Network Working Group
Request for Comments: 3163
Category: Experimental

R. Zuccherato
Entrust Technologies
M. Nystrom
RSA Security

August 2001

ISO/IEC 9798-3 Authentication SASL Mechanism

Status of this Memo

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

Copyright Notice

IESG Note

It is the opinion of the Security Area Directors that this document defines a mechanism to use a complex system (namely PKI certificates) for authentication, but then intentionally discards the key benefits (namely integrity on each transmission). Put another way, it has all of the pain of implementing a PKI and none of the benefits. We should not support it in use in Internet protocols.

The same effect, with the benefits of PKI, can be had by using TLS/SSL, an existing already standards track protocol.

Abstract

This document defines a SASL (Simple Authentication and Security Layer) authentication mechanism based on ISO/IEC 9798-3 and FIPS PUB 196 entity authentication.

1. Introduction

1.1. Overview

This document defines a SASL [RFC2222] authentication mechanism based on ISO/IEC 9798-3 [ISO3] and FIPS PUB 196 [FIPS] entity authentication.

This mechanism only provides authentication using X.509 certificates [X509]. It has no effect on the protocol encodings and does not provide integrity or confidentiality services.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The key benefit of asymmetric (public key) security, is that the secret (private key) only needs to be placed with the entity that is being authenticated. Thus, a private key can be issued to a client, which can then be authenticated by ANY server based on a token generated by the client and the generally available public key. Symmetric authentication mechanisms (password mechanisms such as CRAM-MD5 [RFC2195]) require a shared secret, and the need to maintain it at both endpoints. This means that a secret key for the client needs to be maintained at every server that may need to authenticate the client.

The service described in this memo provides authentication only. There are a number of places where an authentication only service is useful, e.g., where confidentiality and integrity are provided by lower layers, or where confidentiality or integrity services are provided by the application.

1.2. Relationship to TLS

The functionality defined here can be provided by TLS, and it is important to consider why it is useful to have it in both places. There are several reasons for this, e.g.:

- Simplicity. This mechanism is simpler than TLS. If there is only a requirement for this functionality (as distinct from all of TLS), this simplicity will facilitate deployment.

- Layering. The SASL mechanism to establish authentication works cleanly with most protocols. This mechanism can fit more cleanly than TLS for some protocols.

- Proxies. In some architectures the endpoint of the TLS session may not be the application endpoint. In these situations, this mechanism can be used to obtain end-to-end authentication.

- Upgrade of authentication. In some applications it may not be clear at the time of TLS session negotiation what type of authentication may be required (e.g., anonymous, server, client-server). This mechanism allows the negotiation of an anonymous or server authenticated TLS session which can, at a later time, be upgraded to provide the desired level of authentication.

2. Description of Mechanism

2.1. Scope

The mechanism described in this memo provides either mutual or unilateral entity authentication as defined in ISO/IEC 9798-1 [ISO1] using an asymmetric (public-key) digital signature mechanism.

2.2. Authentication modes

This SASL mechanism contains two authentication modes:

- Unilateral client authentication: The client digitally signs a challenge from the server, thus authenticating itself to the server.

- Mutual authentication: The client digitally signs a challenge from the server and the server digitally signs a challenge from the client. Thus both the client and server authenticate each other.

2.3. SASL key

This mechanism has two SASL keys corresponding to the two different modes:

- "9798-U-<algorithm>" for unilateral client authentication.

- "9798-M-<algorithm>" for mutual authentication.

Each SASL key may be used with a list of algorithms. A list of supported algorithms is given in Section 4.

2.4. Unilateral Client Authentication

This section gives a brief description of the steps that are performed for unilateral client authentication. The actual data structures are described fully in Section 3.

a) The server generates a random challenge value R_B and sends it to the client.

b) The client generates a random value R_A and creates a token TokenAB. The token contains R_A, the client's certificate and also a digital signature created by the client over both R_A and R_B. Optionally, it also contains an identifier for the server.

c) The client sends the token to the server.

d) The server verifies the token by:

- verifying the client's signature in TokenAB (this includes full certificate path processing as described in [RFC2459]),

- verifying that the random number R_B, sent to the client in Step 1, agrees with the random number contained in the signed data of TokenAB, and

- verifying that the identifier for the server, if present, matches the server's distinguishing identifier.

