INTERNET-DRAFT
draft-ietf-cat-kerberos-pk-init-16.txt
Updates: RFC 1510bis
expires March 12, 2002

Brian Tung
Clifford Neuman
USC/ISI
Matthew Hur
Microsoft Corporation
Ari Medvinsky
Liberate Technologies
Sasha Medvinsky
Motorola, Inc.
John Wray
Iris Associates, Inc.
Jonathan Trostle

Public Key Cryptography for Initial Authentication in Kerberos

0. Status Of This Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC 2026. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.ietf.org (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

The distribution of this memo is unlimited. It is filed as draft-ietf-cat-kerberos-pk-init-16.txt, and expires March 12, 2002. Please send comments to the authors.

1. Abstract

This document defines extensions (PKINIT) to the Kerberos protocol specification (RFC 1510bis [1]) to provide a method for using public key cryptography during initial authentication. The methods defined specify the ways in which preauthentication data fields and error data fields in Kerberos messages are to be used to transport public key data.

2. Introduction

The popularity of public key cryptography has produced a desire for its support in Kerberos [2]. The advantages provided by public key cryptography include simplified key management (from the Kerberos perspective) and the ability to leverage existing and developing public key certification infrastructures.

Public key cryptography can be integrated into Kerberos in a number of ways. One is to associate a key pair with each realm, which can then be used to facilitate cross-realm authentication; this is the topic of another draft proposal. Another way is to allow users with public key certificates to use them in initial authentication. This is the concern of the current document.

PKINIT utilizes ephemeral-ephemeral Diffie-Hellman keys in combination with RSA keys as the primary, required mechanism. Note that PKINIT supports the use of separate signature and encryption keys.

PKINIT enables access to Kerberos-secured services based on initial authentication utilizing public key cryptography. PKINIT utilizes standard public key signature and encryption data formats within the standard Kerberos messages. The basic mechanism is as follows: The user sends an AS-REQ message to the KDC as before, except that if that user is to use public key cryptography in the initial authentication step, his certificate and a signature accompany the initial request in the preauthentication fields. Upon receipt of this request, the KDC verifies the certificate and issues a ticket granting ticket (TGT) as before, except that the encPart from the AS-REP message carrying the TGT is now encrypted utilizing either a Diffie-Hellman derived key or the user's public key. This message is authenticated utilizing the public key signature of the KDC.

Note that PKINIT does not require the use of certificates. A KDC may store the public key of a principal as part of that principal's record. In this scenario, the KDC is the trusted party that vouches for the principal (as in a standard, non-cross realm, Kerberos environment). Thus, for any principal, the KDC may maintain a symmetric key, a public key, or both.

The PKINIT specification may also be used as a building block for other specifications. PKINIT may be utilized to establish inter-realm keys for the purposes of issuing cross-realm service tickets. It may also be used to issue anonymous Kerberos tickets using the Diffie-Hellman option. Efforts are under way to draft specifications for these two application protocols.

Additionally, the PKINIT specification may be used for direct peer to peer authentication without contacting a central KDC. This application of PKINIT is based on concepts introduced in [6, 7]. For direct client-to-server authentication, the client uses PKINIT to authenticate to the end server (instead of a central KDC), which then issues a ticket for itself. This approach has an advantage over TLS [5] in that the server does not need to save state (cache session keys). Furthermore, an additional benefit is that Kerberos tickets can facilitate delegation (see [6]).

3. Proposed Extensions

This section describes extensions to RFC 1510bis for supporting the use of public key cryptography in the initial request for a ticket granting ticket (TGT).

In summary, the following change to RFC 1510bis is proposed:

* Users may authenticate using either a public key pair or a conventional (symmetric) key. If public key cryptography is used, public key data is transported in preauthentication data fields to help establish identity. The user presents a public key certificate and obtains an ordinary TGT that may be used for subsequent authentication, with such authentication using only conventional cryptography.

Section 3.1 provides definitions to help specify message formats. Section 3.2 describes the extensions for the initial authentication method.

3.1. Definitions

The extensions involve new preauthentication fields; we introduce the following preauthentication types:

PA-PK-AS-REQ	14
PA-PK-AS-REP	15

The extensions also involve new error types; we introduce the following types:

KDC_ERR_CLIENT_NOT_TRUSTED	62
KDC_ERR_KDC_NOT_TRUSTED	63
KDC_ERR_INVALID_SIG	64
KDC_ERR_KEY_TOO_WEAK	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_REVOCATION_STATUS_UNAVAILABLE	74
KDC_ERR_CLIENT_NAME_MISMATCH	75
KDC_ERR_KDC_NAME_MISMATCH	76

We utilize the following typed data for errors:

TD-PKINIT-CMS-CERTIFICATES	101
TD-DH-PARAMETERS	102
TD-TRUSTED-CERTIFIERS	104
TD-CERTIFICATE-INDEX	105

We utilize the following encryption types (which map directly to OIDs):

dsaWithSHA1-CmsOID	9
md5WithRSAEncryption-CmsOID	10
sha1WithRSAEncryption-CmsOID	11
rc2CBC-EnvOID	12

```
rsaEncryption-EnvOID (PKCS#1 v1.5)      13
rsaES-OAEP-ENV-OID   (PKCS#1 v2.0)      14
des-ede3-cbc-Env-OID                    15
```

These mappings are provided so that a client may send the appropriate enctypes in the AS-REQ message in order to indicate support for the corresponding OIDs (for performing PKINIT). The above encryption types are utilized only within CMS structures within the PKINIT preauthentication fields. Their use within the Kerberos EncryptedData structure is unspecified.

