Stable Implementation Agreements for Open Systems Interconnection Protocols: Part 12 - OS Security

Output from the December 1993 Open Systems Environment Implementors' Workshop (OIW)

Acting SIG Chair: **Richard Harris, The Boeing Company**
SIG Editor: **Dr. Mohammad Mirhakkak, MITRE**

PART 12 - SECURITY December 1993 (Stable)

Foreword

This part of the Stable Implementation Agreements was prepared by the Security Special Interest Group (SECSIG) of the Open Systems Environment Implementors' Workshop (OIW) hosted by the National Institute of Standards and Technology (NIST). See Part 1 - Workshop Policies and Procedures of the "Draft Working Implementation Agreements Document."

Text in this part has been approved by the Plenary of the above-mentioned Workshop. This part replaces the previously existing chapter on this subject. There is significant technical change from this text as previously given.

Future changes and additions to this version of these Implementor Agreements will be published as change pages. Deleted and replaced text will be shown as strikeout. New and replacement text will be shown as shaded.

Table of Contents

PART 12 - SECURITY December 1993 (Stable)

List of Figures

List of Tables

Part 12 - Security

Editor's Note - Previous material in this part has been deleted and is no longer applicable.

Introduction

The relationship between protocols and security is accomplished by developing a security profile that binds these two together. Security profiles define protocol specific implementations of security architectures.

A security profile includes the following items:

A grouping of the security services to be offered;

The placement of those security services;

The selection of mechanisms to support the placed security services.

This part completes this sequence of steps for several generalized security architectures. A generalized security architecture is chosen and tailored to derive a protocol-specific security profile. This part is comprised of protocol-specific security profiles and other supporting functions.

Scope

Normative References

[**ISO7498-2**] ISO/IEC 7498-2 *Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture, February 1989.*

[**ISO8649**] ISO/IEC 8649: 1988/Amd 1:1990 *Service Definition for the Association Control Service Element, Amendment 1: Peer-Entity Authentication During Association Establishment.*

[**ISO8650**] ISO/IEC 9594-3 *Information Technology - Open Systems Interconnection - The Directory - Part 3: Abstract Service Definition*.

[**ISO8650/1**] ISO/IEC 8650: 1988/Amd 1:1990 *Protocol Specification for the Association Control Service Element, Amendment 1: Peer-Entity Authentication During Association Establishment*.

[**ISO9594-7**] ISO/IEC 9594-7 *Information Processing Systems - Open Systems Interconnection - The Directory - Part 7: Selected Object Classes, 1990.*

[**ISO9594-8**] ISO/IEC 9594-8 *Information Processing Systems - Open Systems Interconnection - The Directory - Part 8: Authentication Framework, 1990.*

[**ISO10021-4**] ISO/IEC 10021-4 *Information Processing Systems - Text Communication*

Message Transfer System : Abstract Service Definition and Procedures.

[**X.509-88**] CCITT X.509:1988 *The Directory - Authentication Framework.*

[**X.511-88**] CCITT X.511:1988 *The Directory - Abstract Service Definition*.

[**X.411-84**] CCITT X.411:1984 *Message Transfer System - Message Transfer Layer.*

[**X.521-88**] CCITT X.521:1988 *The Directory - Selected Object Classes.*

Definitions

Symbols and Abbreviations

Architectures

The purpose of this clause is to provide guidance on how to build a security architecture based on an OSI application environment and its threats and vulnerabilities.

A Security Architecture specifies the relationship between the set of security services and mechanisms with which protection from threats and vulnerabilities is achieved. It is designed to respond to assessed vulnerabilities, threats, and risks as identified by a security policy. The establishment of security policies is beyond the scope of the OIW.

Introduction

Open Systems Security provides for secure distributed information processing in OSI application environments which are heterogeneous in terms of technology and administration. For example, some environments may require protection from a minimal set of security threats while others require more complete protection.

The sequence of steps by which a security architecture is created for a specific application environment is as follows:

Development of threat analysis;

Determination of security services;

Placement of security services;

Selection of mechanisms;

Selection of algorithms.

These implementation agreements assume that steps a and b have been completed for the specific application. An introduction to the threat analysis process and the determination of security services is included in Annex H.

Generic OSI application environments are defined in Clause 5.2. Generic security services as defined by ISO 7498-2 are grouped into classes in Clause 5.3. A generalized security architecture for each environment is developed by mapping the security classes onto the functional groups of each environment and providing guidance as to at which layer to support the service in Clause 5.4. Guidance on how to select mechanisms suitable for each security service is presented in Clause 5.5.

It is beyond the scope of these implementation agreements to specify the use of one algorithm over another. Clause 7 presents a set of algorithms suitable for various mechanisms.

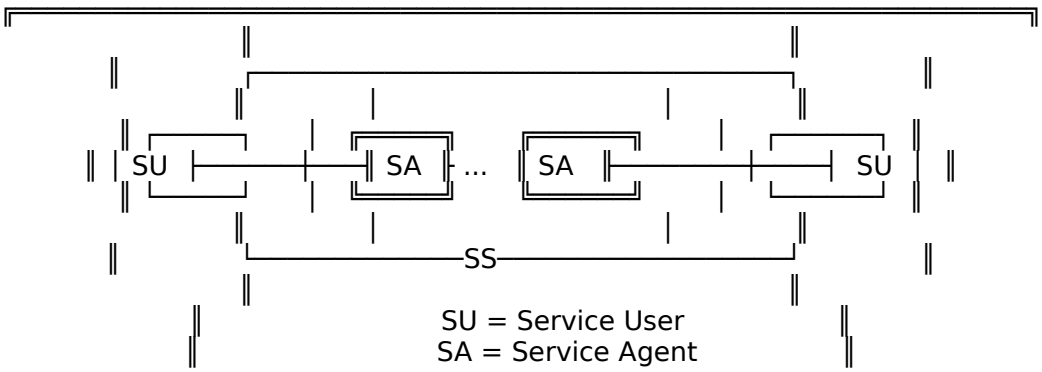
Application Environments

It is useful for the sake of simplification to look at the OSI application environments and to separate them into generic OSI application environments so that security profiles can be developed for each. The environments are: Single Application Association, Application Relay, and Distributed Applications. All applications will operate in one or more of these environments. For example, a Message Handling application that uses a Message Transfer Agent (MTA) to relay mail from one User Agent (UA) to another UA would map to the Application Relay Environment. Likewise a Message Handling application which only includes a UA accessing a Message Store (MS) would map to the Single Application Association Environment.

For each environment, an architectural diagram is provided that portrays the interconnection of the elements. In addition, a set of functional groups are defined each of which is comprised of an interconnected set of elements.

Base Environment

Figure 1 depicts the basic elements of a generic OSI application environment from which all OSI application environments can be derived. In all application environment figures, dashed lines indicate an optional communication path and the double-lined boxes indicate an optional basic element. Ellipses indicate that the previous basic element may be repeated zero or more times.



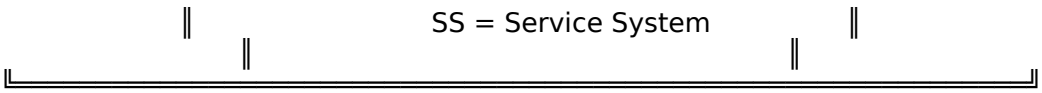


Figure 1 - Basic Elements of a Generic OSI Application Environment

The basic elements are as follows:

Service User (SU): an entity that functions as a service initiator or responder ;

Service Agent (SA): an intermediate entity that actively participates in providing the services between an initiator and a responder;

Service System (SS): zero or more cooperating service agents.

Basic elements that communicate, either through a direct association or indirectly through intermediaries, are classified as a functional group. Functional groups defined in Figure 1 are:

a. f_0: SU -> SU (Service User to Service User directly);

b. f_1: SU => SU (Service User to Service User indirectly);

c. f_2: SU -> SA (Service User to Service Agent directly);

d. f_3: SU => SA (Service User to Service Agent indirectly);

e. f_4: SA -> SA (Service Agent to Service Agent directly);

f. f_5: SA => SA (Service Agent to Service Agent indirectly);

g. f_6: SA -> SU (Service Agent to Service User directly);

h. f_7: SA => SU (Service Agent to Service User indirectly).

Editor's Note - the "->" notation indicates association security relationship and "=>" indicates relay security relationship.

These definitions and this functional group syntax will be used to define generic OSI application environments. In some applications, these functional groups may have to be combined for the purpose of performing a security analysis.

