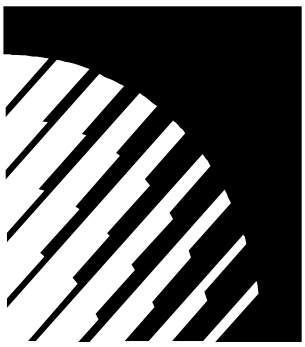


The ATM Forum

Technical Committee

ATM Security Specification
Version 1.0

AF-SEC-0100.001

February, 1999

The ATM Forum
Worldwide Headquarters
2570 West El Camino Real, Suite 304
Mountain View, CA 94040-1313
Tel: +1-650-949-6700
Fax: +1-650-949-6705

Acknowledgments

The production of this specification would not be possible without the enormous amount of effort provided by many individuals. Special acknowledgements for their hard work and dedication go to Mohammad Peyravian and Randy Mitchell, past chair and vice-chair of Security Working Group, and Richard Graveman and Wolfgang Klasen, the current chair and vice-chair.

In addition, the following individuals (listed alphabetically), among others, contributed their time and expertise to the development of this specification:

Asher Altman
Carter Bullard
Louis Finkelstein
Kim Hebda
Alexander Jurisic
Chris Kubic
Scott Lane
Hikaru Morita
Lyndon Pierson
Dan Schnackenberg
Linda Shields
Horace Thompson

Thomas D. Tarman
Brian Rosen
Editors, ATM Forum Security Working Group

Contents

1. Introduction

This specification defines procedures that provide a number of ATM security services. These services fall into three broad categories–security services for user plane virtual circuits, security services for control plane messages, and supporting services (management plane security services are not provided in this specification). User plane security services are performed on a per-virtual-circuit basis, where a "virtual circuit" can be either a virtual channel connection or a virtual path connection. Security support services include security message exchange and negotiation, which are performed at connection establishment via signaling, and/or within the user plane virtual circuit (after the connection is established, but before data transfer). Once the virtual circuit is established, further in-band messaging is provided by security OAM cells, as required by the negotiated cryptographic services.

This specification is organized as follows: Section 1 describes the scope and goals for this specification, provides definitions and references to supporting specifications, and describes compliance requirements. Section 2 provides a top-level reference model for the security services defined in this specification. Section 3 defines the user plane security services, including entity authentication, data confidentiality, data integrity (also known as "data origin authentication"), and label-based access control. In Section 4, the control plane authentication and integrity security services are defined. Security support services are described in Section 5. These services include security information exchange, negotiation, key exchange, session key update, and certification. Section 6 contains the normative annex, which includes detailed specifications for in-band security message exchange, primitives for the Security Services Information Element, specifications of various cryptographic algorithms and modes, and algorithm-specific coding details. Finally, Section 7 contains the informative appendix, which includes examples related to cryptographic resynchronization rate, label-based access control, and encodings of the Security Services Information Element.

1.1. Goals

This specification defines security services that support the following goals:

1. **Support multiple specification-defined algorithms and key lengths**.
   Since each organization has different security requirements, each site will likely want the opportunity to select from a variety of algorithms and/or protocols for their security designs. Furthermore, the import/export laws of some countries place restrictions on which encryption products may be imported/exported. For these reasons, the ATM security infrastructure and mechanisms specified here must support multiple algorithms and/or key lengths.

2. **Define a security infrastructure that provides interoperability among vendors who support one or more of the algorithms defined in this specification**.
   Since it is not feasible to define one default algorithm to meet all organizations' differing security requirements, this specification defines interoperability parameters for several well-known algorithms and common mechanisms for performing security functions using these algorithms. Devices that implement the algorithms and mechanisms as defined in this specification should interoperate with other devices with which they share a common suite of algorithms.

3. **Define a security infrastructure that provides for negotiation of private algorithms not specified in this specification**.
   Because selection of algorithms is such a sensitive issue, some organizations will want unique security algorithms tailored to their needs. The security infrastructure should provide standardized mechanisms for support of such algorithms, including means for agreement between two devices to use a private rather than a specification-defined algorithm.

4. **Maintain compatibility with devices that do not implement the security extensions**.
   Security messaging and secured communications must be transportable by intermediate network elements that do not understand or implement security.

5. **Minimize the impact on other specifications**.
   Security cannot be implemented without some modifications to existing specifications. These changes should be as minimal as possible without compromising the other goals of security. When possible mechanisms should be defined to support backward compatibility with existing specifications (e.g. UNI 3.1 and UNI 4.0.)

6. **Maintain compatibility across successive versions of the Security Specification**.
   Even though this specification is of limited scope, this specification should define extensible mechanisms to support future capabilities.

7. **Define mechanisms that will scale to a large (potentially global) number of users**.
   ATM security must be as scaleable as other ATM protocols in order to provide security in the broadest context.

8. **Define mechanisms that provide separability of authentication and integrity from confidentiality**.
   Some countries (e.g. the US) differentiate confidentiality from authentication and integrity in their export laws. The security mechanisms for ATM should be partitioned such that the functions of authentication and integrity may be implemented independently from confidentiality mechanisms.

1.2. References

1.2.1. Normative References

[1] ATM Forum Technical Committee, “B-ICI Specification, Version 2.0.”

[2] ATM Forum Technical Committee, “User-Network Interface (UNI) Specification,” Version 3.1, September 1994.

[3] ATM Forum Technical Committee, “User-Network Interface (UNI) Signalling Specification,” Version 4.0, April 1996.

[4] ATM Forum, “UNI Signaling 4.0 Security Addendum,”, ATM Forum BTD-CS-UNI-SEC-01.03, October 1998.

[5] ATM Forum, “PNNI Signaling 1.0 Security Addendum,” ATM Forum BTD-CS-PNNI-SEC-01.03, October 1998

[6] Federal Information Processing Standards Publication 46-2 (FIPS PUB 46-2), “Data Encryption Standard (DES),” December 1993.

[7] Federal Information Processing Standards Publication 81 (FIPS PUB 81), “DES Modes of Operation,” December 1980.

[8] Federal Information Processing Standards Publication 180-1 (FIPS PUB 108-1), “Secure Hash Standard,” April 1995.

[9] Federal Information Processing Standards Publication 186 (FIPS PUB 186), “Digital Signature Standard,” May 1994.

[10] Federal Information Processing Standards Publication 188 (FIPS PUB 188), “Standard Security Label for Information Transfer,” September 1994.

[11] IETF, “HMAC: Keyed-Hashing for Message Authentication,” RFC 2104, February 1997.

[12] IETF, “The MD5 Message Digest Algorithm,” RFC 1321, April 1992.

[13] IETF, “Privacy Enhancement for Internet Electronic Mail (PEM),” RFCs 1421-1424, February 1993.

[14] IETF, “U.S. Department of Defense Security Options for the Internet Protocol,” RFC 1108, November 1991.

[15] ISO/IEC 9594-8, 1995 (E), Information Technology -- Open Systems Interconnection -- The Directory: Authentication Framework.

[16] ISO/IEC 9594-8, 1995, Amendment 1: Certificate Extensions.

[17] ISO/IEC 9797, “Information Technology - Security Techniques - Data Integrity Mechanism using a Cryptographic Check Function Employing a Block Cipher Algorithm,” 1994.

[18] ISO/IEC 10118-3, “Hash Functions - Part 3: Dedicated Hash Functions,” 1997.

[19] ITU-T Recommendation I.363, “B-ISDN ATM Adaptation Layer (AAL) Specification,” March 1993.

[20] ITU-T Recommendation I.610, “B-ISDN Operation and Maintenance Principles and Functions,” November 1994.

[21] ITU-T Recommendation Q.2931, “B-ISDN DSS2 User-Network Interface Layer 3 Specification for Basic Call/Connection Control,” February 1995.

[22] ITU-T Recommendation Q.2971, “B-ISDN DSS2 User-Network Interface Layer 3 Specification for Point-to-Multipoint Call/Connection Control,” 1995.

[23] ITU-T Recommendation X.509, “The Directory: Authentication Framework,” 1993.

[24] Public Key Cryptography Standards #1 (PKCS #1), “RSA Encryption Standard,” RSA Laboratories, Version 1.5, November 1993.

[25] ANSI X9.17, “Financial Institution Key Management (Wholesale),” April 1985, p.23.

1.2.2. Informative References

[26] Agnew, Mullin, and Vanstone, “Improved Digital Signature Scheme Based on Discrete Exponentiation,” *Electronic Letters*, Vol. 26, pp. 1024-1025, 1990.

[27] Bird, *et al*., “The KryptoKnight Family of Light-Weight Protocols for Authentication and Key Distribution,” *IEEE/ACM Transactions on Networking*, Vol. 3, no. 1, pp. 31-41, February 1995.

[28] ETSI TCR-TR 028, “Network Aspects (NA); Security Techniques Advisory Group (STAG); Glossary of security terminology,” 1995.

[29] Fujioka, A., *et al.*, “ESIGN: An Efficient Digital Signature Implementation for Smart Cards,” *Proceedings of Eurocrypt ‘91*, Springer-Verlag, pp. 446-457, 1991.

[30] IEEE P1363, “Standard Specifications For Public Key Cryptography,” Draft, Version 1, December 1997.

[31] IETF, “Internet Security Association and Key Management Protocol (ISAKMP),” RFC 2408, November 1998.

[32] Miyaguchi, S., *et al.*, “Expansion of FEAL Cipher,” *NTT Review*, Vol. 2, no. 6, pp. 117-127, November 1990.

[33] Miyaguchi, S., “The FEAL Cipher Family,” *Advances in Cryptology - CRYPTO ‘90*, LNCS 537, pp. 627-638, Springer-Verlag, 1991.

[34] Okamoto, T., “A Fast Signature Scheme Based on Congruential Polynomial Operations,” *IEEE Transactions on Information Theory*, Vol. 36, no. 1, pp. 47-53, 1990.

[35] Schneier, B., *Applied Cryptography*, 2nd edition, John Wiley & Sons, 1996.

1.3. Definitions

Access Control - The application of a set of rules to a request for service to prevent the unauthorized use of the service.

Authentication - The process of corroborating that an entity in an instance of a communication is the one claimed.

Certification Authority - An entity trusted by one or more users to create and assign certificates. Optionally, the certification authority may also create the user’s keys. [28]

Ciphertext Interface – The interface on a security agent that transmits and receives traffic that is cryptographically protected by this security agent’s services.

Confidentiality - The protection of information (e.g., data) from unauthorized disclosure, even in the presence of active, malicious threats.

Cryptographic System - A collection of transformations from plaintext into ciphertext and vice versa, the particular transformation(s) to be used being selected by keys. The transformations are normally defined by a mathematical algorithm. [28]

Digital Signature - Data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery. [28]

Endpoint - The entity initiating a request for a security service (e.g., endsystem). See also “User.”

In-Band Message Exchange - The exchange of security information in the previously established user data connection. This connection could be either a VCC or a VPC, and could be established via SVC (signaling) or PVC procedures.

Initiator - The security agent that generates FLOW-1 of the security message exchange protocol (described in Section 5.1), or inserts label-based access control information (described in Section 3.4).

Integrity - The detection of unauthorized modifications to information, even in the presence of active, malicious modification threats.

Key - A value or representation used to cryptographically transform information.

Key Exchange - The process of communicating cryptographic key information between two or more entities.

Plaintext Interface – The interface on a security agent that transmits and receives traffic that is not cryptographically protected by this security agent's services.

Private Key - In asymmetric (public-key) cryptography, that portion of a user's key pair which is known only to that user.

Public Key - In asymmetric (public-key) cryptography, that portion of a user's key pair which is publicly known.

Replay Prevention - The process of validating that a message previously communicated is not repeated from another source impersonating as the original source.

Responder - The Initiator's peer in the security message exchange protocol, or the target security agent for label-based access control. For label-based access control, multiple target security agents are permitted.

Secret Key - In symmetric (secret-key) cryptography, a confidential key that is shared by all authorized users.

Security Agent (SA) - An entity that initiates, establishes, provides, discontinues, or terminates any of the ATM Forum Version 1.0 Security Services, such as Access Control, Authentication, Confidentiality and/or Data Integrity. A SA conforms to the top-level reference models, as defined in this specification, Section 2.

Security Association - The establishment of a secure relationship between two entities.

Security Negotiation - The process by which a secure environment is initiated, established, or denied between two entities.

Signaling-Based Message Exchange - The exchange of security information in the signaling channel. In this specification, this exchange is accomplished via procedures described in [3] and [4].

User - The entity initiating a request for a security service (e.g., endsystem). See also "Endpoint,"

1.4. Abbreviations

In addition to the abbreviations in the relevant ATM specifications (UNI 4.0, PNNI, etc.), the following abbreviations apply:

CBC - Cipher Block Chaining [7]. A mode of operation for block ciphers (e.g., DES and FEAL).

CBC-MAC - CBC Message Authentication Code [17]. A mechanism for providing message integrity and authenticity that uses a block cipher in CBC mode.

CRL - Certificate Revocation List.

DES - Data Encryption Standard [6]. A U.S. standard (published by NIST) for data encryption.

DES40 - DES with a forty-bit effective key.

DH - Diffie-Hellman. A key agreement algorithm.

DSA - Digital Signature Algorithm [9]. The algorithm specified by the DSS.

DSS - Digital Signature Standard [9]. A U.S. standard (published by NIST) for digital signatures.

ECB - Electronic CodeBook [7]. A mode of operation for block ciphers (e.g., DES and FEAL).

ECC and **EC** - Elliptic Curve Cryptosystem (see Section 6.9). A public-key cryptosystem.

ECKAS-DH - Elliptic Curve Key Agreement Scheme - Diffie-Hellman. The Diffie-Hellman key agreement scheme using elliptic curve cryptography.

ECDSA-like - Elliptic Curve Digital Signature Algorithm.

ESIGN - Efficient digital SIGNature scheme [29]. A digital signature algorithm.

FEAL - Fast Data Encipherment Algorithm [32] and [33]. An encryption algorithm.

HMAC - Hashed Message Authentication Code [11].

H-MD5 - HMAC using the MD5 hash algorithm.

H-SHA and **H-SHA-1** - HMAC using the SHA-1 hash algorithm.

LIJ - Leaf Initiated Join. A method of adding leaves to a point-to-multipoint ATM call upon request of a leaf.

MAC - Message Authentication Code.

MD5 - Message Digest 5 [12]. A hash algorithm that is typically used when generating digital signatures.

NIST - (U.S.) National Institute of Standards and Technology.

PTI - Payload Type Indicator. A field in the ATM cell header that is used to identify the type of the cell payload contents.

PVC - Permanent Virtual Circuit. An ATM virtual circuit established by management operations.

RSA - Rivest, Shamir, and Adleman [[24]. The inventors of this encryption/digital signature algorithm.

SHA-1 - Secure Hash Algorithm (Revision 1) [8]. The hash algorithm specified by the DSS.

SA - Security Agent. A logical entity that implements ATM security functions. See definition for “Security Agent” in Section 1.3.

SAS - Security Association Section.

SSCOP - Service Specific Connection Oriented Protocol. The connection-oriented protocol that is used by UNI signaling for reliable delivery of signaling messages.

SSIE - Security Services Information Element.

SVC - Switched Virtual Circuit. An ATM virtual circuit established by ATM signaling.

UNI - User to Network Interface.

VC - Virtual Circuit

VCC - Virtual Channel Connection

VCI - Virtual Channel Identifier. A field in the ATM cell header.

VPC - Virtual Path Connection

VPI - Virtual Path Identifier. A field in the ATM cell header.

1.5. Specification Scope

The scope of this security specification is indicated in Figure 1. The shaded portions of the matrix show what is within scope for this specification. The structure of Figure 1 reflects the ATM Reference Model, which defines three planes—the user plane, the control plane, and the management plane—with each plane comprising three or more protocol layers—the physical layer, the ATM layer, the ATM Adaptation Layer (AAL), and upper layers as required. The user plane provides transfer of user data across ATM Virtual Channel Connections (VCCs) and Virtual Path Connections (VPCs). The control plane deals with connection establishment, release, and other connection functions, including UNI, NNI, and ICI signaling. The management plane performs management and coordination functions related to both the user and the control planes (including the PNNI functions related to the establishment of a routing infrastructure).

As indicated in Figure 1, this document specifies mechanisms for authentication, confidentiality, data integrity, and access control for the user plane. It also specifies mechanisms for authentication and integrity for the control plane (UNI and NNI signaling). Excluded from the scope of this specification is management plane security; however, to the extent that management plane entities use user plane connections to achieve their ends, the user plane security specified herein may contribute to management plane security. Also within scope is the infrastructure needed to support these security services: negotiation of security services and parameters, key exchange, and certification infrastructure.

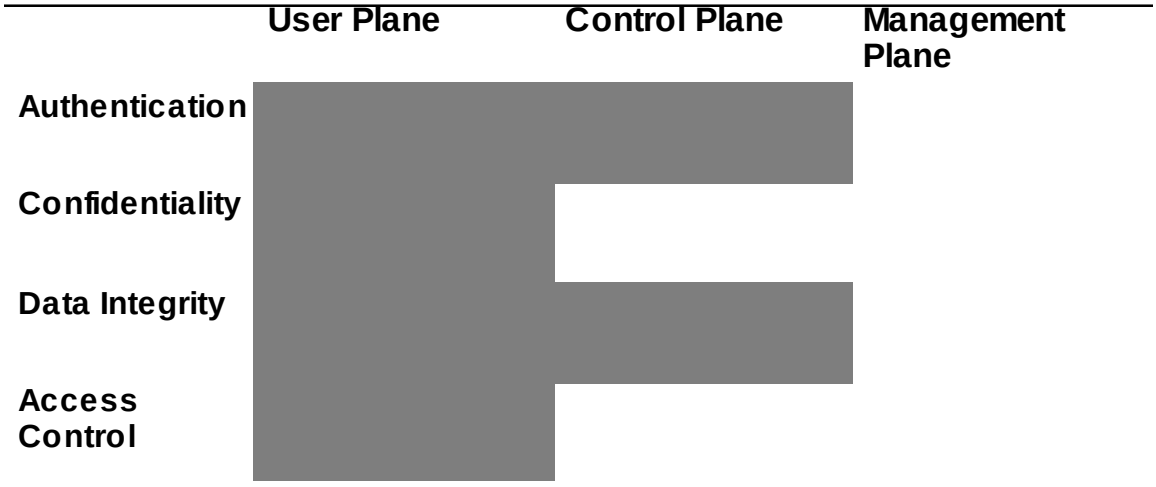


Figure 1. Scope of Security 1.0.

The scope is restricted to *ATM security*: mechanisms that must be implemented in the ATM layer and/or the AAL. The emphasis throughout is on providing *per-connection* security rather than, for example, link or node security. The scope includes all types of ATM connections: channel and path; point-to-point and point-to-multipoint; switched and permanent.

This document provides definitions for specific security mechanisms and protocols to support the development of interoperable ATM security services. These security services are designed to be used in conjunction with appropriate system and network security engineering practices for the enforcement of a security policy.

As stated above, this document specifies security services for the user plane (data connections) and control plane (signaling flows). In addition, the required services that support user plane and control plane security are also specified. The security services that are in the scope of this specification are described in more detail below.

1.5.1. User Plane Security Services

The user plane security services apply on a per VC (virtual circuit) basis, where a VC could be either a VCC (virtual channel connection) or a VPC (virtual path connection). Security services for physical links (which may carry many VCs) are *not* provided in this specification. The following security services for the user plane are defined: authentication, data confidentiality, data integrity, and access control. These services are supported in point-to-point and point-to-multipoint connections for both SVCs and PVCs.

1.5.1.1. Authentication

User plane authentication (also described as "entity authentication") determines at the beginning of the connection that the identities of the calling and/or called parties are genuine. Since this service provides protection against impersonation or "spoofing" threats, it is essential for establishing secure connections. For this reason, authentication is essential for the operation of other security services, including key exchange (described below), and the secure exchange of security negotiation parameters.

Authentication can be either mutual or unilateral. If authentication is mutual, then both parties are authenticated to each other, whereas with unilateral authentication, only one party is authenticated to the other.

Authentication is specified in this document to use cryptographic algorithms, including asymmetric (public key) algorithms (e.g., RSA) and symmetric (secret key) algorithms (e.g., DES-MAC). These classes of algorithms are described in more detail in [35].

1.5.1.2. Confidentiality

User plane confidentiality provides cryptographic mechanisms that protect "user" data on a VC from unauthorized disclosure. (The term "user" refers to the protocol entity that directly uses ATM services.) This specification defines ATM confidentiality at the cell level, rather than the AAL level, because the fixed-length of the ATM cell allows for efficient encryption. Furthermore, only the payload of the cell is encrypted–the header is sent in the clear. This allows the encrypted cell to be switched by the network without decryption at each hop.

The confidentiality service is specified in this document to use symmetric (secret key) algorithms. This is largely due to the fact that these algorithms are typically faster than asymmetric algorithms, which makes them more suited to ATM data encryption.

1.5.1.3. Integrity

The data integrity service (also described as "data origin authentication") provides a mechanism that allows for detection of modification to data values or sequences of data values, even in the presence of malicious modification threats. This service is provided between endpoints at the AAL Service Data Unit (SDU) level for AAL 3/4 and AAL 5. In addition, two options are provided for this service: 1) data integrity without replay/reordering protection, and 2) data integrity with replay/reordering protection.

When data integrity is provided without replay/reordering protection, the source appends a cryptographic signature to the tail of each AAL SDU before transmission. This signature is calculated over the entire AAL SDU. This option is useful to higher layer protocols that provide their own sequence numbers (e.g., TCP), without the added overhead required to duplicate this function at the AAL.

When the data integrity is provided with the replay/reordering protection option, the data integrity service provides protection against replay and reordering attacks, so that “old” or “reordered” AAL-SDUs are detected and may be discarded. At the source, this is achieved by first appending a sequence number to the tail of each AAL-SDU and then computing a signature on the totality of the AAL-SDU including the sequence number. This signature, which protects both the AAL-SDU and the sequence number, is then appended to the total AAL-SDU (which includes the sequence number). This method provides protection for ATM applications that do not implement their own sequence numbers.

As with the confidentiality service, the integrity service is specified in this document to use symmetric (secret key) algorithms.

1.5.1.4. Access Control

Access Control is the application of a set of rules to a request for a service. These rules may depend upon attributes of the invoking entity, such as identity, attributes of referenced parameters, such as a target address, system attributes, such as time, and the history of prior requests by this and/or other client entities. Access control rules can be thought of as a predicate over the state space formed by all such attributes. If the predicate is satisfied, the requested service is performed, if the predicate is not satisfied, the requested service is not performed.

User plane access control requires mechanisms to transport access control information used during connection establishment, as well as mechanisms within ATM components to use that information to determine whether access to the connection should be granted. User plane access control can be based on security labels (e.g., Standard Security Label [10]), source or destination user identities, time of day, type of service, upper-layer protocol (e.g., Internet Protocol) fields, or other parameters that can be determined during connection establishment.

User plane access control is defined for each ATM interface: end-point-to-switch and switch-to-switch. In each case, user plane access control is provided at the ATM layer.

1.5.2. Control Plane Security Services

The control plane is the mechanism that allows devices to configure the network to achieve some objective (for example, to establish a switched virtual circuit). Since control plane messages can affect the state and availability of a network, their protection is extremely important.

In this Security Specification, a mechanism for signaling is defined that can provide strong cryptographic data integrity with replay/reordering protection. This mechanism allows ATM control plane entities to verify the source and contents of a signaling message before resources are allocated to the request. This protects the network from a number of attacks, as described below.

Other security mechanisms for the control plane (e.g. control plane confidentiality) will be examined in later versions of the Security Specification.

1.5.2.1. Authentication and Integrity

Control plane authentication and integrity is the ATM security service that binds an ATM signaling message to its source. By creating this binding, the message recipient can confidently verify that the message originated from its claimed source. This provides a mechanism that mitigates a number of threats. For example, a denial of service attack, which attempts to tear down an active connection by surreptitiously injecting RELEASE or DROP PARTY messages, can be prevented when authenticity assurances are in place for the signaling channel. This service also protects against spoofing and malicious modification threats. In this specification, a mechanism for control plane authentication and integrity between adjacent

signaling entities is defined. The mechanism used is identical to the mechanism used for user plane data integrity with replay/reordering protection.

1.5.3. Support Services

This specification also defines a set of "support services," which are needed to provide scaleable and high-performance security services:

- Security message exchange and negotiation of security options,
- Key exchange,
- Key update,
- Certification infrastructure.

1.5.3.1. Security Message Exchange and Negotiation

In order to perform many of the security services described above, messages must be exchanged between the involved security agents (SAs). This specification describes two methods for security message exchange –message exchange within UNI 4.0 signaling and in-band message exchange (that is, security message exchange within the relevant user plane virtual circuit).

These message exchange methods also provide a mechanism for negotiation of security options. Since security requirements vary among organizations, it is important to provide a variety of security services, algorithms, and key lengths, which meet a wide range of security needs. In addition, the export and/or import laws of some countries place restrictions on which encryption products may be exported/imported. For these reasons, the ATM security mechanisms support multiple security services, algorithms, and key lengths. In order for security agents to agree on common security parameters (such as algorithms and key lengths), these security message exchange methods provide for negotiation of these parameters as part of the security establishment procedure for the VC.

1.5.3.2. Key Exchange

Key exchange is the mechanism by which two security agents exchange secret keys for use by the confidentiality and/or integrity services. In order to protect against "man in the middle"-type attacks, key exchange is often coupled with the authentication service. This can be accomplished by including "confidential" key exchange parameters within the authentication flows.

Like authentication, key exchange is performed with either symmetric (secret key) or asymmetric (public key) algorithms. In addition, key exchange may either be bi-directional (two-way), or uni-directional (one-way).

1.5.3.3. Session Key Update

Session keys are the keys used directly to provide user plane confidentiality and integrity services over an ATM virtual circuit. Due to the potentially high data rate of the virtual circuit, it is imperative that keys be periodically changed to avoid "key stream re-use." This specification defines a session key update service that provides this capability.

This service is performed in two phases–a session key exchange phase and a session key changeover phase. The session key exchange phase uses a "master key," which is exchanged at connection setup (using the key exchange service), to encrypt the new session key. Upon receipt of the encrypted session key, the recipient decrypts the session key using the shared master key, and stores it for the second phase–key changeover.

1.5.3.4. Certification Infrastructure

In a public key cryptosystem, each party (security agent) *X* has a pair of keys: one of which is publicly known, that is *X*'s "public key" (PK_X), and the other known only by *X*, that is *X*'s "private key" (SK_X). In order for party *A* to send secret information to party *B* (or, alternatively, in order for *A* to be able to verify a signature produced by *B*), *A* needs to obtain *B*'s public key, PK_B. Though PK_B is public by definition, it should not be possible for any party *X* to substitute another value (e.g., PK_X) for PK_B. To prevent this kind of attack, public keys can be exchanged in the form of "certificates."

A certificate contains the party's name, its public key, and some additional information and is signed by a trusted party, a "certification authority" (CA). This signature unforgeably binds the public key to the subject party. Any party with access to the CA's public key can verify the genuineness of the certificate (by checking the CA's signature in the certificate) and recover the public key that was certified. Once signed, certificates can be transmitted via non-secure message exchanges.

1.6. Compliance

Compliance for an ATM security implementation is defined on a per security service basis, for each of the five services specified in this document:

1. User Plane Authentication (AUTH)
2. User Plane Confidentiality (CONF)
3. User Plane Data Origin Authentication and Integrity (INTEG)
4. User Plane Access Control (ACC)
5. Control Plane Authentication and Integrity (CP-AUTH)

To claim compliance with this specification, an implementation must support at least one security service (that is, user plane authentication, user plane confidentiality, user plane integrity, access control, or control plane authentication). For the user plane authentication, confidentiality, and integrity services, the implementation must support:

1) One or more of the SME profiles listed in Section 0.

And,

2) At least one of the following:
   a) One or more algorithm profiles listed in Section 1.6.1,
   b) One or more user-defined algorithms.

For user plane access control, the implementation must support the access control profile in Table 4. For control plane authentication, the implementation must support one of the control plane authentication profiles in Table 5.

The compliance statement for the implementation must specify the security services and associated algorithms and security messaging profiles supported. In addition, the compliance statement must specify all additional algorithms implemented for the five security services areas. If those additional algorithms are defined in this specification they must be implemented as specified herein.

1.6.1. Security Algorithm Profiles

The following tables list the algorithms and modes (where applicable) required for each ATM security service. Compliance requirements for each service are specified in their respective sections, and compliance requirements for each algorithm are specified in the normative reference for that algorithm

specified in the respective service sections. These profiles represent a subset of all possible algorithm combinations, and some algorithms defined in this specification do not appear in the algorithm profiles.

Table 1: Authentication Algorithm Profiles.

Profile	Signature	Hash	Notes
AUTH-1	DES/CBC	n/a	1,2
AUTH-2	DSA	SHA-1	1
AUTH-3	ECDSA-like	SHA-1	1
AUTH-4	ESIGN	MD5	1
AUTH-5	FEAL/CBC	n/a	1,2
AUTH-6	RSA	MD5	1

Note 1: Support for the Security Message Exchange (SME) is also required.
Note 2: No hash function is used with this profile.
n/a : not applicable

Table 2: Confidentiality Algorithm Profiles.

Profile	Encryption	Mode	Signature	Key Exchange	Hash	Key Update	Notes
CONF-1	DES	CBC	RSA	RSA	MD5	MD5	1
CONF-2	DES	CBC	DSA	DH	SHA-1	SHA-1	1
CONF-3	DES	CBC	ECDSA-like	ECKAS-DH	SHA-1	SHA-1	1
CONF-4	DES	CBC	DES/CBC	DES/CBC	n/a	MD5	1
CONF-5	DES	Counter	RSA	RSA	MD5	MD5	1
CONF-6	DES	Counter	DSA	DH	SHA-1	SHA-1	1
CONF-7	DES	Counter	ECDSA-like	ECKAS-DH	SHA-1	SHA-1	1
CONF-8	DES	Counter	DES/CBC	DES/CBC	n/a	MD5	1
CONF-9	Triple-DES	CBC	RSA	RSA	MD5	MD5	1
CONF-10	Triple-DES	CBC	DSA	DH	SHA-1	SHA-1	1
CONF-11	Triple-DES	CBC	ECDSA-like	ECKAS-DH	SHA-1	SHA-1	1
CONF-12	Triple-DES	CBC	Triple DES/CBC	Triple DES/CBC	n/a	MD5	1
CONF-13	Triple-DES	Counter	RSA	RSA	MD5	MD5	1
CONF-14	Triple-DES	Counter	DSA	DH	SHA-1	SHA-1	1
CONF-15	Triple-DES	Counter	EC/DSA	EC/DH	SHA-1	SHA-1	1
CONF-16	Triple-DES	Counter	Triple DES/CBC	Triple DES/CBC	n/a	MD5	1
CONF-17	FEAL	CBC	ESIGN	DH	MD5	MD5	1
CONF-18	FEAL	CBC	FEAL/CBC	FEAL/CBC	n/a	MD5	1
CONF-19	FEAL	Counter	ESIGN	DH	MD5	MD5	1
CONF-20	FEAL	Counter	FEAL/CBC	FEAL/CBC	n/a	MD5	1

Note 1: Support for the Security Message Exchange is also required.
n/a : not applicable

Table 3: Integrity Algorithm Profiles.

Profile	MAC	Signature	Key Exchange	Hash	Key Update	Notes
INTEG-1	DES/CBC	RSA	RSA	MD5	MD5	1,2
INTEG-2	DES/CBC	DSA	DH	SHA-1	SHA-1	1,2
INTEG-3	DES/CBC	EC/DSA	EC/DH	SHA-1	SHA-1	1,2
INTEG-4	DES/CBC	DES/CBC	DES/CBC	n/a	MD5	1,2
INTEG-5	H-MD5	RSA	RSA	MD5	MD5	1,2
INTEG-6	H-MD5	DES/CBC	DES/CBC	n/a	MD5	1,2
INTEG-7	H-SHA-1	DSA	DH	SHA-1	SHA-1	1,2
INTEG-8	H-SHA-1	EC/DSA	EC/DH	SHA-1	SHA-1	1,2
INTEG-9	H-SHA-1	DES/CBC	DES/CBC	n/a	SHA-1	1,2
INTEG-10	FEAL/CBC	ESIGN	DH	MD5	MD5	1,2
INTEG-11	FEAL/CBC	FEAL/CBC	FEAL/CBC	n/a	MD5	1,2

Note 1: Support for the Security Message Exchange is also required.
Note 2: Support for the two versions of integrity, i.e., with and without replay protection, is required. The selection is negotiated at call setup time.
n/a : not applicable

Table 4: Access Control Label Profiles.

Profile	Label Formats	Notes
ACC-1	Standard Security Labels	1

Note 1: Because access control labels are processed independently at each security agent that performs access control, support for the Security Message Exchange is not required.

Table 5: Control Plane Authentication Algorithm Profiles.

Profile	MAC	Key Update	Notes
CP-AUTH-1	H-MD5	MD5	1
CP-AUTH-2	H-SHA-1	SHA-1	1
CP-AUTH-3	DES/CBC	SHA-1	1
CP-AUTH-4	FEAL/CBC	MD5	1

Note 1: Because control plane authentication messages are processed independently at each security agent that performs this service, support for the Security Message Exchange is not required.Security Message Exchange Profiles

Three security message exchange profiles are specified for ATM security messaging:

1. In-band security messaging (SME-1),
2. Signaling-based security two-way messaging, with a fall-back to in-band security messaging (SME-2),
3. Signaling-based security two-way messaging for the pt-mpt ADD PARTY message (SME-3).

Table 6 specifies the sections required for compliance with each profile. The implementation shall be compliant with the mechanisms specified in the referenced mechanism section. Note that the initial connection in a pt-mpt call is accomplished using SME-2. SME-3 is defined only for subsequent ADD PARTY messages.

Table 6: ATM Security Message Exchange Profiles.

Profile	ATM Bearer Service	Security Message Service	Message Exchange Protocol	Mechanisms	Notes
SME-1	Pt-pt PVCs Pt-pt SVCs	In-band	5.1.1.1	5.1.5.2 5.1.5.3 6.1	
SME-2	Pt-pt SVCs	Signaling-Based 2- Way, with In-band Fall-Back	5.1.1.2 (for Signaling-Based) 5.1.1.1 (for In-band)	5.1.4.1 5.1.4.4 5.1.5.3 6.1	1
SME-3	Pt-mpt SVCs (ADD PARTY)	Signaling-Based 2- Way	5.1.1.2	5.1.4.2 5.1.4.4	2

Note 1: "In-band Fall-Back" is the ability of the security agent to perform additional messaging using the in-band SME (i.e., SME-1).

Note 2: Support for SME-3 requires SME-2 for the initial connection establishment.

2. Top-Level Reference Models

This informative section presents several reference models that help clarify and define the intended scope and functionality of this specification. It also presents a high-level overview of the security services and mechanisms.

Note that the Access Control security service is different from the other security services–it is not negotiated, and many of the other concepts associated with the other services do not apply to this service. Consequently, this section concentrates on the other three security services; the reader is referred to Section 3.4 for reference models for the access control service.

2.1. Security Associations and ATM Interfaces

A *security agent* negotiates and provides the security services for a VC. Figure 2 shows the simplest case, in which security agents are collocated with ATM end systems. In this case, the end systems interface to an ATM network via a UNI. For each VC that is established by an end system, the Security Agent (SA) at that end system:

- Determines which security services, if any, need to be applied to that VC,
- Negotiates the service or services with the peer SA,
- Authenticates and exchanges keys, if necessary,
- Applies the required security service(s) to the VC.

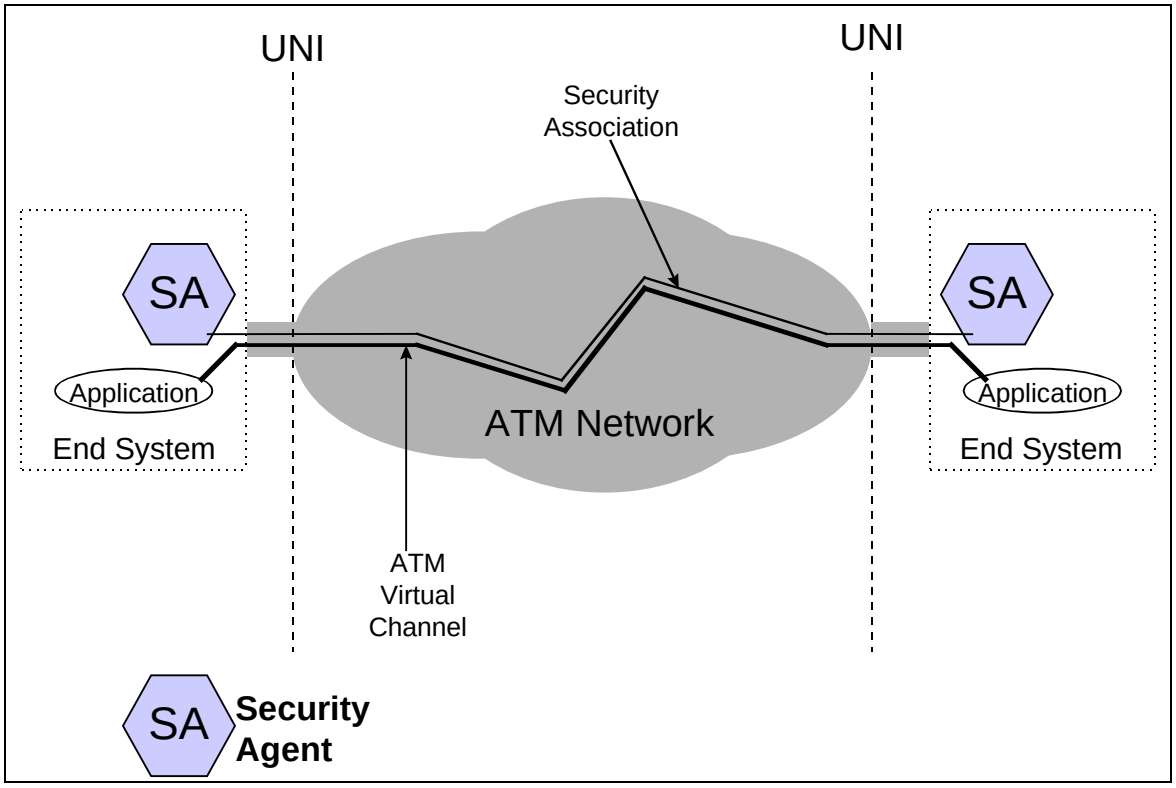


Figure 2. Security Association between Security Agents at End Systems.

The determination of which security services are needed for which VCs is a matter of *security policy*, which is beyond the scope of this document. Also, the existence and nature of any interaction between the end system application and its local security agent is beyond the scope of this document.

The negotiation of services with a peer security agent takes place via Security Information Exchange (SIE), an overview of which Section 2.3 presents. Via negotiation, the peer SAs establish one *Security Association* (or *Security Context*) for each service to be provided. Thus, if authentication, confidentiality, and data integrity are all required for a given VC, there are three Security Associations between the peers, one for each service. A Security Association is the distributed contextual information (e.g., cryptographic algorithm, mode of operation, optional features enabled, etc.) controlling the nature of the security service to be provided for a given VC.

If required, authentication and initial key exchange also occur during service negotiation. Key exchange is required for support of the confidentiality and data integrity services. The negotiation supports the exchange of connection master keys (keys used to generate future connection "session" keys if needed) and initial connection session keys for encryption and/or cryptographic checksums.

The application of the services to the VC varies according to the service. For authentication, the service is effected during negotiation. For confidentiality, encryption is applied to ATM data cell payloads, with maintenance of cryptographic state and call session key updates provided (optionally) via OAM flows. For data integrity, a cryptographic checksum and a sequence number (optional) are appended to AAL5 service data units (SDUs).

Note: The end-system-to-end-system model in Figure 2 is the only one that applies for the data integrity security service. This is because data integrity is applied at the AAL.

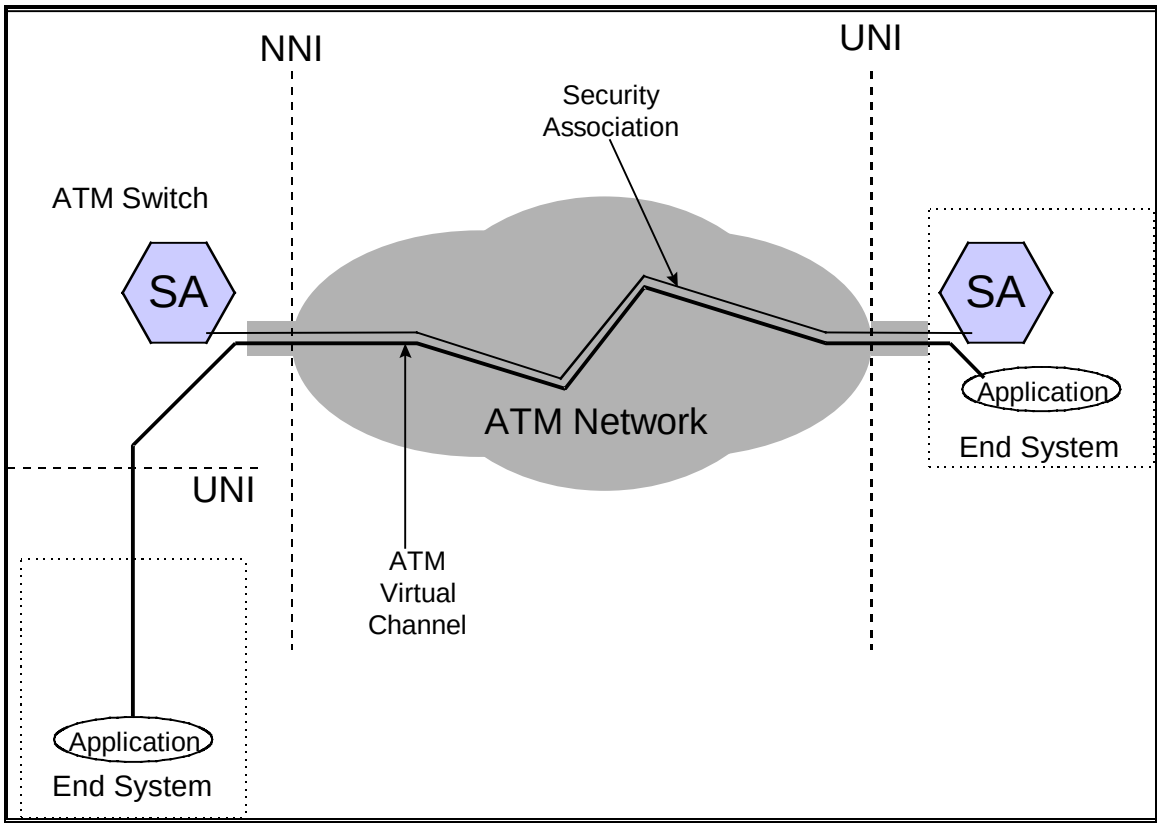


Figure 3. Security Association between a SA at a Switch and a SA at an End System.

Figure 3 shows another configuration of SAs. In this figure, one SA is located within an ATM end system and one is located within an ATM switch. The overall operation is the same as that reflected in Figure 2: determination, negotiation, and application of security services via Security Associations between peer Security Agents. However, Figure 3 highlights some possibilities–and some interactions–which might not be immediately obvious from Figure 2. Figure 3 implies the possibility of "firewall" Security Agents–SAs that provide security services for one or more networks "behind" the firewall. It also implies a close coupling of signaling entities and the security agent. Finally, it implies that switches that come equipped with Security Agents will be required to perform functions not normally associated with switches–for example, encryption of data cell payloads.

The security agent within the switch could maintain an identity distinct from those of any end systems, or it could serve as a *proxy* for one or more end systems. In the former case, the remote peer SA would explicitly authenticate (assuming authentication is required) the SA at the switch (i.e., the SA identifier would not imply collocation with the end system); in the latter case, the remote peer SA would authenticate a virtual SA at the end system. With proxies, the identities and accompanying authentication information (e.g., public and private keys) are kept and vested with the proxy agents rather than distributed among the end systems.

Note that a security service is only applied to that portion of a VC between the peer SAs providing the service. So, as pictured in Figure 3, there is no security applied to the VC between the left-hand end system and the switch containing the security agent. Whether or not this is acceptable, is, once again, a matter of security policy. However, as the next section shows, it is possible to provide different levels of security for different portions of a VC.

2.2. Multiple Security Associations and Nesting

Figure 4 shows that more than one pair of security agents–and more than one security association–might be involved in providing security services for a given VC. In this figure, SA_1 and SA_4 have a security association in which they provide (say) end-to-end authentication. Independent of this association, SA_2 and SA_3 have an association in which they provide confidentiality. These two associations are independent in terms of the services they provide–SA_1 and SA_4 neither know nor care what security services other agents might be providing. This independence allows multiple (and different) security policies to be in force simultaneously for different parts of the network.

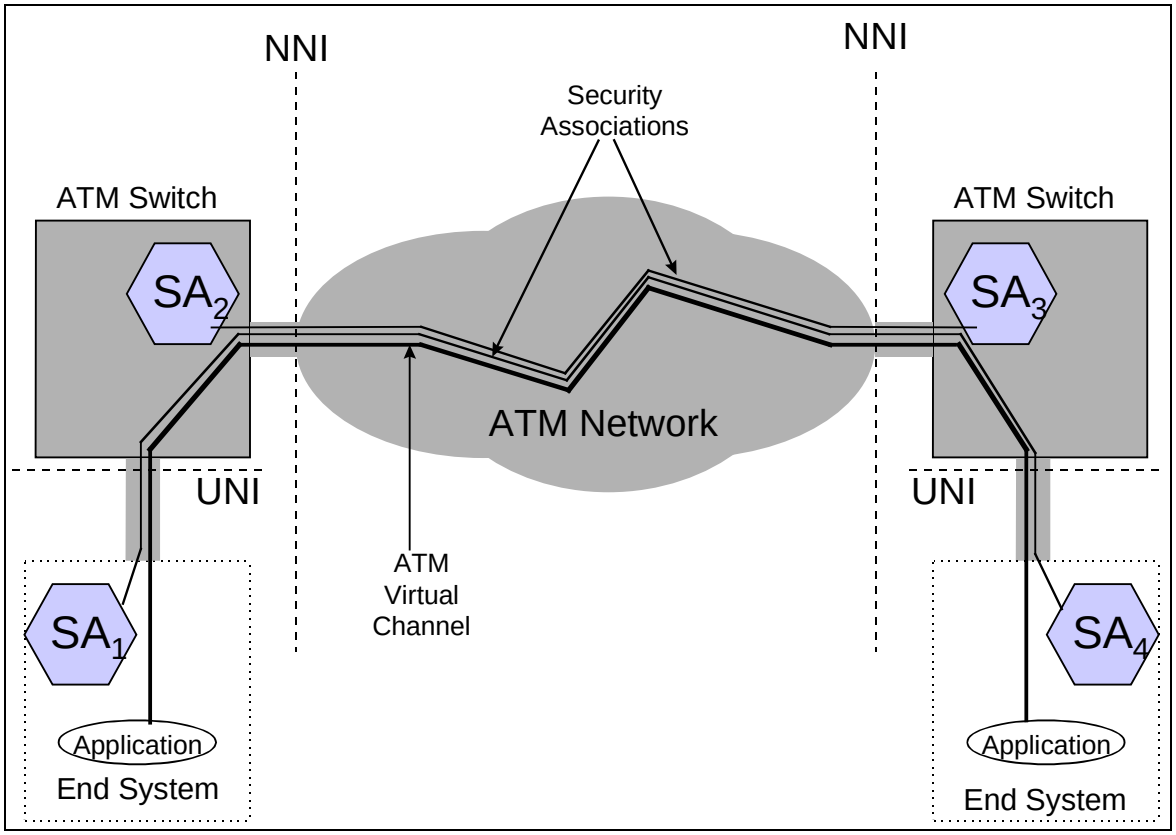


Figure 4. Two Nested Security Associations.

There are some dependencies and restrictions, however, on what might be called the *topology* of security associations; these dependencies result from the mechanisms chosen to negotiate security services and form security associations. By *topology* is meant the way different security associations are allowed to overlap along the path of a VC. Figure 4 shows one type of overlap that is permitted–nesting. We could say that the VC "segment" [SA_2, SA_3] is *contained in* the [SA_1,SA_4] segment, or, in symbols, [SA_2, SA_3] $\subset$ [SA_1,SA_4]. Another permissible relationship among associations is *no overlap*. If, instead of what is pictured, the associations were between SA_1 and SA_2, and SA_3 and SA_4, the segments [SA_1, SA_2] and [SA_3,SA_4] would have no overlap and the associations would be permissible. We could write this [SA_1, SA_2] $\frown$ [SA_3,SA_4]. Similarly, if the associations were between SA_1 and SA_3, and SA_3 and SA_4, the segments [SA_1, SA_3] and [SA_3,SA_4] would have no overlap and the associations would be permissible (this could with justification be called one-point overlap–but it's a permissible arrangement and for simplicity we'll refer to this as "no overlap" also). *Nesting* and *no overlap* are the only permissible arrangements. The mechanisms will not support simultaneous associations between SA_1 and SA_3, and SA_2 and SA_4, for example.

In addition to these topological restrictions, there is also a limit of 16 on the *nesting* level–the total amount of nesting at any given point along the VC's path. Note that this is not a limit *per se* on the total number of security associations for any given VC.

Figure 5 shows one example of the types of topologies that the security mechanisms support. The maximum nesting level in this example is 3. It would not be possible in the situation pictured in this figure to support another security association involving (say) SA_3 and SA_8, as this would violate the topological restrictions.

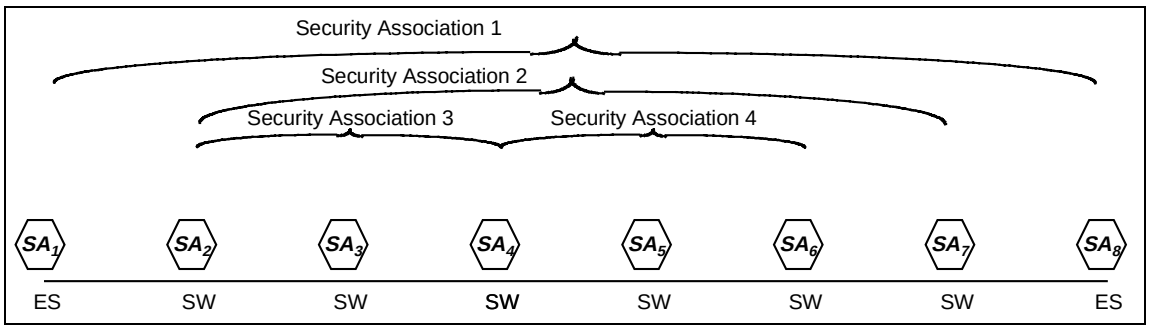


Figure 5. Multiple Associations with Nesting.

2.3. Security Information Exchange

Figure 6 shows the three types of information exchange that might be involved in establishing and maintaining security associations. The Security Message Exchange (SME) protocol is used to negotiate security services, establish security associations, authenticate the peer security agents (if required), and establish connection master keys and initial session keys. The SME can operate over the signaling channel (for networks that support the UNI Signaling 4.0 Security Addendum [4] or PNNI Signaling 1.0 Security Addendum [5]) or over the data channel that is pre-established or in the process of being established. The former is referred to as "Signaling-Base SME" and the latter as "In-band SME."

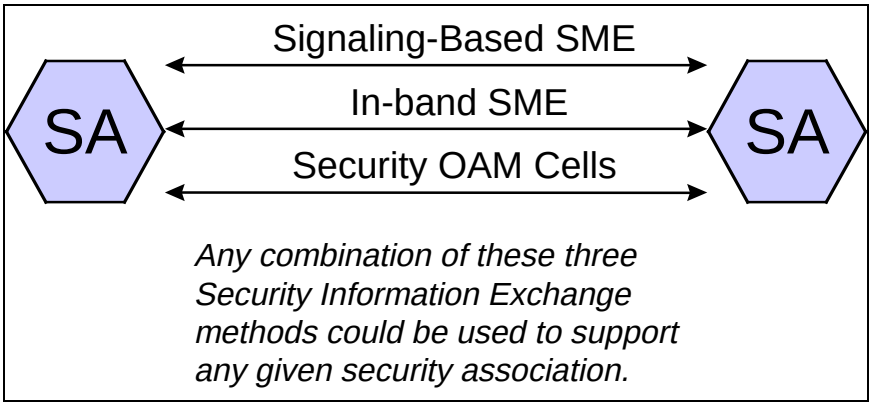


Figure 6. Security Information Exchange Methods.

As stated earlier, signaling-based SME is used in networks that support the UNI Signaling 4.0 Security Addendum [4] or the PNNI Signaling 1.0 Security Addendum [5]. Support for B-ICI and other signaling protocols is not currently defined. An example of SME usage in a network that supports the Security

Addenda is shown in Figure 7 (where “UNI 4.0 + sec” and “PNNI 1.0 + sec” denote links that implement the UNI and PNNI security addenda). In this case, the end systems (with Security Agents) append the SSIE to the signaling message, and as the network establishes the call, the SSIE is passed without modification.

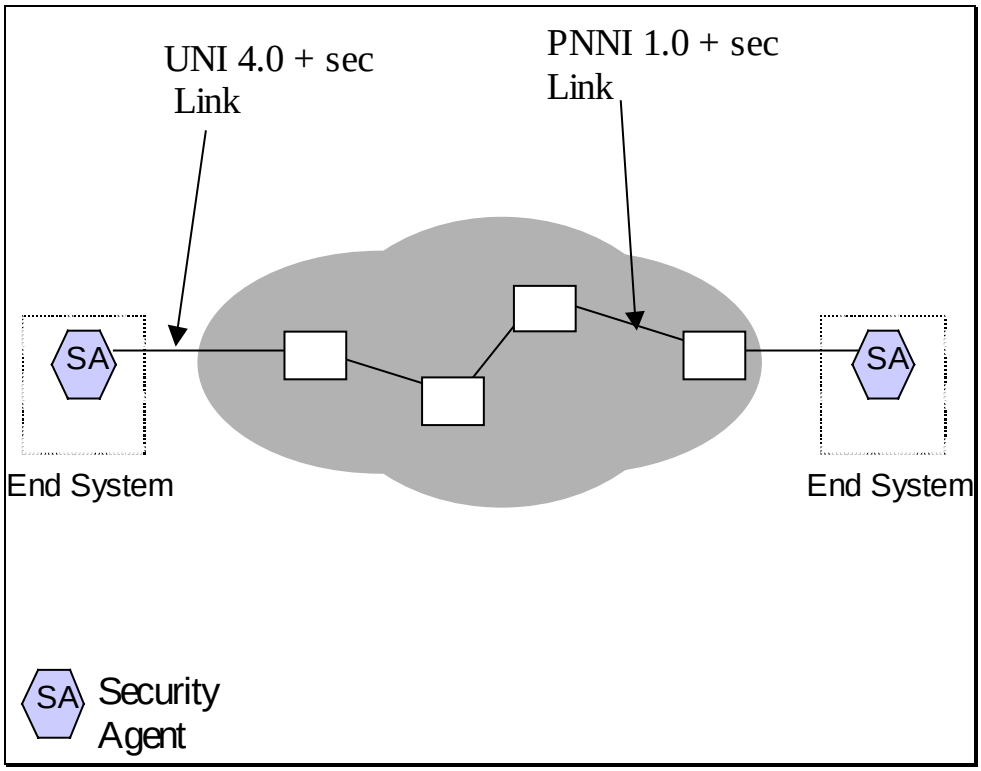


Figure 7. Entire Network Supports Signaling-Based SME.

A more likely scenario is one that contains network elements that do not support signaling-based SME. An example of this scenario is shown in Figure 8.

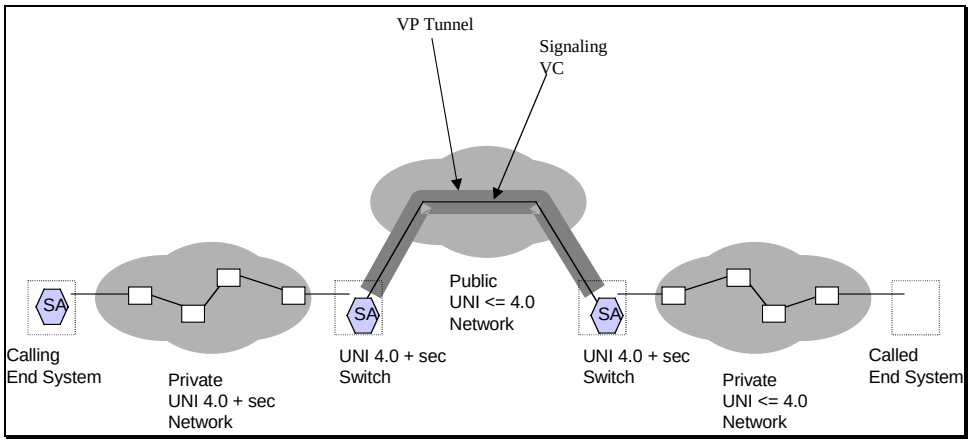


Figure 8. Portion of Network Does Not Support Signaling-Based SME.

In this scenario, the calling endpoint is attached to a network that supports signaling-based SME. The initiating security agent in this endpoint supplies security information along with the signaling message, and the message traverses the private network. When the message reaches the boundary UNI 4.0 + sec switch, it is signaled through a Virtual Path tunnel to the remote switch, which also supports signaling-based SME. Although the public network does not support signaling-based SME, the VP tunnel allows the two border switches to signal each other directly, and hence, allows SME to traverse the public network.

Since the remote switch contains a security agent that terminates the security association, the remaining network elements do not need to support signaling-based SME.

If none of the network elements are expected to support signaling-based SME, then the In-band SME protocol introduced earlier can be used instead. An example where this protocol should be used is illustrated in Figure 9.

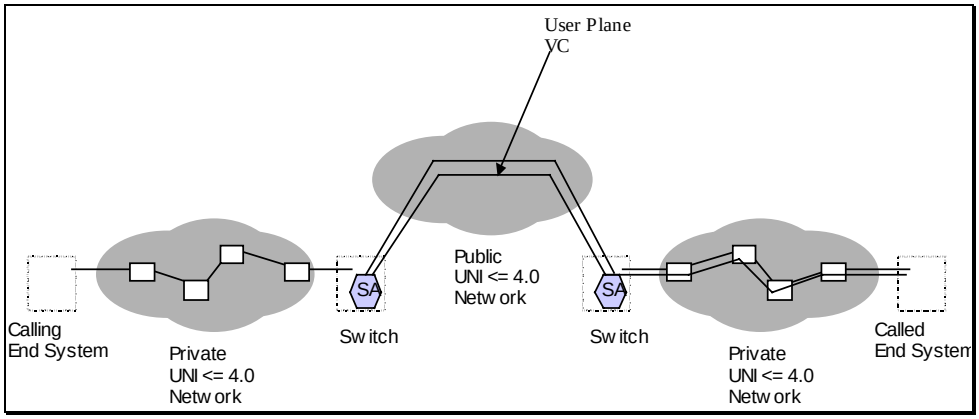


Figure 9. Using the In-band SME Protocol in Networks that do not support Signaling-Based SME.

In this case, the calling end system initiates the connection setup request, and the request propagates to the called end system as usual. As the call setup request progresses, the security agents augment the signaling message with security information, which is dropped by the networks (because they do not recognize the security services information elements). As a result, the downstream security agents do not see security information in the call setup message. As the connection confirmation signal propagates back from the called end system to the calling end system, the security agents augment the signaling message with security information, which is also dropped by the networks. However, as the network propagates the connection confirmation, the user plane virtual circuit is constructed along the way. When the user plane VC connects the two security agents, data transfers from the end systems are blocked, and the SAs perform the SME protocol within this virtual circuit. This protocol is described in more detail in Section 5.1.5.

Note: For voice calls, the voice path is typically cut through (voice traffic is allowed) in both directions except at the terminating exchange. At the terminating exchange the path is not cut through to the called party. The backward path is cut through to allow for the calling party to hear tones and announcements that may occur before answer (e.g., ringing, reorder, busy, “the number you have dialed is no longer in service”) and to avoid speech clipping when the called party picks up and says hello. If the Security Agent blocks the voice path to be used in the backward direction prior to connect (e.g., after an ALERTING or PROGRESS message) then the calling party might abandon the call or miss receiving important in-band information. In the case where an end station requests security services, it should not assume such services are operational until the CONNECT message is received and the SME is complete.

Regardless of the SME method used, after the SME is complete, OAM flows may be used to maintain the state of the security associations. Specifically, Security OAM cells are used:

- To exchange additional connection session keys for confidentiality and data integrity as needed. (Note that the initial session keys are established during SME.)
- To maintain synchronization of encryption engines relying on the counter mode of operation.