In many cases, PKINIT requires the encoding of the X.500 name of a certificate authority as a Realm. When such a name appears as a realm it will be represented using the "Other" form of the realm name as specified in the naming constraints section of RFC 1510bis. For a realm derived from an X.500 name, NAMETYPE will have the value X500-RFC2253. The full realm name will appear as follows:

```
<nametype> + ":" + <string>
```

where nametype is "X500-RFC2253" and string is the result of doing an RFC2253 encoding of the distinguished name, i.e.

```
"X500-RFC2253:" + RFC2253Encode(DistinguishedName)
```

where DistinguishedName is an X.500 name, and RFC2253Encode is a function returing a readable UTF encoding of an X.500 name, as defined by RFC 2253 [11] (part of LDAPv3 [15]).

Each component of a DistinguishedName is called a RelativeDistinguishedName, where a RelativeDistinguishedName is a SET OF AttributeTypeAndValue. RFC 2253 does not specify the order in which to encode the elements of the RelativeDistinguishedName and so to ensure that this encoding is unique, we add the following rule to those specified by RFC 2253:

> When converting a multi-valued RelativeDistinguishedName to a string, the output consists of the string encodings of each AttributeTypeAndValue, in the same order as specified by the DER encoding.

Similarly, in cases where the KDC does not provide a specific policy-based mapping from the X.500 name or X.509 Version 3 SubjectAltName extension in the user's certificate to a Kerberos principal name, PKINIT requires the direct encoding of the X.500 name as a PrincipalName. In this case, the name-type of the principal name MUST be set to KRB_NT-X500-PRINCIPAL. This new name type is defined in RFC 1510bis as:

```
KRB_NT_X500_PRINCIPAL    6
```

For this type, the name-string MUST be set as follows:

```
RFC2253Encode(DistinguishedName)
```

as described above. When this name type is used, the principal's realm MUST be set to the certificate authority's distinguished

name using the X500-RFC2253 realm name format described earlier in this section.

Note that the same string may be represented using several different ASN.1 data types. As the result, the reverse conversion from an RFC2253-encoded principal name back to an X.500 name may not be unique and may result in an X.500 name that is not the same as the original X.500 name found in the client certificate.

RFC 1510bis describes an alternate encoding of an X.500 name into a realm name. However, as described in RFC 1510bis, the alternate encoding does not guarantee a unique mapping from a DistinguishedName inside a certificate into a realm name and similarly cannot be used to produce a unique principal name. PKINIT therefore uses an RFC 2253-based name mapping approach, as specified above.

RFC 1510bis specifies the ASN.1 structure for PrincipalName as follows:

```
PrincipalName ::=   SEQUENCE {
                      name-type[0]     INTEGER,
                      name-string[1]   SEQUENCE OF GeneralString
}
```

The following rules relate to the the matching of PrincipalNames with regard to the PKI name constraints for CAs as laid out in RFC 2459 [12]. In order to be regarded as a match (for permitted and excluded name trees), the following MUST be satisfied.

1. If the constraint is given as a user plus realm name, or as a client principal name plus realm name (as specified in RFC 1510bis), the realm name MUST be valid (see 2.a-d below) and the match MUST be exact, byte for byte.

2. If the constraint is given only as a realm name, matching depends on the type of the realm:

   a. If the realm contains a colon (':') before any equal sign ('='), it is treated as a realm of type Other, and MUST match exactly, byte for byte.

   b. Otherwise, if the realm name conforms to rules regarding the format of DNS names, it is considered a realm name of type Domain. The constraint may be given as a realm name 'FOO.BAR', which matches any PrincipalName within the realm 'FOO.BAR' but not those in subrealms such as 'CAR.FOO.BAR'. A constraint of the form '.FOO.BAR' matches PrincipalNames in subrealms of the form 'CAR.FOO.BAR' but not the realm 'FOO.BAR' itself.

   c. Otherwise, the realm name is invalid and does not match under any conditions.

3.1.1. Encryption and Key Formats

In the exposition below, we use the terms public key and private key generically. It should be understood that the term "public

key" may be used to refer to either a public encryption key or a signature verification key, and that the term "private key" may be used to refer to either a private decryption key or a signature generation key. The fact that these are logically distinct does not preclude the assignment of bitwise identical keys for RSA keys.

In the case of Diffie-Hellman, the key is produced from the agreed bit string as follows:

* Truncate the bit string to the required length.
* Apply the specific cryptosystem's random-to-key function.

Appropriate key constraints for each valid cryptosystem are given in RFC 1510bis.

