Network Working Group
Request for Comments: 4556
Category: Standards Track

L. Zhu
Microsoft Corporation
B. Tung
Aerospace Corporation

June 2006

Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document describes protocol extensions (hereafter called PKINIT) to the Kerberos protocol specification. These extensions provide a method for integrating public key cryptography into the initial authentication exchange, by using asymmetric-key signature and/or encryption algorithms in pre-authentication data fields.

Table of Contents

1. Introduction

The Kerberos V5 protocol [RFC4120] involves use of a trusted third party known as the Key Distribution Center (KDC) to negotiate shared session keys between clients and services and provide mutual authentication between them.

The corner-stones of Kerberos V5 are the Ticket and the Authenticator. A Ticket encapsulates a symmetric key (the ticket session key) in an envelope (a public message) intended for a specific service. The contents of the Ticket are encrypted with a symmetric key shared between the service principal and the issuing KDC. The encrypted part of the Ticket contains the client principal name, among other items. An Authenticator is a record that can be shown to have been recently generated using the ticket session key in the associated Ticket. The ticket session key is known by the client who requested the ticket. The contents of the Authenticator are encrypted with the associated ticket session key. The encrypted part of an Authenticator contains a timestamp and the client principal name, among other items.

As shown in Figure 1, below, the Kerberos V5 protocol consists of the following message exchanges between the client and the KDC, and the client and the application service:

- The Authentication Service (AS) Exchange

  The client obtains an "initial" ticket from the Kerberos authentication server (AS), typically a Ticket Granting Ticket (TGT). The AS-REQ message and the AS-REP message are the request and the reply message, respectively, between the client and the AS.

- The Ticket Granting Service (TGS) Exchange

  The client subsequently uses the TGT to authenticate and request a service ticket for a particular service, from the Kerberos ticket-granting server (TGS). The TGS-REQ message and the TGS-REP message are the request and the reply message respectively between the client and the TGS.

- The Client/Server Authentication Protocol (AP) Exchange

  The client then makes a request with an AP-REQ message, consisting of a service ticket and an authenticator that certifies the client's possession of the ticket session key. The server may optionally reply with an AP-REP message. AP exchanges typically negotiate session-specific symmetric keys.

Usually, the AS and TGS are integrated in a single device also known as the KDC.

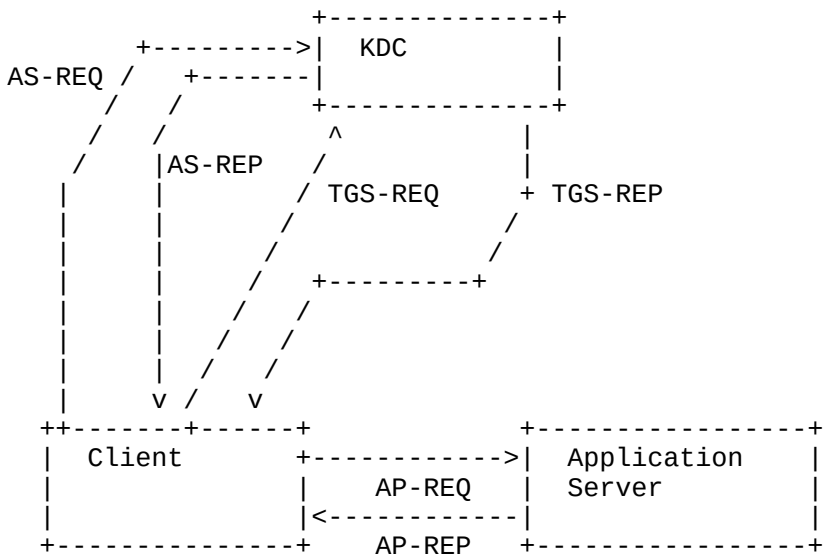

Figure 1: The Message Exchanges in the Kerberos V5 Protocol

In the AS exchange, the KDC reply contains the ticket session key, among other items, that is encrypted using a key (the AS reply key) shared between the client and the KDC. The AS reply key is typically derived from the client's password for human users. Therefore, for human users, the attack resistance strength of the Kerberos protocol is no stronger than the strength of their passwords.

The use of asymmetric cryptography in the form of X.509 certificates [RFC3280] is popular for facilitating data origin authentication and perfect secrecy. An established Public Key Infrastructure (PKI) provides key management and key distribution mechanisms that can be used to establish authentication and secure communication. Adding public-key cryptography to Kerberos provides a nice congruence to public-key protocols, obviates the human users' burden to manage strong passwords, and allows Kerberized applications to take advantage of existing key services and identity management.

The advantage afforded by the Kerberos TGT is that the client exposes his long-term secrets only once. The TGT and its associated session key can then be used for any subsequent service ticket requests. One result of this is that all further authentication is independent of the method by which the initial authentication was performed. Consequently, initial authentication provides a convenient place to integrate public-key cryptography into Kerberos authentication. In addition, the use of symmetric cryptography after the initial exchange is preferred for performance.

This document describes the methods and data formats using which the client and the KDC can use public and private key pairs to mutually authenticate in the AS exchange and negotiate the AS reply key, known only by the client and the KDC, to encrypt the AS-REP sent by the KDC.

2. Conventions Used in This Document

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 [RFC2119].

In this protocol, both the client and the KDC have a public-private key pair in order to prove their identities to each other over the open network. The term "signature key" is used to refer to the private key of the key pair being used.

The encryption key used to encrypt the enc-part field of the KDC-REP in the AS-REP [RFC4120] is referred to as the AS reply key.

An empty sequence in an optional field can be either included or omitted: both encodings are permitted and considered equivalent.

The term "Modular Exponential Diffie-Hellman" is used to refer to the Diffie-Hellman key exchange, as described in [RFC2631], in order to differentiate it from other equivalent representations of the same key agreement algorithm.

3. Extensions

This section describes extensions to [RFC4120] for supporting the use of public-key cryptography in the initial request for a ticket.

Briefly, this document defines the following extensions to [RFC4120]:

1. The client indicates the use of public-key authentication by including a special preauthenticator in the initial request. This preauthenticator contains the client's public-key data and a signature.

2. The KDC tests the client's request against its authentication policy and trusted Certification Authorities (CAs).

3. If the request passes the verification tests, the KDC replies as usual, but the reply is encrypted using either:

   a. a key generated through a Diffie-Hellman (DH) key exchange [RFC2631] [IEEE1363] with the client, signed using the KDC's signature key; or

   b. a symmetric encryption key, signed using the KDC's signature key and encrypted using the client's public key.

   Any keying material required by the client to obtain the encryption key for decrypting the KDC reply is returned in a pre-authentication field accompanying the usual reply.

4. The client validates the KDC's signature, obtains the encryption key, decrypts the reply, and then proceeds as usual.

Section 3.1 of this document enumerates the required algorithms and necessary extension message types. Section 3.2 describes the extension messages in greater detail.

3.1. Definitions, Requirements, and Constants

3.1.1. Required Algorithms

All PKINIT implementations MUST support the following algorithms:

- AS reply key enctypes: aes128-cts-hmac-sha1-96 and aes256-cts-hmac-sha1-96 [RFC3962].
- Signature algorithm: sha-1WithRSAEncryption [RFC3370].
- AS reply key delivery method: the Diffie-Hellman key delivery method, as described in Section 3.2.3.1.

In addition, implementations of this specification MUST be capable of processing the Extended Key Usage (EKU) extension and the id-pkinit-san (as defined in Section 3.2.2) otherName of the Subject Alternative Name (SAN) extension in X.509 certificates [RFC3280].

3.1.2. Recommended Algorithms

All PKINIT implementations SHOULD support the following algorithm:

- AS reply key delivery method: the public key encryption key delivery method, as described in Section 3.2.3.2.

For implementations that support the public key encryption key delivery method, the following algorithms MUST be supported:

a) Key transport algorithms identified in the keyEncryptionAlgorithm field of the type KeyTransRecipientInfo [RFC3852] for encrypting the temporary key in the encryptedKey field [RFC3852] with a public key, as described in Section 3.2.3.2: rsaEncryption (this is the RSAES-PKCS1-v1_5 encryption scheme) [RFC3370] [RFC3447].

b) Content encryption algorithms identified in the contentEncryptionAlgorithm field of the type EncryptedContentInfo [RFC3852] for encrypting the AS reply key with the temporary key contained in the encryptedKey field of the type KeyTransRecipientInfo [RFC3852], as described in Section 3.2.3.2: des-ede3-cbc (three-key 3DES, CBC mode) [RFC3370].