For the purposes of exposition, the functions of the Security Agent may be decomposed into several sections:

- SA_{sme}, those functions of the security agent that interact with signaling, or with a peer security agent, to perform the Security Message Exchange,

- $SA_{service}$, those functions of the security agent that provide the actual authentication, integrity check, and confidentiality services,

- SA_{policy}, those functions of the security agent that determine when and how security must be applied,

- SA_{misc}, the remaining functions of the security agent such as OAM cell processing.

This division is for descriptive purposes only and does not imply how a specific implementation of a security agent is decomposed.

3. Security Services for the User Plane

3.1. Entity Authentication

3.1.1. Authentication Reference Model

Authentication is provided via an exchange of information between SA_{sme}s that proves to the authenticating $SA_{service}$ that the authenticated $SA_{service}$ possesses a secret that is known solely by the authenticated entity (or its proxy) or exclusively by the two entities (or their proxies). Authentication may either be bi-directional (calling party to called party or parties and vice versa) or uni-directional. The authenticated entities in all cases are the Security Agents associated with the user plane entities that terminate the ATM virtual path or channel. If provided, ATM authentication shall be on a *per-VC basis*. This means that

- The decision to authenticate or not is determined on a VC-by-VC basis.
- Authentication is performed once for a connection, at the beginning of that connection.

Authentication is effected by means of nonces and message digests as described in Section 5.1, using either symmetric or asymmetric mechanisms.

As Sections 5.1.2 and 5.1.5 explain, two ways are specified for negotiating security services: signaling-based and in-band. There are differences in the reference models for how the authentication service is provided for the two cases.

Figure 10 shows the object and layer reference models for the authentication service in the case where security messaging is signaling-based. In this case SA_{sme} is called by signaling. The SA_{sme} coordinates with the control entity so that authentication is provided for the user plane entities requesting it.

The object and layer reference models for the case where security messaging is in-band (Figure 11) is similar to the signaling-based model except that the SA_{sme} completes its exchange by passing messages in the user data stream, as shown in Figure 11. Otherwise the object interactions are as for the signaling-based case.

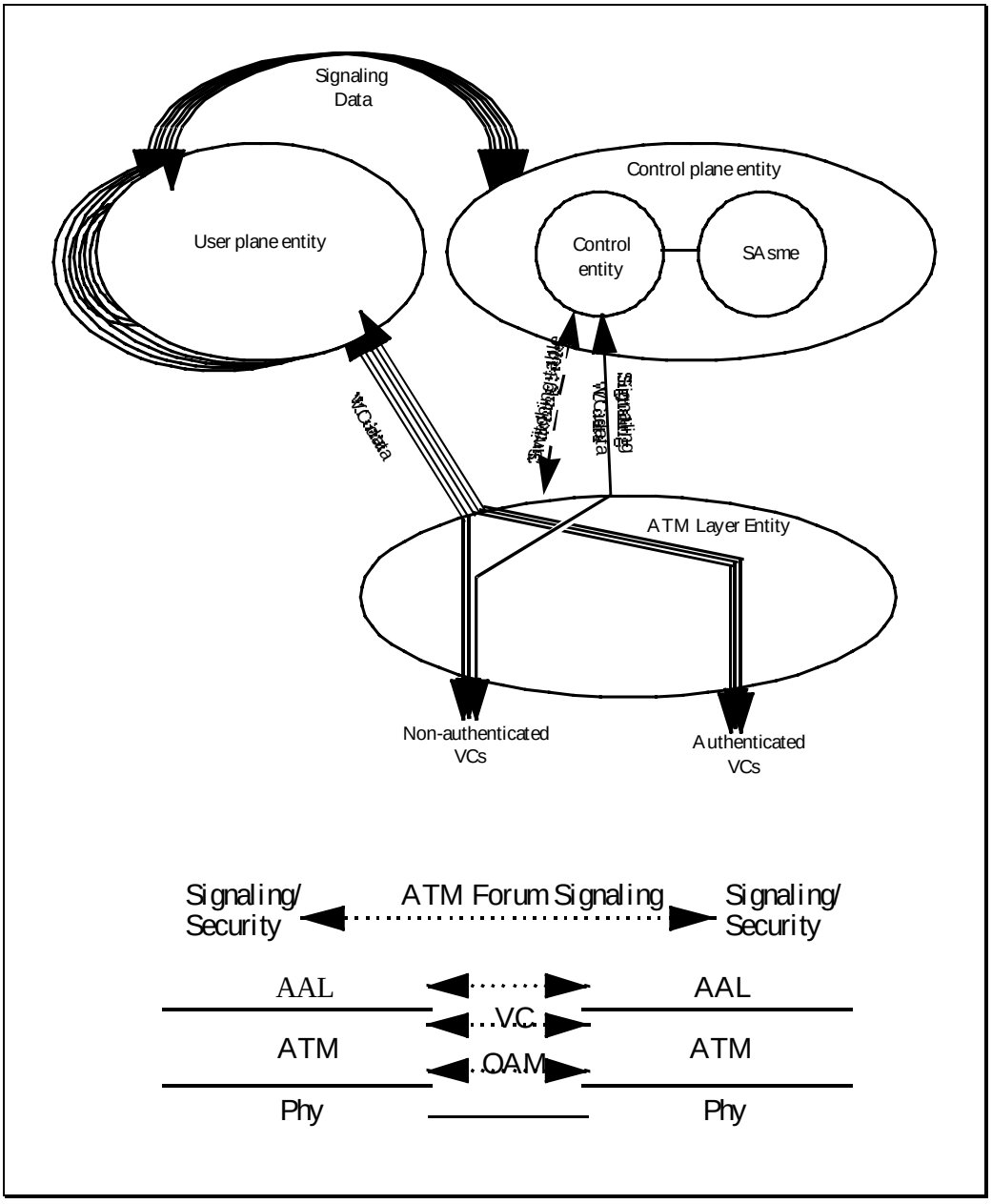


Figure 10. Object and Layer Reference Models for Authentication, Signaling-Based Messaging.

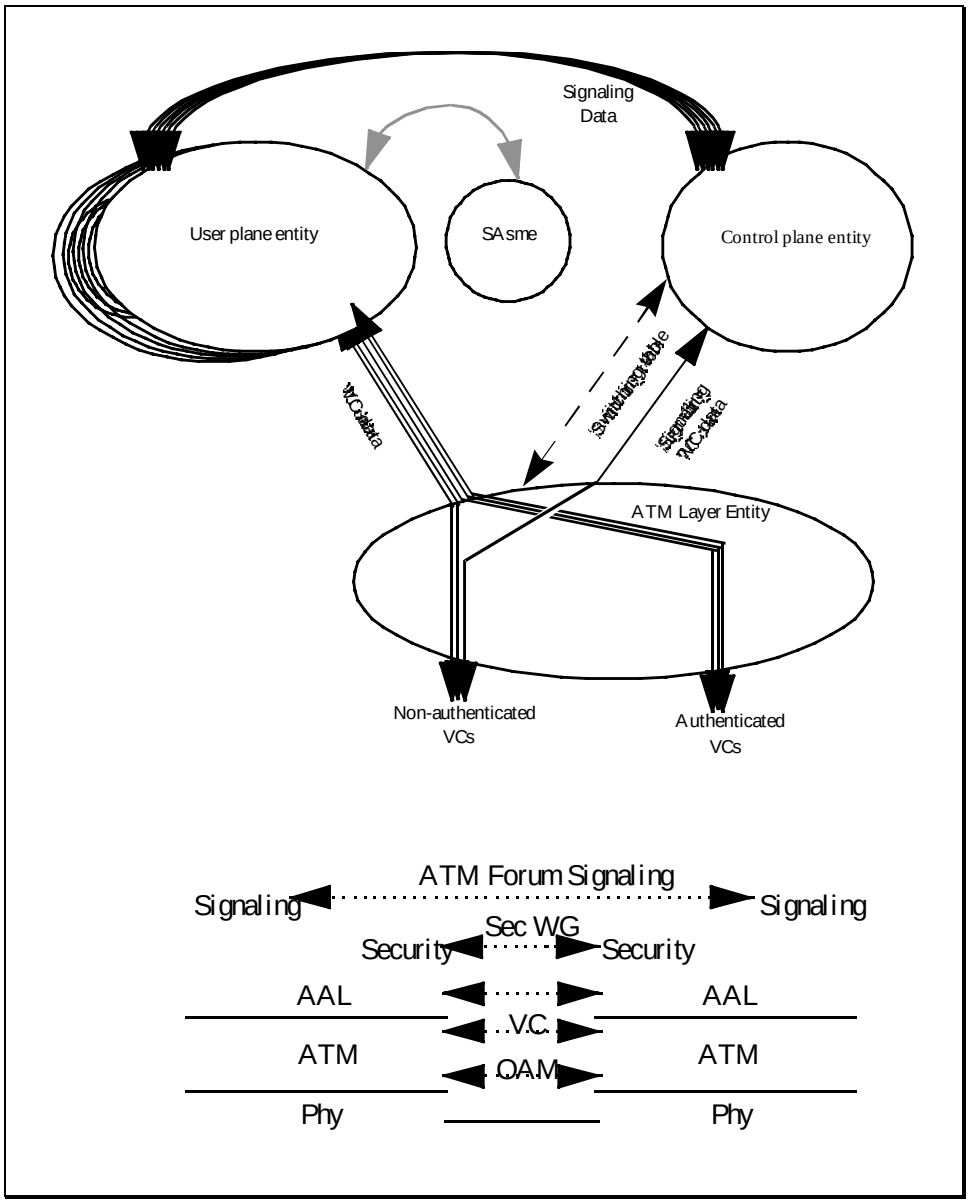


Figure 11. Object and Layer Reference Models for Authentication, In-band Security Messaging.

3.1.2. Authentication Infrastructure and Mechanisms

This authentication protocol is described in the context of the "security message exchange protocol" in Section 5.1. The messages that are used by this protocol are described in Section 5.1.5.3, Section 5.1.4.4, [4], and [5].

3.1.3. Authentication Algorithms

Authentication is provided in the context of the Security Message Exchange protocol, described in Section 5.1. Authentication is performed using either symmetric (secret) key algorithms (such as DES) or asymmetric (public) key algorithms (such as RSA). With symmetric algorithms, before authentication can take place between two nodes, the two nodes need to have a shared secret key. With asymmetric key algorithms, the nodes only need to know each other's public keys.

Before authentication can take place between two security agents, the two $SA_{service}$s need to obtain keys for the cryptographic techniques.

In the case of symmetric algorithms, the two $SA_{service}$s need to have a pair of shared secret uni-directional keys. These keys are pre-placed through methods outside the scope of this document (e.g., manual configuration).

In the case of asymmetric key algorithms, the $SA_{service}$s need to know each other's public keys. These public keys may be obtained in three ways:

1. By retrieving the SA's public key certificate from a public key directory,
2. By exchanging public key certificates directly during security negotiation, or
3. By pre-placing public keys using methods outside the scope of this document.

Note: the generation, distribution, and storage of the secret, private, and/or public keys is outside the scope of this specification.

The following sections define which algorithms may be used with either method (symmetric or asymmetric) of this authentication protocol. Note that before this protocol can commence, both parties must negotiate the algorithm. This negotiation protocol is described in the context of the "security message exchange protocol" in Section 5.1.

Associated with each algorithm is a codepoint that is used for identification when negotiating security options. These codepoints are listed in Section 6.1. Procedures for using user-defined algorithm codepoints are also described in Section 6.1.

3.1.3.1. Asymmetric Digital Signature Algorithms

The following asymmetric (public-key) digital signature algorithms are defined in this specification for user plane authentication.

Algorithm	Normative Reference
RSA	**[**[24]
DSA	[9]
Elliptic Curve/DSA Like	Section 6.9
ESIGN	Section 6.5

3.1.3.2. Symmetric Digital Signature Algorithms

The following symmetric (secret-key) digital signature algorithms are defined in this specification for user plane authentication.

Algorithm	Normative Reference
DES (in CBC mode)	[6], Section 6.6
DES40 (in CBC mode)	Section 6.6, Section 6.7
Triple DES (in CBC mode)	[25], Section 6.6
FEAL (in CBC mode)	[32], [33], Section 6.6

3.1.3.3. Hash Function

The following hash functions are defined in this specification for use with both asymmetric and symmetric digital signature algorithms.

Hash Function	Normative Reference
MD5	[12]
SHA-1	[8]
RIPEMD-160	[18]

3.1.4. Error Processing

Error processing for the entity authentication service is described in the context of the security message exchange protocol in Section 5.1.

3.2. Confidentiality

3.2.1. Confidentiality Reference Model

Confidentiality is provided via *encryption*. If provided, ATM encryption shall be on a *per-VC basis* (recall that "VC" could be either a Virtual Channel Connection [VCC] or Virtual Path Connection [VPC]). This means that:

- Encryption is applied to the sequence of data cells in a single VC.
- The decision to encrypt or not is determined on a VC-by-VC basis.
- For encrypted VCs, encryption parameters are determined (and may vary) on a VC-by-VC basis.

In all cases, encryption is applied to all or part of the 48-byte payloads of the relevant ATM cells. In terms of ATM architecture and layering, then, encryption takes place at the ATM layer. Other approaches to confidentiality (e.g., bulk link encryption, AAL-level encryption) are outside the scope of this document.

As Sections 5.1.2 and 5.1.5 explain, there are two ways specified for negotiating security services: signaling-based and in-band. There are differences in the reference models for how the confidentiality service is provided for the two cases.

Figure 12 shows the object and layer reference models for the confidentiality service in the case where SME is signaling-based. In this case, the SA_{sme} is called by the control plane entity. As explained above, the encryption/decryption function of $SA_{service}$ is logically part of the ATM layer entity, because all encryption is applied on a per-VC basis. When required by the selected cryptographic mode, the encryption/ decryption function of $SA_{service}$ maintains the state via OAM cells carried over end-to-end F4 or F5 flows (for VPCs and VCCs respectively).

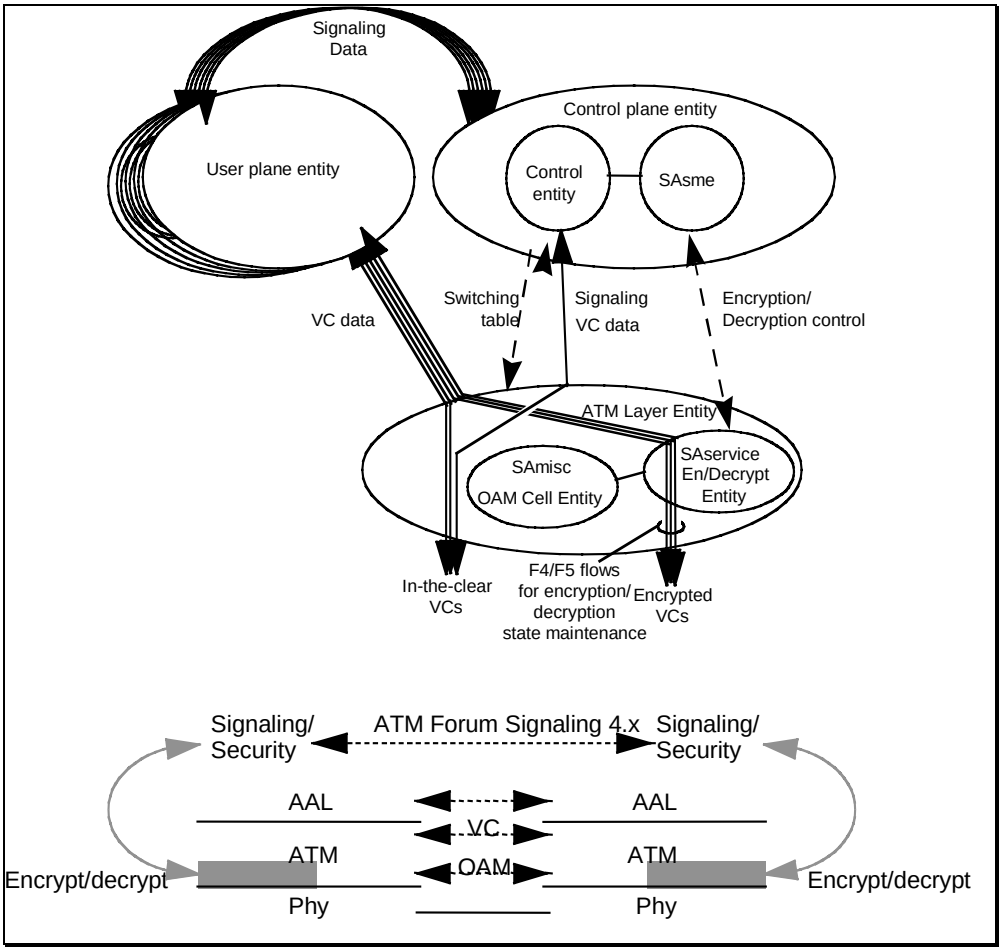


Figure 12. Object and Layer Reference Models for Confidentiality, Signaling-Based Messaging.

The object and layer reference models for the case where security messaging is in-band (Figure 13) are similar to the signaling-based models except for the placement of the SA_{sme}. For the in-band case, the SA_{sme} would move out of the control plane entity, as shown in Figure 13. Otherwise the object interactions are as for the signaling-based case.

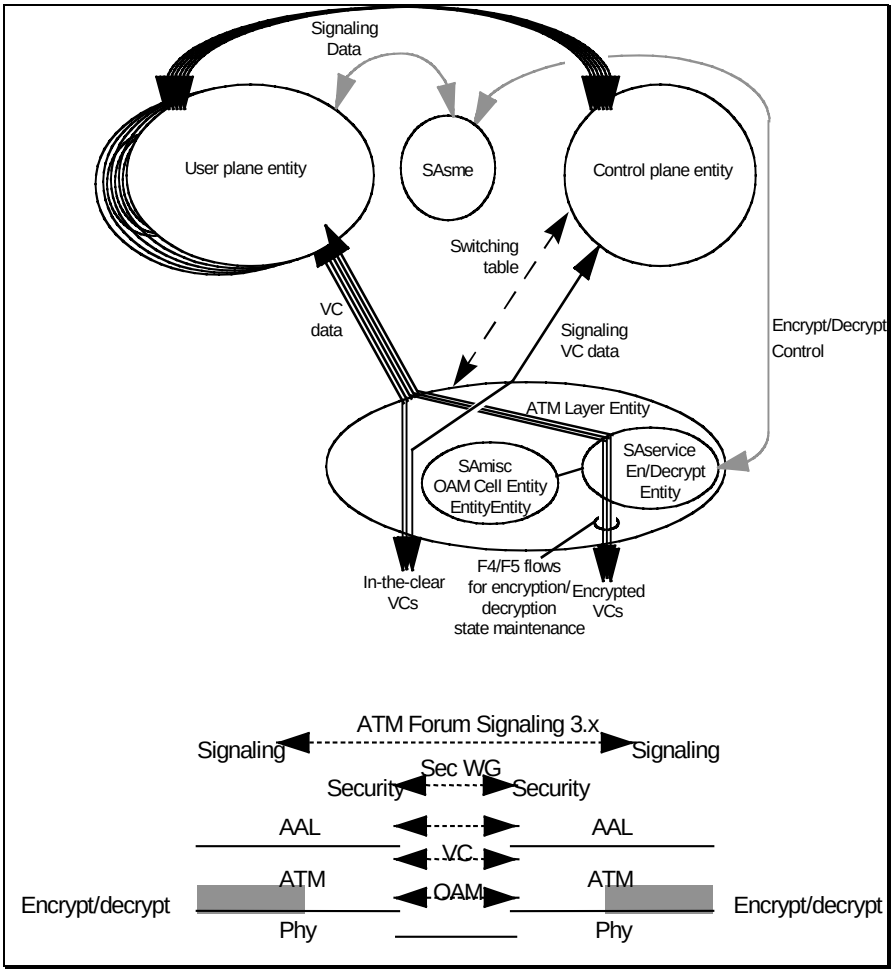


Figure 13. Object and Layer Reference Models for Confidentiality, In-band Security Messaging.

3.2.2. Confidentiality Infrastructure and Mechanisms

In this specification, symmetric (secret key) algorithms provide user plane confidentiality. In addition, the algorithms listed below must be used in a specific mode of operation. Keys for these algorithms can be either pre-placed (configured manually), or determined using the Key Exchange methods described in Section 5.2.

Associated with each algorithm and mode of operation is a codepoint that is used for identification when negotiating security options. These codepoints are listed in Section 6.1. Procedures for using user-defined algorithm codepoints are also described in Section 6.1.

The following algorithms are defined in this specification for user plane confidentiality.

Algorithm	Normative Reference
DES (56 bit effective key)	[6]
DES40 (40 bit effective key)	Section 6.7
Triple DES (112 bit effective key)	[25]
FEAL (64 bit key, no key block parity, N=32)	[32], [33]

The following modes of operation are defined in this specification for user plane confidentiality.

Mode	Normative Reference
Cipher Block Chaining (CBC)	[7]
Counter Mode	Section 6.3
Electronic Codebook (ECB)	[7]

Counter Mode is not self-synchronizing. Therefore, it is necessary to maintain synchronization of cryptographic state between the encrypting and decrypting security agents. This is accomplished via the OAM cells as described in Section 6.4.

This specification provides a mechanism for changing session keys for the confidentiality service. This mechanism is described further in Section 5.3.

3.2.3. Bypass of Special Network Cells

Because the Data Confidentiality service is defined at the ATM layer, it is necessary to clarify its operation with respect to special network cells, that is, cells other than user data cells. Because these special cells, i.e. OAM cells or RM cells, are interpreted by intermediate nodes in the network, they must be excluded by the encryption process. Otherwise they would be rendered meaningless, as it is not possible or desirable to have these intermediate nodes involved in the encryption process. To this end the following shall apply:

- For Virtual Channel Connections (VCCs), cells with a Payload Type Indicator (PTI) field of 1xx (binary) shall bypass the encryption process, and be transmitted in clear (where “x” in the PTI means don’t care). In addition, unless specifically defined herein, these cells shall not change the cryptographic state of the security agent.

- For Virtual Path Connections (VPCs), cells with a Virtual Channel Identifier (VCI) field with the following values: 3, 4, and 6-15 shall bypass the encryption process and be transmitted in clear. In addition, unless specifically defined herein, these cells shall not change the cryptographic state of the security agent.

3.3. Data Origin Authentication and Integrity

3.3.1. Integrity Reference Model

Data integrity is provided by means of cryptographic checksums (also known as hash codes, message authentication codes [MACs], and message digests) appended to AAL 3/4 and AAL 5 “Common Part” service data units (SDUs). Data integrity service is available only for virtual channels, not for virtual paths. If provided, ATM data integrity shall be on a *per-VCC basis*. This means that

- Integrity protections do not apply to VPCs.
- Checksums are applied to the sequence of Common Part SDUs in a VCC.
- The decision to provide integrity or not is determined on a VCC-by-VCC basis.
- For VCCs with checksums, integrity keys are determined (and may vary) on a VCC-by-VCC basis.

In terms of ATM architecture and layering, the data integrity function takes place at the AAL. Other approaches to data integrity are outside the scope of this document.

As Sections 5.1.4 and 5.1.5 explain, there are two methods specified for negotiating security services: signaling-based and in-band. There are differences in the reference models for how the data integrity service is provided for the two cases.

Figure 14 shows the object and layer reference models for the data integrity service in the case where security messaging is signaling-based. In this case, the SA_{sme} is called by the control plane entity upon receipt of, or prior to sending, all messages. As explained above, the data integrity function of $SA_{service}$ is logically part of the AAL entity, because MACs are applied on a per-VCC basis

The object and layer reference models for the case where security messaging is in-band (Figure 15) is similar to the signaling-based model except for the placement of the SA_{sme}. For the in-band case, the SA_{sme} would move out of the control plane entity, as shown in Figure 15. Otherwise the object interactions are as for the signaling-based case.

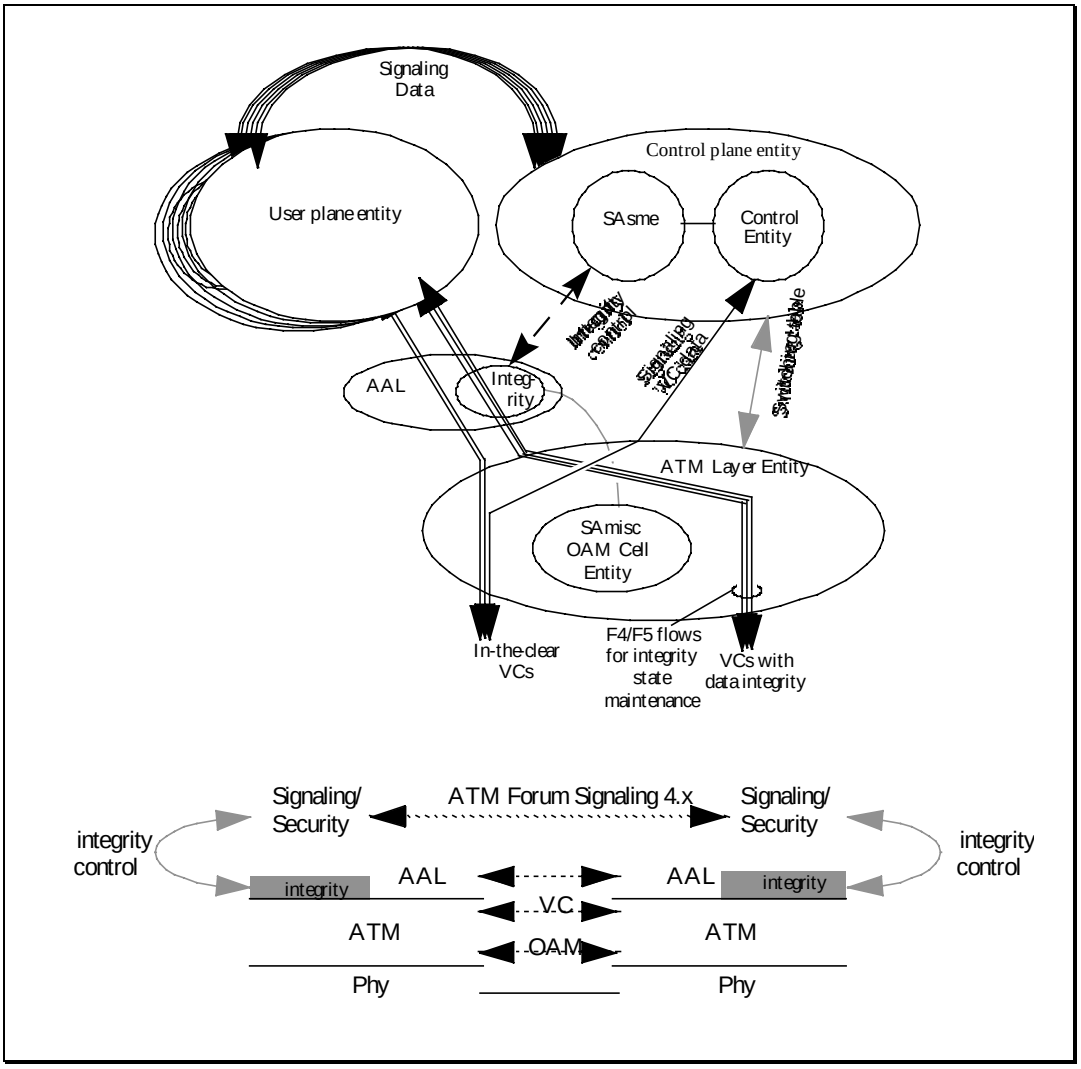


Figure 14. Object and Layer Reference Models for Data Integrity, Signaling-Based Messaging.

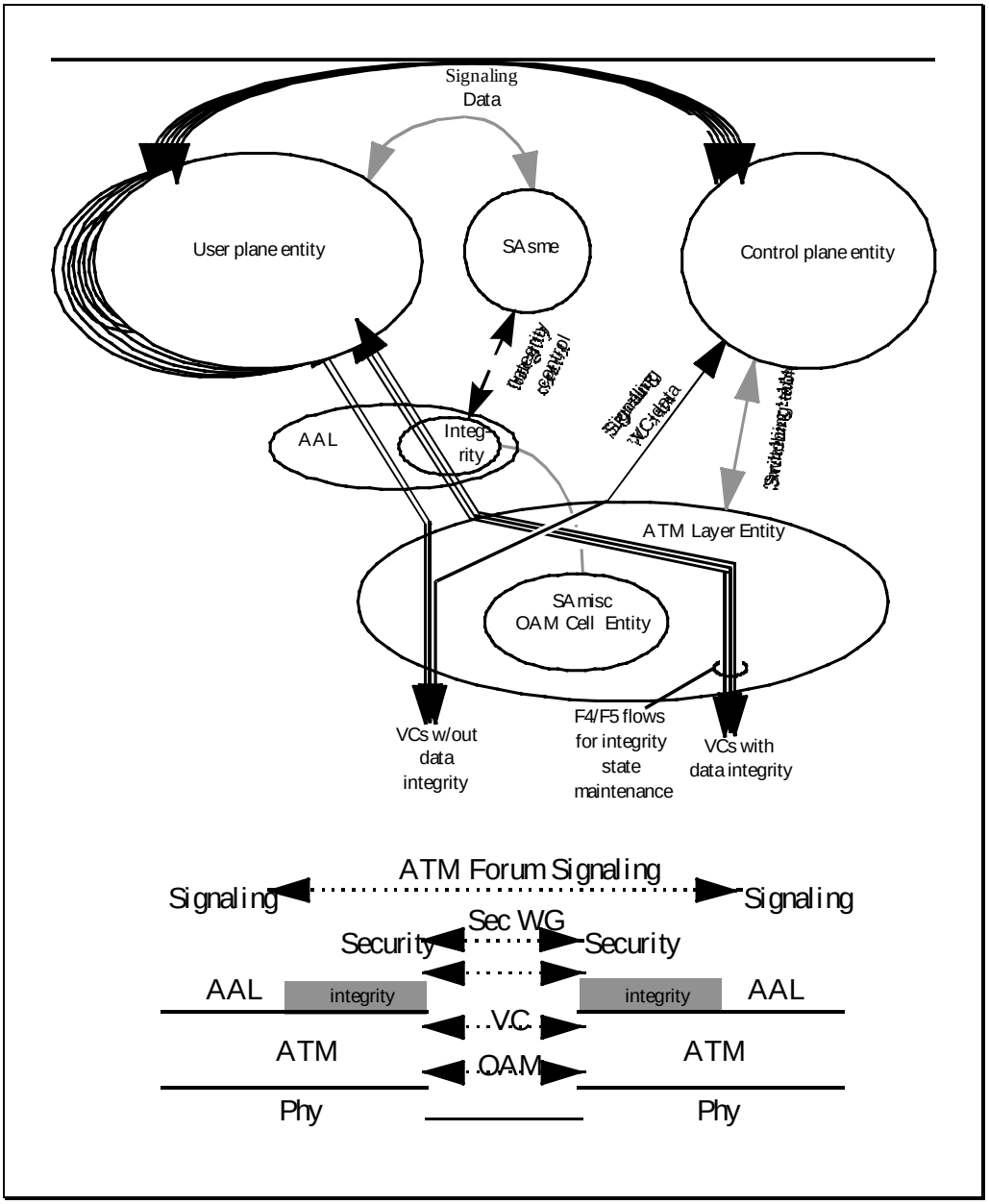


Figure 15. Object and Layer Reference Models for Data Integrity, In-band Security Messaging.

3.3.2. Integrity Infrastructure and Mechanisms

The data integrity mechanism supports data integrity for the “Common Part” AAL SDU in AAL type 3/4 and AAL type 5 for user data on a per VCC basis. The VCC may be a switched VCC (i.e., SVC) or a permanent VCC (i.e., PVC).

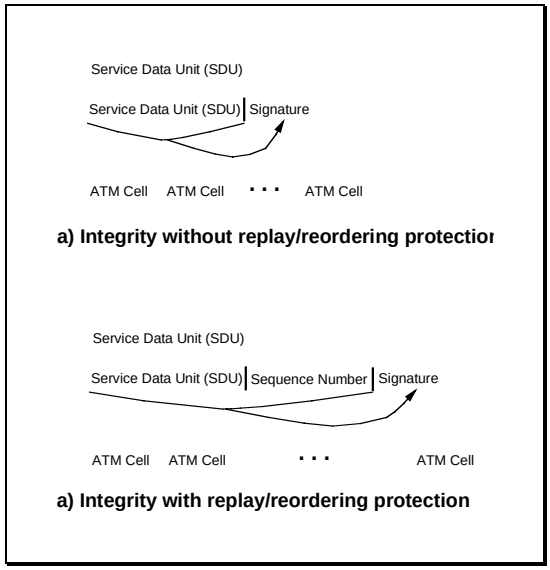


Figure 16. AAL-SDU Level Data Integrity.

The data integrity service is provided with two options: 1) data integrity without replay/reordering protection, and 2) data integrity with replay/reordering protection. The option to be used for a connection may be negotiated at the time the connection is established.

When data integrity is provided without replay/reordering protection, the source appends a cryptographic signature, commonly referred to as message authentication code (MAC), to the tail of each Common Part AAL-SDU before transmitting it, as shown in Figure 16. The signature is computed over the Common Part AAL-SDU.

At the destination, when a Common Part AAL-SDU is received, the security agent verifies the signature. If the signature does not check, it simply discards the AAL-SDU.

When data integrity is provided with replay/reordering protection, the data integrity service provides protection against replay and reordering attacks, so that “old” or “reordered” AAL-SDUs are detected and discarded. At the source, this is achieved by first appending a sequence number to the tail of each Common Part AAL-SDU and then computing a signature on the totality of the Common Part AAL-SDU including the sequence number. This signature, which protects both the AAL-SDU and the sequence number, is then appended to the total Common Part AAL-SDU (which includes the sequence number), as shown in Figure 16.

The sequence number is set to zero when the connection is established and each time the session key for data integrity is updated (i.e., changed over). The sequence number is incremented with each AAL-SDU that the source sends. The sequence number is a 6-byte number. The sequence number shall not wrap around during the life of a session integrity key.

At the destination, when a Common Part AAL-SDU is received, the security agent verifies that the signature is valid. If the signature is not valid, it discards the AAL-SDU. If the signature is valid, then it checks the sequence number to ensure that it is valid.

The receiving security agent tests the sequence number as follows. When there is a previously received valid Common Part AAL-SDU, then the sequence number associated with the current Common Part AAL-SDU is valid only if it’s greater than the sequence number associated with the last valid Common Part AAL-SDU received. When there is no previously received valid Common Part AAL-SDU, the sequence

number associated with the current Common Part AAL-SDU is accepted as valid. If the sequence number fails these tests, the security agent discards the AAL-SDU.

The receiving security agent saves the sequence number associated with the last valid Common Part AAL-SDU received to check the validity of the next Common Part AAL-SDU that it may receive.

3.3.2.1. Signature

The signature algorithm (or MAC) to be used for the connection's integrity service is negotiated at the time the connection is established.

Associated with each algorithm is a codepoint that is used for identification when negotiating security options. These codepoints are listed in Section 6.1. Procedures for using user-defined algorithm codepoints are also described in Section 6.1.

The following algorithms are defined in this specification for user plane integrity.

Algorithm	**Normative Reference**
HMAC-MD5	[11], [12]
HMAC-SHA-1	[11], [8]
HMAC-RIPEMD-160	[11], [18]
DES/CBC MAC	[17], [6], Section 6.6
DES40/CBC MAC	Section 6.6, Section 6.7
Triple DES/CBC MAC	[25], Section 6.6
FEAL/CBC MAC	[32], [33], Section 6.6

As with the confidentiality service, this specification provides a mechanism for changing session keys for the integrity service. This mechanism is described further in Section 5.3. Note, however, that because SKC OAM cells may not arrive on SDU boundaries, key changeover for data integrity that does not consider SDU boundaries may result in data loss.

3.4. Access Control

3.4.1. Access Control Reference Model

Access control is provided on a per-VC basis. In general, access control is performed during connection establishment based on information contained in the security message exchange and network configuration parameters.

As Sections 5.1.2 and 5.1.5 explain, there are two methods specified for negotiating security services: signaling-based and in-band. For both of these methods, there are differences in the reference models for how the access control is described below.

Figure 17 shows the access control layer reference model for the access control service in the case where security messaging uses the signaling-based method of negotiating security services. In this case, the SA_{sme} is logically part of the control plane entity.

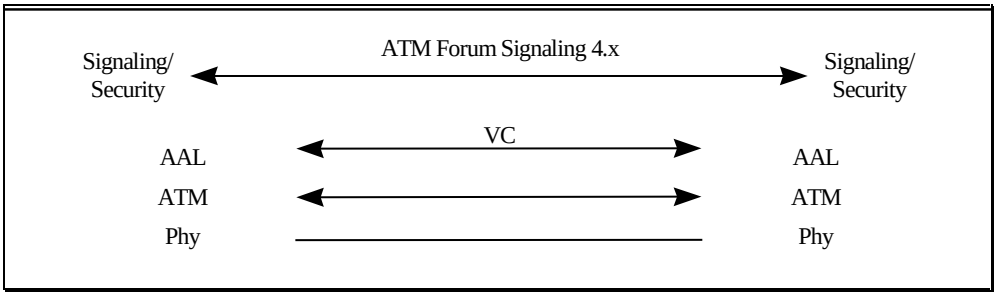


Figure 17. Access Control Layer Reference Model for Signaling-Based Messaging.

The access control layer reference model for the in-band method of negotiating security services (Figure 18) is similar to the signaling-based model except for the placement of the SA_{sme}. For the in-band case, the SA_{sme} would move out of the control plane entity, as shown in Figure 18. Otherwise, the interactions are the same as for the signaling-based case.

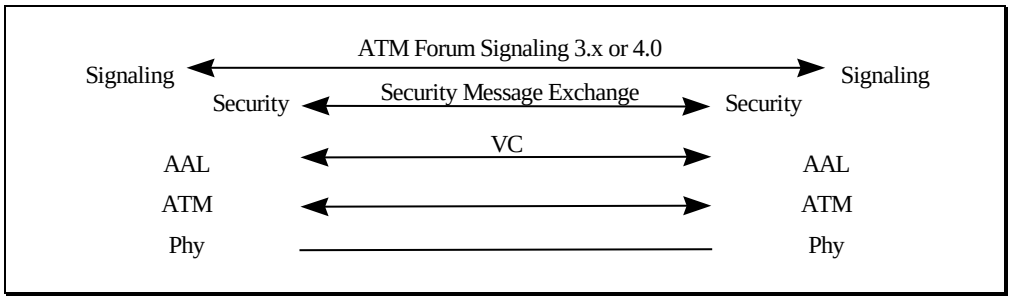


Figure 18. Access Control Layer Reference Model for In-band-Based Messaging.

3.4.2. Access Control Infrastructure and Mechanisms

Associated with each mechanism is a codepoint that is used for mechanism selection when negotiating security options. These codepoints are not listed here, but rather in Section 6.1. Procedures for using user-defined algorithm codepoints are also described in Section 6.1.

Access control data is sent in the initial exchange between security agents so that access control decisions can be made, as necessary, at each ATM component in the connection path during connection establishment.

The following algorithms are defined in this specification for user plane access control.

Algorithm	Normative Reference
Standard Security Label	[10]

Section 7.2 provides additional information on the use of security labels within an ATM network.

3.4.3. Error Processing

If an error prevents establishing a connection, an error message shall be returned to the calling party to enable connection tear-down along the path of the connection setup according to the error processing

procedures described in Sections 5.1.4 and 5.1.5. The amount of diagnostic information provided in the Cause Information Element shall be configurable.

4. Security Services for the Control Plane

4.1. Control Plane Data Origin Authentication and Data Integrity

For Version 1.0 of this specification, Control Plane data origin authentication and data integrity are accomplished in a hop-by-hop manner between adjacent signaling elements. Figure 19 shows the reference model for this service. This service is accomplished by applying the user plane data origin authentication and integrity service (Section 3.3) to the Signaling AAL (SAAL) SDU connecting two signaling entities.

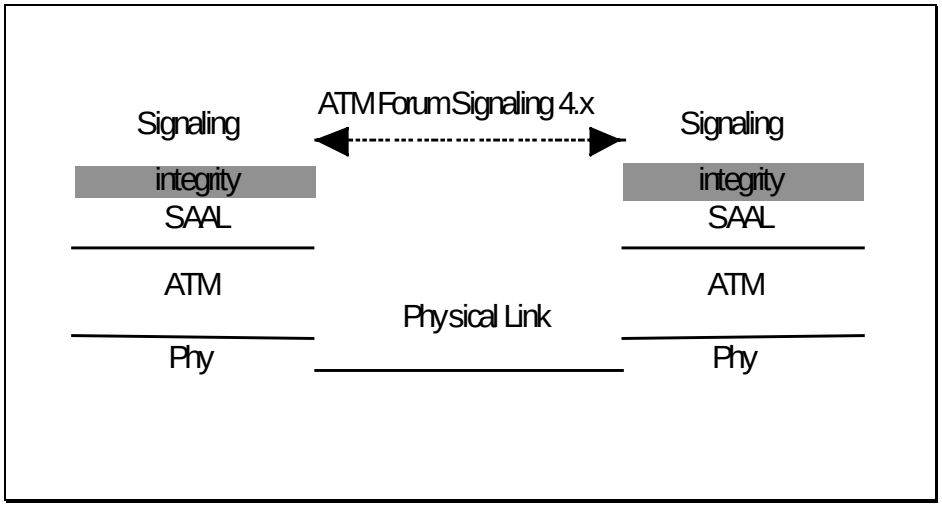


Figure 19. Control Plane Data Origin Authentication and Integrity Layer Reference Model.

The particulars for the Control Plane data origin authentication and integrity service are as follows:

The provision of control plane data origin authentication and data integrity and the algorithms and parameters required to effect this service are not dynamically negotiated; rather, management prearranges them. See Section 3.3.2.1 for a list of the signature algorithms that may be used and Section 6.2.3.2 for the parameters needed to operate those algorithms.

The control plane data origin authentication and data integrity service shall support session key update. The provision of session key update and the algorithms and parameters required to effect it are not dynamically negotiated; rather, management prearranges them. See Section 5.3 for a list of the signature algorithms that may be used and Section 6.2.3.6 for the parameters needed to operate those algorithms.

The two ends of the signaling channel shall be preconfigured with a shared secret master key and initial session key. Session key updates may take place according to the procedures in Section 5.3.

The control plane data origin authentication and data integrity service shall always use the “data integrity with replay/reordering protection” option.

The mechanisms that provide Control Plane data origin authentication and data integrity service are identical to those described in Section 3.3.2 with the following exception:

> *The Control Plane data origin authentication and data integrity service operates at the SAAL SDU level rather than at the AAL5 common part SDU level.*

Thus, the Control Plane data origin authentication and data integrity service operates “above” the SAAL and “below” the signaling protocol. For outgoing signaling messages, sequence numbers and signature fields are added to the SAAL SDU. For incoming signaling messages, the fields are removed from the SDU and processed by the $SA_{service}$ before the signaling messages are passed to the signaling entity for further processing.

4.1.1. Error Conditions

If the two endpoints of a signaling channel are not compatibly configured, the SSCOP connection will never be established, so no signaling messages will ever pass between the two entities. If the AAL5 SDU is modified in transit, it will be discarded as defined in Section 3.3, and SSCOP will retransmit.

5. Support Services

This section describes the mechanisms that are required to support the security services described in Section 3 and Section 4. Specifically, the following support services are addressed in this section:

- Security message exchange protocols and basic negotiation,
- Security messaging in the control plane,
- Security messaging in the user plane,
- Key exchange,
- Session key update,
- Certificates.

In order to negotiate parameters for the security services described in Section 3, and to directly support the entity authentication service described in Section 3.1, a set of security message exchange protocols is required. These protocols are described in Section 5.1 and are summarized below.

The three-way security message exchange protocol described in Section 5.1.1.1 may be used for establishing a point-to-point connection, as well as for the first leaf in a multipoint call. This protocol is used for connections that require negotiation of security options. The three-way exchange has the advantage that it does not use timestamps, and therefore does not require synchronization.

The two-way security message exchange protocol described in Section 5.1.1.2 may also be used for establishing a point to point connection or a point to multipoint connection. This protocol is used for connections that do not require negotiation of security parameters, and for adding leaves to multipoint calls. The disadvantage of the two-way exchange protocol is that it requires time synchronization between the party that generates the security information and the party that validates the security information.

This specification defines two mechanisms for transporting security information. These are the signaling-based security message exchange mechanism (described in Section 5.1.2) and the in-band security message exchange mechanism (described in Section 5.1.5). In both cases, the Security Services Information Element (described in Section 5.1.5.3.6) is used to carry the security information.

The method for performing the two-way exchange protocol in Signaling 4.0 flows is described in detail in Section 5.1.2. (The three-way message exchange protocol is not supported in Signaling 4.0 in this specification.)

For point-to-multipoint connections after the first leaf is established, subsequent leaves are added with a two-way security message exchange. This is because negotiation of security options may be performed only when establishing the first leaf—subsequent leaves must accept the options that the root and the first leaf agreed upon.

The method for performing the two-way and three-way message exchange protocols in the user plane VCC/VPC is described in detail in Section 5.1.5. This method applies to both SVCs and PVCs. In order to provide a reliable transport service for inband message flows, an inband message exchange protocol is defined in Section 5.1.5.3. As with the signaling-based approach, this protocol uses the Security Services Information Element to convey security-related parameters.

PVCs (permanent virtual connections) are provisioned connections. Security services negotiation, authentication, certificate exchange, and key exchange can be done via provisioning at the time PVCs are established, or inband as described in Section 5.1.4.4. Once security services for PVCs are established, the data confidentiality and data integrity services for PVCs are provided the same way as they are provided for SVCs. Likewise, session key update (for data confidentiality and data integrity services) for PVCs is done the same way as it is done for SVCs.

When a SVC or PVC is established, confidentiality and/or integrity (data origin authentication) keys need to be exchanged in order to provide these services. In order to prevent "man in the middle" attacks, key exchange must be accompanied by strong authentication between the user endpoints, as addressed in Sections 5.1.1.1 and 5.1.1.2. Other details associated with the key exchange service are described in Section 5.2.

Once confidentiality and/or integrity keys are exchanged, there is no need for security message exchanges in the middle of connection after the connection is established. However, session keys for these services must be changed periodically. Section 5.3 describes the key update mechanism (for the data confidentiality and data integrity services), which is performed by exchanging a "master key" at the connection setup time and then exchanging subsequent "session" keys under the master key. Session key update messages are exchanged using OAM cells.

Section 5.4 describes the certification infrastructure and mechanisms for transporting certificates and certificate chains. These certificates can be exchanged during the three-way security message exchange protocol, or through some other means that is outside the scope of this specification (e.g. directory servers).

5.1. Security Information Exchange

5.1.1. Security Message Exchange and Negotiation Protocols

The two-way and three-way user plane security message exchange protocols are not dependent on the use of any particular cryptographic algorithm and are based on the "strong authentication" procedures of the ISO/IEC 9594-8 asymmetric (public) key-based authentication and the ISO/IEC 11770-2 symmetric (secret) key-based authentication. Both protocols provide the following functions:

- Unilateral authentication,
- Mutual authentication,
- One- or two-way key exchange.

The three-way message exchange protocol provides the following additional functions:

- Certificate and Certificate Revocation List (CRL) exchange,
- Negotiation of security services and options.

In the two- and three-way security message exchange protocols, one user of the protocol assumes the "initiating" role and the other remote user assumes the "responding" role.

When setting up a point-to-point connection or when setting up the first leaf of a point-to-multipoint connection, the initiator has the option to use either the two-way security message exchange protocol or the three-way security message exchange protocol. When setting up subsequent leaves in a point-to-multipoint connection, the initiator can only use the two-way security message exchange protocol.

The following table shows the symbols and abbreviations used throughout the discussion of two- and three-way security message exchange protocols:

X	Entity *X* distinguished name (ID). Authentication requires that the entity to be authenticated have a distinguished name (ID) and an associated key. So, *X* represents the ID of the authentication entity.
K_X	When *X* uses an asymmetric (public) key algorithm (such as RSA), K_X represents the public or private component of *X*'s asymmetric key. When *X* uses a symmetric (secret) key algorithm (such as DES), K_X is *X*'s symmetric

	(secret) key.
$Enc_{K_X}(text)$	Encryption of *text* under *X*'s key. When *X* uses an asymmetric (public) key algorithm (such as RSA) for encryption, K_X is the public component of *X*'s asymmetric key. When *X* uses a symmetric (secret) key algorithm (such as DES) for encryption, K_X is *X*'s symmetric (secret) key.
$Sig_{K_X}(Hash(text))$	*X*'s digital signature computed over the hash of *text* under *X*'s key. When *X* uses an asymmetric (public) key algorithm (such as RSA) for digital signature, K_X is the private component of *X*'s asymmetric key. When *X* uses a symmetric (secret) key algorithm (such as DES) for digital signature, K_X is *X*'s symmetric (secret) key.
Hash(text)	One-way hash of *text*, where *Hash* is a strong one-way hash function such as MD5.
R_X	Random number (nonce) generated by *X*.
T_X	$T_X = (Time_X, Seq_X)$, time-variant timestamp generated by *X*. The time stamp is a combination of two values ($Time_X$, Seq_X). $Time_X$, is a 4-byte coordinated universal time, the Greenwich Mean Time (GMT) at which the signature was generated. This is the binary encoding of the number of seconds since 00:00:00 GMT on January 1, 1970 (same as UNIX time). Seq_X is a 4-byte sequence number that is incremented with each authentication flow that is sent to the same destination with the same $Time_X$ value. When the $Time_X$ value changes, Seq_X is reset to 0.
{.}	Optional token. A token included in {.} is only needed when the specific security service that requires it is invoked. For example, *{ConfPar}* is present only if the key exchange service is required for the connection.
SecOpt	In the two-way security message exchange protocol, the initiator uses the *SecOpt* token to indicate to the responder what security services are to be provided for the connection. *SecOpt* carries the security services, options, and parameters requested for the connection. This includes the type of security services to be provided for the connection (e.g., authentication, data confidentiality, etc.) and the algorithms/modes of operation to be used for each security service (e.g., authentication using DSA; key exchange using Diffie-Hellman; data confidentiality using DES in CBC mode, etc.). Additionally, *SecOpt* includes the parameters associated with each selected algorithm/mode of operation (e.g., initialization vector for CBC mode).
SecNeg_	In the three-way security message exchange protocol, the "initiator" and "responder" use $SecNeg_a$ and $SecNeg_b$ to negotiate the security services, options, and parameters for the connection. The initiator presents to the responder the following information using the $SecNeg_a$ token: **1)** The security services (e.g., authentication, data confidentiality, etc.) to be provided for the connection and any other security service that will be provided for the connection should the responder request it. **2)** For each security service that the initiator has requested and any other security service that will be provided for the connection should the responder request it, the initiator specifies one or more sets of options for each service. Options should be presented in order of preference, with the highest preference option first and the lowest preference option last. The responder must select the first compatible option presented to

	it. The responder can only select one of the options presented to it. For example, for data confidentiality, the initiator may indicate Triple-DES as first choice and FEAL as second choice. For authentication hash algorithm, the initiator may indicate MD5 as first choice and no other choices. For authentication signature algorithm, the initiator may indicate DSA as first choice and RSA as second choice, etc. **3)** The parameters associated with each option (for example, initialization vector for DES in CBC mode). When the responder receives these three pieces of information (i.e., items 1-3) from the initiator, it makes its selection from the list of options presented to it and communicates that to the initiator through the $SecNeg_b$ token, if the options presented to it are satisfactory. The connection setup can proceed only if the security negotiation step is successful, otherwise the connection setup will fail.
ConfPar_	When the key exchange support service option is invoked, $ConfPar_a$ is used to securely carry the initiator's keys from the initiator to the responder. $ConfPar_a$ contains: 1) the initiator's master key, 2) the initiator's first session key for the data confidentiality service when the data confidentiality service is requested for the connection, and 3) the initiator's first session key for the data integrity service when the data integrity service is requested for the connection. Similarly, $ConfPar_b$ is used to securely carry the responder's keys from the responder to the initiator. $ConfPar_b$ contains: 1) the responder's master key, 2) the responder's first session key for the data confidentiality service when the data confidentiality service is requested for the connection, and 3) the responder's first session key for the data integrity service when the data integrity service is requested for the connection.
Cert	In the three-way security message exchange protocol, when the certificate or certificate revocation list (CRL) exchange support service option is invoked, $Cert_a$ is used to carry the initiator's certificate or CRL chain from the initiator to the responder and $Cert_b$ is used to carry the responder's certificate or CRL chain from the responder to the initiator.

The formats of these tokens are defined in Section 6.1.

In the following discussion, when an asymmetric (public) key algorithm is used, it is assumed that each authentication entity (i.e., *A* and *B*) possesses a public/private key pair. K_a represents the public component of *A*'s asymmetric key when it's used for encryption. K_a represents the private component of *A*'s asymmetric key when it's used for digital signature. Similarly, K_b represents the public component of *B*'s asymmetric key when it's used for encryption. K_b represents the private component of *B*'s asymmetric key when it's used for digital signature.

In the following discussion, when a symmetric (secret) key algorithm is used, it is assumed that the authentication entities *A* and *B* share two uni-directional secret keys K_a and K_b, or a single secret key $K_a = K_b$.

Note: the generation, distribution, and storage of the secret, private, and/or public keys is outside the scope of this specification.

5.1.1.1. Three-Way Security Message Exchange Protocol

Figure 20 shows the three-way security message exchange protocol. This protocol supports both asymmetric and symmetric key algorithms.

When authentication or key exchange is invoked, this protocol uses a nonce-based mutual authentication method. As a result, *A* gets authenticated to *B* and *B* gets authenticated to *A*. However, note that *A* is not authenticated to *B* until FLOW3-3WE is received and checked.

This protocol involves the following steps, and is implemented in the In-Band Security Message Exchange Protocol, described in Section 5.1.5.3. If an error occurs during processing of this protocol, then the error procedures described in Section 5.1.5.3 shall apply.

1. *A* sends FLOW1-3WE (which contains the following required tokens: *A*, R_a, and $SecNeg_a$) to *B*. When certificate or CRL exchange is needed, $Cert_a$ is also included in FLOW1-3WE.

   FLOW1-3WE: $A \blacklozenge B$

   $$A, \{B\}, R_a, SecNeg_a, \{Cert_a\}$$

2. When *B* receives FLOW1-3WE, it carries out the following actions:

   - Checks that *B* itself is the intended recipient, when *B* is included.
   - Extracts $SecNeg_a$ and interprets it for its reply.
   - When authentication or key exchange is required

     ⇒ Extracts the nonce R_a for its reply.

     ⇒ Extracts $Cert_a$ if present and verifies its validity.

3. *B* sends FLOW2-3WE (which contains the following required tokens: *A*, *B*, and $SecNeg_b$) to *A*. When authentication or key exchange is required, the authentication token (i.e., $\{R_a, R_b,,...\}$) is included in the flow. When key exchange is needed, $ConfPar_b$ is also included in FLOW2-3WE. When certificate or CRL exchange is needed, $Cert_b$ is also included in FLOW2-3WE.

   FLOW2-3WE: $B \blacklozenge A$

   $$A, B, SecNeg_b, \{Cert_b\}, \{R_a, R_b, \{Enc_{k_a}(ConfPar_b)\}, Sig_{k_b}(Hash(A, B, R_a, R_b, SecNeg_a, SecNeg_b, \{ConfPar_b\}))\}$$

4. When *A* receives FLOW2-3WE, it carries out the following actions:

   - Checks that *A* itself is the intended recipient.
   - Extracts $SecNeg_b$ and interprets it.
   - When authentication or key exchange is required

     ⇒ Verifies the signature, and thus the integrity of both FLOW1-3WE and FLOW2-3WE.

     ⇒ Checks that the received R_a in FLOW2-3WE is identical to the one that sent in FLOW1-3WE.

     ⇒ Extracts the nonce R_b for its reply.

     ⇒ Extracts $ConfPar_b$ if present and interprets it.

     ⇒ Extracts $Cert_b$ if present and verifies its validity.

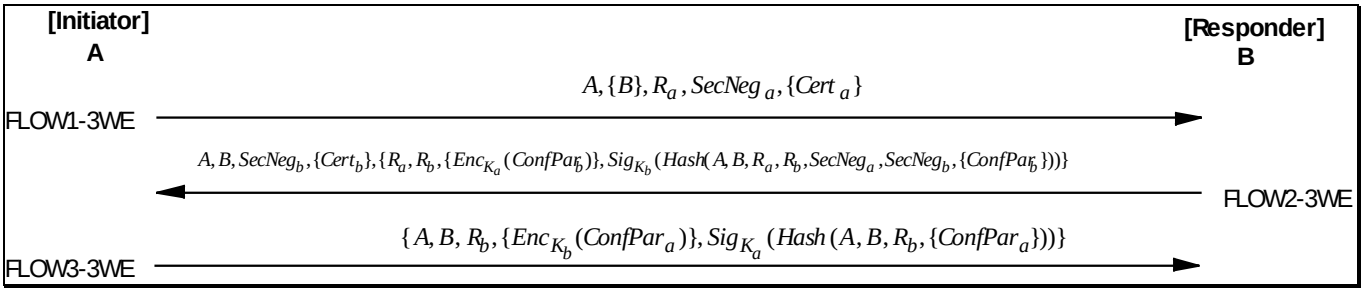


Figure 20. Three-Way Security Message Exchange Protocol.

5. When authentication or key exchange is required, *A* sends FLOW3-3WE (which contains the following required tokens: *A, B, R_b,...*) to *B*. When key exchange is needed, $ConfPar_a$ is also included in FLOW3-3WE.

   FLOW3-3WE: $A \blacklozenge B$

   $\{A, B, R_b, \{Enc_{k_b}(ConfPar_a)\}, Sig_{k_a}(Hash(A, B, R_b, \{ConfPar_a\}))\}$

6. When *B* receives FLOW3-3WE, it carries out the following actions:

   - Checks that *B* itself is the intended recipient.
   - When authentication or key exchange is required

     ⇒ Verifies the signature, and thus the integrity of FLOW3-3WE.

     ⇒ Checks that the received R_b in FLOW3-3WE is identical to the one that sent in FLOW2-3WE.

     ⇒ Extracts $ConfPar_a$ if present and interprets it.

5.1.1.2. Two-Way Security Message Exchange Protocol

Figure 21 shows the two-way security message exchange protocol. This protocol supports both asymmetric and symmetric key algorithms.

When authentication or key exchange is invoked, this protocol uses a timestamp and nonce-based mutual authentication method. As a result, *A* gets authenticated to *B*, and *B* gets authenticated to *A*.

This protocol involves the following steps, and is implemented within UNI 4.0 signaling, as described in Section 5.1.4. If an error occurs during processing of this protocol, then the error procedures described in Section 5.1.4 shall apply.

1. A sends FLOW1-2WE (which contains the following required tokens: *A*, *B*, and *SecOpt*) to *B*. When authentication (initiator -> responder) or key exchange is required, the authentication token (i.e., *{T_a,R_a,...}*) is included in the flow. When key exchange is needed, $ConfPar_a$ is also included in FLOW1-2WE. When certificate or CRL exchange is needed, $Cert_a$ is also included in FLOW1-2WE.

   FLOW1-2WE: $A \blacklozenge B$

   $A, B, SecOpt, \{Cert_a\}, \{T_a, R_a, \{Enc_{k_b}(ConfPar_a)\}, Sig_{k_a}(Hash(A, B, T_a, R_a, SecOpt, \{ConfPar_a\}))\}$

2. When *B* receives FLOW1-2WE, it carries out the following actions:

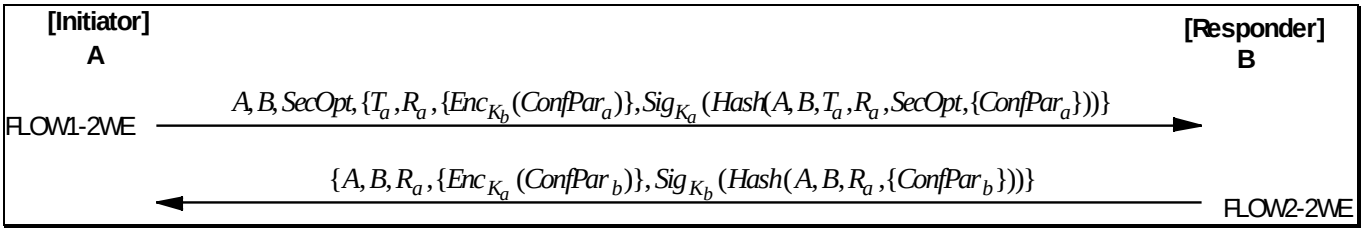


Figure 21. Two-Way Security Message Exchange Protocol.

- Checks that *B* itself is the intended recipient.
- Extracts *SecOpt* and interprets it.
- When authentication or key exchange is required

  ⇒ Verifies the signature, and thus the integrity of FLOW1-2WE.

