NETWORK WORKING GROUP
Internet-Draft
Expires: August 4, 2005

B. Tung
USC Information Sciences Institute
L. Zhu
Microsoft Corporation
January 31, 2005

Public Key Cryptography for Initial Authentication in Kerberos
draft-ietf-cat-kerberos-pk-init-23

Status of this Memo

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 passing digital certificates and associated authenticators in pre-authentication data fields.

Table of Contents

1. Introduction

A client typically authenticates itself to a service in Kerberos using three distinct though related exchanges. First, the client requests a ticket-granting ticket (TGT) from the Kerberos authentication server (AS). Then, it uses the TGT to request a service ticket from the Kerberos ticket-granting server (TGS). Usually, the AS and TGS are integrated in a single device known as a Kerberos Key Distribution Center, or KDC. (In this document, we will refer to both the AS and the TGS as the KDC.) Finally, the client uses the service ticket to authenticate itself to the service.

The advantage afforded by the 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.

As defined in [CLAR], Kerberos authentication exchanges use symmetric-key cryptography, in part for performance. One disadvantage of using symmetric-key cryptography is that the keys must be shared, so that before a client can authenticate itself, he must already be registered with the KDC.

Conversely, public-key cryptography (in conjunction with an established Public Key Infrastructure) permits authentication without prior registration with a KDC. Adding it to Kerberos allows the widespread use of Kerberized applications by clients without requiring them to register first with a KDC--a requirement that has no inherent security benefit.

As noted above, a convenient and efficient place to introduce public-key cryptography into Kerberos is in the initial authentication exchange. This document describes the methods and

data formats for integrating public-key cryptography into Kerberos initial authentication.

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].

3. Extensions

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

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

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] 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 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: AES256-CTS-HMAC-SHA1-96 etype [KCRYPTO].
- Signature algorithm: sha-1WithRSAEncryption [RFC3279].
- KDC AS reply key delivery method: ephemeral-ephemeral Diffie-Hellman exchange (Diffie-Hellman keys are not cached).

3.1.2 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_KDC_NOT_TRUSTED	63
KDC_ERR_INVALID_SIG	64
KDC_ERR_KEY_SIZE	65
KDC_ERR_CERTIFICATE_MISMATCH	66
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

PKINIT uses the following typed data types for errors:

TD-TRUSTED-CERTIFIERS	104
TD-CERTIFICATE-INDEX	105
TD-DH-PARAMETERS	109

PKINIT defines the following encryption types, for use in the AS-REQ message (to indicate acceptance of the corresponding encryption Object Identifiers (OIDs) in PKINIT):

dsaWithSHA1-CmsOID		9
md5WithRSAEncryption-CmsOID		10
sha1WithRSAEncryption-CmsOID		11
rc2CBC-EnvOID		12
rsaEncryption-EnvOID	(PKCS1 v1.5)	13
rsaES-OAEP-EnvOID	(PKCS1 v2.0)	14
des-ede3-cbc-EnvOID		15

The above encryption types are used by the client only within the KDC-REQ-BODY to indicate which Cryptographic Message Syntax (CMS) [RFC3852] algorithms it supports. Their use within Kerberos EncryptedData structures is not specified by this document.

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

Appendix A.

All structures defined in or imported into this document MUST be encoded using Distinguished Encoding Rules (DER) [X690]. All data structures wrapped in OCTET STRINGs must be encoded according to the rules specified in corresponding specifications.

Interoperability note: Some implementations may not be able to decode CMS objects encoded with BER but not DER; specifically, they may not be able to decode infinite length encodings. To maximize interoperability, implementers SHOULD encode CMS objects used in PKINIT with DER.

3.1.3 Algorithm Identifiers

PKINIT does not define, but does make use of, the following algorithm identifiers.