3.1.3. Defined Message and Encryption Types

PKINIT makes use of the following new pre-authentication types:

PA_PK_AS_REQ	16
PA_PK_AS_REP	17

PKINIT also makes use of the following new authorization data type:

AD_INITIAL_VERIFIED_CAS	9

PKINIT introduces the following new error codes:

KDC_ERR_CLIENT_NOT_TRUSTED	62
KDC_ERR_INVALID_SIG	64
KDC_ERR_DH_KEY_PARAMETERS_NOT_ACCEPTED	65
KDC_ERR_CANT_VERIFY_CERTIFICATE	70
KDC_ERR_INVALID_CERTIFICATE	71
KDC_ERR_REVOKED_CERTIFICATE	72
KDC_ERR_REVOCATION_STATUS_UNKNOWN	73
KDC_ERR_CLIENT_NAME_MISMATCH	75
KDC_ERR_INCONSISTENT_KEY_PURPOSE	77
KDC_ERR_DIGEST_IN_CERT_NOT_ACCEPTED	78
KDC_ERR_PA_CHECKSUM_MUST_BE_INCLUDED	79
KDC_ERR_DIGEST_IN_SIGNED_DATA_NOT_ACCEPTED	80
KDC_ERR_PUBLIC_KEY_ENCRYPTION_NOT_SUPPORTED	81

PKINIT uses the following typed data types for errors:

TD_TRUSTED_CERTIFIERS	104
TD_INVALID_CERTIFICATES	105
TD_DH_PARAMETERS	109

The ASN.1 module for all structures defined in this document (plus IMPORT statements for all imported structures) is given in Appendix A.

All structures defined in or imported into this document MUST be encoded using Distinguished Encoding Rules (DER) [X680] [X690] (unless otherwise noted). All data structures carried in OCTET STRINGs MUST be encoded according to the rules specified in the specifications defining each data structure; a reference to the appropriate specification is provided for each data structure.

Interoperability note: Some implementations may not be able to decode wrapped Cryptographic Message Syntax (CMS) [RFC3852] objects encoded with BER; specifically, they may not be able to decode indefinite-length encodings. To maximize interoperability, implementers SHOULD encode CMS objects used in PKINIT with DER.

3.1.4. Kerberos Encryption Types Defined for CMS Algorithm Identifiers

PKINIT defines the following Kerberos encryption type numbers [RFC3961], which can be used in the etype field of the AS-REQ [RFC4120] message to indicate to the KDC the client's acceptance of the corresponding algorithms (including key transport algorithms [RFC3370], content encryption algorithms [RFC3370], and signature algorithms) for use with Cryptographic Message Syntax (CMS) [RFC3852] [RFC3370].

Per [RFC4120], the encryption types in the etype field are in the decreasing preference order of the client. Note that there is no significance in the relative order between any two of different types of algorithms: key transport algorithms, content encryption algorithms, and signature algorithms.

The presence of each of these encryption types in the etype field is equivalent to the presence of the corresponding algorithm Object Identifier (OID) in the supportedCMSTypes field as described in Section 3.2.1. And the preference order expressed in the supportedCMSTypes field would override the preference order listed in the etype field.

Kerberos Encryption Type Name	Num	Corresponding Algorithm OID
id-dsa-with-sha1-CmsOID	9	id-dsa-with-sha1 [RFC3370]
md5WithRSAEncryption-CmsOID	10	md5WithRSAEncryption [RFC3370]
sha-1WithRSAEncryption-CmsOID	11	sha-1WithRSAEncryption [RFC3370]
rc2-cbc-EnvOID	12	rc2-cbc [RFC3370]
rsaEncryption-EnvOID	13	rsaEncryption [RFC3447][RFC3370]
id-RSAES-OAEP-EnvOID	14	id-RSAES-OAEP [RFC3447][RFC3560]
des-ede3-cbc-EnvOID	15	des-ede3-cbc [RFC3370]

The above encryption type numbers are used only to indicate support for the use of the corresponding algorithms in PKINIT; they do not correspond to actual Kerberos encryption types [RFC3961] and MUST NOT be used in the etype field of the Kerberos EncryptedData type [RFC4120]. The practice of assigning Kerberos encryption type numbers to indicate support for CMS algorithms is considered deprecated, and new numbers should not be assigned for this purpose. Instead, the supportedCMSTypes field should be used to identify the algorithms supported by the client and the preference order of the client.

For maximum interoperability, however, PKINIT clients wishing to indicate to the KDC the support for one or more of the algorithms listed above SHOULD include the corresponding encryption type number(s) in the etype field of the AS-REQ.

3.2. PKINIT Pre-authentication Syntax and Use

This section defines the syntax and use of the various pre-authentication fields employed by PKINIT.

3.2.1. Generation of Client Request

The initial authentication request (AS-REQ) is sent as per [RFC4120]; in addition, a pre-authentication data element, whose padata-type is PA_PK_AS_REQ and whose padata-value contains the DER encoding of the type PA-PK-AS-REQ, is included.

```
    PA-PK-AS-REQ ::= SEQUENCE {
       signedAuthPack          [0] IMPLICIT OCTET STRING,
                -- Contains a CMS type ContentInfo encoded
                -- according to [RFC3852].
                -- The contentType field of the type ContentInfo
                -- is id-signedData (1.2.840.113549.1.7.2),
                -- and the content field is a SignedData.
                -- The eContentType field for the type SignedData is
                -- id-pkinit-authData (1.3.6.1.5.2.3.1), and the
                -- eContent field contains the DER encoding of the
                -- type AuthPack.
                -- AuthPack is defined below.
       trustedCertifiers       [1] SEQUENCE OF
                   ExternalPrincipalIdentifier OPTIONAL,
                -- Contains a list of CAs, trusted by the client,
                -- that can be used to certify the KDC.
                -- Each ExternalPrincipalIdentifier identifies a CA
                -- or a CA certificate (thereby its public key).
                -- The information contained in the
                -- trustedCertifiers SHOULD be used by the KDC as
```

```
                -- hints to guide its selection of an appropriate
                -- certificate chain to return to the client.
      kdcPkId                [2] IMPLICIT OCTET STRING
                                 OPTIONAL,
                -- Contains a CMS type SignerIdentifier encoded
                -- according to [RFC3852].
                -- Identifies, if present, a particular KDC
                -- public key that the client already has.
      ...
   }

   DHNonce ::= OCTET STRING

   ExternalPrincipalIdentifier ::= SEQUENCE {
      subjectName            [0] IMPLICIT OCTET STRING OPTIONAL,
                -- Contains a PKIX type Name encoded according to
                -- [RFC3280].
                -- Identifies the certificate subject by the
                -- distinguished subject name.
                -- REQUIRED when there is a distinguished subject
                -- name present in the certificate.
     issuerAndSerialNumber   [1] IMPLICIT OCTET STRING OPTIONAL,
                -- Contains a CMS type IssuerAndSerialNumber encoded
                -- according to [RFC3852].
                -- Identifies a certificate of the subject.
                -- REQUIRED for TD-INVALID-CERTIFICATES and
                -- TD-TRUSTED-CERTIFIERS.
     subjectKeyIdentifier    [2] IMPLICIT OCTET STRING OPTIONAL,
                -- Identifies the subject's public key by a key
                -- identifier.  When an X.509 certificate is
                -- referenced, this key identifier matches the X.509
                -- subjectKeyIdentifier extension value.  When other
                -- certificate formats are referenced, the documents
                -- that specify the certificate format and their use
                -- with the CMS must include details on matching the
                -- key identifier to the appropriate certificate
                -- field.
                -- RECOMMENDED for TD-TRUSTED-CERTIFIERS.
      ...
   }

   AuthPack ::= SEQUENCE {
      pkAuthenticator         [0] PKAuthenticator,
      clientPublicValue       [1] SubjectPublicKeyInfo OPTIONAL,
                -- Type SubjectPublicKeyInfo is defined in
                -- [RFC3280].
                -- Specifies Diffie-Hellman domain parameters
                -- and the client's public key value [IEEE1363].
```

```
                -- The DH public key value is encoded as a BIT
                -- STRING according to [RFC3279].
                -- This field is present only if the client wishes
                -- to use the Diffie-Hellman key agreement method.
          supportedCMSTypes       [2] SEQUENCE OF AlgorithmIdentifier
                                      OPTIONAL,
                -- Type AlgorithmIdentifier is defined in
                -- [RFC3280].
                -- List of CMS algorithm [RFC3370] identifiers
                -- that identify key transport algorithms, or
                -- content encryption algorithms, or signature
                -- algorithms supported by the client in order of
                -- (decreasing) preference.
          clientDHNonce           [3] DHNonce OPTIONAL,
                -- Present only if the client indicates that it
                -- wishes to reuse DH keys or to allow the KDC to
                -- do so (see Section 3.2.3.1).
          ...
       }

       PKAuthenticator ::= SEQUENCE {
          cusec                   [0] INTEGER (0..999999),
          ctime                   [1] KerberosTime,
                -- cusec and ctime are used as in [RFC4120], for
                -- replay prevention.
          nonce                   [2] INTEGER (0..4294967295),
                -- Chosen randomly;  this nonce does not need to
                -- match with the nonce in the KDC-REQ-BODY.
          paChecksum              [3] OCTET STRING OPTIONAL,
                -- MUST be present.
                -- Contains the SHA1 checksum, performed over
                -- KDC-REQ-BODY.
          ...
       }
```

The ContentInfo [RFC3852] structure contained in the signedAuthPack field of the type PA-PK-AS-REQ is encoded according to [RFC3852] and is filled out as follows:

1. The contentType field of the type ContentInfo is id-signedData (as defined in [RFC3852]), and the content field is a SignedData (as defined in [RFC3852]).