  ⇒ Checks that the timestamp is fresh and that the flow is not a replay or out-of-order.

  ⇒ Extracts the nonce R_a for its reply.

  ⇒ Extracts $ConfPar_a$ if present and interprets it.

  ⇒ Extracts $Cert_a$ if present and verifies its validity.

3. When authentication (responder -> initiator) or key exchange is required, *B* sends FLOW2-2WE (which contains the following required tokens: *A, B,* R_a,...) to *A*. When key exchange is needed, $ConfPar_b$ is also included in FLOW2-2WE. When certificate or CRL exchange is needed, $Cert_b$ is also included in FLOW2-2WE.

   FLOW2-2WE: $B \blacklozenge A$

   $$\{A, B, R_a, \{Cert_b\}, \{Enc_{k_a}(ConfPar_b)\}, Sig_{k_b}(Hash(A, B, R_a, \{ConfPar_b\}))\}$$

4. When *A* receives FLOW2-2WE, it carries out the following actions:

   - Checks that *A* itself is the intended recipient.
   - When authentication or key exchange is required

     ⇒ Verifies the signature, and thus the integrity of FLOW2-2WE.

     ⇒ Checks that the received R_a in FLOW2-2WE is identical to the one which sent in FLOW1-2WE.

     ⇒ Extracts $ConfPar_b$ if present and interprets it.

     ⇒ Extracts $Cert_b$ if present and verifies its validity.

The use of timestamp, $T_A = (Time_A, Seq_A)$, will need to have three properties: 1) Freshness: the receiver is assured that the received flow has been originated within a specified range of the receipt time. 2) Uniqueness: the receiver is assured that the received flow has never been received before from the same

sender. 3) Ordering: the receiver is assured that the received flow has not been originated before a previously received flow from the same sender.

When the initiator generates $Time_A$ for FLOW1-2WE, it determines whether its value is greater than the $Time_A$ used in the previous FLOW1-2WE sent to this destination. If current $Time_A$ is the same as the previous one, then the initiator increments Seq_A. Otherwise, it resets Seq_A to 0.

When the responder receives a FLOW1-2WE from a source, it checks whether it has received another FLOW1-2WE from this source within a time-out period *W*.

If the responder has received another FLOW1-2WE from the same source within the time-out period *W*, it compares $Time_A$ values to ensure that the $Time_A$ in the recently received flow is greater than the $Time_A$ in the previously received flow. If they are the same, then it checks that Seq_A is greater than the previous one received from this source. If the timestamp fails these tests, then the authentication fails and the information is discarded.

If the responder has not received another FLOW1-2WE from the same source within the time-out period *W*, then it checks to ensure that $Time_A$ is within time *W* of its (the responder's) current local clock. If the timestamp fails this test, then the authentication fails and the information is discarded. This implies that the initiator and responder must be synchronized within the time period W.

When the timestamp test succeeds, the responder saves the timestamp and holds it for *W* seconds.

The value of the time-out period *W* is an implementation parameter and need not be specified here. The larger the *W* is, the higher is the clock drift tolerance. But a larger *W* requires more storage for timestamps.

5.1.2. Label Transport

Label Based Access Control uses a uni-directional security protocol between two ATM Security Agents, separate from the Security Message Exchange. Labels are transported in the SSIE (described in Section 5.1.3) using either the signaling-based or in-band mechanism. When labels are provided in signaling, any security agent that is involved in the connection SETUP or CONNECT message can make an access control decision based on the contents of the label. Because of this, the label is generally not discarded by the security agent that inspects the label. When labels are passed in-band, the security agents who receive the in-band exchange can enforce the access control policy.

5.1.3. Security Services Information Element

The Security Services Information Element (SSIE) is the principal communication method used by ATM Security Agents to establish security services for ATM VCs during ATM Call Establishment.

The SSIE is designed to provide direct support for transporting security information in signaling and within the user plane connection. This provides direct support for the Security Message Exchange Protocol described in Section 5.1, along with support for other exchange mechanisms.

The SSIE is designed to provide support for the security model outlined in this specification. The SSIE provides direct support for:

1. Signaling-based Security Message Exchange.
2. In-band Security Message Exchange.
3. Multiple Nested Security Services.
4. Proxy Security Agents.
5. User Plane Security Services.

The SSIE is composed of a Security Service Information Element header and any number of Security Association Sections (SAS). Each SAS contains the information needed to establish and maintain the security associations needed to provide a security service. In order to accomplish this, each section identifies the security service that is being established or maintained, specifies the security agent, and specifies the relative security context to which the section pertains.

5.1.3.1. Security Services Information Element Format

This specification relies on the Security Services Information Element (SSIE), which is specified in the UNI Signaling 4.0 Security Addendum [4] for octets 1-4. The use of SSIE octets 5, etc. are specified in the following section.

5.1.3.2. Security Association Section

The Security Association Section provides the information needed to establish a single security service association between two ATM security agents.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
Security Association Service Identifier								5 (Note)
Security Association Section Length								5.1 (Note)
Security Association Section Length (cont.)								5.2 (Note)
Version			Transport Ind.		Flow Indicator		Discard	5.3 (Note)
Scope								5.4 (Note)
Scope								5.5 (Note)
Relative ID								5.6 (Note)
Relative ID								5.7 (Note)
Target Security Entity Identifier								5.8* (Note)
Security Service Data								5.9* (Note)

Note: Octet group 5 can be repeated to form new octet groups numbered sequentially as octet groups 6, 7, … N.

When an SSIE is used within Signaling Support for Security Message Exchange, the total length of the SSIE, including all SASs, is limited in [4] and [5].Coding details for these octets are defined below.

5.1.3.2.1. Security Association Service Identifier

This octet identifies the Security Association Service type for this SAS.

Bits								
8	7	6	5	4	3	2	1	Octet
Security Association Service Identifier								
User	Security Association Service Type							5

User Specified Security Service Identifier (User)

8	**Meaning**
0	ATM Forum Specified Security Service
1	User Specified Security Service Type (Note)

Note: If this bit is set (1), then bits 1-7 are encoded with a user-specified codepoint. If this bit is cleared (0) to indicate ATM Forum Security Service, then the following Security Association Service Types apply:

Security Association Service Type

7	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	1	Security Message Exchange
0	0	0	0	0	1	0	Label Based Access Control

[all other codepoints reserved]

5.1.3.2.2. Length of Security Association Section

This 16-bit octet group is the length of this Security Association Section, excluding the Security Service Identifier byte, and the SAS length bytes, bytes, 5.1-5.2. The minimum length of a SAS is 8 bytes, and so the minimum value for this field is 3.

Bits								
8	7	6	5	4	3	2	1	Octet
Length of Security Association Section								5.1
Length of Security Association Section (cont.)								5.2

Length of Security Association Section

This is a 16 bit binary value.

5.1.3.2.3.Version

These 3 bits denote the specification to which this specific SAS type is compliant.

Bits								
8	7	6	5	4	3	2	1	Octet
Version								5.3

Version

8	**7**	**6**	**Meaning**
0	0	0	Version 1.0 Security Service

5.1.3.2.4. Transport Indicator

This field provides the security agents a means of indicating what transport method will be used to exchange the indicated Security Information Exchange Protocol for this security service. The format of this field is indicated below.

Bits								
8	7	6	5	4	3	2	1	Octet
			Transport Ind.					5.3

Transport Indicator

5	**4**	**Meaning**
0	0	Signaling
1	1	In-Band Messaging

The transport method is a negotiable parameter, developed during ATM call establishment. The initiating security agent must specify its preferred method for exchanging security information in the Initial Security Service SAS. Intermediate and responding security agents can modify this field to indicate that the transport method must change in order to accommodate conditions in the network. Intermediate and responding security agents may also modify this field from Signaling to In-Band messaging if they are unable to continue the SME protocol in signaling. Security agents may NOT, however, modify this field from In-Band to Signaling.

5.1.3.2.5. Flow Indicator

This field is used to identify the flow number of the Security Information Exchange for this Security Association Section. The format of this field is indicated below.

Bits								
8	7	6	5	4	3	2	1	Octet
					Flow Ind.			5.3

Flow Indicator

3	**2**	**Meaning**
0	0	First Flow
0	1	Second Flow
1	0	Third Flow
1	1	Fourth Flow.

The Flow Indicator field declares the flow number of the Security Information Exchange Protocol that the SAS is supporting. When the Flow Indicator is zero (0), the SAS represents the initiation of a new security service, and generally indicates to the responding security agent that a new security association is being requested. When signaling is the SME Transport, all SAS's with zero Flow Indicator values must be introduced in the call's SETUP message. Any SAS's with the Flow Indicator set to zero in the CONNECT message will be treated as processing errors.

The Flow Indicator is used by security agents to indicate what fields in the SAS message header should be used for scoping. A zero value for the Flow Indicator indicates that this SAS is initiating a security service.

The Flow Indicator field has relevance to other fields in the SAS. When the Flow Indicator is zero (0), the Scope field is used by receiving security agents to determine if the SAS is relevant to this security agent. A non-zero value for the Flow Indicator indicates that this SAS is part of an ongoing Security Message Exchange Protocol session, and the Relative ID is significant to both peers. As a result, when the Flow Indicator is non zero, security agents will use the Relative ID field as a Security Association ID to indicate that this particular SAS is significant to the receiving security agent.

5.1.3.2.6. Security Association Section Discard Indicator

This indicator specifies whether a security agent will discard the SAS after processing. The format of this field is indicated below.

Bits								
8	7	6	5	4	3	2	1	Octet
							Discard	5.3

Discard Indicator

1	**Meaning**
0	Do Not Discard SAS After Processing.
1	Discard SAS After Processing.

The discard bit is used to specify if the SAS should be discarded after processing. Most SASs will have this bit set (1). Clearing this bit to (0) allows an initiating security agent to pass an SAS to multiple security agents along the VC path.

In this specification, this bit shall be set (1) when this SAS is used for any services other than label-based access control. This bit may be cleared (0) for Label-Based Access Control when it is intended that the label be presented to all appropriate security agents along the VC path.

5.1.3.2.7. Scope

These octets describe the intended scope of the security services association, providing an implicit identification of the responding security agent for which this SAS is relevant. The format of this field is indicated below.

Bits								
8	7	6	5	4	3	2	1	Octet
Scope								
Region								5.4
Explicit	Local	Remote	Transit	Reserved	Reserved	Reserved	Reserved	
Role								5.5
User	EB	N/L	Reserved	Reserved	Reserved	Reserved	Reserved	

In the path of any given VC, a number of security agents may be at work, depending on the security policies of the involved administrative domains. An ATM User may need to establish peer security associations with a number of security agents in the intended VC's path. As an example, the end system may be required to authenticate to its "first hop" switch, authenticate to a remote network security agent, and support confidentiality security services "end-to-end" before a VC can be established between the calling and called parties. In addition, other security services may also be employed along the VC path that are not known to the calling party; specified by local, intermediate, or remote administrative domains. An example of this would be enterprise-to-enterprise encryption.

Because there are no assurances that SSIE SASs are kept in order as signaled messages are propagated through the network, a specific mechanism is needed to indicate which peer security agent is supposed to act on a specific SSIE SAS. To support nested security services, the SAS message header provides both explicit and non-explicit security peer specifications.

5.1.3.2.7.1. Explicit Security Peer Specification

With explicit peer specification, security agents are explicitly referenced by a Security Entity Identifier that is unique along the VC's path. A receiving security agent takes action on an explicitly directed SSIE SAS, only when the Security Entity Identifier references **THIS** particular security agent, or in the case of a proxy security agent, where the proxy is acting on behalf of the security entity explicitly identified in the SSIE SAS.

When explicit peer specification is being used, the **Explicit** bit (8) of the **Region** field is set (1), and bits 7-1 are all zero (0).

The explicit security peer specification mechanism is not designed to provide any assurances with regard to entity authentication. This mechanism is simply intended to provide assistance in transporting the SSIE SAS to its intended security agent. It is expected that if authentication of the peer security agent is required, then the appropriate SSIE SAS type has been employed that provides the mechanisms required assuring peer security agent identification.

5.1.3.2.7.2. Non-Explicit Security Peer Specification

Non-explicit peer specification provides the initiating security agent with a method to specify the Region and the Role of the intended responder. This can be used to direct SSIE SASs to specific agents, in the absence of explicit mechanisms.

The identifiers that are available for specifying security agents are broken into two categories. The first is the Region, which can be Local, Remote or Transit. The second category is **Role** which specifies if the security agent is providing security for the User, an Enterprise Border (EB), or Network or Link (N/L) security for the links internal to a network for some portion other than the enterprise border.

In support of this function, security agents should have a description of the **Region** and **Role** that the security agent plays, relative to other security agents in the network. This may take the form of explicit assignment of **Regions** and **Roles**, along with the required supporting data. **Region** assignments can be satisfied by providing the network address prefix for the local network that a security agent is acting in. **Role** assignments may be derived from the port type that a signaled message was received from. The actual management of security agent **Region** and **Role** descriptions is a vendor implementation issue.

The intent is that if any **Region** or **Role** bit is set, then the initiating security agent is requesting that potential peer security agents should perform the indicated test. In order for an IE to fall within the scope of a particular security agent, both a **Region** test AND a **Role** test must be TRUE. If all **Region** or **Role** bits are cleared (0), which indicates that the initiating security agent does not indicate the tests to be performed, then the respective **Region** or **Role** test returns TRUE. If any **Region** bits are set, then the test returns the resulting OR of each **Region** test indicated. The same operation applies to the **Role** bits and tests.

5.1.3.2.7.2.1. Region

For simplicity the Region semantics are always interpreted relative to an ATM User initiating security agent. This means that the Calling and Called Party IDs are the reference for determining the intended Region of a particular SSIE SAS.

Local

The security agent and the Calling Party are in the same network. When the **Local** bit is set (1) in the **Scope** field, the security agent tests the Calling Party Address against the security agent's local network prefix. If there is a match, then the SSIE SAS falls within the scope of this security agent.

If the test fails, and the **Local** bit is the only **Region** bit set within the SAS, then the Security Agent (SA) has the option, based on its security policy, of declaring the call in error, removing the SAS, or allowing the SAS to remain in the SSIE. If the SA deems that the call is in error, the call is cleared and an appropriate error cause code is returned.

If the Calling Party Address is not available, the test cannot be performed, and as a result, the **Region** test returns TRUE**.**

Remote

The security agent and the Called Party Address are in the same network. When the Remote bit is set (1), in the Scope field, the security agent compares the Called Party Address to the security agent's local network prefix. If they are in the same network then the SSIE SAS falls within the scope of this security agent.

Transit

The security agent is neither in the Calling Party nor Called Party networks. When the Transit bit is set (1) in the Scope field, the security agent compares both the Calling and Called Party Addresses to the security

agent' s local network prefix. If the security agent is not in either network, then the SSIE SAS falls within the scope of this security agent.

5.1.3.2.7.2.2. Role

A Security Agent (SA) can be located at different positions in the network, depending upon the security policy and function that it is performing. SAs can be located at an end user device to protect and/or authenticate the traffic to or from that device. In addition, SAs may be located at the border of a private enterprise protecting or authenticating the traffic entering or leaving the enterprise. Finally, SAs may be located at the border of a public enterprise protecting the traffic entering or leaving the public enterprise. The actual type of interface (e.g., UNI and PNNI) that the SA is on is as relevant to a security policy as the logical location or function that the SA is performing. The **Role** bits are defined as follows:

1. User - SA provides security for the user or end station.
2. Enterprise Border (EB) - SA provides security services at the entrance or exit of an enterprise (public or private).
3. Network/Link (N/L) - SA provides security services for the links internal to a network or for some portion other than the enterprise border.

Examples of the use of the **Role** bit are provided below. All bits are set to zero except where noted:

1. Targeting a security agent at the destination end user device: Remote = 1, User = 1.
2. Policy may require that all calls leaving the local enterprise be authenticated. Targeting a security agent at the local enterprise border: Local = 1, EB = 1.
3. Policy may require that a private enterprise authenticates calls to the public carrier network or that a public carrier authenticates to another public carrier. Targeting a call to the public carrier network: Transit =1, EB =1.
4. An end station may need to request security services from a proxy SA acting on behalf of the user. Targeting a security exchange with the local proxy SA: Local = 1, User = 1.
5. A node may request security services from the next node in the network. Targeting a security exchange with the next node: Local = 1, N/L = 1.
6. The initiating SA may not know which type of responding SA will process the call. In this case: Remote = 1, User =1, EB = 1, N/L = 1.

5.1.3.2.7.3. Relative ID Security Peer Specification

The previous methods are used by the initiating security agent to specify the intended responding security agent at the time of security service initiation, i.e. when the Flow Indicator is zero (0). During SETUP, the security service association is binding to the Security Association ID that is being established. This process is described in detail in the next section.

In the development of a security association, the Security Message Exchange Protocol may be employed to support a multi-flow communication between security agents. The 2nd, 3rd and 4th messages needed to support the 2 Flow, 3 Flow, and In-Band Security Message Exchange have, in this case, non-zero Flow Indicator numbers. With these messages, the Relative ID field is used to identify the appropriate Relative Security Association ID.

When explicit security agent specification is not available, the Relative ID is used exclusively as an identifier for a responding security agent's security association. When non-explicit mechanisms are being employed, for flows other than flow one (1), all non-reserved bits in the Scope field (octet 5.4, bits 5-8, and octet 5.5, bits 6-8) are set to one (1) and all other (reserved) bits in the Scope field are cleared to zero (0).

In conditions where the peer security agent's explicit ID is known, the explicit scoping mechanism can still be used to direct the SSIE SAS to the appropriate security agent. This is indicated in the SSIE SAS where the **Region** field has the **Explicit** bit set (1), bits 7-1 are zero, and an optional Target Security Entity

Identifier is included in the message. A receiving agent uses its explicit scoping methods to determine if it is the intended security agent. Because the Flow Indicator field is non-zero, the receiving agent uses the Relative ID field to identify the specific security association that this SSIE SAS applies to.

5.1.3.2.8. Relative Identifier

The Relative Identifier provides Security Association Identification and explicit Security Association Section ordering. The format of this field is indicated below

Bits								
8	7	6	5	4	3	2	1	Octet
Relative ID								
Security Association ID								5.6
SAS Number				Reserved				5.7

Security Association ID (Octet 5.6.)

This octet represents the Security Association Identifier to which this Security Association Section applies.

SAS Number (Bits 8-5 Octet 5.7.)

This nibble provides precedence order for SASs within an SSIE. This is used to provide explicit ordering of multiple SASs that are involved in the establishment of the same service (have the same Security Association ID).

Reserved (Bits 4-1 Octet 5.7.)

This nibble is reserved for future use and shall be encoded to all zeros (0).

A security service is generally managed through the establishment of a security service association between two security agents. In the event that there are multiple security agents that are active on a single VC, or when a single security agent is providing multiple security services, there is a need to support multiple security associations within a given VC.

The Relative ID is used to identify the security association that a SSIE SAS belongs, and to provide a means of specifying the order in which SSIE SASs will be processed within that association. The procedures developed around the Relative ID provide the mechanisms needed to support multiple nested security services on a single VC.

The Relative ID is divided into two fields, the Security Association ID and the SAS Number. Relative ID numbers are assigned by initiating security agents, and **SHALL NOT** be modified by any other security agents along the VC.

Once established, the Security Association ID is used by each security agent throughout the life of the VC.

As a SETUP message proceeds through the network, when an initiating security agent wants to establish a security association with a responding security agent, it must allocate a Security Association ID for that service. Each SSIE SAS that is used to support this new security association will share the same Security Association ID. With 8 bits available for the Security Association ID, the greatest number of concentric security services available on a given VC is 256.

To allocate a new Security Association ID, the security agent scans all existing SSIE SASs, and notes the largest Security Association ID. If no SSIE SASs exist, then the Security Association ID is zero (0). Otherwise, the new Security Association ID for this service is the greatest Security Association ID encountered plus (+) one (1). This is an 8-bit ADD operation. If, as a result of this ADD operation, the new Security Association ID is zero (0), then a processing error has occurred, and the call must fail, with an appropriate error cause code. An initiating security agent has some freedom as to the allocation of Security

Association ID's. Some conditions exist where the next allocated Security Association ID should be greater than that indicated by the method above, especially where a single security agent is acting as an initiating and responding security agent for multiple security associations.

Receiving security agents process SSIE SASs based on the SASs' 12-bit Relative ID, in descending order. If the initiating security agent requires explicit ordering of security SASs by the responding security agent, then the SASs must be numbered to reflect that ordering precedence. The 4 bit SAS Number field is used for this purpose. The SAS number is 4 bits long. This limits the total number of strictly ordered SSIE SASs per security service to sixteen (16). If order is not a requirement, then the SSIE SASs can share the same SAS number. If ordering is not required then the number of SASs per security service is unlimited. Combinations of ordered and non-ordered SSIE SASs can be accommodated using this convention.

Receiving security agents use scoping to identify those SSIE SASs that are appropriate for this security agent. The receiving agent sorts the "scoped" SASs in descending order, and then processes them based on Security Association ID. Each unique Security Association ID in the collected list of SASs represents a unique security association.

If the security association requires further processing, such as when using the Security Message Exchange Protocol's 2 Flow and 3 Flow methods, the Security Association ID is used in subsequent SSIE SASs to reference the peer security association.

5.1.3.2.9. Target Security Entity Identifier

This optional octet group is the explicit security agent identifier of the target security agent for this specific SAS. This group is included only when the Explicit bit is set and all other bits are zeroed in the Scope field (Octets 5.4 and 5.5). This value can be one of the Security Agent Identification Primitives: Initiator Distinguished Name, Responder Distinguished Name, or Security Agent Identifier, which are defined in Section 6.2.1 of this specification.

5.1.3.2.10. Security Service Data Section

This section defines the format of the Security Service Data Section of the SAS. There are two types of data sections: security message exchange and label-based access control. Each of these data sections is composed of multiple instances of Security Service Data Section Primitives. The Security Service Data Section Primitives are specified in detail in Section 6.1 of this specification.

5.1.3.2.10.1. Security Message Exchange Data

The Security Message Exchange Protocol is the principal protocol used by two peer security agents to communicate security parameters needed to establish a security service. This protocol is specified in detail in Sections 5.1.1.1 and 5.1.1.2 of this specification. When non-signaling based transport methods are indicated, this section is optional. SSIE SAS based responses should indicate if the optional section was parsed, by returning it in the CONNECT message.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
0	0	1	0	0	1	x	x	5.9
						SME Type		
Security Message Exchange Format								
Security Entity Identifier								5.9.1 etc
Security Service Specification Section								5.9.2 etc
Confidentiality Section								5.9.3 etc
Authentication Section								5.9.4 etc

Security Message Exchange Protocol Type (Octet 5.9 bits 2-1)

2	1	Meaning
0	0	Undefined
0	1	Two Way Security Exchange Protocol, Flow 2 is optional
1	0	Two Way Security Exchange Protocol, Flow 2 is required
1	1	Three Way Security Exchange Protocol

Security Entity Identifier (Octet 5.9.1 etc)

This octet group provides the identity of the agents involved in the security exchange. The primitives that can appear in this section are described in detail in Section 6.2.1.

Security Service Specification Section (Octet 5.9.2 etc)

This octet group contains primitives that provide the security negotiation parameters that are used in the Security Message Exchange Protocol, and is described in detail in Sections 6.2.2 and 6.2.3.

Confidentiality Section (Octet 5.9.3 etc)

The Confidentiality Section Primitive is described in detail in Section 6.2.4.

Authentication Section (Octet 5.9.4 etc)

This Authentication Section Primitive is described in detail in Section 6.2.5.

5.1.3.2.10.2. Label Based Access Control

Label Based Access Control is a uni-directional security service where an initiating ATM security agent provides an access control label from which another security agent can perform an ATM level access control decision. Access control labels will generally be limited to a region within which the label has significance. When labels are provided in signaling, any security agent that is involved in the connection SETUP or CONNECT message can make an access control decision based on the contents of the label. Because of this, the label is generally not discarded by the security agent that inspects the label. When labels are passed in-band, the security agents that receive the in-band exchange can enforce the access control policy. The SSIE is used to transport security labels.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
0	0	1	0	1	0	0	0	
Label Identifier								5.9
Label Length								5.9.1
Label Type								5.9.2
Label-Specific Data								5.9.3, etc.

Label Length (Octet 5.9.1)

This octet indicates the length of the Label-Specific Data. It can be any value in the range from 0 to 255.

Label Type (Octet 5.9.2)

0000 0001 FIPS 188

This octet group indicates the security label for the connection in the FIPS 188 format.

Label-Specific Data (Octets 5.9.3, etc)

⇒ FIPS 188 Coding Details [10]

5.1.4. Message Exchange within UNI 4.0 Signaling

This section describes the signaling support for the security message exchange protocols in point-to-point and point-to-multipoint connections.

5.1.4.1. Point-to-Point Connections

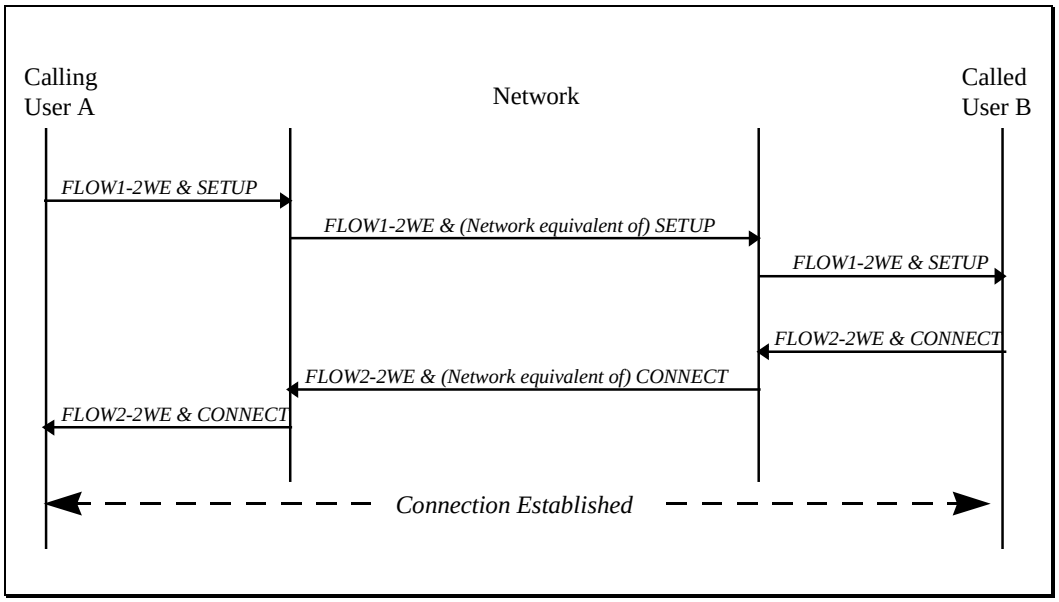


Figure 22. Point-to-Point Signaling Support for Two-Way Security Message Exchange.

Figure 22 shows the signaling support for the two-way security message exchange protocol in point-to-point connections. (Note: signaling support for the three-way security message exchange protocol is not provided in this specification.)

At the UNI interface, FLOW1-2WE is carried in the SETUP message and FLOW2-2WE is carried in the CONNECT message.

At the PNNI and B-ICI interfaces, FLOW1-2WE is carried in the network equivalent of the UNI SETUP message and FLOW2-2WE is carried in the network equivalent of the UNI CONNECT message.

5.1.4.2. Point-to-Multipoint Connections

The signaling procedures for point-to-multipoint call establishment are described in ATM Forum UNI 4.0 [3] signaling and ITU Q.2971 [22].

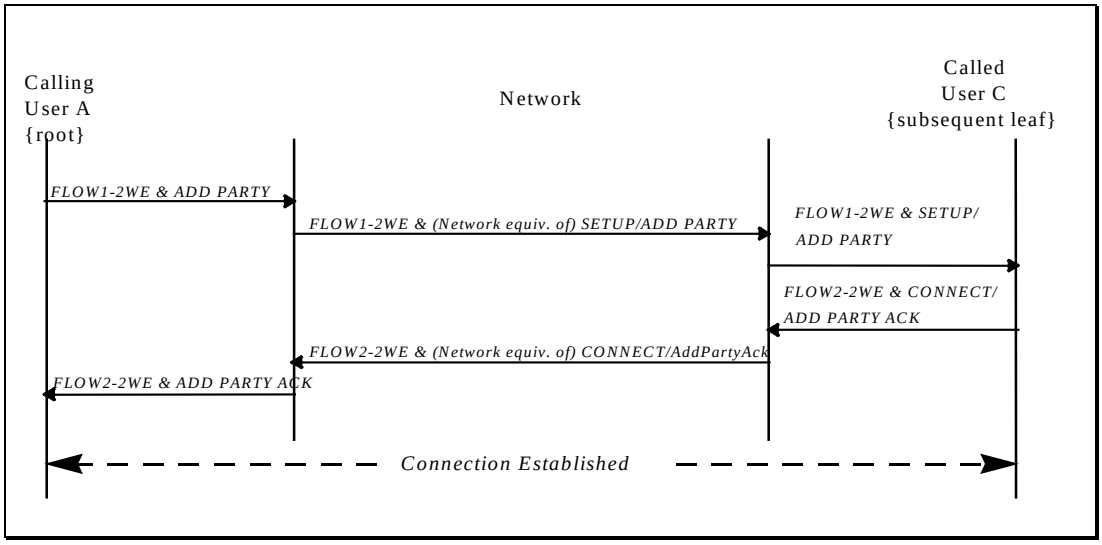


Figure 23. Point to Multipoint Signaling Support for Two-Way Security Message Exchange for Subsequent Leaf Setup.

The connection setup for the first party (leaf) in point-to-multipoint connections is the same as the connection setup in point-to-point connections. So, the two-way security message exchange protocol is supported the same way as in point-to-point connections.

The connection setup for the subsequent parties (leaves) is supported with the two-way security message exchange.

Figure 23 shows the signaling support for the two-way security message exchange protocol when setting up subsequent parties (leaves).

At the UNI interface on the calling side, FLOW1-2WE is carried in the ADD PARTY message and FLOW2-2WE is carried in the ADD PARTY ACK message.

At the UNI interface on the called side, FLOW1-2WE is carried in the SETUP message and FLOW2-2WE is carried in the CONNECT message.

At the PNNI and B-ICI interfaces, FLOW1-2WE is carried in the network equivalent of the UNI ADD PARTY message and FLOW2-2WE is carried in the network equivalent of the UNI CONNECT message.

5.1.4.3. Leaf Initiated Join Capability

The signaling procedures for leaf initiated join capability are described in Section 6.0 of ATM Forum UNI Signaling 4.0.

There are two modes of operation associated with the leaf initiated join capability: root prompted join and leaf prompted join without root notification. The first SETUP message from the root will indicate whether this is the root prompted join or leaf prompted join without root notification.

5.1.4.3.1. Root Prompted Join

In this mode of operation, the root of the connection handles the LIJ request. The root adds leaves to or removes leaves from a new or established connection using the point-to-multipoint procedures. This type of connection is referred to as a root LIJ connection.

In this mode of operation, since the addition (or removal) of a leaf is handled by the root, the point-to-multipoint security procedures described above still apply and no additional enhancement is needed.

5.1.4.3.2. Leaf Prompted Join Without Root Notification

In this mode of operation, if the leaf's request is for an existing connection, the network handles the request. The root is not notified when a leaf is added to or dropped from the connection. This type of connection is referred to as a network LIJ connection.

In this mode of operation, since the addition (or removal) of a leaf is not handled by the root, the current point-to-multipoint security procedures described above do not apply. Support for this mode of operation is outside the scope of this specification.

5.1.4.4. Security Agent Procedures for Signaling-Based Message Exchange

5.1.4.4.1. General Procedures

The general security procedures shall apply for all procedures defined in Section 5.1.4.4.

Refer to Section 1.3 for definition of "Security Agent."

The SA_{sme} shall set the Pass Along indicator [5] to one, so those PNNI nodes that do not implement a SA along the path of the VC will pass the SSIE to the next node. The IE Action Indicator in the IE Instruction Field of the SSIE shall be set to 001 "Discard Information Element and Proceed". It is recommended that Security Agents be capable of handling the status report generated from the value 010 "Discard Information Element, Proceed and Report Status." Loss of the SSIE will be noted in the CONNECT message so that appropriate security policy actions can be invoked.

Security Association Sections (SASs) within the SSIE are processed as "atomic" entities, and so are processed to completion prior to processing of subsequent SASs.

Security agents, when modifying information elements in the signaled message, shall conform to the procedures that govern the format and behavior of the object information elements. Specifically, the relevant procedures described in UNI 4.0 [3], Q.2931 [21], and Q.2971 [22] shall apply.

In the case where errors occur during processing and the security service cannot be performed or established, the security agent shall notify signaling to clear the call, and provide a cause code and diagnostic.

The following procedures describe actions taken by security agents upon receipt of a message containing an SSIE. Version 1 Security Agents ignore SSIEs occurring in any other message.

5.1.4.4.2. Initiating Security Agent Procedures

5.1.4.4.2.1. Call/Connection Request

Upon receipt of a SETUP message by an initiating ATM security agent, all SSIE Security Association Sections that are present in the signaled message are parsed and the greatest Relative ID Security Association ID is noted. If there are no SSIE SASs present, the greatest Relative ID is zero.

Based on the security service that the security agent is to provide, the initiating security agent generates the appropriate Security Association Sections (SAS) for inclusion in the message, providing all mandatory fields that may be required. The initiating SA will ensure that:

1. The Version number for all ATM Forum Version 1.0 Security Services is zero.

2. The transport method is chosen and indicated in the Transport Indicator field, using the guidelines specified in Section 5.1.3.2.4 of this document.

3. The Security Message Exchange State is set to zero. This indicates that this SAS contains information for FLOW1 of the two-way or three-way message exchange protocol.

4. If the SAS is intended for only one security agent, then the **Discard** bit is set to 1. Generally, label based access control services can be intended for more than one target security agent, and so for these services the **Discard** bit may be set to zero.

5. If the security agent intends to use an explicit Target Security Agent Identifier to identify the intended target security agent, the **Explicit** Scope bit is set and an optional **Target Security Entity Identifier** is included in the SAS header.

   If the Target Security Entity Identifier is not used, then an appropriate **Scope** description is selected to identify the **Region** and **Role** of the intended target security agent. Details on the use of this field are specified in Section 5.1.3.2.7.

6. The security agent allocates a new **Security Association ID** value for the security services SAS, as described in Section 5.1.3.2.8. The Security Association ID is noted and is used to identify the corresponding SSIE SAS in the expected CONNECT message.

7. Each Security Association Section is given a **SAS Number** that specifies the precedence order for processing SASs within the same Security Association. SASs will be processed in descending order. The order of processing SASs with the same SAS Number is undetermined.

If an SSIE does not already exist in the SETUP message, an SSIE header is prepended to the SASs that are to be added to the message. The new SASs are then added to the signaled message accordingly.

If any error occurs in processing, the security agent should return the failure condition "Security Protocol Processing Error" (see Section 5.1.6), with an appropriate diagnostic for the condition that generated the error.

At this point, the message is returned to the signaling entity.

5.1.4.4.2.2. Call/Connection Acceptance

Upon receipt of a CONNECT message by an initiating ATM security agent, all SSIE Security Association Sections that are present in the signaled message are parsed and those SASs that have Relative ID Security Association IDs that match those being supported by this security agent are processed. If there are no SSIE SASs present that match any expected Relative ID Security Association IDs, then an error has occurred and the security agent will return the failure condition, "Missing Required Security Information Element" (see Section 5.1.6). If only a partial set of expected Relative ID Security Association IDs is present in the CONNECT message, then the initiating SA shall determine if the call should be in error. This decision is based on the SASs received and the initiating SA's security policy.

If there is no SSIE present then the initiating SA shall determine if the call should be in error. This decision is based on the initiating SA's security policy. If the SASs are unresolved or partially resolved, the initiating SA can revert to in-band message exchange to establish the security service.

The initiating security agent processes all matching SASs, in descending order based on the Relative ID. If an error occurs during processing, the security agent will return the failure condition "Security Protocol Processing Failure."

For each SAS, the initiating security agent processes the contents of the SAS to completion. If the processing completes successfully, then the initiating security agent may perform additional processing, depending on the value of the Transport Indicator field. If the Transport Indicator is Signaling-Based

messaging, then no additional processing is performed. If the Transport Indicator is In-Band Messaging, then the initiating security agent performs the In-Band Security Message Exchange as outlined in Section 5.1.5. If an error occurs during extended SME processing, the security agent will return the failure condition "Security Protocol Processing Failure." Upon successful completion of the extended SME, the initiating security agent processes the next appropriate SAS.

At this point, the message is returned to the signaling entity.

5.1.4.4.3. Responding Security Agent Procedures

5.1.4.4.3.1. Call/Connection Request

Upon receipt of a SETUP message by a responding security agent, if an SSIE is present, all SSIE Security Association Sections that are present in the signaled message are parsed and the SASs that are intended for this security agent are collected. The security agent selects all the appropriate SSIE SASs based on either the explicitly specified **Target Security Entity Identifier** or based on the **Region** and **Role** bits of the SAS **Scope** field, using the procedures described in Section 5.1.3.2.7. From this collection, SASs that have a **Security Service Identifier** that match a security service that can be provided by this security agent are retained. All other SASs are left unmodified. If the SETUP message does not contain a required SSIE or a required SAS, the security agent, based on policy, will either revert to in-band message exchange to establish the security service or clear the call with the failure condition "Missing Required Information Element" (see Section 5.1.6).

The responding security agent processes all matching SASs, in descending order based on the SAS's Relative ID. If an error occurs during processing, the security agent will return the failure condition "Security Protocol Processing Failure."

For each new Security Association ID number encountered in the collection of appropriate SASs, a new security association is allocated and subsequently referenced by Security Association ID number. All SASs with the same Security Association ID are processed within the same security association context, and they are processed in descending order based on SAS number. SASs with identical 8 bit Security Association ID numbers are processed in undetermined order.

Each SAS is processed to completion, before processing can proceed to the next SAS.

Any subsequent flows or responses to processed SASs are held, pending the arrival of the appropriate CONNECT message for this VC.

If the **Discard** bit in the Scope field is set (1), the security agent removes the SAS from the SSIE and adjusts the SSIE length. If no other SASs remain, the SSIE is removed from the signaled message. If the **Discard** bit is zero (0), the SAS is retained, unmodified, in the signaled message.

At this point, the message is returned to the signaling entity.

5.1.4.4.3.2. Call/Connection Acceptance

Upon receipt of a CONNECT message by a responding security agent, all SSIE Security Association Sections that are present in the signaled message are parsed and the greatest Relative ID Security Association ID is noted. If there are no SSIE SASs present, the greatest Relative ID is zero. The security agent determines that the Relative ID of any pending second (2^{nd}) flow or response SASs are not less than the greatest observed Relative ID in the current message. Once this is done, the security agent builds one or more SASs (with the Flow Indicator set to 1 to indicate a FLOW2 message), and adds or appends the pending SASs to the SSIE. If any processing errors occur, the security agent will return an appropriate error condition and the signaling entity will clear the call with the failure condition "Security Protocol Processing Failure" (see Section 5.1.6).

At this point, the message is returned to the signaling entity.

5.1.4.5. Endpoint Requests for Security Services

An endpoint or host that does not provide security services may (as an option) request security services from a downstream security agent as follows:

1. Insert a Security Services Information Element into the SETUP message on the signaling channel.

2. Set the Security Message Exchange Protocol Type field (octet 5.9) to the "undefined" codepoint ("0 0").

3. Insert any combination of the following optional fields, indicating the desired service(s) and/or algorithm(s):

    - Security service declaration,
    - Data confidentiality algorithm,
    - Data integrity algorithm,
    - Hash algorithm,
    - Signature algorithm,
    - Key exchange algorithm,
    - Session key update algorithm,
    - Access control algorithm.

    Multiple algorithms may be inserted in preference order for each requested security service.

Upon receipt of such a request, the security agent replaces the Security Services Information Element with one of its own choosing, based on its security policy. A security agent is under no obligation to use the options requested by the endpoint or host. It should be noted that this process is not secure, and that a trusted link between the endpoint and security agent is assumed.

5.1.4.6. Acknowledgements to Endpoints Requesting Security Services

A security agent shall acknowledge a security request received from an endpoint as follows:

1. Insert a Security Services Information Element into the CONNECT message on the signaling channel.

2. Set the security message exchange protocol to the unspecified codepoint ("0 0").

3. Indicate which service(s) and/or algorithm(s) were negotiated by the security agents, only for those instances where a specific request was made.

The endpoint or host would then have the option of rejecting the call if it did not get the service(s) and/or algorithm(s) it desired.

5.1.5. Message Exchange within the User Plane

This section defines the procedures for performing the security message exchange protocol within the user plane virtual circuit. These procedures apply for VCs that are established via signaling (SVCs), as well as VCs established via management (PVCs).

5.1.5.1. Security Message Exchange for Signaled Point-to-Point Connections

The in-band security message exchange protocol is a method that is suitable for exchanging security messages between ATM entities when signaling is not available either by policy or lack of support. After the user connection is completed successfully, the Security Agents (SAs) exchange security information in-

band over the user traffic channel. (The SAs block the user traffic from passing between the users until after the SAs have completed the security exchange.)

Figure 24 presents point-to-point call setup and security exchange using the hosts' user plane for a Switched Virtual Circuit (SVC) connection.

The ATM calling user transmitting an ordinary SETUP message requesting a point-to-point connection starts the process. The security agent determines the connection is established either by directly processing the signaling messages in a standalone implementation, or through an indication from the signaling function if the SA is hosted in a switch or end system. The method the security agent uses to determine whether a connection requires security services is a local issue and is not specified by this document. This applies to both the initiating and responding security agents. In some environments, the security services will be required for all connections. However, security could be applied selectively to connections between designated hosts. The source and destination ATM addresses could be used as a means of determining to which connections to apply security.

Once the ATM connection is established, the SAs perform certain security-related exchanges before either SA allows user traffic to pass. The security exchange occurs in-band on the user connection using a security exchange protocol and is used to perform such tasks as authentication and key exchange. In the case of a VPC connection request, the security exchange is performed over a well-known VCI within the VPC. This process is described in more detail below.

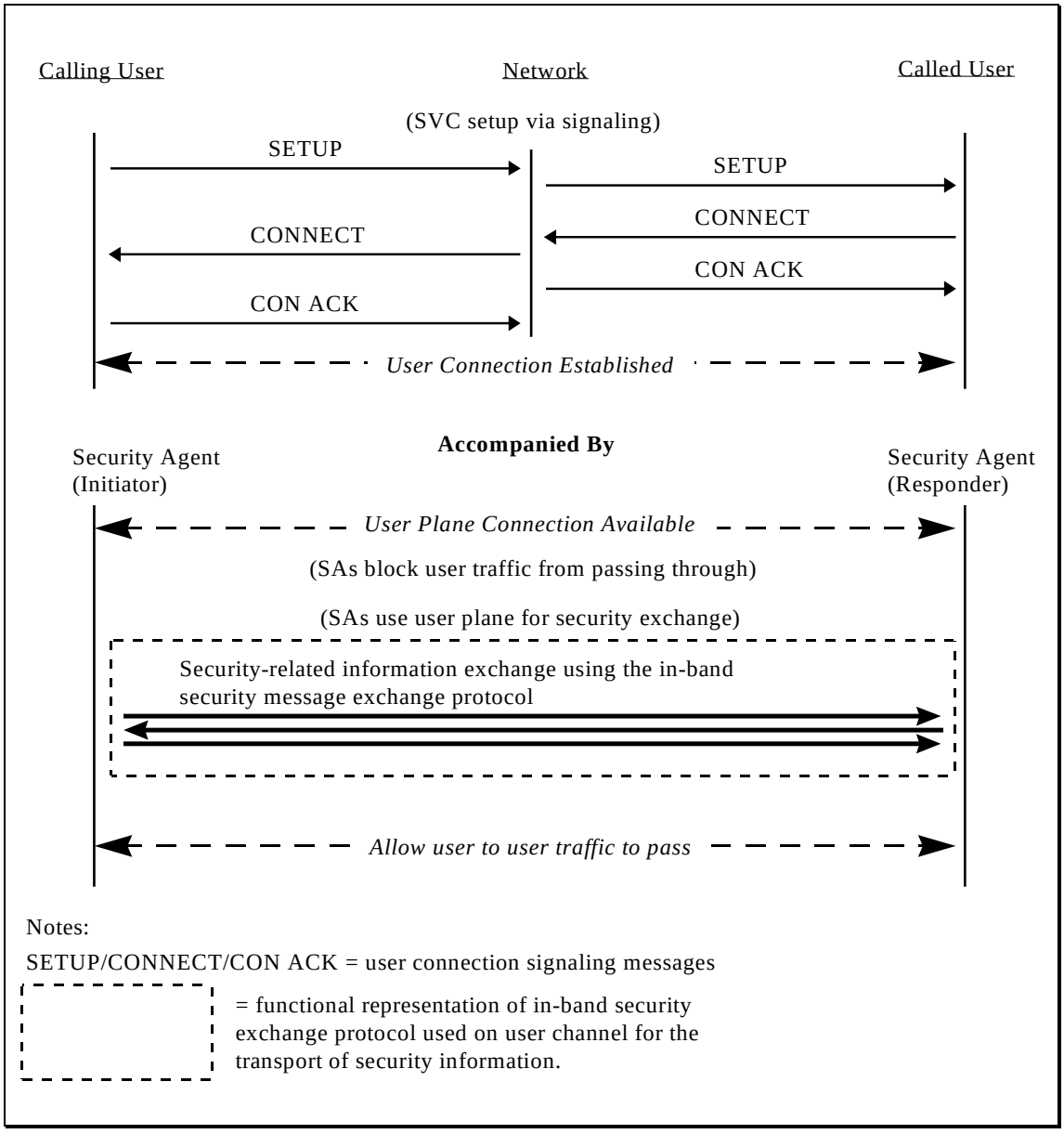


Figure 24. Point-to-Point Security Exchange Using the Hosts' User Plane for SVCs.

Once the in-band user connection is available for use by the SAs, the initiator SA sets up a session with the responder SA. This session is used to exchange security information.

The mechanisms used for security message exchange shall use the Security Message Exchange Protocols described in Section 5.1.5.3 for point-to-point connections. The in-band secure message exchange will only send those information elements that are required for the security exchange (e.g., Security Services IE, etc.), and not the entire signaling message (i.e., SETUP message).

The initiator and responder SAs block user traffic on the specified connection from passing until the SAs' peers have completed their security exchange.

Once the security protocol is complete (e.g., authentication, key exchange), the initiator and responder SAs allow user plane traffic to flow (i.e., "un-block" user traffic).

A SA can be collocated with the end user or at a switch. For example, the internal interface to the SA could be implemented as an application layer interface at the end user device or keyed off of signaling messages received at a switch.

5.1.5.1.1. Details of Security Call Establishment

Before In-Band Security Message Exchange can be invoked, a bi-directional connection between the initiating and responding users must be established. In-Band Security Messages flow over this connection. The underlying connection need not provide assured data delivery. The In-Band Security Message Exchange Protocol described in Section 5.1.5.3 has its own assured message exchange mechanism. The In-Band Security Message Exchange operation is independent of whether the underlying connection provides assured or non-assured data delivery. The underlying connection must allow messages of variable length to be sent. For these reasons, AAL5 shall be used as the data delivery service for in-band security message exchange.

5.1.5.1.1.1. Establish Point-to-Point on Calling Side

The initiating security agent shall support establishment of secure point-to-point Switched Virtual Circuit (SVC) connections initiated from the calling side as follows:

Upon receipt of a SETUP message arriving on the user side, the SA shall examine the traffic parameters contained within the message to determine if the connection can support a full duplex SME.

If the SA determines that the requested connection is point-to-point and can support the necessary SME, the SA shall forward the message toward the called user.

If the SA determines that the user requested point-to-point connection and the cell rates cannot support the necessary SME or the user requested point-to-multipoint connection, the SA shall clear the call with the failure condition “Security Protocol Processing Failure” (see Section 5.1.6)

Note: Alternatives to clearing the call if the connection does not support the bandwidth for in-band SME, such as out-of-band exchange or renegotiation of traffic parameters, are for future study.

Upon receipt of a CONNECT message from the called user accepting the connection (or some other indication) the SA shall invoke the in-band security exchange protocol as specified in Section 5.1.5.3 over the user connection and prevent the CONNECT message (or some internal indication) from being sent to the calling user until the security exchange completes. If the connection is a VPC connection request, the security exchange shall use VPI=*x* (where *x* is the VPI assigned for the VPC) and VCI=1 (i.e., well known VCI).

Note: The SA may elect to test the traffic parameters contained in the CONNECT message to ensure that in-band SME can be supported.

Upon completion of the in-band security exchange protocol indicating that the exchange completed successfully, the initiator SA shall “un-block” the ATM connection, allowing user cells to flow, and forward the CONNECT message over the user side interface. If the security exchange is deemed unsuccessful by the initiator SA, the SA shall request that the ATM connection be cleared. After the connection is cleared, the End User may attempt to retry the connection establishment.

5.1.5.1.1.2. Establish Point to Point on Called Side

The responding SA shall support establishment of secure point-to-point Switched Virtual Circuit (SVC) connections initiated from the called side as follows:

Upon indication that an inbound connection request has been received on the Network side, i.e., a SETUP message, the SA shall relay the SETUP message to the user side and wait for the associated CONNECT message.

The SA shall assure that any traffic received on the user side associated with the connection request will be blocked until the security exchange has completed successfully.

After receipt of an indication that the CONNECT message has been received from the called user accepting the connection, the SA shall allow the CONNECT message to be forwarded to the network side and wait for the in-band security exchange protocol to be invoked over the user connection. If this is a VPC connection request, the SA security exchange shall use VPI=*x* (where *x* is the VPI for the VPC) and VCI=1 (i.e., well known VCI).

Upon completion of the in-band security exchange protocol indicating that the exchange completed successfully, the responder SA shall "un-block" the ATM connection, allowing user cells to flow. If the security exchange is deemed unsuccessful by the responder SA, it shall request that the connection be cleared. After the connection is cleared, the End User may attempt to retry the connection establishment.

5.1.5.2. Security Message Exchange for Permanent Point-to-Point Connections

The basic premise of this approach is to simplify and strengthen "secure" PVC provisioning. After the user connection is completed successfully, the security agents exchange security information in-band over the user traffic channel. (The SAs block the user traffic from passing between the users until after the SAs have completed the security exchange.)

Figure 25 presents point-to-point call setup and security exchange using the hosts' user plane for a PVC connection.

The process is started by an administrative (e.g., Network Manager) provisioning of a point-to-point connection. If the host requires security services, a security agent (SA) must be notified of the connection request. This can also be accomplished administratively. Each SA must also be told if it is the initiator or responder of the security exchange.

The SAs perform certain security-related exchanges before either SA allows user traffic to pass. The security exchange occurs in-band on the user connection using a security exchange protocol and is used to perform such tasks as authentication and key exchange. In the case of a VPC connection request, the security exchange is performed over a well-known VCI within the VPC. This process is described in more detail below.

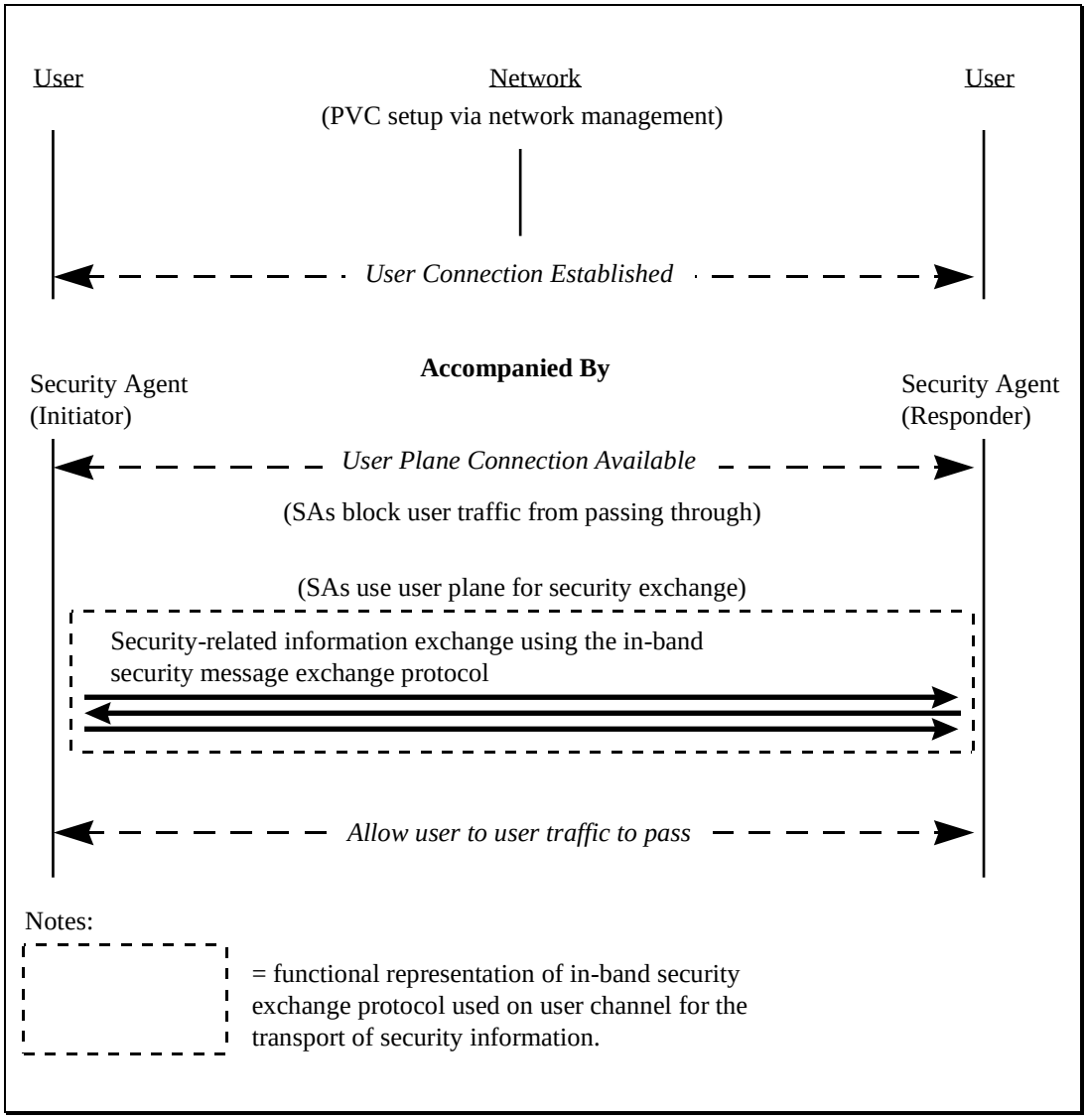


Figure 25. Point-to-Point Security Exchange using the Hosts' User Plane for PVCs.

Once the in-band user connection is available for use by the SAs, the initiator SA sets up a session with the responder SA. This session is used to exchange security information.

The mechanisms used for security message exchange shall use the Security Message Exchange Protocols described in Section 5.1.5.3 for point-to-point connections (the same as used for SVC in-band exchange). The in-band secure message exchange will only send those information elements that are required for the security exchange (e.g., Security Services IE, etc.), and not the entire signaling message (i.e. SETUP message).

The initiator and responder SA block user traffic on the specified connection from passing through it until the SA peers have completed their security exchange.

Once the security exchange is complete (e.g., authentication, key exchange), the initiator and responder SAs allow user plane traffic to flow (i.e., "un-block" user traffic).

A SA can be collocated with the end user or at a switch.

5.1.5.2.1. Details of Secure Call Provisioning for PVCs

Before the In-Band Security Message Exchange can be invoked, a bi-directional connection between the initiating and responding users must be established. In-Band Security Messages flow over this connection. The underlying connection need not provide assured data delivery. The In-Band Security Message Exchange Protocol described in Section 5.1.5.3 has its own assured message exchange mechanism. The In-Band Security Message Exchange operation is independent of whether the underlying connection provides assured or non-assured data delivery. The underlying connection must allow messages of variable length to be sent. For these reasons, AAL5 shall be used as the data delivery service for in-band security message exchange.

5.1.5.2.1.1. Establish Point-to-point—Initiator Side

The initiating security agent shall support establishment of secure point-to-point Permanent Virtual Circuit (PVC) connections (initiator side) as follows:

Upon notification that a connection is requesting security services, the SA shall assure that any traffic received associated with the connection request will be blocked until the security exchange has completed successfully. The SA must also be notified that it is to function as the initiator.

Once the SA has been notified of the secure connection request and its SA function (i.e., Initiator) the SA shall invoke the in-band security exchange protocol over the user connection. If this is a VPC connection request, the security exchange shall use VPI=*x* and VCI=1 (i.e., well known VCI), where *x* is the VPI of the VPC being requested.

If the protocol fails, the SA shall terminate its current attempt to secure the PVC (the user traffic will continue to be blocked).

Upon completion of the in-band security exchange protocol indicating that the exchange completed successfully, the initiator SA shall request that the traffic associated with the connection be “un-blocked.” If the security exchange is deemed unsuccessful by the initiator SA, the SA shall terminate its current attempt to secure the PVC (the user traffic will continue to be blocked).

5.1.5.2.1.2.Establish Point to Point—Responder Side

The responding SA shall support establishment of secure point-to-point Permanent Virtual Circuit (PVC) connections as follows:

Upon notification that a connection is requesting security services, the SA shall assure that any traffic received associated with the connection request will be blocked until the security exchange has completed successfully. The SA must also be notified that it is to function as the responder.

Once the SA has been notified of the secure connection request and its SA function (i.e., Responder) the SA shall wait for the in-band security exchange protocol to be invoked by the Initiator over the user connection and start a PVC Wait timer (P_Wait). If this is a VPC connection request, the SA security exchange shall use VPI=*x* and VCI=1 (i.e., well known VCI), where *x* is the VPI of the VPC being requested. (Note that the use of VCI=1 for in-band security message exchange is temporary, and that VCI=1 is available to other functions once the in-band SME protocol successfully completes.)

If the P_Wait timer expires, or if the protocol fails, the SA shall terminate its current attempt to secure the PVC (the user traffic will continue to be blocked). This timer is canceled at completion of the security exchange, or if the connection is cleared. The value of P_Wait is defined in Section 5.1.5.3.5, Table 7.

Upon completion of in-band security exchange protocol indicating that the exchange completed successfully, the responder SA shall request that the traffic associated with the connection be “un-blocked.”

If the security exchange is deemed unsuccessful by the responder SA, the SA shall terminate its current attempt to secure the PVC (the user traffic will continue to be blocked).

5.1.5.3. In-Band Security Message Exchange Protocol

The three-way security message exchange protocol as described in Section 5.1 is used for in-band messaging. No changes to this protocol are required to operate in-band over the user channel. However, for the in-band operation of this protocol, two additional messages are used: CONFIRM-AP and FAULT.

CONFIRM-AP: The Responder sends this message to the Initiator in reply to a FLOW-3 message. When the Initiator receives CONFIRM-AP, it concludes that the Responder received FLOW-3 and the protocol completed successfully. This message is an integral part of the three-way security message exchange procedure because its purpose is to provide notification to the Initiator (i.e., the calling user) that the Responder received the FLOW3 and the Initiator may begin transmitting data.

FAULT: In error scenarios, this message is used to carry cause data (identifying the error) from the partner that detected the error to the other remote partner.

(The format of these messages is defined below in Section 5.1.5.3.2.)

5.1.5.3.1.Protocol Procedures

The following is a textual description of the in-band security message exchange protocol procedures contained in the finite state machine tables of Section 6.1. If any discrepancies exist between the following text and the FSM tables in Section 6.1, the FSM tables shall take precedence.

Figure 26 shows the normal processing at the Initiator.

AAL5 shall be used for transport of in-band security message exchange protocol messages.

The Initiator starts the message exchange process by sending a FLOW-1 message, starting timer T100, and keeping track of the number of times FLOW-1 has been sent by using variable I-Flow1-Retry-Count. If a FLOW-2 message in reply to the FLOW-1 message is not received before the expiry of T100, the FLOW-1 message is retransmitted and timer T100 restarted.

The Initiator increases its retry counter, I-Flow1-Retry-Count, each time it sends (or resends) the FLOW-1 message. If T100 expires when I-Flow1-Retry-Count is greater than I-MAX-FLOW1-RETRY the message exchange fails.

If the Initiator receives a FLOW-2 message that is invalid, the message exchange fails.

If the Initiator receives a valid FLOW-2 message, it stops timer T100, starts timer T101, and sends a FLOW-3 message to the Responder.