PKINIT uses the following algorithm identifier for Diffie-Hellman key agreement [RFC3279]:

```
     dhpublicnumber
```

PKINIT uses the following signature algorithm identifiers [RFC3279]:

```
    sha-1WithRSAEncryption (RSA with SHA1)
    md5WithRSAEncryption   (RSA with MD5)
    id-dsa-with-sha1       (DSA with SHA1)
```

PKINIT uses the following encryption algorithm identifiers [RFC3447] for encrypting the temporary key with a public key:

```
    rsaEncryption          (PKCS1 v1.5)
    id-RSAES-OAEP          (PKCS1 v2.0)
```

PKINIT uses the following algorithm identifiers [RFC3370][RFC3565] for encrypting the reply key with the temporary key:

```
    des-ede3-cbc           (three-key 3DES, CBC mode)
    rc2-cbc                (RC2, CBC mode)
    id-aes256-CBC          (AES-256, CBC mode)
```

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 [CLAR]; in addition, a pre-authentication field contains data signed by the client's private signature key, as follows:

```
    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-pkauthdata (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 TrustedCA OPTIONAL,
                -- A list of CAs, trusted by the client, that can
                -- be used to validate KDC certificates.
   kdcCert                 [2] IMPLICIT OCTET STRING
                               OPTIONAL,
                -- Contains a CMS type IssuerAndSerialNumber encoded
                -- according to [RFC3852].
                -- Identifies a particular KDC certificate, if the
                -- client already has it.
   ...
}

DHNonce ::= OCTET STRING

TrustedCA ::= CHOICE {
   caName                  [1] IMPLICIT OCTET STRING,
                -- Contains a PKIX type Name encoded according to
                -- [RFC3280].
   issuerAndSerial         [2] IMPLICIT OCTET STRING,
                -- Contains a CMS type IssuerAndSerialNumber encoded
                -- according to [RFC3852].
                -- Identifies a specific CA certificate.
   ...
}

AuthPack ::= SEQUENCE {
   pkAuthenticator         [0] PKAuthenticator,
   clientPublicValue       [1] SubjectPublicKeyInfo OPTIONAL,
                -- Defined in [RFC3280].
                -- Present only if the client wishes to use the
```



```
                -- Diffie-Hellman key agreement method.
   supportedCMSTypes       [2] SEQUENCE OF AlgorithmIdentifier
                               OPTIONAL,
```

```
                -- List of CMS encryption types supported by
                -- client in order of (decreasing) preference.
        clientDHNonce           [3] DHNonce OPTIONAL,
                -- Present only if the client indicates that it
                -- wishes to cache 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 [CLAR], 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,
                -- Contains the SHA1 checksum, performed over
                -- KDC-REQ-BODY.
        ...
    }
```

The ContentInfo [RFC3852] structure for the signedAuthPack field 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-pkauthdata: { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2) pkinit(3) pkauthdata(1) }.

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 certificates field of the type SignedData contains the client's certificate and additional certificates intended to facilitate certification path construction, so that the KDC can validate the client's certificate and verify the signature over the type AuthPack. The certificates field MUST NOT contain

"root" CA certificates.

6. 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. For the Diffie-Hellman key agreement method, implementations MUST support Oakley 1024-bit MODP well-known group 2 [RFC2412] and SHOULD support Oakley 2048-bit MODP well-known group 14 and Oakley 4096-bit MODP well-known group 16 [RFC3526]. They MAY support Oakley 185-bit EC2N group 4 [RFC2412]. The Diffie-Hellman group size should be chosen so as to provide sufficient cryptographic security. The exponents should have at least twice as many bits as the symmetric keys that will be derived from them [ODL99].

7. The client may wish to cache DH keys or to allow the KDC to do so. If so, then the client includes the clientDHNonce field. This nonce string needs to be as long as the longest key length of the symmetric key types that the client supports. This nonce MUST be chosen randomly.

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 looks for the client's certificate in the signedAuthPack (based on the signerInfo) and validate this certificate.