2.5. Mutual Authentication

This section gives a brief description of the steps that are performed for mutual authentication. The actual data structures are described fully in Section 3.

a) The server generates a random challenge value R_B and sends it to the client.

b) The client generates a random value R_A and creates a token TokenAB. The token contains R_A, the client's certificate and also a digital signature created by the client over both R_A and R_B. Optionally, it also contains an identifier for the server.

c) The client sends the token to the server.

d) The server verifies the token by:

- verifying the client's signature in TokenAB (this includes full certificate path processing as described in [RFC2459]),

- verifying that the random number R_B, sent to the client in Step 1, agrees with the random number contained in the signed data of TokenAB, and

- verifying that the identifier for the server, if present, matches the server's distinguishing identifier.

e) The server creates a token TokenBA. The token contains a third random value R_C, the server's certificate and a digital signature created by the server over R_A, R_B and R_C. Optionally, it also contains an identifier for the client.

f) The server sends the token to the client.

g) The client verifies the token by:

- verifying the server's signature in TokenBA (this includes full certificate path processing as described in [RFC2459]),

- verifying that the random number R_B, received by the client in Step 1, agrees with the random number contained in the signed data of TokenBA,

- verifying that the random number R_A, sent to the server in Step 2, agrees with the random number contained in the signed data of Token BA and

- verifying that the identifier for the client, if present, matches the client's distinguishing identifier.

3. Token and Message Definition

Note - Protocol data units (PDUs) SHALL be DER-encoded [X690] before transmitted.

3.1. The "TokenBA1" PDU

TokenBA1 is used in both the unilateral client authentication and mutual authentication modes and is sent by the server to the client.

TokenBA1 contains a random value, and, optionally, the servers name and certificate information.

```
TokenBA1 ::= SEQUENCE {
     randomB   RandomNumber,
     entityB   [0] GeneralNames OPTIONAL,
     certPref  [1] SEQUENCE SIZE (1..MAX) OF TrustedAuth OPTIONAL
}
```

3.2. The "TokenAB" PDU

TokenAB is used in the unilateral client authentication and mutual authentication modes and is sent by the client to the server. TokenAB contains a random number, entity B's name (optionally), entity certification information, an (optional) authorization identity, and a signature of a DER-encoded value of type TBSDataAB. The certA field is used to send the client's X.509 certificate (or a URL to it) and a related certificate chain to the server.

The authID field is to be used when the identity to be used for access control is different than the identity contained in the certificate of the signer. If this field is not present, then the identity from the client's X.509 certificate shall be used.

```
TokenAB ::= SEQUENCE {
     randomA   RandomNumber,
     entityB   [0] GeneralNames OPTIONAL,
     certA     [1] CertData,
     authID    [2] GeneralNames OPTIONAL,
     signature SIGNATURE { TBSDataAB }

}(CONSTRAINED BY {-- The entityB and authID fields shall be included
  -- in TokenAB if and only if they are also included in TBSDataAB.
  -- The entityB field SHOULD be present in TokenAB whenever the
  -- client believes it knows the identity of the server.--})

TBSDataAB ::= SEQUENCE {
     randomA RandomNumber,
     randomB RandomNumber,
     entityB [0] GeneralNames OPTIONAL,
     authID  [1] GeneralNames OPTIONAL
}
```

3.3. The "TokenBA2" PDU

TokenBA2 is used in the mutual authentication mode and is sent by the server to the client. TokenBA2 contains a random number, entity A's name (optionally), certification information, and a signature of a DER-encoded value of type TBSDataBA. The certB field is to be used to send the server's X.509 certificate and a related certificate chain to the client.

```
TokenBA2 ::= SEQUENCE {
     randomC   RandomNumber,
     entityA   [0] GeneralNames OPTIONAL,
     certB     [1] CertData,
     signature SIGNATURE { TBSDataBA }
}(CONSTRAINED BY {-- The entityA field shall be included in TokenBA2
  -- if and only if it is also included in TBSDataBA.  The entityA
  -- field SHOULD be present and MUST contain the client's name
  -- from their X.509 certificate.--})

TBSDataBA ::= SEQUENCE {
     randomB RandomNumber,
     randomA RandomNumber,
     randomC RandomNumber,
     entityA GeneralNames OPTIONAL
}
```