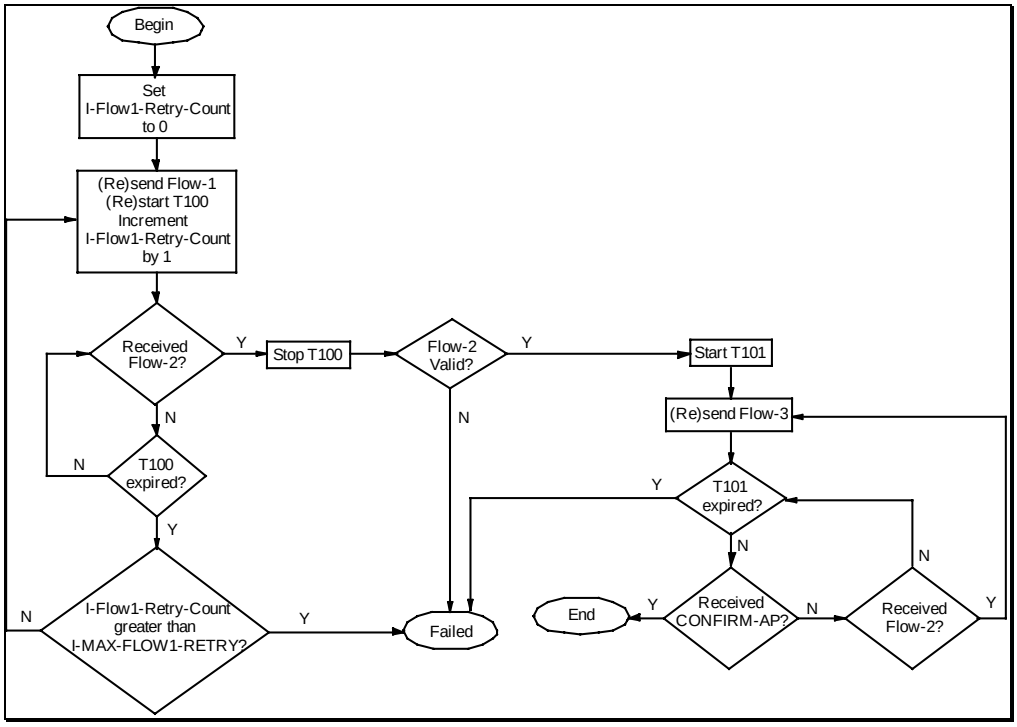


Figure 26. Protocol Procedure at the Initiator.

The Initiator assumes that the message exchange is successful when a CONFIRM-AP message is received. However, if a FLOW-2 message is received before T101 expires or before a CONFIRM-AP message is received, it resends a FLOW-3 message by concluding that the received FLOW-2 message is an indication that the FLOW-3 it sent earlier has not reached the Responder. The Initiator continues this behavior until T101 expires, or a CONFIRM-AP message is received, whichever is earlier.

The Initiator concludes that the protocol completed successfully as soon as a CONFIRM-AP message is received.

The Initiator concludes that the message exchange failed if T101 expires.

Figure 27 shows the normal processing at the Responder.

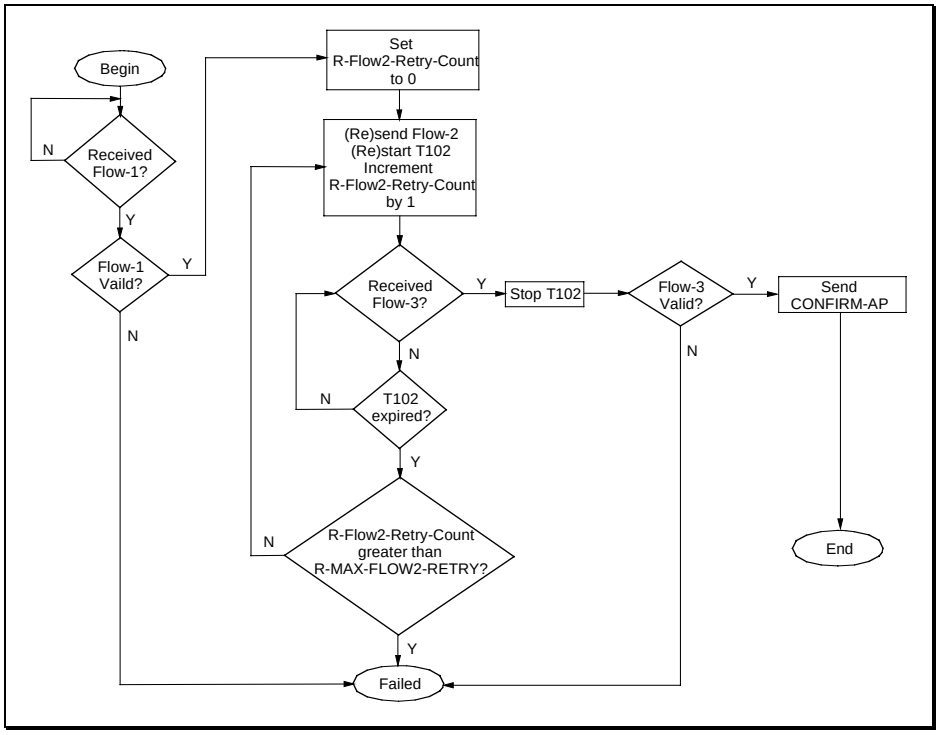


Figure 27. Protocol Procedure at the Responder.

The Responder waits until it receives a FLOW-1 message. If the FLOW-1 message is not valid, the message exchange fails.

If the Responder receives a valid FLOW-1 message, it starts timer T102, sends a FLOW-2 message, and keeps track of the number of times FLOW-2 has been sent in variable R-Flow2-Retry-Count. If a FLOW-3 message in reply to the FLOW-2 message is not received before the expiry of T102, the FLOW-2 message is retransmitted and timer T102 restarted.

The Responder increases its retry counter, R-Flow2-Retry-Count, each time it sends (or resends) the FLOW-2 message. If T102 expires when R-Flow2-Retry-Count is greater than R-MAX-FLOW2-RETRY the message exchange fails.

If the Responder receives a FLOW-3 message that is invalid, the message exchange fails.

If the Responder receives a valid FLOW-3 message, the message exchange is successful. The Responder sends a CONFIRM-AP message to the Initiator so that the Initiator can conclude that the protocol concluded successfully.

5.1.5.3.2.General Message Format and Message Field Coding

Within the In-band Security Message Exchange Protocol, every message shall consist of the following parts:

1. Message type,
2. Message length,
3. Variable length fields, as required.

Fields 1 and 2 are common to all of the messages and shall always be present, while field 3 is specific to each message type. This organization is illustrated in Figure 28.

Message type	1
Message length	2
Message length (continued)	3
Variable length information elements as required	4, etc

Figure 28. General Message Organization.

5.1.5.3.2.1. Message Type

The purpose of the message type is to identify the function of the message being sent. The message type is the first part of every message. The message type is coded as shown in Figure 29.

8	7	6	5	4	3	2	1	Octets
Message Type								1

Figure 29. Message Type.

Message Type (Octet 1)

Bits								Meaning
8	7	6	5	4	3	2	1	
0	0	1	-	-	-	-	-	In-band security message
0	0	1	0	0	0	0	1	Flow1-3WE
0	0	1	0	0	0	1	0	Flow2-3WE
0	0	1	0	0	0	1	1	Flow3-3WE
0	0	1	1	0	0	0	1	CONFIRM-AP
0	0	1	1	0	0	1	0	FAULT

Note - Although these message types are for in-band messages only, the codepoints were chosen so as not to overlap with the signaling codepoints for message type.

5.1.5.3.2.2. Message Length

The purpose of the message length is to identify the length of the contents of a message. It is the binary coding of the number of octets of the message contents, excluding the octets used for "message type" and for the message length indicator itself.

If the message length contains no further octets, the message length is coded to all "0's".

The message length is the second field of every message. The message length is a 16-bit value, coded as shown in Figure 30.

8	7	6	5	4	3	2	1	Octets
Message length								1
Message length (continued)								2

Figure 30. Message Length.

5.1.5.3.2.3. Variable Length Information Elements

5.1.5.3.2.3.1. Security Services Information Element

The Security Services Information Element (SSIE) is defined in Section 5.1.2 of this specification. When in-band messaging is used, the SME Flow Indicator in the SSIE, defined in section 5.1.3.2.5, shall be ignored.

5.1.5.3.2.3.2. Cause Information Element

The Cause information element format defined in Q.2931[21] shall be used for transporting cause codes within the context of the In-Band Message Exchange Protocol. Refer to Section 5.1.5.3.6 for cause values.

5.1.5.3.3. Message Contents for In-band SME Messages

5.1.5.3.3.1. Flow 1-3WE

This message is sent from the initiator to the responder to initiate the three-way security message exchange.

Message Type: Flow 1-3WE

Field or Information Element	Reference	Type	Length
Message Type	Section 5.1.5.3.2.1	M	1
Message Length	Section 5.1.5.3.2.2	M	2
Security Services Information Element	Section 5.1.2	M	12 - *n*

5.1.5.3.3.2. Flow 2-3WE

This message is sent from the responder to the initiator in response to Flow1-3WE.

Message Type: Flow 2-3WE

Field or Information Element	Reference	Type	Length
Message Type	Section 5.1.5.3.2.1	M	1
Message Length	Section 5.1.5.3.2.2	M	2
Security Services Information Element	Section 5.1.2	M	12 - *n*

5.1.5.3.3.3. Flow 3-3WE

This message is sent from the initiator to the responder in response to Flow2-3WE.

Message Type: Flow 3-3WE

Field or Information Element	Reference	Type	Length
Message Type	Section 5.1.5.3.2.1	M	1
Message Length	Section 5.1.5.3.2.2	M	2
Security Services Information Element	Section 5.1.2	M	12 - *n*

5.1.5.3.3.4. CONFIRM-AP

This message is sent from the responder to the initiator in response to Flow3-3WE. When the initiator receives the CONFIRM-AP message, it concludes that the responder received FLOW3-3WE and the protocol completed successfully. This message is an integral part of the three-way security message exchange procedure because its purpose is to provide notification to the initiator (i.e., calling user) that the responder received the Flow3 and the initiator may begin transmitting data.

Message Type: CONFIRM-AP

Field or Information Element	Reference	Type	Length
Message Type	Section 5.1.5.3.2.1	M	1
Message Length	Section 5.1.5.3.2.2	M	2

5.1.5.3.3.5. Fault

In error scenarios, this message is used to carry cause data (identifying the error) from the partner that detected the error to the other remote partner.

Message Type: FAULT

Field or Information Element	Reference	Type	Length
Message Type	Section 5.1.5.3.2.1	M	1
Message Length	Section 5.1.5.3.2.2	M	2
Cause	Refer to Q.2931 [21]	M	4 - 34

5.1.5.3.4. Timer Definitions

The following is a description of the timers that are used for the in-band operation of the three-way security message exchange protocol. The values of these timers can be found in Section 5.1.5.3.5, Table 7.

1. T100: This timer is used by the Initiator of the protocol to determine whether it needs to resend FLOW-1 to the Responder. The timer is started when the Initiator sends FLOW-1. If the Initiator has not received FLOW-2 before the timer expires, then the Initiator resends FLOW-1 and starts the timer again. The timer is stopped when a FLOW-2 message is received.

2. T101: This timer is used by the Initiator of the protocol to ensure that it is waiting as long as the Responder is present and is potentially resending FLOW-2 to indicate that the partner has not received FLOW-3 from the Initiator. The timer is started the first time the Initiator sends FLOW-3. As long as this timer has not expired, the receipt of FLOW-2 triggers the sending of FLOW-3.

3. T102: This timer is used by the Responder in the protocol to determine whether it needs to resend FLOW-2 to the Initiator. The timer is started when the Responder sends FLOW-2. If the Responder has not received FLOW-3 before the timer expires, then the Responder resends FLOW-2 to the Initiator and starts the timer again. The timer is stopped when a FLOW-3 message is received.

The following variables (retry counters) are used in conjunction with the timers used in the protocol.

1. I-Flow1-Retry-Count: This variable is used in conjunction with timer T100 by the Initiator of the protocol and counts the number of times that FLOW-1 has been sent. FLOW-1 may be sent up to a maximum of I-MAX-FLOW1-RETRY times.

2. R-Flow2-Retry-Count: This variable is used in conjunction with timer T102 by the Responder in the protocol and counts the number of times that FLOW-2 has been sent. FLOW-2 may be sent up to a maximum of R-MAX-FLOW2-RETRY times.

The following is a description of the constants that are used in conjunction with the retry counter variables used in the protocol. The values of these constants can be found in Section 5.1.5.3.5, Table 8.

1. I-MAX-FLOW1-RETRY: This constant indicates the maximum number of times that the Initiator may try to resend FLOW-1 to the Responder.

2. R-MAX-FLOW2-RETRY: This constant indicates the maximum number of times that the Responder may try to resend FLOW-2 to the Initiator.

5.1.5.3.5. Timer Values

The protocols for in-band (user plane-based) security message exchange use a number of timers in their procedures. These timers are summarized in the following table:

Table 7: Timers for In-Band Security Message Exchange

Timer Name	Timer Value
P_Wait	600 seconds
T100	3 seconds
T101	9 seconds
T102	3 seconds

In addition, the in-band security message exchange protocols use the following constant definitions:

Table 8: Constant Values for In-Band Security Message Exchange

Constant Name	Constant Value
I_MAX_FLOW1_RETRY	2
R_MAX_FLOW2_RETRY	2

5.1.5.3.6. Protocol Error Handling

Detailed error handling procedures are implementation dependent. However, capabilities facilitating the orderly treatment of error conditions provided for in this section shall be provided in each implementation.

When the Initiator or Responder detects an error, it sends a FAULT message to its partner to report the error, enters the failed state, and aborts. All errors are defined to be unrecoverable errors. The protocol aborts upon sending or receiving a FAULT message.

For SVC-initiated calls in the failed state, the security agent shall clear the call. Error recovery for PVC initiated calls is implementation specific.

A "Cause" information element describes the reason for an error and provides diagnostic information. This information element is carried in the FAULT message. The Cause information element may be repeated in a FAULT message.

The following cause codes are defined.

In-Band Message Exchange Cause Codes	
Number	**Meaning**
111	protocol error, unspecified
100	invalid information element contents
102	recovery on timer expiry
63	service or option not available
97	message type non-existent or not implemented
96	mandatory information element missing
47	resource unavailable
Note 1	*security services failure* (Note 2)

Note 1:

The value of the "Security Services Failure" cause code is defined in the UNI Signaling 4.0 Security Addendum [4].

Note 2:

The specific security protocol failure (e.g., Authentication failure) shall be included in the Diagnostics field of the Cause IE when the cause value = "security services failure."

5.1.6. Security Information Exchange Error Processing

The following text applies when security message exchange occurs in UNI 4.0 signaling, in-band, or both.

When errors occur in establishing or maintaining a security service on a specific VC, the VC must be put into a state where User Data transport is not supported. For establishing or established VCs this is accomplished by clearing the call. When clearing a call based on failure of a security agent to establish or maintain a security service, an appropriate Cause code will be chosen by the security agent that is involved with the fault, and a diagnostic is provided that provides adequate information for the peer security agent to understand the nature of the fault condition.

Security diagnostics are specific to the types of error conditions that can happen within a security agent and are limited to 28 bytes in length. The amount of information that is disclosed in a security agent diagnostic may have a security impact, and as such each diagnostic has a mandatory section and optional sections that provide more detail on the condition that generated the error.

1. Missing Required Security Information Element.

   When this error condition occurs, the call is cleared with Cause code # 96 "Mandatory Information Element Missing" (see Q.2931 [21]), and the diagnostic will be at most 28 bytes of a valid SSIE.

A 4-byte SSIE header is mandatory. Optionally, the diagnostic can contain valid SAS segments that will declare to the originator the type of security service that is required by the security agent. When providing SASs, the security agent can use existing fields to provide additional information, such as the security agent's region and role bits.

2. Security Protocol Processing Error.

   When this error condition occurs, the call is cleared with Cause code "Security Services Failure" (see the UNI Signaling 4.0 Security Addendum [4] for the value of this cause code, and see Q.2931 [21] for the Cause Information Element format). The diagnostic will be at most 28 bytes of a valid SSIE. A 4-byte SSIE header is mandatory. Optionally, the diagnostic can contain valid SAS segments which will indicate which SAS was being acted on when the processing error occurred. The SAS may include the Relative ID of the SAS that generated the error.

3. Security Policy Violation

   When this error condition occurs, the call is cleared with Cause code " Security Services Failure" (see the UNI Signaling 4.0 Security Addendum [4] for the value of this cause code, and see Q.2931 [21] for the Cause Information Element format). The diagnostic will be at most 28 bytes of a valid SSIE. This diagnostic should contain valid SAS segments which indicate the service that is not allowable based on the security policy that was violated. The SAS may include the Relative ID of the SAS that generated the error.

   Reporting this specific error condition is optional, as local policy may want to minimize the amount of information disclosed when a security error occurs. When reporting this specific error condition is not possible, a general "Security Protocol Processing Error" should be reported.

5.1.7. Security OAM Cells

5.1.7.1. Overview of ATM Layer OAM Cell Flows

At the ATM layer, two OAM flows are defined: F4 (virtual path level) and F5 (virtual channel level), which are used for the operation and maintenance of connections. F4 flow is for virtual path connections (VPCs) and F5 flow is for virtual channel connections (VCCs). F4 and F5 flows are for all types of ATM connections. At the ATM layer, the F4 and F5 flows are carried via OAM cells. F4 and F5 flows are bi-directional. In all cases, OAM cell flows must conform to the traffic contract for their respective VCC/VPCs. A detailed description on OAM flows and functions is given in [20].

Note: These mechanisms rely on sufficient bandwidth to sustain OAM cell flows. If sufficient bandwidth is not provided, loss or corruption of user data can occur, and errors may arise in the security mechanism itself.

Table 9: OAM Cell Type and Function Identifiers.

OAM Cell Type	Codepoint	Function Type	Codepoint
Fault management	0001	AIS	0000
		RDI	0001
		Continuity check	0100
		Loopback	1000
Performance management	0010	Forward monitoring	0000
		Backward monitoring	0001
		Monitoring & reporting	0010
Activation/Deactivation	1000	Performance monitoring	0000
		Continuity check	0001
System management	1111	Security – non-real-time	0001
		Security – real-time	0010

The OAM cell formats for ATM layer OAM flows are shown in Figure 31. The OAM cell type (4-bit) field indicates the type of management function performed by the OAM cell. Currently, four OAM types are defined: fault management, performance management, activation/deactivation, and system management. The function type (4-bit) field indicates the function performed by the OAM cell within the management function. Table 9 shows the codepoints assigned for the OAM cell type field and the function type field. The function specific field (45 bytes) carries the information related to the actual function of the OAM flow. The reserved (6-bit) field is not used and left for future use. The error detection code (EDC) field carries a CRC-10 error detection code computed over the cell data payload.

5.1.7.1.1. F4 Flow of OAM Cells

The virtual path connection (VPC) operational information is carried via the F4 flow OAM cells. There are two types of F4 flows: end-to-end F4 flow and segment F4 flow. End-to-end F4 flows are terminated at the endpoints of a VPC and segment F4 flows are terminated at the connecting points terminating a VPC segment. A VPC segment is defined as either a single VPC link or a concatenation of multiple VPC links. Multiple VPC segments can be defined along a VPC path but they cannot overlap.

End-to-end and segment F4 flow OAM cells are passed unmodified by all intermediate nodes. Intermediate nodes can insert new F4 flow OAM cells, however, the end-to-end F4 flow OAM cells are removed only by the endpoints of the VPC and the segment F4 flow OAM cells are removed only by the VPC segment terminating points.

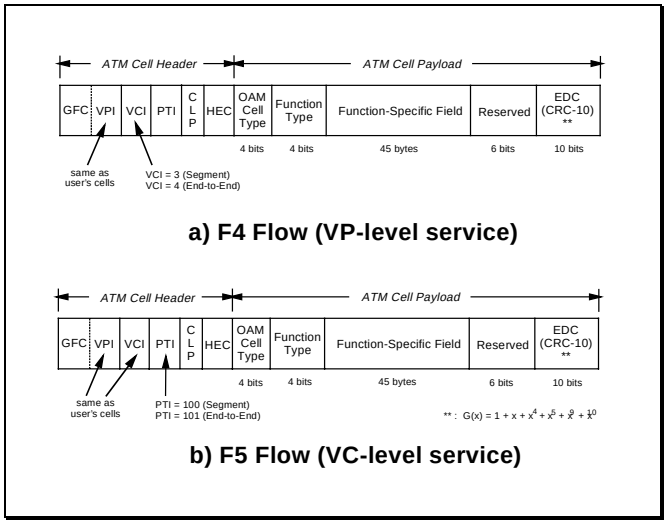


Figure 31. F4 and F5 OAM Cells.

F4 flow OAM cells follow exactly the same physical route as the user's data cells of the VPC. These OAM cells have the same VPI value as the user's data cells of the VPC and are identified by pre-assigned VCI values. The same pre-assigned VCI values are used for both directions of the F4 flows. Two unique VCI values are used for every VPC as shown in Figure 31. The VCI value = 4 is used to identify end-to-end F4 flows and the VCI value = 3 is used to identify segment F4 flows.

5.1.7.1.2. F5 Flow of OAM Cells

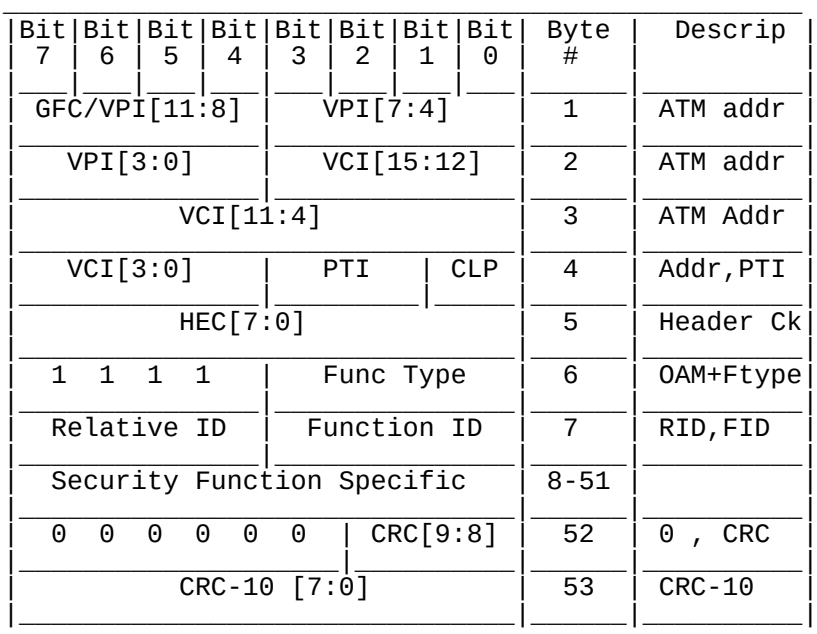
The virtual channel connection (VCC) operational information is carried via the F5 flow OAM cells. There are two types of F5 flows: end-to-end F5 flow and segment F5 flow. End-to-end F5 flows are terminated at the endpoints of a VCC and segment F5 flows are terminated at the connecting points terminating a VCC segment. A VCC segment is defined as either a single VCC link or a concatenation of multiple VCC links. Multiple VCC segments can be defined along a VCC path but they cannot overlap.

End-to-end and segment F5 flow OAM cells are passed unmodified by all intermediate nodes. Intermediate nodes can insert new F5 flow OAM cells, however, the end-to-end F5 flow OAM cells are removed only by the endpoints of the VCC and the segment F5 flow OAM cells are removed only by the VCC segment terminating points.

F5 flow OAM cells follow exactly the same physical route as the user's data cells of the VCC. These OAM cells have the same VPI/VCI values as the user's data cells of the VCC and are identified by pre-assigned codepoints of the payload type identifier (PTI). The same pre-assigned VCI values are used for both directions of the F5 flows. Two unique PTI values are used for every VCC as shown in Figure 31. The PTI value = 5 (101) is used to identify end-to-end F5 flows and the PTI value = 4 (100) is used to identify segment F5 flows.

5.1.7.2. Security OAM Cell Formats

Two Security OAM cell types are defined—one for non-time-critical processes (called non-real-time), and one for time-critical processes (called real-time). The distinction is to differentiate cells that can be processed independently of the user traffic from those that have direct impact on user traffic. The SKE and SKC cells, defined in sections 5.1.7.2.1.1 and 5.1.7.2.2.1, are examples of non-real-time and real-time security OAM cells, respectively. The generic format for security OAM cells is shown in Figure 32.



Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Byte #	Descrip
GFC/VPI[11:8]				VPI[7:4]				1	ATM addr
VPI[3:0]				VCI[15:12]				2	ATM addr
VCI[11:4]								3	ATM Addr
VCI[3:0]				PTI			CLP	4	Addr,PTI
HEC[7:0]								5	Header Ck
1	1	1	1	Func Type				6	OAM+Ftype
Relative ID				Function ID				7	RID,FID
Security Function Specific								8-51	
0	0	0	0	0	0	CRC[9:8]		52	0 , CRC
CRC-10 [7:0]								53	CRC-10

Figure 32. Generic Security OAM Cell Format.

Notes:

1) For VCCs the PTI field is set to 101 (binary).
2) For VPCs the VCI field is set to 0004 (hex).
3) The CLP bit is always set to 0.
4) For non-real-time security OAM cells, the Func Type field is set to 0001 (binary).
5) For real-time security OAM cells, the Func Type field is set to 0010 (binary).
6) The use of the Relative ID field is defined in Section 5.1.7.3.
7) The use of the Function ID field is described in Table 10 and Table 11.

5.1.7.2.1. Non-Real-Time Security OAM Cell Formats

Non-Real-Time (NRT) security OAM cells are defined as having an OAM function type of 0001 (binary). Up to 16 NRT cell types are possible, as defined by the Function ID field. The code points for the Function ID field for NRT security cells are defined in Table 10.

Table 10: Function ID Code Points for Non-Real-Time Security OAM Cells.

Function ID (binary)	Security Function
0001	Data Confidentiality Session Key Exchange (SKE)
0010	Data Integrity Session Key Exchange (SKE)
all others	not defined

5.1.7.2.1.1. Data Confidentiality SKE OAM Cell Format

The format of the Data Confidentiality SKE OAM Cell is defined in Figure 33.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Byte #	Descrip
GFC/VPI[11:8]				VPI[7:4]				1	ATM addr
VPI[3:0]				VCI[15:12]				2	ATM addr
VCI[11:4]								3	ATM Addr
VCI[3:0]				PTI			CLP	4	Addr,PTI
HEC[7:0]								5	Header Ck
1	1	1	1	0	0	0	1	6	OAM+Ftype
Relative ID				0	0	0	1	7	RID,FID
Bank ID				Reserved				8	BID,RES
Reserved								9	Reserved
Key Number								10-13	KN
Encrypted Session Key (ESK)								14-45	ESK
Reserved								46-51	Reserved
0	0	0	0	0	0	CRC[9:8]		52	0 , CRC
CRC-10 [7:0]								53	CRC-10

Figure 33. Session Key Exchange (SKE) OAM Cell Format.

Notes:

1) The use of the Relative ID field is defined in Section 5.1.7.3.

2) The Bank ID field is an alternating pattern of 0 hex or F hex, for successive key updates.

3) The key number (KN) field, a 32-bit field, indicates the key number associated with the new session key. Each session key is assigned a number. The first session key to be exchanged by the session key update protocol is assigned 1; the second session key is assigned 2, and so forth. The KN is a cryptographically protected field which is used by the key update protocol to ensure freshness, uniqueness, and ordering of session key updates and to synchronize the initiator and responder to the same session key.

4) The Encrypted Session Key (ESK), a 256-bit field, contains the next session key encrypted using the master key for the connection. For cases when the session key being transported is less than 256 bits in length, it is contained in the least significant bits of the ESK field, with the most significant bits being padded with zeros (before encryption).

5) Reserved bytes are set to 6A hex, reserved bit fields less than 1 byte in length are set to all zeros.

6) The reserved bits included after the 4-bit Bank ID field are provided so that the KN and ESK fields are aligned on 16 bit boundaries, to simplify high speed implementations.

5.1.7.2.1.2.Data Integrity SKE OAM Cell Format

The format of the Data Integrity SKE OAM Cell is defined in Figure 33, except that the Function ID field is set to 0010 (binary).

5.1.7.2.2. Real Time Security OAM Cell Formats

Real Time (RT) security OAM cells are defined as having an OAM function type of 0010 (binary). Up to 16 RT cell types are possible, as defined by the Function ID field. The code points for the Function ID field for RT security cells are defined in **Table 11**.

Table 11: Function ID Code Points for Real Time Security OAM Cells.

Function ID (binary)	Security Function
0001	Data Confidentiality Session Key Changeover (SKC)
0010	Data Integrity Session Key Changeover (SKC)
all others	not defined

5.1.7.2.2.1.Data Confidentiality SKC OAM Cell Format

The format of the Data Confidentiality SKC OAM Cell is defined in Figure 34.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Byte #	Descrip
GFC/VPI[11:8]				VPI[7:4]				1	ATM addr
VPI[3:0]				VCI[15:12]				2	ATM addr
VCI[11:4]								3	ATM Addr
VCI[3:0]				PTI			CLP	4	Addr,PTI
HEC[7:0]								5	Header Ck
1	1	1	1	0	0	1	0	6	OAM+F type
Relative ID				0	0	0	1	7	RID,FID
Bank ID				Reserved				8	BID,RES
Reserved								9	Reserved
Key Number								10-13	KN
State Vector (SV)								14-21	SV
Reserved								22-51	Reserved
0	0	0	0	0	0	CRC[9:8]		52	0 , CRC
CRC-10 [7:0]								53	CRC-10

Figure 34. Session Key Changeover (SKC) OAM Cell Format.

Notes:

1) The use of the Relative ID field is defined in Section 5.1.7.3.

2) The Bank ID field is an alternating pattern of 0 hex or F hex, for successive key updates.

3) The key number (KN) field, a 32-bit field, indicates the key number associated with the new session key. Each session key is assigned a number. The first session key to be exchanged by the session key update protocol is assigned 1; the second session key is assigned 2, and so forth.

4) The State Vector (SV) contains the new state vector when using the counter mode of the Data Confidentiality service, or it is set to all zeros otherwise.

5) Reserved bytes are set to 6A hex, reserved bit fields less than 1 byte in length are set to all zeros.

6) The reserved bits included after the 4-bit Bank ID field are provided so that the KN and SV fields are aligned on 16 bit boundaries, to simplify high speed implementations.

5.1.7.2.2.2. Data Integrity SKC OAM Cell Format

The format of the Data Integrity SKC OAM Cell is defined in Figure 34, except that the Function ID field is set to 0010 (binary).

5.1.7.3. Use of the Security OAM Cell Relative ID Field

The security OAM cell's relative ID (or RID) field is used to support symmetric nesting of security agents and/or security services. Security OAM cells provide a means of communication between security agents throughout the lifetime of a connection. The relative ID field provides the mechanism to indicate which security agent a particular OAM cell applies to, when nesting is employed. Up to 16 layers of nesting are supported.

There are several potential uses for nested security agents. The primary one is to provide different security "domains" within a network, where an inner domain might provide different security characteristics than an outer domain. By allowing nesting, users in the outer domain can share the same network as the inner domain users. Nesting can also be used to combine security functions that happen to use separate security agents (e.g., privacy and integrity). Figure 35 shows an example of nested security agents and the effect on the relative id (RID) field.

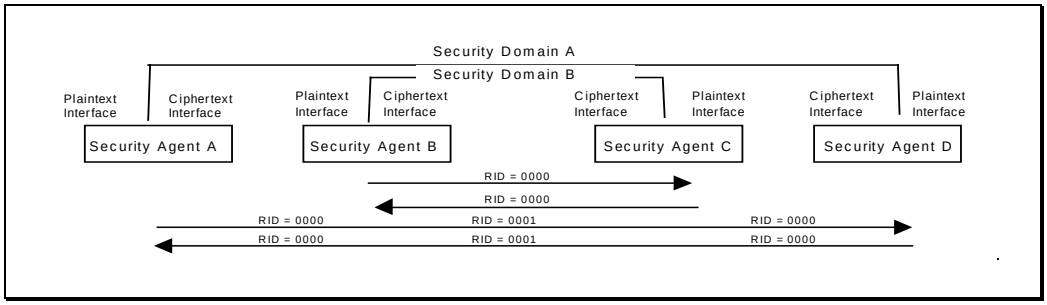


Figure 35. Example of Nested Security Agents.

The procedures for processing security OAM cells are described in Sections 5.1.7.3.1, 5.1.7.3.2, and 5.1.7.3.3. These procedures apply only to those security agents that insert and/or terminate security OAM cells for purposes of key update or cryptographic resynchronization. Security agents not performing these functions shall pass security OAM cells transparently.

Implementations containing multiple security services in a single physical device, e.g. integrity and encryption, shall treat each service as a separate logical entity with regards to RID processing.

A security agent contains two interfaces: plaintext and ciphertext. The plaintext interface is the interface over which the "unprotected" traffic is transmitted or received. A security agent receives "unprotected" ATM traffic on its plaintext interface, applies security services to the traffic, and forwards the "protected" traffic via the ciphertext interface. A security agent receives protected ATM traffic on its ciphertext interface, removes the security protection and forwards the "unprotected" traffic over its plaintext interface. Security OAM cells flow in both the plaintext-to-ciphertext direction and ciphertext-to-plaintext direction. The procedures for processing the relative ID field depend on the direction of the Security OAM cell flow.

5.1.7.3.1. Initiation of Security OAM Cells

The following procedures apply to source security agents that insert security OAM cells.

Security OAM cells are introduced by security agents through the ciphertext interface. When a security OAM cell is injected into the cell stream by a security agent (for cryptographic resynchronization or key update purposes), the relative ID shall be set to zero.

5.1.7.3.2.Processing for Security OAM Cells Received on the Plaintext Interface

Security OAM cells will be received on the plaintext side of a security agent if there are two or more nested pairs of security agents providing confidentiality or integrity services on a given connection. A security agent which receives a Security OAM cell on its plaintext interface, with the Function ID matching the service the source SA is providing, shall increment the relative ID by one, compute a new CRC-10 over the cell contents, circumvent the security function, and forward the cell to the ciphertext interface. The relative order of the security OAM cell and other cells on the connection shall not be perturbed. If the relative ID field of a received security OAM cell contains all ones, the cell is discarded.

5.1.7.3.3.Processing for Security OAM Cells Received on the Ciphertext Interface

A Security OAM cell received on the ciphertext interface is intended for the security agent if the relative ID field is set to zero. If the relative ID is set to zero the security agent shall remove the security OAM cell from the cell stream and use its contents to perform the security OAM function. If the relative ID is set to a value other than zero, the security agent shall decrement the relative ID, compute a new CRC-10 over the cell contents, circumvent the security function, and forward the cell through the plaintext interface. The relative order of the security OAM cell and other cells on the connection shall not be perturbed.

5.2. Key Exchange

5.2.1. Key Exchange Infrastructure and Mechanisms

The ATM key exchange protocol is described in the context of "security message exchange protocol" in Section 5.1. Key exchange is performed by generating initial session keys for each direction in the VC and a shared master key. These keys are encrypted for exchange using an Initial Key Exchange Key. When using a public key encryption algorithm such as RSA for key exchange, the Initial Key Exchange Key is the public key of the recipient. When using a public key development algorithm such as Diffie-Hellman, the Initial Key Exchange Key is a portion of the shared value developed by the algorithm from the users' public keys. When using a symmetric key algorithm, the Initial Key Exchange Key is a shared secret key established prior to key exchange.

5.2.2. Key Exchange Algorithms

Key exchange is performed using either symmetric (secret) key algorithms such as DES or asymmetric (public) key algorithms such as RSA. If a symmetric key algorithm is used, the two nodes must have already established a shared secret key. With asymmetric algorithms, on the other hand, each node need only know the public key of the other node.

Note that the generation, distribution, and storage of these (usually long-lived) secret, private, and/or public keys is outside the scope of this specification.

Associated with each algorithm is a codepoint that is used for identification when negotiating security options. These codepoints are listed in Section 6.1. Procedures for using user-defined algorithm codepoints are also described in Section 6.1.

The following apply when performing key exchange:

- When performing user plane key exchange, one of the authentication and key exchange protocols described in Section 5.1 shall be used.
- When performing key exchange, an authentication algorithm (either user-defined, or listed in Section 3.1) shall be used.

5.2.2.1. Asymmetric Algorithms for Key Exchange

The following asymmetric (public-key) algorithms are defined in this specification for key exchange.

Algorithm	Normative Reference
RSA	**[**[24]
Diffie-Hellman	Section 6.8.4
ECKAS-DH	Section 6.9.5

For Diffie-Hellman Key Exchange, the size of the modulus (*p*) shall be one of the following: 512, 768, 1024, or 2048 bits.

5.2.2.2. Symmetric Algorithms for Key Exchange

The following symmetric (secret-key) algorithms are defined in this specification for key exchange.

Algorithm	Normative Reference
DES (in CBC mode)	[6], [7]
DES40 (in CBC mode)	Section 6.7
Triple DES (in CBC mode)	[25], [7]
FEAL (in CBC mode)	[32], [33], [7]

5.2.3. Generation of Master Keys

A Master Key is a key used between security agents for encrypting and decrypting session keys after initial connection establishment. Master Keys are generated during the security message exchange protocol described in Section 5.1. A single Master Key is used for both full duplex and simplex (e.g., point-multipoint) calls. Master Keys can be developed with public key encryption algorithms, e.g., RSA, or with public key development algorithms, e.g., Diffie-Hellman. The procedures for generating the Master Key with public key encryption algorithms and public key development algorithms are slightly different. The maximum size of a Master Key is 256 bits.

5.2.3.1. Master Keys on Point-To-Point Connections

To develop the Master Key for point-to-point connections using a public key encryption algorithm, e.g., RSA, 128 bit random values are selected by both the initiator and responder. These 128 bit values are exchanged in the Master Key octet group of the Confidential parameters (Section 6.2.4.1). The exchanged values are decrypted and concatenated to form the Master Key. The concatenation is performed as follows: the initiator value forms the most significant or leftmost half, and the responder value forms the least significant or rightmost half.

To develop the Master Key for point-to-point connections using a public key development algorithm, e.g., Diffie-Hellman, the low order 256 bits of the resultant shared secret value is the Master Key. The Master Key octet group is not required and will not be included in the Confidential Parameters. To accommodate this procedure, and also for security reasons, the modulus size for the Diffie-Hellman algorithm shall be at least 512 bits, and the modulus size for the Elliptic Curve Key Agreement Scheme-Diffie Hellman algorithm shall be at least 130 bits. The Initial Key Exchange Key is then set equal to the Master Key value and used as input to the Zmask algorithm for encryption of the Confidential Parameters.

5.2.3.2. Master Keys on Point-To-Multipoint Connections

For the case of point-to-multipoint calls, the addition of leaf nodes is handled differently than the initial call setup. This is because the Master Key for the connection has already been established (as described above), and need only be distributed to the new leaves.

When using public key encryption algorithms (e.g., RSA) the Master Key is encrypted in the responder's public key by the initiator and sent to the leaf as described in Section 5.1.

When using a public key development algorithm (e.g., Diffie-Hellman) the Master Key for the point-multipoint connection is encrypted using the Zmask algorithm defined in Section 6.8.4.1. The Initial Key Exchange Key used in the encryption algorithm (as an input to the hash) for subsequent leaves after the initial call setup must be pre-established. The pre-establishment of this key can be performed either through manual configuration or through prior key exchange. In both cases the encrypted Master Key for the connection is sent in the *ConfPar* token, along with the encrypted session key.

5.2.4. Generation of Initial Session Keys

Session keys are the keys used by the encryption algorithm for the data confidentiality service, or the integrity algorithm for the data integrity service. The two security agents involved in a security exchange randomly create session keys, and they are encrypted and transferred using the security message exchange protocol. The method used to encrypt the initial session keys is different when using public key encryption algorithms than it is when using public key development algorithms.

When using public key encryption algorithms or public key development algorithms, Initial Session Keys (ISKs) for confidentiality and/or integrity are created using a random number generator, and are exchanged in the session key octet group of the Confidential Parameters octet group. The Confidential Parameters octet group is encrypted in the other party's public key if using a public key encryption algorithm. If using a public key development algorithm, the Confidential Parameters octet group containing the ISKs is encrypted in the shared Initial Key Exchange Key. This is accomplished by performing the Zmask operation, as specified in Section 6.8.4.1, on the Initial Key Exchange Key and using the Zmask value to encrypt the Confidential Parameters.

For simplex calls (e.g., pt-mpt) only a single session key is needed. This key is always provided by the SA providing security for the root on pt-mpt calls, and by the SA providing security for the node which is generating the traffic on the simplex connection., i.e., the node which has bandwidth allocated in the forward direction. The *ConfPar* token is sent in both directions for simplex connections, but only one value

is used; the other discarded. For full duplex calls, separate initial session keys and initialization vectors shall be used in each direction.

5.2.5. Error Processing

Error processing for the key exchange service is described in the context of the security message exchange protocol in Section 5.1.

5.3. Session Key Update

5.3.1. Session Key Update Protocol

When the initiator (or responder) wants to use a new session key, it sends key update OAM cells within the user data stream to exchange the new session key with the remote partner and to indicate to the remote partner when to start using the new session key. The key update OAM flows are end-to-end F4 flows for VPCs and end-to-end F5 flows for VCCs, respectively.

The session key update protocol involves two processes: exchanging a new session key between the initiator and responder, and changing over from the old session key to the new session key. The first process is referred to as "session key exchange" (SKE) and the second process is referred to as "session key changeover" (SKC). The format for these OAM cells is defined in Section 5.1.7.

In order to change keys at high speeds, without disrupting service to the end user, two session keys are required: a current-key and a next-key. The next-key is delivered from the source to the destination using the SKE cell. The destination stores the next-key in a separate memory location until needed. The Bank ID and/or key number fields in the SKE cell are used to identify the next-key (with respect to the current key). The actual changeover occurs when the SKC cell is sent.

The process of performing key updates is independent in each direction of data flow, for full duplex connections. It is the responsibility of the source (e.g., the encrypting side of the data confidentiality service) of each data flow to initiate the key update in its direction. In point-to-multipoint connections there are one source and multiple destinations. Since point-to-multipoint connections are one-directional (from the source to destinations), only the root updates the session key, and the same session key is used by the root as well as all the leaves.

5.3.1.1. Session Key Exchange (SKE) Process

The SKE OAM cell is used to securely transfer the next session key from the source to the destination(s). Each key update uses a sequential key number for synchronization between the source and destination(s), as well as for cryptographic protection. A 32 bit numbering scheme is used, which means that 2^{32} - 1 session key updates can be performed on a connection before the key number wraps around (which will not happen in 2500 years even if the session key is updated every hour).

The next session key is encrypted using the master key obtained during secure call establishment. This is a 32-byte (or 256 bit) field that provides sufficient length for all session key lengths currently identified, with room for growth in the future. The actual length of the information in this field is dependent on the session key length of the algorithm being used.

The SKE process is derived from the method described by Bird, *et al*. in [27]. The process is defined using both the MD5 and SHA-1 hash functions. The determination of which to use is negotiated during secure call establishment.

5.3.1.1.1. SKE Processing at the source (or key update initiator)

Given a shared secret "master key" that is exchanged using the authentication/key exchange protocol at connection establishment time, the source can inform the destination(s) of a new session key that will be used for subsequent data traffic. When the source wants to send a new session key to the destination(s), it invokes the SKE process.

The format of the SKE OAM cell is defined in Section 5.1.7.2.1.1. The Bank ID is a 4-bit binary indication, coded as all zeros or all ones, which indicates the bank associated with the next session key. The key number (KN) is a 4-byte field, which indicates the key number associated with the new session key. Each session key is assigned a number. The first session key to be exchanged by the session key update protocol is assigned 1; the second session key is assigned 2, and so forth (using binary unsigned coding).

The key number field is used to maintain key update synchronization between the source and destination (i.e., to ensure both ends are using the same session key). In addition it is cryptographically protected to ensure freshness, uniqueness, and ordering of session key updates. This protection is described below.

The following procedure is used to generate the SKE OAM cell. It is assumed that the source *A* shares a secret master key, $K_{m,AB}$, with the destination *B*. In a point-to-multipoint connection, it is assumed that the same master key is shared by the source and all the destinations.

A generates the new Bank ID = current Bank ID $\oplus$ 0F hex.

A increments its key number (i.e., KN_A) by one. KN_A is set to 0 at the time the connection is established.

A generates a new session key $K_{s,AB}$. This session key is generated randomly, and is checked to ensure it is not "weak" (with respect to the data confidentiality algorithm or the data integrity algorithm).

A encrypts the session key under the master key as defined in section 6.8.5 or 6.8.6.

Each source shall insure that the combination of the master key and the key number are locally unique for each key update, to maintain the strength of the security. This does not include the extremely rare case when randomly generated master keys might happen to be identical.

A sends the Bank ID, key number (i.e., KN_A), and the encrypted session key (i.e., $Enc_{K_{m,AB}}(K_{s,AB})$) to *B* in the SKE OAM cell. The session key, $K_{s,AB}$, can have any length up to 256 bits (which is the length of the encrypted session key field of the SKE OAM). For cases when it is less than 256 bits long, the key shall reside in the low order bits, and the high order bits shall be set to zero. For Triple DES keys (using the $E_{k1}(D_{k2}(E_{k1}()))$ convention), the *k*2 key shall use the low order 64 bits, and the *k*1 key the next higher 64 bits.

Note that the old session key remains "active" at the source and destination(s) until session key changeover (SKC) is performed. The SKE OAM cell may get lost and the source will not be able to detect that (since there is no acknowledgment). To improve the probability that the session key exchange is successful, the source shall transmit the SKE OAM cell at least three (3) times for L_{SKE} seconds. The value of L_{SKE} is an implementation parameter and need not be standardized.

5.3.1.1.2.SKE Processing at the Destination (or Key Update Responder)

Upon receipt of an SKE OAM cell, the destination *B* extracts the Bank ID, key number KN_A, and the encrypted session key, and performs the following steps:

Verifies the 10-bit OAM cell CRC is correct, and discards the cell if it is not.

Checks the received KN_A. If this is the first session key received from *A* in a SKE OAM cell, the key number is valid if it is greater than or equal to one. If this is not the first session key received from *A* in a

SKE OAM cell, the key number is valid only if it is greater than the key number associated with the most recent (i.e., the previous) session key received from *A*. If these tests fail, then the SKE OAM cell is discarded and no further action is taken.

If the key number is valid, then *B* recovers the new session key by:

$$K_{s,AB} = Enc_{K_{m,AB}}(K_{s,AB}) \quad Mask$$

The mask is computed as defined in section 6.8.5 using MD5, or section 6.8.6 for SHA-1.

The recovered key (or next-key) is stored in memory, until the corresponding SKC cell is received. The Bank ID, the KN, or both can be used to access the next-key.

5.3.1.2. Session Key Changeover (SKC) Process

After the SKE process is completed from the source point of view (i.e., after the SKE OAM cell is transmitted multiple times for L_{SKE} seconds), the source invokes the session key changeover process to indicate to the destination(s) when to start using the new session key. The format of the SKC cell shall be as shown Section 5.1.7.2.2.1.

The source sends the SKC OAM cell that instructs the destination(s) to start using the new session key on the cells following the SKC OAM cell. The SKC OAM cell carries the key number associated with the new session key to which the destination(s) shall switch. The SKC OAM cell is sent multiple times to guarantee receipt at the destination(s) in the presence of cell loss. The SKC OAM cell is not cryptographically protected (since it does not carry any confidential information).

5.3.1.2.1. SKC Processing at the source (or key update initiator)

The source performs the following steps:

The source shall wait at least 1 second after sending the last SKE cell before sending the corresponding SKC cell, to allow the destination sufficient time to complete the SKE processing.

The source sets the Bank ID and the *KN* of the SKC cell to the most recent values sent in the SKE cell.

The source sets the State Vector (SV) of the SKC cell to all zeros, unless the counter mode (of the data confidentiality service) is being used, in which case the SV is initialized as described in Section 6.4.

The source injects the SKC OAM cell into the connection undergoing key changeover in such a manner that the session key(s) negotiated by the SKE process and the state vector (if applicable) are used on the next user cell associated with that connection. This includes the case when the next user cell on that connection immediately follows the SKC cell.

Note that in counter mode the SKC cell performs a cryptographic resynchronization as well as a key changeover. It is permissible to send an SKC cell expressly for this purpose, without any key changeover. This is accomplished by setting the Bank ID and key number to the values currently in use.

The SKC OAM cell may get lost and the source will not be able to detect that (since there is no acknowledgment). To improve the probability that the session key changeover is successful, the source shall transmit the SKC OAM cell at least three (3) times for L_{SKC} seconds. The value of L_{SKC} is an implementation parameter and need not be standardized. It is also permissible to insert several SKC cells back-to-back. The value of L_{SKC} is an implementation parameter and need not be standardized.

As stated previously, multiple SKC cells are inserted in anticipation of cell loss. It is important to clarify that for the counter mode, the state vector will be different each time the SKC cell is sent if any user cells

occur in the interim. If no user cells occur between SKC cells (e.g., back-to-back SKC cells), this field remains unchanged.

5.3.1.2.2. SKC Processing at the destination (or key update responder)

Upon receipt of an SKC OAM cell, the destination extracts the Bank ID, key number, and state vector (if applicable) and performs the following steps:

The destination verifies that the 10-bit OAM cell CRC is correct, and discards the cell if it is not.

The destination shall process the SKC OAM cell on the connection undergoing key changeover in such a manner that the session key negotiated by the SKE process and the state vector (if applicable) included in the SKC cell itself are used on the next cell received on that connection. This includes the case when this cell immediately follows the SKC cell.

The destination shall use either the Bank ID or the key number to access the session key negotiated by the SKE process and put it in place for use by the next user cell.

If the counter mode (of the data confidentiality service) is being used, the state vector shall be initialized as described in Section 6.4.4.3. For CBC or ECB modes, the state vector field is not used.

Note the SKC cell also performs a cryptographic resynchronization in the counter mode case.

5.3.1.3. Session Key Exchange Algorithms

Associated with each algorithm is a codepoint that is used for identification when negotiating security options. These codepoints are listed in Section 6.1. Procedures for using user-defined algorithm codepoints are also described in Section 6.1.

The following algorithms are defined in this specification for session key exchange.

Algorithm	Normative Reference
MD5	Section 6.8.5
SHA-1	Section 6.8.6
RIPEMD-160	Section 6.8.6, [18]

5.4. Certification and Certificate Revocation Lists

The security message exchange protocol described in Section 5.1 also provides for the exchange of public key certificates.

The X.509 v1 certificate shall be used for ATM certification when certificate exchange is performed in the signaling channel. When certificate exchange is performed in-band (in the user data channel), one of the X.509 v1, v2, and v3 certificates shall be used.

6. Normative Annex

6.1. In-Band Security Message Exchange Finite State Machines (FSMs)

The Finite State Machines (FSMs) described in this section specify the intended behavior for the in-band security message exchange protocol. These FSMs correspond to the textual procedures described in Section 5.1.5.3 of this specification. If there are any discrepancies between the textual procedures and the FSM tables, the FSM tables shall take precedence.

The FSMs covers two potential configurations:

1. Initiator of security message exchange,
2. Responder to security message exchange.

The FSMs are described in five sections:

1. The FSM Graphical Views are shown in Section 6.1.1.
2. All FSM States are described in section 6.1.2.
3. All FSM Events are described in section 6.1.3.
4. All FSM Actions are described in section 6.1.4.
5. The FSM Summary Tables are shown in section 6.1.5.

6.1.1. FSM Graphical View

Figure 36 show the Initiator FSM Graphical View.

Figure 37 shows the Responder FSM graphical view.

The FSM Graphical View is intended as a complement to the FSM Summary Table. If conflicts between the FSM Summary Table and the FSM Graphical View exist, the FSM Summary Table will take precedence.

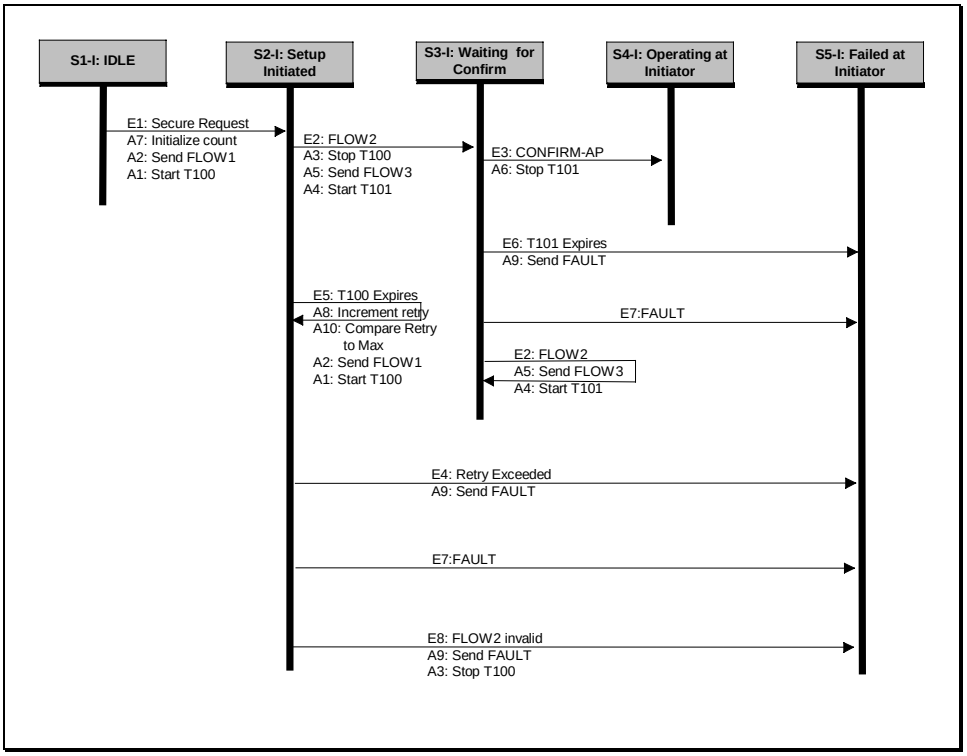


Figure 36. Initiator FSM.

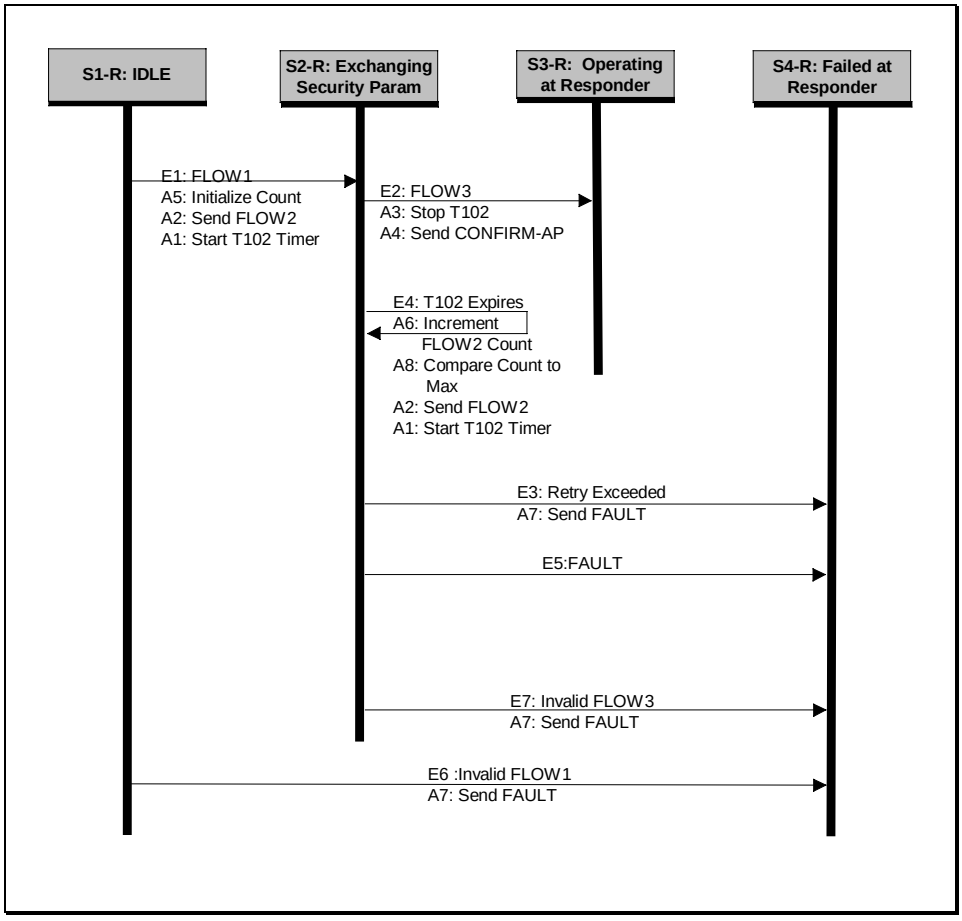


Figure 37. Responder FSM.

6.1.2. FSM States

Table 12: States

INITIATOR -I			
Number	**Name**	**Messages Outstanding**	**Description**
S1 -I	Idle	None	No requests for secure connection outstanding • This is always the first initiator state • Waiting to start operation
S2 -I	Setup Initiated	FLOW1	Initiator has initiated a secure connection by sending a FLOW1 to the responder
S3 -I	Waiting for Confirma-tion	FLOW3	Initiator has sent a FLOW3 to the Responder and is waiting for the Responder to confirm completion of the message exchange
S4 -I	Operating at Initiator	None	Initiator has made the connection available to the user
S5 -I	Failed		Error occurred during the message exchange protocol. For errors detected at the initiator: • A FAULT message is sent to the responder
RESPONDER -R			
Number	**Name**	**Messages Outstanding**	**Description**
S1-R	Idle	None	No requests for secure connection outstanding • This is always the first responder state • Waiting to start operation
S2-R	Exchanging Security Parameters	FLOW2	Responder has replied to the initiator's request with a FLOW2 message.
S3-R	Operating at Responder	None	Responder has made the connection available to the user
S4-R	Failed		Error occurred during the message exchange protocol. For errors detected at the responder: • A FAULT message is sent to the initiator

6.1.3. FSM Events

Table 13: Events

	INITIATOR	
Number	**Name**	**Description**
E1	Secure Request	The initiator has been requested to secure a connection
E2	Valid FLOW2 Received	FLOW2 message has been received • this event is expected
E3	CONFIRM-AP Received	CONFIRM-AP message has been received • this event is expected
E4	I-FLOW1 Retry Count Exceeded	Initiator has sent FLOW1 the maximum number of times as defined in section 5.1.5.3, Table 8 • this event results in a fault at the initiator
E5	T100 Expires	Timer 100 has exceeded time shown in section 5.1.5.3, Table 7 • this event results in a fault at the initiator
E6	T101 Expires	Timer 101 has exceeded time shown in section 5.1.5.3, Table 7 • this event results in a fault at the initiator
E7	FAULT Received	FAULT message has been received • this event indicates to the initiator that the responder has gone to the Fault state and the reason code
E8	Invalid FLOW2 Received	FLOW2 message with errors has been received • this event results in a fault at the initiator
	RESPONDER	
Number	**Name**	**Description**
E1	Valid FLOW1 Received	FLOW1 message has been received • this event is expected at the responder
E2	Valid FLOW3 Received	FLOW3 message has been received • this event is expected at the responder
E3	R-FLOW2 Retry Count Exceeded	Responder has sent FLOW2 the maximum number of times as defined in section 5.1.5.3, Table 8 • this event results in a fault at the responder
E4	T102 Expires	Timer 102 has exceeded time shown in section 5.1.5.3, Table 7 • this event results in a fault at the responder
E5	FAULT Received	FAULT message has been received • this event indicates to the responder that the initiator has gone to the Fault state and the reason code
E6	Invalid FLOW1 Received	FLOW1 message with errors has been received • this event results in a fault at the responder
E7	Invalid FLOW3 Received	FLOW3 message with errors has been received • this event results in a fault at the responder

6.1.4. FSM Actions

Table 14: Actions

	INITIATOR	
Number	**Name**	**Description**
A1	Start T100 Timer	Start timer T100
A2	Send FLOW1	Send a FLOW1 message from initiator to responder
A3	Stop T100 Timer	Stop timer T100
A4	Start T101 Timer	Start timer T101
A5	Send FLOW3	Send a FLOW3 message from initiator to responder
A6	Stop T101 Timer	Stop timer T101
A7	Initialize FLOW1 count	Set FLOW1 retry counter to 0
A8	Increment FLOW1 retry	Increment the counter for the number of times FLOW1 message has been sent
A9	Send FAULT	Send a FAULT message from initiator to responder
A10	Compare to Max	Compare the FLOW1 sent count to the Maximum Retry Value
	RESPONDER	
Number	**Name**	**Description**
A1	Start T102 Timer	Start timer T102
A2	Send FLOW2	Send a FLOW2 message from responder to initiator
A3	Stop T102 Timer	Stop timer T102
A4	Send CONFIRM-AP	Send a CONFIRM-AP message from responder to initiator
A5	Initialize FLOW2 count	Set FLOW2 retry counter to 0
A6	Increment FLOW2 retry	Increment the counter for the number of times FLOW2 message has been sent
A7	Send FAULT	Send a FAULT message from responder to initiator
A8	Compare to Max	Compare the FLOW2 sent count to the Maximum Retry Value

6.1.5. FSM Summary Table

Table 15: Indicator Summary.