If the KDC cannot find a certification path to validate the client's certificate, it sends back an error of type KDC_ERR_CANT_VERIFY_CERTIFICATE. The accompanying e-data for this error is a TYPED-DATA (as defined in [CLAR]). For this error, the data-type is TD-TRUSTED-CERTIFIERS, and the data-value is the DER encoding of

```
TrustedCertifiers ::= SEQUENCE OF OCTET STRING
            -- The OCTET STRING contains a PKIX type Name encoded
            -- according to [RFC3280].
```

If, while processing the certification path, the KDC determines that the signature on one of the certificates in the signedAuthPack is invalid, it returns an error of type KDC_ERR_INVALID_CERTIFICATE. The accompanying e-data for this error is a TYPED-DATA, whose data-type is TD-CERTIFICATE-INDEX, and whose data-value is the DER encoding of the index into the CertificateSet field, ordered as sent

by the client:


```
CertificateIndex ::= OCTET STRING
            -- Contains a CMS type IssuerAndSerialNumber encoded
            -- according to [RFC3852].
            -- IssuerAndSerialNumber of certificate with an
            -- invalid signature.
```

If more than one certificate signature is invalid, the KDC MAY send one TYPED-DATA per invalid signature.

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

In addition to validating the client's certificate, the KDC MUST also check that this certificate properly maps to the client's principal name as specified in the AS-REQ as follows:

1. If the KDC has its own mapping from the name in the client's certificate to a Kerberos name, it uses that Kerberos name.

2. Otherwise, if the client's certificate contains a SubjectAltName extension with a Kerberos name in the otherName field, it uses that name.

   The otherName field (of type AnotherName) in the SubjectAltName extension MUST contain KRB5PrincipalName as defined below.

   The type-id field of the type AnotherName is id-pksan:

```
id-pksan OBJECT IDENTIFIER ::=
  { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2)
    x509-sanan (2) }
```

   The value field of the type AnotherName is the DER encoding of the following ASN.1 type:

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

If the KDC does not have its own mapping and there is no Kerberos name present in the client's certificate, or if the name in the

request does not match the name in the certificate (including the realm name), the KDC MUST return error code KDC_ERR_CLIENT_NAME_MISMATCH. There is no accompanying e-data for this error.

Even if the client's certificate is validated and it 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-pkekuoid in the extensions field of the client's certificate:

```
    id-pkekuoid OBJECT IDENTIFIER ::=
      { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2)
        pkinit(3) pkekuoid(4) }
           -- PKINIT client authentication.
           -- Key usage bits that may be consistent: digitalSignature
           -- nonRepudiation, and (keyEncipherment or keyAgreement).
```

As a matter of local policy, the KDC may decide to reject requests on the basis of the absence or presence of specific EKU OIDs. KDCs implementing this requirement SHOULD also accept the EKU KeyPurposeId id-ms-sc-logon (1.3.6.1.4.1.311.20.2.2) as meeting the requirement, as there are a large number of client certificates deployed for use with PKINIT which have this EKU.

The KDC MUST return the error code KDC_ERR_CLIENT_NOT_TRUSTED if the client's certificate is not accepted.

Once the client's certificate is accepted, the KDC can then verify the client's signature over the type AuthPack according to [RFC3852]. If the signature fails to verify, the KDC MUST return error

KDC_ERR_INVALID_SIG. There is no accompanying e-data for this error.

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 clock skew times in [CLAR] 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 error code KDC_ERR_KEY_SIZE. The accompanying e-data is a TYPED-DATA, whose data-type is TD-DH-PARAMETERS, and whose data-value is the DER encoding of the following:


```
TD-DH-PARAMETERS ::= SEQUENCE OF DomainParameters
            -- Type DomainParameters is defined in [RFC3279].
            -- Contains a list of Diffie-Hellman group
            -- parameters in decreasing preference order.
```

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

The KDC MUST return error code KDC_ERR_CERTIFICATE_MISMATCH if the client included a kdcCert field in the PA-PK-AS-REQ and the KDC does not have the corresponding certificate.

The KDC MUST return error code KDC_ERR_KDC_NOT_TRUSTED if the client did not include a kdcCert field, but did include a trustedCertifiers field, and the KDC does not possesses a certificate issued by one of the listed certifiers.