2. The eContentType field for the type SignedData is id-pkinit-authData: { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2) pkinit(3) authData(1) }. Notes to CMS implementers: the signed attribute content-type MUST be present in this SignedData instance, and its value is id-pkinit-authData according to [RFC3852].

3. The eContent field for the type SignedData contains the DER encoding of the type AuthPack.

4. The signerInfos field of the type SignedData contains a single signerInfo, which contains the signature over the type AuthPack.

5. The AuthPack structure contains a PKAuthenticator, the client public key information, the CMS encryption types supported by the client, and a DHNonce. The pkAuthenticator field certifies to the KDC that the client has recent knowledge of the signing key that authenticates the client. The clientPublicValue field specifies Diffie-Hellman domain parameters and the client's public key value. The DH public key value is encoded as a BIT STRING according to [RFC3279]. The clientPublicValue field is present only if the client wishes to use the Diffie-Hellman key agreement method. The supportedCMSTypes field specifies the list of CMS algorithm identifiers that are supported by the client in order of (decreasing) preference, and can be used to identify a signature algorithm or a key transport algorithm [RFC3370] in the keyEncryptionAlgorithm field of the type KeyTransRecipientInfo, or a content encryption algorithm [RFC3370] in the contentEncryptionAlgorithm field of the type EncryptedContentInfo [RFC3852] when encrypting the AS reply key as described in Section 3.2.3.2. However, there is no significance in the relative order between any two of different types of algorithms: key transport algorithms, content encryption algorithms, and signature algorithms. The clientDHNonce field is described later in this section.

6. The ctime field in the PKAuthenticator structure contains the current time on the client's host, and the cusec field contains the microsecond part of the client's timestamp. The ctime and cusec fields are used together to specify a reasonably accurate timestamp [RFC4120]. The nonce field is chosen randomly. The paChecksum field MUST be present and it contains a SHA1 checksum that is performed over the KDC-REQ-BODY [RFC4120]. In order to ease future migration from the use of SHA1, the paChecksum field is made optional syntactically: when the request is extended to negotiate hash algorithms, the new client wishing not to use SHA1 will send the request in the extended message syntax without the paChecksum field. The KDC conforming to this specification MUST

return a KRB-ERROR [RFC4120] message with the code KDC_ERR_PA_CHECKSUM_MUST_BE_INCLUDED (see Section 3.2.3). That will allow a new client to retry with SHA1 if allowed by the local policy.

7. The certificates field of the type SignedData contains certificates intended to facilitate certification path construction, so that the KDC can verify the signature over the type AuthPack. For path validation, these certificates SHOULD be sufficient to construct at least one certification path from the client certificate to one trust anchor acceptable by the KDC [RFC4158]. The client MUST be capable of including such a set of certificates if configured to do so. The certificates field MUST NOT contain "root" CA certificates.

8. The client's Diffie-Hellman public value (clientPublicValue) is included if and only if the client wishes to use the Diffie-Hellman key agreement method. The Diffie-Hellman domain parameters [IEEE1363] for the client's public key are specified in the algorithm field of the type SubjectPublicKeyInfo [RFC3279], and the client's Diffie-Hellman public key value is mapped to a subjectPublicKey (a BIT STRING) according to [RFC3279]. When using the Diffie-Hellman key agreement method, implementations MUST support Oakley 1024-bit Modular Exponential (MODP) well-known group 2 [RFC2412] and Oakley 2048-bit MODP well-known group 14 [RFC3526] and SHOULD support Oakley 4096-bit MODP well-known group 16 [RFC3526].

   The Diffie-Hellman field size should be chosen so as to provide sufficient cryptographic security [RFC3766].

   When MODP Diffie-Hellman is used, the exponents should have at least twice as many bits as the symmetric keys that will be derived from them [ODL99].

9. The client may wish to reuse DH keys or to allow the KDC to do so (see Section 3.2.3.1). If so, then the client includes the clientDHNonce field. This nonce string MUST be as long as the longest key length of the symmetric key types that the client supports. This nonce MUST be chosen randomly.

The ExternalPrincipalIdentifier structure is used in this document to identify the subject's public key thereby the subject principal. This structure is filled out as follows:

1. The subjectName field contains a PKIX type Name encoded according to [RFC3280]. This field identifies the certificate subject by the distinguished subject name. This field is REQUIRED when

there is a distinguished subject name present in the certificate being used.

2. The issuerAndSerialNumber field contains a CMS type IssuerAndSerialNumber encoded according to [RFC3852]. This field identifies a certificate of the subject. This field is REQUIRED for TD-INVALID-CERTIFICATES and TD-TRUSTED-CERTIFIERS (both structures are defined in Section 3.2.2).

3. The subjectKeyIdentifier [RFC3852] field identifies the subject's public key by a key identifier. When an X.509 certificate is referenced, this key identifier matches the X.509 subjectKeyIdentifier extension value. When other certificate formats are referenced, the documents that specify the certificate format and their use with the CMS must include details on matching the key identifier to the appropriate certificate field. This field is RECOMMENDED for TD-TRUSTED-CERTIFIERS (as defined in Section 3.2.2).

The trustedCertifiers field of the type PA-PK-AS-REQ contains a list of CAs, trusted by the client, that can be used to certify the KDC. Each ExternalPrincipalIdentifier identifies a CA or a CA certificate (thereby its public key).

The kdcPkId field of the type PA-PK-AS-REQ contains a CMS type SignerIdentifier encoded according to [RFC3852]. This field identifies, if present, a particular KDC public key that the client already has.

3.2.2. Receipt of Client Request

Upon receiving the client's request, the KDC validates it. This section describes the steps that the KDC MUST (unless otherwise noted) take in validating the request.

The KDC verifies the client's signature in the signedAuthPack field according to [RFC3852].

If, while validating the client's X.509 certificate [RFC3280], the KDC cannot build a certification path to validate the client's certificate, it sends back a KRB-ERROR [RFC4120] message with the code KDC_ERR_CANT_VERIFY_CERTIFICATE. The accompanying e-data for this error message is a TYPED-DATA (as defined in [RFC4120]) that contains an element whose data-type is TD_TRUSTED_CERTIFIERS, and whose data-value contains the DER encoding of the type TD-TRUSTED-CERTIFIERS:

```
TD-TRUSTED-CERTIFIERS ::= SEQUENCE OF
                ExternalPrincipalIdentifier
            -- Identifies a list of CAs trusted by the KDC.
            -- Each ExternalPrincipalIdentifier identifies a CA
            -- or a CA certificate (thereby its public key).
```

Each ExternalPrincipalIdentifier (as defined in Section 3.2.1) in the TD-TRUSTED-CERTIFIERS structure identifies a CA or a CA certificate (thereby its public key) trusted by the KDC.

Upon receiving this error message, the client SHOULD retry only if it has a different set of certificates (from those of the previous requests) that form a certification path (or a partial path) from one of the trust anchors acceptable by the KDC to its own certificate.

If, while processing the certification path, the KDC determines that the signature on one of the certificates in the signedAuthPack field is invalid, it returns a KRB-ERROR [RFC4120] message with the code KDC_ERR_INVALID_CERTIFICATE. The accompanying e-data for this error message is a TYPED-DATA that contains an element whose data-type is TD_INVALID_CERTIFICATES, and whose data-value contains the DER encoding of the type TD-INVALID-CERTIFICATES:

```
TD-INVALID-CERTIFICATES ::= SEQUENCE OF
                ExternalPrincipalIdentifier
            -- Each ExternalPrincipalIdentifier identifies a
            -- certificate (sent by the client) with an invalid
            -- signature.
```

Each ExternalPrincipalIdentifier (as defined in Section 3.2.1) in the TD-INVALID-CERTIFICATES structure identifies a certificate (that was sent by the client) with an invalid signature.

If more than one X.509 certificate signature is invalid, the KDC MAY include one IssuerAndSerialNumber per invalid signature within the TD-INVALID-CERTIFICATES.

The client's X.509 certificate is validated according to [RFC3280].

Depending on local policy, the KDC may also check whether any X.509 certificates in the certification path validating the client's certificate have been revoked. If any of them have been revoked, the KDC MUST return an error message with the code KDC_ERR_REVOKED_CERTIFICATE; if the KDC attempts to determine the revocation status but is unable to do so, it SHOULD return an error message with the code KDC_ERR_REVOCATION_STATUS_UNKNOWN. The certificate or certificates affected are identified exactly as for the error code KDC_ERR_INVALID_CERTIFICATE (see above).

Note that the TD_INVALID_CERTIFICATES error data is only used to identify invalid certificates sent by the client in the request.

The client's public key is then used to verify the signature. If the signature fails to verify, the KDC MUST return an error message with the code KDC_ERR_INVALID_SIG. There is no accompanying e-data for this error message.

In addition to validating the client's signature, the KDC MUST also check that the client's public key used to verify the client's signature is bound to the client principal name specified in the AS-REQ as follows:

1. If the KDC has its own binding between either the client's signature-verification public key or the client's certificate and the client's Kerberos principal name, it uses that binding.