States x Events:	**S1-I**	**S2-I**	**S3-I**	**S4-I**	**S5-I**
E1	A7, A2, A1 S2-I	N/A	N/A	N/A	N/A
E2	N/A	A3, A5, A4 S3-I	A5, A4 S3-I	N/A	N/A
E3	N/A	N/A	A6 S4-I	N/A	N/A
E4	N/A	A9 S5-I	N/A	N/A	N/A
E5	N/A	A8, A10, A2,A1 S2-I	N/A	N/A	N/A
E6	N/A	N/A	A9 S5-I	N/A	N/A
E7	N/A	S5-I	S5-I	N/A	N/A
E8	N/A	A9, A3 S5-I	N/A	N/A	N/A

Table 16: Responder Summary.

States x Events:	**S1-R**	**S2-R**	**S3-R**	**S4-R**
E1	A5, A2, A1 S2-R	N/A	N/A	N/A
E2	N/A	A3, A4 S3-R	N/A	N/A
E3	N/A	A7 S4-R	N/A	N/A
E4	N/A	A6, A8, A2, A1 S2-R	N/A	N/A
E5	N/A	S4-R	N/A	N/A
E6	A7 S4-R	N/A	N/A	N/A
E7	N/A	A7 S4-R	N/A	N/A

6.2. Security Service Data Primitives

6.2.1. Security Agent Identification Primitives

This document specifies the use of “distinguished names” for initiating, responding, and explicitly targeted security agents. The distinguished name shall be a globally unique identifier for the security agent. There are two distinguished name formats specified here: the X.509 distinguished name and the globally-unique ATM address of the ATM network element that hosts the security agent. There are currently no

conventions for X.509 distinguished names for security agents; furthermore, X.509 distinguished names can be long–possibly long enough to limit their use to the in-band message exchange.

6.2.1.1. Initiator Distinguished Name

This octet group contains the Distinguished Name (ID) of the initiating security agent in a Security Message Exchange.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	0	0	0	0	1	0	
Initiator Distinguished Name Identifier								x.1
Initiator Distinguished Name Length								x.2
Initiator Distinguished Name Type								x.3
Initiator Distinguished Name Value								x.4 etc.

Initiator Distinguished Name Length (Octet x.2)

A binary number indicating the length in octets of the Initiator distinguished name type and value fields, contained in octets x.3, x.4, etc.

Initiator Distinguished Name Type (Octet x.3)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	X.509 Distinguished Name
0	0	0	0	0	0	1	0	ATM End System Address

Initiator Distinguished Name Value (Octet x.4 etc.)

This octet group contains the value of the distinguished name (ID) of the Initiator of the Security Message Exchange Protocol.

X.509 Distinguished Name

This document does not specify naming conventions for security agents. However, it is anticipated that those conventions will be developed and that X.509 distinguished names will need to be passed in security messages. The coding and length of this field shall be determined by the coding and length of the same names when carried in X.509 certificates–that is, according to the Basic Encoding Rules (BER, ISO 8825),with the restrictions imposed by X.509 to make BER deterministic.

ATM End System Address (AESA)

This field contains the ATM address associated with the Initiator. The address is coded as described in ISO 8348, Addendum 2, using the preferred binary encoding. For further details on using this field, consult Section 3.0 of [3]. When these addresses need to be used in X.509 certificates, the following ASN.1 production shall be used to define the attribute used for the distinguished name:

```
aTMAddress ATTRIBUTE        ::= {
        WITH SYNTAX                             OCTET STRING
        EQUALITY MATCHING RULE   octetStringMatch
        SUBSTRINGS MATCHING RULE        octetSubstringsMatch

        ID                                      ...aTMAddress }
```

6.2.1.2. Responder Distinguished Name

This octet group contains the Distinguished Name (ID) of the responding security agent in a Security Message Exchange.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	0	0	0	0	1	0	
Responder Distinguished Name Identifier								x.1
Responder Distinguished Name Length								x.2
Responder Distinguished Name Type								x.3
Responder Distinguished Name Value								x.4 etc.

Responder Distinguished Name Length (Octet x.2)

A binary number indicating the length in octets of the Responder distinguished name type and value fields, contained in octets x.3, x.4, etc.

Responder Distinguished Name Type (Octet x.3)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	X.509 Distinguished Name
0	0	0	0	0	0	1	0	ATM End System Address

Responder Distinguished Name Value (Octet x.4 etc.)

This octet group contains the value of the distinguished name (ID) of the Responder of the Security Message Exchange Protocol.

X.509 Distinguished Name

This document does not specify naming conventions for security agents. However, it is anticipated that those conventions will be developed and that X.509 distinguished names will need to be passed in security messages. The coding and length of this field shall be determined by the coding and length of the same names when carried in X.509 certificates–that is, according to the Basic Encoding Rules (BER, ISO 8825),with the restrictions imposed by X.509 to make BER deterministic.

ATM End System Address (AESA)

This field contains the ATM address associated with the Responder. The address is coded as described in ISO 8348, Addendum 2, using the preferred binary encoding. For further details on using this field, consult Section 3.0 of [3]. When these addresses need to be used in X.509 certificates, the following

ASN.1 production shall be used to define the attribute used for the distinguished name:

```
aTMAddress ATTRIBUTE        ::= {
        WITH SYNTAX                             OCTET STRING
        EQUALITY MATCHING RULE    octetStringMatch
        SUBSTRINGS MATCHING RULE        octetSubstringsMatch
        ID                                      ...aTMAddress }
```

6.2.1.3. Security Agent Identifier

This octet group contains the distinguished name (ID) of the explicitly targeted security agent in a Security Message Exchange.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	0	0	0	0	1	0	
Security Agent Distinguished Name Identifier								x.1
Security Agent Distinguished Name Length								x.2
Security Agent Distinguished Name Type								x.3
Security Agent Distinguished Name Value								x.4 etc.

Security Agent Distinguished Name Length (Octet x.2)

A binary number indicating the length in octets of the Security Agent distinguished name type and value fields, contained in octets x.3, x.4, etc.

Security Agent Distinguished Name Type (Octet x.3)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	X.509 Distinguished Name
0	0	0	0	0	0	1	0	ATM End System Address

Security Agent Distinguished Name Value (Octet x.4 etc.)

This octet group contains the value of the distinguished name (ID) of the Security Agent of the Security Message Exchange Protocol.

X.509 Distinguished Name

This document does not specify naming conventions for security agents. However, it is anticipated that those conventions will be developed and that X.509 distinguished names will need to be passed in security messages. The coding and length of this field shall be determined by the coding and length of the same names when carried in X.509 certificates–that is, according to the Basic Encoding Rules (BER, ISO 8825),with the restrictions imposed by X.509 to make BER deterministic.

ATM End System Address (AESA)

This field contains the ATM address associated with the security agent. The address is coded as described in ISO 8348, Addendum 2, using the preferred binary encoding. For further details on using this field, consult Section 3.0 of

[3]. When these addresses need to be used in X.509 certificates, the following ASN.1 production shall be used to define the attribute used for the distinguished name:

```
aTMAddress ATTRIBUTE        ::= {
        WITH SYNTAX                             OCTET STRING
        EQUALITY MATCHING RULE    octetStringMatch
        SUBSTRINGS MATCHING RULE        octetSubstringsMatch
        ID                                      ...aTMAddress }
```

6.2.2. Security Service Specification Section

The Security Service Specification Section provides a description of the security services that are requested for a particular security association. Services are specified in two ways, through either a Security Service Declaration, and/or through a specific Security Service Option and Description. The Security Service Declaration, provided as a concise method of declaring to a peer or proxy security agent the services that will be used on the VC, is usually used when the SME will occur In-Band. The Security Service Option and Description Sections are designed for detailed security service declaration and negotiation. The Option is used to declare the required security service, the Description is used to specify in detail the specific service.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	0	0	1	0	0	0	
Security Service Specification Section Identifier								x.1
Security Service Declaration Section								x.2 etc*
Security Service Option Section								x.3 etc*
Security Service Algorithm Section								x.4 etc*

Security Service Declaration Section (Octet Group x.2)

This optional octet group declares the specific security services that are requested for a VC. Details are found below.

Security Service Option Section (Octet Group x.3)

This optional octet group indicates specific options for the indicated security service. Details are found below.

Security Service Algorithm Section (Octet Group x.4)]

This octet group describes the details of security service algorithms for the indicated security service. Details are found below.

6.2.2.1. Security Service Declaration

The Security Service Declaration Section provides a minimal description of the security services that are requested/supported by a security agent. The Security Service Declaration can be employed with in-band security message exchange. In this situation, the security service will be negotiated after the VC is

established. This method of declaring the security service can be employed when communicating a security requirement to a proxy security agent.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	0	0	1	0	1	0	
Security Service Declaration Identifier								x.1
Security Service Declaration								
x	x	x	x	x	x	x	x	x.2

Security Service Declaration (Octet x.2)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	0	Unspecified
0	0	0	0	0	0	0	1	Confidentiality Service
0	0	0	0	0	0	1	0	Integrity Service
0	0	0	0	0	1	0	0	Authentication Service
0	0	0	0	1	0	0	0	Key Exchange Service
0	0	0	1	0	0	0	0	Certificate Exchange Service
0	0	1	0	0	0	0	0	Session Key Update Service
0	1	0	0	0	0	0	0	Access Control Service

6.2.2.2. Security Service Option Primitives

In the 3-way exchange, the initiator indicates what security services are required, what security services can be supported should the responder requires them, and what security services are not supported for the connection. In the 2-way exchange, the initiator indicates what security services are required and what security services are not supported for the connection. In the 3-way exchange, the responder indicates what security services are required and what security services are not supported for the connection.

6.2.2.2.1. Data Confidentiality Service Options

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	0	1	0	0	0	0	
Data Confidentiality Service Option Identifier								x.1
Data Confidentiality Service Options								
x	x	x	x	x	x	x	x	x.2

Data Confidentiality Service Options (Octet x.2)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	0	Not Supported
0	0	0	0	0	0	0	1	Supported At ATM Cell Level (Note)
1	0	0	0	0	0	0	1	Required At ATM Cell Level

Note: This codepoint applies only when used by the initiator in FLOW1-3WE.

6.2.2.2.2. Data Integrity Service Options

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	0	1	0	0	1	0	
Data Integrity Service Options Identifier								x.1
Data Integrity Service Options								
x	x	x	x	x	x	x	x	x.2

Data Integrity Service Options (Octet x.2)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	0	Not supported
0	0	0	0	0	0	0	1	Supported With Replay / Reordering Protection (Note)
0	0	0	0	0	0	1	0	Supported Without Replay / Reordering Protection (Note)
1	0	0	0	0	0	0	1	Required With Replay/Reordering Protection
1	0	0	0	0	0	1	0	Required Without Replay/Reordering Protection

Note: This codepoint applies only when used by the initiator in FLOW1-3WE.

6.2.2.2.3. Authentication Service Options

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	0	1	0	0	1	1	
Authentication Service Options Identifier								x.1
Authentication Service Options								
x	x	x	x	x	x	x	x	x.2

Authentication Service Options (Octet x.2)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	0	Not supported
0	0	0	0	0	0	0	1	Supports Authentication (Note)
1	0	0	0	0	0	0	1	Requires Authentication

Note: This codepoint applies only when used by the initiator in FLOW1-3WE.

6.2.2.2.4. Key Exchange Service Options

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	0	1	0	1	0	0	
Key Exchange Service Options Identifier								x.1
Key Exchange Service Options								
x	x	x	x	x	x	x	x	x.2

Key Exchange Service Options (Octet x.2)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	0	Not supported
0	0	0	0	0	0	0	1	Supports Key Exchange (Note 1)
0	1	0	0	0	0	0	1	Requires Key Exchange

Note 1: This codepoint applies only when used by the initiator in FLOW1-3WE.

6.2.2.2.5. Session Key Update Service Options

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	0	1	0	1	0	1	
Session Key Update Service Options Identifier								x.1
Session Key Update Service Options								
x	x	x	x	x	x	x	x	x.2

Session Key Update Service Options (Octet x.2)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	0	Not supported
0	0	0	0	0	0	0	1	Supports Session Key Update (Note)
1	0	0	0	0	0	0	1	Requires Session Key Update

Note: This codepoint applies only when used by the initiator in FLOW1-3WE.

6.2.2.2.6. Access Control Service Options

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	0	1	0	1	1	0	
Access Control Service Options Identifier								x.1
Access Control Service Options								
x	x	x	x	x	x	x	x	x.2

Access Control Service Options (Octet x.2)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	0	Not supported
0	0	0	0	0	0	0	1	Supports Access Control (Note)
1	0	0	0	0	0	0	1	Requires Access Control

Note: This codepoint applies only when used by the initiator in FLOW1-3WE to indicate that it supports access control.

6.2.3. Security Algorithm Primitives

The security algorithm section specifies the algorithms to be used for the indicated security services. This includes the algorithm's mode of operation that is to be used. In addition, the parameters associated with each algorithm are also included.

This specification allows for the use of user (or vendor) specified algorithms. If a user (or vendor) specified algorithm is used, then the optional octet group containing the user's (or vendor's) Organizationally Unique Identifier (OUI) shall be used. The value of the OUI is unique to the user (or vendor), and is assigned by the Institute for Electrical and Electronics Engineers (IEEE).

6.2.3.1. Data Confidentiality Algorithm

This octet group indicates an algorithm for the data confidentiality service.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	1	0	0	0	0	0	
Data Confidentiality Algorithm Identifier								x.1
Length of Data Confidentiality Algorithm Contents								x.2
Data Confidentiality Algorithm								x.3
OUI								x.4 (Note 1)
OUI (continued)								x.5 (Note 1)
OUI (continued)								x.6 (Note 1)
Data Confidentiality Mode of Operation								x.7
OUI								x.8 (Note 2)
OUI (continued)								x.9 (Note 2)
OUI (continued)								x.10 (Note 2)
Data Confidentiality Algorithm Details								x.11 etc.
Data Confidentiality Mode Details								x.12 etc.

Length of Data Confidentiality Algorithm Contents (Octet x.2)

A binary number indicating the length in octets of the data confidentiality algorithm/mode fields, contained in octets x.3, x.4, x.5 etc.

Data Confidentiality Algorithm (Octet x.3)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	DES with 56 bits effective key
0	0	0	0	0	0	1	0	DES with 40 bits effective key
0	0	0	0	0	0	1	1	Triple-DES with 112 bits effective key
0	0	0	0	0	1	0	0	FEAL, N=32, 64 bit key, no key block parity
1	x	x	x	x	x	x	x	User-defined data confidentiality algorithm (the value of bits 1-7 is user-defined)

OUI (Octets x.4 - x.6)

Note 1: if the data confidentiality algorithm is "user defined," then the IEEE OUI of the vendor/user that assigned the user-defined codepoint shall be included in these octets. Otherwise, these octets are not included.

Data Confidentiality Mode of Operation (Octet x.7)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	0	Unspecified
0	0	0	0	0	0	0	1	CBC
0	0	0	0	0	0	1	0	Counter Mode
0	0	0	0	0	0	1	1	ECB
1	x	x	x	x	x	x	x	User-defined data confidentiality mode of operation (the value of bits 1-7 is user-defined)

OUI (Octets x.8 - x.10)

Note 2: if the data confidentiality mode of operation is "user defined," then the IEEE OUI of the vendor/user that assigned the user-defined codepoint shall be included in these octets. Otherwise, these octets are not included.

Data Confidentiality Algorithm Details (Octet x.11 etc)

These octets indicate coding details for each data confidentiality algorithm. These details are found below:

⇒ DES coding details
- No further detail is required

⇒ DES40 coding details
Two eight-octet fields are added to the data confidentiality algorithm information element details. The first eight octets contain a binary encoded 64-bit random number. The second eight octets contain the binary encoding of that random number encrypted under the DES40 key in the ECB mode.

⇒ Triple DES coding details
- No further detail is required

⇒ FEAL coding details
- No further detail is required

Data Confidentiality Mode Details (Octet x.12 etc)

These octets indicate coding details for each data confidentiality mode of operation. These details are found below:

⇒ CBC mode coding details
A 64-bit (8-byte) binary encoded value. This is the initialization vector (IV) for the CBC mode.

⇒ Counter mode coding details
- No further detail is required

⇒ ECB mode coding details
- No further detail is required.

6.2.3.2. Data Integrity Algorithm

This octet group indicates an algorithm for the data integrity service.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	1	0	0	0	1	0	
Data Integrity Algorithm Identifier								x.1
Length of Data Integrity Algorithm Contents								x.2
User	Replay	Algorithm						
Data Integrity Algorithm								x.3
OUI								x.4 (Note)
OUI (continued)								x.5 (Note)
OUI (continued)								x.6 (Note)
Data Integrity Algorithm Details								x.7 etc.

Length of Data Integrity Algorithm Contents (Octet x.2)

A binary number indicating the length in octets of the data integrity algorithm fields, contained in octets x.3, x.4, etc.

User Defined Indicator (Octet x.3 Bit 8)

8	**Meaning**
0	ATM Forum Defined Algorithm
1	User Defined Algorithm

When User Defined Algorithms are indicated, the Bit 7 is no longer an indicator of Replay/Reordering Protection.

Replay/Reordering Protection Indicator (Octet x.3 Bit 7)

7	**Meaning**
0	No replay/reordering protection provided.
1	Replay/reordering protection provided.

Data Integrity Algorithm (Octet x.3 Bits 6 - 1)

6	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	1	HMAC-MD5
0	0	0	0	1	0	HMAC-SHA-1
0	0	0	0	1	1	HMAC-RIPEMD-160
0	0	0	1	0	0	MAC generated using DES in CBC mode
0	0	0	1	0	1	MAC generated using DES-40 in CBC mode
0	0	0	1	1	0	MAC generated using Triple-DES in CBC mode
0	0	0	1	1	1	MAC generated using FEAL in CBC mode

OUI (Octets x.4 - x.6)

Note: if the data integrity algorithm is “user defined,” then the IEEE OUI of the vendor/user that assigned the user-defined codepoint shall be included in these octets. Otherwise, these octets are not included.

Data Integrity Algorithm Details (Octet x.7 etc.)

These octets indicate coding details for each data integrity algorithm. These details are found below:

⇒ HMAC-MD5 coding details
- No further detail is required.

⇒ HMAC-SHA-1 coding details
- No further detail is required.

⇒ HMAC-RIPEMD-160 coding details
- No further detail is required.

⇒ DES in CBC mode MAC details
- No further detail is required.

⇒ DES-40 in CBC mode MAC details
- No further detail is required.

⇒ Triple DES in CBC mode MAC details
- No further detail is required.

⇒ FEAL in CBC mode MAC details
- No further detail is required.

6.2.3.3. Hash Algorithm

This octet group indicates a hashing algorithm for the authentication service.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	1	0	0	1	0	0	
Hash Algorithm Identifier								x.1
Length of Hash Algorithm Contents								x.2
Hash Algorithm								x.3
OUI								x.4 (Note)
OUI (continued)								x.5 (Note)
OUI (continued)								x.6 (Note)
Hash Algorithm Details								x.7 etc*

Length of Hash Algorithm Contents (Octet x.2)

A binary number indicating the length in octets of the hash algorithm fields, contained in octets x.3, x.4, etc.

Hash Algorithm (Octet x.3)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	MD5
0	0	0	0	0	0	1	0	SHA-1
0	0	0	0	0	0	1	1	RIPEMD-160
1	x	x	x	x	x	x	x	User-defined hash algorithm (the value of bits 1-7 is user-defined)

OUI (Octets x.4 - x.6)

Note: if the hash algorithm is "user defined," then the IEEE OUI of the vendor/user that assigned the user-defined codepoint shall be included in these octets. Otherwise, these octets are not included.

Hash Algorithm Details (Octet x.7 etc.)

These octets indicate coding details for each hash algorithm. These details are found below:

⇒ MD5 coding details
- No further detail is required.

⇒ SHA-1 coding details
- No further detail is required.

⇒ RIPEMD-160 coding details
- No further detail is required.

6.2.3.4. Signature Algorithm

This octet group indicates a signature algorithm for the Authentication Service.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	1	0	0	1	1	0	
Signature Algorithm Identifier								x.1
Length of Signature Algorithm Contents								x.2
Signature Algorithm								x.3
OUI								x.4 (Note)
OUI (continued)								x.5 (Note)
OUI (continued)								x.6 (Note)
Signature Algorithm Details								x.7 etc*.

Length of Signature Algorithm Contents (Octet x.2)

A binary number indicating the length in octets of the signature algorithm fields, contained in octets x.3, x.4, etc.

Signature Algorithm (Octet x.3)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	RSA
0	0	0	0	0	0	1	0	DSA
0	0	0	0	0	0	1	1	Elliptic Curve/DSA like
0	0	0	0	0	1	0	0	ESIGN
0	0	0	0	0	1	0	1	DES/CBC
0	0	0	0	0	1	1	0	DES40/CBC
0	0	0	0	0	1	1	1	Triple DES/CBC
0	0	0	0	1	0	0	0	FEAL/CBC
1	x	x	x	x	x	x	x	User-defined algorithm (the value of bits 1-7 is user-defined)

OUI (Octets x.4 - x.6)

Note: if the signature algorithm is "user defined," then the IEEE OUI of the vendor/user that assigned the user-defined codepoint shall be included in these octets. Otherwise, these octets are not included.

Signature Algorithm Details (Octet x.7 etc.)

These optional octets indicate coding details for each signature algorithm. These details are found below:

⇒ RSA coding details
- No further detail is required.

⇒ DSA coding details
- No further detail is required.

⇒ Elliptic Curve/DSA coding details
- No further detail is required (cf. Section 6.9)

⇒ ESIGN coding details

The order of *k*:
This octet contains the binary encoding of the order of *k* (i.e., *X*, in the relation $k = 2^X$)

⇒ DES/CBC coding details
- No further detail required.

⇒ DES40/CBC coding details
- No further detail required.

⇒ Triple DES/CBC coding details
- No further detail required.

⇒ FEAL/CBC coding details
- No further detail required.

6.2.3.5. Key Exchange Algorithm

This octet group indicates an algorithm for key exchange. This algorithm is used to encrypt the contents part of the "Security confidential parameters" information element.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	1	0	1	0	0	0	
Key Exchange Algorithm Identifier								x.1
Length of Key Exchange Algorithm Contents								x.2
Key Exchange Algorithm								x.3
OUI								x.4 (Note)
OUI (continued)								x.5 (Note)
OUI (continued)								x.6 (Note)
Key Exchange Algorithm Details								x.7 etc.*

Length of Key Exchange Algorithm Contents (Octet x.2)

A binary number indicating the length in octets of the key exchange algorithm fields, contained in octets x.3, x.4, etc.

Key Exchange Algorithm (Octet x.3)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	RSA
0	0	0	0	0	0	1	0	Diffie-Hellman
0	0	0	0	0	0	1	1	ECKAS-DH (prime field)
0	0	0	0	0	1	0	0	ECKAS-DH (characteristic 2 field)
0	0	0	0	0	1	0	1	DES/CBC
0	0	0	0	0	1	1	0	DES40/CBC
0	0	0	0	0	1	1	1	Triple DES/CBC
0	0	0	0	1	0	0	0	FEAL/CBC
1	x	x	x	x	x	x	x	User-defined key exchange algorithm (the value of bits 1-7 is user-defined)

OUI (Octets x.4 - x.6)

Note: if the key exchange algorithm is “user defined,” then the IEEE OUI of the vendor/user that assigned the user-defined codepoint shall be included in these octets. Otherwise, these octets are not included.

Key Exchange Algorithm Details (Octet x.7 etc.)

These octets indicate coding details for each key exchange algorithm. These details are found below:

⇒ RSA coding details
- No further detail is required.

⇒ Diffie-Hellman coding details

Diffie-Hellman public key identifier (Octet x.7a):
This octet is the identifier for the Diffie-Hellman public key.

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	Diffie-Hellman public key

Diffie-Hellman public key length (Octet x.7b):
This octet specifies the length of the Diffie-Hellman public key in octets.

Diffie-Hellman public key value (Octet x.7c etc.):
This field contains octet string representation of the Diffie-Hellman public key value. This field contains the initiator's Diffie-Hellman public key value (*X*) when this information element is sent in FLOW1-3WE. This field contains the responder's Diffie-Hellman public key value (*Y*) when this information element is sent in FLOW2-3WE.

Diffie-Hellman modulus identifier (Octet x.8a):
This octet is the identifier for the Diffie-Hellman public modulus.

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	1	0	Diffie-Hellman public modulus

Diffie-Hellman modulus length (Octet x.8b):
This octet specifies the length of the Diffie-Hellman public modulus in octets.

Diffie-Hellman modulus value (Octet x.8c etc.):
This field contains octet string representation of the Diffie-Hellman public modulus value (*P*).

Diffie-Hellman base identifier (Octet x.9a):
This octet is the identifier for the Diffie-Hellman public base.

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	1	1	Diffie-Hellman public base

Diffie-Hellman base length (Octet x.9b):
This octet specifies the length of the Diffie-Hellman public base in octets.

Diffie-Hellman base value (Octet x.9c etc.):
This field contains octet string representation of the Diffie-Hellman public base value (*G*).

⇒ Elliptic Curve Key Agreement Scheme-Diffie Hellman Analogue (ECKAS-DH) coding details for prime fields (cf. Section 6.9.5):

ECKAS-DHp public key identifier (Octet x.7a):
This octet is the identifier for the compressed ECKAS-DHp public key.

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	ECKAS-DHp public key

ECKAS-DHp public key length (Octet x.7b):
This octet specifies the length of the compressed ECKAS-DHp public key in octets.

ECKAS-DHp public key value (Octet x.7c etc.):
This field contains octet string representation of the compressed ECKAS-DHp public key value. This field contains the initiator's compressed ECKAS-DHp public key value (*aP*) when this information element is sent in FLOW1-3WE. This field contains the responder's compressed ECKAS-DHp public key value (*bP*) when this information element is sent in FLOW2-3WE.

ECKAS-DHp modulus identifier (Octet x.8a):

This octet is the identifier for the ECKAS-DHp public modulus (*p*).

8	7	6	5	4	3	2	1	Meaning
0	0	0	0	0	0	1	0	ECKAS-DHp public modulus

ECKAS-DHp modulus length (Octet x.8b):
This octet specifies the length of the ECKAS-DHp public modulus in octets.

ECKAS-DHp modulus value (Octet x.8c etc.):
This field contains octet string representation of the ECKAS-DHp public modulus value (*p*).

ECKAS-DHp base identifier (Octet x.9a):
This octet is the identifier for the compressed ECKAS-DHp public base (*P*) (cf. Section 6.9).

8	7	6	5	4	3	2	1	Meaning
0	0	0	0	0	0	1	1	ECKAS-DHp public base

ECKAS-DHp base length (Octet x.9b):
This octet specifies the length of the compressed ECKAS-DHp public base in octets.

ECKAS-DHp base value (Octet x.9c etc.):
This field contains octet string representation of the compressed ECKAS-DHp public base.

ECKAS-DHp Weierstrass coefficient identifier (Octet x.10a):
This octet is the identifier for the ECKAS-DHp Weierstrass coefficient (*a*) (cf. Section 6.9).

8	7	6	5	4	3	2	1	Meaning
0	0	0	0	0	1	0	0	ECKAS-DHp Weierstrass coefficient

ECKAS-DHp Weierstrass coefficient length (Octet x.10b):
This octet specifies the length of the ECKAS-DHp Weierstrass coefficient (*a*) in octets.

ECKAS-DHp Weierstrass coefficient value (Octet x.10c etc.):
This field contains octet string representation of the ECKAS-DHp Weierstrass coefficient (*a*).

ECKAS-DHp Weierstrass coefficient identifier (Octet x.11a):
This octet is the identifier for the ECKAS-DHp Weierstrass coefficient (*b*) (cf. Section 6.9).

8	7	6	5	4	3	2	1	Meaning
0	0	0	0	0	1	0	1	ECKAS-DHp Weierstrass coefficient

ECKAS-DHp Weierstrass coefficient length (Octet x.11b):
This octet specifies the length of the ECKAS-DHp Weierstrass coefficient (*b*) in octets.

ECKAS-DHp Weierstrass coefficient value (Octet x.11c etc.):

This field contains octet string representation of the ECKAS-DHp Weierstrass coefficient (*b*).

⇒ Elliptic Curve Key Agreement Scheme-Diffie Hellman Analogue (ECKAS-DH) coding details for characteristic 2 (cf. Section 6.9.5):

ECKAS-DH2 public key identifier (Octet x.7a):
This octet is the identifier for the compressed ECKAS-DH2 public key.

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	ECKAS-DH2 public key

ECKAS-DH2 public key length (Octet x.7b):
This octet specifies the length of the compressed ECKAS-DH2 public key in octets.

ECKAS-DH2 public key value (Octet x.7c etc.):
This field contains octet string representation of the compressed ECKAS-DH2 public key value. This field contains the initiator's compressed ECKAS-DH2 public key value (*aP*) when this information element is sent in FLOW1-3WE. This field contains the responder's compressed ECKAS-DH2 public key value (*bP*) when this information element is sent in FLOW2-3WE.

ECKAS-DH2 field size identifier (Octet x.8a):
This octet is the identifier for the ECKAS-DH2 field size (*m*).

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	1	0	ECKAS-DH2 field size

ECKAS-DH2 field size length (Octet x.8b):
This octet specifies the length of the ECKAS-DH2 field size (*m*) in octets.

ECKAS-DH2 field size value (Octet x.8c etc.):
This field contains octet string representation of the ECKAS-DH2 field size value (*m*).

ECKAS-DH2 base identifier (Octet x.9a):
This octet is the identifier for the compressed ECKAS-DH2 public base (*P*) (cf. Section 6.9).

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	1	1	ECKAS-DH2 public base

ECKAS-DH2 base length (Octet x.9b):
This octet specifies the length of the compressed ECKAS-DH2 public base in octets.

ECKAS-DH2 base value (Octet x.9c etc.):
This field contains octet string representation of the compressed ECKAS-DH2 public base.

ECKAS-DH2 Weierstrass coefficient (*a*) identifier (Octet x.10a):
This octet is the identifier for the ECKAS-DH2 Weierstrass coefficient (*a*) (cf. Section 6.9).

8	7	6	5	4	3	2	1	Meaning
0	0	0	0	0	1	0	0	ECKAS-DH2 Weierstrass coefficient (*a*)

ECKAS-DH2 Weierstrass coefficient (*a*) length (Octet x.10b):
This octet specifies the length of the ECKAS-DH2 Weierstrass coefficient (*a*) in octets.

ECKAS-DH2 Weierstrass coefficient (*a*) value (Octet x.10c etc.):
This field contains octet string representation of the ECKAS-DH2 Weierstrass coefficient (*a*).

ECKAS-DH2 Weierstrass coefficient (*b*) identifier (Octet x.11a):
This octet is the identifier for the ECKAS-DH2 Weierstrass coefficient (*b*) (cf. Section 6.9).

8	7	6	5	4	3	2	1	Meaning
0	0	0	0	0	1	0	1	ECKAS-DH2 Weierstrass coefficient (*b*)

ECKAS-DH2 Weierstrass coefficient (*b*) length (Octet x.11b):
This octet specifies the length of the ECKAS-DH2 Weierstrass coefficient (*b*) in octets.

ECKAS-DH2 Weierstrass coefficient (*b*) value (Octet x.11c etc.):
This field contains octet string representation of the ECKAS-DH2 Weierstrass coefficient (*b*).

⇒ DES/CBC coding details

- No further detail required.

⇒ DES40/CBC coding details

- No further detail required.

⇒ Triple DES/CBC coding details

- No further detail required.

⇒ FEAL/CBC coding details

- No further detail required.

6.2.3.6. Session Key Update Algorithm

This octet group indicates a session key update algorithm for updating the data confidentiality and data integrity session keys.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	1	0	1	0	1	0	
Session Key Update Algorithm Identifier								x.1
Length of Session Key Update Algorithm Contents								x.2
Session Key Update Algorithm								x.3
OUI								x.4 (Note)
OUI (continued)								x.5 (Note)
OUI (continued)								x.6 (Note)
Session Key Update Algorithm Details								x.7 etc.

Length of Session Key Update Algorithm Contents (Octet x.2)

A binary number indicating the length in octets of the key exchange algorithm fields, contained in octets x.3, x.4, etc.

Session Key Update Algorithm (Octet x.3)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	SKE with MD5
0	0	0	0	0	0	1	0	SKE with SHA-1
0	0	0	0	0	0	1	1	SKE with RIPEMD-160
1	x	x	x	x	x	x	x	User-defined key exchange algorithm (the value of bits 1-7 is user-defined)

OUI (Octets x.4 - x.6)

Note: if the session key update algorithm is “user defined,” then the IEEE OUI of the vendor/user that assigned the user-defined codepoint shall be included in these octets. Otherwise, these octets are not included.

Session Key Update Algorithm Details (Octet x.7 etc.)

These octets indicate coding details for each session key update algorithm. These details are found below:

⇒ SKE with MD5 coding details
- No further detail is required.

⇒ SKE with SHA-1 coding details
- No further detail is required.

⇒ SKE with RIPEMD-160 coding details
- No further detail is required.

6.2.3.7. Authentication Algorithm Profile Group

This octet group indicates a set of algorithms supporting authentication service.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	1	0	1	1	0	0	
Authentication Algorithm Group Identifier								x.1
Length of Authentication Algorithm Group Contents								x.2
Authentication Algorithm Group								x.3
OUI								x.4
OUI (continued)								x.5
OUI (continued)								x.6
1	0	1	1	0	0	1	0	
Signature Algorithm Details Identifier								x.7
Length of Signature Algorithm Details								x.8
Signature Algorithm Details								x.9 etc.
1	0	1	1	0	1	0	0	
Hash Algorithm Details Identifier								x.10 etc.
Length of Hash Algorithm Details								x.11 etc.
Hash Algorithm Details								x.12 etc.

Length of Authentication Algorithm Group Contents (Octet x.2)

A binary number indicating the length in octets of the authentication algorithm group fields contained in x.3, x.4, etc.

Authentication Algorithm Group (Octet x.3)

8 7 6 5 4 3 2 1	**Meaning**
0 0 0 0 0 0 0 1	AUTH-1 Signature=DES/CBC Hash=Not Applicable
0 0 0 0 0 0 1 0	AUTH-2 Signature=DSA Hash=SHA-1
0 0 0 0 0 0 1 1	AUTH-3 Signature=EC/DSA Hash=SHA-1
0 0 0 0 0 1 0 0	AUTH-4 Signature=ESIGN Hash=MD5
0 0 0 0 0 1 0 1	AUTH-5 Signature=FEAL/CBC Hash=Not Applicable
0 0 0 0 0 1 1 0	AUTH-6 Signature=RSA

	Hash=MD5
1 x x x x x x x	User-defined Authentication Algorithm Group (the value of bits 1-7 is user defined)

OUI (Octets x.4-x.6)

Note: If the authentication algorithm group is “user defined,” then the IEEE OUI of the vendor/user that assigned the user-defined codepoint shall be included in these octets. Otherwise, these octets are not included.

Signature Algorithm Details (Octet x.7 etc.)

These optional octets include a binary number indicating the length of the signature algorithm details and the coding details for each signature algorithm. A list of signature algorithm details is provided in the description of the signature algorithm primitive (Section 6.2.3.4).

Hash Algorithm Details (Octet x.10 etc.)

These optional octets include a binary number indicating the length of the hash algorithm details and the coding details for each hash algorithm. A list of hash algorithm details is provided in the description of the hash algorithm primitive (Section 6.2.3.3).

6.2.3.8. Integrity Algorithm Profile Group

This octet group indicates a set of algorithms supporting integrity service.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	1	0	1	1	1	0	
Integrity Algorithm Group Identifier								x.1
Length of Integrity Algorithm Group Contents								x.2
Integrity Algorithm Group								x.3
OUI								x.4
OUI (continued)								x.5
OUI (continued)								x.6
1	0	1	1	0	1	1	0	
MAC Algorithm Details Identifier								x.7
Length of MAC Algorithm Details								x.8
MAC Algorithm Details								x.9 etc.
1	0	1	1	0	0	1	0	
Signature Algorithm Details Identifier								x.10 etc.
Length of Signature Algorithm Details								x.11 etc.
Signature Algorithm Details								x.12 etc.
1	0	1	1	1	0	0	0	
Key Exchange Algorithm Details Identifier								x.13 etc.
Length of Key Exchange Algorithm Details								x.14 etc.
Key Exchange Algorithm Details								x.15 etc.
1	0	1	1	1	0	1	0	
Key Update Algorithm Details Identifier								x.16 etc.
Length of Key Update Algorithm Details								x.17 etc.
Key Update Algorithm Details								x.18 etc.
1	0	1	1	0	1	0	0	
Hash Algorithm Details Identifier								x.19 etc.
Length of Hash Algorithm Details								x.20 etc.
Hash Algorithm Details								x.21 etc.

Length of Integrity Algorithm Group Contents (Octet x.2)

A binary number indicating the length in octets of the integrity algorithm group fields contained in x.3, x.4, etc.

Integrity Algorithm Group (Octet x.3)

8 7 6 5 4 3 2 1	**Meaning**
0 0 0 0 0 0 0 1	INTEG-1 MAC=DES/CBC Signature=RSA Key Exchange=RSA Key Update=MD5 Hash=MD5
0 0 0 0 0 0 1 0	INTEG-2 MAC=DES/CBC Signature=DSA Key Exchange=DH Key Update=SHA-1 Hash=SHA-1

0 0 0 0 0 0 1 1	INTEG-3 MAC=DES/CBC Signature=EC/DSA Key Exchange=EC/DH Key Update=SHA-1 Hash=SHA-1
0 0 0 0 0 1 0 0	INTEG-4 MAC=DES/CBC Signature=DES/CBC Key Exchange=DES/CBC Key Update=MD5 Hash=Not Applicable
0 0 0 0 0 1 0 1	INTEG-5 MAC=HMD5 Signature=RSA Key Exchange=RSA Key Update=MD5 Hash=MD5
0 0 0 0 0 1 1 0	INTEG-6 MAC=HMD5 Signature=DES/CBC Key Exchange=DES/CBC Key Update=MD5 Hash=Not Applicable
0 0 0 0 0 1 1 1	INTEG-7 MAC=HSHA-1 Signature=DSA Key Exchange=DH Key Update=SHA-1 Hash=SHA-1
0 0 0 0 1 0 0 0	INTEG-8 MAC=HSHA-1 Signature=EC/DSA Key Exchange=EC/DH Key Update=SHA-1 Hash=SHA-1
0 0 0 0 1 0 0 1	INTEG-9 MAC=HSHA-1 Signature=DES/CBC Key Exchange=DES/CBC Key Update=SHA-1 Hash=Not Applicable

0 0 0 0 1 0 1 0	INTEG-10 MAC=FEAL/CBC Signature=ESIGN Key Exchange=DH Key Update=MD5 Hash=MD5
0 0 0 0 1 0 1 1	INTEG-11 MAC=FEAL/CBC Signature=FEAL/CBC Key Exchange=FEAL/CBC Key Update=MD5 Hash=Not Applicable
1 x x x x x x x	User-defined Authentication Algorithm Group (the value of bits 1-7 is user defined)

OUI (Octets x.4-x.6)

Note: If the integrity algorithm group is “user defined,” then the IEEE OUI of the vendor/user which assigned the user-defined codepoint shall be included in these octets. Otherwise, these octets are not included.

MAC Algorithm Details (Octet x.7 etc.)

These optional octets include a binary number indicating the length of the MAC algorithm details and the coding details for each MAC algorithm. A list of MAC algorithm details is provided in (Section 6.2.3.2).

Signature Algorithm Details (Octet x.10 etc.)

These optional octets include a binary number indicating the length of the signature algorithm details and the coding details for each signature algorithm. A list of signature algorithm details is provided in the description of the signature algorithm primitive (Section 6.2.3.4).

Key Exchange Algorithm Details (Octet x.13 etc.)

These optional octets include a binary number indicating the length of the key exchange algorithm details and the coding details for each key exchange algorithm. A list of key exchange algorithm details is provided in the description of the key exchange algorithm primitive (Section 6.2.3.5).

Key Update Algorithm Details (Octet x.16 etc.)

These optional octets include a binary number indicating the length of the key update algorithm details and the coding details for each key update algorithm. A list of key update algorithm details is provided in the description of the key update algorithm primitive (Section 6.2.3.6).

Hash Algorithm Details (Octet x.19 etc.)

These optional octets include a binary number indicating the length of the hash algorithm details and the coding details for each hash algorithm. A list of hash algorithm details is provided in the description of the hash algorithm primitive (Section 6.2.3.3).

6.2.3.9. Confidentiality Algorithm Profile Group

This octet group indicates a set of algorithms supporting confidentiality service.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	0	1	1	0	0	0	0	
Confidentiality Algorithm Group Identifier								x.1
Length of Confidentiality Algorithm Group Contents								x.2
Confidentiality Algorithm Group								x.3
OUI								x.4
OUI (continued)								x.5
OUI (continued)								x.6
1	0	1	1	1	1	1	0	
Encryption Algorithm Details Identifier								x.7
Length of Encryption Algorithm Details								x.8
Encryption Algorithm Details								x.9 etc.
1	0	1	1	0	0	1	0	
Signature Algorithm Details Identifier								x.10 etc.
Length of Signature Algorithm Details								x.11 etc.
Signature Algorithm Details								x.12 etc.
1	0	1	1	1	0	0	0	
Key Exchange Algorithm Details Identifier								x.13 etc.
Length of Key Exchange Algorithm Details								x.14 etc.
Key Exchange Algorithm Details								x.15 etc.
1	0	1	1	1	0	1	0	
Key Update Algorithm Details Identifier								x.16 etc.
Length of Key Update Algorithm Details								x.17 etc.
Key Update Algorithm Details								x.18 etc.
1	0	1	1	0	0	1	0	
Hash Algorithm Details Identifier								x.19 etc.
Length of Hash Algorithm Details								x.20 etc.
Hash Algorithm Details								x.21 etc.

Length of Confidentiality Algorithm Group Contents (Octet x.2)

A binary number indicating the length in octets of the integrity algorithm group fields contained in x.3, x.4, etc.

Confidentiality Algorithm Group (Octet x.3)

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	CONF-1 Encryption=DES/CBC Signature=RSA Key Exchange=RSA Key Update=MD5 Hash=MD5

0 0 0 0 0 0 1 0	CONF-2 Encryption=DES/CBC Signature=DSA Key Exchange=DH Key Update=SHA-1 Hash=SHA-1
0 0 0 0 0 0 1 1	CONF-3 Encryption=DES/CBC Signature=EC/DSA Key Exchange=EC/DH Key Update=SHA-1 Hash=SHA-1
0 0 0 0 0 1 0 0	CONF-4 Encryption=DES/CBC Signature=DES/CBC Key Exchange=DES/CBC Key Update=MD5 Hash=Not Applicable
0 0 0 0 0 1 0 1	CONF-5 Encryption=DES/Counter Signature=RSA Key Exchange=RSA Key Update=MD5 Hash=MD5
0 0 0 0 0 1 1 0	CONF-6 Encryption=DES/Counter Signature=DSA Key Exchange=DH Key Update=SHA-1 Hash=SHA-1
0 0 0 0 0 1 1 1	CONF-7 Encryption=DES/Counter Signature=EC/DSA Key Exchange=EC/DH Key Update=SHA-1 Hash=SHA-1
0 0 0 0 1 0 0 0	CONF-8 Encryption=DES/Counter Signature=DES /CBC Key Exchange=DES /CBC Key Update=MD5 Hash=Not Applicable

0 0 0 0 1 0 0 1	CONF-9 Encryption=Triple-DES/CBC Signature=RSA Key Exchange=RSA Key Update=MD5 Hash=MD5
0 0 0 0 1 0 1 0	CONF-10 Encryption=Triple-DES/CBC Signature=DSA Key Exchange=DH Key Update=SHA-1 Hash=SHA-1
0 0 0 0 1 0 1 1	CONF-11 Encryption=Triple-DES/CBC Signature=EC/DSA Key Exchange=EC/DH Key Update=SHA-1 Hash=SHA-1
0 0 0 0 1 1 0 0	CONF-12 Encryption=Triple-DES/CBC Signature= Triple-DES/CBC Key Exchange= Triple-DES/CBC Key Update=MD5 Hash=Not Applicable
0 0 0 0 1 1 0 1	CONF-13 Encryption=Triple-DES/Counter Signature=RSA Key Exchange=RSA Key Update=MD5 Hash=MD5
0 0 0 0 1 1 1 0	CONF-14 Encryption=Triple-DES/Counter Signature=DSA Key Exchange=DH Key Update=SHA-1 Hash=SHA-1
0 0 0 0 1 1 1 1	CONF-15 Encryption=Triple-DES/Counter Signature=EC/DSA Key Exchange=EC/DH Key Update=SHA-1 Hash=SHA-1

0 0 0 1 0 0 0 0	CONF-16 Encryption=Triple-DES/Counter Signature= Triple-DES/CBC Key Exchange= Triple-DES/CBC Key Update=MD5 Hash=Not Applicable
0 0 0 1 0 0 0 1	CONF-17 Encryption=FEAL/CBC Signature=ESIGN Key Exchange=DH Key Update=MD5 Hash=MD5
0 0 0 1 0 0 1 0	CONF-18 Ecryption=FEAL/CBC Signature=FEAL/CBC Key Exchange=FEAL/CBC Key Update=MD5 Hash=Not Applicable
0 0 0 1 0 0 1 1	CONF-19 Encryption=FEAL/Counter Signature=ESIGN Key Exchange=DH Key Update=MD5 Hash=MD5
0 0 0 1 0 1 0 0	CONF-20 Encryption=FEAL/Counter Signature=FEAL/CBC Key Exchange=FEAL/CBC Key Update=MD5 Hash=Not Applicable
1 x x x x x x x	User-defined Authentication Algorithm Group (the value of bits 1-7 is user defined)

OUI (Octets x.4-x.6)

Note: If the confidentiality algorithm group is “user defined,” then the IEEE OUI of the vendor/user which assigned the user-defined codepoint shall be included in these octets. Otherwise, these octets are not included.

Encryption Algorithm Details (Octet x.7 etc.)

These optional octets include a binary number indicating the length of the encryption algorithm details and the coding details for each encryption algorithm. A list of encryption algorithm details is provided in data confidentiality algorithm primitive (Section 6.2.3.1).

Signature Algorithm Details (Octet x.10 etc.)

These optional octets include a binary number indicating the length of the signature algorithm details and the coding details for each signature algorithm. A list of signature algorithm details is provided in the description of the signature algorithm primitive (Section 6.2.3.4).

Key Exchange Algorithm Details (Octet x.13 etc.)

These optional octets include a binary number indicating the length of the key exchange algorithm details and the coding details for each key exchange algorithm. A list of key exchange algorithm details is provided in the description of the key exchange algorithm primitive (Section 6.2.3.5).

Key Update Algorithm Details (Octet x.16 etc.)

These optional octets include a binary number indicating the length of the key update algorithm details and the coding details for each key update algorithm. A list of key update algorithm details is provided in the description of the key update algorithm primitive (Section 6.2.3.6).

Hash Algorithm Details (Octet x.19 etc.)

These optional octets include a binary number indicating the length of the hash algorithm details and the coding details for each hash algorithm. A list of hash algorithm details is provided in the description of the hash algorithm primitive (Section 6.2.3.3).

6.2.4. Confidential Section

The security confidentiality section provides for the exchange of confidential information in the SME. This optional section provides an opportunity to include encrypted data in an SAS.

Bits								
8	7	6	5	4	3	2	1	Octets
1	1	0	0	0	0	0	0	
Confidential Section Identifier								x.1
Length of Confidential Section								x.2
Length of Confidential Section (cont.)								x.3
Confidential Data								x.4 etc

Length of Confidential Section (Octet x.2, x.3)

This 16-bit octet group provides the length of the Confidential Section. This is the length of the ciphertext, which may not be the same as the plaintext.

Confidential Data (Octet x.4 etc)

This Octet group contains encrypted data that requires decryption in order to be meaningful to the security agent. In the SME, this is the encryption of the Confidential Parameters primitive (defined below), using the key exchange algorithm referenced by the SAS identifier.

6.2.4.1. Confidential Parameters

The Security Message Exchange Protocol supports the exchange of confidentiality parameters. These SAS primitives can be consolidated in a single Confidential Parameters section, or can be included individually. This is the original confidential parameters section.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	1	0	0	0	1	0	0	
Confidential Parameters Identifier								x.1
Master Key								x.2 etc
First Data Confidentiality Session Key								x.3 etc
First Data Integrity Session Key								x.4 etc

Master Key (Octet x.2 etc. Optional)

This octet group provides the Master Key for the cryptographic services. Further details can be found in Section 5.2.3.

First Data Confidentiality Session Key (Octet x.3 etc. Optional)

This field contains the initial Data Confidentiality Session Key. Further details can be found in Section 5.2.4.

First Data Integrity Session Key (Octet x.4 etc. Optional)

This field contains the initial Data Integrity Session Key. Further details can be found in Section 5.2.4.

6.2.4.2. Master Key

This octet group contains the key used to encrypt subsequent session key.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	1	0	0	1	0	0	0	
Master Key Identifier								x.1
Length of Master Key								x.2
Master Key Value								x.3 etc.

Master Key Length (Octet x.2)

A binary number indicating the length in octets of the Master Key value contained in octets x.3, etc.

Master Key Value (Octet x.3 etc.)

This field contains the binary encoding of the Master Key value, with bit 8 of the first octet being the most significant bit, and bit 1 of the last octet being the least significant bit. The Master Key length must be an integer multiple of 8 bits.

6.2.4.3. First Data Confidentiality Session Key

This octet group contains the first session key to be used to provide the data confidentiality service.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	1	0	0	1	0	1	0	
First Data Confidentiality Session Key Identifier								x.1
Length of First Data Confidentiality Session Key								x.2
First Data Confidentiality Session Key Value								x.3 etc.

First Data Confidentiality Session Key Length (Octet x.2)

A binary number indicating the length in octets of the First Data Confidentiality Session Key Value contained in octets x.3, etc.

First Data Confidentiality Session Key Value (Octet x.3 etc.)

This field contains the binary encoding of The First Data Confidentiality Session Key Value, with bit 8 of the first octet being the most significant bit, and bit 1 of the last octet being the least significant bit. The First Data Confidentiality Session Key length must be an integer multiple of 8 bits. Algorithm-specific details for this field are as follows:

⇒ When the data confidentiality algorithm is DES with 56-bit effective key, the 64-bit (8-byte) key value (which includes parity) is encoded in binary in this field.

⇒ When the data confidentiality algorithm is Triple-DES with 112-bit effective key, the 128-bit (16-byte) key value (which includes parity) is encoded in binary in this field as follows: The “first key” is encoded in binary in octets 21.2 through 21.9 and the “second key” is encoded in binary in octets 21.10 through 21.17 of this field.

⇒ When the data confidentiality algorithm is FEAL with 64-bit effective key, the 64-bit (8-byte) key is encoded in binary in this field.

6.2.4.4. First Data Integrity Session Key

This octet group contains the first session key to be used to provide the data integrity service.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	1	0	0	1	1	0	0	
First Data Integrity Session Key Identifier								x.1
Length of First Data Integrity Session Key								x.2
First Data Integrity Session Key Value								x.3 etc.

First Data Integrity Session Key Length (Octet x.2)

A binary number indicating the length in octets of the First Data Integrity Session Key value contained in octets x.3, etc.

First Data Integrity Session Key Value (Octet x.3 etc.)

This field contains the binary encoding of the First Data Integrity Session Key value, with bit 8 of the first octet being the most significant bit, and bit 1 of the last octet being the least significant bit. The First Data Integrity Session Key length must be an integer multiple of 8 bits.

6.2.5. Authentication Section

This octet group specifies the SAS Authentication Section. This section is intended to support authentication as specified in the Security Message Exchange Protocol. However, the design provides a generic authentication service that can be applied to any part of the SSIE. All fields of this section are optional.

The specified digital signature type determines the digital signature's scope. When used in the Security Message Exchange Protocol, the digital signature must be a Security Message Exchange Digital Signature.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	1	0	1	0	0	0	0	
Authentication Section Identifier								x.1
Initiator Random Number								x.2 etc
Responder Random Number								x.3 etc
Time Variant Time Stamp								x.4 etc
Credentials								x.5 etc
Digital Signature								x.6 etc

Initiator Random Number (Octet x.2 etc Optional)

This section provides the initiator supplied nonce, if provided in the exchange. Further details on the format and use of this field are found below.

Responder Random Number (Octet x.3 etc Optional)

This section provides the responder supplied nonce, if provided in the exchange. Further details on the format and use of this field are found below.

Time Variant Time Stamp (Octet x.4 etc Optional)

This section specifies a unique time stamp/sequence number that will provide replay protection. Further details on the format and use of this field are found below.

Credentials (Octet x.5 etc Optional.)

This section is employed when credential presentation or exchange is required in the Security Message Exchange. Further details on using this field are found below.

Digital Signature (Octet x.6 etc)

This section is the Digital Signature. Further details on using this field are found below.

6.2.5.1. Initiator Random Number (Nonce)

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	1	0	1	0	1	0	0	
Initiator Random Number Identifier								x.1
Initiator Random Number Value								x.2
Initiator Random Number Value (cont.)								x.3
Initiator Random Number Value (cont.)								x.4
Initiator Random Number Value (cont.)								x.5

Initiator Random Number Identifier (Octet x.2-5)

This field is a 32 bit binary random number.

6.2.5.2. Responder Random Number (Nonce)

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	1	0	1	0	1	0	1	
Responder Random Number Identifier								x.1
Responder Random Number Value								x.2
Responder Random Number Value (cont.)								x.3
Responder Random Number Value (cont.)								x.4
Responder Random Number Value (cont.)								x.5

Responder Random Number Identifier (Octet x.2-5)

This field is a 32 bit binary random number.

6.2.5.3. Time-Variant Time Stamp

This field provides a label that can be used as a unique identifier for a given security IE, such as an IE used in an authentication exchange. This field consists of two 32 bit (four octet) integers, one for the time stamp (with one second resolution), and the other for a monotonically increasing sequence number. The timestamp shall be encoded in Octets x.2 - x.5, and the sequence number shall be encoded in Octets x.6 - x.9. Each of these integers are encoded as 32 bit unsigned binary integers, with bit 8 of the first octet being the most significant bit, and bit 1 of the fourth octet being the least significant bit. For further details on using this field, consult Sections 5.1 and 5.1.1.2.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	1	0	1	0	1	1	0	
Time-Variant Time Stamp Identifier								x.1
Time Stamp Value								x.2
Time Stamp Value (cont.)								x.3
Time Stamp Value (cont.)								x.4
Time Stamp Value (cont.)								x.5
Sequence Number								x.6
Sequence Number (cont.)								x.7
Sequence Number (cont.)								x.8
Sequence Number (cont.)								x.9

Time Stamp Value (Octet x.2-5)

The time stamp value is a 4-byte coordinated universal time, the Greenwich Mean Time (GMT) at which the signature was generated. This value is the binary encoding of the number of seconds since 00:00:00 GMT on January 1, 1970 (same as UNIX time). This is encoded in binary in octets x.2 through x.5.

Sequence Number (Octet x.6-9)

The sequence number is a 4-octet binary number that is incremented with each authentication flow that is sent to the same destination with the same time stamp value. When the time stamp value changes, the sequence number is reset to 0. The sequence number value is encoded in binary in octets x.6 through x.9.

6.2.5.4. Credentials

Certificate exchange is an optional feature of the Security Message Exchange Protocol, and is in general, a credential presentation feature. User defined credential types may contain a complete set of security services, and as such, additional authentication sections, such as time stamps, random numbers, and signatures, may not be required in the authentication process.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	1	0	1	1	0	0	0	
Credential Identifier								x.1
Length of Credential Contents								x.2
Length of Credential Contents (cont.)								x.3
Credential Type								x.4
Credential Value								x.5 etc.

Length of Credential Contents (Octets x.2 - 3):

The overall length of the SSIE limits maximum length of the credential, when signaling channel is used for certificate exchange

Credential Type (Octet x.3):

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	X.509 Certificate V1
0	0	0	0	0	0	1	0	X.509 Certificate V2
0	0	0	0	0	0	1	1	X.509 Certificate V3
1	x	x	x	x	x	x	x	User defined credential type. (the value of bits 1-7 is user-defined)

Credential Value (Octet x.4 etc.):

This field contains the binary encoding of the credentials value, the format of which conforms to the type specified in Octet x.3.

6.2.5.5. Security Message Exchange Digital Signature

This octet group contains the signature value computed over the objects required by the Security Exchange Message Protocol. The signature algorithm is specified in other sections of the SSIE SAS. The length is the length of the entire octet group, excluding the identifier and the length octets.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	1	0	1	1	0	1	0	
Security Message Exchange Digital Signature Identifier								x.1
Security Message Exchange Digital Signature Length								x.2
Digital Signature Value								x.3 etc.

Security Message Exchange Digital Signature Length (Octet x.2)

A binary number indicating the length in octets of the digital signature value contained in octets x.3 etc.

Security Message Exchange Digital Signature Value (Octet x.3 etc.)

This field contains the binary encoding of the signature value. This is signature algorithm specific and both security agents in the SME must know the signature algorithm used. Further details on constructing the digital signature buffer and calculating this field are found in the following sections of this document.

Algorithm-specific details for the signature are described below:

⇒ When the signature algorithm is RSA, this field contains the octet string representation of the digital signature.

⇒ When the signature algorithm is DSA, the two digital signature components R and S are encoded as octet strings in this field as follows:

DSA "R" parameter identifier (Octet x.3a):
This octet is the identifier for the R component of the DSA digital signature.

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	R Parameter

DSA "R" parameter length (Octet x.3b):
This octet specifies the length of R in octets.

DSA "R" parameter value (Octet x.3c etc.):
This field contains the octet string representation of R.

DSA "S" parameter identifier (Octet x.4a):
This octet is the identifier for the S component of the DSA digital signature.

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	1	0	S Parameter

DSA "S" parameter length (Octet x.4b):
This octet specifies the length of S in octets.

DSA "S" parameter value (Octet x.4c etc.):
This field contains the octet string representation of S.

⇒ When the signature algorithm is ESIGN, the digital signature components *S* are encoded as octet strings in this field as follows:

ESIGN "S" parameter identifier (Octet x.3a):
This octet is the identifier for the ESIGN digital signature S.

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	S Parameter

ESIGN "S" parameter length (Octet x.3b):
This octet specifies the length of S in octets.

ESIGN "S" parameter value (Octet x.3c etc.):
This field contains the octet string representation of S.

⇒ When the signature algorithm is Elliptic curve Digital Signature Algorithm (ECDSA-like), the two integers *r* and *s* are encoded as octet strings in this field as follows (cf. Section 6.9.3):

ECSDA-like "*r*" parameter identifier (Octet x.3a):
This octet is the identifier for the *r* component of the ECDSA-like digital signature.

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	0	1	*r* Parameter

ECSDA-like "*r*" parameter length (Octet x.3b):
This octet specifies the length of *r* in octets.

ECDSA-like "*r*" parameter value (Octet x.3c etc.):
This field contains the octet string representation of *r*.

ECSDA-like "s" parameter identifier (Octet x.4a):
This octet is the identifier for the *s* component of the ECDSA-like digital signature.

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Meaning**
0	0	0	0	0	0	1	0	*s* Parameter

ECSDA-like "s" parameter length (Octet x.4b):
This octet specifies the length of *s* in octets.

ECDSA-like "*s*" parameter value (Octet x.4c etc.):
This field contains the octet string representation of *s*.

6.2.5.5.1. Digital Signature Buffer–Two-Way Exchange Protocol

The contents and structure of the digital signature buffer for flows 1 and 2 of the Two-Way Security Message Exchange Protocol are defined below.

6.2.5.5.1.1. FLOW1-2WE

For FLOW1-2WE, the SME Digital Signature primitive contains the digital signature as defined in Section 5.1.1.2, and reproduced below:

$$Sig_{Ka}(Hash(A, B, T_a, R_a, SecOpt, \{ConfPar_a\}))$$

In order to calculate this signature, the hash value is calculated across the buffer shown in **Error! Reference source not found.**. Bit ordering within each octet is identical to the bit ordering within the respective primitives of the Security Services Information Element. In addition, this buffer is padded according to the appropriate digital signature algorithm procedures.