3.4. The "TrustedAuth" type

```
TrustedAuth ::= CHOICE {
     authorityName         [0] Name,
          -- SubjectName from CA certificate
     issuerNameHash        [1] OCTET STRING,
          -- SHA-1 hash of Authority's DN
     issuerKeyHash         [2] OCTET STRING,
          -- SHA-1 hash of Authority's public key
     authorityCertificate  [3] Certificate,
          -- CA certificate
     pkcs15KeyHash         [4] OCTET STRING
          -- PKCS #15 key hash
}
```

The TrustedAuth type can be used by a server in its initial message ("TokenBA1") to indicate to a client preferred certificates/public key pairs to use in the authentication.

A trusted authority is identified by its name, hash of its name, hash of its public key, its certificate, or PKCS #15 key hash. If identified by its name, then the authorityName field in TrustedAuth contains the SubjectName of its CA certificate. If it is identified by the hash of its name then the issuerNameHash field contains the SHA-1 hash of the DER encoding of SubjectName from its CA certificate. If it is identified by the hash of its public key then the issuerKeyHash field contains the SHA-1 hash of the authority's public key. The hash shall be calculated over the value (excluding tag and length) of the subject public key field in the issuer's certificate. If it is identified by its certificate then the authorityCertificate field contains its CA certificate. If it is

identified by the PKCS #15 key hash then the pkcs15KeyHash field contains the hash of the CA's public key as defined in PKCS #15 [PKCS15] Section 6.1.4.

3.5. The "CertData" type

The certification data is a choice between a set of certificates and a certificate URL.

The certificate set alternative is as in [RFC2630], meaning it is intended that the set be sufficient to contain chains from a recognized "root" or "top-level certification authority" to all of the sender certificates with which the set is associated. However, there may be more certificates than necessary, or there may be fewer than necessary.

Note - The precise meaning of a "chain" is outside the scope of this document. Some applications may impose upper limits on the length of a chain; others may enforce certain relationships between the subjects and issuers of certificates within a chain.

When the certURL type is used to specify the location at which the user's certificate can be found, it MUST be a non-relative URL, and MUST follow the URL syntax and encoding rules specified in [RFC1738]. The URL must include both a scheme (e.g., "http" or "ldap") and a scheme-specific part. The scheme-specific part must include a fully qualified domain name or IP address as the host.

```
CertData ::= CHOICE {
     certificateSet     SET SIZE (1..MAX) OF Certificate,
     certURL            IA5String,
     ... -- For future extensions
}
```

3.6. The "RandomNumber" type

A random number is simply defined as an octet string, at least 8 bytes long.

```
RandomNumber ::= OCTET STRING (SIZE(8..MAX))
```

3.7. The "SIGNATURE" type

This is similar to the "SIGNED" parameterized type defined in [RFC2459], the difference being that the "SIGNATURE" type does not include the data to be signed.

```
SIGNATURE { ToBeSigned } ::= SEQUENCE {
     algorithm AlgorithmIdentifier,
     signature BIT STRING
}(CONSTRAINED BY {-- Must be the result of applying the signing
  -- operation indicated in "algorithm" to the DER-encoded octets of
  -- a value of type -- ToBeSigned })
```

3.8. Other types

The "GeneralNames" type is defined in [RFC2459].

4. Supported Algorithms

The following signature algorithms are recognized for use with this mechanism, and identified by a key. Each key would be combined to make two possible SASL mechanisms. For example the DSA-SHA1 algorithm would give 9798-U-DSA-SHA1, and 9798-M-DSA-SHA1. All algorithm names are constrained to 13 characters, to keep within the total SASL limit of 20 characters.

The following table gives a list of algorithm keys, noting the object identifier and the body that assigned the identifier.

