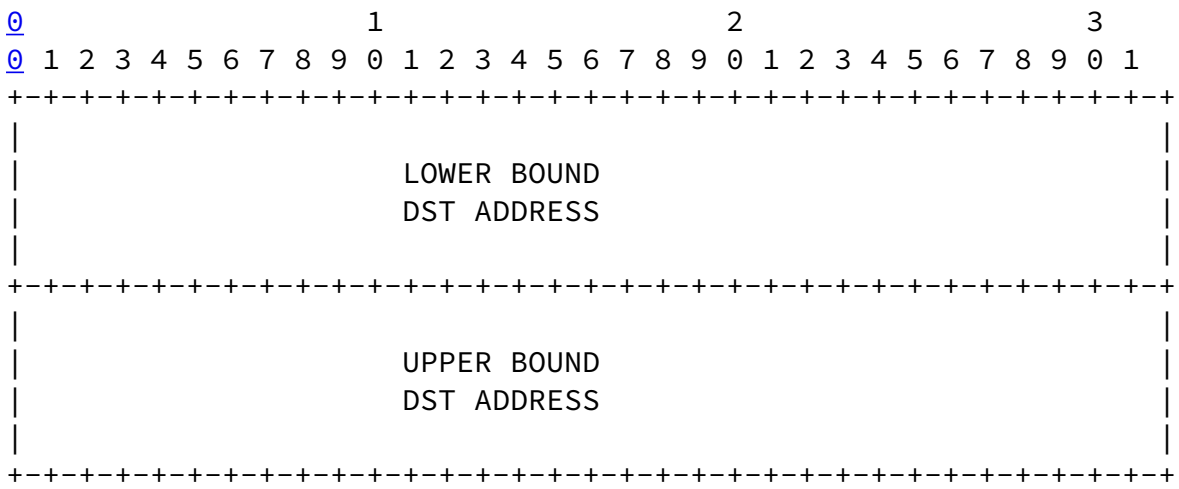
A.13 DST_IPV6_ADDR_SUBNET

X   0
DATA_TYPE   13
LENGTH   32 times the number of subnets in the list.
                A length of 0 indicates any subnet.
list   Yes
DATA_VALUE



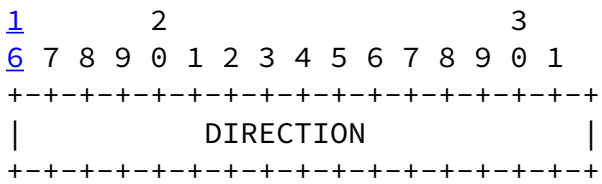
A.14 DST_IPV6_ADDR_RANGE

X   0
DATA_TYPE   14
LENGTH   32 times the number of address ranges in the list.
                A length of 0 indicates any address.
list   Yes
DATA_VALUE



A.15 DIRECTION

X 1
DATA_TYPE 15
LENGTH TV attribute, no length
list No
DATA_VALUE



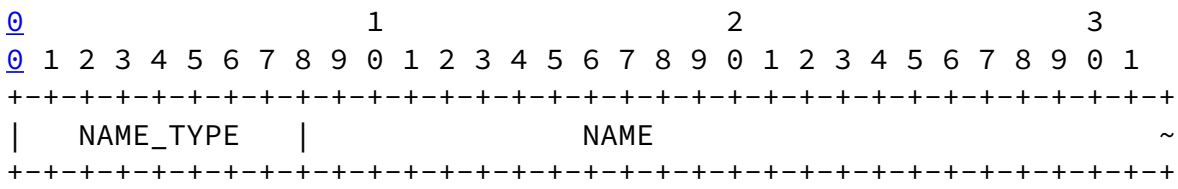
DIRECTION

In/Outbound	0
Inbound	1
Outbound	2

Direction is with respect to the senders interface.

A.16 USER_NAME

X 0
DATA_TYPE 16
LENGTH 1 plus the length of NAME
A length of 0 indicates any name.
list No
DATA_VALUE



NAME_TYPE

822_EMAIL	0
DIST_NAME	1

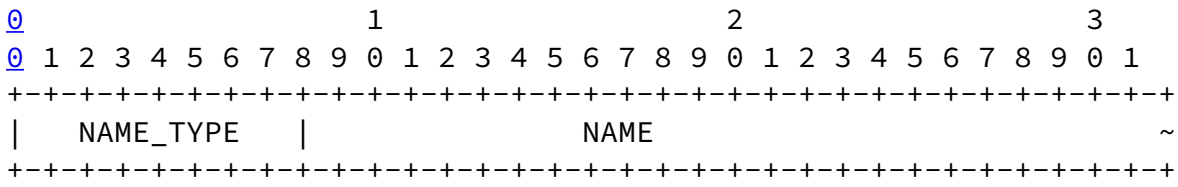
NAME

Name of type NAME.
[**** probably should describe in more detail]