2. Otherwise, if the client's X.509 certificate contains a Subject Alternative Name (SAN) extension carrying a KRB5PrincipalName (defined below) in the otherName field of the type GeneralName [RFC3280], it binds the client's X.509 certificate to that name.

   The type of the otherName field is AnotherName. The type-id field of the type AnotherName is id-pkinit-san:

```
id-pkinit-san OBJECT IDENTIFIER ::=
  { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2)
    x509SanAN (2) }
```

   And the value field of the type AnotherName is a KRB5PrincipalName.

```
KRB5PrincipalName ::= SEQUENCE {
    realm                   [0] Realm,
    principalName           [1] PrincipalName
}
```

If the Kerberos client name in the AS-REQ does not match a name bound by the KDC (the binding can be in the certificate, for example, as described above), or if there is no binding found by the KDC, the KDC MUST return an error message with the code KDC_ERR_CLIENT_NAME_MISMATCH. There is no accompanying e-data for this error message.

Even if the certification path is validated and the certificate is mapped to the client's principal name, the KDC may decide not to accept the client's certificate, depending on local policy.

The KDC MAY require the presence of an Extended Key Usage (EKU) KeyPurposeId [RFC3280] id-pkinit-KPClientAuth in the extensions field of the client's X.509 certificate:

```
id-pkinit-KPClientAuth OBJECT IDENTIFIER ::=
  { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2)
    pkinit(3) keyPurposeClientAuth(4) }
       -- PKINIT client authentication.
       -- Key usage bits that MUST be consistent:
       -- digitalSignature.
```

The digitalSignature key usage bit [RFC3280] MUST be asserted when the intended purpose of the client's X.509 certificate is restricted with the id-pkinit-KPClientAuth EKU.

If this EKU KeyPurposeId is required but it is not present, or if the client certificate is restricted not to be used for PKINIT client authentication per Section 4.2.1.13 of [RFC3280], the KDC MUST return an error message of the code KDC_ERR_INCONSISTENT_KEY_PURPOSE. There is no accompanying e-data for this error message. KDCs implementing this requirement SHOULD also accept the EKU KeyPurposeId id-ms-kp-sc-logon (1.3.6.1.4.1.311.20.2.2) as meeting the requirement, as there are a large number of X.509 client certificates deployed for use with PKINIT that have this EKU.

As a matter of local policy, the KDC MAY decide to reject requests on the basis of the absence or presence of other specific EKU OIDs.

If the digest algorithm used in generating the CA signature for the public key in any certificate of the request is not acceptable by the KDC, the KDC MUST return a KRB-ERROR [RFC4120] message with the code KDC_ERR_DIGEST_IN_CERT_NOT_ACCEPTED. The accompanying e-data MUST be encoded in TYPED-DATA, although none is defined at this point.

If the client's public key is not accepted with reasons other than those specified above, the KDC returns a KRB-ERROR [RFC4120] message with the code KDC_ERR_CLIENT_NOT_TRUSTED. There is no accompanying e-data currently defined for this error message.

The KDC MUST check the timestamp to ensure that the request is not a replay, and that the time skew falls within acceptable limits. The recommendations for clock skew times in [RFC4120] apply here. If the check fails, the KDC MUST return error code KRB_AP_ERR_REPEAT or KRB_AP_ERR_SKEW, respectively.

If the clientPublicValue is filled in, indicating that the client wishes to use the Diffie-Hellman key agreement method, the KDC SHOULD check to see if the key parameters satisfy its policy. If they do

not, it MUST return an error message with the code KDC_ERR_DH_KEY_PARAMETERS_NOT_ACCEPTED. The accompanying e-data is a TYPED-DATA that contains an element whose data-type is TD_DH_PARAMETERS, and whose data-value contains the DER encoding of the type TD-DH-PARAMETERS:

```
TD-DH-PARAMETERS ::= SEQUENCE OF AlgorithmIdentifier
            -- Each AlgorithmIdentifier specifies a set of
            -- Diffie-Hellman domain parameters [IEEE1363].
            -- This list is in decreasing preference order.
```

TD-DH-PARAMETERS contains a list of Diffie-Hellman domain parameters that the KDC supports in decreasing preference order, from which the client SHOULD pick one to retry the request.

The AlgorithmIdentifier structure is defined in [RFC3280] and is filled in according to [RFC3279]. More specifically, Section 2.3.3 of [RFC3279] describes how to fill in the AlgorithmIdentifier structure in the case where MODP Diffie-Hellman key exchange is used.

If the client included a kdcPkId field in the PA-PK-AS-REQ and the KDC does not possess the corresponding key, the KDC MUST ignore the kdcPkId field as if the client did not include one.

If the digest algorithm used by the id-pkinit-authData is not acceptable by the KDC, the KDC MUST return a KRB-ERROR [RFC4120] message with the code KDC_ERR_DIGEST_IN_SIGNED_DATA_NOT_ACCEPTED. The accompanying e-data MUST be encoded in TYPED-DATA, although none is defined at this point.

3.2.3. Generation of KDC Reply

If the paChecksum filed in the request is not present, the KDC conforming to this specification MUST return a KRB-ERROR [RFC4120] message with the code KDC_ERR_PA_CHECKSUM_MUST_BE_INCLUDED. The accompanying e-data MUST be encoded in TYPED-DATA (no error data is defined by this specification).

Assuming that the client’s request has been properly validated, the KDC proceeds as per [RFC4120], except as follows.

The KDC MUST set the initial flag and include an authorization data element of ad-type [RFC4120] AD_INITIAL_VERIFIED_CAS in the issued ticket. The ad-data [RFC4120] field contains the DER encoding of the type AD-INITIAL-VERIFIED-CAS:

```
AD-INITIAL-VERIFIED-CAS ::= SEQUENCE OF
                ExternalPrincipalIdentifier
            -- Identifies the certification path with which
            -- the client certificate was validated.
            -- Each ExternalPrincipalIdentifier identifies a CA
            -- or a CA certificate (thereby its public key).
```

The AD-INITIAL-VERIFIED-CAS structure identifies the certification path with which the client certificate was validated. Each ExternalPrincipalIdentifier (as defined in Section 3.2.1) in the AD-INITIAL-VERIFIED-CAS structure identifies a CA or a CA certificate (thereby its public key).

Note that the syntax for the AD-INITIAL-VERIFIED-CAS authorization data does permit empty SEQUENCEs to be encoded. Such empty sequences may only be used if the KDC itself vouches for the user's certificate.

The AS wraps any AD-INITIAL-VERIFIED-CAS data in AD-IF-RELEVANT containers if the list of CAs satisfies the AS' realm's local policy (this corresponds to the TRANSITED-POLICY-CHECKED ticket flag [RFC4120]). Furthermore, any TGS MUST copy such authorization data from tickets used within a PA-TGS-REQ of the TGS-REQ into the resulting ticket. If the list of CAs satisfies the local KDC's realm's policy, the TGS MAY wrap the data into the AD-IF-RELEVANT container; otherwise, it MAY unwrap the authorization data out of the AD-IF-RELEVANT container.

Application servers that understand this authorization data type SHOULD apply local policy to determine whether a given ticket bearing such a type *not* contained within an AD-IF-RELEVANT container is acceptable. (This corresponds to the AP server's checking the transited field when the TRANSITED-POLICY-CHECKED flag has not been set [RFC4120].) If such a data type is contained within an AD-IF-RELEVANT container, AP servers MAY apply local policy to determine whether the authorization data is acceptable.

A pre-authentication data element, whose padata-type is PA_PK_AS_REP and whose padata-value contains the DER encoding of the type PA-PK-AS-REP (defined below), is included in the AS-REP [RFC4120].

```
PA-PK-AS-REP ::= CHOICE {
   dhInfo                  [0] DHRepInfo,
            -- Selected when Diffie-Hellman key exchange is
            -- used.
   encKeyPack              [1] IMPLICIT OCTET STRING,
            -- Selected when public key encryption is used.
            -- Contains a CMS type ContentInfo encoded
```

```
                -- according to [RFC3852].
                -- The contentType field of the type ContentInfo is
                -- id-envelopedData (1.2.840.113549.1.7.3).
                -- The content field is an EnvelopedData.
                -- The contentType field for the type EnvelopedData
                -- is id-signedData (1.2.840.113549.1.7.2).
                -- The eContentType field for the inner type
                -- SignedData (when unencrypted) is
                -- id-pkinit-rkeyData (1.3.6.1.5.2.3.3) and the
                -- eContent field contains the DER encoding of the
                -- type ReplyKeyPack.
                -- ReplyKeyPack is defined in Section 3.2.3.2.
   ...
}

DHRepInfo ::= SEQUENCE {
   dhSignedData            [0] IMPLICIT OCTET STRING,
                -- Contains a CMS type ContentInfo encoded according
                -- to [RFC3852].
                -- The contentType field of the type ContentInfo is
                -- id-signedData (1.2.840.113549.1.7.2), and the
                -- content field is a SignedData.
                -- The eContentType field for the type SignedData is
                -- id-pkinit-DHKeyData (1.3.6.1.5.2.3.2), and the
                -- eContent field contains the DER encoding of the
                -- type KDCDHKeyInfo.
                -- KDCDHKeyInfo is defined below.
   serverDHNonce           [1] DHNonce OPTIONAL,
                -- Present if and only if dhKeyExpiration is
                -- present in the KDCDHKeyInfo.
   ...
}

KDCDHKeyInfo ::= SEQUENCE {
   subjectPublicKey        [0] BIT STRING,
                -- The KDC's DH public key.
                -- The DH public key value is encoded as a BIT
                -- STRING according to [RFC3279].
   nonce                   [1] INTEGER (0..4294967295),
                -- Contains the nonce in the pkAuthenticator field
                -- in the request if the DH keys are NOT reused,
                -- 0 otherwise.
   dhKeyExpiration         [2] KerberosTime OPTIONAL,
                -- Expiration time for KDC's key pair,
                -- present if and only if the DH keys are reused.
                -- If present, the KDC's DH public key MUST not be
                -- used past the point of this expiration time.
                -- If this field is omitted then the serverDHNonce
```

```
                -- field MUST also be omitted.
        ...
    }
```

The content of the AS-REP is otherwise unchanged from [RFC4120]. The KDC encrypts the reply as usual, but not with the client's long-term key. Instead, it encrypts it with either a shared key derived from a Diffie-Hellman exchange or a generated encryption key. The contents of the PA-PK-AS-REP indicate which key delivery method is used.