Key	Object Id	Body
RSA-SHA1-ENC	1.2.840.113549.1.1.5	RSA
DSA-SHA1	1.2.840.10040.4.3	ANSI
ECDSA-SHA1	1.2.840.10045.4.1	ANSI

Support of the RSA-SHA1-ENC algorithm is RECOMMENDED for use with this mechanism.

5. Examples

5.1. IMAP4 example

The following example shows the use of the ISO/IEC 9798-3 Authentication SASL mechanism with IMAP4 [RFC2060].

The base64 encoding of challenges and responses, as well as the "+ " preceding the responses are part of the IMAP4 profile, not part of this specification itself (note that the line breaks in the sample authenticators are for editorial clarity and are not in real authenticators).

```
S: * OK IMAP4 server ready
C: A001 AUTHENTICATE 9798-U-RSA-SHA1
S: + MAoECBI4l1h5h0eY
C: MIIBAgQIIxh5I0h5RYegD4INc2FzbC1yLXVzLmNvbaFPFk1odHRwOi8vY2VydHMt
   ci11cy5jb20vY2VydD9paD1odmNOQVFFRkJRQURnWUVBZ2hBR2hZVFJna0ZqJnNu
   PUVQOXVFbFkzS0RlZ2pscjCBkzANBgkqhkiG9w0BAQUFAAOBgQCkuC2GgtYcxGG1
   NEzLA4bh5lqJGOZySACMmc+mDrV7A7KAgbpO2OuZpMCl7zvNt/L3OjQZatiX8d1X
   buQ40l+g2TJzJt06o7ogomxdDwqlA/3zp2WMohlI0MotHmfDSWEDZmEYDEA3/eGg
   kWyi1v1lEVdFuYmrTr8E4wE9hxdQrA==
S: A001 OK Welcome, 9798-U-RSA-SHA1 authenticated user: Magnus
```

6. IANA Considerations

By registering the 9798-<U/M>-<algorithm> protocols as SASL mechanisms, implementers will have a well-defined way of adding this authentication mechanism to their product. Here is the registration template for the SASL mechanisms defined in this memo:

SASL mechanism names:	9798-U-RSA-SHA1-ENC 9798-M-RSA-SHA1-ENC 9798-U-DSA-SHA1 9798-M-DSA-SHA1 9798-U-ECDSA-SHA1 9798-M-ECDSA-SHA1 ; For a definition of the algorithms see Section 4 of this memo.
Security Considerations:	See Section 7 of this memo
Published specification:	This memo
Person & email address to contact for further information:	See Section 9 of this memo.
Intended usage:	COMMON
Author/Change controller:	See Section 9 of this memo.

7. Security Considerations

The mechanisms described in this memo only provides protection against passive eavesdropping attacks. They do not provide session privacy or protection from active attacks. In particular, man-in-the-middle attacks aimed at session "hi-jacking" are possible.

The random numbers used in this protocol MUST be generated by a cryptographically strong random number generator. If the number is chosen from a small set or is otherwise predictable by a third party, then this mechanism can be attacked.

The inclusion of the random number R_A in the signed part of TokenAB prevents the server from obtaining the signature of the client on data chosen by the server prior to the start of the authentication mechanism. This measure may be required, for example, when the same key is used by the client for purposes other than entity authentication. However, the inclusion of R_B in TokenBA2, whilst necessary for security reasons which dictate that the client should check that it is the same as the value sent in the first message, may not offer the same protection to the server, since R_B is known to the client before R_A is chosen. For this reason a third random number, R_C, is included in the TokenBA2 PDU.

8. Bibliography

[FIPS] FIPS 196, "Entity authentication using public key cryptography," Federal Information Processing Standards Publication 196, U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, Virginia, 1997.

[ISO1] ISO/IEC 9798-1: 1997, Information technology - Security techniques - Entity authentication - Part 1: General.

[ISO3] ISO/IEC 9798-3: 1997, Information technology - Security techniques - Entity authentication - Part 3: Mechanisms using digital signature techniques.