A.17 SYSTEM_NAME

X	0
DATA_TYPE	17
LENGTH	1 plus the length of NAME A length of 0 indicates any name.
list	No

DATA_VALUE



NAME_TYPE

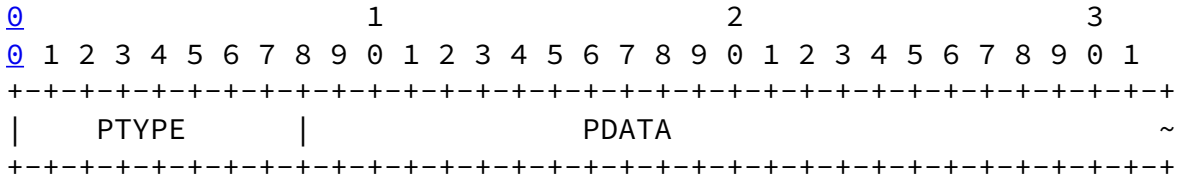
DNS_NAME	0
DIST_NAME	1
822_NAME	2
X400_ADDR	3
DIR_NAME	4
EDI_PARTY_NAME	5
URI	6
IPADDR	7
REGID	8
OTHER	255

NAME

Name of type NAME.
[***** probably should describe in more detail]

A.18 XPORT_PROTOCOL

X	0
DATA_TYPE	18
LENGTH	1 plus length of pdata A length of 0 indicates any address.
list DATA_VALUE	No (see below)



PTYPE Describes the rest of the data:

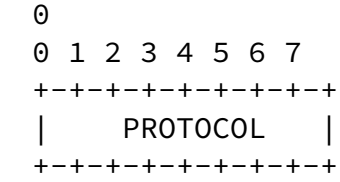
ANY	0
OPAQUE	1
LIST	2
RANGE	3

PDATA

Not used if PTYPE is ANY or OPAQUE

LIST

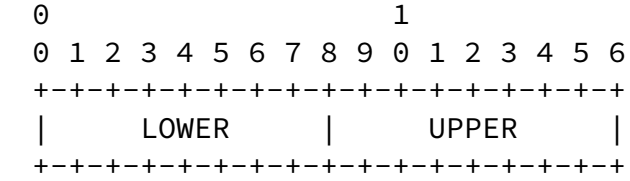
indicates a list whose elements look like the following:



The length of pdata to be used as part of the LENGTH field is 1 times the number of elements in the list.

RANGE

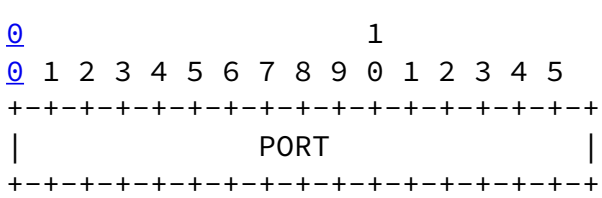
indicates a range of protocol values whose lower bound is LOWER, and upper bound is UPPER.



The length of pdata to be used as part of the LENGTH field is 2.

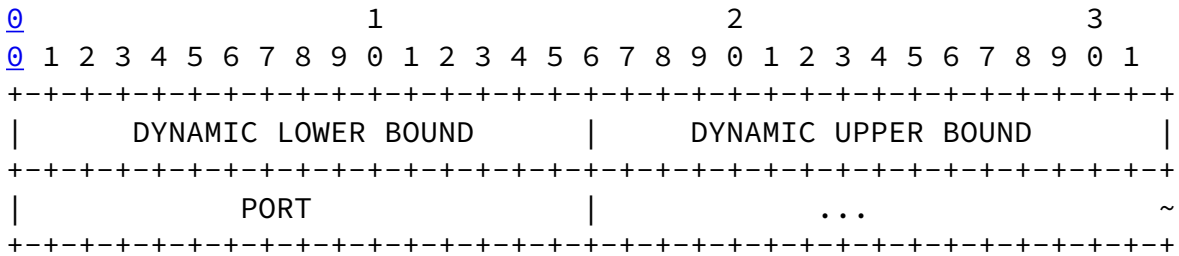
A.19 SRC_PORT

X	0
DATA_TYPE	19
LENGTH	2 times the number of ports in the list. A length of 0 indicates any port.
list	Yes
DATA_VALUE	



A.20 SRC_PORT_DYNAMIC

X	0
DATA_TYPE	20
LENGTH	4 plus 2 times the number of ports in the list. A length of 4 indicates any port.
list	See Below
DATA_VALUE	



The use of this attribute indicates that dynamic port allocation is permitted. Communications that are intitiated with any of the ports indicated, may then dynamically allocate any of the ports listed within the LOWER and UPPER BOUNDS, inclusive.