Single Application Association Environment

The Single Application Association Environment covers applications which are designed to operate over Single Application Associations (as defined in ISO 9545) between one pair of application-entity-invocations (AEIs). This environment specifically includes the case of recovery, i.e., different associations may exist at different times between one pair of AEIs.

Examples of applications to which this environment applies are as follows:

FTAM;

Network Management;

Virtual Terminal.

Applications such as MHS, Directory Services, and TP are only partially covered by this environment because some of their service elements may use store and forward or chaining types of relay functions. The environments that apply to these applications are the Application Relay and Distributed Applications Environments respectively.

Architectural Diagram

Figure 2 portrays the architectural diagram for the Single Application Association Environment.

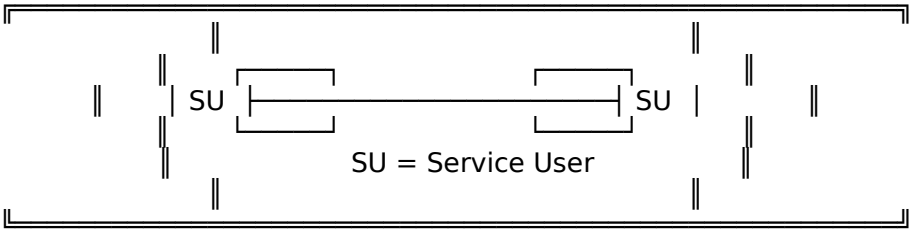


Figure 2 - Architectural Diagram for Single Application Association Environment

Functional Groups

The following functional group is defined for the Single Application Association Environment:

f_0:SU -> SU.

Application Relay Environment

The Application Relay Environment covers applications which are designed to operate with

the active participation of at least one service agent in support of transferring a service user request from an initiator to a responder. When more than one service agent is present, they function sequentially.

An example of an application to which this environment applies is Message Handling Systems.

Architectural Diagram

Figure 3 portrays the architectural diagram for the Application Relay Environment. In all application environment figures, dashed lines indicate an optional communication path and the double-lined boxes indicate an optional basic element. Ellipses indicate that the previous basic element may be repeated zero or more times.

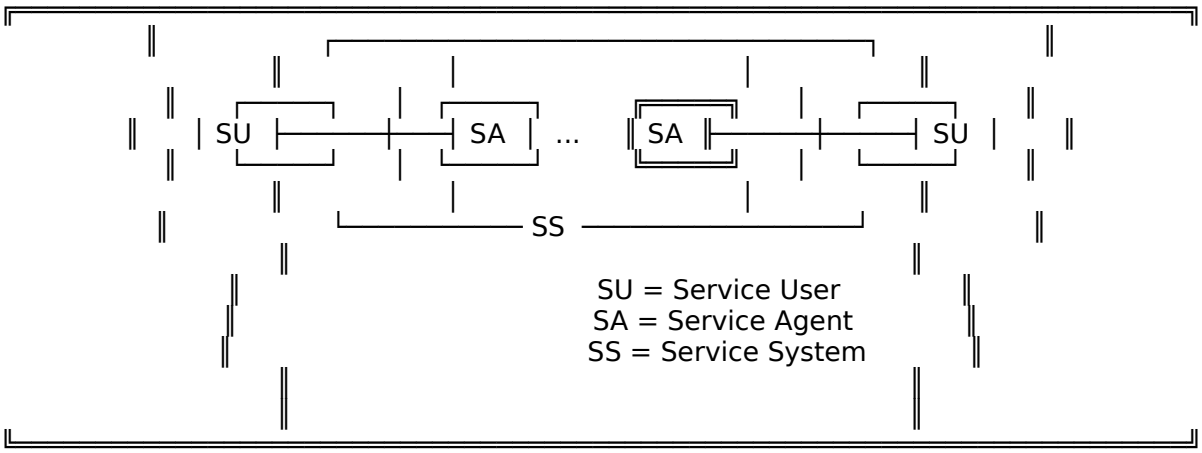


Figure 3 - Architectural diagram for Application Relay Environment

Functional Groups

The following functional groups are defined and added for the Application Relay Environment:

a. f_2: SU -> SA;

b. f_3: SU => SA;

c. f_4: SA -> SA;

d. f_6: SA -> SU.

Distributed Applications Environment

PART 12 - SECURITY December 1993 (Stable)

The Distributed Application Environment covers applications which are designed to operate with the active participation of zero or more service agents which may process a service user request. Processing may include modifying, interpreting, or transferring the service user request or its data. When more than one service agent is present, they may function in parallel, sequentially, or both.

Architectural diagram

Figure 4 portrays the architectural diagram for the Distributed Applications Environment. In all application environment figures, dashed lines indicate an optional communication path and the double-lined boxes indicate an optional basic element. Ellipses indicate that the previous basic element may be repeated zero or more times.

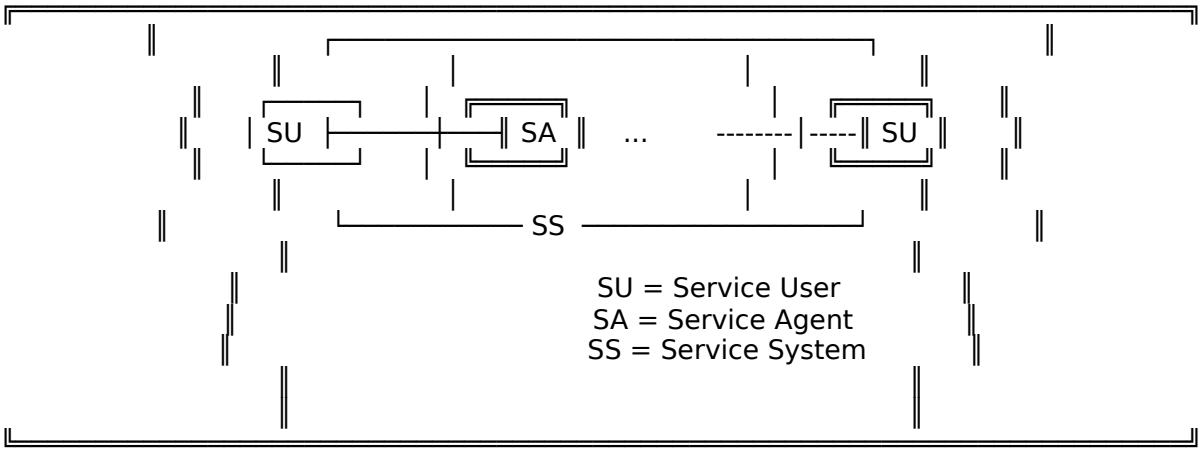


Figure 4 - Architectural diagram for Distributed Applications Environment

Functional Groups

The following functional groups are defined and added for the Distributed Applications Environment:

f_0: SU -> {SU; ... };

f_1: SU => {SU; ... };

f_2: SU -> {SA; ... };

f_3: SU => {SA; ... };

f_4: SA -> {SA; ... };

f_5: SA => {SA; ... };

f_6: SA -> {SU; ... };

f_7: SA => {SU; ... }.

Security Classes

Security classes are defined to provide a framework on which to build security profiles. Each class specifies the required security services. The services specified in each class are the generic security services as defined by ISO 7498-2. For each application's profile, specific security services are chosen for each class. For example, data integrity is a generic security service for which there exists five distinct data integrity services. One or more specific security services must be specified to meet the requirements of a security class in an application specific security profile.

The classes are organized into two similar hierarchies as shown in Table 1. Each level of each hierarchy is a superset of the security services required of the immediately preceding level. For each level in the hierarchies, the same set of security services are required, except that one hierarchy includes confidentiality services.

Table 1 - Security Classes

SECURITY SERVICES	SECURITY CLASSES	
		ADD CONF
AUTH. & ACCESS CONTROL	S0	
		S0A
ADD DATA INTEGRITY	S1	
		S1A
ADD NON-REPUDIATION	S2	
		S2A

There are two interesting properties of these relationships between the classes. First, each level of the confidentiality hierarchy is a superset of the other hierarchy at the same level and a superset of the confidentiality hierarchy at the immediately preceding level. For example, class S2A is a superset of classes S2 and S1A.

Second, for two entities each supporting a distinct security class in a different hierarchy, the best level of service that can be achieved between them is the class in the non-confidentiality hierarchy at the same level as the lowest class of the two entities. For example, if one entity supports class S2 and the other supports class S1A, the best class of service achievable is S1.

Editor's Note - This is not a mechanism for negotiated services. That is a future work item.