[PKCS15] RSA Laboratories, "The Public-Key Cryptography Standards - PKCS #15 v1.1: Cryptographic token information syntax standard", June 6, 2000.

[RFC1738] Berners-Lee, T., Masinter L. and M. McCahill "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[RFC2060] Crispin, M., "Internet Message Access Protocol - Version 4rev1", RFC 2060, December 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2195] Klensin, J., Catoe, R. and P. Krumviede "IMAP/POP AUTHorize Extension for Simple Challenge/Response", RFC 2195, September 1997.

[RFC2222] J. Meyers, "Simple Authentication and Security Layer", RFC 2222, October 1997.

[RFC2459] Housley, R., Ford, W., Polk, W. and D. Solo "Internet X.509 Public Key Infrastructure: X.509 Certificate and CRL Profile", RFC 2459, January 1999.

[RFC2630] R. Housley, "Cryptographic Message Syntax", RFC 2630, June 1999.

[X509] ITU-T Recommendation X.509 (1997) | ISO/IEC 9594-8:1998, Information Technology - Open Systems Interconnection - The Directory: Authentication Framework.

[X690] ITU-T Recommendation X.690 (1997) | ISO/IEC 8825-1:1998, Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

9. Authors' Addresses

Robert Zuccherato
Entrust Technologies
1000 Innovation Drive
Ottawa, Ontario
Canada K2K 3E7

Phone: +1 613 247 2598
EMail: robert.zuccherato@entrust.com

Magnus Nystrom
RSA Security
Box 10704
121 29 Stockholm
Sweden

Phone: +46 8 725 0900
EMail: magnus@rsasecurity.com

APPENDICES

A. ASN.1 modules

A.1. 1988 ASN.1 module

```
   SASL-9798-3-1988

   DEFINITIONS IMPLICIT TAGS ::=

   BEGIN

   -- EXPORTS ALL --

   IMPORTS

   Name, AlgorithmIdentifier, Certificate
        FROM PKIX1Explicit88 {iso(1) identified-organization(3) dod(6)
        internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
        id-pkix1-explicit-88(1)}

   GeneralNames
        FROM PKIX1Implicit88 {iso(1) identified-organization(3) dod(6)
        internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
        id-pkix1-implicit-88(2)};

   TokenBA1 ::= SEQUENCE {
        randomB   RandomNumber,
        entityB   [0] GeneralNames OPTIONAL,
        certPref  [1] SEQUENCE SIZE (1..MAX) OF TrustedAuth OPTIONAL
   }

   TokenAB ::= SEQUENCE {
        randomA   RandomNumber,
        entityB   [0] GeneralNames OPTIONAL,
        certA     [1] CertData,
        authID    [2] GeneralNames OPTIONAL,
        signature SEQUENCE {
             algorithm AlgorithmIdentifier,
             signature BIT STRING
       }
   } -- The entityB and authID fields shall be included in TokenAB
     -- if and only if they are also included in TBSDataAB.  The entityB
     -- field SHOULD be present in TokenAB whenever the client
     -- believes it knows the identity of the server.
     -- The signature operation shall be done on a
     -- DER-encoded value of type TBSDataAB.
```

```
   TBSDataAB ::= SEQUENCE {
        randomA RandomNumber,
        randomB RandomNumber,
        entityB [0] GeneralNames OPTIONAL,
        authID  [1] GeneralNames OPTIONAL
   }

   TokenBA2 ::= SEQUENCE {
        randomC   RandomNumber,
        entityA   [0] GeneralNames OPTIONAL,
        certB     [1] CertData,
        signature SEQUENCE {
             algorithm AlgorithmIdentifier,
             signature BIT STRING
        }
   } -- The entityA field shall be included in TokenBA2
     -- if and only if it is also included in TBSDataBA.  The entityA
     -- field SHOULD be present and MUST contain the client's name
     -- from their X.509 certificate.  The signature shall be done
     -- on a DER-encoded value of type TBSDataBA.

   TBSDataBA ::= SEQUENCE {
        randomB RandomNumber,
        randomA RandomNumber,
        randomC RandomNumber,
        entityA GeneralNames OPTIONAL
   }

   TrustedAuth ::= CHOICE {
        authorityName         [0] Name,
             -- SubjectName from CA certificate
        issuerNameHash        [1] OCTET STRING,
             -- SHA-1 hash of Authority's DN
        issuerKeyHash         [2] OCTET STRING,
             -- SHA-1 hash of Authority's public key
        authorityCertificate  [3] Certificate,
             -- CA certificate
        pkcs15KeyHash         [4] OCTET STRING
             -- PKCS #15 key hash
   }

   CertData ::= CHOICE {
        certificateSet     SET SIZE (1..MAX) OF Certificate,
        certURL            IA5String
   }

   RandomNumber ::= OCTET STRING (SIZE(8..MAX))
```

```
   END
```