DYNAMIC LOWER BOUND
: Lower bound of the range of ports that may be dynamically allocated. If this and DYNAMIC UPPER BOUND are both 0, then any port may be dynamically allocated.

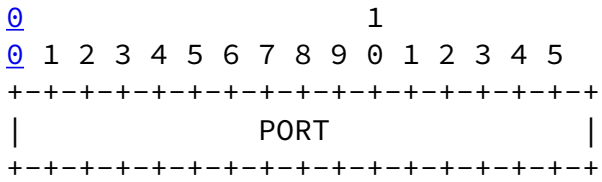
DYNAMIC UPPER BOUND
: Upper bound of the range of ports that may be dynamically

allocated. If this and DYNAMIC LOWER BOUND are both 0, then any port may be dynamically allocated.

PORT
Port that the communication must be initiated with. This may be a list of ports.

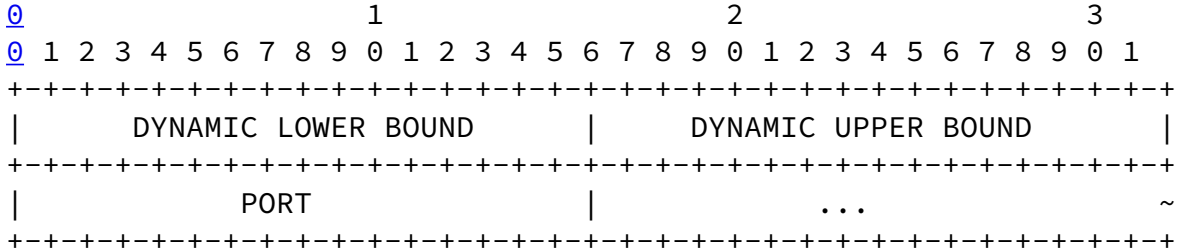
A.21 DST_PORT

X	0
DATA_TYPE	21
LENGTH	2 times the number of ports in the list. A length of 0 indicates any port.
list DATA_VALUE	Yes



A.22 DST_PORT_DYNAMIC

X	0
DATA_TYPE	22
LENGTH	4 plus 2 times the number of ports in the list. A length of 4 indicates any port.
list DATA_VALUE	See Below



The use of this attribute indicates that dynamic port allocation is permitted. Communications that are intitiated with any of the

ports indicated, may then dynamically allocate any of the ports listed within the LOWER and UPPER BOUNDS, inclusive.

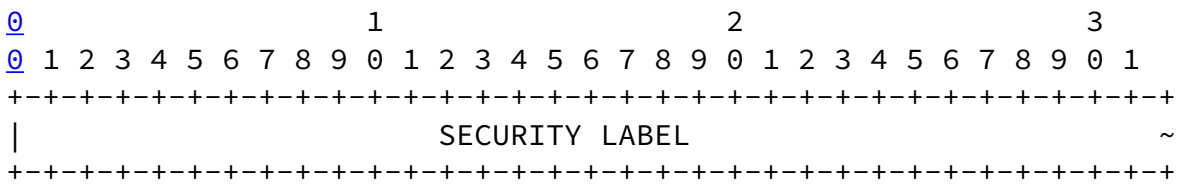
DYNAMIC LOWER BOUND
Lower bound of the range of ports that may be dynamically allocated. If this and DYNAMIC UPPER BOUND are both 0, then any port may be dynamically allocated.

DYNAMIC UPPER BOUND
Upper bound of the range of ports that may be dynamically allocated. If this and DYNAMIC LOWER BOUND are both 0, then any port may be dynamically allocated.

PORT
Port that the communication must be initiated with. This may be a list of ports.