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 of type KRB5KDC_ERR_ETYPE_NOSUPP.

3.2.3 Generation of KDC Reply

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

The KDC MUST set the initial flag and include an authorization data of type AD-INITIAL-VERIFIED-CAS in the issued ticket. The value is an OCTET STRING containing the DER encoding of InitialVerifiedCAs:

```
InitialVerifiedCAs ::= SEQUENCE OF SEQUENCE {
   ca                      [0] IMPLICIT OCTET STRING,
            -- Contains a PKIX type Name encoded according to
            -- [RFC3280].
   validated               [1] BOOLEAN,
   ...
}
```

The KDC MAY wrap any AD-INITIAL-VERIFIED-CAS data in AD-IF-RELEVANT containers if the list of CAs satisfies the KDC's realm's policy (this corresponds to the TRANSITED-POLICY-CHECKED ticket flag [CLAR]). Furthermore, any TGS must copy such authorization data from tickets used in a PA-TGS-REQ of the TGS-REQ to the resulting ticket, including the AD-IF-RELEVANT container, if present.

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 checking the transited field when the TRANSITED-POLICY-CHECKED flag has not been set [CLAR].) 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.

The AS-REP is otherwise unchanged from [CLAR]. 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:

```
PA-PK-AS-REP ::= CHOICE {
   dhInfo                  [0] DHRepInfo,
   encKeyPack              [1] IMPLICIT OCTET STRING,
            -- 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-pkrkeydata
                -- (1.2.840.113549.1.7.3) 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-pkdhkeydata (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,
                -- KDC's public key, y = g^x mod p.
                -- MUST be ASN.1 encoded as an INTEGER;
                -- This encoding is then used as the contents
                -- (i.e., the value) of this BIT STRING field.
        nonce                   [1] INTEGER (0..4294967295),
                -- Contains the nonce in the PKAuthenticator of the
                -- request if cached DH keys are NOT used,
```

```
                -- 0 otherwise.
        dhKeyExpiration         [2] KerberosTime OPTIONAL,
                -- Expiration time for KDC's cached values, present
                -- if and only if cached DH keys are used.  If this
                -- field is omitted then the serverDHNonce field
                -- MUST also be omitted.
        ...
    }
```

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-pkdhkeydata: { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2) pkinit(3) pkdhkeydata(2) }.

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

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

5. The certificates field of the type SignedData contains the KDC's certificate and additional certificates intended to facilitate certification path construction, so that the client can validate the KDC's certificate and verify the KDC's signature over the

type KDCDHKeyInfo. This field may only be left empty if the client did include a kdcCert field in the PA-PK-AS-REQ, indicating that the client already has the KDC's certificate. The certificates field MUST NOT contain "root" CA certificates.

6. 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 dhKeyExperiation 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 reply key for use to decrypt the KDC reply [CLAR] is derived as follows:

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

```
DHKey = g^(xb * xa) mod p
```

where xb and xa are the KDC's and client's private exponents, respectively. DHKey, padded first with leading zeros as needed to make it as long as the modulus p, is represented as a string of octets in big-endian order (such that the size of DHKey in octets is the size of the modulus p).

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

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 defined in the kcrypto profile [KCRYPTO] for the enctype of the reply key.

4. When cached DH keys are used, 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 reply key k is:

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

3.2.3.2 Using Public Key Encryption

In this case, the PA-PK-AS-REP contains a ContentInfo structure wrapped in an OCTET STRING. The reply key for use to decrypt the KDC reply [CLAR] is encrypted in the encKeyPack field, which contains data of type ReplyKeyPack:

```
ReplyKeyPack ::= SEQUENCE {
   replyKey                [0] EncryptionKey,
            -- Contains the session key used to encrypt the
            -- enc-part field in the AS-REP.
   nonce                   [1] INTEGER (0..4294967295),
            -- Contains the nonce in the PKAuthenticator of the
            -- request.
   ...
}
```

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-pkrkeydata: { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2) pkinit(3) pkrkeydata(3) }.