Security Class S0

The Security Class S0 includes implementation of the following security services:

a) S0 = Authentication and Access Control.

The Security Class S0A adds the confidentiality service to the Class S0 as follows:

b) S0A = S0 + Confidentiality.

Security Class S1

The Security Class S1 adds the Data Integrity Service to class S0 as follows:

a) S1 = S0 + Data Integrity.

The Security Class S1A adds the Confidentiality Service to Class S1 as follows:

b) S1A = S1 + Confidentiality

Security Class S2

The Security Class S2 adds the Non-repudiation Service to Class S1 as follows:

S2 = S1 + Non-repudiation

The Security Class S2A adds the Confidentiality Service to Class S2 as follows:

b) S2A = S2 + Confidentiality

Guidelines for OIW Application Profile Development

Key Management

[ISO7498-2] defines Key Management (KM) as the "generation, storage, distribution, deletion, archiving, and application of keys in accordance with a security policy." The Security SIG recognizes that security policies are outside the scope of IAs, and it is inappropriate to make general recommendations in the absence of a KM framework.

Security Algorithms

PART 12 - SECURITY December 1993 (Stable)

> **Editor's Note -** Implementors are cautioned that security of an algorithm may change at any time. Therefore, the WIA must be consulted in order to determine if there is more current information.

The algorithms included here are listed in no particular order (beyond categorization by type of algorithm). It is beyond the scope of these agreements to recommend the use of one algorithm over another. However, if a vulnerability is known to exist a reference will be provided along with a recommendation not to use the algorithm.

This clause references a definitive specification for each algorithm, which includes an object identifier. In general, control of the definitive specification is expected to be outside the scope of the OIW. The benefit of not controlling the specification is that the organization that developed the algorithm is best situated to maintain and have knowledge of the security of the algorithm. Algorithms for which there is no controlling organization are defined in an Annex in this Part.

For each algorithm, its typical usage is stated, its definitive reference is given, and its object identifier is included for reference purposes.Optionally, additional information may be included, for example a reference to known vulnerabilities.

Implementors should be aware that export of products using cryptography may be subject to export restrictions. In general, use of cryptography not involving confidentiality is subject to Commerce Department regulations, while use of cryptography for confidentiality is controlled by (more stringent) State Department regulations. It is the implementor's responsibility to determine any export restrictions which apply to a given product, as the export controls may change from time to time.

> **Editor's Note -** Some of the references are RFCs, Internet Drafts, and PKCS documents. We need to include information on how to access these documents.

Message Digests

These message digest algorithms (or hash algorithms) compute a fixed size representation of an input stream. They have different performance characteristics and employ different computational techniques, making each suitable for different applications.

Square-Mod-N

Square-Mod-N is a hash algorithm that is used to compute a fixed size representation of an input stream. It is defined in [X.509] and its object identifier is defined there as:

```
sqmod-n ALGORITHM
PARAMETER BlockSize
::= {hashAlgorithm 1}

BlockSize ::= INTEGER
```

Recent research regarding the square-mod-n one-way hash function described in Annex D of

[X.509] has revealed that the function is not secure. Its use, therefore, is discouraged.

> **Editor's Note -** We need the reference that identifies its vulnerabilities so we can recommend it not be used.

MD2

MD2 is a message digest algorithm that employs accepted, traditional computational techniques. Its speed is the slowest of the message digests listed here.

It is defined in Internet Draft [a] and its object identifier is defined there as:

```
md2 ALGORITHM
PARAMETER NULL
::= {iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 2}
```

> **Editor's Note -** There is a Directory SIG OID for this algorithm.

The reference includes a source code implementation of the algorithm written in the C programming language. MD2 is copyrighted and its use may require specific permission or a license. Details are stated in the Internet Draft.

MD4

MD4 is a message digest algorithm that employs non-traditional computational techniques to enhance its speed in software and hardware with native 32-bit arithmetic. Its speed is the fastest of the message digests listed here.

It is defined in Internet Draft [b] and its object identifier is there as:

```
md4 ALGORITHM
PARAMETER NULL
::= {iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 4}
```

This reference includes a source code implementation of the algorithm written in the C programming language.

It is suggested that MD4 be used only with applications for which performance is critical.

> **Editor's Note -** We need to include text from the MD4/5 Internet Drafts which describes the differences between the two algorithms and the preference for MD5.

MD5

MD5 is a message digest algorithm which is based on the techniques of MD4, but with additional enhancements to counter proposed attacks. A detailed description of the

changes can be found in [c]..

MD5 is defined in Internet Draft [c] and its object identifier is defined there as:

```
md5 ALGORITHM
PARAMETER NULL
::= {iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 5}
```

This reference includes a source code implementation of the algorithm written in the C programming language.

SHA

This algorithm is the NIST Secure Hash Algorithm [ab]. It is based on concepts similar to those used in MD4 and MD5, and outputs a 160-bit digest.

```
sha ALGORITHM
PARAMETER NULL
::= {algorithm 18}
```

Editor's Note - This and other algorithms may be registered by ISO instead, in which case this text will be adjusted prior to moving to Stable Agreements, or if necessary as Alignment Errata.

Reversible Public Key Algorithms

These algorithms are asymmetric; separate keys are used for encryption and decryption. They also have the property that applying the encipherment function followed by the decipherment function has the same effect as applying the decipherment function followed by the encipherment function. This is useful if a single algorithm is needed to provide both confidentiality (e.g., transport of symmetric keys) and authentication/integrity (e.g., digital signatures).

RSA is a public key (asymmetric) cryptographic algorithm, typically used in conjunction with message digest (or hash) algorithms to create digital signatures and for confidential distribution of symmetric keys. It may also be used to exchange confidential messages.

The RSA algorithm is defined in [d] and is also described in Annex C of [X.509]. The RSA technology is patented in the United States [e][f].

According to [X.509], the ASN.1 BIT STRING containing the public key will contain the BER encoding of the modulus and exponent:

```
SEQUENCE {
        n       INTEGER,        -- modulus
        e       INTEGER }       -- public exponent
```

RSA (X.509)

RSA is defined in [X.509] and its object identifier is defined there as:

```
rsa ALGORITHM
PARAMETER KeySize
::= {encryptionAlgorithm 1}

KeySize ::= INTEGER
```

The key size specifies the length in bits of the RSA public key modulus.

The definition of this algorithm does not include specification of padding rules. If one assumes that the data is treated as an integer and padded with zero bits on the left, the algorithm is subject to various attacks, such as those described in [ah], which render it unsuitable for some applications, e.g., multi-recipient mail, notarization. In such cases RSAEncryption is preferred.

RSA Encryption

RSA Encryption is defined in PKCS #1 [g] and its object identifier is defined there as:

```
rsaEncryption ALGORITHM
PARAMETER NULL
::= {iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1}
```

This algorithm defines various types of block padding depending on whether the block is being encrypted using a public or private key. The padding protects against various attacks documented in the literature.

RSA Signature

This algorithm [ad] is compatible with IS 9796 [ae], with the Sign and Verify functions required to be those in Annex A of ISO 9796.

```
rsaSignature ALGORITHM
PARAMETER NULL
::= {algorithm 11}
```

This algorithm provides additional redundancy in the construction of the signature block, and ensures that it is not a natural power. (If the signature block is a natural power, one can forge a signature by simply taking the e-th root where e is the public exponent. E.g., if e is 3, one could potentially forge a signature, if the block is a natural cube, by taking the (integer) cube root. However, the chance that an integer near a given x is a cube is quite small if x is large; the probability $x^{-2/3}$, so if x is about 2^{505} (as is the case for 512-bit RSA), then the probability is about 2^{-337}.)

Irreversible Public Key Algorithms

These algorithms are not reversible, as defined in section 7.2. Typically, different algorithms are used for encryption and signature. This section defines several signature-only algorithms. Note that these algorithms expand the plaintext, producing output which is significantly larger than the input block or digest. These algorithms are of use in authentication-only systems, and are generally not subject to export restrictions.

El Gamal

ElGamal is a public key (asymmetric) digital signature algorithm. It is defined in [k]. Its object identifier is:

```
ElGamal ALGORITHM
PARAMETER NULL
::= {encryptionAlgorithm 1}
```

Editor's Note - This OID was assigned by the Directory SIG.

In [X.509], the ASN.1 data element subjectPublicKey defined as BIT STRING should be interpreted in the case of ElGamal as being of type:

```
SEQUENCE {
prime INTEGER, -- p
base INTEGER,  -- alpha
key INTEGER  -- public key, Y
}
```

Also, in [X.509], the value associated with the ENCRYPTED MACRO should be interpreted in the case of ElGamal as being of type:

```
SEQUENCE {
s INTEGER,
r INTEGER
}
```

The ElGamal technology is patented in the United States [f].