A.23 SEC_LABELS

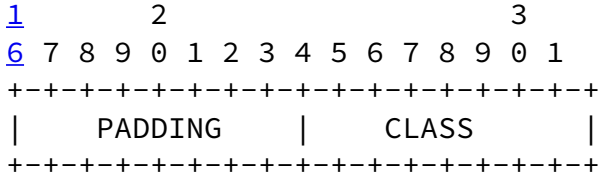
X 0
DATA_TYPE 23
LENGTH Variable.
list No
DATA_VALUE



A.24 V6CLASS

X 1
DATA_TYPE 24
LENGTH TV attribute, no length
list No

DATA_VALUE

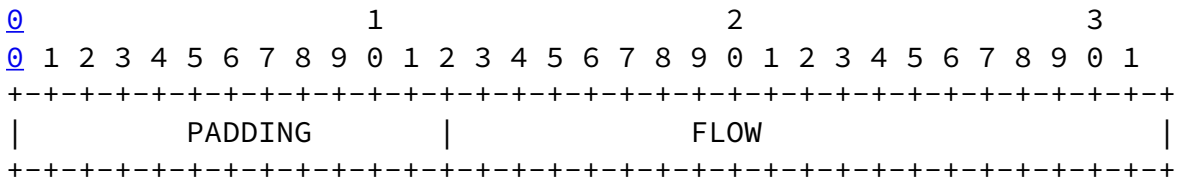


PADDING set to 0

CLASS   class value

A.25 V6FLOW

X   0
DATA_TYPE   25
LENGTH   4
list   No
DATA_VALUE

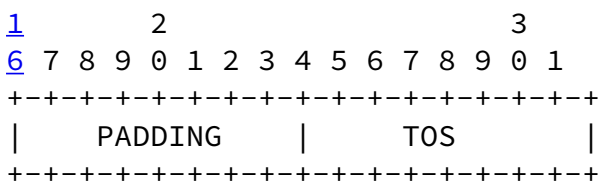


PADDING   set to 0

FLOW   set to flow value

A.26 V4TOS

X   1
DATA_TYPE   26
LENGTH   TV attribute, no length
list   No
DATA_VALUE



PADDING   set to 0

TOS   type of service value

A.27 ACTION

X   1
DATA_TYPE   27
LENGTH   TV attribute, no length
list   No

DATA_VALUE

```
1       2                   3
6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             ACTION            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

DIRECTION

Deny	0
Permit	1

A.28 SRC_PORT_RANGE

X	0
DATA_TYPE	28
LENGTH	4 times the number of port ranges in the list. A length of 0 indicates any port.
list	Yes

DATA_VALUE

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       PORT LOWER BOUND        |       PORT UPPER BOUND        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A.29 DST_PORT_RANGE

X	0
DATA_TYPE	29
LENGTH	4 times the number of port ranges in the list. A length of 0 indicates any port.
list	Yes

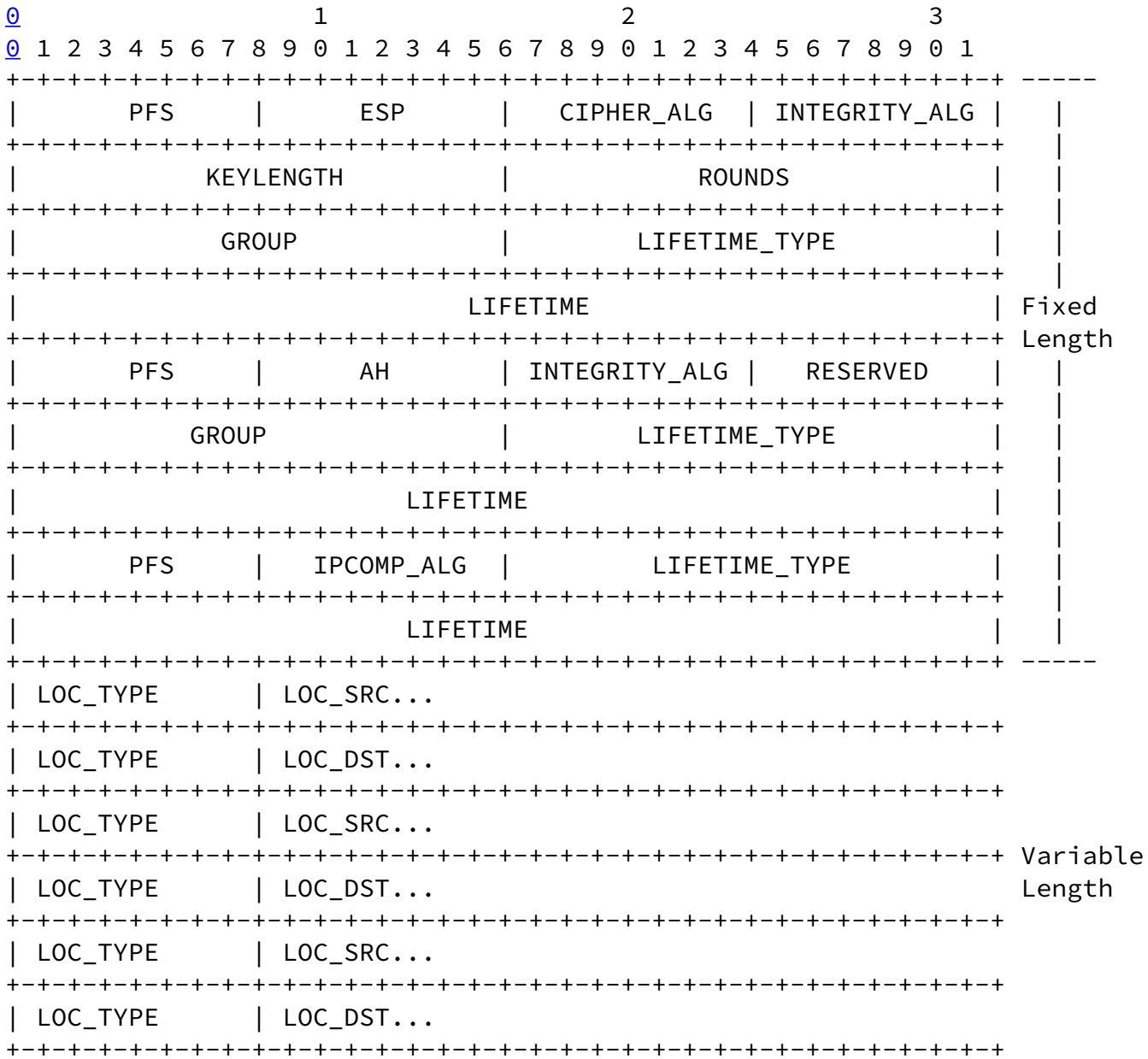
DATA_VALUE

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       PORT LOWER BOUND        |       PORT UPPER BOUND        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A.30 IPSEC_ACTION

X	0
DATA_TYPE	50
LENGTH	Variable
list	Yes

DATA_VALUE



PFS

FALSE	0
TRUE	1

ESP

NOT_REQUIRED	0
TUNNEL_MODE	1
TRANSP_MODE	2

CIPHER_ALG

NONE	0
ANY	1
RFC1829_IV64	2
DES	3
DES3	4
RC5	5
IDEA	6
CAST	7

BLOWFISH	8
3IDEA	9
RFC1829_IV32	10
RC4	11

KEYLENGTH

The first octet corresponds to the minimum value and the second octet corresponds to the maximum value. If no range exist the first octet indicates the keylength. The second octet contains a value of (00)hex.

ROUNDS

The first octet corresponds to the minimum value and the second octet corresponds to the maximum value. If no range exist the first octet indicates the rounds. The second octet contains a value of (00)hex.