Table 17: Digital Signature Buffer for FLOW1-2WE.

Symbol	Description of Primitive	Octets	Refer to Section
A	Initiator Distinguished Name	X.1 - X.4, etc.	6.2.1.1
B	Responder Distinguished Name	X.1 - X.4, etc	6.2.1.2
T_a	Time-Variant Timestamp	X.1 - X.9	6.2.5.3
R_a	Initiator Random Number	X.1 - X.5	6.2.5.1
SecOpt	Security Service Specification Section	X.1, X.2, etc., X.3, etc.	6.2.2
{$ConfPar_a$}	Confidential Parameters	X.1, X.2, etc., X.3, etc., X.4, etc	6.2.4.1 (Note)

Note: If present, this field is a *plain-text* field. That is, this field contains the confidential parameters as they exist either before encryption (if the digital signature is generated by the sending agent), or after decryption (if the digital signature is validated by the receiving agent). For this reason, this field should be erased immediately after the hash function is computed.

6.2.5.5.1.2. FLOW2-2WE

For FLOW2-2WE, the SME Digital Signature primitive contains the digital signature as defined in Section 5.1.1.2, and reproduced below:

$$Sig_{Kb}(Hash(A, B, R_a, \{ConfPar_b\}))$$

In order to calculate this signature, the hash value is calculated across the buffer shown in **Error! Reference source not found.**. Bit ordering within each octet is identical to the bit ordering within the respective primitives of the Security Services Information Element. In addition, this buffer is padded according to the appropriate digital signature algorithm procedures.

Table 18: Digital Signature Buffer for FLOW2-2WE.

Symbol	Description of Primitive	Octets	Refer to Section
A	Initiator Distinguished Name	X.1 - X.4, etc.	6.2.1.1
B	Responder Distinguished Name	X.1 - X.4, etc	6.2.1.2
R_a	Initiator Random Number	X.1 - X.5	6.2.5.1 (Note 1)
{$ConfPar_b$}	Confidential Parameters	X.1, X.2, etc., X.3, etc., X.4, etc	6.2.4.1 (Note 2)

Note 1: These octets are received from the initiator in FLOW1-2WE.

Note 2: If present, this field is a *plain-text* field. That is, this field contains the confidential parameters as they exist either before encryption (if the digital signature is generated by the sending agent), or after decryption (if the digital signature is validated by the receiving agent). For this reason, this field should be erased immediately after the hash function is computed.

6.2.5.5.2. Digital Signature Buffer–Three-Way Exchange Protocol

The contents and structure of the digital signature buffer for flows 2 and 3 of the Three-Way Security Message Exchange Protocol are defined below.

6.2.5.5.2.1. FLOW2-3WE

For FLOW2-3WE, the SME Digital Signature primitive contains the digital signature as defined in Section 5.1.1.1, and reproduced below:

$$Sig_{Kb}(Hash(A, B, R_a, R_b, SecNeg_a, SecNeg_b, \{ConfPar_b\}))$$

In order to calculate this signature, the hash value is calculated across the buffer shown in Table 19: Digital Signature Buffer for FLOW2-3WE.. Bit ordering within each octet is identical to the bit ordering within the respective primitives of the Security Services Information Element. In addition, this buffer is padded according to the appropriate digital signature algorithm procedures.

Table 19: Digital Signature Buffer for FLOW2-3WE.

Symbol	Description of Primitive	Octets	Refer to Section
A	Initiator Distinguished Name	X.1 - X.4, etc.	6.2.1.1
B	Responder Distinguished Name	X.1 - X.4, etc	6.2.1.2
R_a	Initiator Random Number	X.1 - X.5	6.2.5.1 (Note 1)
R_b	Responder Random Number	X.1 - X.5	6.2.5.2
$SecNeg_a$	Security Service Specification Section	X.1, X.2, etc., X.3, etc.	6.2.2 (Note 2)
$SecNeg_b$	Security Service Specification Section	X.1, X.2, etc., X.3, etc.	6.2.2
$\{ConfPar_b\}$	Confidential Parameters	X.1, X.2, etc., X.3, etc., X.4, etc	6.2.4.1 (Note 3)

Note 1: These octets are received from the initiator in FLOW1-3WE.

Note 2: These octets are received from the initiator in FLOW1-3WE, and are not sent in FLOW2-3WE.

Note 3: If present, this field is a *plain-text* field. That is, this field contains the confidential parameters as they exist either before encryption (if the digital signature is generated by the sending agent), or after decryption (if the digital signature is validated by the receiving agent). For this reason, this field should be erased immediately after the hash function is computed.

6.2.5.5.2.2. FLOW3-3WE

For FLOW3-3WE, the SME Digital Signature primitive contains the SME digital signature as defined in Section 5.1.1.1, and reproduced below:

$$Sig_{Ka}(Hash(A, B, R_b, \{ConfPar_a\}))$$

In order to calculate this signature, the hash value is calculated across the buffer shown in Table 20. Bit ordering within each octet is identical to the bit ordering within the respective primitives of the Security Services Information Element. In addition, this buffer is padded according to the appropriate digital signature algorithm procedures.

Table 20: Digital Signature Buffer for FLOW3-3WE.

Symbol	Description of Primitive	Octets	Refer to Section
A	Initiator Distinguished Name	X.1 - X.4, etc.	6.2.1.1
B	Responder Distinguished Name	X.1 - X.4, etc	6.2.1.2
R_b	Responder Random Number	X.1 - X.5	6.2.5.2 (Note 1)
$\{ConfPar_a\}$	Confidential Parameters	X.1, X.2, etc., X.3, etc., X.4, etc	6.2.4.1 (Note 2)

Note 1: These octets are received from the responder in FLOW2-3WE.

Note 2: If present, this field is a *plain-text* field. That is, this field contains the confidential parameters as they exist either before encryption (if the digital signature is generated by the sending agent), or after decryption (if the digital signature is validated by the receiving agent). For this reason, this field should be erased immediately after the hash function is computed.

6.2.5.6. SAS Digital Signature

This octet group contains a digital signature computed over an SAS. The digital signature value is computed over the entire contents of the SAS, up to, but not including, this SAS primitive. An SAS can contain any number of digital signatures. The context of all signatures in a particular SAS, i.e., the signature format, algorithm, and keys needed to calculate the signature, are identical, as the scope of the context of each signature is defined by the SAS.

Bits								
8	7	6	5	4	3	2	1	Octet(s)
1	1	0	1	1	1	0	0	
SAS Digital Signature Identifier								x.1
Length of Digital Signature Contents								x.2
Length of Digital Signature Contents (cont.)								x.3
Digital Signature Value								x.4 etc.

Digital Signature Length (Octet x.2-3)

A binary number indicating the length in octets of the digital signature value contained in octet group x.4, etc.

Digital Signature Value (Octet x.4 etc.)

This field contains the binary encoding of the signature value. The signature is computed over the encoded contents of this Security Association Section (excluding this digital signature octet group). The contents of this SAS must remain intact as this SSIE (and SAS) travels through the network. That is, the network must not change or reorder the contents of this SAS.

The algorithm-specific details for the SAS Digital Signature Value are identical to those for the SME Digital Signature Value, found in Section 6.2.5.5.

6.3. DES and FEAL Encryption Bit Ordering

6.3.1. Bit Ordering for Data

The ATM layer encryption process is used to encrypt the 48 payload bytes contained in user cells. The 48 payload bytes are numbered 1 to 48, and within a byte the bits are numbered 8 to 1 (from left to right), as defined in the UNI Specification [2]. The DES standard [6] and FEAL standard [32], [33], define 64 bit input and output data blocks, with the bits numbered 1 to 64 (from left to right). Each ATM cell payload contains 384 bits, and hence must be divided into 6 equal blocks of 64 bits to perform the encryption.

The mapping of bits from the cell payload into the DES or FEAL encryption algorithm, and then back to the payload, shall be as shown in Table 21.

Table 21 DES and FEAL Encryption Algorithm Data Bit Ordering

Cell Bit Position	8	7	6	5	4	3	2	1	•••	8	7	6	5	4	3	2	1
Payload Byte Number	1								•••	8							
	9								•••	16							
	17								•••	24							
	25								•••	32							
	33								•••	40							
	41								•••	48							
				⇓					⇓				⇓				
DES/ FEAL Bit Position	1	2	3	4	5	6	7	8	•••	57	58	59	60	61	62	63	64

6.3.1.1. The Effect of the Mode of Operation

The mapping indicated above applies directly to the ECB mode of operation. For CBC mode it is intended that the left most cipher text output bit (DES/FEAL bit position 1) be *xor*ed with the leftmost payload bit (bit 8 of payload bytes 1, 9, ... 41). The result is sent back into DES/FEAL in the left most input bit position (DES/FEAL bit position 1), and so on. For the Counter Mode it is intended that the leftmost output bit from

the DES/FEAL algorithm (DES/FEAL bit position 1) be *xor*ed with the leftmost payload bit (bit 8 of payload bytes 1, 9, ... 41), with the result put back in the leftmost payload bit, and so on. In addition, bit 63 of the state vector is input into the DES/FEAL bit position 1, bit 62 of the state vector into DES/FEAL bit position 2, and so on.

6.3.2. Bit Ordering for Keys

Encrypted DES/FEAL session keys are transported in the Security Confidential Parameters Information Element during secure call setup, and in Session Key Exchange OAM cells for key updates. In both cases, only the 64 bit key is carried when using FEAL. When using DES, the 64 bit or 128 bit keys for DES or Triple DES are carried. The use of the DES parity bits after transport is implementation specific, and is not defined here.

The mapping of DES/FEAL keys from either an Information Element, or an SKE OAM cell, shall be as shown in Table 22.

Table 22: DES/FEAL Encryption Algorithm Key Bit Ordering.

Bit Position	8	7	6	5	4	3	2	1	•••	8	7	6	5	4	3	2	1
Byte Number	*i*								•••	i + 7							
				⇓					⇓				⇓				
DES Bit Position	1	2	3	4	5	6	7	8	•••	57	58	59	60	61	62	63	64

Note: Byte number *i* represents the first byte of the key in either an information element or OAM cell.

6.4. The Counter Mode of Operation

6.4.1. Purpose

This version of the counter mode is intended for use with any 64-bit block encryption algorithm.

Block (or codebook) encryption algorithms are rarely used in a stand-alone fashion because they have several security weaknesses. The most serious is that patterns in the plain text are not adequately concealed in the cipher text. In addition it is possible to remove, duplicate, and interchange cipher text without detection. Typically some form of a feedback mechanism, along with an exclusive OR operation, is used to overcome these problems. The feedback information must be retained in memory at both the encryptor and decryptor, on a per connection basis.

The counter mode is well suited for use with ATM encryption. It provides an efficient implementation allowing it to scale to higher data rates than other modes, such as cipher block chaining. With the counter mode the feedback information can be updated and restored to memory in parallel with the encryption algorithm processing. Second, and perhaps more important, it allows multiple blocks of plain text to be encrypted in parallel. For example, an entire 48-byte cell payload can be encrypted in one operation using 6 copies of the encryption algorithm (with a 64-bit block size). Because each of the 6 payload blocks is given a unique segment number, implementations that use the parallel encryption approach are completely interoperable with those that don't.

6.4.2. Description

In counter mode both the encryptor and decryptor are synchronously producing identical key stream, based on each end having identical State Vectors (SVs) feeding like-keyed encryption algorithms. This state vector, which is described in more detail in Section 6.4.6, consists of several fields that contain the results from various counters and a Linear Recurring Sequence. This ensures that unique key stream values are used for each block that is encrypted.

Once the key stream value is determined, the encryptor XORs the key stream with the plain text to produce cipher text. Similarly, the decryptor XORs the key stream with the cipher text to recover the original plain text. A block diagram of this process is shown in Figure 38. Note that both the encryptor and decryptor use the encryption algorithm to "encrypt" the SV.

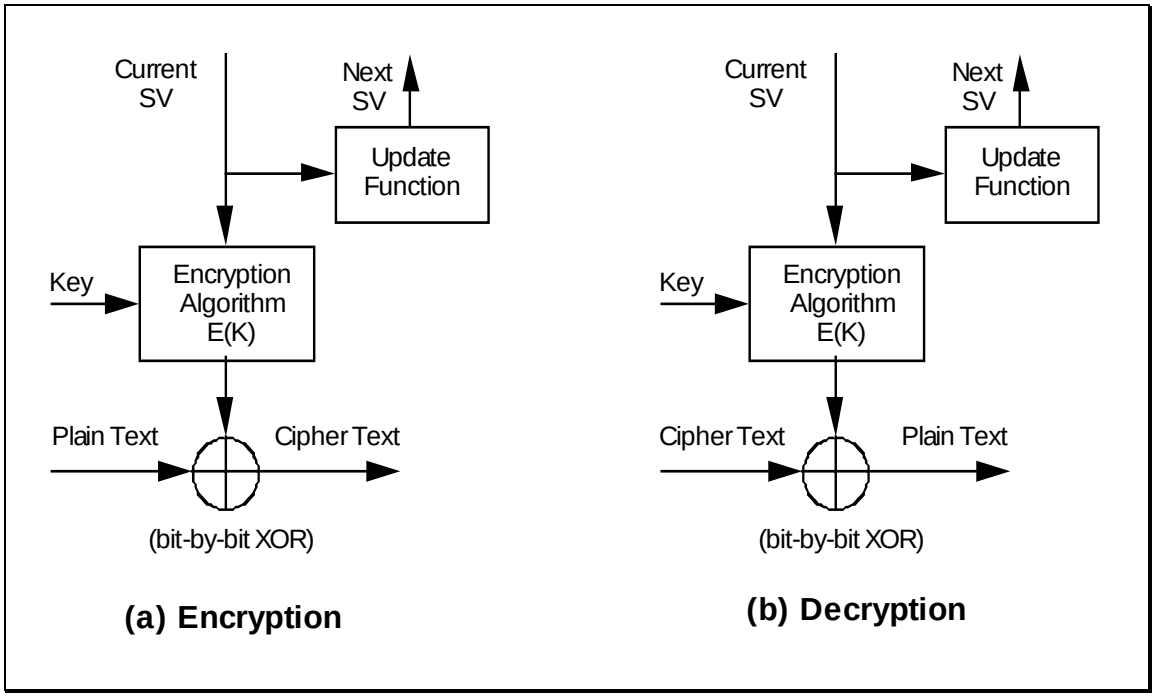


Figure 38. Counter Mode Block Diagram.

The SV must be changed on every new encryption; otherwise the generator will produce identical key streams. The definition of the SV, and the associated processing to update it, are described in later sections.

6.4.3. Properties

With the counter mode, cell loss in the ATM network causes the decryptor to lose cryptographic synchronization. The mechanisms for recovering synchronization are described in the following sections. The counter mode exhibits no error extension however, as a bit error in the cipher text produces only a single bit error in the plain text.

6.4.4. Cryptographic Synchronization

Cryptographic synchronization is maintained on a per connection basis. To recover from synchronization loss the decryptor must be provided with the same State Vector (SV) that the encryptor is using, for a particular connection. To accomplish this the encryptor transmits a resync message containing the new SV.

The encryptor begins using the new SV on the very next cell to occur on that connection. The decryptor, upon receipt of the resync message, loads the new SV into its internal memory. Cell sequence integrity ensures that the next cell to arrive at the decryptor, on the connection in question, will correspond with the new SV.

6.4.4.1. SKC OAM Cell

The session key changeover cell defined in Section 5.1.7.2.2 shall be used for both key changeover and resynchronization. A key changeover occurs when the Bank ID and key number fields in the SKC OAM cell are different from those currently in use. For a resync cell they are the same. For the counter mode, it is permissible simply to change the Bank ID of the periodic resync SKC OAM cell processing to perform a key changeover (after the corresponding SKE processing is completed). If this approach is taken, no additional SKC OAM cells are required.

6.4.4.2. Encryptor Processing of SKC OAM Cells

It is the job of the encryptor to insert the SKC OAM cells periodically, at the rate described in Section 7.1. This applies to both ends of a full duplex connection. A new SV is computed as follows (see Section 6.4.6 for a definition of the fields in the SV): the jump number is incremented, the I/R is set accordingly, the SEQ and SEG fields are set to all zeros. The LFSR is set to the default preset value, 05A5A hex. The new SV is loaded into internal memory and may also be loaded into the payload of the SKC OAM cell, although only the jump number is required. The encryptor shall set the other fields in the SV either to their default preset values or to all zeros. The CRC-10 is computed and inserted at the end of the cell. The new SV is used to encrypt the next incoming cell on that connection.

For the case of a key changeover, the encryptor may reset the jump number to any value, although zero is recommended. If multiple key changeover SKC OAM cells are transmitted, the jump number is reset for the first copy, and then incremented thereafter.

6.4.4.3. Decryptor Processing of SKC OAM Cells

Upon receipt of a SKC OAM cell the decryptor verifies that the CRC-10 is correct, and discards the cell if it is not. It also verifies that the jump number (see Section 6.4.6 for a definition of the fields in the SV) is greater than the current jump number in use on that connection, and discards the cell if it is not. An exception to this rule is made in the case of a key changeover. When the Bank ID and key number of an SKC OAM cell are different from those currently in use, any jump number is acceptable. If either the CRC-10 or jump number test fails, the decryptor discontinues processing the OAM cell. If both tests pass, it continues as follows. The decryptor sets the LFSR to the default preset value, 05A5A HEX. The I/R bit is unchanged, and the SEQ and SEG fields are set to all zeros. The decryptor loads the new jump number into its internal memory and begins using the new SV on the next incoming cell of that connection. The decryptor discards the SKC OAM cell, removing it from the cell stream.

6.4.5. Cell Loss Insensitivity

In some applications the performance of secure connections, in the presence of cell loss, may not be satisfactory. AAL information is used to improve this performance. The AAL type of the connection is obtained during call establishment.

6.4.5.1. AAL1 and AAL3/4 Connections

Cells on AAL1 or AAL3/4 connections contain a 3 or 4 bit cell sequence number, respectively. This information is extracted from the cell payload and used as part of the SV. The LFSR is only updated when

the sequence number of the received cell is less than or equal to the sequence number of the previous cell on that connection (i.e., the sequence number has wrapped around). It is held constant otherwise. Other than those cells where the sequence number wraps around, the only part of the SV that changes is the sequence number extracted from the cell. This gives immunity to cell loss (up to the maximum size provided by the sequence number minus one, 7 for AAL1 and 15 for AAL3/4), by maintaining SV synchronization in the event of dropped cells.

The sequence number must be available from the cipher text cell for this scheme to work. Therefore, this field is masked (i.e. excluded) from the XOR function that otherwise encrypts or decrypts the payload with key stream.

6.4.5.2. AAL5 Connections

AAL5 uses a bit in the PTI field (i.e., PTI = 001 or 011) of the ATM cell header to indicate end-of-message cells. After encrypting or decrypting an end-of-message cell, the LFSR is preset to 05A5A hex, and the jump number is incremented. Thus, the process automatically resynchronizes at the beginning of each new packet, provided the end-of-message cell itself is not lost. Even though the cells after the lost cell(s) are decrypted incorrectly, no additional negative effects are seen by the user, since the AAL5 receive processing discards the entire packet anyway. This process improves the performance of the counter mode to be roughly identical to the self-synchronizing modes. If an end-of-message cell is lost, the SKC OAM cell will eventually resynchronize the decryptor.

6.4.6. State Vector (SV) Definition

The format of the 64 bit SV is shown in Figure 39. The definition of each of the fields follows.

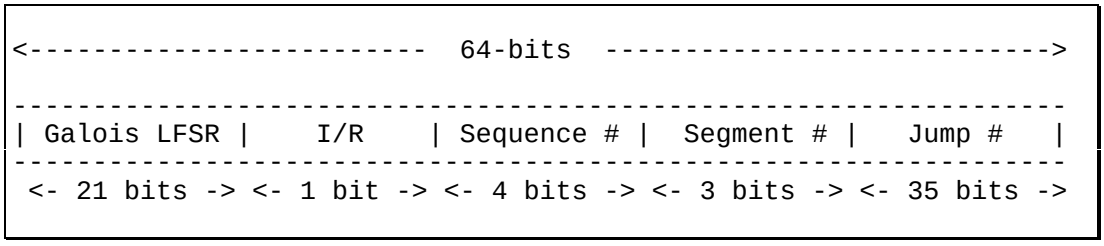


Figure 39. 64 Bit State Vector (SV) Definition.

6.4.6.1. Galois Linear Feedback Shift Register (LFSR)

The LFSR is a linear non-repeating sequence that is stepped once per cell or once per sequence of cells depending on AAL type. The LFSR is preset back to its starting value each time the connection is resynchronized. The maximum resync interval and the cell rate determine the 21-bit size of the LFSR. At 622 Mbps the maximum interval between resyncs should be 1 second. A full bandwidth connection at 622 Mbps produces roughly 1.4 million cells/sec. The LFSR, when clocked at the cell rate, must produce a non-repeating sequence longer than this. A 21-bit LFSR produces 2E21-1 = 2,097,151 unique values, satisfying this condition. The LFSR on a full bandwidth 622 Mbps connection would not repeat for 1.48 sec. To prevent LFSR wrap-around on higher bandwidth connections, the resync period should be proportionally shortened such that the number of cells sent during the maximum resync period is less than 2,097,151.

An LFSR must implement a primitive polynomial mod 2 in order to maximize its non-repeating sequence. Non-primitive polynomials will cause the LFSR to repeat at shorter intervals. From Schneier [35] Table 16.2, the polynomial xE21 + xE2 + 1 is selected and a Galois implementation of this polynomial is specified.

The Galois implementation is as shown in Figure 40. Bit 0 is the leftmost bit and the shift is to the left.

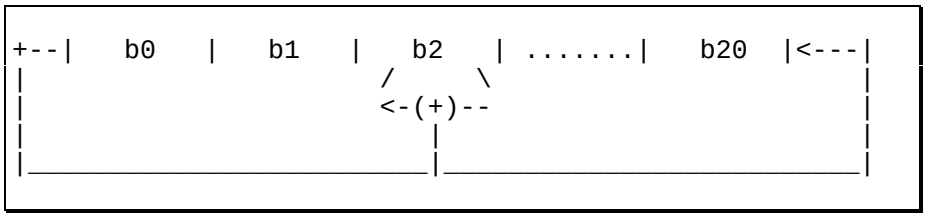


Figure 40. Galois LFSR.

The update algorithm for the Galois LFSR is defined as follows:

```
If b0 = 1 (SV negative):
  then
    flip b2 (XOR with pre-defined mask word)
    rotate left (shift left 1 bit) and set b20 to 1
  else
    rotate left (shift left 1 bit) and set b20 to 0
endif
```

6.4.6.1.1. LFSR Format in SV

The 21-bit LFSR field of the SV is the first 21 bits of the SV as shown in Figure 41.

SV Bit #	63	62	61	60	...	45	44	43
LFSR bit#	0	1	2	3	...	18	19	20

Figure 41. 21-bit LFSR Format

6.4.6.1.2. LFSR Processing

For AAL5 connection types the LFSR is updated (or clocked) once per cell, after the cell is encrypted or decrypted.

For AAL1 and AAL3/4 connections, the LFSR is updated (or clocked) each time the incoming cell sequence number (read from the cell) is less than or equal to the previous cell sequence number (stored in SV memory), on that connection. That is, each time a new cell sequence is detected. For this case the LFSR is updated (or clocked) before the cell is encrypted or decrypted.

The LFSR is initialized to a fixed value (005A5A hex) at the beginning of a connection (i.e., during call setup), and preset to that fixed value when a Secure Key Changeover (SKC) OAM cell is received. (SKC cells are sent to resync the connection.) For AAL5 connections the LFSR is also preset to 0005A5A hex when an end-of-message cell is received.

Note that the preset value 0005A5A is used to encrypt/decrypt the first cell in all cases. On AAL1 and AAL3/4 the “previous” sequence number is initialized to zero (in SV memory) so that the first incoming cell (with seq# = 1) does not update the LFSR.

In all cases, no action need be taken if the receiver's LFSR wraps-around. Because the encryptor's LFSR is always reset before it wraps, the receiver will experience an out-of-sync condition before its LFSR ever wraps around, and, once out-of-sync, LFSR wrap-around is of no significance to the receiver.

6.4.6.2. Initiator/Responder bit

Duplex connections may use the same key for traffic encryption in both directions. If both directions use the same SV then a response that enclosed the original message would have part of the cipher text identical to the original message's cipher text. To prevent such an occurrence, an initiator/responder bit is used in the SV. This forces the responder's key stream to be different so enclosing the original plain text would produce different cipher text. The I/R bit is set to one for cells flowing away from the calling party (initiator) toward the called party (responder), and set to zero in the opposite direction.

The I/R bit field of the SV is bit 42 as shown in Figure 42. Initiator = 1; Responder = 0.

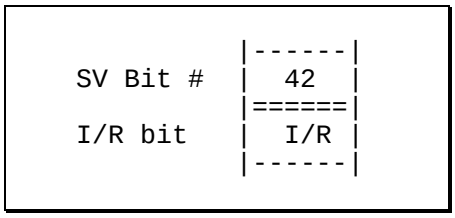


Figure 42. I/R Bit Format.

6.4.6.3. Sequence number

For AAL1 connections the most significant bit of the sequence number (bit 41) is set to zero. The remaining three bits are extracted from the sequence number within the payload of the cell.

For AAL3/4 connections the sequence number is extracted from the four bit sequence number in the payload of the cell. For all other connection types these four bits are set to all zeros.

The 4-bit sequence number field of the SV is bits 41-38 as shown in Figure 43. The MSB is the leftmost bit.

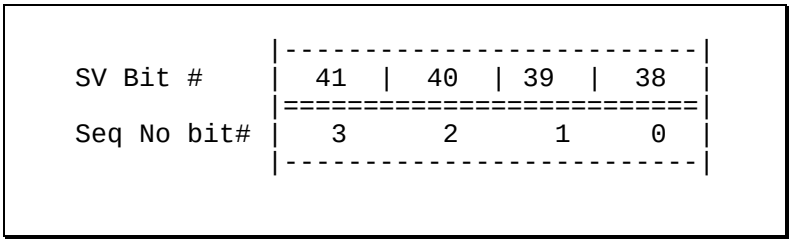


Figure 43. Sequence Number Bit Format.

6.4.6.4. Segment number

The 384-bit ATM cell payload is segmented into six 64-bit segments for encryption or decryption. The LFSR is held constant for the entire cell payload. The segment number is a 3-bit field that defines which 64-bit segment within the payload is being encrypted or decrypted. The segment number for the first 64 bit segment (i.e., octets 1-8 of the payload) is 000, followed by 001, and so on using a binary count through 101. The values 110 and 111 are not used. The format of the segment number in the SV is shown in Figure 44. The MSB is the leftmost bit.

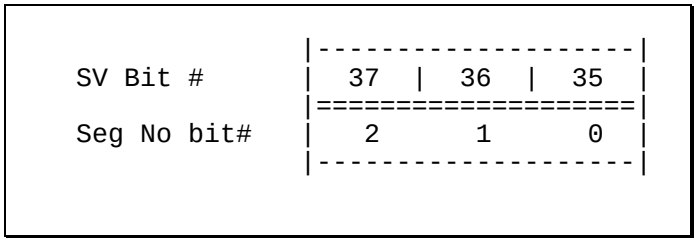


Figure 44. Segment Number Bit Format.

6.4.6.5. Jump Number

The jump number is initialized to all zeros at call setup time and incremented each time a resync SKC OAM cell is sent or, on AAL5 connections, with each end-of-message cell. A 35-bit jump number allows 2E35-1 or 3.44x10E+10 resyncs without repeating.

This ensures the SV is always unique so that repeating patterns in the plain text will not show through in the cipher text.

For example, a full bandwidth 622 Mbps connection (1,416,907 cells/sec), with a cell loss ratio of 10E-4, requires 1417 resyncs per second (10x the expected number of cells lost per second). At this rate the jump number won't repeat for 2.42x10E+07 seconds, about 280 days or 9 months. At this point in time, key stream will start to repeat. To avoid reusing key stream, the session key should be changed prior to the expiration of this period. The implementation should prevent Jump number wrap-around. A Jump number of all ones should be considered an error condition.

The format of the Jump number is shown in Figure 45. The MSB is on the left. The jump number is incremented as a binary count.

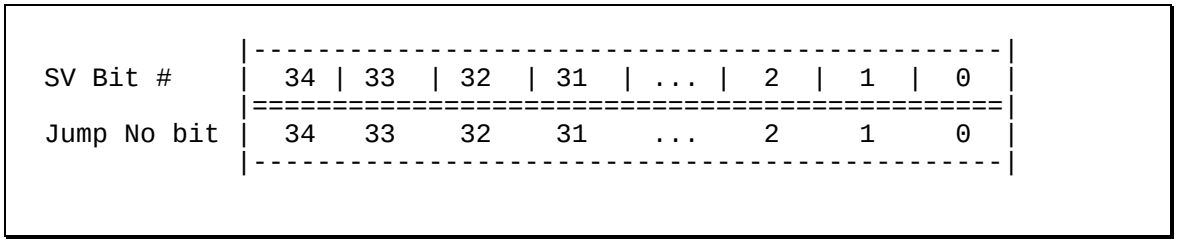
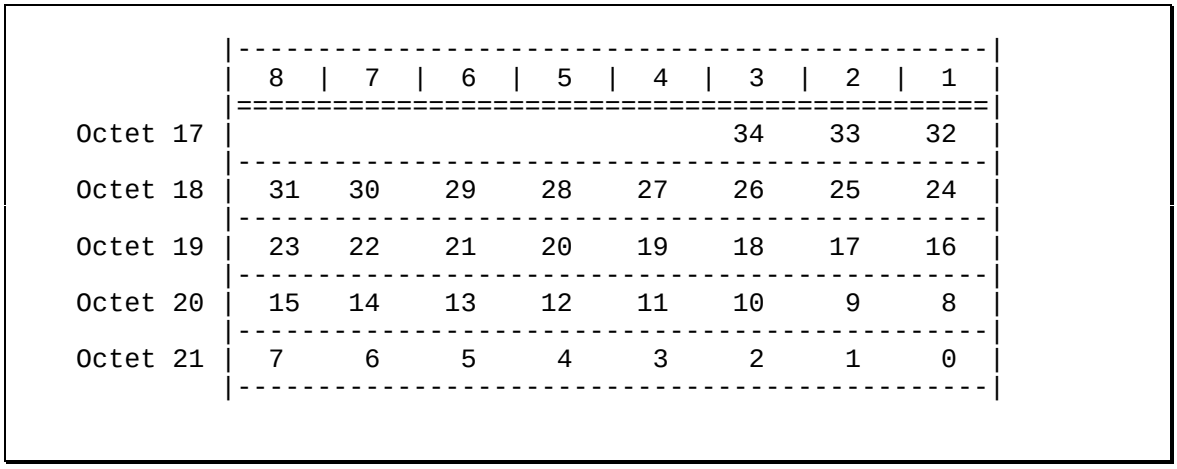


Figure 45. 35-Bit Jump Number Format.

Of the 64-bit SV, the jump number is the only field required to be transmitted to the receiver during a resync or key changeover. The other fields in the SV are preset to their default values.

When an SKC OAM cell is transmitted, the sender places its current Jump number in Octets 17-21 as shown below in Figure 46. The remaining bits (bits 4-8 in Octet 17 and Octets 14-16) are set to zero.

The receiver copies the Jump number from the SKC OAM cell into the SV for that connection only if the new Jump number is greater than the previous Jump number. SKC OAM cells containing Jump numbers less than or equal to the previous Jump number are rejected and this should be treated as an error condition.



	8	7	6	5	4	3	2	1
Octet 17						34	33	32
Octet 18	31	30	29	28	27	26	25	24
Octet 19	23	22	21	20	19	18	17	16
Octet 20	15	14	13	12	11	10	9	8
Octet 21	7	6	5	4	3	2	1	0

Figure 46. Jump Number Position Within SKC OAM Cell.

6.4.7. Using the Counter Mode with Triple DES

When using Triple DES, or multiples of any encryption algorithm, the application of the counter mode must be clarified. It is intended that the counter be applied as follows:

Cipher text = $E_{k1}(D_{k2}(E_{k1}(SV)))$ XOR plain text

Plain text = $E_{k1}(D_{k2}(E_{k1}(SV)))$ XOR cipher text

6.5. Asymmetric Authentication Using ESIGN

6.5.1. Basic Procedure

Private Key:

p; a large prime number,

q; a large prime number ($p > q$).

Public Key:

$n = p^2 q$.

k; an integer ($4 \leq k$),

Signature Generating:

x; chosen at random ($0 \leq x \leq pq-1$),

d; the least integer that is larger or equal to

$|n|/3$ where $|n|$ is a bit length of *n*,

m; a input data to be signed ($m < n/2^{2d}$),

w; the least integer that is larger or equal to $(m\ 2^{2d} - (x^k \bmod n))/(pq)$,

$y = w/(k\ x^{k-1}) \bmod p$,

$S = x + ypq$,

$ESIGN_{(p, q)}(m)$ is defined by a value S.

Verifying:

$ESIGN_{(n, k)}(S)$ is defined as the upper d-bit value of $S^k \bmod n$. When $m' = ESIGN_{(n, k)}(S)$, accept (m', S) as valid if m' contains information which a verifier wants to verify.

Note : In normal case, accept (M, S) if $H' = H''$ where m' contains a hash value H' = Hash(M), and H'' = Hash(M) is newly generated from an original message M by a verifier.

6.5.2. Precomputation Procedure

ESIGN's basic procedure can be accelerated if the Signature Generating process is divided into two parts:

Signature Generating:

a) Precomputation

x; chosen at random ($0 \le x \le pq-1$),

$u = x^k \bmod n$,

$v = 1/(k\ x^{k-1}) \bmod p$.

b) Signing (accelerated)

w; the least integer that is larger or equal to $(m\ 2^{2d} - u)/(pq)$,

$y = wv \bmod p$,

$S = x + ypq$.

Even if one cannot determine an input data m, parameters u and v related to x can be computed separately. Just after selecting m, only the remaining part b) is calculated by using u and v. ESIGN can be accelerated to about ten times faster than the non-precomputation procedure. For example, if one uses a typical 8-bit microprocessor (f=5 MHz), a signature generation delay time can be reduced from about 0.4 s to 0.05 s where its precomputation delay is about 0.4 s.

6.5.3. Example

A small example for ESIGN is presented. Although the size of private keys (p, q) of ESIGN should be more than 170 bits, the example is useful to understand.

Private Key :(Note[1])

p = 1000033,

q = 1000003.

Public Key:

n = 1000069001287003267.

Signature Generating:

x = 878087808780,

k = 1024,

m = 641219,

w = 172197; the least integer that is larger or equal to

(641216 x 1099511627776 - (878087808780^{1024} *mod* 1000069001287003267))/10403

y = 172197/(1024 x 878087808780^{1023}) *mod* 1000033 = 49898, (Note[2])

S = 878087808780 + 49898 x 1000033 x 1000003 = 49900674420748682,

d = 20; the least integer that is larger or equal to $|n|/3$ where $|n|$

is a bit length 60.

Verifying:

Accept (m', S) as valid because the upper 20-bit value of $m' = S^k \bmod n$

= 49900674420748682^{1024} *mod* 1000069001287003267 = 705028336500982591

(= 1001110010001100001110001001011000011011100111000011001111111 in binary

representation) equals 641219 (= 10011100100011000011 in binary representation) = m''

where m'' contains information which a verifier wants to verify.

---

[1] Private key candidates for p, q can be easily checked by division by all prime numbers from 2 to their square roots, respectively. However, since the real candidates are larger than 100 bits, one must check them by using more efficient primality testing methods such as the Rabin method.

[2] The equation of y is modified to:
y = 172197 x 878087808780 / (1024 x 878087808780^{1024}) *mod* 1000033.
After 1024 x 878087808780^{1024} *mod* 1000033 is computed, one can compute its inverse modulo, $(1024 \times 878087808780^{1024})^{-1}$ *mod* 1000033. This is carried out by using the extended Euclidian algorithm.

6.6. Block Cipher/CBC Mode Message Authentication Code

Another method for providing the MAC for AAL SDU data integrity is to use a block cipher in the cipher block chaining (CBC) mode of operation, which is described in detail in ISO/IEC 9797 [17]. Note that when this method is used for data integrity protection, the block cipher is responsible for generating a MAC, **not** for encrypting data.

6.6.1. Padding and Blocking

This method first requires that the AAL SDU data be padded. The SDU may be appended with as few (possibly none) "0" bits as necessary to obtain a data string whose length (in bits) is an integer multiple of *n* (the block size of the algorithm).

The resulting data is divided into *n*-bit blocks (D_1,D_2,...,D_q). This padded data is only used for calculating and verifying the MAC. Consequently, the padding bits (if any) are not stored or transmitted with the data.

6.6.2. The Cryptographic Key

The key should be randomly or pseudo-randomly generated. If the same algorithm is used for encipherment of the message, the key used for the calculation of the MAC should be different from that used for encipherment.

6.6.3. The Initial Stage

The MAC is calculated as illustrated in Figure 47.

The input register is initialized with the first block (D_1).This input data (I_1) is passed through the algorithm (A), which uses a key (K) to produce *n* bits in the output register (O_1).

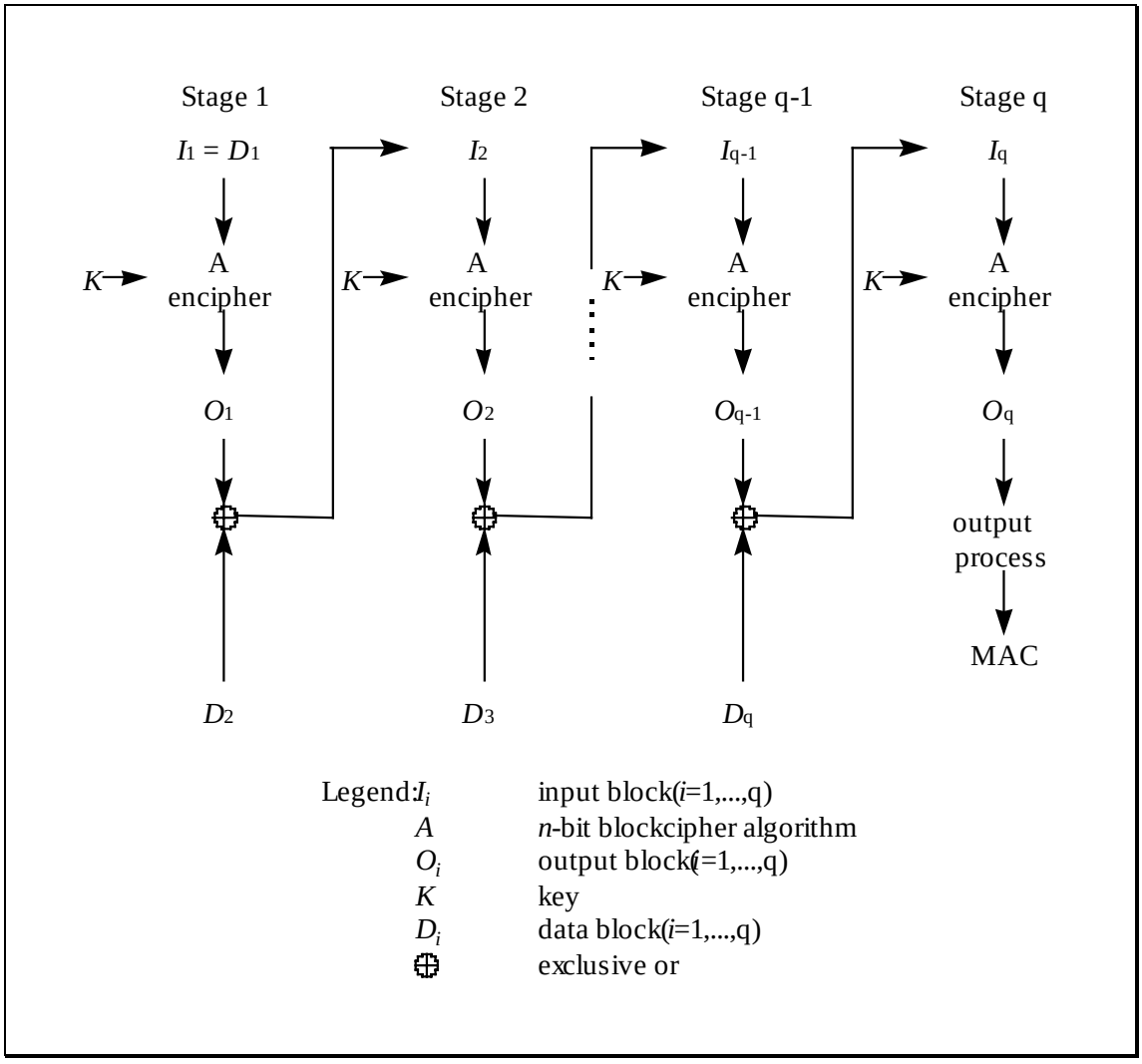


Figure 47. CBC-MAC Calculation.

6.6.4. Subsequent Stages

The next *n* bits of data (D_2) are bitwise exclusive or'ed with the *n* bits of the output register (O_1) and the result is loaded into the input register of the next stage (I_2). The contents of the input register (I_2) is passed through the algorithm (A), which uses the key (K) to produce *n* bits in the output register (O_2).

This operation continues until all blocks have been processed. The result will be the output block (O_q).

6.6.5. Output Process

To increase the strength of the CBC MAC, the output block O_q is subjected to an output process. This process uses an additional key, K1, which is the bitwise complement of the key original key K (as shown in the examples in Sections 6.6.6.1 and 6.6.6.2).

The n-bit block (O_q) is first generated using key (K) in the procedure specified in Sections 6.6.3 and 6.6.4. After O_q is generated, two additional steps shall then be performed (see Figure 48):

a) decipher the output (O_q) using key (K1) to obtain (O'_q):
b) encipher (O'_q) using key (K) to obtain (O''_q).

This completes the output process, and the resulting *n* bits is the MAC.

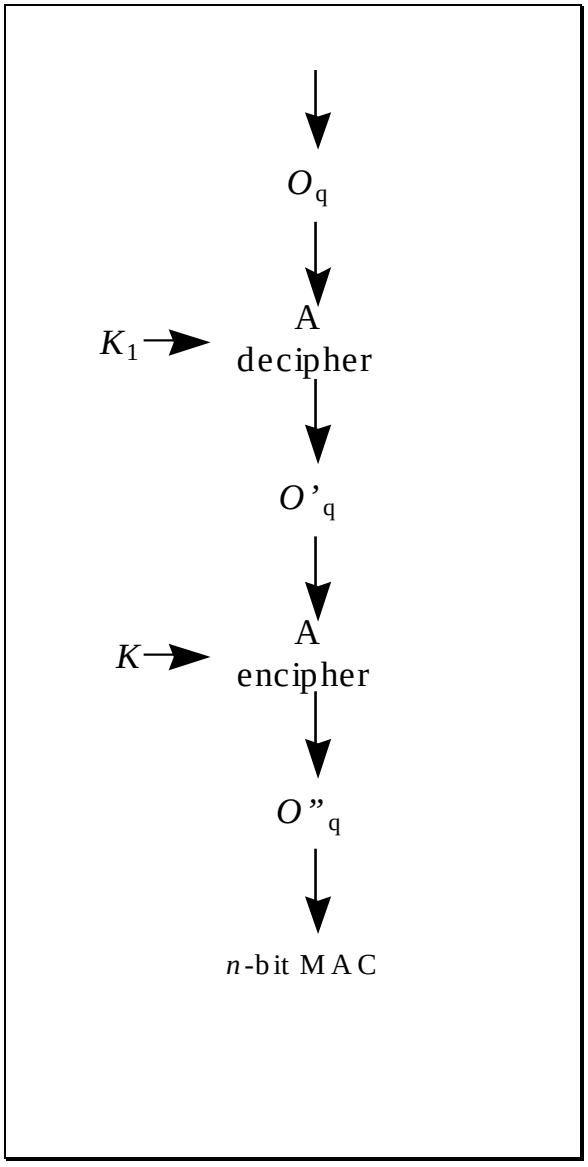


Figure 48. CBC-MAC Output Process.

6.6.6. CBC-MAC Examples

This section presents examples of the generation of a MAC employing the DES and FEAL block encipherment algorithms. The messages are 7-bit ASCII codes (no parity) for "Now_is_the_time_for_all_" and "Now_is_the_time_for_it" where "_" denotes a blank. The first message does not require any padding. The key (K) is 0123456789ABCDEF. The key (K1) for the output process is FEDCBA9876543210.

6.6.6.1. Examples using DES

Example 1: Now_is_the_time_for_all_

```
----------------------------------------
key(K)         : 01 23 45 67 89 AB CD EF
----------------------------------------
D1             : 4E 6F 77 20 69 73 20 74
D2             : 68 65 20 74 69 6D 65 20
D3             : 66 6F 72 20 61 6C 6C 20
----------------------------------------
I1 = D1        : 4E 6F 77 20 69 73 20 74
O1             : 3F A4 0E 8A 98 4D 48 15
I2 = O1 XOR D2: 57 C1 2E FE F1 20 2D 35
O2             : 0B 2E 73 F8 8D C5 85 6A
I3 = O2 XOR D3: 6D 41 01 D8 EC A9 E9 4A
O3             : 70 A3 06 40 CC 76 DD 8B
----------------------------------------
```

Output Process:

```
----------------------------------------
key(K1)        : FE DC BA 98 76 54 32 10
----------------------------------------
O'3            : B4 8D 36 EC 7A D5 69 4F
O"3            : A1 C7 2E 74 EA 3F A9 B6
----------------------------------------
```

The MAC is O"3 = A1 C7 2E 74 EA 3F A9 B6

Example 2: Now_is_the_time_for_it

```
----------------------------------------
key(K)         : 01 23 45 67 89 AB CD EF
----------------------------------------
D1             : 4E 6F 77 20 69 73 20 74
D2             : 68 65 20 74 69 6D 65 20
D3             : 66 6F 72 20 69 74 00 00
----------------------------------------
I1 = D1        : 4E 6F 77 20 69 73 20 74
O1             : 3F A4 0E 8A 98 4D 48 15
I2 = O1 XOR D2: 57 C1 2E FE F1 20 2D 35
O2             : 0B 2E 73 F8 8D C5 85 6A
I3 = O2 XOR D3: 6D 41 01 D8 E4 B1 85 6A
O3             : E4 5B 3A D2 B7 CC 08 56
----------------------------------------
```

Output Process:

```
----------------------------------------
key(K1)       : FE DC BA 98 76 54 32 10
----------------------------------------
O'3           : 32 8A C7 8B A1 CA 0B 3F
O"3           : 2E 2B 14 28 CC 78 25 4F
----------------------------------------
```

The MAC is O"3 = `2E 2B 14 28 CC 78 25 4F`

6.6.6.2. Examples using FEAL-32

Example 1: Now_is_the_time_for_all_

```
----------------------------------------
key(K)        : 01 23 45 67 89 AB CD EF
----------------------------------------
D1            : 4E 6F 77 20 69 73 20 74
D2            : 68 65 20 74 69 6D 65 20
D3            : 66 6F 72 20 61 6C 6C 20
----------------------------------------
I1 = D1       : 4E 6F 77 20 69 73 20 74
O1            : BF 68 7E 85 FC 63 A5 2A
I2 = O1 XOR D2: D7 0D 5E F1 95 0E C0 0A
O2            : 30 95 39 95 1A CF 62 1B
I3 = O2 XOR D3: 56 FA 4B B5 7B A3 0E 3B
O3            : 2A 7A C2 E7 67 0E B9 EC
----------------------------------------
```

Output Process:

```
----------------------------------------
key(K1)       : FE DC BA 98 76 54 32 10
----------------------------------------
O'3           : 5E D4 4C 5A D1 E5 61 EF
O"3           : 88 02 6E 69 54 5B 63 23
----------------------------------------
```

The MAC is O"3 = `88 02 6E 69 54 5B 63 23`

Example 2: Now_is_the_time_for_it

```
----------------------------------------
key(K)        : 01 23 45 67 89 AB CD EF
----------------------------------------
D1            : 4E 6F 77 20 69 73 20 74
D2            : 68 65 20 74 69 6D 65 20
D3            : 66 6F 72 20 69 74 00 00
----------------------------------------
I1 = D1       : 4E 6F 77 20 69 73 20 74
O1            : BF 68 7E 85 FC 63 A5 2A
I2 = O1 XOR D2: D7 0D 5E F1 95 0E C0 0A
O2            : 30 95 39 95 1A CF 62 1B
```

```
I3 = O2 XOR D3: 56 FA 4B B5 73 BB 62 1B
O3            : C9 78 9A 54 72 24 D8 E2
----------------------------------------
```

Output Process:

```
----------------------------------------
key(K1)       : FE DC BA 98 76 54 32 10
----------------------------------------
O'3           : AA 10 A1 5B 3D 95 12 F7
O"3           : B6 72 A6 02 1C EA D2 03
----------------------------------------
```

The MAC is O"3 = `B6 72 A6 02 1C EA D2 03`

6.7. DES-40

DES40 is a variant of DES. It uses an effective 40-bit key instead of the 56-bit key.

The key is presented as a 64-bit value with bits numbered k1 through k64 (where k1 is the most significant bit, and k64 is the least significant bit). k8, k16, k24, k32, k40, k48, k56, and k64 are ignored by DES and can be used to parity check the key.

The following bits shall be set to 0:

k1-4, k17-k20, k33-k36, k49-k52.

These are the top 4 bits of the 1st, 3rd, 5th, and 7th bytes of the key.

6.8. Algorithm-Specific Use of Security Message Exchange Protocols

The security message exchange protocols described in Section 5.1 are not dependent on the use of any specific cryptographic algorithm. In the following sections, the use of these protocols with Diffie-Hellman for key exchange, with RSA for digital signature and key exchange, and with DSA for digital signature are specified.

6.8.1. RSA - Digital Signature and Key Exchange

RSA is a public-key algorithm that works for both encryption and digital signatures. The security of RSA depends on the difficulty of factoring large numbers. The public and private keys of RSA are generated as follows: Two random positive large prime integers, p and q are selected. Their product $n = pq$ is computed. n is called the public modulus. A random positive integer e less than n and relatively prime to product $(p-1)(q-1)$ is selected. This means that e and $(p-1)(q-1)$ have no common factor except 1. Then, a positive integer d such that $de-1$ is divisible by $(p-1)(q-1)$ is computed. In other words, $d = e^{-1} \bmod((p-1)(q-1))$. The values e and d are called the public and private exponent, respectively. The public key is the pair (e, n) and the private key is the pair (d, n). Once the public and private keys are computed, the prime factors, p and q are no longer needed and shall be either kept secret or destroyed.

An important property of RSA is that the steps of encryption and decryption can be reversed, i.e., $data = RSA_{(e,n)}(RSA_{(d,n)}(data)) = RSA_{(d,n)}(RSA_{(e,n)}(data))$. This property allows it to work for both encryption as well as for digital signatures.

In the following discussion, it is assumed that each authentication entity (*A* and *B*) possesses a public/private RSA key pair. Each authentication entity must either know or obtain its partner's public key. The three-way security message exchange protocol provides an option for the authentication entities to exchange certificates (which contain public key information). When supported, the authentication entities may use the certificate exchange option of the three-way security message exchange protocol to obtain each other's public key.

RSA can be used in both the two- and three-way security message exchange protocols for both digital signature as well as key exchange.

In the following discussions, upper-case italic symbols (e.g., *PS*) denote octet strings and lower-case italic symbols (e.g., *n*) denote integers.

6.8.1.1. RSA Encryption/Decryption Processes

The following sections describe the encryption/decryption process that shall be used in the two- and three-way security message exchange protocol when RSA is used for digital signature and/or key exchange. The encryption/decryption process described here is based on the methods of PKCS#1 [[24].

6.8.1.1.1. RSA Encryption Process

Let the length of the modulus *n* in octets be *K*. The length *K* of the modulus shall be at least 12 octets to accommodate the block formats defined here. The message (*M*) to be encrypted shall have a length which is an integer multiple of 8 bits. To encrypt *M*, first divide it into data blocks of length *K*-11 octets, which is positive since the length *K* is at least 12 octets.

$$M = M_1, M_2, \ldots, M_m \qquad \text{where } m \geq 1$$

The last data block (i.e., M_m) may be of length less than *K*-11 octets. Encrypt each data block M_i using the four-step process described below to obtain EM_i. Then, concatenate the individually encrypted data blocks to obtain the encrypted message *EM*.

$$EM = EM_1, EM_2, \ldots, EM_m \qquad \text{where } m \geq 1$$

Let *D* represent any data block (M_i) of the message *M*. The encryption process of *D* consists of four steps which are described below.

1. **Encryption-block formatting:** Format the data block *D* to an encryption block (*EB*) octet string as follows.

   EB = 00, BT, PS, 00, D

   A 00 octet, a block type (*BT*) octet, a padding string (*PS*) octets, and a 00 octet are prepended to the data block *D* to form the encryption block (*EB*). The leading 00 octet ensures that the encryption block *EB* value will be less than the modulus *n* when it's converted to an integer. The block type *BT* octet shall be set to 01 for private-key operation and shall be set to 02 for public-key operation. The padding string *PS* shall consist of $K - 3 - \|D\|$ octets (where $\|D\|$ is the length of *D* in octets). The *PS* octet string shall be pseudorandomly generated and must have no octet with 00 value. The randomness of *PS* strengthens the security of RSA encryption. So, the total length of the encryption block (*EB*) is equal to *K* octets.

2. **Octet-string to integer conversion:** Convert the encryption block *EB* to an integer x. Let $EB_1, EB_2, \ldots, EB_K$ be the octets of *EB* from first to last. Then the integer x shall satisfy

$$x = \sum_{i=1}^{K} 2^{8(K-i)} EB_i \qquad \text{where } EB_i \in [0..255]$$

With this representation, the first octet EB_1 is the most significant in the integer x and the last octet EB_K is the least significant.

3. **RSA computation:** Compute the RSA classical computation of the integer x as follows. Raise x to the power c modulo n to obtain y, the integer encrypted data block.

$$y = x^c \bmod n \qquad 0 \le y < n$$

For a private-key operation, the integer c is an entity's private exponent d; for a public-key operation, the integer c is an entity's public exponent e.

4. **Integer to octet-string conversion:** Convert the integer encrypted data y to an octet string *ED* of length K octets. The encrypted data *ED* shall satisfy

$$y = \sum_{i=1}^{K} 2^{8(K-i)} ED_i \qquad \text{where } ED_i \in [0..255]$$

where $ED_1, ED_2, \ldots, ED_K$ are the octets of *ED* from first to last. The first octet ED_1 is the most significant in the integer and the last octet ED_K is the least significant.

6.8.1.1.2. RSA Decryption Process

Let the length of the modulus *n* in octets be *K*. To decrypt the encrypted message *EM*, first divide it into *m* data blocks of length *K* octets. (Note that the length of *EM* is an integer multiple of *K* octets.)

$$EM = EM_1, EM_2, \ldots, EM_m \qquad \text{where } m \ge 1$$

Decrypt each data block EM_i using the four-step process described below to obtain M_i. Then, concatenate the individually decrypted data blocks to obtain the message *M*.

$$M = M_1, M_2, \ldots, M_m \qquad \text{where } m \ge 1$$

Let *ED* represent any encrypted data block (EM_i) of the message *EM*. The decryption process of *ED* consists of four steps that are described below.

1. **Octet-string to integer conversion:** Convert the encrypted data *ED* of length *K* octets to an integer y. The octet string *ED* shall satisfy

$$y = \sum_{i=1}^{K} 2^{8(K-i)} ED_i \qquad \text{where } ED_i \in [0..255]$$

where $ED_1, ED_2, \ldots, ED_K$ are the octets of *ED* from first to last. The first octet ED_1 is the most significant in the integer and the last octet ED_K is the least significant.

2. **RSA computation:** Compute the RSA classical computation of the integer y as follows. Raise y to the power c modulo n to obtain x, the integer decrypted data block.

$$x = y^c \bmod n \qquad 0 \le x < n$$

For a private-key operation, the integer c is an entity's private exponent d ; for a public-key operation, the integer c is an entity's public exponent e .

3. **Integer to octet-string conversion:** Convert the integer x to encryption block *EB* of length *K* octets. Let $EB_1, EB_2, \ldots, EB_K$ be the octets of *EB* from first to last. Then the integer x shall satisfy

$$x = \sum_{i=1}^{K} 2^{8(K-i)} EB_i \qquad \text{where } EB_i \in [0..255]$$

With this representation, the first octet EB_1 is the most significant in the integer x and the last octet EB_K is the least significant.

4. **Encryption-block parsing:** Parse the encryption block (*EB*) octet string into a block type *BT*, a padding string *PS*, and the data block *D* as follows.

EB = 00, BT, PS, 00, D

The padding scheme allows the encryption block *EB* to be parsed unambiguously without knowing the length of the data block *D*. This is because the padding string *PS* contains no octets with value 00 and the padding string is separated from the data block *D* by an octet with value 00.

6.8.1.2. Digital Signature

The following sections describe the digital signature generation and validation with RSA using the methods of PKCS#1 [[24].

Signature Generation:

For digital signature computation, the message (*M*) to be signed is first reduced using a hash algorithm (e.g., SHA-1) and then the hash value is encrypted with the RSA private key of the signer (i.e., the authentication entity that's generating the signature) using the RSA encryption process described above.

Refer to Sections 6.2.5.5.1 and 6.2.5.5.2 for specifications on constructing the digital signature buffer (or message) *M*.

The procedure to compute the digital signature (*S*) over the message *M* is as follows.

1. Run a hash algorithm (e.g., SHA-1) over *M* to produce a hash octet string *H*.

$$H = Hash(M)$$

2. Encrypt *H* with the RSA private key of the signer (i.e., the authentication entity that's generating the signature) using the RSA encryption process described above to obtain a digital signature octet string *S*.

$$S = RSA_{(d,n)}(H)$$

The signature octet string *S* is placed in the "Digital Signature Value" field in the appropriate digital signature primitive of the Security Services Information Element (SSIE).

Signature Verification:

For digital signature verification, the signature (*S*) octet string is first decrypted with the signer's RSA public key (using the RSA decryption process described above) and then compared with the hash of the message (*M*).

The procedure to verify the digital signature (*S*) over the message *M* is as follows.

1. Decrypt the digital signature *S* with the RSA public key of the signer (i.e., the authentication entity that generated the signature) using the RSA decryption process described above to obtain the octet string H'.
$$H' = RSA_{(e,n)}(S)$$

2. Run the hash algorithm used to generate the signature over *M* to produce a hash octet string H''.
$$H'' = Hash(M)$$

3. Compare the octet strings H' and H''. If they are equal, the signature verification process shall be successful, and it shall fail otherwise.

6.8.1.3. Key Exchange

Key exchange with RSA is achieved by encrypting the key (*C*) to be exchanged with the RSA public key of the recipient (i.e., the entity that should receive the key). The "Confidential Data" part (i.e., octet x.4 and higher) of the "Confidential Section" of the SSIE (see Section 6.2.4) contains the key (or keys) to be exchanged.

Sender's procedure:

Let *C* represent the octet string of the "Confidential Data" in the SSIE. The sender first encrypts *C* with the RSA public key of the receiver using the RSA encryption process described above to obtain an octet string *EC*.

$$EC = RSA_{(e,n)}(C)$$

Then, the sender replaces the entire octet string of the "Confidential Data" in the SSIE with *EC* and adjusts the length field for the Confidential Section if the length of EC is different from the length of *C*.

Receiver's procedure:

The receiver decrypts *EC* with its RSA private key using the RSA decryption process described above to obtain the octet string *C*.

$$C = RSA_{(d,n)}(EC)$$

6.8.2. DSA - Digital Signature

DSA is a public-key algorithm that can only be used for digital signatures. The security of DSA depends on the difficulty of computing discrete logarithms modulo a prime (i.e., in a prime finite field). A detailed description of the DSA algorithm can be found in [9].

DSA uses a number of public parameters in its operation. These parameters are *p* (a prime modulus), *q* (a prime divisor of *p*-1), and *g* (a generator). These parameters, along with the public key *y*, are contained in the DSA certificate for the user that generates the DSA digital signature (the private key *x* is kept secret by the user). In the following discussion of ATM authentication using DSA, it is assumed that each authentication entity (*A* and *B*) possesses a public/private DSA key pair, and that any entity which wishes to verify the signature has access to *p*, *q*, *g*, and *y*.