A.2. 1997 ASN.1 module

```
   SASL-9798-3-1997

   DEFINITIONS IMPLICIT TAGS ::=

   BEGIN

   -- EXPORTS ALL --

   IMPORTS

   AlgorithmIdentifier, Name, Certificate
        FROM PKIX1Explicit93 {iso(1) identified-organization(3) dod(6)
        internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
        id-pkix1-explicit-93(3)}

   GeneralNames
        FROM PKIX1Implicit93 {iso(1) identified-organization(3) dod(6)
        internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
        id-pkix1-implicit-93(4)};

   TokenBA1 ::= SEQUENCE {
        randomB   RandomNumber,
        entityB   [0] GeneralNames OPTIONAL,
        certPref  [1] SEQUENCE SIZE (1..MAX) OF TrustedAuth OPTIONAL
   }

   TokenAB ::= SEQUENCE {
        randomA   RandomNumber,
        entityB   [0] GeneralNames OPTIONAL,
        certA     [1] CertData,
        authID    [2] GeneralNames OPTIONAL,
        signature SIGNATURE { TBSDataAB }
   }(CONSTRAINED BY {-- The entityB and authID fields shall be included
     -- in TokenAB if and only if they are also included in TBSDataAB.
     -- The entityB field SHOULD be present in TokenAB whenever the
     -- client believes it knows the identity of the server.--})

   TBSDataAB ::= SEQUENCE {
        randomA RandomNumber,
        randomB RandomNumber,
        entityB [0] GeneralNames OPTIONAL,
        authID  [1] GeneralNames OPTIONAL
   }
```

```
   TokenBA2 ::= SEQUENCE {
        randomC   RandomNumber,
        entityA   [0] GeneralNames OPTIONAL,
        certB     [1] CertData,
        signature SIGNATURE { TBSDataBA }
   }(CONSTRAINED BY {-- The entityA field shall be included in TokenBA2
     -- if and only if it is also included in TBSDataBA.  The entityA
     -- field SHOULD be present and MUST contain the client's name
     -- from their X.509 certificate.--})

   TBSDataBA ::= SEQUENCE {
        randomB RandomNumber,
        randomA RandomNumber,
        randomC RandomNumber,
        entityA GeneralNames OPTIONAL
   }

   TrustedAuth ::= CHOICE {
        authorityName         [0] Name,
             -- SubjectName from CA certificate
        issuerNameHash        [1] OCTET STRING,
             -- SHA-1 hash of Authority's DN
        issuerKeyHash         [2] OCTET STRING,
             -- SHA-1 hash of Authority's public key
        authorityCertificate  [3] Certificate,
             -- CA certificate
        pkcs15KeyHash         [4] OCTET STRING
             -- PKCS #15 key hash
   }

   CertData ::= CHOICE {
        certificateSet     SET SIZE (1..MAX) OF Certificate,
        certURL            IA5String,
        ... -- For future extensions
   }

   RandomNumber ::= OCTET STRING (SIZE(8..MAX))

   SIGNATURE { ToBeSigned } ::= SEQUENCE {
        algorithm AlgorithmIdentifier,
        signature BIT STRING
   }(CONSTRAINED BY {-- Must be the result of applying the signing
     -- operation indicated in "algorithm" to the DER-encoded octets of
     -- a value of type -- ToBeSigned })

   END
```

Full Copyright Statement

Acknowledgement

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