3.1.2. Algorithm Identifiers

PKINIT does not define, but does permit, the algorithm identifiers listed below.

3.1.2.1. Signature Algorithm Identifiers

The following signature algorithm identifiers specified in [8] and in [12] are used with PKINIT:

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

3.1.2.2 Diffie-Hellman Key Agreement Algorithm Identifier

The following algorithm identifier shall be used within the SubjectPublicKeyInfo data structure: dhpublicnumber

This identifier and the associated algorithm parameters are specified in RFC 2459 [12].

3.1.2.3. Algorithm Identifiers for RSA Encryption

These algorithm identifiers are used inside the EnvelopedData data structure, for encrypting the temporary key with a public key:

rsaEncryption (RSA encryption, PKCS#1 v1.5)
id-RSAES-OAEP (RSA encryption, PKCS#1 v2.0)

Both of the above RSA encryption schemes are specified in [13]. Currently, only PKCS#1 v1.5 is specified by CMS [8], although the CMS specification says that it will likely include PKCS#1 v2.0 in the future. (PKCS#1 v2.0 addresses adaptive chosen ciphertext vulnerability discovered in PKCS#1 v1.5.)

3.1.2.4. Algorithm Identifiers for Encryption with Secret Keys

These algorithm identifiers are used inside the EnvelopedData data structure in the PKINIT Reply, for encrypting the reply key with the temporary key:

```
    des-ede3-cbc (3-key 3-DES, CBC mode)
    rc2-cbc      (RC2, CBC mode)
```

The full definition of the above algorithm identifiers and their corresponding parameters (an IV for block chaining) is provided in the CMS specification [8].

3.2. Public Key Authentication

Implementation of the changes in this section is REQUIRED for compliance with PKINIT.

3.2.1. Client Request

Public keys may be signed by some certification authority (CA), or they may be maintained by the KDC in which case the KDC is the trusted authority. Note that the latter mode does not require the use of certificates.

The initial authentication request is sent as per RFC 1510bis, except that a preauthentication field containing data signed by the user's private key accompanies the request:

```
PA-PK-AS-REQ ::= SEQUENCE {
                                -- PA TYPE 14
    signedAuthPack          [0] ContentInfo,
                                    -- Defined in CMS [8];
                                    -- SignedData OID is {pkcs7 2}
                                    -- AuthPack (below) defines the
                                    -- data that is signed.
    trustedCertifiers       [1] SEQUENCE OF TrustedCas OPTIONAL,
                                    -- This is a list of CAs that the
                                    -- client trusts and that certify
                                    -- KDCs.
    kdcCert                 [2] IssuerAndSerialNumber OPTIONAL
                                    -- As defined in CMS [8];
                                    -- specifies a particular KDC
                                    -- certificate if the client
                                    -- already has it.
    encryptionCert          [3] IssuerAndSerialNumber OPTIONAL
                                    -- For example, this may be the
                                    -- client's Diffie-Hellman
                                    -- certificate, or it may be the
                                    -- client's RSA encryption
                                    -- certificate.
}

TrustedCas ::= CHOICE {
    principalName         [0] KerberosName,
                                -- as defined below
    caName                [1] Name
                                -- fully qualified X.500 name
                                -- as defined by X.509
    issuerAndSerial       [2] IssuerAndSerialNumber
                                -- Since a CA may have a number of
                                -- certificates, only one of which
                                -- a client trusts
```

}

The type of the ContentInfo in the signedAuthPack is SignedData. Its usage is as follows:

The SignedData data type is specified in the Cryptographic Message Syntax, a product of the S/MIME working group of the IETF. The following describes how to fill in the fields of this data:

1. The encapContentInfo field MUST contain the PKAuthenticator and, optionally, the client's Diffie Hellman public value.

   a. The eContentType field MUST contain the OID value for pkauthdata: iso (1) org (3) dod (6) internet (1) security (5) kerberosv5 (2) pkinit (3) pkauthdata (1)

   b. The eContent field is data of the type AuthPack (below).

2. The signerInfos field contains the signature of AuthPack.

3. The Certificates field, when non-empty, contains the client's certificate chain. If present, the KDC uses the public key from the client's certificate to verify the signature in the request. Note that the client may pass different certificate chains that are used for signing or for encrypting. Thus, the KDC may utilize a different client certificate for signature verification than the one it uses to encrypt the reply to the client. For example, the client may place a Diffie-Hellman certificate in this field in order to convey its static Diffie Hellman certificate to the KDC to enable static-ephemeral Diffie-Hellman mode for the reply; in this case, the client does NOT place its public value in the AuthPack (defined below). As another example, the client may place an RSA encryption certificate in this field. However, there MUST always be (at least) a signature certificate.

4. When a DH key is being used, the public exponent is provided in the subjectPublicKey field of the SubjectPublicKeyInfo and the DH parameters are supplied as a DomainParameters in the AlgorithmIdentitfier parameters. The DH paramters SHOULD be chosen from the First and Second defined Oakley Groups [The Internet Key Exchange (IKE) RFC-2409], if a server will not accept either of these groups, it will respond with a krb-error of KDC_ERR_KEY_TOO_WEAK. The accompanying e-data is a SEQUENCE of TypedData that includes type TD-DH-PARAMETERS (102) whose data-value is DomainParameters with appropriate Diffie-Hellman parameters for the client to use.