In order to generate a digital signature, the Digital Signature Standard [9] specifies the use of the Secure Hash Algorithm, which is defined in [8]. SHA-1 is used in DSS to generate a "fingerprint" of the message, which is then encrypted using DSA to bind the fingerprint to the signing entity. In order to generate the

SHA-1 “fingerprint,” the message must be padded such that its length is an integer multiple of 512 bits. This procedure, which is defined in [8], is summarized below:

SHA-1 padding procedure

Pad the message to produce a new message M, the length of which is a multiple of 512 bits. The padding is performed as follows:

1. Store the current length (in bits) of the message for later use.
2. Append a “1” bit to the original message data, followed by enough “0” bits to make the length of the new message 64 bits short of a 512 bit boundary.
3. Store the binary encoding of the message length (in bits) in the final 64 bits of the new message M. At this point, the length of M is a multiple of 512 bits.

In order to perform ATM authentication using SHA-1 and DSA, octet strings (e.g., padded message, DSA parameters, etc.) must be converted to integers. This conversion is defined in [9], and is summarized below:

Octet-String to Integer Conversion

An integer x is computed from a block of K octets $B_1, B_2, ..., B_K$ as follows:

$$x = \sum_{i=1}^{K} 2^{8(K-i)} B_i \qquad \text{where } B_i \in [0..255]$$

where B_1 (the leftmost octet) is the most significant octet.

In order to transport DSA parameters in ATM information elements, integers (e.g., public parameters, signatures, etc.) must be converted to octet strings. This conversion is defined in [9] and is summarized below:

Integer to Octet-String Conversion

Given an integer x, determine a block of K octets $B_1, B_2, ..., B_K$ such that the following relation applies:

$$x = \sum_{i=1}^{K} 2^{8(K-i)} B_i \qquad \text{where } B_i \in [0..255]$$

where B_1 (the leftmost octet) is the most significant octet.

6.8.2.1. DSA Digital Signature Generation

The following steps are required to produce a DSA digital signature for use in the Security Message Exchange Protocols:

1. Construct the digital signature buffer according to the procedures defined in Sections 6.2.5.5.1and 6.2.5.5.2.
2. Pad the digital signature buffer to produce a message *M* using the padding procedure defined in [8] (and summarized earlier).

3. Calculate the SHA-1 hash *H=SHA(M)* as defined in [8] and convert the resulting octet string to a 160-bit (20 octet) integer as specified in [9] (and summarized earlier).

4. Calculate *r* and *s* as defined in [9].

5. Convert the 160-bit (20-octet) quantities *r* and *s* to octet strings using the procedure defined in [9] (and summarized earlier).

6. Insert the octet strings *r* and *s* into the "Digital Signature Value" in the appropriate digital signature primitive of the Security Services Information Element (SSIE).

6.8.2.2. DSA Digital Signature Verification

The following steps are required to validate a DSA digital signature when used in the Security Message Exchange Protocols:

1. Construct the digital signature buffer according to the procedures defined in Sections 6.2.5.5.1 and 6.2.5.5.2.

2. Pad the message to produce a new message *M'* using the padding procedure defined in [8] (and summarized earlier).

3. Calculate the SHA-1 hash *H'=SHA(M')* as defined in [8] and convert the resulting octet string to a 160-bit (20 octet) integer as specified in [9] (and summarized earlier).

4. Extract the received values of *r* and *s* from the "Digital Signature Value" field of the appropriate digital signature primitive of the SSIE. Convert the *r* and *s* octet strings to 160 bit (20 octet) integers *r'* and *s'* using the procedure defined in [9] (and summarized earlier).

5. Calculate *v* as defined in [9].

6. If *v=r'*, then the signature verification process shall be successful. Otherwise, it shall fail.

6.8.3. ESIGN - Digital Signature

ESIGN is a public-key algorithm that can only be used for digital signatures. The security of ESIGN depends on the difficulty of factoring large numbers. A detailed description of the ESIGN algorithm can be found in Section 6.5.

The public and private keys of ESIGN are generated as follows: Two random positive large prime integers, *p* and *q* ($p > q$) are selected. Their product $n = p^2 q$ is computed. *n* is called the public modulus. The private key for signature process is the pair (p, q) ($p > q$) and the public key for verification process is the pair (n, k) where k is an integer ($4 \le k$). A positive integer *d* is computed which is the least integer that is larger or equal to $|n|/3$ where $|n|$ is a bit length of *n*.

ESIGN Signature Process $SD = ESIGN_{(p, q)}(SB)$

1. **Octet-string to integer conversion:** Convert the signature block *SB* to an integer *m*. Let an integer *d* be $8K$ and $SB_1, SB_2, ..., SB_K$ be the octets of *SB* from first to last. Then, an integer *m* to be signed shall satisfy

$$m = \sum_{i=1}^{K} 2^{8(K-i)} SB_i \qquad \text{where} \qquad SB_i \in [0,\ldots,255]$$

With this representation, the first octet SB_1 has the most significance in the integer m and the last octet SB_K has the least significance.

2. **ESIGN computation:** Compute the ESIGN computation of the integer m to obtain S as follows: After a random positive integer x less than pq - 1 is selected,

$$w = \frac{\square m2^{2d} - \left(x^k \bmod n\right)}{pq},$$

$$y = \frac{w}{kx^{k-1}} \bmod p \text{, and}$$

$$S = x + ypq .$$

Clear definitions are described in Section 6.5.

3. **Integer to octet-string conversion:** Convert the integer signed data S to an octet string SD of length $3K$ octets. The signed data SD shall satisfy

$$S = \sum_{i=1}^{3K} 2^{8(3K-i)} SD_i \quad \text{where} \quad SD_i \llcorner [0,\ldots,255]$$

where SD_1, SD_2, ..., SD_{3K} are the octets of SD from first to last. The first octet SD_1 has the most significance in the integer and the last octet SD_{3K} has the least significance.

ESIGN Verification Process $SB = ESIGN_{(n,k)}(SD)$

1. **Octet-string to integer conversion:** Convert the signed data SD of length $3K$ octets to an integer S. The octet string SD shall satisfy

$$S = \sum_{i=1}^{3K} 2^{8(3K-i)} SD_i \quad \text{where} \quad SD_i \llcorner [0,\ldots,255]$$

where SD_1, SD_2, ..., SD_{3K} are the octets of SD from first to last. The first octet SD_1 has the most significance in the integer and the last octet SD_{3K} has the least significance.

2. **ESIGN computation:** Compute the ESIGN computation of the integer S to obtain m as follows;

$$m = \frac{S^k \bmod n}{2^{2d}} \square .$$

Clear definitions are described in Section 6.5.

3. **Integer to octet-string conversion:** Convert the integer m to signature block SB of length K octets. Let SB_1, SB_2, ..., SB_K be the octets of SB from first to last. Then the integer m shall satisfy

$$m = \sum_{i=1}^{K} 2^{8(K-i)} SB_i \quad \text{where} \quad SB_i \llcorner [0,\ldots,255]$$

With this representation, the first octet SB_1 has the most significance in the integer m and the last octet SB_K has the least significance.

6.8.3.1. ESIGN Signature Generation:

For digital signature computation, the message (*M*) to be signed is first reduced using a hash algorithm (e.g., SHA-1) and then the hash value is signed with the ESIGN private key of the signer (i.e., the authentication entity that's generating the signature) using the ESIGN signature process in [29] [34].

Refer to Sections 6.2.5.5.1 and 6.2.5.5.2 for specifications on constructing the digital signature buffer (or message) *M*.

The procedure to compute the digital signature (*SD*) over the message *M* is as follows:

1. Run a hash algorithm (e.g., SHA-1) over *M* to produce a hash octet string *H*.

   $$H = Hash(M)$$

2. Format the hash data *H* to a signature block (*SB*) octet string as follows.

   $$SB = 00, PS, FF, H$$

   A *00* octet, a padding string (*PS*) octets, and a *FF* octet are prepended to the hash value *H* to form the signature block (*SB*). The leading *00* octet ensures that the signature block *SB* value will be less than 2^d when it's converted to an integer. The padding string *PS* shall consist of *K*-2-||*H*|| octets (where ||*H*|| is the length of *H* in octet). The *PS* octet string shall have no octet with *FF* value. The total length of the signature block (*SB*) is equal to *K* octets.

3. Sign to *SB* with the ESIGN private key of the signer (i.e., the authentication entity that's generating the signature) using the ESIGN signature process described above to obtain a digital signature octet string *SD*. The total length of the signed data (*SD*) is equal to *3K* octets.

   $$SD = ESIGN_{(p, q)}(SB)$$

The signature octet string *SD* is placed in the "Digital Signature Value" field in the appropriate digital signature primitive of the Security Services Information Element (SSIE).

6.8.3.2. ESIGN Signature Verification:

For digital signature verification, the signature (*SD*) octet string is verified with the signer's ESIGN public key and the hash of the message (*M*). The procedure to verify the digital signature (*SD*) over the message *M* is as follows:

1. Verify the digital signature *SD* with the ESIGN public key of signer (i.e., the authentication entity that generated the signature) using the ESIGN verification process described above to obtain the octet string *SB'*.

   $$SB' = ESIGN_{(n, k)}(SD).$$

2. Parse the signature block (*SB'*) octet string into a padding string *PS*, and the data block *H'* as follows.

   $$SB' = 00, PS, FF, H'$$

   The padding scheme allows the signature block *SB'* to be parsed unambiguously without knowing the length of the hash value *H'*. This is because the padding string *PS* contains no octets with value *FF* and the padding string is separated from the data block *H'* by an octet with value *FF*. Then, the *H'* is picked out to verify.

3. Run the hash algorithm used to generate the signature over *M* to produce a hash octet string *H"*.

$H'' = Hash(M)$

Compare the octet string H' and H''. If H' is equal to H'', the signature verification process shall be successful, and it shall fail otherwise.

6.8.4. Diffie-Hellman Key Exchange

Diffie-Hellman is a public-key algorithm that allows two parties (say *A* and *B*) to come up with a secret shared key that is known only to them. The two parties first agree on a pair of public parameters: p, a positive large prime integer and g, a positive integer. These two parameters are not secret. p is called the public modulus and g is called the public base. The base must satisfy $0 < g < p$. The party *A* generates a private random integer a such that $0 < a < p-1$ and computes $x = g^a \bmod p$, where $0 < x < p$. The values a and x are called the private and public key of *A*, respectively. Similarly, the party *B* generates a private random integer b such that $0 < b < p-1$ and computes $y = g^b \bmod p$, where $0 < y < p$. The values b and y are called the private and public key of *B*, respectively. Then, the two parties exchange their public keys, i.e., *A* sends x to *B* and *B* sends y to *A*. The public keys are not secret. Then, each party independently computes the Diffie-Hellman integer secret shared key z. That is *A* computes $z = y^a \bmod p$ and *B* computes $z = x^b \bmod p$, where $0 < z < p$.

Since Diffie-Hellman is a key exchange algorithm and does not protect against man-in-the-middle attack, it shall be used in combination with a digital signature algorithm (such DSA, RSA, etc.) to protect the exchange of public keys when used over an insecure channel. Additionally, the use of the Diffie-Hellman algorithm itself is limited to the three-way security message exchange protocol. The reason for this limitations is that the two-way security message exchange protocol cannot support Diffie-Hellman because encrypting $ConfPar_a$ in flow 1 would require the knowledge of the secret shared key, which is not established yet.

The procedure for adding leaves to a pt-mpt call, when the Diffie-Hellman key exchange algorithm is used, is defined in Section 5.2.3.

6.8.4.1. Diffie-Hellman With Three-Way Security Message Exchange Protocol

The initiator first selects p (the public modulus) and g (the public base) and computes its public key x. The initiator converts the integers p, g, and x to octet strings *P*, *G*, and *X*, respectively.

Integer to octet-string or octet-string to integer conversion: Let r be an integer and *R* be its corresponding octet string of length K octets. The octet string *R* shall satisfy

$$r = \sum_{i=1}^{K} 2^{8(K-i)} R_i \qquad \text{where } R_i \in [0..255]$$

where $R_1, R_2, \ldots, R_K$ are the octets of *R* from first to last. The first octet R_1 is the most significant in the integer and the last octet R_K is the least significant.

The initiator sends *P*, *G*, and *X* in $SecNeg_a$ of FLOW1-3WE to the responder. Upon receiving *P*, *G*, and *X*, the responder computes its public key y and the secret shared key z. The responder converts its public key y to octet string *Y* and sends it to the initiator in $SecNeg_b$ of FLOW2-3WE. Upon receiving *Y*, the initiator computes the secret shared key z. Both the initiator and responder convert the secret shared key z to octet string *Z*.

The low order 256 bits of Z shared by the initiator and responder is the Initial Key Exchange Key $\overline{Z}$ used to encrypt $ConfPar_a$ and $ConfPar_b$.

Let *C* represent the "Confidential Data" part (i.e., octet x.4 and higher) of the "Confidential Section" of the SSIE (see Section 6.2.4). To encrypt *C* with the secret shared key $\overline{Z}$, a *Zconcat* is first built from $\overline{Z}$ as follows: $\|Z\| - \|C\|$, where *Zconcat* is the concatenation of $|Z_1|$, $|Z_2|$, etc. such that its total length is greater than or equal to the length of *C*. Each $|Z_i|$ (where $|Z_1|$) is computed as follows:

$$Z_i = SHA(\overline{Z}, KeyFill, A, B, i, R_A, \overline{Z}, ShaFill)$$

First, the shared key $\overline{Z}$ is appended *KeyFill* to the next 512-bit boundary, using the pad with length technique defined for SHA-1 (Secure Hash Algorithm) in [8]. Then, the filled shared key is concatenated with (immediately followed by) the source's distinguished name *A*, concatenated with (immediately followed by) the destination's distinguished name *B*, concatenated with (immediately followed by) a one-octet (8-bit) counter i, concatenated with (immediately followed by) a random number R_A generated by the source, and concatenated with (immediately followed by) the shared key $\overline{Z}$ again. A trailing pad (i.e., *ShaFill*) with length to the next 512-bit boundary for the entire message is added by SHA-1 itself. Then, the 20-byte SHA-1 message digest is calculated (as defined in [8]) and the result is $|Z_i|$.

Then, *Zmask* is formed by removing the rightmost $\|C\| - \|\|$ octets of *Zconcat* (where $\|Zconcat\|$ is the length of *Zconcat* in octets and $\|C\|$ is the length of *C* in octets), so that *Zmask* has the same length as *C*. The encrypted *C* is obtained by *xor*ing *Zmask* with *C*. That is, $EC = Zmask \oplus C$. Then, the sender replaces the entire octet string of the "Confidential Data" in the SSIE with *EC*. To recover *C*, the encrypted *C* (i.e., *EC*) is *xor*ed with *Zmask*. That is, $C = Zmask \oplus EC$.

6.8.5. Session Key Exchange using MD5

The hash function to be used in SKE for a connection may be negotiated at the time the connection is established. Here the MD5 method of RFC-1321 [12] is defined as one option for use with SKE. The shared master key, $K_{m,AB}$, is not constrained by this scheme to any particular size.

The "*Mask*" for xoring with the session key is obtained using MD5 as follows:

$$HalfMask1 = MD5(K_{m,AB}, KeyFill, A, KN_A, c1, K_{m,AB}, MD5Fill)$$

$HalfMask2 = MD5(K_{m,AB}, KeyFill, A, KN_A, c2, K_{m,AB}, MD5Fill)$ if session key > 128 bits

$HalfMask2 = 0$ if session key < 128 bits

$Mask = (HalfMask2, HalfMask1)$

Where the constants c1 = 0000 0000 hex, and c2 = 5555 AAAA hex for the initiator, and where the constants c1 = 00FF FF00 hex, and c2 = 6666 CCCC hex for the responder

Note: for simplex connections (e.g., pt-mpt) the constants associated with the initiator are used.

The first half mask is computed as follows: The *KeyFill* is appended to the shared master key to the next 512-bit boundary, using the pad with length technique defined for MD5 in RFC-1321 [12]. Then, the filled master key is concatenated with (immediately followed by) *A*, concatenated with (immediately followed by) KN_A, concatenated with (immediately followed by) the constant *c*1, and concatenated with (immediately followed by) the original variable length master key again. A trailing pad (i.e., *MD5Fill*) with length to the

next 512-bit boundary for the entire message is added by MD5 itself. Then, the 16-byte MD5 message digest is calculated (as defined in RFC-1321 [12]), and the result is the *HalfMask*1.

The length of the session key being encrypted determines the computation of the second half mask. If it is greater than 128 bits, the second half mask is computed in a similar fashion to the first half mask, except that the constant c2 is used in place of c1. If the session key is 128 bits or less, the second half mask is simply set to zero.

The *Mask* is then obtained by concatenating the two *HalfMask*s. The *Mask* length is 32 bytes, which is equal to the length of the "encrypted session key" field of the SKE OAM cell. To obtain the encrypted session key, the session key is first prepended with zeros (with *ZeroPad*) so that its total length is 256 bits (i.e., 32 bytes). If the session key length is already 256 bits, no padding is required. Then, the padded session key is xored with the *Mask* to obtain the 32-byte encrypted session key.

$$Enc_{K_{m,AB}}(K_{s,AB}) = (ZeroPad, K_{s,AB}) \quad Mask$$

To recover the session key, $K_{s,AB}$, from the encrypted session key, $Enc_{K_{m,AB}}(K_{s,AB})$, the encrypted session key is first xored with the *Mask* and then the prepended *ZeroPad* is removed.

6.8.6. Session Key Exchange using SHA-1 and RIPEMD-160

The hash function, H to be used in SKE for a connection may be negotiated at the time the connection is established. Here the SHA-1 [8] and the RIPEMD-160 [18] methods are defined as two options for use with SKE. The shared master key, $K_{m,AB}$, is not constrained by this scheme to any particular size.

The "*Mask*" for xoring with the session key is obtained using H as follows:

$$HalfMask1 = trunc_{128}(H(K_{m,AB}, KeyFill, A, KN_A, c1, K_{m,AB}, H\_Fill))$$

$$HalfMask2 = trunc_{128}(H(K_{m,AB}, KeyFill, A, KN_A, c2, K_{m,AB}, H\_Fill)) \text{ if session key > 128 bits}$$

$$HalfMask2 = 0 \text{ if session key < 128 bits}$$

$$Mask = (HalfMask2, HalfMask1)$$

Where the constants c1 = 0000 0000 hex, and c2 = 5555 AAAA hex for the initiator, and where the constants c1 = 00FF FF00 hex, and c2 = 6666 CCCC hex for the responder.

The first half mask is computed as follows: The *KeyFill* is appended to the shared master key to the next 512-bit boundary, using the pad with length technique defined for SHA-1 in [8] or RIPEMD-160 in [18]. Then, the filled master key is concatenated with (immediately followed by) *A*, concatenated with (immediately followed by) KN_A, concatenated with (immediately followed by) the constant c1, and concatenated with (immediately followed by) the original variable length master key again. H itself adds a trailing pad (i.e., *H_Fill*) to the next 512-bit boundary for the entire message. Then, the 20-byte H message digest is calculated (as defined in [8] or [18]). The H digest is then truncated to 128 bits by throwing away the high order 32 bits, and the result is the *HalfMask*1.

The length of the session key being encrypted determines the computation of the second half mask. If it is greater than 128 bits, the second half mask is computed in a similar fashion to the first half mask, except that the constant c2 is used in place of c1. If the session key is 128 bits or less, the second half mask is simply set to zero.

The *Mask* is then obtained by concatenating the two *HalfMask*s. The *Mask* length is 32 bytes, which is equal to the length of the "encrypted session key" field of the SKE OAM cell. To obtain the encrypted session key, the session key is first prepended with zeros (with *ZeroPad*) so that its total length is 256 bits (i.e., 32 bytes). If the session key length is already 256 bits, no padding is required. Then, the padded session key is xored with the *Mask* to obtain the 32-byte encrypted session key.

$$Enc_{K_{m,AB}}(K_{s,AB}) = (ZeroPad, K_{s,AB}) \quad Mask$$

To recover the session key, $K_{s,AB}$, from the encrypted session key, $Enc_{K_{m,AB}}(K_{s,AB})$, the encrypted session key is first xored with the *Mask* and then the prepended *ZeroPad* is removed.

6.9. Asymmetric Authentication and Key Exchange Using Elliptic Curve Cryptosystems

This section describes the method of performing authentication and key exchange using the elliptic curve cryptosystem. This section is self-contained, that is, all references indicated in this section (Section 6.9) can be found in Section 6.9.6.

Many public-key cryptographic systems are based on exponentiation operations in large finite mathematical groups. The cryptographic strength of these systems is derived from the believed computational intractability of computing logarithms in these groups. The most commonly seen groups are the multiplicative group of Z_p (the integers modulo a prime p) and F_{2^m} (characteristic 2 finite fields). The primary advantages of these groups have been their rich theory, easily understood structure, and straightforward implementation. However, they are not the only groups that have the requisite properties. In particular the mathematical structures known as elliptic curves have the requisite mathematical properties, have a rich theory, and are especially amenable to efficient implementation in hardware or software.

The algebraic system defined on the points of an elliptic curve provides an alternate means to implement the ElGamal and ElGamal-like public key encryption and signature protocols. These protocols are described in the literature in the algebraic system Z_p, integers modulo p, where p is a prime. For example, the Digital Signature Algorithm (DSA) defined in ANSI X9.30-1-1995 [ANS 95] and in NIST's FIPS 186-1993 [P1363; NIS 93], is an ElGamal-like signature scheme defined over Z_p. Precisely the same protocol for signing can be defined over the points on an elliptic curve.

Elliptic curves as algebraic/geometric entities have been studied extensively for the past 150 years, and from these studies has emerged a rich and deep theory. Elliptic curve systems as applied to ElGamal protocols were first proposed in 1985 independently by Neal Koblitz from the University of Washington, and Victor Miller, who was then at IBM, Yorktown Heights.

The security of the cryptosystems using elliptic curves hinges on the intractability of the discrete logarithm problem in the algebraic system. Since 1985, this problem has received attention from leading mathematicians around the world. Unlike the case of the discrete logarithm problem in finite fields, or the problem of factoring integers, there is no subexponential-time algorithm known for the elliptic curve discrete logarithm problem. The best algorithm known to date takes fully exponential time.

The primary advantage of elliptic curve systems is their high cryptographic strength relative to the key size. The attractiveness of the elliptic curve cryptosystems will increase relative to other public-key cryptosystems, as computing power improvements force an increase in the key size. The significantly shorter key size results in significantly shorter certificates and system parameters. These advantages manifest themselves in many ways, including:

- Storage efficiencies,
- Bandwidth savings, and
- Computational efficiencies.

The computational efficiencies lead in turn to:

- Higher speeds,
- Lower power consumption, and
- Code size reductions.

The computational efficiencies available with the use of these implementations, combined with the efficiencies of elliptic curves in general, are particularly beneficial in applications where bandwidth, processing capacity, power availability, or storage are constrained. Examples are wireless communications, handhold computing, broadcast, and smart card applications.

Associated with any finite field F_q there are on the order of q different elliptic curves that can be formed and used for the cryptosystems. Thus, for a fixed finite field with q elements and with q large, there are many choices for the elliptic curve group. Since each elliptic curve operation requires a number of more basic operations in the underlying finite field F_q, a finite field may be selected with a very efficient software or hardware implementation, and there remain an enormous number of choices for the cryptosystem.

6.9.1. Definitions, Abbreviations, Symbols, and Notation

6.9.1.1. Definitions And Abbreviations

addition rule	An *addition rule* describes the addition of two elliptic curve points P_1 and P_2 to produce a third elliptic curve point P_3.
basis	A representation of the elements of the finite field F_{2^m}. Two special kinds of basis are *optimal normal basis* and *polynomial basis*.
bit string	A bit string is an ordered sequence of 0s and 1s.
characteristic 2 finite field	A finite field containing 2^m elements, where $m \geq 1$ is an integer.
cryptographic hash function	A (mathematical) function that maps values from a large (possibly very large) domain into a smaller range. It satisfies the following properties: 1. it is computationally infeasible to find any input which maps to any pre-specified output; 2. it is computationally infeasible to find any two distinct inputs that map to the same output.
ECDSA-like	Elliptic curve analog of the NIST Digital Signature Algorithm (DSA).
EC	Elliptic Curve.
elliptic curve	An *elliptic curve* is a set of points specified by two parameters a and b, which are elements of a field F_q. The elliptic curve is said to be defined over F_q, and F_q is sometimes called the *underlying field*. If q is an odd prime p, $p > 3$, (so the field is F_p), then the Weierstrass equation defining the curve is of the form $y^2 = x^3 + ax + b$, where $((4a^3 + 27b^2) \bmod p) \neq 0$. If q is a power of 2 (so the field is F_{2^m}), then the Weierstrass equation defining the curve is of the form $y^2 + xy = x^3 + ax^2 + b$, where $b \neq 0$.
elliptic curve key pair	Given particular elliptic curve parameters, an *elliptic curve key pair* consists of an elliptic curve private key and the corresponding elliptic curve public key.

elliptic curve private key	Given particular elliptic curve parameters, an *elliptic curve private key* consists of a statistically unique and unpredictable integer d in the interval $[2, n - 2]$.
elliptic curve public key	Given particular elliptic curve parameters and an elliptic curve private key d, the corresponding *elliptic curve public key* consists of the elliptic curve point $Q = dP$.
elliptic curve parameters	These parameters specify an underlying field F_q, the type of basis used to represent the elements of F_q, the equation of an elliptic curve over F_q, an elliptic curve point P of prime order, and the order n of P.
elliptic curve point	If E is an elliptic curve defined over a field F_q, then an *elliptic curve point P* is either (i) a pair of field elements (x_P, y_P) (where $x_P, y_P \in F_q$) such that the values $x = x_P$ and $y = y_P$ satisfy the equation defining E, or (ii) a special point O called the *point at infinity*.
nonsupersingular	If an elliptic curve is not supersingular, it is called *nonsupersingular*. Only nonsupersingular elliptic curves are considered in this specification.
octet	An *octet* is a bit string of length 8. A hexadecimal string of length 2 represents an octet. The first hexadecimal digit represents the four leftmost bits of the octet, and the second hexadecimal digit represents the four rightmost bits of the octet. For example, 9*D* represents the bit string 10011101. An octet also represents an integer in the interval [0,255]. For example, 9*D* represents the integer 157.
octet string	An octet string is an ordered sequence of octets.
order of a point	The *order of a point P* is the smallest positive integer n such that $nP = \mathsf{O}$ (the point at infinity).
order of a curve	The *order of an elliptic curve E* defined over the field F_q is the number of points on the elliptic curve E, including O, and is denoted by $\#E(F_q)$.
owner	The entity whose identity is associated with a private/public key pair.
point compression	Let P be a point (x_P, y_P) on an elliptic curve E defined over a field F_q. *Point compression* allows the point P to be represented using x_P and a single additional bit $\tilde{y}_P$ derived from x_P and y_P. If q is a prime number, then $\tilde{y}_P$ is the rightmost bit of y_P. If q is a power of 2, then $\tilde{y}_P$ is 0 if $x_P = 0$; if $x_P \neq 0$, then $\tilde{y}_P$ is equal to the rightmost bit of the field element $y_P \bullet x_P^{-1}$.
point compression octet (*PC*)	The rightmost bit of *PC* shall be equal to the value of $\tilde{y}_P$. The second rightmost bit is 1 if this indicates the point at infinity and is zero otherwise. The remaining bits of the octet shall be set to zero.
prime finite field	A finite field containing p elements, where p is an odd prime number.
scalar multiplication	If k is a positive integer, then kP denotes the point obtained by adding together k copies of the point P. The process of computing kP from P and k is called *scalar multiplication*.
The Secure Hash Algorithm,	SHA-1 implements a hash function which maps messages of length

Revision 1 (SHA-1)	less than 2^{64} bits to hash values of length exactly 160 bits.
supersingular	An elliptic curve E defined over F_p is *supersingular* if the order of E is $p + 1$. An elliptic curve defined over F_{2^m} is supersingular if its defining equation is of the form: $y^2 + cy = x^3 + ax + b$ where: $a,b,c \in F_{2^m}$ and $c \neq 0$. The use of such a curve is prohibited by this specification.
verifier	The entity that verifies the authenticity of a digital signature.
x-coordinate	The *x-coordinate* of an elliptic curve point, $P = (x_P, y_P)$, is x_P.
y-coordinate	The *y-coordinate* of an elliptic curve point, $P = (x_P, y_P)$, *is* y_P.

6.9.1.2. Symbols And Notation

$\lceil a \rceil$	Ceiling: the smallest integer $\geq a$. For example, $\lceil 5 \rceil = 5$ and $\lceil 5.3 \rceil = 6$.
$\lfloor a \rfloor$	Floor: the largest integer $\leq a$. For example, $\lfloor 5 \rfloor = 5$ and $\lfloor 5.3 \rfloor = 5$.
$[a,b]$	The interval of integers between and including a and b.
$X \oplus Y$	Bitwise exclusive-or of two bit strings X and Y of the same bit length.
$X\|\|Y$	Concatenation of two strings X and Y. X and Y are either both bit strings, or both octet strings.
$\|\|X\|\|$	Length in octets of the octet string X.
$\log_2 x$	The logarithmic function to the base 2.
a mod n	The unique remainder r, $0 \leq r \leq n - 1$, when integer a is divided by n. For example, 23 mod 7 = 2 since $23 = 7 \bullet 3 + 2$.
Z_p or F_p	The finite field containing p elements, where p is prime.
F_{2^m}	The finite field containing 2^m elements.
m	The *degree* of the finite field F_{2^m}.
F_q	The finite field containing q elements. For this specification, q will either be a prime number (p) or a power of 2 (2^m).
t	The length of a field element in bits; $t = \lceil \log_2 q \rceil$. In particular, if $q = 2^m$, then a field element in F_{2^m} can be represented as a bit string of bit length $t = m$.
l	The length of a field element in octets; $l = \lceil t / 8 \rceil$.
E	An elliptic curve.
P	A *base* point (x_P, y_P) on an elliptic curve.
0	A special point on an elliptic curve, called the point at infinity. This is the additive identity of the elliptic curve group
n	The order of the point P. For this specification, n is a prime number.
h	The maximum length, in bits, of an integer in the interval $[0, n - 1]$; $h = \lceil \log_2 n \rceil$.
f	The maximum length, in octets, of an integer in the interval $[0, n - 1]$; $f = \lceil h / 8 \rceil$.
$E(F_q)$	The set of all points on an elliptic curve E defined over F_q and including the point at infinity 0.

$\#E(F_q)$	If E is defined over F_q, then $\#E(F_q)$ denotes the number of points on the curve. $\#E(F_q)$ is called the order of the curve E.
$\tilde{y}_P$	The representation of the y-coordinate of a point P when point compression is used.
M	Message to be signed.
M'	Message as received.
mod	Modulo.
mod n	Arithmetic modulo n.
p	An odd prime number.
Q	EC public key.

6.9.2. Mathematical Conventions

6.9.2.1. Finite Field Arithmetic

This section describes the representations that shall be used for the elements of the underlying finite field F_q.

6.9.2.1.1. The Finite Field F_p

Let p be an odd prime, $p > 3$. The finite field F_p is comprised of the set of integers $\{0, 1, 2, ..., p - 1\}$.

6.9.2.1.2. The Finite Field F_{2^m}

The finite field F_{2^m} is comprised of all bit-strings of bit length m.

6.9.2.1.3. Polynomial Basis

Let $f(x) = x^m + f_{m-1}x^{m-1} + ... + f_2x^2 + f_1x + f_0$, $f_i \in F_2$, be an irreducible polynomial of degree m over F_2, i.e., $f(x)$ cannot be factored into two polynomials over F_2, each of degree less than m. The *finite field* F_{2^m} is comprised of all polynomials over F_2 of degree less than m:

$$F_{2^m} = \{a_{m-1}x^{m-1} + ... + a_2x^2 + a_1x + a_0: a_i \in \{0,1\}\},$$

i.e., $\{1, x, x^2, \ldots, x^{m-1}\}$ is the polynomial basis of F_{2^m}. The field element $(a_{m-1}x^{m-1} + ... + a_2x^2 + a_1x + a_0)$ is usually denoted by the binary string $(a_{m-1}...a_1a_0)$ of length m, so that

$$F_{2^m} = \{(a_{m-1}...a_1a_0): a_i \in \{0,1\}\}.$$

Thus the elements of F_{2^m} can be represented by the set of all binary strings of length m.

Field elements are added and multiplied as follows:

- *Field addition*: $(a_{m-1}...a_1a_0) + (b_{m-1}...b_1b_0) = (c_{m-1}...c_1c_0)$ where $c_i = a_i + b_i$ in the field F_2. That is, field addition is performed componentwise.
- *Field multiplication*: $(a_{m-1}...a_1a_0) \cdot (b_{m-1}...b_1b_0) = (r_{m-1}...r_1r_0)$, where the polynomial $(r_{m-1}x^{m-1} + ... + r_1x + r_0)$ is the remainder when the polynomial $(a_{m-1}x^{m-1} + ... + a_2x^2 + a_1x + a_0) \cdot (b_{m-1}x^{m-1} + ... + b_2x^2 + b_1x + b_0)$ is divided by $f(x)$ over F_2.

This method of representing F_{2^m} is called a *polynomial basis representation*.

For performance reasons we choose the irreducible polynomial *f(x)* to have as few nonzero terms as possible (as *f(x)* is not divisible by *x* and has an odd number of terms, it has at least three terms). If, for a fixed *m*, there are two polynomials with the same number of terms, we choose the one for which *f(2)* is minimum (here *f(2)* is evaluated as a polynomial over the integers). For all field sizes proposed in this specification *f(x)* will be a trinomial, i.e., of the form $x^m + x^k + 1$ or a pentomial, i.e., of the form $x^m + x^k + x^j + x^i + 1$.

6.9.2.1.4. Field Sizes and Conversion Between Different Bases

The following field sizes are stipulated in order to enhance interoperability for F_{2^m}:

$$m = 113, 163, 191, 239, 359, 431$$

For conversion between two (polynomial) bases B_1 and B_2 for the field over F_2 we refer to Section 6.9.9.1.3.

For F_p, the prime p shall be in the range of 2^{113} to 2^{256}.

6.9.2.1.5. Curve Parameters and Identifiers

Curve parameters for the field sizes and curve types recommended herein follow. Each curve should be assigned a curve identifier (CID) in order to simplify its use and interoperability. The CID should be represented by an octet. Example curve parameters have been taken from X9.62 whenever possible. The assigning of CIDs does not preclude the use of other curves. The curves below can be used together with user defined algorithms (e.g., for key exchange, see Section 6.2.3.5).

Random curves over F_{2^m}

m = 113-bits

CID Octet =

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
0	0	0	0	0	0	0	1

The field $F_{2^{113}}$ is generated by the irreducible trinomial:

f = 020000 00000000 00000000 00000201

The curve E: $y^2 + xy = x^3 + ax^2 + b$ over $F_{2^{113}}$

SEED = 106B2D4C 1DF4D696 E6768756 15175C93 B87B1BC3

a = 0040DC 7BDE0697 D611D0EA 04048409

b = 01F92F 62E93FF8 760B6C1A F67E15CE

Base point *G* (with point compression):
0301F5BA CF3501B7 D83AC956 9E6B9613

Order of *G*:

n = 010000 00000000 00D0DD64 9933E2D9

h = 02

m = 163-bits (from ANSI X9.62)

CID Octet =

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
0	0	0	0	0	0	1	0

The field $F_{2^{163}}$ is generated by the irreducible pentanomial:

f = 08 00000000 00000000 00000000 00000000 00000107

The curve *E*: $y^2 + xy = x^3 + ax^2 + b$ over $F_{2^{163}}$

SEED = D2C0FB15 760860DE F1EEF4D6 96E67687 56151754

a = 07 2546B543 5234A422 E0789675 F432C894 35DE5242
b = 00 C9517D06 D5240D3C FF38C74B 20B6CD4D 6F9DD4D9

Base point *G* (with point compression):
0307 AF699895 46103D79 329FCC3D 74880F33 BBE803CB

Order of *G*:
n = 04 00000000 00000000 0001E60F C8821CC7 4DAEAFC1
h = 02

m = 191-bits (from ANSI X9.62)

CID Octet =

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
0	0	0	0	0	0	1	1

The field F_2^{191} is generated by the irreducible trinomial:
f = 80000000 00000000 00000000 00000000 00000201
The curve *E*: $y^2 + xy = x^3 + ax^2 + b$ over F_2^{191}

SEED = 4E13CA54 2744D696 E6768756 1517552F 279A8C84
a = 2866537B 67675263 6A68F565 54E12640 276B649E F7526267
b = 2E45EF57 1F00786F 67B0081B 9495A3D9 5462F5DE 0AA185EC

Base point *G* (with point compression):
02 36B3DAF8 A23206F9 C4F299D7 B21A9C36 9137F2C8 4AE1AA0D

Order of *G*:
n = 40000000 00000000 00000000 04A20E90 C39067C8 93BBB9A5
h = 02

m = 239-bits (from ANSI X9.62)

CID Octet =

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
0	0	0	0	0	1	0	0

The field F_2^{239} is generated by the irreducible trinomial:
f = 8000 00000000 00000000 00000000 00000000 00000010 00000001
The curve *E*: $y^2 + xy = x^3 + ax^2 + b$ over F_2^{239}

SEED = D34B9A4D 696E6768 75615175 CA71B920 BFEFB05D
a = 3201 0857077C 5431123A 46B80890 6756F543 423E8D27 87757812 5778AC76
b = 7904 08F2EEDA F392B012 EDEFB339 2F30F432 7C0CA3F3 1FC383C4 22AA8C16

Base point *G* (with point compression):
025792 7098FA93 2E7C0A96 D3FD5B70 6EF7E5F5 C156E16B 7E7C8603 8552E91D

Order of *G*:
n = 2000 00000000 00000000 00000000 000F4D42 FFE1492A 4993F1CA D666E447
h = 04

Random curves over F_p
p = 113-bits

CID Octet =

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
0	0	0	0	0	1	0	1

The field F_p is generated by the prime:
p = 9263910021405187536998134712315669
The curve *E*: $y^2 = x^3 + ax + b$ over F_{113}

SEED = 10D488F3 EB4B2AFD 9134D696 E6768756 151754A6

r = 0018E5 8515A4E3 6A6DB07E C53D7491
a = 00FB5E B4B66007 70467852 25CE9DCA
b = 01BC31 341D57E6 FB75C850 0C38BD72

Base point *G* (with point compression):
0300FA46 F57A6313 20092F0D A8DD2E33

Order of *G*:
n = 01C8BE FBAE245F 039911DC ABDCC82D
h = 01

p = 163-bits

CID Octet =	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
	0	0	0	0	0	1	1	0

The field F_p is generated by the prime:
p = 744157000185125307832505951007652061060660492226 7

The curve is *E*: $y^2 = x^3 + ax + b$ over F_{163}

SEED = 33855928 86AB4D69 6E676875 61517593 5F3CA3E1
r = 00 3FEA47B8 B292641C 57F9BF84 BAECDE8B B3ADCE30
a = 04 3182D283 FCE38807 30C9A2FD D3F60165 29A166AF
b = 02 0C61E945 9E53D887 1BCAADC2 DFC8AD52 25228035

Base point *G* (with point compression):
0300 17E70122 77E1B4E4 3F7BF746 57E8BE08 BACA175B

Order of *G*:
n = 05 177B8A2A 0FD6A4FF 55CCA7B8 A1E21C88 BD53B2C1
h = 01

p = 192-bits (from ANSI X9.62)

CID Octet =	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
	0	0	0	0	0	1	1	1

The field F_p is generated by the prime:
p = 6277101735386680763835789423207666416083908700390324961279
The curve *E*: $y^2 = x^3 + ax + b$ over F_{192}

SEED = 3045AE6F C8422F64 ED579528 D38120EA E12196D5
r = 3099D2BB BFCB2538 542DCD5F B078B6EF 5F3D6FE2 C745DE65
a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC
b = 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1

Base point *G* (with point compression):
03 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012

Order of *G*:
n = FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831
h = 01

p = 239-bits (from ANSI X9.62)

CID Octet =	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
	0	0	0	0	1	0	0	0

The field F_p is generated by the prime:
p = 8834235323891921647916487503603088853144765972529603627924508606096998 39

The curve *E*: $y^2 = x^3 + ax + b$ over F_{239}

SEED = E43BB460 F0B80CC0 C0B07579 8E948060 F8321B7D
r = 28B8 5EC1ECC1 9EFE769E B741A6D1 BA29476A A5A8F261 0957D6EF E78D3783
a = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FFFFFFFF 80000000 00007FFF FFFFFFFC
b = 6B01 6C3BDCF1 8941D0D6 54921475 CA71A9DB 2FB27D1D 37796185 C2942C0A

Base point *G* (with point compression):
020FFA 963CDCA8 816CCC33 B8642BED F905C3D3 58573D3F 27FBBD3B 3CB9AAAF

Order of *G*:
n = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FF9E5E9A 9F5D9071 FBD15226 88909D0B
h = 01

Koblitz curves over F_2^m

m = 163-bits
CID Octet = **8 7 6 5 4 3 2 1**
0 0 0 0 1 0 0 1
The field F_2^{163} is generated by the irreducible pentanomial:
f = 08 00000000 00000000 00000000 00000000 000000C9
The curve *E*: $y^2 + xy = x^3 + ax^2 + b$ over F_2^{163}

SEED = NO
a = 00 00000000 00000000 00000000 00000000 00000001
b = 00 00000000 00000000 00000000 00000000 00000001

Base point *G* (with point compression):
0302 FE13C053 7BBC11AC AA07D793 DE4E6D5E 5C94EEE8

Order of *G*:
n = 04 00000000 00000000 00020108 A2E0CC0D 99F8A5EF
h = 02

m = 239-bits
CID Octet = **8 7 6 5 4 3 2 1**
0 0 0 0 1 0 1 0
The field F_2^{239} is generated by the irreducible trinomial:
f = 8000 00000000 00000000 40000000 00000000 00000000 00000000 00000001
The curve *E*: $y^2 + xy = x^3 + ax^2 + b$ over F_2^{239}

SEED = NO
a = 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
b = 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

Base point *G* (with point compression):
0329A0 B6A887A9 83E97309 88A68727 A8B2D126 C44CC2CC 7B2A6555 193035DC

Order of *G*:
n = 2000 00000000 00000000 00000000 005A79FE C67CB6E9 1F1C1DA8 00E478A5
h = 04

6.9.2.2. Data Representation

The data types in this specification are octet strings, integers, field elements, and points.

Figure 49 provides a cross-reference for the sections defining conversions between data types that shall be used in the algorithms specified in this specification. The number on a line is the section number where the conversion technique is specified.

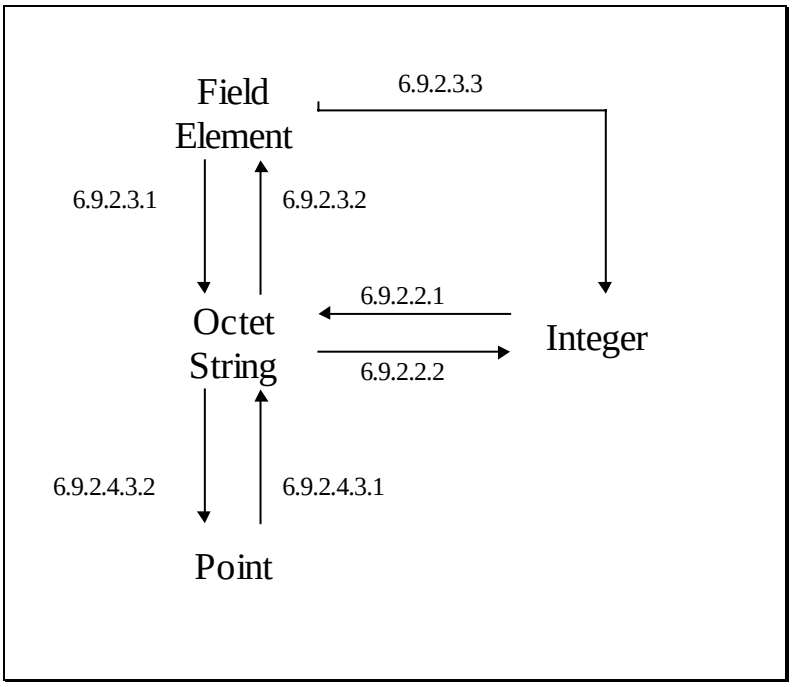


Figure 49. Data Types and Conversion Conventions.

6.9.2.2.1. Integer-to-Octet-String Conversion

Input: A nonnegative integer x, and the intended length k of the octet string satisfying:

$$2^{8k} > x.$$

Output: An octet string M of length k octets.

1. Let $M_1, M_2, ..., M_k$ be the octets of M from leftmost to rightmost.
2. The octets of M shall satisfy:

$$x = \sum_{i=1}^{k} 2^{8(k-i)} M_i.$$

6.9.2.2.2. Octet-String-to-Integer Conversion

Input: An octet string M of length k octets.

Output: An integer x.

1. Let $M_1, M_2, ..., M_k$ be the octets of M from leftmost to rightmost.
2. M shall be converted to an integer x *satisfying:*

$$x = \sum_{i=1}^{k} 2^{8(k-i)} M_i.$$

6.9.2.3. Finite Field Element Representations

6.9.2.3.1. Field-Element-to-Octet-String Conversion

Input: An element α in the field F_q.

Output: An octet string S of length $l = \lceil t / 8 \rceil$ octets, where $t = \lceil \log_2 q \rceil$.

1. If q is an odd prime, then α must be an integer in the interval: $[0, q - 1]$;
   α shall be converted to an octet string of length l octets using the technique specified in Section 6.9.2.2.1.
2. If $q = 2^m$, then α must be a bit string of length m bits. Let $s_1,s_2,\ldots,s_m$ be the bits of α from leftmost to rightmost. Let $S_1,S_2,\ldots,S_l$ be the octets of S from leftmost to rightmost. The rightmost bit s_m shall become the rightmost bit of the last octet S_l, and so on through the leftmost bit s_1, which shall become the $(8l - m + 1)^{th}$ bit of the first octet S_1. The leftmost $(8l - m)$ bits of the first octet S_1 shall be zero.

6.9.2.3.2. Octet-String-to-Field-Element Conversion

Input: An octet string S of length l octets, and an indication of the field F_q used.

Output: An element α in F_q.

1. If q is an odd prime, then convert S to an integer α using the technique specified in Section 6.9.2.2.2. It is an error if α does not lie in the interval $[0, q - 1]$.
2. If $q = 2^m$, then α shall be a bit string of length m bits. Let $s_1,s_2,\ldots,s_m$ be the bits of α from leftmost to rightmost. Let $S_1,S_2,\ldots,S_l$ be the octets of S from leftmost to rightmost. The rightmost bit of the last octet S_l shall become the rightmost bit s_m, and so on through the $(8l - m + 1)^{th}$ bit of the first octet S_1, which shall become the leftmost bit s_1. The leftmost $(8l - m)$ bits of the first octet S_1 are not used.

6.9.2.3.3. Field-Element-to-Integer Conversion

Input: An element α in the field F_q.

Output: An integer x.

1. If q is an odd prime then $x = \alpha$ (no conversion is required).
2. If $q = 2^m$, then α must be a bit string of length m bits. Let $s_1,s_2,\ldots,s_m$ be the bits of α from leftmost to rightmost. α shall be converted to an integer x satisfying:

$$x = \sum_{i=1}^{m} 2^{(m-i)} s_i$$

6.9.2.4. Elliptic Curve Parameters, Keys, and Point Representations

6.9.2.4.1. Elliptic Curve Parameters

Elliptic curve parameters may either be common to several key pairs (*common elliptic curve parameters*) or specific to one key pair (*specific elliptic curve parameters*). The elliptic curve parameters may be public; the security of the system does not rely on these parameters being secret. Two cases are distinguished:

1. When the underlying field is F_p (p an odd prime), and

2. When the underlying field is F_{2^m}.

6.9.2.4.1.1. Elliptic Curve Parameters Over F_p

Elliptic curve parameters over F_p shall consist of the following parameters:

a. A field size $q = p$ which defines the underlying finite field F_q, where p, $p > 3$, shall be a prime number;
b. Two field elements a and b in F_q which define the equation of the elliptic curve E: $y^2 = x^3 + ax + b$;
c. Two field elements x_P and y_P in F_q which define a point $P = (x_P, y_P)$ on E of prime order; and
d. The order n of the point P. It must be the case that $n > 2^{150}$.

6.9.2.4.1.2. Elliptic Curve Parameter Validation Over F_p

The following conditions shall be used to verify the elliptic curve parameters over F_q. A user of the system may alternately verify these conditions.

a. Verify that $q = p$ is an odd prime number.
b. Verify that $(4a^3 + 27b^2) \neq 0 \pmod{p}$.
c. Verify that $y_P^2 \equiv x_P^3 + ax_P + b \pmod{p}$.
d. Verify that n is prime and that $n > 2^{150}$.
e. Verify that $nP = 0$.
f. Verify that the MOV condition holds (Section 6.9.8.1).

If any of the above verifications fail, then reject the elliptic curve parameters.

6.9.2.4.1.3. Elliptic Curve Parameters Over F_{2^m}

Elliptic curve parameters over F_{2^m} shall consist of the following parameters:

a. A field size $q = 2^m$ which defines the underlying finite field $F_{q;}$
b. Two field elements a and b in F_q which define the equation of the elliptic curve E;
c. Two field elements x_P and y_P in F_q which define a point $P = (x_P, y_P)$ on E of prime order; and
d. The order n of the point P. It must be the case that $n > 2^{150}$.

6.9.2.4.1.4. Elliptic Curve Parameter Validation Over F_{2^m}

The following conditions may be used to verify the elliptic curve parameters over F_{2^m}:

a. Verify that $q = 2^m$ for some m.
b. Verify that $b \neq 0$.
c. Verify that $y_P^2 + x_P y_P = x_P^3 + ax_P^2 + b$ in F_{2^m}
d. Verify that n is prime and that $n > 2^{150}$.
e. Verify that $nP = 0$.
f. Verify that the MOV condition holds (Section 6.9.8.1).

If any of the above verifications fail, then reject the elliptic curve parameters.

6.9.2.4.2. Key Generation

Given particular elliptic curve parameters, an elliptic curve key pair shall be generated by performing the following operations:

1. Select a statistically unique and unpredictable integer d in the interval $[2, n - 2]$.

It is acceptable to use a random or pseudorandom number. If a pseudorandom number is used, it shall be generated using the procedure of ANSI X9.30, Part 1 [ANS95], Appendix B.

2. Compute the point $Q = (x_Q, y_Q) = dP$.
3. The key pair is (Q,d), where Q is the public key, and d is the private key.

The key and algorithm details are conveyed by the Security Services Information Element as described in Section 6.2.3.5.

6.9.2.4.3. Representing an Elliptic Curve Point

A finite point on E is specified by two elements x and y in F_q satisfying the defining equation for E. These are called the *affine coordinates* for the point. The point at infinity O has no affine coordinates. For purposes of internal computation, it is most convenient to represent O by a pair of coordinates (x,y) not on E. In the characteristic 2 case the simplest choice is O=(0,0). For $q=p$, one chooses O=(0,0) unless $b=0$ in which case O=(0,1).

An elliptic curve point P (which is not the point at infinity $\mathcal{O}$) is represented by two field elements, the x-coordinate of P and the y-coordinate of P: $P = (x_P, y_P)$. The point can be represented compactly by storing only the x-coordinate x_P and a certain bit $\tilde{y}_P$ derived from the x-coordinate x_P and the y-coordinate y_P. To enhance interoperability we mandate that implementations externally represent elliptic curve points in the *compressed form* as described in Section 6.9.1 (Definitions, Abbreviations, Symbols, and Notation).

6.9.2.4.3.1. Point-to-Octet-String conversion

Input: An elliptic curve point (x_1, y_1).

Output: An octet string PO, of length $l + 1$ octets.

1. Convert the field element x_1 to an octet string X_1 (see Section 6.9.2.3.1).
2. Compute the bit $\tilde{y}_1$ (see Section 6.9.1).
3. Assign the single octet PC the value 00 if $\tilde{y}_1$ is 0, or the value 01 if $\tilde{y}_1$ is 1.
4. The result is the octet string $PO = PC||X_1$.

6.9.2.4.3.2. Octet-String-to-Point conversion

Input: An octet string PO.

Output: An elliptic curve point (x_1, y_1).

1. Verify that the leftmost octet of PO is 00 or 01. Parse PO as follows: $PO = PC||X_1$ where PC is a single octet, and X_1 is an octet string of length l octets. Set the bit $\tilde{y}_1$ to be equal to 0 if PC = 00, or 1 if PC = 01 . It is an error if PO is not $l + 1$ octets in length.
2. Convert X_1 to a field element x_1 (see Section 6.9.2.3.2).
3. Convert $(x_1, \tilde{y}_1)$ to an elliptic curve point (x_1, y_1) (see Section 6.9.1).
4. The result is (x_1, y_1).

6.9.3. The Elliptic Curve Digital Signature Algorithm (ECDSA-like)

6.9.3.1. Signature Generation

This section describes the ECDSA-like signature generation process. This signature is suggested for standardization in ISO/IEC JTC1. It is similar to the DSA scheme and uses also some ideas from [P1363; AMO 91].

The signature generation process consists of three steps: message digesting, elliptic curve computations, and modular computations.

The input to the signature process is:

- The message, *M*, of arbitrary length, which is represented by a bit string.
- Elliptic curve parameters q, a, b, $P=(x_P, y_P)$, and n.
- An elliptic curve private key d.

To sign a message *M*, *A* should do the following.

6.9.3.1.1. Message Digesting

Compute the hash value $e = H(M)$ using the hash function SHA-1 per ANSI X9.30-199X, Part 2[ANS93]. e is represented as an integer of length 160 bits.

6.9.3.1.2. Elliptic Curve Computations

1. Select a statistically unique and unpredictable integer k in the interval $[2, n - 2]$.
2. Compute the elliptic curve point $(x_1, y_1) = kP$.
3. Convert the field element x_1 to an integer $\overline{x}_1$, as described in Section 6.9.2.3.3.
4. Set $r = \overline{x}_1 \mod n$.
5. If $r = 0$ then go to step 1.

6.9.3.1.3. Modular Computations

1. Compute $s = (kr - e)\, d^{-1} \mod n$ in the case of ECDSA-like.
2. If $s = 0$ then go to Step 1 of Section 6.9.3.1.2.

6.9.3.1.4. The Signature

The output from the signature generation process shall be the signature for *M*, which is represented as the two integers, r and s.

Note: If k is chosen correctly (see Section 6.9.3.1.2, item 1), the probability that either $r = 0$ or $s = 0$ is negligibly small.

6.9.3.2. Signature Verification

This section describes the ECDSA-like signature verification process.

The signature verification process consists of three steps: message digesting, elliptic curve computations, and signature checking.

The input to the signature verification process is:

- The received message, *M'*, represented as a bit string.
- The received signature for *M*, represented as the two integers, *r'* and *s'*.
- Elliptic curve parameters *q*, *a*, *b*, $P=(x_P, y_P)$, and *n*.
- An elliptic curve public key, *Q*.

To verify *A*'s signature on message *M*, *B* should do the following.

6.9.3.2.1. Message Digesting

Compute the hash value $e = H(M')$ using the hash function SHA-1 per ANSI X9.30-199X, Part 2.
e is represented as an integer of length 160 bits.

6.9.3.2.2.Elliptic Curve Computations for ECDSA-like

1. If *r'* is not an integer in the interval [1, *n* -1], then reject the signature.
2. If *s'* is not an integer in the interval [1, *n* -1], then reject the signature.
3. Compute $w = (r')^{-1} \bmod n$.
4. Compute $u_1 = sw \bmod n$ and $u_2 = ew \bmod n$.
5. Compute the elliptic curve point $(x_1, y_1) = u_2P + u_1Q$.

6.9.3.2.3.Signature Checking

1. Convert the field element x_1 to an integer $\bar{x}_1$, as described in Section 6.9.2.3.3.
2. Compute $v = \bar{x}_1 \bmod n$.
3. If *r' = v*, then the signature is verified and the verifier can have a high level of confidence that the received message was sent by the party holding the secret key *d* corresponding to *Q*.

If *r'* does not equal *v*, then the message may have been modified, the message may have been incorrectly signed by the signatory, or the message may have been signed by an impostor. The message should be considered invalid.

6.9.4. ECDSA-Like Asymmetric Authentication

An asymmetric authentication scheme based on elliptic curves can be derived from the signature scheme presented in Section 6.9.3 using the principle of "challenge-and-response."

6.9.4.1. Challenge

This section describes the challenge process.

B chooses at random a challenge integer *c*, where $0 < c < n$, and sends it to *A*.

6.9.4.2. Response Generation

This section describes the response generation process.

The response generation process consists of two steps: elliptic curve computations and modular computations.

The input to the response process is:

- The challenge *c*.
- Elliptic curve parameters.

- An elliptic curve private key *d*.

To generate a response to *B*'s challenge, *A* should do the following.

6.9.4.2.1. Elliptic Curve Computations

1. Choose a statistically unique and unpredictable integer k in the interval [2, n - 2].
2. Compute the curve point $R = (x_1,y_1) := kP$.
3. Convert the field element x_1 to the integer $\bar{x}_1$ as described in Section 6.9.2.3.3
4. Set $r = \bar{x}_1 \mod n$.
5. If $r = 0$, go to step 1.

6.9.4.2.2. Modular Computations

1. Compute $s := (kr - c)d^{-1} \mod n$ in the case of ECDSA-like.
2. If $s = 0$ then go to Step 1 of 6.9.4.2.1.

6.9.4.2.3. The response

The *response* (*r*,*s*) shall be sent to *B*.

6.9.4.3. Response verification

This section describes the response verification process.

There are two steps to the response verification process: elliptic curve computations and response checking.

The input to the response verification process is:

- The challenge *c*.
- Elliptic curve parameters.
- An elliptic curve public key *Q*.
- The response (*r*,*s*).

To verify *A*'s response, *B* should do the following.

6.9.4.3.1. Elliptic Curve Computations for ECDSA-like

1. If r' is not an integer in the interval [1, n -1], then reject the response.
2. If s' is not an integer in the interval [1, n -1], then reject the response.
3. Compute $w = (r's')^{-1} \mod n$.
4. Compute $u_1 = sw \mod n$ and $u_2 = cw \mod n$.
5. Compute the elliptic curve point $(x_1, y_1) = u_2P + u_1Q$.

6.9.4.3.2. Response Checking

1. Convert the field element x_1 to an integer $\bar{x}_1$, as described in Section 6.9.2.3.3.
2. Compute $v = \bar{x}_1 \mod n$.
3. If $r' = v$, then the response is verified and the verifier can have a high level of confidence that the response was sent by the party holding the secret key *d* corresponding to *Q*.

6.9.5. Elliptic Curve Key Agreement Scheme—Diffie-Hellman Analogue (ECKAS-DH)

This section describes the Elliptic Curve Key Agreement Scheme—Diffie-Hellman Analogue.

The input to the scheme is:

- Elliptic curve parameters.

However, it is not necessary that the users know the order *n* of the point *P*. The public and private keys of the users are not involved in this protocol.

6.9.5.1. Key Computations

To agree upon a common key, *A* and *B* do the following calculations:

1. *A* chooses a random integer *a* and determines the point *aP*. *A* sends *aP* to *B*.
2. *B* chooses a random integer *b* and determines the point *bP*. *B* sends *bP* to *A*.
3. *A* determines *a*(*bP*), *B* determines *b*(*aP*). Both points coincide and constitute the common key.

6.9.6. References

Elliptic curves cryptosystems were first proposed in 1985 independently by Neal Koblitz [P1363; Kob 87] and Victor Miller [P1363; Mil 86]. Since then, much research has been done towards improving the efficiency of these systems and evaluating their security. For a summary of this work, consult [P1363; Men 93]. A description of a hardware implementation of an elliptic curve cryptosystem can be found in [P1363; AMV 93].

Three good references on the theory of finite fields are the books of McEliece [P1363; Mce 87], Lidl and Niederreiter [P1363; LN 87] and Jungnickel [P1363; Jun 93]. The article [P1363; ABM 93] discusses how to efficiently perform arithmetic operations in finite fields of characteristic 2. A hardware implementation of arithmetic in such fields which exploits the properties of optimal normal bases is described in [P1363; AMO 91].

The NIST Digital Signature Algorithm (DSA) is described in [ASN 95] and [P1363; NIS 93]. The Secure Hash Algorithm (P1363; SHA-1) is described in [P1363; ANS 93] and [P1363; NIS 94]. Abstract Syntax Notation One (P1363; ASN.1), Basic Encoding Rules (BER) and Distinguished Encoding Rules (DER) are described in [P1363; ISO ??a], [P1363; ISO??b] and [P1363; ISO??c] respectively.

[ANS95] ANSI X9.30-1995, Part 1: *Public key cryptography using irreversible algorithms for the financial services industry: The Digital Signature Algorithm (Revised)*

[P1363] IEEE P1363: Standard for Public Key Cryptography (Draft), ftp://stdsbbs.ieee.org/pub/p1363/predrafts/.

[MOV 93] A. MENEZES, T. OKAMOTO and S. VANSTONE, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Transactions on Information Theory **39** (1993), 1639-1646.