4. The eContent field for the inner type SignedData contains the DER encoding of the type ReplyKeyPack.

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

6. The certificates field of the inner type SignedData contains the KDC's certificate and additional certificates intended to facilitate certification path construction, so that the client can validate the KDC's certificate and verify the KDC's signature over the type ReplyKeyPack. This field may only be left empty if the client included a kdcCert field in the PA-PK-AS-REQ, indicating that the client already has the KDC's certificate. The certificates field MUST NOT contain "root" CA certificates.

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

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

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 shared key using the same procedure used by the KDC as defined in Section 3.2.3.1. Otherwise, the message contains an encKeyPack, and the client decrypts and verifies the temporary encryption key.

In either case, the client MUST validate the KDC's certificate and verify the signature in the SignedData according to [RFC3852]. Unless the client can otherwise prove that the KDC's certificate is for the target KDC (i.e., the subject distinguished name in the KDC certificate is bound to the hostname or IP address of the KDC authenticating the client), it MUST do the following to verify the responder's identity:

1. The client checks to see if the included certificate contains a Subject Alternative Name extension [RFC3280] carrying a dNSName or an iPAddress (if the KDC is specified by an IP address instead of a name). If it does, it MUST check to see if that name value matches that of the KDC it believes it is communicating with, with

matching rules specified in [RFC3280].

2. The client verifies that the KDC's certificate MUST contain the EKU KeyPurposeId [RFC3280] id-pkkdcekuoid:


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

If all applicable checks are satisfied, the client then decrypts the enc-part of the KDC-REP in the AS_REP with the resulting key, and then proceeds as described in [CLAR].

3.3 KDC Indication of PKINIT Support

If pre-authentication is required, but was not present in the request, per [CLAR] 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 a PA-PK-AS-REQ element 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 of PA-PK-AS-REQ entry 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

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.

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. Using such keys to wrap data encrypted under stronger conventional cryptosystems may be inappropriate.

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

PKINIT uses 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.

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.

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.

5. Acknowledgements

The following people have made significant contributions to this draft: Paul Leach, Phil Hallin, Kelvin Yiu, Sam Hartman, Love Hornquist Astrand, Ken Raeburn, Nicolas Williams, John Wray, Jonathan Trostle, Tom Yu, Jeffrey Hutzelman, David Cross, Dan Simon and Karthik Jaganathan.

Special thanks to Clifford Neuman, Mat Hur and Sasha Medvinsky who wrote earlier versions of this document.

The authors are indebt 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. IANA Considerations

This document has no actions for IANA.

7. References

7.1 Normative References

[CLAR] RFC-Editor: To be replaced by RFC number for draft-ietf-krb-wg-kerberos-clarifications. Work in Progress.

[KCRYPTO] RFC-Editor: To be replaced by RFC number for draft-ietf-

krb-wg-crypto. Work in Progress.

[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.

[RFC3565] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, July 2003.

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

[X690] ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and

Distinguished Encoding Rules (DER), ITU-T Recommendation X.690 (1997) | ISO/IEC International Standard 8825-1:1998.

7.2 Informative References

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

Authors' Addresses

Brian Tung
USC Information Sciences Institute
4676 Admiralty Way Suite 1001, Marina del Rey CA
Marina del Rey, CA 90292
US

Email: brian@isi.edu

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

Email: lzhu@microsoft.com

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.

            DomainParameters
                FROM PKIX1Algorithms88 { iso(1)
                  identified-organization(3) dod(6)
                  internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
                  id-mod-pkix1-algorithms(17) }
                  -- As defined in RFC 3279.