5. The KDC may wish to use cached Diffie-Hellman parameters (see Section 3.2.2, KDC Response). To indicate acceptance of cached parameters, the client sends zero in the nonce field of the PKAuthenticator. Zero is not a valid value for this field under any other circumstances. If cached parameters are used, the client and the KDC MUST perform key derivation (for the appropriate cryptosystem) on the

```
        resulting encryption key, as specified in RFC 1510bis.  (With
        a zero nonce, message binding is performed using the nonce
        in the main request, which must be encrypted using the
        encapsulated reply key.)

AuthPack ::= SEQUENCE {
    pkAuthenticator         [0] PKAuthenticator,
    clientPublicValue       [1] SubjectPublicKeyInfo OPTIONAL
                                -- if client is using Diffie-Hellman
                                -- (ephemeral-ephemeral only)
}

PKAuthenticator ::= SEQUENCE {
    cusec                   [0] INTEGER,
                                -- for replay prevention as in RFC 1510bis
    ctime                   [1] KerberosTime,
                                -- for replay prevention as in RFC 1510bis
    nonce                   [2] INTEGER,
                                -- zero only if client will accept
                                -- cached DH parameters from KDC;
                                -- must be non-zero otherwise
    pachecksum              [3] Checksum
                                -- Checksum over KDC-REQ-BODY
                                -- Defined by Kerberos spec;
                                -- must be unkeyed, e.g. sha1 or rsa-md5
}

SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm                   AlgorithmIdentifier,
                                -- dhPublicNumber
    subjectPublicKey            BIT STRING
                                -- for DH, equals
                                -- public exponent (INTEGER encoded
                                -- as payload of BIT STRING)
}   -- as specified by the X.509 recommendation [7]

AlgorithmIdentifier ::= SEQUENCE {
    algorithm                   OBJECT IDENTIFIER,
                                -- for dhPublicNumber, this is
                                -- { iso (1) member-body (2) US (840)
                                -- ansi-x942(10046) number-type(2) 1 }
                                -- from RFC 2459 [12]
    parameters                  ANY DEFINED by algorithm OPTIONAL
                                -- for dhPublicNumber, this is
                                -- DomainParameters
}   -- as specified by the X.509 recommendation [7]

DomainParameters ::= SEQUENCE {
    p                           INTEGER, -- odd prime, p=jq +1
    g                           INTEGER, -- generator, g
    q                           INTEGER, -- factor of p-1
    j                           INTEGER OPTIONAL, -- subgroup factor
    validationParms             ValidationParms OPTIONAL
}   -- as defined in RFC 2459 [12]

ValidationParms ::= SEQUENCE {
    seed                        BIT STRING,
```

```
                                -- seed for the system parameter
                                -- generation process
    pgenCounter                 INTEGER
                                -- integer value output as part
                                -- of the of the system parameter
                                -- prime generation process
}   -- as defined in RFC 2459 [12]
```

If the client passes an issuer and serial number in the request, the KDC is requested to use the referred-to certificate. If none exists, then the KDC returns an error of type KDC_ERR_CERTIFICATE_MISMATCH. It also returns this error if, on the other hand, the client does not pass any trustedCertifiers, believing that it has the KDC's certificate, but the KDC has more than one certificate. The KDC should include information in the KRB-ERROR message that indicates the KDC certificate(s) that a client may utilize. This data is specified in the e-data, which is defined in RFC 1510bis revisions as a SEQUENCE of TypedData:

```
TypedData ::=  SEQUENCE {
                data-type      [0] INTEGER,
                data-value     [1] OCTET STRING,
} -- per Kerberos RFC 1510bis
```

where one of the TypedData elements is:
data-type = TD-PKINIT-CMS-CERTIFICATES = 101
data-value = CertificateSet // as specified by CMS [8]

The PKAuthenticator carries information to foil replay attacks, to bind the pre-authentication data to the KDC-REQ-BODY, and to bind the request and response. The PKAuthenticator is signed with the client's signature key.

3.2.2. KDC Response

Upon receipt of the AS_REQ with PA-PK-AS-REQ pre-authentication type, the KDC attempts to verify the client's certificate chain, if one is provided in the request. This is done by verifying the certification path against the KDC's policy of legitimate certifiers.