[FR 94] G. FREY and F. RUECK, *A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves*, Mathematics of Computation **62** (1994), 865-874.

[BDR 96] M. Blaze, W.Diffe, R.Rivest, B.Schneier, T.Shimomura, E. Thompson, and M. Wiener, *Minimal key lengths for symmetric ciphers to provide adequate commercial security*, January 1996.

[Odl 95] A. Odlyzko, The Future of Integer Factorization, Cryptobytes, volume 1, number 2, summer 1995, 5-12.

[IXR 93] Ian F. Blake, XuHong Gao, Ronald C. Mullin, Scott A. Vanstone and Tomik Yaghoobian, *Applications of Finite Fields*, Kluwer Academic Publishers, 1993.

6.9.7. Security Considerations [informative]

This section is provided as initial guidance for implementers of this specification. This information should be expected to change over time. Implementers should review the current state of the art at the time of implementation.

This section summarizes the best attacks known on the elliptic curve discrete logarithm problem, which is the basis for the security of elliptic curve systems. Estimates of security levels for elliptic curve parameters of various sizes are provided.

Notation

E denotes an elliptic curve over the finite field F_q, $P \in E(F_q)$ is a point of order n, where n is a prime number and $n > 2^{150}$.

The Elliptic Curve Discrete Logarithm Problem

The elliptic curve discrete logarithm problem (ECDLP) is the following: given E, P and $Q \in E$, determine the integer l, $0 \le l \le n-1$, such that $Q = lP$, provided that such an integer exists.

The best algorithm known to date for ECDLP is the Pollard-ρ method [P1363; Pol 78] which takes about $\sqrt{\pi n / 2}$ steps. By a *step* here, we shall mean an elliptic curve addition. The Pollard-ρ can be parallelized (see[P1363; OW 94]) so that if m processors are used then the expected number of steps by each processor before a single discrete logarithm is obtained is $(\sqrt{\pi n / 2}) / m$.

The special classes of elliptic curves, called *supersingular curves*, have been prohibited in this specification by the requirement of the MOV condition (see Section 6.9.8.1). This is because there is a method for efficiently reducing the discrete logarithm problem in these curves to the discrete logarithm problem in a finite field.

Software Attacks

We assume that a MIPS (Million Instructions Per Second) machine can perform 4×10^4 elliptic curve additions per second. (This estimate is indeed conservative—an ASIC built for performing elliptic curve operations over the field $F_{2^{155}}$ has a 40 MHz clock-rate and can perform roughly 40,000 elliptic operations per second.) Then, the number of elliptic curve additions that can be performed by a 1 MIPS machine in one year is

$$(4 \leftarrow 10^4) ?(60 \leftarrow 60 \leftarrow 24 \leftarrow 365) \cup 2^{40}.$$

Table 23 shows, for various values of n, the computing power required to compute a single discrete logarithm. As an example, if 10,000 computers each rated at 1,000 MIPS are available, and $n \cup 2^{150}$, then an elliptic curve discrete logarithm can be computed in 3,800 years.

Odlyzko [Odl 95] has estimated that if 0.1% of the world's computing power were available for one year to work on a collaborative effort to break some challenge cipher, then the computing power available would be 10^8 MIPS years in 2004 and 10^{10} to 10^{11} MIPS years in 2014.

Field size (in bits)	Size of *n* (in bits)	$\sqrt{\pi n / 2}$	MIPS years
155	150	2^{75}	3.8×10^{10}
158	153	2^{77}	1.1×10^{11}
209	204	2^{102}	5.0×10^{18}
210	205	2^{103}	7.1×10^{18}
239	234	2^{117}	1.6×10^{23}

Table 23: Computing Power to Compute Elliptic Curve Logarithms with the Pollard-ρ Method.

To put the numbers in Table 23 into some perspective, Table 24 (due to Odlyzko [Odl 95]) shows the computing power required to factor integers with current versions of the general number field sieve.

Size of *n* (in bits)	MIPS years
512	3×10^{4}
768	2×10^{8}
1024	3×10^{11}
1280	1×10^{14}
1536	3×10^{16}
2048	3×10^{20}

Table 24: Computing Power Required to Factor Integers Using the General Number Field Sieve.

Hardware Attacks

A more promising attack (for well-funded attackers) on elliptic curve systems would be to build special-purpose hardware for a parallel search. Van Oorschot and Wiener [P1363; OW 94] provide a detailed study of such a possibility. They estimated that if $n \cup 10^{36} \cup 2^{120}$, then a machine with m = 325,000 processors that could be built for about $10 million would compute a single discrete logarithm in about 35 days.

It must be emphasized that these estimates were made for specific elliptic curve parameters having $n \cup 10^{36} \cup 2^{120}$. This specification mandates that the parameter *n* should satisfy

$$n > 2^{150} \cup 10^{45}.$$

Discussion

It should be pointed out that for the software and hardware attacks described above, computation of a single elliptic curve discrete logarithm has the effect of revealing a *single* user's private key. The same effort must be repeated in order to determine another user's private key.

In [BDR 96], Blaze et al. report on the minimum key lengths required for secure symmetric-key encryption schemes (such as DES and IDEA). Their report comes to the following conclusion:

> *To provide adequate protection against the most serious threats—well-funded commercial enterprises or government intelligence agencies—keys used to protect data today should be at least 75 bits long. To protect information adequately for the next 20 years in the face of expected advances in computing power, keys in newly deployed systems should be at 90 bits long.*

Extrapolating these conclusions to the case of elliptic curves, we see that *n* should be at least 150 bits for short-term security, and at least 180 bits for medium-term security. This extrapolation is justified by the following considerations:

1. Exhaustive search through a *k*-bit symmetric-key cipher takes about the same time as the Pollard-ρ algorithm applied to an elliptic curve having a 2*k*-bit parameter *n*.
2. Both exhaustive search with a symmetric-key cipher and the Pollard-ρ algorithm can be parallelized with a linear speedup.
3. A basic operation with elliptic curves (addition of two points) is computationally more expensive than a basic operation in a symmetric-key cipher (encryption of one block).
4. In both symmetric-key ciphers and elliptic curve systems, a "break" has the same effect: it recovers a single private key.

6.9.8. Required Number-Theoretic Algorithms [normative]

6.9.8.1. The MOV and Frey-Rueck Condition

The reduction attack of Menezes, Okamoto, and Vanstone [MOV 93] reduces the discrete logarithm problem in an elliptic curve (ECDLP) over F_q to the discrete logarithm in the finite field F_{q^B} for some $B \geq 1$. Other work has treated problems that are closely related to ECDLP. Frey and Rueck [P1363; NR 93] used a variant of the Tate pairing for abelian varieties over local fields to obtain an attack, which works on projective irreducible nonsingular curves of genus *g* over finite fields.

The attack is only practical if *B* is small; this is not the case for most elliptic curves. The *MOV and Frey-Rueck condition* ensures that an elliptic curve is not vulnerable to this reduction attack .

Before performing the algorithm, it is necessary to select an MOV and Frey-Rueck threshold. This is a positive integer *B* such that taking discrete logarithms over F_{q^B} is at least as difficult as taking elliptic discrete logarithms over F_q. For this specification, a value $B \geq 20$ is required. This algorithm is used in elliptic curve parameter validation (Sections 6.9.2.4.1.2 and 6.9.2.4.1.4.) and elliptic curve parameter generation (Section 6.9.8.3.2.).

Input: An MOV and Frey-Rueck threshold *B*, a prime-power *q*, and a prime *n*.

Output: The message "True" if the MOV and Frey-Rueck condition is satisfied for an elliptic curve over F_q with a base point of order *n*; the message "False" otherwise.

1. Set $t = 1$.
2. For *i* from 1 to *B* do
   2.1 Set $t = t \cdot q \bmod n$.
   2.2 If $t = 1$ then output "False" and stop.
3. Output "True."

6.9.8.2. Primality

6.9.8.2.1. A Probabilistic Primality Test

If *n* is a large positive integer, the following probabilistic algorithm (the *Miller-Rabin test*) [P1363; Knu 81, p.379] will determine whether *n* is prime or composite, with arbitrary small probability of error.
This algorithm is used in elliptic curve parameter validation (Sections 6.9.2.4.1.2 and 6.9.2.4.1.4), elliptic curve parameter generation (Section 6.9.8.3.2.), and in checking for near primality (Section 6.9.8.2.2.).

Input: A large odd integer *n*, a positive integer *T*.
Output: The message "prime" or "composite."

1. Compute *v* and odd *w* such that $n - 1 = 2^v w$.

2. For j from 1 to T do
   2.1 Choose random a in the interval [2, n - 1].
   2.2 Set $b = a^w \bmod n$.
   2.3 If b = 1 or n - 1 go to Step 2.6.
   2.4 For i from 1 to v - 1 do
      2.4.1 Set $b = b^2 \bmod n$.
      2.4.2 If $b = n$ - 1 go to Step 2.6.
      2.4.3 If b = 1 output "composite" and stop.
      2.4.4 Next i.
   2.5. Output "composite" and stop.
   2.6. Next j.
3. Output "prime."

If the algorithm outputs "composite," then n is composite. If the algorithm outputs "prime," then n is *probably* prime. The probability of a false "prime" conclusion is less than 2^{-2T}. Thus the probability of error can be made negligible by taking large enough T (say, $T = 50$).

6.9.8.2.2. Checking for Near Primality

Given a trial division bound ℓ_{max}, we say that the positive integer k is *smooth* if every prime divisor of k is at most ℓ_{max}. Given a large positive integer r_{min}, we say that u is *nearly prime* if $u = kn$ for some prime $n \geq r_{min}$ and some smooth integer k. The following algorithm checks for near primality. The algorithm is used in elliptic curve parameter generation (Section 6.9.8.3.2.).

Input: Positive integers u, ℓ_{max}, and r_{min}.

Output: If u is nearly prime, a prime $r \geq r_{min}$ and a smooth integer k such that: $u = kn$.

If u is not nearly prime, the message "not nearly prime."

1. Set $n = u$, $k = 1$.
2. For ℓ from 2 to ℓ_{max} do
   2.1 If ℓ is composite then go to Step 2.3.
   2.2 While (ℓ divides n)
      2.2.1 Set $n = n / \ell$ and $k = k \cdot \ell$.
      2.2.2 If $n < r_{min}$ then output "not nearly prime" and stop.
   2.3 Next ℓ.
3. If n is prime (see Section 6.9.8.2.1) then output k and n and stop.
4. Output "not nearly prime."

6.9.8.3. Elliptic Curve Algorithms

6.9.8.3.1. Finding a Point of Large Prime Order

If the order $\#E(F_q) = u$ of an elliptic curve E is nearly prime, the following algorithm efficiently produces a random point on E whose order is the large prime factor n of $u = kn$. The algorithm is used in elliptic curve parameter generation (Section 6.9.8.3.2.).

Input: A prime n, a positive integer k not divisible by n, and an elliptic curve E over the field F_q with $\#E(F_q) = u$.

Output: If $u = kn$, a point P on E of order n. If not, the message "wrong order."

1. Generate a random point G (not 0) on E (see Section 6.9.9.2.1).

2. Set $P = kG$.
3. If $P = 0$ then go to Step 1.
4. Set $Q = nP$.
5. If $Q \neq 0$ then output "wrong order" and stop.
6. Output G.

6.9.8.3.2. Selecting an Appropriate Curve and Point at Random

Given a field size q, a lower bound r_{min} for the point order, and trial division bound ℓ_{max}, the following is the procedure that shall be used for randomly choosing a curve and point.

1. If $q = p$, then check the primality of p using the technique defined in Section 6.9.8.2.1.
   If $q = 2^m$ and an optimal normal basis is desired, then check that such a basis exists by consulting Table 5.1 of [IXR 93, p. 100].
2. Randomly select parameters $a, b \in F_q$ to define the elliptic curve equation.
   - In the case that q is a prime, verify that $4a^3 + 27b^2 \neq 0 \pmod p$.
     - The curve equation is $E: y^2 = x^3 + ax + b$.
     - In the case that $q = 2^m$, verify that $b \neq 0$. The curve equation is $E: y^2 + xy = x^3 + ax^2 + b$.
3. Compute the order of the curve $u = \#E(F_q)$. (See Note 5 below.)
4. Test u for near primality using the technique defined in Section 6.9.8.2.2. If the result is "not nearly prime," then go to Step 2. Otherwise, $u=kn$ where k is smooth and n is prime.
5. Check the MOV condition (see Section 6.9.8.1) with inputs B, q, and n. If the result is "False," then go to Step 2.
6. Find a point P on E of order n using the technique defined in Section 6.9.8.3.1.
7. Output the curve E and the point P.

Notes:
1. r_{min} shall be selected so that $r_{min} > 2^{150}$. The security level can be increased by selecting a larger r_{min} (e.g., $r_{min} > 2^{160}$ or $r_{min} > 2^{200}$).
2. If q is prime then the order u of an elliptic curve E over F_q satisfies $q + 1 - 2\sqrt{q} \leq u \leq q + 1 + 2\sqrt{q}$. Hence r_{min} must be $\leq q+1-2\sqrt{q}$.
3. If $q = 2^m$ then the order u of an elliptic curve E over F_q satisfies $q + 1 - 2\sqrt{q} \leq u \leq q + 1 + 2\sqrt{q}$, and u is even. Hence r_{min} must be $\leq (q+1-2\sqrt{q})/2$.
4. ℓ_{max} is typically a small integer (e.g., $\ell_{max} = 255$).
5. The order $\#E(F_q)$ can be computed by using Schoof's algorithm [P1363; Sco 85]. Although the basic algorithm is quite inefficient, several dramatic improvements and extensions of this method have been discovered in recent years. Currently it is feasible to compute orders of elliptic curves over F_p where p is as large as 10^{499}, and orders of elliptic curves over F_{2^m} where m is as large as 1300. Of more relevance is that the orders of elliptic curves over fields such as $F_{2^{155}}$ can now be computed in less than 4 minutes on a workstation [P1363; LM 95].

6.9.9. Other Number-theoretic algorithms [informative]

6.9.9.1. Polynomials Over a Finite Field

6.9.9.1.1. GCDs over a Finite Field

If $f(t)$ and $g(t) \neq 0$ are two polynomials with coefficients in the field F_q, then there is a unique monic polynomial $d(t)$ of largest degree that divides both $f(t)$ and $g(t)$. The polynomial $d(t)$ is called the *greatest common divisor* or *gcd* of $f(t)$ and $g(t)$. The following algorithm computes the gcd of two polynomials.

Input: A finite field F_q and two polynomials $f(t)$, $g(t) \neq 0$ over F_q.
Output: $d(t) = \gcd(f(t), g(t))$.

1. Set $a(t) = f(t)$, $b(t) = g(t)$.
2. While $b(t) \neq 0$
   - 2.1. Set $c(t)$ = the remainder when $a(t)$ is divided by $b(t)$.
   - 2.2. Set $a(t) = b(t)$.
   - 2.3. Set $b(t) = c(t)$.
3. Let α be the leading coefficient of $a(t)$ and output $\alpha^{-1}a(t)$.

6.9.9.1.2. Finding a Root of an Irreducible Polynomial in F_{2^m}

If $f(t)$ is an irreducible polynomial (mod 2) of degree m, then $f(t)$ has m distinct roots in the field F_{2^m}. A random root can be found efficiently using the following algorithm.

Input: An irreducible polynomial of degree m over F_2, and a field F_{2^m}.

Output: A random root of $f(t)$ in F_{2^m}.

1. Set $g(t) = f(t)$.
2. While $\deg(g) > 1$
   - 2.1. Choose random $u \in F_{2^m}$.
   - 2.2. Set $c(t) = ut$.
   - 2.3. For i from 1 to m - 1 do
     - 2.3.1. $c(t) = (c(t)^2 + ut) \bmod g(t)$.
   - 2.4. Set $h(t) = \gcd(c(t), g(t))$.
   - 2.5. If $h(t)$ is constant or $\deg(g) = \deg(h)$ then go to step 2.1.
   - 2.6. If $2\deg(h) > \deg(g)$ then set $g(t) = g(t) / h(t)$; else $g(t) = h(t)$.
3. Output $g(0)$.

6.9.9.1.3. Change of Basis

Given a field F_{2^m} and two bases B_1 and B_2 for the field, the following algorithm allows conversion between bases B_1 and B_2.

1. Let $f(t)$ be the (irreducible) reduction polynomial of B_2.
2. Let γ be a root of $f(t)$ computed with respect to B_1. (γ can be computed using the technique defined in Section 6.9.9.1.2.)
3. Let Γ be the matrix:

$$\Gamma = \begin{matrix} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,m-1} \\ \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{m-1,0} & \gamma_{m-1,1} & \cdots & \gamma_{m-1,m-1} \end{matrix},$$

   where the entries $\gamma_{i,j}$ are defined as follows:

- If B_2 is a polynomial basis then:

$$1 = (\gamma_{0,0}\ \gamma_{0,1}\ \ldots\ \gamma_{0,m-1}) = (1,1,\ldots,1),$$
$$\gamma = (\gamma_{1,0}\ \gamma_{1,1}\ \ldots\ \gamma_{1,m-1}),$$
$$\gamma^2 = (\gamma_{2,0}\ \gamma_{2,1}\ \ldots\ \gamma_{2,m-1}),$$
$$\vdots$$
$$\gamma^{m-1} = (\gamma_{m-1,0}\ \gamma_{m-1,1}\ \ldots\ \gamma_{m-1,m-1}),$$

  with respect to B_1, where $\gamma_{1,i}$, $i = 0,\ldots, m\text{-}1$, are obtained from 6.9.9.1.2, and the entries $\gamma_{i,j}$, $j=0,1,\ldots,m\text{-}1$, are computed by multiplication of the previous line by γ and reduction by the irreducible polynomial of B_1, for $i = 2,\ldots,m\text{-}1$.

- If B_2 is an optimal normal basis, then:

$$\gamma = (\gamma_{0,0}\ \gamma_{0,1}\ \ldots\ \gamma_{0,m-1})$$
$$\gamma^2 = (\gamma_{1,0}\ \gamma_{1,1}\ \ldots\ \gamma_{1,m-1})$$
$$\gamma^4 = (\gamma_{2,0}\ \gamma_{2,1}\ \ldots\ \gamma_{2,m-1})$$
$$\vdots$$
$$\gamma^{2^{m-1}} = (\gamma_{m-1,0}\ \gamma_{m-1,1}\ \ldots\ \gamma_{m-1,m-1})$$

  with respect to B_1, where $\gamma_{1,i}$, $i = 0,\ldots, m\text{-}1$, are obtained from 6.9.9.1.2, and the entries $\gamma_{i,j}$ $j=0,1,\ldots,m\text{-}1$, are computed by squaring of the previous line and reduction by the irreducible polynomial of B_1, for $i = 1,\ldots,m\text{-}1$.

4. If an element has representation $(\beta_0\ \beta_1\ \ldots\ \beta_{m-1})$ with respect to B_2, then its representation with respect to B_1 is

$$(\alpha_0\ \alpha_1\ \ldots\ \alpha_{m-1}) = (\beta_0\ \beta_1\ \ldots\ \beta_{m-1})\ \Gamma.$$

   If an element has representation $(\alpha_0\ \alpha_1\ \ldots\ \alpha_{m-1})$ with respect to B_1, then its representation with respect to B_2 is

$$(\beta_0\ \beta_1\ \ldots\ \beta_{m-1}) = (\alpha_0\ \alpha_1\ \ldots\ \alpha_{m-1})\ \Gamma^{-1}.$$

6.9.9.2. Elliptic Curve Algorithms

6.9.9.2.1. Finding a Point on an Elliptic Curve

The following algorithms provide an efficient method for finding a statistically unbiased and unpredictable point (other than 0) on a given elliptic curve over a finite field.

Case I: Curves over F_p

Input: A prime p and the parameters a, b of an elliptic curve E over F_p.
Output: A randomly generated point (other than 0) on E.

1. Choose random integer x with $0 \le x < p$.
2. Set $\alpha \leftarrow x^3 + ax + b \bmod p$.
3. If $\alpha = 0$ then output $(x, 0)$ and stop.
4. If $\alpha^{(p-1)/2} \equiv -1 \pmod p$ then go to Step 1.
5. Find an integer y such that $y^2 \equiv \alpha \pmod p$.
6. Output (x, y).

Case* II*: Curves over F_{2^m}**.**

Input: A field F_{2^m} and the parameters a, b of an elliptic curve E over F_{2^m}.
Output: A randomly generated point (other than 0) on E.

1. Choose random x in F_{2^m}.
2. If $x = 0$ then output $(0, b^{2^{m-1}})$ and stop.
3. Set $\alpha = x^3 + ax^2 + b$.

4. If $\alpha = 0$ then output $(x, 0)$ and stop.
5. Set $\beta = x^{-2} \alpha$.
6. If $Tr(\beta) = 1$ then go to Step 1 (Tr is the *trace* function).
7. Find an element z such that $z^2 + z = \beta$, see [P1363, A.4.7].
8. Set $y = xz$.
9. Output (x, y).

6.9.9.2.2. Point Decompression

Point decompression (prime case)

Suppose that we are given the x-coordinate x_P of P and the bit $\tilde{y}_P$. Then y_P can be recovered as follows.

1. Compute the field element $\alpha = x_P^3 + ax_P + b \bmod p$.
2. Compute a square root β of $\alpha \bmod p$, see [P1363, A2.5].
3. If the rightmost bit of β is equal to $\tilde{y}_P$ then set $y_P = \beta$. Otherwise, set $y_P = p - \beta$.

Point decompression (characteristic 2 case)

Suppose that we are given $\tilde{x}$, which is the x-coordinate x_P of P and the bit $\tilde{y}$. Then the coordinates of the point P can be recovered as follows.

1. Set $x := \tilde{x}$.
2. If $x=0$ then let y be a square root of b (to compute a square root of b square b m-1 times) and go to Step 8.
3. Compute the field element $\beta = x_P + a + bx_P^{-2}$ in F_{2^m}., see [P1363, A.4.4].
4. Find a field element z such that $z^2 + z = \beta$, see [P1363, A.4.7]. If the output is "no solution exists," then return an error message and stop.
5. Let $\tilde{z}$ be the right most bit of z.
6. If $\tilde{y} = \tilde{z}$ then $z = z + 1$.
7. Compute $y = z\,x$.
8. Output (x,y).

7. Informative Appendix

7.1. Determination of Resynchronization Rate

Choice of Resynchronization rate is dependent on several factors including actual traffic rates and cell loss rates. The following information is presented as one possible method for choosing a resynchronization rate.

7.1.1. Expected Cell Loss Rates

Network cell loss statistics can be used to determine the appropriate resynchronization rate. Typical error rate requirements for ATM are specified by the Cell Loss Ratio (CLR), which is defined as: lost cells / transmitted cells. CLR requirements differ by type of traffic and Quality of Service (QoS). A typical worst case CLR requirement for an ATM network is 10^{-4} for low-priority Packet Video/Audio traffic when the Cell Loss Priority (CLP) bit in the cell header is set to one. This means that a single 622 Mbps connection carrying this type of traffic with a cell loss ratio of 10^{-4} will lose sync 142 times per second on average, assuming no cell loss insensitivity. This is a worst case example. More typical error ratios, e.g., 10^{-7} result in 0.14 lost cells per second on a full bandwidth 622 Mbps connection.

To conserve the bandwidth required for resynchronization, it is necessary to resynchronize based on the bandwidth of the connection and the negotiated QoS. Higher bandwidth connections are resynchronized more often than those of lower bandwidths are. Connections with lower CLR may resync at a lower rate than high CLR connections.

7.1.2. Determining CLR

The encryptor can use the expected CLR of the connection to adjust its frequency of transmitting resync messages. The Quality of Service is specified by the host to the switch in the QoS parameter of the SETUP message. The encryptor can intercept the SETUP message and use the QoS parameter to determine the CLR, based on a user configurable mapping function. The CLR may also be available as an optional QoS parameter in UNI Signaling 4.0 [3] messages.

7.1.3. Resynchronization Rate

The resynchronization interval can be chosen to be one tenth the cell loss rate (CLR). For example, a connection that loses 100 cells per second on average will have a resync rate of 1000 resyncs per second. Making the resync rate ten times the cell loss rate will ensure that on average only 5 percent of the traffic is lost due to loss of cryptographic synchronization. The resync rate for each connection can be based on the following formula:

Resync rate (resyncs/second) = cell loss ratio (cells dropped/cells transmitted) x sustainable cell rate (cells transmitted/second) x 10.

Using the above formula, the maximum resync rate for a 622 Mbps connection will be 1417 times a second, for a CLR of 10^{-4}. This results in an added overhead of 0.01%. There may be value in establishing a minimum resync rate.

Note: there is no value in resynchronizing in the middle of an AAL5 message, since any cell loss causes the entire message to be discarded by the receiver.

7.2. Label-Based Access Control

7.2.1. Scenarios

ATM access control can occur either within an ATM end-point or at any intermediate ATM node (i.e., an ATM switch or an in-line security device). In each case, network administrators establish (either through the network management protocol or through direct configuration of the ATM devices) that an ATM physical interface requires access control parameters. During connection establishment, an ATM component for such an interface is required to communicate the necessary access control parameters, and an ATM device receiving these access control parameters would perform access control using those parameters. This exchange of access control data could occur across (1) an end-point to private switch interface, (2) a private switch to private switch interface, (3) a private switch to public switch interface, or (4) a public switch to public switch interface. Also, the initiator of the access control parameters could be either the initiator of the connection, some intermediate ATM switch, or an ATM security device. A typical application for label-based access controls is shown in Figure 50.

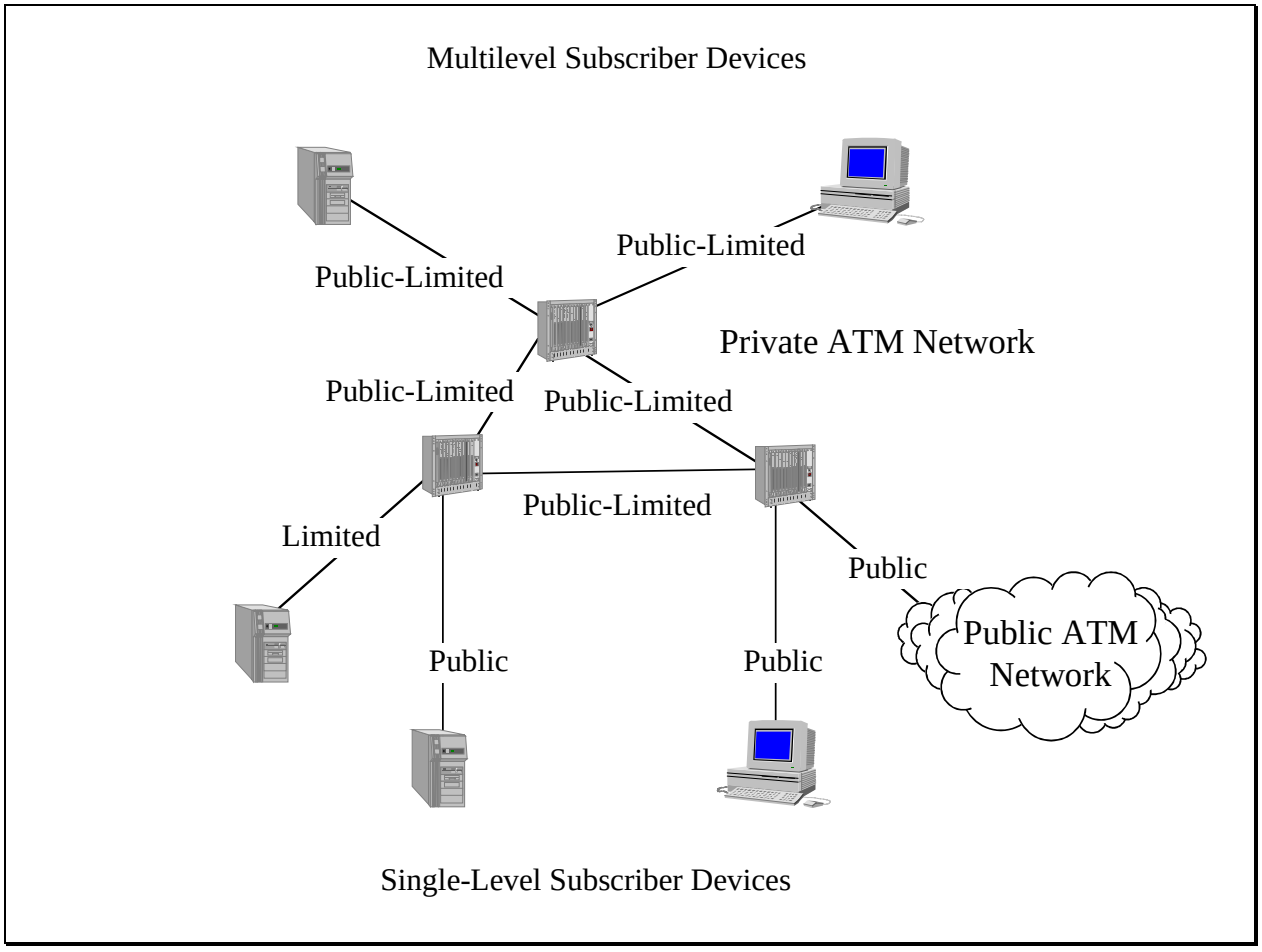


Figure 50. Typical Access Control Scenarios.

In this context, multilevel refers to a device that can interpret security labels and can maintain separation of data marked at different security levels. For this scenario, an enterprise marks its data as either “public,”

indicating that the data is releasable to the public, or “limited,” indicating that the data must stay within the enterprise. In this scenario, the multilevel devices (switches and multilevel subscriber devices) within the private ATM network ensure that only publicly releasable information is sent to public subscriber devices or through the public ATM switches. Within the private ATM network, the multilevel devices (switches and multilevel subscriber devices) ensure that all connections are labeled.

This scenario requires labeling mechanisms within the private UNI and P-NNI. Figure 51 shows a scenario using labeling mechanisms within public ATM switches, requiring labeling mechanisms within the public UNI and B-ICI.

In this second scenario, the public network provider offers a multilevel access control service to subscriber networks, enabling selected private networks to communicate. Each subscriber interface is labeled with the set of companies permitted to send data across that interface. This mechanism restricts which private networks can communicate with a specific company’s private networks. For example, neither Company A nor Company B could communicate directly with Company C’s private networks. However, companies A and C do share a common private network (perhaps a consortium, common supplier, or common customer) through which they might communicate, depending on the policy of that private network. The multilevel devices (i.e., the public switches and the switches within the multilevel subscriber networks) ensure that each connection is labeled.

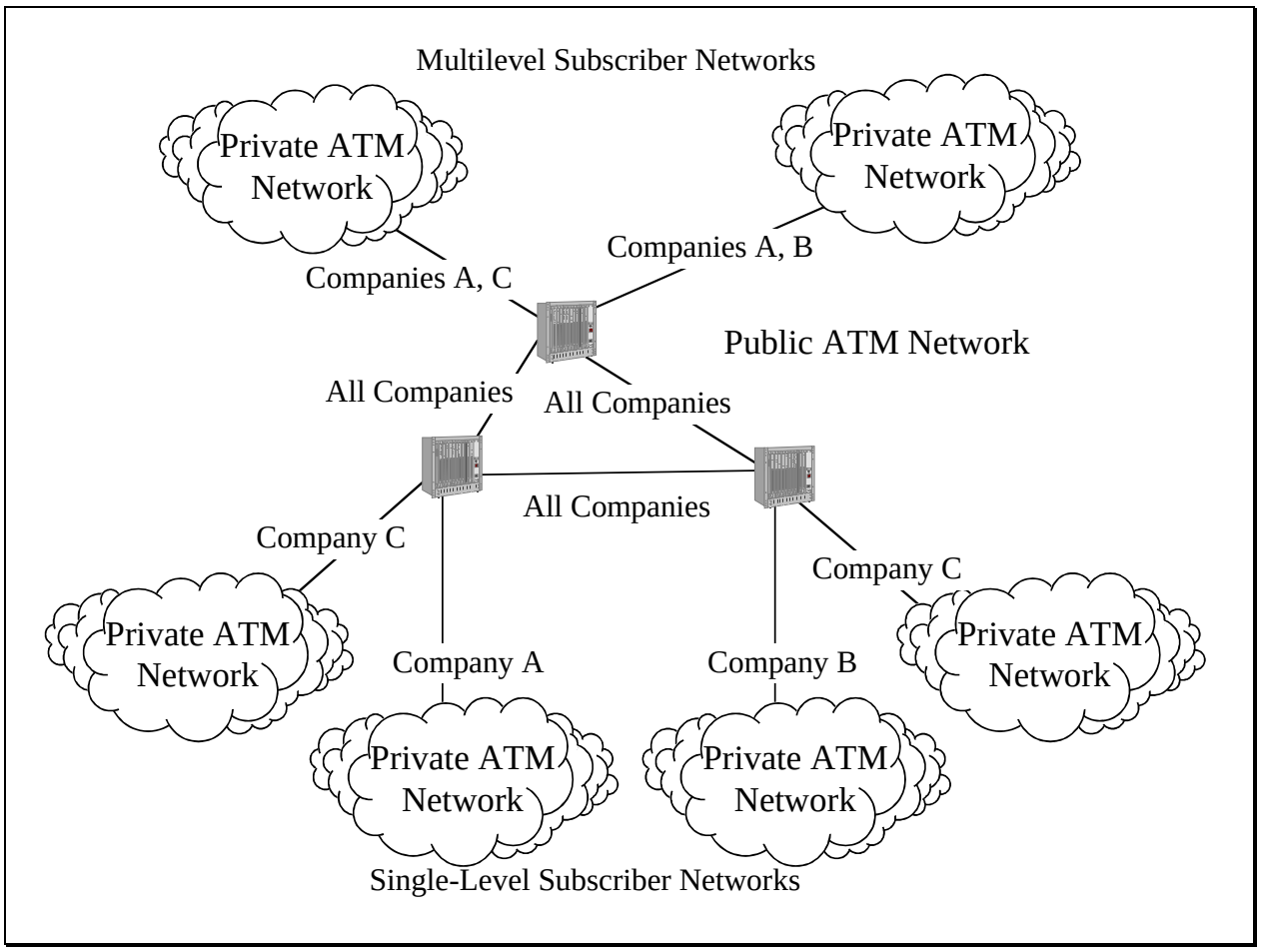


Figure 51. Access Control Scenario for Public Switches.

Access control enforcement is the same for both scenarios. On receipt of a connection request, the multilevel devices ensure that the connection request either (1) came from a single-level interface, in which case the connection is labeled at the level of the interface; or (2) came from a multilevel interface, in which case the connection must be in the set of security levels permitted for that interface. On transmission of a connection request, the multilevel devices ensure that the connection security level is transmitted during the connection setup protocol if the request is placed on an interface requiring labeling (e.g., a multilevel interface).

7.2.2. Label-Based Access Control Concepts

Typical label-based access control mechanisms are based on a concept of sensitivity levels. A sensitivity level represents the sensitivity of the data to disclosure. For example, a sensitivity level might indicate the extent to which a company's data must be protected, with different sensitivity levels (e.g., public data, proprietary, company confidential, company limited, and competition sensitive) indicating different degrees of control.

A typical sensitivity level comprises two components: (1) a hierarchical level and (2) a set of access categories. These levels are used with a partially ordered relation on the set of sensitivity levels. This relation determines whether one sensitivity level is less than, equal to, greater than, or not comparable with

another sensitivity level. The hierarchical level component of a sensitivity level is linearly ordered, with its own "less than" relation. The relation used for the access category sets is set containment. That is, one set of access categories is said to be "less than or equal to" another set of access categories if the first is a subset of the second. The first is said to be "less than" the second if the first is a proper subset of the second. Thus, sensitivity level SL1=(HL1, SC1) is less than or equal to sensitivity level SL2=(HL2, SC2) provided HL1 is less than or equal to HL2 and SC1 is a subset of SC2, where HL1 and HL2 are the hierarchical level components and SC1 and SC2 are the access category set components of SL1 and SL2, respectively.

Sensitivity labels are the physical representation of a sensitivity level. There may be multiple representations of the same sensitivity level. For a given sensitivity level, there are typically multiple representations (or sensitivity labels), including the human readable label (e.g., company limited/finance) and the internal representation of that label within a computing system.

7.2.3. Label-Based Access Control Rules

Label-based access control mechanisms within a network component require that each network object (i.e., packet or connection) have an associated sensitivity level and typically assign maximum and minimum sensitivity levels to each external physical interface to enable enforcement of the following rules.

1. The system may transmit a network object across a physical interface provided the object's sensitivity level is less than or equal to the interface's maximum sensitivity level.

2. The system may accept a network object across a physical interface for delivery provided the object's sensitivity level is less than or equal to the interface's maximum sensitivity level *and* greater than or equal to the interface's minimum sensitivity level.

Note that the object sensitivity level may be determined either by a label attached to the object or implicitly by the sensitivity level of the physical interface in the case when the maximum and minimum sensitivity levels of the physical interface are equal.

In the context of performing label-based access control only on connection setup, the network object is the connection, and the label would be carried in the setup message.

7.3. Security Services Information Element Examples

7.3.1. Initiate In-Band Security Message Exchange

7.3.1.1. Simple In-Band Exchange Indication

This example declares that a security service will be required on this developing VC, and that In-Band Security Message Exchange will be used between the two security agents to establish the service. The initiating security agent places this SSIE in the call SETUP message:

Bits								
8	7	6	5	4	3	2	1	Octet(s)
Security Services Information Element								
x	x	x	x	x	x	x	x	1
Information element identifier								
1 Ext	Coding Standard		Information Element Instruction Field					2
			Flag	Reserved	Information Element Action Indicator			
0x0008 (8)								3 - 4
Length of Security Services Information Element								
0	0	0	0	0	0	0	1	
Security Association Service Identifier (SME)								5
0x0005 (5)								
Security Association Section Length								6 - 7
0	0	0	1	1	0	0	1	
Version			Transport Ind.		Flow Indicator		Discard	8
0	0	1	0	0	0	0	0	
Region (Remote)								9
1	1	1	0	0	0	0	0	
Role (Any)								10
0	0	0	0	0	0	0	1	
Relative ID (Security Association ID)								11
0	0	0	1	0	0	0	0	
Relative ID (SAS Number)				reserved				12

The responding security agent places this SSIE in the call CONNECT message:

Bits								
8	7	6	5	4	3	2	1	Octet(s)
Security Services Information Element								1
x	x	x	x	x	x	x	x	
Information element identifier								
1 Ext	Coding Standard		Information Element Instruction Field					2
			Flag	Reserved	Information Element Action Indicator			
0x0008 (8)								3 - 4
Length of Security Services Information Element								
0	0	0	0	0	0	0	0	5
Security Association Service Identifier								
0x0005 (5)								6 - 7
Security Association Section Length								
0	0	0	1	1	0	1	1	8
Version			Transport Ind.		Flow Indicator		Discard	
1	1	1	1	1	1	1	1	9
Region (Remote)								
1	1	1	1	1	1	1	1	10
Role (Any)								
0	0	0	0	0	0	0	1	11
Relative ID (Security Association ID)								
0	0	0	1	0	0	0	0	12
Relative ID (SAS Number)				reserved				

7.3.1.2. With Requested Service Declaration

This example initiates an In-Band Security Message Exchange between two security agents. The initiating security agent declares its intent to authenticate and support data integrity in an In-Band exchange, providing the responding security agent the ability to deny the call if authentication and integrity are not supported. The initiating security agent places this SSIE in the call SETUP message:

Bits								
8	7	6	5	4	3	2	1	Octet(s)
Security Services Information Element								1
x	x	x	x	x	x	x	x	
Information element identifier								
1 Ext	Coding Standard		Information Element Instruction Field					2
			Flag	Reserved	Information Element Action Indicator			
0x000B (11)								3 - 4
Length of Security Services Information Element								
0	0	0	0	0	0	0	1	5
Security Association Service Identifier (SME)								
0x0008 (8)								6 - 7
Security Association Section Length								
0	0	0	1	1	0	0	1	8
Version			Transport Ind.		Flow Indicator		Discard	
0	0	1	0	0	0	0	0	9
Region (Remote)								
1	1	1	0	0	0	0	0	10
Role (Any)								
0	0	0	0	0	0	0	1	11
Relative ID (Security Association ID)								
0	0	0	1	0	0	0	0	12
Relative ID (SAS Number)				reserved				
1	0	0	0	1	0	0	0	13
Security Service Specification Section Identifier								
1	0	0	0	1	0	1	0	14
Security Service Declaration Identifier								
0	0	0	0	0	1	1	0	15
Security Service Declaration (authentication and integrity)								

In this example, the responding security agent accepts the call, despite the fact that it cannot support data integrity. In the CONNECT message, the responding security agent must also make a security service declaration, indicating that it can support establishing authentication in an In-Band exchange. It is then up to the initiating security agent to decide if the VC and the subsequent In-Band negotiation should proceed. If the initiating security agent cannot continue due to its local security policy, i.e., the VC must support cryptographic integrity, then the initiating security agent will instruct its signaling entity to clear the call with the appropriate cause code and diagnostic, “Security Policy Violation” (see Section 5.1.6). The responding security agent places this SSIE in the call CONNECT message:

Bits								
8	7	6	5	4	3	2	1	Octet(s)
Security Services Information Element								
x	x	x	x	x	x	x	x	1
Information element identifier								
1 Ext	Coding Standard		Information Element Instruction Field					2
			Flag	Reserved	Information Element Action Indicator			
0x000B (11)								
Length of Security Services Information Element								3 - 4
0	0	0	0	0	0	0	1	
Security Association Service Identifier (SME)								5
0x0008 (8)								
Security Association Section Length								6 - 7
0	0	0	1	1	0	1	1	
Version			Transport Ind.		Flow Indicator		Discard	8
1	1	1	1	1	1	1	1	
Region (Remote)								9
1	1	1	0	0	0	0	0	
Role (Any)								10
0	0	0	0	0	0	0	1	
Relative ID (Security Association ID)								11
0	0	0	1	0	0	0	0	
Relative ID (SAS Number)				reserved				12
1	0	0	0	1	0	0	0	
Security Service Specification Section Identifier								13
1	0	0	0	1	0	1	0	
Security Service Declaration Identifier								14
0	0	0	0	0	1	0	0	
Security Service Declaration (authentication)								15

7.3.2. Two Way Security Message Exchange

This example provides endpoint-to-endpoint authentication using the 2-way Security Message Exchange protocol in signaling. Because in 2-way SME, the distinguished name of the responding security agent (B) is known, explicit security agent addressing is used in the SSIE. Since (B) is included in the SSIE SAS explicit scope field, it is not needed in the SME section of the SAS. Initiator (A) declares that it requires authentication and provides the optional fields needed to provide replay protection, T_a and R_a.

FLOW1-2WE: *A –> B*

A, B, SecOpt, {T_a*,* R_a*, {*Enc_{Kb} *(*$ConfPar_a$*)}, *Sig_{Ka} *(Hash (A, B,* T_a*,* R_a*, SecOpt, {*$ConfPar_a$*}))}*

Bits								
8	7	6	5	4	3	2	1	Octet(s)
Security Services Information Element								
x	x	x	x	x	x	x	x	1
Information element identifier								
1 Ext	Coding Standard		Information Element Instruction Field					2
			Flag	Reserved	Information Element Action Indicator			
0x008A (138)								
Length of Security Services Information Element								3 - 4
0	0	0	0	0	0	0	1	
Security Association Service Identifier (SME)								5
0x87 (135)								
Security Association Section Length								6 - 7
0	0	0	0	0	0	0	1	
Version			Transport Ind.		Flow Indicator		Discard	8
1	0	0	0	0	0	0	0	
Scope (Explicit)								9
0	0	0	0	0	0	0	0	
Role (Explicit)								10
0	0	0	0	0	0	0	1	
Relative ID (Security Association ID)								11
0	0	0	1	0	0	0	0	
Relative ID (SAS Number)				reserved				12
1	0	0	0	0	1	0	0	
Security Agent Identifier *(B)*								13
0	0	0	1	0	1	0	0	
Security Agent Identifier Length (0x14) (20)								14
0	0	0	0	0	1	0	0	
Security Agent Identifier Type (AESA NSAP ICD)								15
0x4302340515231912110098096245F2A8B2127400								
Security Agent Identifier Value								16 - 35
0	0	1	0	0	1	1	0	
Security Message Exchange Format (Two Way)								36
1	0	0	0	0	0	1	0	
Initiator Distinguished Name Identifier (*A*)								37
0	0	0	1	0	1	0	0	
Initiator Distinguished Name Length (0x14)								38
0	0	0	0	0	1	0	0	
Initiator Distinguished Name Type (AESA NSAP ICD)								39
0x4302340515231912110098096218430DE6372812								
Initiator Distinguished Name Value								40 - 59

1	0	0	0	1	0	0	0	
Security Service Specification Section (*SecOpt*)								60
1	0	0	1	0	0	1	1	
Authentication Service Options								61
1	0	0	0	0	0	0	1	
Requires Authentication								62
1	1	0	1	0	1	1	0	
Time-Variant Time Stamp								63
0x67456721 0x00000001								
Time-Variant Time Stamp Value								64 - 71
1	1	0	1	0	1	0	0	
Initiator Random Number *(R_Q)*								72
0xB564A3E1								
Initiator Random Number Value								73 - 76
1	1	0	1	1	0	1	0	
Security Message Exchange Digital Signature Identifier								77
0	1	0	0	0	0	0	0	
Security Message Exchange Digital Signature Length (64)								78
Digital Signature Value								79 - 142.

FLOW2-2WE: *B–>A*

A, B, R_a, {$EncK_a$ (ConfPar$_b$)}, $SigK_b$ (Hash (A, B, R_a, {ConfPar$_b$}))}

Bits								
8	7	6	5	4	3	2	1	Octet(s)
Security Services Information Element								
x	x	x	x	x	x	x	x	1
Information element identifier								
1 Ext	Coding Standard		Information Element Instruction Field					2
			Flag	Reserved	Information Element Action Indicator			
0x007E (126)								
Length of Security Services Information Element								3 - 4
0	0	0	0	0	0	0	1	
Security Association Service Identifier (SME)								5
0x007B (123)								
Security Association Section Length								6 - 7
0	0	0	0	0	0	1	1	
Version			Transport Ind.		Flow Indicator		Discard	8
1	1	1	1	1	1	1	1	
Region (remote)								9
1	1	1	0	0	0	0	0	
Role (any)								10
0	0	0	0	0	0	0	1	
Relative ID (Security Association ID)								11
0	0	0	1	0	0	0	0	
Relative ID (SAS Number)				reserved				12
0	0	1	0	0	1	1	0	
Security Message Exchange Format (Two Way)								13
1	0	0	0	0	0	1	0	
Initiator Distinguished Name Identifier (*A)*								14
0	0	0	1	0	1	0	0	
Initiator Distinguished Name Length								15
0	0	0	0	0	1	0	0	
Initiator Distinguished Name Type (AESA NSAP ICD)								16
0x43023405152319121100980962l8430DE6372812								
Initiator Distinguished Name Value								17 – 36
1	0	0	0	0	0	1	1	
Responder Distinguished Name Identifier (*B)*								37
0	0	0	1	0	1	0	0	
Responder Distinguished Name Length								38
0	0	0	0	0	1	0	0	
Responder Distinguished Name Type (AESA NSAP ICD)								39
0x43023405152319121100980962l5F2A8B2127400								
Initiator Distinguished Name Value								40 – 59

1	1	0	1	0	1	0	0	
Initiator Random Number Identifier								60
0xB564A3E1								
Initiator Random Number Value								61 - 64
1	1	0	1	1	0	1	0	
Security Message Exchange Digital Signature Identifier								65
0	1	0	0	0	0	0	0	
Security Message Exchange Digital Signature Length (64)								66
Digital Signature Value								67 - 130.

7.3.3. Three Way Security Message Exchange

This example establishes endpoint-to-endpoint confidentiality, authentication, key exchange, and session key update using the 3-way Security Message Exchange protocol , in-band.

Initiator (A) requests confidentiality, authentication, key exchange, and session key update. To do this A declares the five security services it requires and then specifies two choices for the confidentiality algorithm, a signature algorithm, a hash algorithm, a key exchange algorithm, and a session key update algorithm. The confidentiality algorithms are DES/CBC and FEAL/CBC, the signature algorithm is FEAL/CBC, the hash algorithm is HMAC-MD5, the key exchange algorithm is FEAL/CBC, and session key update algorithm uses HMAC-MD5.

FLOW1-3WE: *A–>B*

$$A, \{B\}, SecNeg_a, R_a, \{Cert_a\}$$

Bits								
8	7	6	5	4	3	2	1	Octet(s)
0	0	1	0	0	0	0	1	
In-Band Message Type								1
0x0053 (99)								
In-Band Message Length								2 - 3
Security Services Information Element								
x	x	x	x	x	x	x	x	4
Information element identifier								
1 Ext	Coding Standard		Information Element Instruction Field					
			Flag	Reserved	Information Element Action Indicator			5
0x0050 (96)								
Length of Security Services Information Element								6 - 7
0	0	0	0	0	0	0	1	
Security Association Service Identifier (SME)								8
0x4D (93)								
Security Association Section Length								9 - 10
0	0	0	1	1	0	0	1	
Version			Transport Ind.		Flow Indicator		Discard	11
0	0	1	0	0	0	0	0	
Region (remote)								12
1	1	1	0	0	0	0	0	
Role (any)								13
0	0	0	0	0	0	0	1	
Relative ID (Security Association ID)								14
0	0	0	1	0	0	0	0	
Relative ID (SAS Number)			reserved					15
1	0	0	0	0	1	0	0	

Security Agent Identifier *{B}*								16
0	0	0	1	0	1	0	0	
Security Agent Identifier Length (0x14)								17
0	0	0	0	0	1	0	0	
Security Agent Identifier Type (AESA NSAP ICD)								18
0x43023405152319121100980962 45F2A8B2127400								
Security Agent Identifier Value								19 - 38
0	0	1	0	0	1	1	1	
Security Message Exchange Format (Three Way)								39
1	0	0	0	0	0	1	0	
Initiator Distinguished Name Identifier (*A*)								40
0	0	0	1	0	1	0	0	
Initiator Distinguished Name Length (0x14)								41
0	0	0	0	0	1	0	0	
Initiator Distinguished Name Type (AESA NSAP ICD)								42
0x43023405152319121100980962 18430DE6372812								
Initiator Distinguished Name Value								43 - 62
1	0	0	0	1	0	0	0	
Security Service Specification Section (*$SecNeg_A$*)								63
1	0	0	0	1	0	1	0	
Security Service Declaration Identifier								64
0	0	1	0	1	1	1	1	
Security Service Declaration								65
1	0	1	0	0	0	0	0	
Data Confidentiality Algorithm Identifier								66
0	0	0	1	0	0	1	0	
Length of Data Confidentiality Algorithm Contents (18)								67
0	0	0	0	0	0	1	1	
Data Confidentiality Algorithm (Triple DES 112 key)								68
0	0	0	0	0	0	1	0	
Data Confidentiality Algorithm Mode of Operation (Counter Mode)								69
1	0	1	0	0	0	0	0	
Data Confidentiality Algorithm Identifier								70
0	0	0	0	1	1	0	0	
Length of Data Confidentiality Algorithm Contents (12)								71
0	0	0	0	0	1	0	0	
Data Confidentiality Algorithm (FEAL)								72
0	0	0	0	0	0	0	1	
Data Confidentiality Algorithm Mode of Operation (CBC)								73
0x238609F0E71A9D17								
Data Confidentiality Algorithm Details Initialization Vector (FEAL)								74 - 81
1	0	1	0	0	1	1	0	
Signature Algorithm Identifier								82
0	0	0	0	0	0	0	1	
Length of Signature Algorithm Identifier								83
0	0	0	0	1	0	0	0	
Signature Algorithm (FEAL/CBC)								84

1	0	1	0	0	1	0	0	
Hash Algorithm Identifier								85
0	0	0	0	0	0	0	1	
Length of Hash Algorithm Contents (1)								86
1	0	1	0	0	0	0	1	
Hash Algorithm (HMAC-MD5)								87
1	0	1	0	1	0	0	0	
Key Exchange Algorithm Identifier								88
0	0	0	0	0	0	0	1	
Length of Key Exchange Algorithm Contents (1)								89
0	0	0	0	0	0	0	1	
Key Exchange Algorithm (FEAL/CBC)								90
1	0	1	0	1	0	1	0	
Session Key Update Algorithm Identifier								91
0	0	0	0	0	0	0	1	
Length of Session Key Update Algorithm Contents (1)								92
0	0	0	0	0	0	1	0	
Session Key Update Algorithm (SKE with HMAC-MD5)								93
0	1	0	1	0	0	0	0	
Authentication Section Identifier								94
1	1	0	1	0	1	0	0	
Initiator Random Number Identifier (R_A)								95
0x7A52D245								
Initiator Random Number Value								96 - 99

Responder (B) requires confidentiality, authentication, and key exchange. The confidentiality algorithm is FEAL/CBC, the signature algorithm is FEAL/CBC, the hash algorithm is HMAC-MD5, and the key exchange algorithm is FEAL/CBC.

FLOW2-3WE: B –> A

A, B, $SecNeg_b$, {{Enc_{Ka} ($ConfPar_b$)}, R_a, R_b, {$Cert_b$}, Sig_{Kb} (Hash (A, B, R_a, R_b, $SecNeg_a$, $SecNeg_b$, {$ConfPar_b$}))}

Bits								
8	7	6	5	4	3	2	1	Octet(s)
0	0	1	0	0	0	1	0	
In-Band Message Type								1
0x0099 (153)								
In-Band Message Length								2 - 3
Security Services Information Element								
x	x	x	x	x	x	x	x	4
Information element identifier								
1 Ext	Coding Standard		Information Element Instruction Field					
			Flag	Reserved	Information Element Action Indicator			5
0x0095 (149)								6 – 7
Length of Security Services Information Element								
0	0	0	0	0	0	0	1	
Security Association Service Identifier (SME)								8
0x0092 (146)								
Security Association Section Length								9 – 10
0	0	0	1	1	0	1	1	
Version			Transport Ind.		Flow Indicator		Discard	11
1	1	1	1	1	1	1	1	
Region								12
1	1	1	0	0	0	0	0	
Role (any)								13
0	0	0	0	0	0	0	1	
Relative ID (Security Association ID)								14
0	0	0	1	0	0	0	0	
Relative ID (SAS Number)				reserved				15
0	0	1	0	0	1	1	1	
Security Message Exchange Format (Three Way)								16
1	0	0	0	0	0	1	0	
Initiator Distinguished Name Identifier (*A*)								17
0	0	0	1	0	1	0	0	
Initiator Distinguished Name Length								18
0	0	0	0	0	1	0	0	
Initiator Distinguished Name Type (AESA NSAP ICD)								19

0x43023405152319121100980962184 30DE6372812								
Initiator Distinguished Name Value								20 – 39
1	0	0	0	0	0	1	1	
Responder Distinguished Name Identifier (*B)*								40
0	0	0	1	0	1	0	0	
Responder Distinguished Name Length								41
0	0	0	0	0	1	0	0	
Responder Distinguished Name Type (AESA NSAP ICD)								42
0x43023405152319121100980962 45F2A8B2127400								
Initiator Distinguished Name Value								43 – 62
1	0	0	0	1	0	0	0	
Security Service Specification Identifier (*SecNeg*$_b$)								63
1	0	0	0	1	0	1	0	
Security Service Declaration Identifier								64
0	0	1	0	1	1	0	1	
Security Service Declaration								65
1	0	1	0	0	0	0	0	
Data Confidentiality Algorithm Identifier								66
0	0	0	0	1	1	0	0	
Length of Data Confidentiality Algorithm Contents (12)								67
0	0	0	0	1	0	0	0	
Data Confidentiality Algorithm (FEAL)								68
0	0	0	0	0	0	1	0	
Data Confidentiality Algorithm Mode of Operation (CBC)								69
0x238609F0E71A9D17								
Data Confidentiality Algorithm Details Initialization Vector (FEAL CBC)								70 – 77
1	0	1	0	0	1	1	0	
Signature Algorithm Identifier								78
0	0	0	0	0	0	0	1	
Length of Signature Algorithm Identifier								79
0	0	0	0	1	0	0	0	
Signature Algorithm (FEAL/CBC)								80
1	0	1	0	0	1	0	0	
Hash Algorithm Identifier								81
0	0	0	0	0	0	0	1	
Length of Hash Algorithm Contents (1)								82
0	0	0	0	0	0	0	1	
Hash Algorithm (HMAC-MD5)								83
1	0	1	0	1	0	0	0	
Key Exchange Algorithm Identifier								84
0	0	0	0	0	0	0	1	
Length of Key Exchange Algorithm Contents (1)								85
0	0	0	0	1	0	0	0	
Key Exchange Algorithm (FEAL/CBC)								86
1	0	1	0	1	0	1	0	
Session Key Update Algorithm Identifier								87
0	0	0	0	0	0	0	1	

Length of Session Key Update Algorithm Contents (1)								88
0	0	0	0	0	0	0	1	
Session Key Update Algorithm (SKE with MD5)								89
0	1	0	0	0	0	0	0	
Confidential Section Identifier *(ConfPar$_b$)*								90
0	0	0	1	1	0	0	0	
Length of Confidential Section (64)								91
Confidential Data								92 - 115
1	1	0	1	0	0	0	0	
Authentication Section Identifier								116
1	1	0	1	0	1	0	0	
Initiator Random Number Identifier (R_a)								117
0x7A52D245								
Initiator Random Number Value								118 - 125
1	1	0	1	0	1	0	1	
Responder Random Number Identifier (R_b)								126
0x3A29657D								
Responder Random Number Value								127 - 134
1	1	0	1	1	0	1	0	
Security Message Exchange Digital Signature Identifier								135
0	0	0	1	0	1	0	0	
Security Message Exchange Digital Signature Length (64)								136
Digital Signature Value								137 - 156.

FLOW3-3WE: A –> B

{A, B, {Enc$_{Kb}$ (ConfPar$_a$)}, R$_b$, Sig$_{Ka}$ (Hash (A, B, R$_b$, {ConfPar$_a$}))}

Bits								
8	7	6	5	4	3	2	1	Octet(s)
0	0	1	0	0	0	1	1	
In-Band Messsage Type								1
0x0084 (132)								
In-Band Message Length								2-3
Security Services Information Element								
x	x	x	x	x	x	x	x	4
Information element identifier								
1 Ext	Coding Standard		Information Element Instruction Field					5
			Flag	Reserved	Information Element Action Indicator			
0x0080 (128)								
Length of Security Services Information Element								6 - 7
0	0	0	0	0	0	0	1	
Security Association Service Identifier (SME)								8
0x007D (125)								
Security Association Section Length								9 - 10
0	0	0	1	1	1	0	1	
Version			Transport Ind.		Flow Indicator		Discard	11
1	1	1	1	1	1	1	1	
Scope								12
0	0	0	0	0	0	0	1	
Relative ID (Security Association ID)								13
0	0	0	1	0	0	0	0	
Relative ID (SAS Number)				reserved				14
0	0	1	0	0	1	1	1	
Security Message Exchange Format (Three Way)								15
1	0	0	0	0	0	1	0	
Initiator Distinguished Name Identifier (*A*)								16
0	0	0	1	0	1	0	0	
Initiator Distinguished Name Length								17
0	0	0	0	0	1	0	0	
Initiator Distinguished Name Type (AESA NSAP ICD)								18
0x43023405152319121100980962184 30DE6372812								
Initiator Distinguished Name Value								19 - 38
1	0	0	0	0	0	1	1	
Responder Distinguished Name Identifier (*B*)								39
0	0	0	1	0	1	0	0	
Responder Distinguished Name Length								40
0	0	0	0	0	1	0	0	

Responder Distinguished Name Type (AESA NSAP ICD)								41
0x43023405152319121100980962 45F2A8B2127400								
Initiator Distinguished Name Value								42 - 61
1	1	0	0	0	0	0	0	
Confidential Section Identifier *(ConfPar$_a$)*								62
0	0	0	1	1	0	0	0	
Length of Confidential Section (24)								63
Confidential Data								64 - 87
0	1	0	1	0	0	0	0	
Authentication Section Identifier								88
1	0	1	1	0	1	0	1	
Responder Random Number Identifier (R_b)								89
0x3A29657D								
Initiator Random Number Value								90 - 93
1	0	1	1	1	0	1	0	
Security Message Exchange Digital Signature Identifier								94
0	1	0	0	0	0	0	0	
Security Message Exchange Digital Signature Length (64)								95
Digital Signature Value								96 - 135

FLOW3-3 CONFIRM-AP: B –> A

CONFIRM-AP

Bits								
8	7	6	5	4	3	2	1	Octet(s)
0	0	1	1	0	0	0	1	
In-Band Message Type								1
0x0000 (0)								
In-Band Message Length								2 - 3