INTEGRITY_ALG

NONE	0
ANY	1
HMAC_MD5	2
HMAC_SHA1	3
HMAC_DES	4
KEYED_MD5	5
HMAC_RIPEM	6

GROUP

MODP (modular exponentiation group)	1
ECP (elliptic curve group over GF[P])	2
EC2N (elliptic curve group over GF[2^N])	3

values 4-65000 are reserved to IANA. Values 65001-65535 are for private use among mutually consenting parties.

LIFETIME_TYPE

This 2 octet field indicates type of lifetime.

seconds	1
kilobytes	2

values 3-65000 are reserved to IANA. Values 65001-65535 are for private use among mutually consenting parties.

LIFETIME

This 4 octet field indicates the SA lifetime. For a given "Lifetime_Type" the value of the "Lifetime" attribute defines the actual length of the SA life-- either a number of seconds, or a number of kilobytes protected.

LOC_TYPE

This 1 octet field indicates the contents of the LOC_SRC or LOC_DST field. If this field is 0 then the LOC_SRC or LOC_DST will be omitted.

0 NONE
1 IPv4 address
2 IPv6 address
3 DNS Name
4 Defaults

LOC_SRC

Variable length field depending on LOC_TYPE. IF LOC_TYPE is (04) then this field is 1 octet in length an it may only take the following fields:

1 ANY
2 DEST
3 HOST
4 LOCAL-SG
5 REMOTE-SG

LOC_DST

See LOC_SRC.

AH

NOT_REQUIRED	0
TUNNEL_MODE	1
TRANSP_MODE	2

IPCOMP_ALG

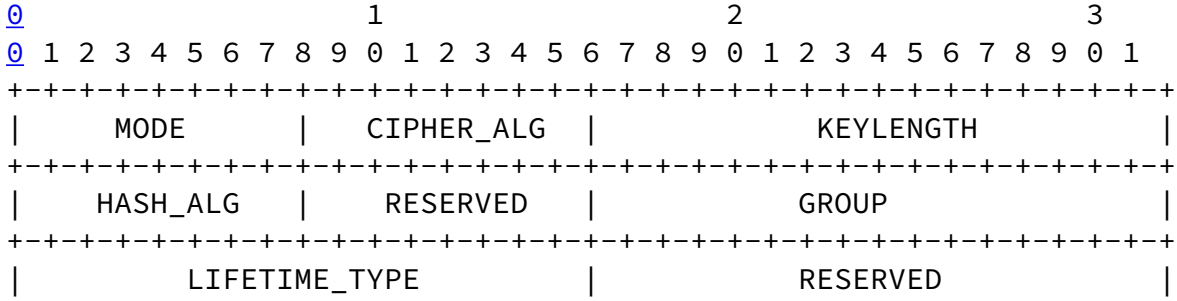
ANY	0
OUI	1
DEFLATE	2
LZS	3
V42BIS	4

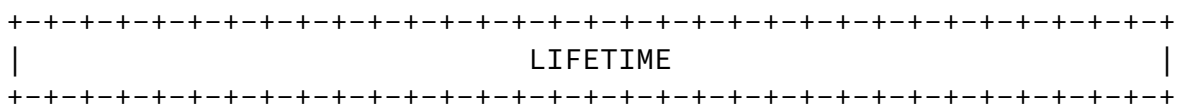
RESERVED

This 1 or 2 octet field (see paylaod formats above) is primarily used for padding purposes. Its value is always 0.

A.31 ISAKMP_ACTION

X	0
DATA_TYPE	51
LENGTH	13 BYTES
list	No
DATA_VALUE	





MODE

This octet indicates the IKE mode of operation.

MAIN	0
AGRESSIVE	1

CIPHER_ALG

This octet indicates which cipher should be used for the ISAKMP phase 1 negotiation.

ANY	0
DES	1
IDEA	2
BLOWFISH	3
RC5	4
DES3	5
CAST	6

KEYLENGTH

The first octet corresponds to the minimum value and the second octet corresponds to the maximum value. If no range exist the first octet indicates the keylength. The second octet contains a value of (00)hex.

HASH_ALG

This octet indicates which algorithm should be used for the ISAKMP phase 1 negotiation.

ANY	0
MD5	1
SHA1	2
TIGER	3

GROUP