Editor's Note - Should we describe and define OIDs for the message digest with ElGamal signature algorithms? There is a Directory SIG OID for md2WithElGamal.

DSA

The NIST Digital Signature Algorithm [aa] is a variant of ElGamal which produces a shorter signature size. Its object identifier is:

```
dsa ALGORITHM
PARAMETER NULL
::= {algorithm 12}
```

In [X.509], the ASN.1 data element subjectPublicKey defined as BIT STRING should be interpreted in the case of DSA as being of type:

```
DSAPublicKey ::= SEQUENCE {
        key             INTEGER,        -- y (public key)
        params          DSAParameters OPTIONAL }

DSAParameters ::= SEQUENCE {
        prime1          [0]     INTEGER,        -- p
        prime2          [1]     INTEGER,        -- q
        base            [2]     INTEGER }       -- g
```

Also, in [X.509], the value associated with the ENCRYPTED MACRO (i.e. the signature value) should be interpreted in the case of DSA as being of type:

```
SEQUENCE {
        s INTEGER,
        r INTEGER }
```

Key Exchange

Diffie-Hellman

Diffie-Hellman Key Exchange is a public key (asymmetric) algorithm whereby two parties, without any prior arrangements, can agree upon some shared (secret) information. The parties exchange public information which, in conjunction with private information retained by each user, may be used to compute a common value. This value is typically used as a symmetric key, for example, to encrypt further communications between the parties.

The Diffie-Hellman Key Exchange is defined in [h] and is also described in [j]. The Diffie-Hellman Key Exchange is patented in the United States [i][f].

The object identifier is defined in PKCS #3 [j] as:

```
dhKeyAgreement ALGORITHM
PARAMETER DHParameter
::= {iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) pkcs-3(3) 1}

DHParameter ::= SEQUENCE {
prime                   INTEGER, -- p
base                            INTEGER   -- g
privateValueLength      INTEGER OPTIONAL
}
```

Diffie-Hellman with Common Parameters

This version of Diffie-Hellman assumes the use of a common modulus and generator, which are distributed by external means rather than being conveyed in the parameter component of the AlgorithmIdentifier. The patent restrictions in the previous section still apply.

The object identifier is defined as:

```
dhWithCommonModulus ALGORITHM
PARAMETER NULL
::= {algorithm 16}

DHParameter ::= SEQUENCE {
prime INTEGER, -- p
base INTEGER   -- g
}
```

Signature Algorithms

This section specifies a number of signature algorithms, i.e., hash algorithms combined with appropriate asymmetric encryption algorithms.

Message Digests with RSA

The algorithms listed below are signature algorithms that combine a message digest algorithm with the RSA cryptographic algorithm to produce a digital signature.

> **Editor's Note -** The OIDs below have been assigned by the Directory SIG and the Security SIG. Should we explain why they do not appear in a single tree?

Square-Mod-N with RSA

Square-Mod-N is a signature algorithm that combines the Square-Mod-N hash algorithm with the RSA cryptographic algorithm to produce a digital signature. This algorithm is defined in [X.509] and its object identifier is defined there as:

```
sqmod-Nwithrsa ALGORITHM
PARAMETER KeyAndBlockSize
::= {signatureAlgorithm 1}

KeyAndBlockSize ::= INTEGER
```

Recent research regarding the square-mod-n one-way hash function described in Annex D of [X.509] has revealed that the function is not secure. Its use, therefore, is discouraged.

MD2 with RSA

Its object identifier is:

```
md2WithRsa ALGORITHM
PARAMETER NULL
::= {signatureAlgorithm 1}
```

This OID was assigned by the Directory SIG.

MD4 with RSA

Its object identifier is:

```
md4WithRSA ALGORITHM
PARAMETER NULL
::= {algorithm 2}
```

MD5 with RSA

Its object identifier is:

```
md5WithRSA ALGORITHM
PARAMETER NULL
::= {algorithm 3}
```

Message Digests with RSA Encryption

The algorithms listed below are signature algorithms that combine a message digest algorithm with the RSA Encryption cryptographic algorithm to produce a digital signature.

MD2 with RSA Encryption

MD2 with RSA encryption is defined in PKCS #1 [g] and its object identifier is defined there as:

```
md2WithRSAEncryption ALGORITHM
PARAMETER NULL
::= {iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) pkcs-1(1) 2}
```

MD4 with RSA Encryption

Its object identifier is:

```
md4WithRSAEncryption ALGORITHM
PARAMETER NULL
::= {algorithm 4}
```

MD5 with RSA Encryption

MD5 with RSA Encryption is defined in PKCS #1 [g] and its object identifier is defined there as:

```
md5WithRSAEncryption ALGORITHM
PARAMETER NULL
::= {iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) pkcs-1(1) 4}
```

DSA With SHA

This signature algorithm produces a 320-bit signature. SHA is the only hash algorithm which may be used with DSA. Its object identifier is

```
dsaWithSHA ALGORITHM
PARAMETER NULL
::= {algorithm 13)
```

RSA Signature With SHA

This algorithm uses the RSA Signature algorithm to sign a 160-bit SHA digest. Its object identifier is

```
shaWithRSASignature
PARAMETER NULL
::= {algorithm 15}
```

Symmetric Encryption Algorithms

Data Encryption Standard

The Data Encryption Standard (DES) is a secret key (symmetric) cryptographic algorithm. It is defined in FIPS 46-1 [l]. It is also defined as DEA-1 in ANSI X3.92-1981 [m].Implementors will also find several other references useful. FIPS PUB 74 [p] provides guidance on the implementation and use of DES and includes a complete specification of the algorithm. SPEC PUB 500-20 [p] describes the design and operation of the NIST (formerly NBS) testbed that is used for the validation of DES implementations. It specifies a set of 291 test cases that have been designed to exercise every basic element of the algorithm, and as a further check on the correctness of an implementation, it specifies an extensive Monte Carlo analysis. SPEC PUB 500-61 describes the design of four maintenance tests for DES implementations. The tests consist of an iterative test procedure that uses a small program and minimum data. The tests are designed to be independent of implementation and to be fast enough to test devices during actual operation. The tests are defined as four specific stopping points in a general testing process and satisfy four testing requirements of

increasing degree of completeness on the thoroughness of testing desired.

There are four modes of operation of the DES, as specified by FIPS 81 [n] and ANSI X3.106-1983 [o]. The modes specify how the data will be encrypted and decrypted. In all cases the key is 64 bits. Use of DES for encryption (i.e., all modes discussed below except DES-MAC) are subject to export controls.

DES-ECB

This is the Electronic Codebook mode of operation. Its object identifier is:

```
desECB ALGORITHM
PARAMETER NULL
::= {algorithm 6}
```

This mode should be used to encrypt small blocks (e.g., other DES keys). Its use is deprecated for block encryption since it allows cryptanalysis of repeated block values (i.e., the same plaintext in the same place relative to the block), as well as reassembling messages from known blocks.

DES-CBC

This is the Cipher Block Chaining mode of operation. Its object identifier is:

```
desCBC ALGORITHM
PARAMETER CBCParameter
::= {algorithm 7}
```

The PARAMETER is needed to specify the Initialization Vector, which need not be kept secret.

This mode should be used to encrypt multiple blocks, where the full message is available. The random IV prevents codebook analysis of the start of the chain. The IV may be public.

This mode will propagate a single bit error in one plaintext block into all succeeding blocks, and will propagate a single bit error in the ciphertext into a garbled plaintext block on decryption as well as a single bit error in the **next** plaintext block.

The following padding mechanism from [w] should be used if the data to be encrypted is octet aligned, unless the security policy dictates otherwise:

The input to the DES CBC encryption process must be padded to a multiple of 8 octet, in the following manner. Let n be the length in octets of the input. Pad the input by appending 8-(n mod 8) octet to the end of the message, each having the value 8-(n mod 8), the number of octets being added. In hexadecimal, the possible paddings are: 01, 0202, 030303, 04040404, 0505050505, 060606060606, 07070707070707, and 0808080808080808. All input is padded with 1 to 8 octets to produce a multiple of 8 octets in length. The padding can be removed unambiguously after decryption.