If the client does not wish to use the Diffie-Hellman key delivery method (the clientPublicValue field is not present in the request) and the KDC does not support the public key encryption key delivery method, the KDC MUST return an error message with the code KDC_ERR_PUBLIC_KEY_ENCRYPTION_NOT_SUPPORTED. There is no accompanying e-data for this error message.

In addition, the lifetime of the ticket returned by the KDC MUST NOT exceed that of the client's public-private key pair. The ticket lifetime, however, can be shorter than that of the client's public-private key pair. For the implementations of this specification, the lifetime of the client's public-private key pair is the validity period in X.509 certificates [RFC3280], unless configured otherwise.

3.2.3.1. Using Diffie-Hellman Key Exchange

In this case, the PA-PK-AS-REP contains a DHRepInfo structure.

The ContentInfo [RFC3852] structure for the dhSignedData field is filled in as follows:

1. The contentType field of the type ContentInfo is id-signedData (as defined in [RFC3852]), and the content field is a SignedData (as defined in [RFC3852]).

2. The eContentType field for the type SignedData is the OID value for id-pkinit-DHKeyData: { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2) pkinit(3) DHKeyData(2) }. Notes to CMS implementers: the signed attribute content-type MUST be present in this SignedData instance, and its value is id-pkinit-DHKeyData according to [RFC3852].

3. The eContent field for the type SignedData contains the DER encoding of the type KDCDHKeyInfo.

4. The KDCDHKeyInfo structure contains the KDC's public key, a nonce, and, optionally, the expiration time of the KDC's DH key being reused. The subjectPublicKey field of the type

KDCDHKeyInfo field identifies KDC's DH public key. This DH public key value is encoded as a BIT STRING according to [RFC3279]. The nonce field contains the nonce in the pkAuthenticator field in the request if the DH keys are NOT reused. The value of this nonce field is 0 if the DH keys are reused. The dhKeyExpiration field is present if and only if the DH keys are reused. If the dhKeyExpiration field is present, the KDC's public key in this KDCDHKeyInfo structure MUST NOT be used past the point of this expiration time. If this field is omitted, then the serverDHNonce field MUST also be omitted.

5. The signerInfos field of the type SignedData contains a single signerInfo, which contains the signature over the type KDCDHKeyInfo.

6. The certificates field of the type SignedData contains certificates intended to facilitate certification path construction, so that the client can verify the KDC's signature over the type KDCDHKeyInfo. The information contained in the trustedCertifiers in the request SHOULD be used by the KDC as hints to guide its selection of an appropriate certificate chain to return to the client. This field may be left empty if the KDC public key specified by the kdcPkId field in the PA-PK-AS-REQ was used for signing. Otherwise, for path validation, these certificates SHOULD be sufficient to construct at least one certification path from the KDC certificate to one trust anchor acceptable by the client [RFC4158]. The KDC MUST be capable of including such a set of certificates if configured to do so. The certificates field MUST NOT contain "root" CA certificates.

7. If the client included the clientDHNonce field, then the KDC may choose to reuse its DH keys. If the server reuses DH keys, then it MUST include an expiration time in the dhKeyExpiration field. Past the point of the expiration time, the signature over the type DHRepInfo is considered expired/invalid. When the server reuses DH keys then, it MUST include a serverDHNonce at least as long as the length of keys for the symmetric encryption system used to encrypt the AS reply. Note that including the serverDHNonce changes how the client and server calculate the key to use to encrypt the reply; see below for details. The KDC SHOULD NOT reuse DH keys unless the clientDHNonce field is present in the request.

The AS reply key is derived as follows:

1. Both the KDC and the client calculate the shared secret value as follows:

a) When MODP Diffie-Hellman is used, let DHSharedSecret be the shared secret value. DHSharedSecret is the value ZZ, as described in Section 2.1.1 of [RFC2631].

DHSharedSecret is first padded with leading zeros such that the size of DHSharedSecret in octets is the same as that of the modulus, then represented as a string of octets in big-endian order.

Implementation note: Both the client and the KDC can cache the triple (ya, yb, DHSharedSecret), where ya is the client's public key and yb is the KDC's public key. If both ya and yb are the same in a later exchange, the cached DHSharedSecret can be used.

2. Let K be the key-generation seed length [RFC3961] of the AS reply key whose enctype is selected according to [RFC4120].

3. Define the function octetstring2key() as follows:

```
octetstring2key(x) == random-to-key(K-truncate(
                          SHA1(0x00 | x) |
                          SHA1(0x01 | x) |
                          SHA1(0x02 | x) |
                          ...
                          ))
```

where x is an octet string; | is the concatenation operator; 0x00, 0x01, 0x02, etc. are each represented as a single octet; random-to-key() is an operation that generates a protocol key from a bitstring of length K; and K-truncate truncates its input to the first K bits. Both K and random-to-key() are as defined in the kcrypto profile [RFC3961] for the enctype of the AS reply key.

4. When DH keys are reused, let n_c be the clientDHNonce and n_k be the serverDHNonce; otherwise, let both n_c and n_k be empty octet strings.

5. The AS reply key k is:

```
k = octetstring2key(DHSharedSecret | n_c | n_k)
```

3.2.3.2. Using Public Key Encryption

In this case, the PA-PK-AS-REP contains the encKeyPack field where the AS reply key is encrypted.

The ContentInfo [RFC3852] structure for the encKeyPack field is filled in as follows:

1. The contentType field of the type ContentInfo is id-envelopedData (as defined in [RFC3852]), and the content field is an EnvelopedData (as defined in [RFC3852]).

2. The contentType field for the type EnvelopedData is id-signedData: { iso (1) member-body (2) us (840) rsadsi (113549) pkcs (1) pkcs7 (7) signedData (2) }.

3. The eContentType field for the inner type SignedData (when decrypted from the encryptedContent field for the type EnvelopedData) is id-pkinit-rkeyData: { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2) pkinit(3) rkeyData(3) }. Notes to CMS implementers: the signed attribute content-type MUST be present in this SignedData instance, and its value is id-pkinit-rkeyData according to [RFC3852].

4. The eContent field for the inner type SignedData contains the DER encoding of the type ReplyKeyPack (as described below).

5. The signerInfos field of the inner type SignedData contains a single signerInfo, which contains the signature for the type ReplyKeyPack.

6. The certificates field of the inner type SignedData contains certificates intended to facilitate certification path construction, so that the client can verify the KDC's signature for the type ReplyKeyPack. The information contained in the trustedCertifiers in the request SHOULD be used by the KDC as hints to guide its selection of an appropriate certificate chain to return to the client. This field may be left empty if the KDC public key specified by the kdcPkId field in the PA-PK-AS-REQ was used for signing. Otherwise, for path validation, these certificates SHOULD be sufficient to construct at least one certification path from the KDC certificate to one trust anchor acceptable by the client [RFC4158]. The KDC MUST be capable of including such a set of certificates if configured to do so. The certificates field MUST NOT contain "root" CA certificates.

7. The recipientInfos field of the type EnvelopedData is a SET that MUST contain exactly one member of type KeyTransRecipientInfo. The encryptedKey of this member contains the temporary key that is encrypted using the client's public key.

8. The unprotectedAttrs or originatorInfo fields of the type EnvelopedData MAY be present.

If there is a supportedCMSTypes field in the AuthPack, the KDC must check to see if it supports any of the listed types. If it supports more than one of the types, the KDC SHOULD use the one listed first. If it does not support any of them, it MUST return an error message with the code KDC_ERR_ETYPE_NOSUPP [RFC4120].