This 2 octet field indicates which group should be used during the ISAKMP phase 1 or phase 2 negotiation.

MODP (modular exponentiation group)	1
ECP (elliptic curve group over GF[P])	2
EC2N (elliptic curve group over GF[2^N])	3

values 4-65000 are reserved to IANA. Values 65001-65535 are for private use among mutually consenting parties.

LIFETIME_TYPE

This 2 octet field indicates type of lifetime.

seconds	1
kilobytes	2

values 3-65000 are reserved to IANA. Values 65001-65535 are for private use among mutually consenting parties.

LIFETIME

This 4 octet field indicates the SA lifetime. For a given "Lifetime_Type" the value of the "Lifetime" attribute defines the actual length of the SA life-- either a number of seconds, or a number of kilobytes protected.

RESERVED

This 1 or 2 octet field (see paylaod formats above) is primarily used for padding purposes. Its value is always 0.

Note: for the variable-size fields (i.e., LOC_SRC and LOC_DST), add padding until the 4-octet boundary. Since the LEN of the LOC_SRC or LOC_DST fields is known the receiver can parse the field and then discard the rest of the bits through the 4-octet boundary.

APPENDIX B

Decorrelation Example.

This appendix demonstrates the decorrelation algorithm on a sample set of policies. This uses the optimizations presented.

The correlated policy set C:

	src	dst	prot	sport	dport	user	sec level
C1	199.93/16	199.100.2/24	TCP	*	22	lsanchez	sec
C2	199.93/16	199.100.2/24	TCP	*	*	lsanchez	conf
C3	199.93/16	199.100.2/24	UDP	*	*	lsanchez	*
C4	199.93/16	199.100.2/24	UDP	*	52	*	*
C5	199.93/16	199.100.2/24	*	*	*	*	*
C6	*	*	*	*	*	*	*

B.1 policy C1

	src	dst	prot	sport	dport	user	sec level
C1:	199.93/16	199.100.2/24	TCP	*	22	lsanchez	sec

By step 1, C1 becomes U1:

The current uncorrelated policy set:

U1	199.93/16	199.100.2/24	TCP	*	22	lsanchez	sec

B.2 policy C2

	src	dst	prot	sport	dport	user	sec level
C2:	199.93/16	199.100.2/24	TCP	*	*	lsanchez	conf

C2 is uncorrelated with U1 because the security levels do not overlap. By step 2, C2 is added to U.

The current uncorrelated policy set:

U1	199.93/16	199.100.2/24	TCP	*	22	lsanchez	sec
U2	199.93/16	199.100.2/24	TCP	*	*	lsanchez	conf

B.3 policy C3