Editor's Note - If adding the padding rules would cause existing implementations to break,

this should be registered as a separate algorithm identifier. Note, however, that [FIPS 81] specifies its own padding rules for padding binary data, in the absence of application-defined rules such as those above; those rules require an indication (which could be conveyed as an algorithm PARAMETER) of whether the data has been padded or not.

DES-OFB

This is the Output Feedback mode of operation. Its object identifier and parameters are:

```
desOFB ALGORITHM
PARAMETER FBParameter
::= {algorithm 8}
```

The parameters are needed to specify an IV and the number of feedback bits.

This mode may be used to encrypt multiple blocks where the error extension properties of DES-CBC are undesirable. A single bit error in the ciphertext will cause only a single bit error in the output plaintext.

DES-CFB

This is the Cipher Feedback mode of operation. Its object identifier and parameters are

```
desCFB ALGORITHM
PARAMETER FBParameter
::= {algorithm 9}
```

The parameters are needed to specify an IV and the number of feedback bits.

This mode may be used when the plaintext is made available in pieces, e.g., a character (8-bit CFB) or a bit (1-bit CFB) at a time. This mode will propagate a single bit error in one plaintext block into all succeeding blocks, and will propagate a single bit error in the ciphertext into a single-bit error in the corresponding plaintext character as well as garbling of the next 8 bytes or so of output (the exact amount depends on the feedback size).

DES-MAC

DES-MAC is a Message Authentication Code algorithm (cryptographic checksum) based on the DES that uses a single 64-bit DES key.

It is specified in FIPS 113 [s] and is equivalent to the binary mode defined in ANSI X9.9-1986 [t]. Its object identifier and parameter are:

```
desMAC ALGORITHM
PARAMETER MACParameter
::= {algorithm 10}
```

The parameter is needed to specify the MAC length in bits.

DES-MAC is equivalent to DES-CBC using an all zero Initialization Vector (IV), with all but the last cipher output block discarded. Separate keys (where one may simply be a variant of the other) should be used if both DES-CBC encrypting and MACing the same data.

> **Editor's Note -** We need to include the reference which specifies the vulnerability when the same key is used to DES-CBC encrypt and MAC the same data, and recommends the use of separate keys.

DES-EDE

The DES algorithm in Encrypt-Decrypt-Encrypt (EDE) mode, as defined by [af] for encryption and decryption with pairs of 64-bit keys, might be used for key or MAC encryption when symmetric key management is employed. (The mechanism is subject to the same constraints as DES ECB, but is cryptographically stronger.) Given the pair of keys, the data is enciphered with the first key, deciphered with the second key, and enciphered again with the first key to perform encryption; the process is reversed for decryption. Note that if both keys are the same, the result is equivalent to a single encryption under the single key. The key may be represented as a single 128-bit string with the first 64 bits being the first key and the last 64 bits being the second key.

```
desEDE ALGORITHM
PARAMETER NULL
::= {algorithm 17}
```

RC2-CBC

RC2-CBC is a symmetric block encryption algorithm. It is proprietary to RSA Data Security, Inc., and a license from them is required to use the algorithm. The algorithm uses an 8-byte key and operates on an 8-byte block, with cipher block chaining as in DES. The recommended padding is as described above for DES-CBC: the final block is padded to an 8-byte boundary by appending 8 - (n mod 8) bytes, each having the value 8 - (n mod 8), where n is the total number of bytes being encrypted. The speed is comparable to DES.

```
rc2CBC ALGORITHM
PARAMETER RC2-CBCParameter
::={iso(1) member-body(2) US(840) rsadsi(113549) encryptionAlgorithm(3) 2}

RC2-CBCParameter ::= CHOICE {IV, SEQUENCE {version RC2Version, IV}}

-- with IV only, version defaults to 65

IV ::= OCTET STRING -- 8 octets
RC2Version ::= INTEGER -- 0 to 255, defined by RSADSI
```

The version number relates to the security level. Different versions of RC2 provide different security levels, some of which are exportable.

RC-4

RC-4 is a symmetric block encryption algorithm. It is proprietary to RSA Data Security, Inc., and a license from them is required to use the algorithm. The RC4 key size is variable, 1 to 256 bytes; the block size is one byte. RC4 is a stream cipher, and it exclusive-ors a pseudorandom sequence generated from the key to encrypt or decrypt; a given key should therefore be used only once. RC4 is very fast.

rc4 ALGORITHM
PARAMETER NULL
::={iso(1) member-body(2) US(840) rsadsi(113549) encryptionAlgorithm(3) 4}

ASN.1

Distinguished Encoding Rules

In order to allow verification of digital signatures produced by the SIGNED and SIGNATURE MACROs of [ISO9594-8], it is necessary to define a set of distinguished encoding rules to produce an unambiguous encoding of a given abstract syntax value. [ISO9594-8] defines a number of such encoding rules (8.7), but is, unfortunately, underspecified in the following areas:

Ordering of SET OF components;

Handling of unused trailing zero bits;

Invocation and designation of new character sets in some of the character string types.

The following rules remove these ambiguities:

The [ISO9594-8] distinguished encoding rules are always used;

For SET OF types, components are sorted into ascending order of the distinguished encodings of the components;

For BIT STRINGS with unused trailing bits, if the type definition that specifies the bits have significance, then they are included in the encoding; otherwise they are not;

For those character strings which allow it, escape sequences are generated to invoke and designate new register entries only when the register entry for the character currently being encoded is different from that currently designated for G0, C0, or C1. All designations shall be into G0 or C0. (It is assumed that all characters have entries in the ISO Registry of Coded Character Sets.)

NOTE - Rules b,c, and d are taken from [ISO/CD8825-3] (Nov. 1990), the ASN.1 Distinguished Encoding Rules. Other features of [ISO/CD8825-3], which conflict with [ISO9594-8] (e.g., length encoding for constructors), are NOT used by this IA.

It is recommended that whenever the SIGNED or SIGNATURE macro is to be applied to an object, the object should be transferred in its distinguished encoded form. In this way, when the resources required to encode or decode an object exceed the resources required to apply the SIGNED or SIGNATURE macro, a receiving entity may apply the macro immediately, thus realizing enhanced performance. However, if the macro application is unsuccessful, the object must be distinguished encoded and the macro re-applied to determine its actual success or failure.

Lower Layers Security

Upper Layers Security

This clause addresses the provision of security services in the Upper Layers. The Upper Layers Security Model specifies the interactions among the Upper Layers in providing and using security services [ISO/CD10745].

Security Mechanisms

Peer Entity Authentication

ACSE authentication extensions [ISO8649][ISO8650/1] support two-way authentication through the definition of a new functional unit. When this functional unit is employed, additional parameters are provided by the A-ASSOCIATE service to indicate this requirement and convey authentication information between entities. The ASN.1 definition for this information is given below:

from [ISO8650/1]:

```
Mechanism-name ::= OBJECT IDENTIFIER
        --This field shall be present if authentication-value is of type ANY.

Authentication-value  :=      CHOICE {
        charstring            [0] IMPLICIT GraphicString,
        bitstring             [1] IMPLICIT BIT STRING,
        external              [2] IMPLICIT EXTERNAL,
        other                 [3] ANY -- Defined by Mechanism-name }

        --The abstract syntax of authentication-value is determined by the authentication-
        mechanism
        --used during association establishment. The authentication-mechanism is either
explicitly
        --denoted by the OBJECT IDENTIFIER value for Mechanism-name, or it is know
implicitly by
```

--prior agreement between the communicating partners. If "other" is chosen, then
--"Mechanism-name" must be present in accordance with ISO 8824.

These agreements define the following mechanisms for use with this ACSE functional unit:

simple-strong authentication mechanism.

Simple-Strong Authentication

Operation

The operation of the simple-strong authentication mechanisms are based upon [ISO9594-3] and [ISO9594-8] standards. The sending system is the entity requesting authentication of its identity, and the receiving system is the entity performing the authentication. The sending system supplies data for the ACSE authentication field of the A-ASSOCIATE primitive. The receiving ACSE obtains the ACSE authentication data from the A-ASSOCIATE PDU, and it performs the authentication check. If the check is successful, the association formation succeeds or fails depending upon other circumstances and parameters. The use of the ACSE authentication fields support both the simple and strong credentials variants of the [ISO9594-8] authentication exchanges.

Certificates for use with strong authentication must be compatible with [ISO9594-8]. Certificates procured for use with Internet Privacy Enhanced Mail [u][v][w][x] are completely compatible with [ISO9594-8] and may (subject to licensing restrictions) be used by the strong authentication mechanism. However, Privacy Enhanced Mail uses only a subset of the suggested [ISO9594-7] name forms, and might not support certain name forms of interest to specific OIW applications. Examples include Application Entity names and certain name forms defined by the North American Directory Forum in NADF-123 [y].