If the KDC cannot find a trusted client certificate chain within the PA-PK-AS-REQ, then the KDC sends back an error message of type KDC_ERR_CANT_VERIFY_CERTIFICATE. Certificate chain validation is defined in RFC 2459 [12]. The accompanying e-data for this error code is a SEQUENCE of TypedData that includes type TD-TRUSTED-CERTIFIERS (104) whose data-value is an OCTET STRING which is the DER encoding of

```
    TrustedCertifiers ::= SEQUENCE OF PrincipalName
                          -- X.500 name encoded as a principal name
                          -- see Section 3.1
```

If while verifying a certificate chain the KDC determines that the signature on one of the certificates in the CertificateSet from the signedAuthPack fails verification, then the KDC returns an error of type KDC_ERR_INVALID_CERTIFICATE. The accompanying

e-data is a SEQUENCE of TypedData that includes type TD-CERTIFICATE-INDEX (105) whose data-value is an OCTET STRING which is the DER encoding of the index into the CertificateSet ordered as sent by the client.

```
    CertificateIndex  ::= INTEGER
                          -- 0 = 1st certificate,
                          --     (in order of encoding)
                          -- 1 = 2nd certificate, etc
```

The KDC may also check whether any of the certificates in the client's chain has been revoked. If one of the certificates has been revoked, then the KDC returns an error of type KDC_ERR_REVOKED_CERTIFICATE; if such a query reveals that the certificate's revocation status is unknown or not available, then if required by policy, the KDC returns the appropriate error of type KDC_ERR_REVOCATION_STATUS_UNKNOWN or KDC_ERR_REVOCATION_STATUS_UNAVAILABLE. In any of these three cases, the affected certificate is identified by the accompanying e-data, which contains a CertificateIndex as described for KDC_ERR_INVALID_CERTIFICATE.

If the certificate chain can be verified, but the name of the client in the certificate does not match the client's name in the request, then the KDC returns an error of type KDC_ERR_CLIENT_NAME_MISMATCH. There is no accompanying e-data field in this case.

Even if all succeeds, the KDC may--for policy reasons--decide not to trust the client. In this case, the KDC returns an error message of type KDC_ERR_CLIENT_NOT_TRUSTED. One specific case of this is the presence or absence of an Enhanced Key Usage (EKU) OID within the certificate extensions. The rules regarding acceptability of an EKU sequence (or the absence of any sequence) are a matter of local policy. For the benefit of implementers, we define a PKINIT EKU OID as the following: iso (1) org (3) dod (6) internet (1) security (5) kerberosv5 (2) pkinit (3) pkekuoid (2).

If a trust relationship exists, the KDC then verifies the client's signature on AuthPack. If that fails, the KDC returns an error message of type KDC_ERR_INVALID_SIG. Otherwise, the KDC uses the timestamp (ctime and cusec) in the PKAuthenticator to assure that the request is not a replay. The KDC also verifies that its name is specified in the PKAuthenticator.

If the clientPublicValue field is filled in, indicating that the client wishes to use Diffie-Hellman key agreement, then the KDC checks to see that the parameters satisfy its policy. If they do not (e.g., the prime size is insufficient for the expected encryption type), then the KDC sends back an error message of type KDC_ERR_KEY_TOO_WEAK. The accompanying e-data is a SEQUENCE of TypedData that includes type TD-DH-PARAMETERS (102) whose data-value is DomainParameters with appropriate Diffie-Hellman parameters for the client to retry the request. Otherwise, it generates its own public and private values for the response.

The KDC also checks that the timestamp in the PKAuthenticator is

within the allowable window and that the principal name and realm are correct. If the local (server) time and the client time in the authenticator differ by more than the allowable clock skew, then the KDC returns an error message of type KRB_AP_ERR_SKEW as defined in RFC 1510bis.

Assuming no errors, the KDC replies as per RFC 1510bis, except as follows. The user's name in the ticket is determined by the following decision algorithm:

1. If the KDC has a mapping from the name in the certificate to a Kerberos name, then use that name.
   Else
2. If the certificate contains the SubjectAltName extention and the local KDC policy defines a mapping from the SubjectAltName to a Kerberos name, then use that name.
   Else
3. Use the name as represented in the certificate, mapping as necessary (e.g., as per RFC 2253 for X.500 names). In this case the realm in the ticket MUST be the name of the certifier that issued the user's certificate.

Note that a principal name may be carried in the subjectAltName field of a certificate. This name may be mapped to a principal record in a security database based on local policy, for example the subjectAltName may be kerberos/principal@realm format. In this case the realm name is not that of the CA but that of the local realm doing the mapping (or some realm name chosen by that realm).

If a non-KDC X.509 certificate contains the principal name within the subjectAltName version 3 extension, that name may utilize KerberosName as defined below, or, in the case of an S/MIME certificate [14], may utilize the email address. If the KDC is presented with an S/MIME certificate, then the email address within subjectAltName will be interpreted as a principal and realm separated by the "@" sign, or as a name that needs to be mapped according to local policy. If the resulting name does not correspond to a registered principal name, then the principal name is formed as defined in section 3.1.

The trustedCertifiers field contains a list of certification authorities trusted by the client, in the case that the client does not possess the KDC's public key certificate. If the KDC has no certificate signed by any of the trustedCertifiers, then it returns an error of type KDC_ERR_KDC_NOT_TRUSTED.

KDCs should try to (in order of preference):
1. Use the KDC certificate identified by the serialNumber included in the client's request.
2. Use a certificate issued to the KDC by one of the client's trustedCertifier(s);

If the KDC is unable to comply with any of these options, then the KDC returns an error message of type KDC_ERR_KDC_NOT_TRUSTED to the client.