            KerberosTime, TYPED-DATA, PrincipalName, Realm, EncryptionKey
                FROM KerberosV5Spec2 { iso(1) identified-organization(3)
                  dod(6) internet(1) security(5) kerberosV5(2)
                  modules(4) krb5spec2(2) } ;

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

        id-pkauthdata  OBJECT IDENTIFIER  ::= { id-pkinit 1 }
        id-pkdhkeydata OBJECT IDENTIFIER  ::= { id-pkinit 2 }
        id-pkrkeydata  OBJECT IDENTIFIER  ::= { id-pkinit 3 }
        id-pkekuoid    OBJECT IDENTIFIER  ::= { id-pkinit 4 }
        id-pkkdcekuoid OBJECT IDENTIFIER  ::= { id-pkinit 5 }

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

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

        td-trusted-certifiers INTEGER ::=       104
        td-certificate-index 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-pkauthdata (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 TrustedCA OPTIONAL,
            -- A list of CAs, trusted by the client, that can
            -- be used to validate KDC certificates.
   kdcCert                 [2] IMPLICIT OCTET STRING
                               OPTIONAL,
            -- Contains a CMS type IssuerAndSerialNumber encoded
            -- according to [RFC3852].
            -- Identifies a particular KDC certificate, if the
            -- client already has it.
   ...
}

DHNonce ::= OCTET STRING

TrustedCA ::= CHOICE {
   caName                  [1] IMPLICIT OCTET STRING,
            -- Contains a PKIX type Name encoded according to
            -- [RFC3280].
   issuerAndSerial         [2] IMPLICIT OCTET STRING,
            -- Contains a CMS type IssuerAndSerialNumber encoded
            -- according to [RFC3852].
            -- Identifies a specific CA certificate.
   ...
}

AuthPack ::= SEQUENCE {
   pkAuthenticator         [0] PKAuthenticator,
   clientPublicValue       [1] SubjectPublicKeyInfo OPTIONAL,
            -- Defined in [RFC3280].
            -- Present only if the client wishes to use the
            -- Diffie-Hellman key agreement method.
   supportedCMSTypes       [2] SEQUENCE OF AlgorithmIdentifier
                               OPTIONAL,
            -- List of CMS encryption types supported by
            -- client in order of (decreasing) preference.
   clientDHNonce           [3] DHNonce OPTIONAL,
            -- Present only if the client indicates that it
            -- wishes to cache 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 [CLAR], 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,
            -- Contains the SHA1 checksum, performed over
            -- KDC-REQ-BODY.
   ...
}

TrustedCertifiers ::= SEQUENCE OF OCTET STRING
            -- The OCTET STRING contains a PKIX type Name encoded
            -- according to [RFC3280].

CertificateIndex ::= OCTET STRING
            -- Contains a CMS type IssuerAndSerialNumber encoded
            -- according to [RFC3852].

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

InitialVerifiedCAs ::= SEQUENCE OF SEQUENCE {
   ca                      [0] IMPLICIT OCTET STRING,
            -- Contains a PKIX type Name encoded according to
            -- [RFC3280].
   validated               [1] BOOLEAN,
   ...
}

PA-PK-AS-REP ::= CHOICE {
   dhInfo                  [0] DHRepInfo,
   encKeyPack              [1] IMPLICIT OCTET STRING,
            -- 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-pkrkeydata
```

```
                -- (1.2.840.113549.1.7.3) 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-pkdhkeydata (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,
                -- KDC's public key, y = g^x mod p.
                -- MUST be ASN.1 encoded as an INTEGER;
                -- This encoding is then used as the contents
                -- (i.e., the value) of this BIT STRING field.
        nonce                   [1] INTEGER (0..4294967295),
                -- Contains the nonce in the PKAuthenticator of the
                -- request if cached DH keys are NOT used,
                -- 0 otherwise.
        dhKeyExpiration         [2] KerberosTime OPTIONAL,
                -- Expiration time for KDC's cached values, present
                -- if and only if cached DH keys are used.  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.
   nonce                   [1] INTEGER (0..4294967295),
            -- Contains the nonce in the PKAuthenticator of the
            -- request.
```



```
   ...
}

TD-DH-PARAMETERS ::= SEQUENCE OF DomainParameters
            -- Contains a list of Diffie-Hellman group
            -- parameters in decreasing preference order.
END
```

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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