Data Structure

Mechanism Name

The following is the ASN.1 description of the authentication data structure for simple or strong authentication:

```
simple-strong-auth-mechanism OBJECT IDENTIFIER ::= {iso (1)
                    identified-organization (3)
                        oiw (14)
                        secsig (3)
                    authentication-mechanisms (3)
                simple-strong-identity-authentication (1)
                                                    }
```

Authentication Value

The authentication value is conveyed in the other option of the authentication-value field of ACSE authentication.

```
Authentication-Value ::=
SEQUENCE OF DirectoryAbstractService.Credentials
```

This data type is defined in ASN.1 module DirectoryAbstractService of [ISO9594-3] as modified through resolution of Directory Defect Report Numbers 9594/052 and 063. The semantics of all fields are as specified in clause 8.1.2.1 of [ISO9594-3].

The Authentication-Value is defined as a SEQUENCE because it is permitted to pass credentials for multiple entities in the authentication value. It is the responsibility of the application to determine the specific meaning and use of multiple credentials in such a case. It is anticipated that specific applications (e.g., Network Management) would provide specifications for handling multiple credentials within their own clauses of this Part.

This authentication mechanism may employ any registered authentication algorithm; however, it is recommended that the algorithms identified in clause 7 be used.

Options

For the Simple Credentials option of Credentials, the following agreements apply. Conforming implementations are not required to employ the OPTIONAL validity sequence of the SimpleCredential data element. Receiving implementations that do not employ the validity sequence must reject an authentication value which does contain this sequence. Conforming implementations shall employ the optional password field of the SimpleCredential data element.

Note that the password may be hashed using one way functions and the other validity fields. Password is either cleartext, Protected1 or Protected2 according to [ISO9594-8].

External Authentication Mechanisms

Externally defined authentication exchanges may employ the external [2] option of the authentication-value field of ACSE authentication. In this case it is recommended that the mechanism-name be omitted, with the particular mechanism in use being implied by the abstract syntax identified in the external construct.

Kerberos Version 5

One instance of an external authentication mechanism is the Kerberos mechanism defined in [z]. The Kerberos specification assigned the following object identifier to an abstract syntax suitable for use in this way:

[TBD]

Integrity/Data Origin Authentication Transformation

This transformation is a specialization of gulsSignedTransformation, which is defined in clause D.4 of DIS 11586-1. This transformation uses the following parameters, and provides additional details on the operation of the encoding and decoding processes.

1) The initEncRules field has the value { joint-iso-ccitt asn1(1) ber-derived(2) der(1) }, i.e., DER.

2) The signOrSealAlgorithm element shall be keyed-hash-seal:

```
keyed-hash-seal ALGORITHM
        PARAMETER NULL
::= { algorithm 23 }
```

The keyed-hash-seal algorithm is specified in the encoding process description below.

3) The hash algorithm, if the hashAlgorithm element is not present, shall default to md5.

Editor's Note - Points 2 and 3 are redundant with text in the NM Agreements. This should be resolved before progressing to the Stable Agreements.

4) The keyInformation field is not present.

Encoding process: When a value of an abstract syntax is to be sealed for transmission, the following procedures apply:

1) Encode the output data type of the transformation using the ASN.1 Distinguished Encoding Rules, with the shared secret key used as the value of the appendix component. (Since automatic tagging is used, this is equivalent to encoding the unprotectedItem using DER, and enclosing it in the intermediateValue and output data type using BER.)

NOTE - This encoding is only for purposes of the security transformation, and does not mean DER must be used to encode the PDU for transmission, i.e., as the transfer syntax.

2) Hash the complete DER encoding of the value derived in step 1.

NOTE - The current definition of the gulsSignedTransformation is unduly restrictive in that cryptographic operations are only applied to the intermediateValue element of the output data type, rather than the entire type. This is being submitted as a ballot comment on DIS 11586-1.

3) Insert the hash value into the appendix component of the output data type, which is the xformedDataType element of the transmitted PDV.

Encoding process local inputs: Identifier of hash algorithm and any required algorithm parameters, and shared secret key. (Most currently registered hash algorithms have a NULL parameter.)

Decoding process: When a received PDV to be verified, the following procedures apply:

1) Extract and save the received hash value contained in the appendix component of the received xformedDataType component of the received PDV.

2) Replace the value in the appendix component of the xformedDataType component with the shared secret key.

NOTE - The extraction and replacement of the seal field may be performed directly on the ASN.1 encoded PDU if the length of the secret key and the hash digest are equal. Otherwise, the PDU must be decoded and reencoded.

3) Hash the DER encoding of the xformedDataType element. (Reencoding may be avoided if the unprotectedItem encoding is distinguished, and the generic protecting transfer syntax defined in DIS 11586-4 is used.)

4) Compare the hash extracted in step 1 with the hash derived in step 3. If they are equal, then the seal is valid; otherwise an error is signalled.

Decoding process local inputs: Identifier of hash algorithm and any required algorithm parameters, and shared secret key.

Decoding process outputs: Recovered unprotected item. and an indication of whether the seal is valid.

Errors: An error condition occurs if seal verification fails.

Security services: Data origin authentication, data integrity.

Message Handling System (MHS) Security

All current MHS security relevant text appears in Part 8, clause 10.

Directory Services Security

Network Management Security

This clause outlines an approach to providing security services for OSI Network Management. The goals of this approach are to provide security in a manner that is simple and straight-forward to implement, and to avoid any unnecessary computational and managerial overhead. The approach also takes into consideration the need for different levels of security services within different network management domains, and the near term requirement for interoperability of network management entities over heterogeneous network types.

Threats

For the purpose of discussion, threats are divided into two categories: primary and secondary threats. Primary threats are those considered to be applicable to the full range of network management implementations, while secondary threats are considered to be applicable to the more limited range of highly secure implementations.

The primary threats to be protected against are the following:

The masquerading of a manager or agent entity;

The fabrication or modification of Common Management Information Protocol (CMIP) data units.

By countering primary threats, disruption of network management services by the casual user can be avoided.

The secondary threats to be protected against are the following:

All primary threats;

The disclosure of CMIP data units;

The replay, reflection, reordering, insertion, or deletion of CMIP data units.

Security Services

Basic Security Services

The security services required to counter primary threats are:

Peer entity authentication;

Data origin authentication;

Connectionless integrity.

Peer entity authentication is to occur during the establishment of an application association. If the association is successfully established, the underlying security mechanism provides information that is subsequently used in data origin authentication. There the information may be included in or, in some other way, transform the data units of subsequent exchanges so that they can be identified as originating from an authenticated entity. Both authentication security services are to be provided at the application level of the protocol.

Connectionless integrity insures that data units originating from an authenticated source are

not modifiable without detection. When combined with a strong data origin authentication mechanism, the ability to fabricate new data units is also countered. Connectionless integrity may be provided at either the application level of the protocol or within one of the lower levels of the protocol (i.e., transport or network).

Enhanced Security Services

The security services required to counter secondary threats are:

All basic security services with the possible exception of connectionless integrity;

Connectionless confidentiality;

Connection integrity with or without recovery.

Both connectionless confidentiality and connection integrity may be provided at either the application level of protocol or within one of the lower levels of protocol. The latter provision is assumed here. Enhanced security services are not discussed further in this note, but to be issued as a requirement for the lower layer protocol and service standards, and according to functional standards to be developed.

Security Mechanisms

Peer Entity Authentication

Peer Entity Authentication will use the ACSE authentication mechanism and associated data types as defined in clause 9 of this Part of the IAs. The specific authentication mechanism to be supported is the Simple-Strong Authentication defined in 9.1.1.1.

Support of ACSE authentication is optional.

Connectionless Integrity

Editor's Note - Proposed text for this clause appears in WIA Part 12, clause 12.3.2.