Furthermore, the KDC computes the checksum of the AS-REQ in the client request. This checksum is performed over the type AS-REQ, and the protocol key [RFC3961] of the checksum operation is the replyKey, and the key usage number is 6. If the replyKey's enctype is "newer" [RFC4120] [RFC4121], the checksum operation is the required checksum operation [RFC3961] of that enctype.

```
ReplyKeyPack ::= SEQUENCE {
   replyKey                [0] EncryptionKey,
            -- Contains the session key used to encrypt the
            -- enc-part field in the AS-REP, i.e., the
            -- AS reply key.
   asChecksum              [1] Checksum,
           -- Contains the checksum of the AS-REQ
           -- corresponding to the containing AS-REP.
           -- The checksum is performed over the type AS-REQ.
           -- The protocol key [RFC3961] of the checksum is the
           -- replyKey and the key usage number is 6.
           -- If the replyKey's enctype is "newer" [RFC4120]
           -- [RFC4121], the checksum is the required
           -- checksum operation [RFC3961] for that enctype.
           -- The client MUST verify this checksum upon receipt
           -- of the AS-REP.
   ...
}
```

Implementations of this RSA encryption key delivery method are RECOMMENDED to support RSA keys at least 2048 bits in size.

3.2.4. Receipt of KDC Reply

Upon receipt of the KDC's reply, the client proceeds as follows. If the PA-PK-AS-REP contains the dhSignedData field, the client derives the AS reply key using the same procedure used by the KDC, as defined in Section 3.2.3.1. Otherwise, the message contains the encKeyPack field, and the client decrypts and extracts the temporary key in the encryptedKey field of the member KeyTransRecipientInfo and then uses that as the AS reply key.

If the public key encryption method is used, the client MUST verify the asChecksum contained in the ReplyKeyPack.

In either case, the client MUST verify the signature in the SignedData according to [RFC3852]. The KDC's X.509 certificate MUST be validated according to [RFC3280]. In addition, unless the client can otherwise verify that the public key used to verify the KDC's signature is bound to the KDC of the target realm, the KDC's X.509 certificate MUST contain a Subject Alternative Name extension [RFC3280] carrying an AnotherName whose type-id is id-pkinit-san (as defined in Section 3.2.2) and whose value is a KRB5PrincipalName that matches the name of the TGS of the target realm (as defined in Section 7.3 of [RFC4120]).

Unless the client knows by some other means that the KDC certificate is intended for a Kerberos KDC, the client MUST require that the KDC certificate contains the EKU KeyPurposeId [RFC3280] id-pkinit-KPKdc:

```
id-pkinit-KPKdc OBJECT IDENTIFIER ::=
  { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2)
    pkinit(3) keyPurposeKdc(5) }
       -- Signing KDC responses.
       -- Key usage bits that MUST be consistent:
       -- digitalSignature.
```

The digitalSignature key usage bit [RFC3280] MUST be asserted when the intended purpose of the KDC's X.509 certificate is restricted with the id-pkinit-KPKdc EKU.

If the KDC certificate contains the Kerberos TGS name encoded as an id-pkinit-san SAN, this certificate is certified by the issuing CA as a KDC certificate, therefore the id-pkinit-KPKdc EKU is not required.

If all applicable checks are satisfied, the client then decrypts the enc-part field of the KDC-REP in the AS-REP, using the AS reply key, and then proceeds as described in [RFC4120].

3.3. Interoperability Requirements

The client MUST be capable of sending a set of certificates sufficient to allow the KDC to construct a certification path for the client's certificate, if the correct set of certificates is provided through configuration or policy.

If the client sends all the X.509 certificates on a certification path to a trust anchor acceptable by the KDC, and if the KDC cannot verify the client's public key otherwise, the KDC MUST be able to process path validation for the client's certificate based on the certificates in the request.

The KDC MUST be capable of sending a set of certificates sufficient to allow the client to construct a certification path for the KDC's certificate, if the correct set of certificates is provided through configuration or policy.

If the KDC sends all the X.509 certificates on a certification path to a trust anchor acceptable by the client, and the client can not verify the KDC's public key otherwise, the client MUST be able to process path validation for the KDC's certificate based on the certificates in the reply.

3.4. KDC Indication of PKINIT Support

If pre-authentication is required but was not present in the request, per [RFC4120] an error message with the code KDC_ERR_PREAUTH_FAILED is returned, and a METHOD-DATA object will be stored in the e-data field of the KRB-ERROR message to specify which pre-authentication mechanisms are acceptable. The KDC can then indicate the support of PKINIT by including an empty element whose padata-type is PA_PK_AS_REQ in that METHOD-DATA object.

Otherwise if it is required by the KDC's local policy that the client must be pre-authenticated using the pre-authentication mechanism specified in this document, but no PKINIT pre-authentication was present in the request, an error message with the code KDC_ERR_PREAUTH_FAILED SHOULD be returned.

KDCs MUST leave the padata-value field of the PA_PK_AS_REQ element in the KRB-ERROR's METHOD-DATA empty (i.e., send a zero-length OCTET STRING), and clients MUST ignore this and any other value. Future extensions to this protocol may specify other data to send instead of an empty OCTET STRING.

4. Security Considerations

The security of cryptographic algorithms is dependent on generating secret quantities [RFC4086]. The number of truly random bits is extremely important in determining the attack resistance strength of the cryptosystem; for example, the secret Diffie-Hellman exponents must be chosen based on n truly random bits (where n is the system security requirement). The security of the overall system is significantly weakened by using insufficient random inputs: a sophisticated attacker may find it easier to reproduce the environment that produced the secret quantities and to search the resulting small set of possibilities than to locate the quantities in the whole of the potential number space.

Kerberos error messages are not integrity protected; as a result, the domain parameters sent by the KDC as TD-DH-PARAMETERS can be tampered with by an attacker so that the set of domain parameters selected could be either weaker or not mutually preferred. Local policy can configure sets of domain parameters acceptable locally, or disallow the negotiation of DH domain parameters.

The symmetric reply key size and Diffie-Hellman field size or RSA modulus size should be chosen so as to provide sufficient cryptographic security [RFC3766].

When MODP Diffie-Hellman is used, the exponents should have at least twice as many bits as the symmetric keys that will be derived from them [ODL99].

PKINIT raises certain security considerations beyond those that can be regulated strictly in protocol definitions. We will address them in this section.

PKINIT extends the cross-realm model to the public-key infrastructure. Users of PKINIT must understand security policies and procedures appropriate to the use of Public Key Infrastructures [RFC3280].

In order to trust a KDC certificate that is certified by a CA as a KDC certificate for a target realm (for example, by asserting the TGS name of that Kerberos realm as an id-pkinit-san SAN and/or restricting the certificate usage by using the id-pkinit-KPKdc EKU, as described in Section 3.2.4), the client MUST verify that the KDC certificate's issuing CA is authorized to issue KDC certificates for that target realm. Otherwise, the binding between the KDC certificate and the KDC of the target realm is not established.

How to validate this authorization is a matter of local policy. A way to achieve this is the configuration of specific sets of intermediary CAs and trust anchors, one of which must be on the KDC certificate's certification path [RFC3280], and, for each CA or trust anchor, the realms for which it is allowed to issue certificates.

In addition, if any CA that is trusted to issue KDC certificates can also issue other kinds of certificates, then local policy must be able to distinguish between them; for example, it could require that KDC certificates contain the id-pkinit-KPKdc EKU or that the realm be specified with the id-pkinit-san SAN.

It is the responsibility of the PKI administrators for an organization to ensure that KDC certificates are only issued to KDCs, and that clients can ascertain this using their local policy.

Standard Kerberos allows the possibility of interactions between cryptosystems of varying strengths; this document adds interactions with public-key cryptosystems to Kerberos. Some administrative policies may allow the use of relatively weak public keys. When using such weak asymmetric keys to protect/exchange stronger symmetric Keys, the attack resistant strength of the overall system is no better than that of these weak keys [RFC3766].

PKINIT requires that keys for symmetric cryptosystems be generated. Some such systems contain "weak" keys. For recommendations regarding these weak keys, see [RFC4120].

PKINIT allows the use of the same RSA key pair for encryption and signing when doing RSA encryption-based key delivery. This is not recommended usage of RSA keys [RFC3447]; by using DH-based key delivery, this is avoided.

Care should be taken in how certificates are chosen for the purposes of authentication using PKINIT. Some local policies may require that key escrow be used for certain certificate types. Deployers of PKINIT should be aware of the implications of using certificates that have escrowed keys for the purposes of authentication. Because signing-only certificates are normally not escrowed, by using DH-based key delivery this is avoided.

PKINIT does not provide for a "return routability" test to prevent attackers from mounting a denial-of-service attack on the KDC by causing it to perform unnecessary and expensive public-key operations. Strictly speaking, this is also true of standard Kerberos, although the potential cost is not as great, because standard Kerberos does not make use of public-key cryptography. By using DH-based key delivery and reusing DH keys, the necessary crypto processing cost per request can be minimized.

When the Diffie-Hellman key exchange method is used, additional pre-authentication data [RFC4120] (in addition to the PA_PK_AS_REQ, as defined in this specification) is not bound to the AS_REQ by the mechanisms discussed in this specification (meaning it may be dropped or added by attackers without being detected by either the client or the KDC). Designers of additional pre-authentication data should take that into consideration if such additional pre-authentication data can be used in conjunction with the PA_PK_AS_REQ. The future work of the Kerberos working group is expected to update the hash algorithms specified in this document and provide a generic mechanism to bind additional pre-authentication data with the accompanying AS_REQ.