The KDC encrypts the reply not with the user's long-term key, but

with the Diffie Hellman derived key or a random key generated for this particular response which is carried in the padata field of the TGS-REP message.

```
PA-PK-AS-REP ::= CHOICE {
                        -- PA TYPE 15
    dhSignedData       [0] ContentInfo,
                        -- Defined in CMS [8] and used only with
                        -- Diffie-Hellman key exchange (if the
                        -- client public value was present in the
                        -- request).
                        -- SignedData OID is {pkcs7 2}
                        -- This choice MUST be supported
                        -- by compliant implementations.
    encKeyPack         [1] ContentInfo
                        -- Defined in CMS [8].
                        -- The temporary key is encrypted
                        -- using the client public key
                        -- key.
                        -- EnvelopedData OID is {pkcs7 3}
                        -- SignedReplyKeyPack, encrypted
                        -- with the temporary key, is also
                        -- included.
}
```

The type of the ContentInfo in the dhSignedData is SignedData. Its usage is as follows:

When the Diffie-Hellman option is used, dhSignedData in PA-PK-AS-REP provides authenticated Diffie-Hellman parameters of the KDC. The reply key used to encrypt part of the KDC reply message is derived from the Diffie-Hellman exchange:

1. Both the KDC and the client calculate a secret value (g^ab mod p), where a is the client's private exponent and b is the KDC's private exponent.

2. Both the KDC and the client take the first N bits of this secret value and convert it into a reply key. N depends on the reply key type.

    a. For example, if the reply key is DES, N=64 bits, where some of the bits are replaced with parity bits, according to FIPS PUB 74.

    b. As another example, if the reply key is (3-key) 3-DES, N=192 bits, where some of the bits are replaced with parity bits, according to FIPS PUB 74.

3. The encapContentInfo field MUST contain the KdcDHKeyInfo as defined below.

    a. The eContentType field MUST contain the OID value for pkdhkeydata: iso (1) org (3) dod (6) internet (1) security (5) kerberosv5 (2) pkinit (3) pkdhkeydata (2)

    b. The eContent field is data of the type KdcDHKeyInfo

(below).

4. The certificates field MUST contain the certificates necessary for the client to establish trust in the KDC's certificate based on the list of trusted certifiers sent by the client in the PA-PK-AS-REQ. This field may be empty if the client did not send to the KDC a list of trusted certifiers (the trustedCertifiers field was empty, meaning that the client already possesses the KDC's certificate).

5. The signerInfos field is a SET that MUST contain at least one member, since it contains the actual signature.

6. If the client indicated acceptance of cached Diffie-Hellman parameters from the KDC, and the KDC supports such an option (for performance reasons), the KDC should return a zero in the nonce field and include the expiration time of the parameters in the dhKeyExpiration field. If this time is exceeded, the client SHOULD NOT use the reply. If the time is absent, the client SHOULD NOT use the reply and MAY resubmit a request with a non-zero nonce (thus indicating non-acceptance of cached Diffie-Hellman parameters). As indicated above in Section 3.2.1, Client Request, when the KDC uses cached parameters, the client and the KDC MUST perform key derivation (for the appropriate cryptosystem) on the resulting encryption key, as specified in RFC 1510bis.

```
KdcDHKeyInfo ::= SEQUENCE {
                            -- used only when utilizing Diffie-Hellman
  subjectPublicKey      [0] BIT STRING,
                            -- Equals public exponent (g^a mod p)
                            -- INTEGER encoded as payload of
                            -- BIT STRING
  nonce                 [1] INTEGER,
                            -- Binds response to the request
                            -- Exception: Set to zero when KDC
                            -- is using a cached DH value
  dhKeyExpiration       [2] KerberosTime OPTIONAL
                            -- Expiration time for KDC's cached
                            -- DH value
}
```

The type of the ContentInfo in the encKeyPack is EnvelopedData. Its usage is as follows:

The EnvelopedData data type is specified in the Cryptographic Message Syntax, a product of the S/MIME working group of the IETF. It contains a temporary key encrypted with the PKINIT client's public key. It also contains a signed and encrypted reply key.

1. The originatorInfo field is not required, since that information may be presented in the signedData structure that is encrypted within the encryptedContentInfo field.

2. The optional unprotectedAttrs field is not required for PKINIT.

3. The recipientInfos field is a SET which MUST contain exactly one member of the KeyTransRecipientInfo type for encryption with a public key.

   a. The encryptedKey field (in KeyTransRecipientInfo) contains the temporary key which is encrypted with the PKINIT client's public key.

4. The encryptedContentInfo field contains the signed and encrypted reply key.

   a. The contentType field MUST contain the OID value for id-signedData: iso (1) member-body (2) us (840) rsadsi (113549) pkcs (1) pkcs7 (7) signedData (2)

   b. The encryptedContent field is encrypted data of the CMS type signedData as specified below.

      i. The encapContentInfo field MUST contains the ReplyKeyPack.