```
     src         dst            prot   sport  dport     user     sec level
C3:  199.93/16   199.100.2/24   UDP      *      *      lsanchez      *
```

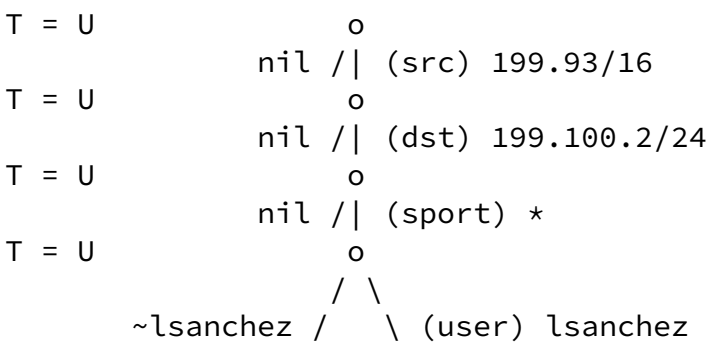
C3 is uncorrelated with U because it uses UDP while both policies in U use TCP. By step 2, C3 is added to U.

The current uncorrelated policy set:

```
U1   199.93/16   199.100.2/24   TCP      *      22     lsanchez     sec
U2   199.93/16   199.100.2/24   TCP      *      *      lsanchez     conf
U3   199.93/16   199.100.2/24   UDP      *      *      lsanchez      *
```


B.4 policy C4

```
     src         dst            prot   sport  dport     user     sec level
C4:  199.93/16   199.100.2/24   UDP      *      52        *          *
```



We can end the algorithm here, since

1) the ~lsanchez branch:

```
199.93/16  199.100.2/24  UDP      *      52    ~lsanchez    *
```

is uncorrelated with T since ~lsanchez does not overlap any policies in T.

2) The lsanchez branch:

```
199.93/16  199.100.2/24  UDP      *      52    lsanchez     *
```

is overriden by the policy U3.

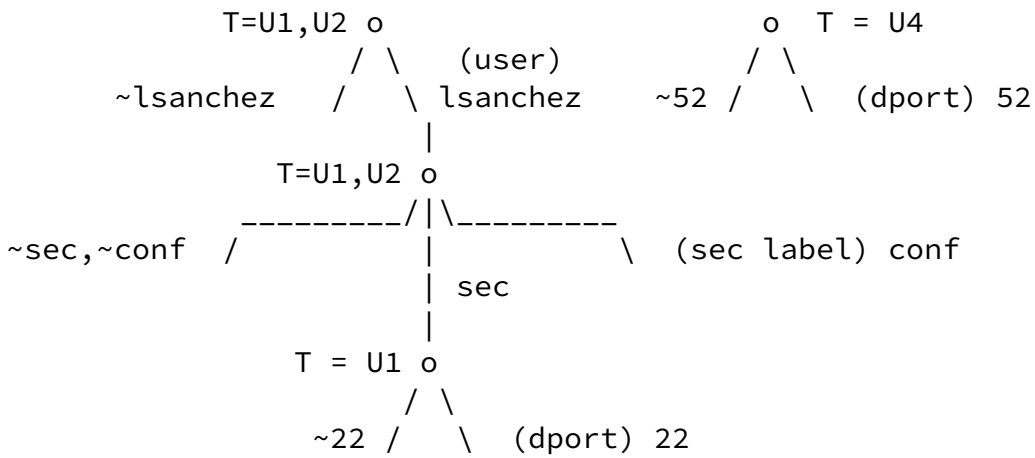
The current uncorrelated policy set:

```
U1  199.93/16  199.100.2/24  TCP      *      22    lsanchez    sec
U2  199.93/16  199.100.2/24  TCP      *      *     lsanchez    conf
U3  199.93/16  199.100.2/24  UDP      *      *     lsanchez     *
U4  199.93/16  199.100.2/24  UDP      *      52    ~lsanchez    *
```


B.5 policy C5

```
     src        dst           prot   sport  dport     user    sec level
C5:  199.93/16  199.100.2/24   *       *      *        *         *