The key usage number 6 used by the asChecksum field is also used for the authenticator checksum (cksum field of AP-REQ) contained in the PA-TGS-REQ preauthentication data contained in a TGS-REQ [RFC4120]. This conflict is present for historical reasons; the reuse of key usage numbers is strongly discouraged.

5. Acknowledgements

The following people have made significant contributions to this document: Paul Leach, Stefan Santesson, Sam Hartman, Love Hornquist Astrand, Ken Raeburn, Nicolas Williams, John Wray, Tom Yu, Jeffrey Hutzelman, David Cross, Dan Simon, Karthik Jaganathan, Chaskiel M Grundman, and Jeffrey Altman.

Andre Scedrov, Aaron D. Jaggard, Iliano Cervesato, Joe-Kai Tsay, and Chris Walstad discovered a binding issue between the AS-REQ and AS-REP in draft -26; the asChecksum field was added as the result.

Special thanks to Clifford Neuman, Matthew Hur, Ari Medvinsky, Sasha Medvinsky, and Jonathan Trostle who wrote earlier versions of this document.

The authors are indebted to the Kerberos working group chair, Jeffrey Hutzelman, who kept track of various issues and was enormously helpful during the creation of this document.

Some of the ideas on which this document is based arose during discussions over several years between members of the SAAG, the IETF CAT working group, and the PSRG, regarding integration of Kerberos and SPX. Some ideas have also been drawn from the DASS system. These changes are by no means endorsed by these groups. This is an attempt to revive some of the goals of those groups, and this document approaches those goals primarily from the Kerberos perspective.

Lastly, comments from groups working on similar ideas in DCE have been invaluable.

6. References

6.1. Normative References

[IEEE1363] IEEE, "Standard Specifications for Public Key Cryptography", IEEE 1363, 2000.

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

[RFC2412] Orman, H., "The OAKLEY Key Determination Protocol", RFC 2412, November 1998.

[RFC2631] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, June 1999.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[RFC3370] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, August 2002.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003.

[RFC3560] Housley, R., "Use of the RSAES-OAEP Key Transport Algorithm in Cryptographic Message Syntax (CMS)", RFC 3560, July 2003.

[RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, April 2004.

[RFC3852] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

[RFC3962] Raeburn, K., "Advanced Encryption Standard (AES) Encryption for Kerberos 5", RFC 3962, February 2005.

[RFC4086] Eastlake, D., 3rd, Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

[X680] ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002, Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.

[X690] ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002, Information technology - ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

6.2. Informative References

[ODL99] Odlyzko, A., "Discrete logarithms: The past and the future, Designs, Codes, and Cryptography (1999)". April 1999.

[RFC4121] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, July 2005.

[RFC4158] Cooper, M., Dzambasow, Y., Hesse, P., Joseph, S., and R. Nicholas, "Internet X.509 Public Key Infrastructure: Certification Path Building", RFC 4158, September 2005.

Appendix A. PKINIT ASN.1 Module

```
KerberosV5-PK-INIT-SPEC {
        iso(1) identified-organization(3) dod(6) internet(1)
        security(5) kerberosV5(2) modules(4) pkinit(5)
} DEFINITIONS EXPLICIT TAGS ::= BEGIN

IMPORTS

    SubjectPublicKeyInfo, AlgorithmIdentifier
        FROM PKIX1Explicit88 { iso (1)
          identified-organization (3) dod (6) internet (1)
          security (5) mechanisms (5) pkix (7) id-mod (0)
          id-pkix1-explicit (18) }
          -- As defined in RFC 3280.

    KerberosTime, PrincipalName, Realm, EncryptionKey, Checksum
        FROM KerberosV5Spec2 { iso(1) identified-organization(3)
          dod(6) internet(1) security(5) kerberosV5(2)
          modules(4) krb5spec2(2) };
          -- as defined in RFC 4120.

id-pkinit OBJECT IDENTIFIER ::=
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) kerberosv5(2) pkinit (3) }

id-pkinit-authData      OBJECT IDENTIFIER  ::= { id-pkinit 1 }
id-pkinit-DHKeyData     OBJECT IDENTIFIER  ::= { id-pkinit 2 }
id-pkinit-rkeyData      OBJECT IDENTIFIER  ::= { id-pkinit 3 }
id-pkinit-KPClientAuth  OBJECT IDENTIFIER  ::= { id-pkinit 4 }
id-pkinit-KPKdc         OBJECT IDENTIFIER  ::= { id-pkinit 5 }

id-pkinit-san OBJECT IDENTIFIER ::=
  { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2)
    x509SanAN (2) }

pa-pk-as-req INTEGER ::=                 16
pa-pk-as-rep INTEGER ::=                 17

ad-initial-verified-cas INTEGER ::=       9

td-trusted-certifiers INTEGER ::=       104
td-invalid-certificates INTEGER ::=     105
td-dh-parameters INTEGER ::=            109

PA-PK-AS-REQ ::= SEQUENCE {
   signedAuthPack          [0] IMPLICIT OCTET STRING,
            -- Contains a CMS type ContentInfo encoded
```

```
                -- according to [RFC3852].
                -- The contentType field of the type ContentInfo
                -- is id-signedData (1.2.840.113549.1.7.2),
                -- and the content field is a SignedData.
                -- The eContentType field for the type SignedData is
                -- id-pkinit-authData (1.3.6.1.5.2.3.1), and the
                -- eContent field contains the DER encoding of the
                -- type AuthPack.
                -- AuthPack is defined below.
      trustedCertifiers       [1] SEQUENCE OF
                   ExternalPrincipalIdentifier OPTIONAL,
                -- Contains a list of CAs, trusted by the client,
                -- that can be used to certify the KDC.
                -- Each ExternalPrincipalIdentifier identifies a CA
                -- or a CA certificate (thereby its public key).
                -- The information contained in the
                -- trustedCertifiers SHOULD be used by the KDC as
                -- hints to guide its selection of an appropriate
                -- certificate chain to return to the client.
      kdcPkId                 [2] IMPLICIT OCTET STRING
                                  OPTIONAL,
                -- Contains a CMS type SignerIdentifier encoded
                -- according to [RFC3852].
                -- Identifies, if present, a particular KDC
                -- public key that the client already has.
      ...
   }

   DHNonce ::= OCTET STRING

   ExternalPrincipalIdentifier ::= SEQUENCE {
      subjectName            [0] IMPLICIT OCTET STRING OPTIONAL,
                -- Contains a PKIX type Name encoded according to
                -- [RFC3280].
                -- Identifies the certificate subject by the
                -- distinguished subject name.
                -- REQUIRED when there is a distinguished subject
                -- name present in the certificate.
     issuerAndSerialNumber   [1] IMPLICIT OCTET STRING OPTIONAL,
                -- Contains a CMS type IssuerAndSerialNumber encoded
                -- according to [RFC3852].
                -- Identifies a certificate of the subject.
                -- REQUIRED for TD-INVALID-CERTIFICATES and
                -- TD-TRUSTED-CERTIFIERS.
     subjectKeyIdentifier    [2] IMPLICIT OCTET STRING OPTIONAL,
                -- Identifies the subject's public key by a key
                -- identifier.  When an X.509 certificate is
                -- referenced, this key identifier matches the X.509
```

```
                -- subjectKeyIdentifier extension value.  When other
                -- certificate formats are referenced, the documents
                -- that specify the certificate format and their use
                -- with the CMS must include details on matching the
                -- key identifier to the appropriate certificate
                -- field.
                -- RECOMMENDED for TD-TRUSTED-CERTIFIERS.
    ...
}

AuthPack ::= SEQUENCE {
    pkAuthenticator         [0] PKAuthenticator,
    clientPublicValue       [1] SubjectPublicKeyInfo OPTIONAL,
                -- Type SubjectPublicKeyInfo is defined in
                -- [RFC3280].
                -- Specifies Diffie-Hellman domain parameters
                -- and the client's public key value [IEEE1363].
                -- The DH public key value is encoded as a BIT
                -- STRING according to [RFC3279].
                -- This field is present only if the client wishes
                -- to use the Diffie-Hellman key agreement method.
    supportedCMSTypes       [2] SEQUENCE OF AlgorithmIdentifier
                                OPTIONAL,
                -- Type AlgorithmIdentifier is defined in
                -- [RFC3280].
                -- List of CMS algorithm [RFC3370] identifiers
                -- that identify key transport algorithms, or
                -- content encryption algorithms, or signature
                -- algorithms supported by the client in order of
                -- (decreasing) preference.
    clientDHNonce           [3] DHNonce OPTIONAL,
                -- Present only if the client indicates that it
                -- wishes to reuse DH keys or to allow the KDC to
                -- do so.
    ...
}

PKAuthenticator ::= SEQUENCE {
    cusec                   [0] INTEGER (0..999999),
    ctime                   [1] KerberosTime,
                -- cusec and ctime are used as in [RFC4120], for
                -- replay prevention.
    nonce                   [2] INTEGER (0..4294967295),
                -- Chosen randomly; this nonce does not need to
                -- match with the nonce in the KDC-REQ-BODY.
    paChecksum              [3] OCTET STRING OPTIONAL,
                -- MUST be present.
                -- Contains the SHA1 checksum, performed over
```

```
                -- KDC-REQ-BODY.
      ...
   }

   TD-TRUSTED-CERTIFIERS ::= SEQUENCE OF
                   ExternalPrincipalIdentifier
                -- Identifies a list of CAs trusted by the KDC.
                -- Each ExternalPrincipalIdentifier identifies a CA
                -- or a CA certificate (thereby its public key).

   TD-INVALID-CERTIFICATES ::= SEQUENCE OF
                   ExternalPrincipalIdentifier
                -- Each ExternalPrincipalIdentifier identifies a
                -- certificate (sent by the client) with an invalid
                -- signature.

   KRB5PrincipalName ::= SEQUENCE {
       realm                   [0] Realm,
       principalName           [1] PrincipalName
   }

   AD-INITIAL-VERIFIED-CAS ::= SEQUENCE OF
                   ExternalPrincipalIdentifier
                -- Identifies the certification path based on which
                -- the client certificate was validated.
                -- Each ExternalPrincipalIdentifier identifies a CA
                -- or a CA certificate (thereby its public key).

   PA-PK-AS-REP ::= CHOICE {
      dhInfo                   [0] DHRepInfo,
                -- Selected when Diffie-Hellman key exchange is
                -- used.
      encKeyPack               [1] IMPLICIT OCTET STRING,
                -- Selected when public key encryption is used.
                -- Contains a CMS type ContentInfo encoded
                -- according to [RFC3852].
                -- The contentType field of the type ContentInfo is
                -- id-envelopedData (1.2.840.113549.1.7.3).
                -- The content field is an EnvelopedData.
                -- The contentType field for the type EnvelopedData
                -- is id-signedData (1.2.840.113549.1.7.2).
                -- The eContentType field for the inner type
                -- SignedData (when unencrypted) is
                -- id-pkinit-rkeyData (1.3.6.1.5.2.3.3) and the
                -- eContent field contains the DER encoding of the
                -- type ReplyKeyPack.
                -- ReplyKeyPack is defined below.
      ...
```

```
}

DHRepInfo ::= SEQUENCE {
   dhSignedData            [0] IMPLICIT OCTET STRING,
            -- Contains a CMS type ContentInfo encoded according
            -- to [RFC3852].
            -- The contentType field of the type ContentInfo is
            -- id-signedData (1.2.840.113549.1.7.2), and the
            -- content field is a SignedData.
            -- The eContentType field for the type SignedData is
            -- id-pkinit-DHKeyData (1.3.6.1.5.2.3.2), and the
            -- eContent field contains the DER encoding of the
            -- type KDCDHKeyInfo.
            -- KDCDHKeyInfo is defined below.
   serverDHNonce           [1] DHNonce OPTIONAL,
            -- Present if and only if dhKeyExpiration is
            -- present.
   ...
}

KDCDHKeyInfo ::= SEQUENCE {
   subjectPublicKey        [0] BIT STRING,
            -- The KDC's DH public key.
            -- The DH public key value is encoded as a BIT
            -- STRING according to [RFC3279].
   nonce                   [1] INTEGER (0..4294967295),
            -- Contains the nonce in the pkAuthenticator field
            -- in the request if the DH keys are NOT reused,
            -- 0 otherwise.
   dhKeyExpiration         [2] KerberosTime OPTIONAL,
            -- Expiration time for KDC's key pair,
            -- present if and only if the DH keys are reused.
            -- If present, the KDC's DH public key MUST not be
            -- used past the point of this expiration time.
            -- If this field is omitted then the serverDHNonce
            -- field MUST also be omitted.
   ...
}

ReplyKeyPack ::= SEQUENCE {
   replyKey                [0] EncryptionKey,
            -- Contains the session key used to encrypt the
            -- enc-part field in the AS-REP, i.e., the
            -- AS reply key.
   asChecksum              [1] Checksum,
           -- Contains the checksum of the AS-REQ
           -- corresponding to the containing AS-REP.
           -- The checksum is performed over the type AS-REQ.
```

```
                  -- The protocol key [RFC3961] of the checksum is the
                  -- replyKey and the key usage number is 6.
                  -- If the replyKey’s enctype is "newer" [RFC4120]
                  -- [RFC4121], the checksum is the required
                  -- checksum operation [RFC3961] for that enctype.
                  -- The client MUST verify this checksum upon receipt
                  -- of the AS-REP.
          ...
       }

       TD-DH-PARAMETERS ::= SEQUENCE OF AlgorithmIdentifier
                   -- Each AlgorithmIdentifier specifies a set of
                   -- Diffie-Hellman domain parameters [IEEE1363].
                   -- This list is in decreasing preference order.
       END
```