Annex (normative)

ISPICS Requirements List

Annex (normative)

Errata

Table B.1 - SIA Part 12 changes

NO. OF ERRATA	TYPE	REFERENCED DOCUMENT	CLAUSE	NOTES

Annex (normative)

TBD

Annex (informative)

Security Algorithms and Attributes

```
OIWSECSIGAlgorithmObjectIdentifiers  {i(1) identified-organization(3)
                            oiw(14) secsig(3)
                            oIWSECSIGAlgorithmObjectIdentifiers(1)}
DEFINITIONS =
BEGIN

EXPORTS
-- to be determined

IMPORTS
-- none

-- category of information object
-- defining our own here; perhaps the definition should be imported from
-- {joint-iso-ccitt ds(5) modules(1) usefulDefinitions(0)}

algorithm       OBJECT IDENTIFIER ::= {iso(1) identified-organization(3)
                                oiw(14) secsig(3) algorithm(2)}

-- macros

-- taken from {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7)}
ALGORITHM MACRO  ::=
BEGIN
TYPE NOTATION        ::= "PARAMETER" type
VALUE NOTATION       ::= value(VALUE OBJECT IDENTIFIER)
END -- of ALGORITHM

-- algorithms

md4WithRSA ALGORITHM
  PARAMETER NULL
  ::= {algorithm 2}

md5WithRSA ALGORITHM
  PARAMETER NULL
  ::= {algorithm 3}

md4WithRSAEncryption ALGORITHM
  PARAMETER NULL
  ::= {algorithm 4}

desECB ALGORITHM
  PARAMETER NULL
  ::= {algorithm 6}
```

```
desCBC ALGORITHM
        PARAMETER CBCParameter
        ::= {algorithm 7}

CBCParameter ::= IV

desOFB ALGORITHM
        PARAMETER FBParameter
        ::= {algorithm 8}

desCFB ALGORITHM
        PARAMETER FBParameter
        ::= {algorithm 9}

FBParameter ::= SEQUENCE {
        iv IV,
        numberOfBits NumberOfBits
        }

NumberOfBits ::= INTEGER    -- Number of feedback bits (1 to 64 bits)
```

Editor's Note - Check FIPS PUB 81 for allowed ranges of feedback bits and specify ranges here as a comment.

```
        IV ::= OCTET STRING  -- 8 octets

desMAC ALGORITHM
        PARAMETER MACParameter
        ::= {algorithm 10}

MACParameter ::= INTEGER    -- Length of MAC (16, 24, 32, 40, 40 or 64 bits)
```

Editor's Note - Check FIPS PUB 113 for allowed

```
rsaSignature ALGORITHM
        PARAMETER NULL
        ::= { algorithm 11 }

dsa ALGORITHM
        PARAMETER DSAParameters
        ::= { algorithm 12 }

dsaWithSHA ALGORITHM
        PARAMETER DSAParameters
        ::= { algorithm 13}

shaWithRSASignature
        PARAMETER NULL
        ::= { algorithm 15 }

dhWithCommonModulus ALGORITHM
```

```
        PARAMETER NULL
        ::= { algorithm 16 }

desEDE ALGORITHM
        PARAMETER NULL
        ::= { algorithm 17 }
sha ALGORITHM
        PARAMETER NULL
        ::= { algorithm 18 }

keyed-hash-seal ALGORITHM
        PARAMETER NULL
        ::= { algorithm 23 }

END -- of Algorithm Object Identifier Definitions
```

Annex (normative)

References for Security Algorithms

[a] Kaliski, B., The MD2 Message-Digest Algorithm, Internet Draft draft-rsadsi-kaliski-md2-00.txt, July 1, 1991.

[b] Rivest, R. and S. Dusse, The MD4 Message-Digest Algorithm, Internet Draft draft-rsadsi-rivest-md4-00.txt, July 1, 1991.

[c] Rivest, R. and S. Dusse, The MD5 Message-Digest Algorithm, Internet Draft draft-rsadsi-rivest-md5-01.txt, July 10, 1991.

[d] Rivest, R. L., A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM, Volume 21, Number 2, February 1978, pp. 120-126.

[e] Rivest, Ronald L., Adi Shamir and Leonard M. Adleman, Cryptographic Communications System and Method, United States Patent No. 4,405,829, September 20, 1983.

[f] Fougner, R.B., Public Key Standards and Licenses, Internet Request for Comments (RFC) 1170, January 1991.

[g] RSA Data Security, Inc., PKCS #1: RSA Encryption Standard, Version 1.4, June 3, 1991.

[h] Diffie, W., and M.E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory, IT-22, pp. 644-654, 1976.

[i] Hellman, Martin E., Bailey W. Diffie and Ralph C. Merkle, Cryptographic Apparatus and Method, United States Patent No. 4,200,770, April 29, 1980.

[j] RSA Data Security, Inc., PKCS #3: Diffie-Hellman Key-Agreement Standard, Version 1.3, June 3, 1991.

[k] ElGamal, T., A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Transactions on Information Theory, IT-31, Number 4, July 1985, pp. 469-472.

[l] Federal Information Processing Standards Publication (FIPS PUB) 46-1, Data Encryption Standard, U.S. Department of Commerce/National Bureau of Standards, Supersedes FIPS PUB 46, January 15, 1977, Reaffirmed January 22, 1988.

[m] ANSI X3.92-1981, Data Encryption Algorithm, American National Standards Institute, Approved December 30, 1980.

[n] Federal Information Processing Standards Publication (FIPS PUB) 81, DES Modes of Operation, U.S. Department of Commerce/National Bureau of Standards, December 2, 1980.

[o] ANSI X3.106-1983, Data Encryption Algorithm - Modes of Operation, American National Standards Institute, Approved May 16, 1983.

[p] Federal Information Processing Standards Publication (FIPS PUB) 74, Guidelines for Implementing and Using the NBS Data Encryption Standard, U.S. Department of Commerce/National Bureau of Standards, April 1, 1981.

[q] Gait, Jason, Validating the Correctness of Hardware Implementations of the NBS Data Encryption Standard, Special Publication 500-20, U.S. Department of Commerce/National Bureau of Standards, Issued November 1977, Revised September 1980.

[r] Gait, Jason, Maintenance Testing for the Data Encryption Standard, Special Publication 500-61, U.S. Department of Commerce/National Bureau of Standards, August 1980.

[s] Federal Information Processing Standards Publication (FIPS PUB) 113, Computer Data Authentication, U.S. Department of Commerce/National Bureau of Standards, May 30, 1985.

[t] American National Standard X9.9-1986, Financial Institution Message Authentication (Wholesale), American Bankers Association, April 7, 1986.

[u] Linn, John, Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures, Internet Draft draft-ietf-pem-msgproc-01.txt, September 1991.

[v] Kent, Steve, Privacy Enhancement for Internet Electronic Mail: Part II -- Certificate-Based Key Management, Internet Draft draft-ietf-pem-keymgmt-00.txt, June 1991.

[w] Balenson, David. M, Privacy Enhancement for Internet Electronic Mail: Part III -- Algorithms, Modes, and Identifiers, Internet Draft draft-ietf-pem-algorithms-00.txt, August 1991.

[x] Kaliski, Burton. S, Privacy Enhancement for Internet Electronic Mail: Part IV -- Notary, Co-Issuer, CRL-Storing and CRL-Retrieving Services, Internet Draft draft-ietf-pem-notary-00.txt, July 1991.

[y] North American Directory Forum, A Naming Scheme for c=US, Request for Comments 1255, September 1991.

[z] Kohl, John and B. Clifford Neuman, The Kerberos Network Authentication Service, Internet Draft cat-kerberos-00.txt, June 1991.

[aa] Proposed FIPS xx, Digital Signature Standard, U.S. Dept. of Commerce/National Institute of Standards and Technology, 1992. Also published as ANS X9.30-199x, *Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry*, Part 1: *The Digital Signature Algorithm (DSA)*.

[ab] Proposed FIPS xx, Secure Hash Standard, U.S. Dept. of Commerce/National Institute of Standards and Technology, 1992. Also published as ANS X9.30-199x, *Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry*, Part 1: *The Secure Hash Algorithm (SHA)*.

[ac] ANS X9.31-199x, *Public Key Cryptography Using Reversible Algorithms for the*

Financial Services Industry, Part 2: *Hash Algorithms*.

[ad] ANS X9.31-199x, *Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry*, Part 1: *The RSA Signature Algorithm* .

[ae] ISO/IEC IS 9796, *Digital Signature Scheme Giving Message Recovery*, 1991.

[af] ANS X9.17-1985, *Financial Institution Key Management (Wholesale)*, American Bankers Association, April 4, 1985, Section 7.2.

[ag] D. Coppersmith, *Analysis of ISO/CCITT Document X.509 Annex D*, IBM Research Division, Yorktown Heights, June 1989.

[ah] J. Moore, "Protocol Failures in Cryptosystems," *Proceedings of the IEEE*, vol. 76, no. 5, pp. 594-601, May 1988.