         * The eContentType field MUST contain the OID value for pkrkeydata: iso (1) org (3) dod (6) internet (1) security (5) kerberosv5 (2) pkinit (3) pkrkeydata (3)

         * The eContent field is data of the type ReplyKeyPack (below).

      ii. The certificates field MUST contain the certificates necessary for the client to establish trust in the KDC's certificate based on the list of trusted certifiers sent by the client in the PA-PK-AS-REQ. This field may be empty if the client did not send to the KDC a list of trusted certifiers (the trustedCertifiers field was empty, meaning that the client already possesses the KDC's certificate).

      iii. The signerInfos field is a SET that MUST contain at least one member, since it contains the actual signature.

```
ReplyKeyPack ::= SEQUENCE {
                          -- not used for Diffie-Hellman
    replyKey             [0] EncryptionKey,
                             -- from RFC 1510bis
                             -- used to encrypt main reply
                             -- ENCTYPE is at least as strong as
                             -- ENCTYPE of session key
    nonce                [1] INTEGER,
                             -- binds response to the request
                             -- must be same as the nonce
                             -- passed in the PKAuthenticator
}
```

3.2.2.1. Use of transited Field

Since each certifier in the certification path of a user's certificate is equivalent to a separate Kerberos realm, the name of each certifier in the certificate chain MUST be added to the transited field of the ticket. The format of these realm names is defined in Section 3.1 of this document. If applicable, the transit-policy-checked flag should be set in the issued ticket.

3.2.2.2. Kerberos Names in Certificates

The KDC's certificate(s) MUST bind the public key(s) of the KDC to a name derivable from the name of the realm for that KDC. X.509 certificates MUST contain the principal name of the KDC (defined in RFC 1510bis) as the SubjectAltName version 3 extension. Below is the definition of this version 3 extension, as specified by the X.509 standard:

```
subjectAltName EXTENSION ::= {
    SYNTAX GeneralNames
    IDENTIFIED BY id-ce-subjectAltName
}

GeneralNames ::= SEQUENCE SIZE(1..MAX) OF GeneralName

GeneralName ::= CHOICE {
    otherName       [0] OtherName,
    ...
}

OtherName ::= SEQUENCE {
    type-id         OBJECT IDENTIFIER,
    value           [0] EXPLICIT ANY DEFINED BY type-id
}
```

For the purpose of specifying a Kerberos principal name, the value in OtherName MUST be a KerberosName, defined as follows:

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

This specific syntax is identified within subjectAltName by setting the type-id in OtherName to krb5PrincipalName, where (from the Kerberos specification) we have

```
krb5 OBJECT IDENTIFIER ::= { iso (1)
                             org (3)
                             dod (6)
                             internet (1)
                             security (5)
                             kerberosv5 (2) }

krb5PrincipalName OBJECT IDENTIFIER ::= { krb5 2 }
```

(This specification may also be used to specify a Kerberos name

within the user's certificate.) The KDC's certificate may be signed directly by a CA, or there may be intermediaries if the server resides within a large organization, or it may be unsigned if the client indicates possession (and trust) of the KDC's certificate.

Note that the KDC's principal name has the instance equal to the realm, and those fields should be appropriately set in the realm and principalName fields of the KerberosName. This is the case even when obtaining a cross-realm ticket using PKINIT.

3.2.3. Client Extraction of Reply

The client then extracts the random key used to encrypt the main reply. This random key (in encPaReply) is encrypted with either the client's public key or with a key derived from the DH values exchanged between the client and the KDC. The client uses this random key to decrypt the main reply, and subsequently proceeds as described in RFC 1510bis.

3.2.4. Required Algorithms

Not all of the algorithms in the PKINIT protocol specification have to be implemented in order to comply with the proposed standard. Below is a list of the required algorithms:

* Diffie-Hellman public/private key pairs
    * utilizing Diffie-Hellman ephemeral-ephemeral mode
* SHA1 digest and RSA for signatures
* SHA1 digest for the Checksum in the PKAuthenticator
    * using Kerberos checksum type 'sha1'
* 3-key triple DES keys derived from the Diffie-Hellman Exchange
* 3-key triple DES Temporary and Reply keys

4. Logistics and Policy

This section describes a way to define the policy on the use of PKINIT for each principal and request.

The KDC is not required to contain a database record for users who use public key authentication. However, if these users are registered with the KDC, it is recommended that the database record for these users be modified to an additional flag in the attributes field to indicate that the user should authenticate using PKINIT. If this flag is set and a request message does not contain the PKINIT preauthentication field, then the KDC sends back as error of type KDC_ERR_PREAUTH_REQUIRED indicating that a preauthentication field of type PA-PK-AS-REQ must be included in the request.

5. Security Considerations

PKINIT raises a few security considerations, which we will address in this section.

First of all, PKINIT extends the cross-realm model to the public key infrastructure. Anyone using PKINIT must be aware of how the certification infrastructure they are linking to works.