       T = U              o
                     nil /| (src) 199.93/16
       T = U              o
                     nil /| (dst) 199.100.2/24
       T = U              o
                     nil /| (sport) *
       T = U              o
                  _______/|\_______
       ~UDP,~TCP /        |        \ (prot) UDP
                          |         o  T=U3,U4
                          | TCP     |\_________
                          |         |          \
                          |         | lsanchez  | (user) ~lsanchez
```



We can end the algorithm here since:

1) The branch:
199.93/16 199.100.2/24 ~UDP,~TCP * * * *
is uncorrelated with T = U.

2) The branch:
199.93/16 199.100.2/24 UDP * * lsanchez *
is overridden by U3.

3) The branch:
199.93/16 199.100.2/24 UDP * ~52 ~lsanchez *
is uncorrelated with T = U4.

4) The branch:
199.93/16 199.100.2/24 UDP * 52 ~lsanchez *
is overridden by U4.

5) The branch:
199.93/16 199.100.2/24 TCP * * ~lsanchez *
is uncorrelated with T = U1, U2.

6) The branch:
199.93/16 199.100.2/24 TCP * * lsanchez ~sec,~conf
is uncorrelated with T = U1, U2.

7) The branch:
199.93/16 199.100.2/24 TCP * * lsanchez conf
is overridden by U2.

```
8) The branch:
199.93/16  199.100.2/24  TCP      *      ~22   lsanchez    sec
is uncorrelated with T = U1.

9) The branch:
199.93/16  199.100.2/24  TCP      *      22    lsanchez    sec
is overridden by U1.

The current uncorrelated policy set:
U1  199.93/16  199.100.2/24  TCP      *      22    lsanchez    sec
U2  199.93/16  199.100.2/24  TCP      *      *     lsanchez    conf
U3  199.93/16  199.100.2/24  UDP      *      *     lsanchez     *
U4  199.93/16  199.100.2/24  UDP      *      52    ~lsanchez    *

U5  199.93/16  199.100.2/24 ~UDP,~TCP *      *       *          *
U6  199.93/16  199.100.2/24  UDP      *      ~52   ~lsanchez    *
U7  199.93/16  199.100.2/24  TCP      *      *     ~lsanchez    *
U8  199.93/16  199.100.2/24  TCP      *      *     lsanchez   ~sec,~conf
U9  199.93/16  199.100.2/24  TCP      *      ~22   lsanchez    sec
```

B.6 policy C6

```
     src        dst           prot   sport  dport     user    sec level
C6:    *            *         *       *      *        *         *

       T = U              o
                         /|
           ~199.93/16   / | (src) 199.93/16
       T = U              o
                         /|
        ~199.100.2/24   / | (dst) 199.100.2/24
```

```
We can end the algorithm here since:

1) The branch:
~199.93/16       *        *       *      *        *         *
is uncorrelated with all the policies in T.

2) The branch:
199.93/16  ~199.100.2/24  *       *      *        *         *
is uncorrelated with all the policies in T.

3) The branch:
199.93/16  199.100.2/24   *       *      *        *         *
is overridden by policy C5.
```

The uncorrelated version of C:

U1	199.93/16	199.100.2/24	TCP	*	22	lsanchez	sec
U2	199.93/16	199.100.2/24	TCP	*	*	lsanchez	conf
U3	199.93/16	199.100.2/24	UDP	*	*	lsanchez	*
U4	199.93/16	199.100.2/24	UDP	*	52	~lsanchez	*
U5	199.93/16	199.100.2/24	~UDP,~TCP	*	*	*	*
U6	199.93/16	199.100.2/24	UDP	*	~52	~lsanchez	*
U7	199.93/16	199.100.2/24	TCP	*	*	~lsanchez	*
U8	199.93/16	199.100.2/24	TCP	*	*	lsanchez	~sec,~conf
U9	199.93/16	199.100.2/24	TCP	*	~22	lsanchez	sec
U10	199.93/16	~199.100.2/24	*	*	*	*	*
U11	~199.93/16	*	*	*	*	*	*

Disclaimer

The views and specification here are those of the authors and are not necessarily those of their employers. The authors and their employers specifically disclaim responsibility for any problems arising from correct or incorrect implementation or use of this specification.

Author Information

Luis A. Sanchez
BBN Technologies
GTE Internetworking
10 Moulton Street
Cambridge, MA 02140
USA
Email: lsanchez@bbn.com
Telephone: +1 (617) 873-3351

Matthew N. Condell
BBN Technologies
GTE Internetworking
10 Moulton Street
Cambridge, MA 02140
USA
Email: mcondell@bbn.com
Telephone: +1 (617) 873-6203