[ai] Miller,S.P., B.C. Neuman, J.I. Schiller, and J.H. Saltzer, "Project Athena Technical Plan Section E.2.1: Kerberos Authentication and Authorization System," Project Athena, MIT, December 1987.

Annex (informative)

Bibliography

[**1**] ISO/IEC JTC1 SC21 N3614 Information Retrieval, Transfer, and Management for OSI

[**2**] ISO/IEC DP 9796 Data Cryptographic Techniques

[**3**] Secure Data Network System (SDNS): Key Management Profile - Communications Protocol Requirements (SDN-601/NIST IR 90-4262)

[**4**] SDNS: Message Security Protocol (SDN-701/NIST IR 90-4250)

[**5**] SDNS: Directory (SDN-702/NIST IR 90-4250)

[**6**] ISO/IEC JTC1 SC21/WG1 N5002 Security ASE

[**7**] Access Control Information Specification (ACIS)

[**8**] SDNS: Key Management Protocol - Definition of Services Provided (SDN-902/NIST IR 90-4262)

[**9**] SDNS: Key Management Protocol - Specification of the Protocol (SDN-903/NIST IR 90-4262)

[**10**] ISO/IEC JTC1 SC21/WG1 N4110 Authentication ASE Exchange

[**11**] SDNS: Security Protocol 3 (SDN-301/NIST IR 90-4250)

[**12**] SDNS: Security Protocol 4 (SDN-401/NIST IR 90-4250)

[**13**] SDNS: Key Management Protocol - SDNS Traffic Key (SDN-906/NIST IR 90-4262)

[**14**] ISO/IEC JTC1 SC21/WG1 N5001 Upper Layers Security Model

[**15**] ISO/IEC JTC1 SC21/WG1 F29 N5045 Access Control Framework

[**16**] ISO/IEC JTC1 SC21/WG1 F30 Authentication Framework

[**17**] ISO/IEC JTC1 SC21/WG1 F31 N5047 Integrity Framework

[**18**] ISO/IEC JTC1 SC21/WG1 F32 N5046 Non-Repudiation

[**19**] ISO/IEC JTC1 SC21/WG4 N3775 Security Audit Trail

[**20**] ISO/IEC JTC1 SC21/WG1 N4110 Authentication ASE Exchange

[**21**] ISO/IEC JTC1 SC21/WG7 N4022 Key Management Framework

[**22**] ISO/IEC JTC1 SC21/WG1 N5048 Confidentiality Framework

PART 12 - SECURITY December 1993 (Stable)

[**23**] ISO/IEC JTC1 SC21/WG1 N5049 Guide to OSI Security Standards

[**24**] ISO/IEC JTC1 SC21/WG1 N5044 Security Framework Overview

[**25**] RFC-1113, Privacy Enhancement for Internet Electronic Mail: Part I - Message Encipherment and Authentication Procedures.

[**26**] RFC-1114, Privacy Enhancement for Internet Electronic Mail: Part II - Certificate-Based Key Management.

[**27**] RFC-1115, Privacy Enhancement for Internet Electronic Mail: Part III - Algorithms, Modes, and Identifiers (August 1989).

[**28**] Network Layer ISO/IEC JTC1 SC6

[**29**] Transport Layer ISO/IEC JTC1 SC6 6285

[**30**] Lower Layer ISO/IEC JTC1 SC6 6227

[**31**] ANSI X9.9 DES Encryption Algorithum

Annex (informative)

ElGamal

The information in this subclause includes a tutorial description of the ElGamal scheme for digital signature using the notation defined in the Directory Documents, [ISO9594-8]. It is intended that much of the tutorial information provided in this subclause will be moved to the security agreements sometime in the future.

Background

The ElGamal digital signature scheme is based on earlier work done by Diffie and Hellman [b] in which it was suggested that a likely candidate for a one-way function is the *discrete exponential function*

(1)

where x is an integer between 1 and p-1 inclusive, where p is a very large prime number, and where α is an integer such that $1 \leq \alpha \geq p$ and $\{\alpha \bmod p, \alpha^2 \bmod p, ..., \alpha^{p-1} \bmod p\}$ is equal to the set {1, 2, ..., p-1}. In algebraic terminology, such an α is called a *primitive element*. References on the topic of primitive roots and elements are [aa] and [ab].

Now, in the real number system, if $y = \alpha^x$, then by definition of the logarithm we can solve for x using $x = \log_\alpha(y)$. The same idea extends to solving eq (1) for x so that inverting $f(x)$ requires calculating *discrete logarithms*. The reason Diffie and Hellman suspected eq (1) is one-way is that for suitable p, it is computationally difficult to invert $f(x)$. According to the current state of the art, computing discrete logs for suitable p has been found to require a number of operations roughly equivalent to

(2)

where b is the number of bits in p, and c is estimated at $c = .69$ according to [ac]. This can be compared to only about $2 \log_2 p$ multiplications for discrete exponentiation. If in fact the best known algorithm for computing discrete logs is near optimal then Expression (2)is a good measure of the problem's complexity (for a properly chosen p) and the discrete exponential function has all the qualities of a one-way function as described by Diffie and Hellman.

Digital Signature

Private Key: X_s denotes the private key for user X. X_s is a randomly chosen integer which user X keeps secret.

Public Key: X_p denotes the public key for user X and is calculated using the corresponding private key such that

(3)

where

p is a prime satisfying the requirements listed in 12.2.2.4.

α is a primitive element mod p.

Note that p and α could be used globally, but because they should be easily changeable (see 12.2.2.4 for information about why these two parameters should be easily changeable) it would probably be preferable for each user to choose his/her own p and α. If users choose their own, then p and α must be made available to the recipient for use in the signature verification process.

Signing Procedure: Suppose user A wants to sign a message intended for recipient B. The basic idea is to compute a two part signature (r, s) for the message m such that

(4)

where h is a one-way hash function.

Compute the signature (r, s) as follows.

Choose a random number k, uniformly between 0 and p-1 such that k and p-1 have no common divisor except 1 (i.e., gcd(k,p-1)=1).

Compute r such that

(5)

Use r to solve for the corresponding s as follows.

rewrite eq (4) using eq (5) and the definition of the public key to get

(6)

Combining exponents, get

(7)

eq (7) implies that

(8)

Note that eq (8) has a single solution for s because k was chosen such that gcd(k,p-1)=1. See [ad] for supporting theorem.

now solve for s and get

(9)

where I is computed such that $k * I \equiv 1 \pmod{p\text{-}1}$.

The ElGamal signature is comparable in size to the corresponding RSA signature.

Verification

The recipient receives *Ap*, *m*, *r*, *s*, α, and *p* and computes both sides of eq (4) and then compares the results.

Known Constraints on Parameters

The following list of constraints is the result of a search of current literature and may not be complete:

p must be prime;

p must be large.

Note that Expression (2) can be used to speculate on the level of security afforded by crypto systems based on the discrete log problem. Breaking the ElGamal scheme has not been proven to be equivalent to finding discrete logs, but if we assume equivalence then we can estimate how large p should be for a desired level of security.

For instance,suppose we wanted to use Expression (2) to decide how large *p* should be so that we can be reasonably sure the system cannot be broken (using the best *known* algorithm) in a practical amount of time. To be on the conservative side, we decide we want to protect against a special purpose machine that can perform 10^{15} operations per second. Specifically, we want to know how large *p* should be so that such a machine would take at least one year to break the system.

In one year, the hypothetical machine can perform 3×10^{22} operations. To find the size of the desired *p*, solve the following equation for *b*.

(10)

We get . This is the number of bits in the desired *p*. So, the magnitude of the desired *p* is about 2^{606} which is roughly 266×10^{180}.

Hence, to be reasonably sure of attaining the desired level of security, we find a prime number greater than 266×10^{180} which satisfies all the other criteria listed in this subclause. Our confidence, however, is strictly based on the assumption that breaking ElGamal is as difficult as finding discrete logs and the assumption that the best known algorithm for finding discrete logs is near optimal.

p should occasionally be changed. This requirement is discussed in [ae] and is related to the discovery of new algorithms for computing discrete logarithms in *GF*(*p*).

p-1 must have at least one large prime factor. This requirement is discussed in [ae] and is imposed by the Silverman-Pohlig-Hellman algorithm p which computes discrete logarithms in *GF*(*p*) using on the order operations and a comparable amount of storage, where *r* is the largest prime factor in *p*-1.

p should not be the square of any prime. A subexponential-time algorithm for computing discrete logarithms in $GF(p^2)$ has been found. See [af]for details.