Appendix B. Test Vectors

Function octetstring2key() is defined in Section 3.2.3.1. This section describes a few sets of test vectors that would be useful for implementers of octetstring2key().

Set 1:
=====
Input octet string x is:

```
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Output of K-truncate() when the key size is 32 octets:

```
  5e e5 0d 67 5c 80 9f e5 9e 4a 77 62 c5 4b 65 83
  75 47 ea fb 15 9b d8 cd c7 5f fc a5 91 1e 4c 41
```

```
   Set 2:
   =====
   Input octet string x is:

     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

   Output of K-truncate() when the key size is 32 octets:

     ac f7 70 7c 08 97 3d df db 27 cd 36 14 42 cc fb
     a3 55 c8 88 4c b4 72 f3 7d a6 36 d0 7d 56 78 7e


   Set 3:
   ======
   Input octet string x is:

     00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
     10 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e
     0f 10 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d
     0e 0f 10 00 01 02 03 04 05 06 07 08 09 0a 0b 0c
     0d 0e 0f 10 00 01 02 03 04 05 06 07 08 09 0a 0b
     0c 0d 0e 0f 10 00 01 02 03 04 05 06 07 08 09 0a
     0b 0c 0d 0e 0f 10 00 01 02 03 04 05 06 07 08 09
     0a 0b 0c 0d 0e 0f 10 00 01 02 03 04 05 06 07 08

   Output of K-truncate() when the key size is 32 octets:

     c4 42 da 58 5f cb 80 e4 3b 47 94 6f 25 40 93 e3
     73 29 d9 90 01 38 0d b7 83 71 db 3a cf 5c 79 7e


   Set 4:
   =====
   Input octet string x is:

     00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
     10 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e
     0f 10 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d
     0e 0f 10 00 01 02 03 04 05 06 07 08 09 0a 0b 0c
     0d 0e 0f 10 00 01 02 03 04 05 06 07 08
```

Output of K-truncate() when the key size is 32 octets:

```
00 53 95 3b 84 c8 96 f4 eb 38 5c 3f 2e 75 1c 4a
59 0e d6 ff ad ca 6f f6 4f 47 eb eb 8d 78 0f fc
```

Appendix C. Miscellaneous Information about Microsoft Windows PKINIT Implementations

Earlier revisions of the PKINIT I-D were implemented in various releases of Microsoft Windows and deployed in fairly large numbers. To enable the community to interoperate better with systems running those releases, the following information may be useful.

KDC certificates issued by Windows 2000 Enterprise CAs contain a dNSName SAN with the DNS name of the host running the KDC, and the id-kp-serverAuth EKU [RFC3280].

KDC certificates issued by Windows 2003 Enterprise CAs contain a dNSName SAN with the DNS name of the host running the KDC, the id-kp-serverAuth EKU, and the id-ms-kp-sc-logon EKU.

It is anticipated that the next release of Windows is already too far along to allow it to support the issuing KDC certificates with id-pkinit-san SAN as specified in this RFC. Instead, they will have a dNSName SAN containing the domain name of the KDC, and the intended purpose of these KDC certificates will be restricted by the presence of the id-pkinit-KPKdc EKU and id-kp-serverAuth EKU.

In addition to checking that the above are present in a KDC certificate, Windows clients verify that the issuer of the KDC certificate is one of a set of allowed issuers of such certificates, so those wishing to issue KDC certificates need to configure their Windows clients appropriately.

Client certificates accepted by Windows 2000 and Windows 2003 Server KDCs must contain an id-ms-san-sc-logon-upn (1.3.6.1.4.1.311.20.2.3) SAN and the id-ms-kp-sc-logon EKU. The id-ms-san-sc-logon-upn SAN contains a UTF8-encoded string whose value is that of the Directory Service attribute UserPrincipalName of the client account object, and the purpose of including the id-ms-san-sc-logon-upn SAN in the client certificate is to validate the client mapping (in other words, the client’s public key is bound to the account that has this UserPrincipalName value).

It should be noted that all Microsoft Kerberos realm names are domain-style realm names and strictly in uppercase. In addition, the UserPrincipalName attribute is globally unique in Windows 2000 and Windows 2003.

Authors' Addresses

Larry Zhu
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

EMail: lzhu@microsoft.com

Brian Tung
Aerospace Corporation
2350 E. El Segundo Blvd.
El Segundo, CA 90245
US

EMail: brian@aero.org

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).