Also, as in standard Kerberos, PKINIT presents the possibility of interactions between different cryptosystems of varying strengths, and this now includes public-key cryptosystems. Many systems, for instance, allow the use of 512-bit public keys. Using such keys to wrap data encrypted under strong conventional cryptosystems, such as triple-DES, may be inappropriate.

Care should be taken in how certificates are choosen for the purposes of authentication using PKINIT. Some local policies require that key escrow be applied for certain certificate types. People deploying PKINIT should be aware of the implications of using certificates that have escrowed keys for the purposes of authentication.

As described in Section 3.2, PKINIT allows for the caching of the Diffie-Hellman parameters on the KDC side, for performance reasons. For similar reasons, the signed data in this case does not vary from message to message, until the cached parameters expire. Because of the persistence of these parameters, the client and the KDC are to use the appropriate key derivation measures (as described in RFC 1510bis) when using cached DH parameters.

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 needless expensive cryptographic operations. Strictly speaking, this is also true of base Kerberos, although the potential cost is not as great in base Kerberos, because it does not make use of public key cryptography.

Lastly, PKINIT calls for randomly generated keys for conventional cryptosystems. Many such systems contain systematically "weak" keys. For recommendations regarding these weak keys, see RFC 1510bis.

6. Transport Issues

Certificate chains can potentially grow quite large and span several UDP packets; this in turn increases the probability that a Kerberos message involving PKINIT extensions will be broken in transit. In light of the possibility that the Kerberos specification will require KDCs to accept requests using TCP as a transport mechanism, we make the same recommendation with respect to the PKINIT extensions as well.

7. Bibliography

[1] J. Kohl, C. Neuman. The Kerberos Network Authentication Service (V5). Request for Comments 1510.

[2] B.C. Neuman, Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, IEEE Communications, 32(9):33-38. September 1994.

[3] M. Sirbu, J. Chuang. Distributed Authentication in Kerberos Using Public Key Cryptography. Symposium On Network and Distributed System Security, 1997.

[4] B. Cox, J.D. Tygar, M. Sirbu. NetBill Security and Transaction Protocol. In Proceedings of the USENIX Workshop on Electronic Commerce, July 1995.

[5] T. Dierks, C. Allen. The TLS Protocol, Version 1.0 Request for Comments 2246, January 1999.

[6] B.C. Neuman, Proxy-Based Authorization and Accounting for Distributed Systems. In Proceedings of the 13th International Conference on Distributed Computing Systems, May 1993.

[7] ITU-T (formerly CCITT) Information technology - Open Systems Interconnection - The Directory: Authentication Framework Recommendation X.509 ISO/IEC 9594-8

[8] R. Housley. Cryptographic Message Syntax. draft-ietf-smime-cms-13.txt, April 1999, approved for publication as RFC.

[9] PKCS #7: Cryptographic Message Syntax Standard, An RSA Laboratories Technical Note Version 1.5 Revised November 1, 1993

[10] R. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc. A Description of the RC2(r) Encryption Algorithm March 1998.
Request for Comments 2268.

[11] M. Wahl, S. Kille, T. Howes. Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names. Request for Comments 2253.

[12] R. Housley, W. Ford, W. Polk, D. Solo. Internet X.509 Public Key Infrastructure, Certificate and CRL Profile, January 1999. Request for Comments 2459.

[13] B. Kaliski, J. Staddon. PKCS #1: RSA Cryptography Specifications, October 1998. Request for Comments 2437.

[14] S. Dusse, P. Hoffman, B. Ramsdell, J. Weinstein. S/MIME Version 2 Certificate Handling, March 1998. Request for Comments 2312.

[15] M. Wahl, T. Howes, S. Kille. Lightweight Directory Access Protocol (v3), December 1997. Request for Comments 2251.

[16] ITU-T (formerly CCITT) Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1) Rec. X.680 ISO/IEC 8824-1

[17] PKCS #3: Diffie-Hellman Key-Agreement Standard, An RSA Laboratories Technical Note, Version 1.4, Revised November 1, 1993.

8. Acknowledgements

Some of the ideas on which this proposal 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 proposal approaches those goals primarily from the Kerberos perspective. Lastly, comments from groups working on similar ideas in DCE have been invaluable.

9. Expiration Date

This draft expires March 12, 2002.

10. Authors

Brian Tung
Clifford Neuman
USC Information Sciences Institute
4676 Admiralty Way Suite 1001
Marina del Rey CA 90292-6695
Phone: +1 310 822 1511
E-mail: {brian, bcn}@isi.edu

Matthew Hur
Microsoft Corporation
One Microsoft Way
Redmond WA 98052
Phone: +1 425 707 3336
E-mail: matthur@microsoft.com

Ari Medvinsky
Liberate Technologies
2 Circle Star Way
San Carlos CA 94070
E-mail: ari@liberate.com

Sasha Medvinsky
Motorola, Inc.
6450 Sequence Drive
San Diego, CA 92121
+1 858 404 2367
E-mail: smedvinsky@gi.com

John Wray
Iris Associates, Inc.
5 Technology Park Dr.
Westford, MA 01886
E-mail: John_Wray@iris.com

Jonathan Trostle
E-mail: jtrostle@world.std.com