NETWORK WORKING GROUP
Internet-Draft
Expires: June 6, 2005

B. Tung
C. Neuman
USC Information Sciences Institute
L. Zhu
M. Hur
Microsoft Corporation
S. Medvinsky
Motorola, Inc.
December 6, 2004

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

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 preauthentication 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. Finally, the client uses the service ticket to authenticate itself to the service.

The advantage afforded by the TGT is that the client need explicitly request a ticket and expose his credentials only once. The TGT and its associated session key can then be used for any subsequent 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, Kerberos authentication exchanges use symmetric-key cryptography, in part for performance. One cost 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].

In this document, we will refer to both the AS and the TGS as the KDC.

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 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 symmetric encryption key, signed using the KDC's signature key and encrypted using the client's encryption key; or

b. a key generated through a Diffie-Hellman exchange with the client, signed using the KDC's signature key.

Any keying material required by the client to obtain the Encryption key is returned in a preauthentication 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 defines the necessary message formats. Section 3.2 describes their syntax and use 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-1 digest and RSA.
- Reply key delivery method: RSA or ephemeral-ephemeral Diffie-Hellman.

3.1.2 Defined Message and Encryption Types

PKINIT makes use of the following new preauthentication 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 KRB_AS_REQ message (to indicate acceptance of the corresponding encryption 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 CMS [RFC2630] 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 this document MUST be encoded using Distinguished Encoding Rules (DER) [X690]. All imported data structures must be encoded according to the rules specified in

Kerberos [CLAR] or CMS [RFC2630] as appropriate.

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 [FIPS74]:

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 [RFC2437] 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 [RFC2630] for encrypting the reply key with the temporary key:

```
des-ede3-cbc           (three-key 3DES, CBC mode)
rc2-cbc                (RC2, CBC mode)
aes256_CBC             (AES-256, CBC mode)
```

3.2 PKINIT Preauthentication Syntax and Use

This section defines the syntax and use of the various

preauthentication fields employed by PKINIT.

3.2.1 Client Request

The initial authentication request (KRB_AS_REQ) is sent as per [CLAR]; in addition, a preauthentication field contains data signed by the client's private signature key, as follows:

```
    WrapContentInfo ::= OCTET STRING (CONSTRAINED BY {
            -- Contains a BER encoding of ContentInfo.
    })


    WrapIssuerAndSerial ::= OCTET STRING (CONSTRAINED BY {
            -- Contains a BER encoding of IssuerAndSerialNumber.
    })


    PA-PK-AS-REQ ::= SEQUENCE {
        signedAuthPack          [0] IMPLICIT WrapContentInfo,
                                    -- Type is SignedData.
                                    -- Content is AuthPack
                                    -- (defined below).
        trustedCertifiers       [1] SEQUENCE OF TrustedCA OPTIONAL,
                                    -- A list of CAs, trusted by
                                    -- the client, used to certify
                                    -- KDCs.
        kdcCert                 [2] IMPLICIT WrapIssuerAndSerial
                                    OPTIONAL,
                                    -- Identifies a particular KDC
                                    -- certificate, if the client
                                    -- already has it.
        clientDHNonce           [3] DHNonce OPTIONAL,
        ...
    }


    TrustedCA ::= CHOICE {
        caName                  [1] Name,
                                    -- Fully qualified X.500 name
                                    -- as defined in [RFC3280].
        issuerAndSerial         [2] IMPLICIT WrapIssuerAndSerial,
                                    -- Identifies a specific CA
                                    -- certificate.
        ...
    }
```

```
AuthPack ::= SEQUENCE {
    pkAuthenticator         [0] PKAuthenticator,
    clientPublicValue       [1] SubjectPublicKeyInfo OPTIONAL,
                                -- Defined in [RFC3280].
                                -- Present only if the client
                                -- is using ephemeral-ephemeral
                                -- Diffie-Hellman.
    supportedCMSTypes       [2] SEQUENCE OF AlgorithmIdentifier
                                OPTIONAL,
                                -- List of CMS encryption types
                                -- supported by client in order
                                -- of (decreasing) preference.
    ...
}


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),
    paChecksum              [3] OCTET STRING,
                                -- Contains the SHA1 checksum,
                                -- performed over KDC-REQ-BODY.
    ...
}
```

The ContentInfo in the signedAuthPack is filled out as follows:

1. The eContent field contains data of type AuthPack. It MUST contain the pkAuthenticator, and MAY also contain the client's Diffie-Hellman public value (clientPublicValue).

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

3. The signerInfos field MUST contain the signature over the AuthPack.

4. The certificates field MUST contain at least a signature

verification certificate chain that the KDC can use to verify the signature over the AuthPack. The certificate chain(s) MUST NOT contain the root CA certificate.

5. If a Diffie-Hellman key is being used, the parameters MUST be chosen from Oakley Group 2 or 14. Implementations MUST support Group 2; they are RECOMMENDED to support Group 14 (See [RFC2409]).

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

3.2.2 Validation 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 must look for a client certificate in the signedAuthPack. If it cannot find one signed by a CA it trusts, 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 Name
```

If, while verifying the certificate chain, 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 ::= IssuerAndSerialNumber
```

```
                        -- IssuerAndSerialNumber of
                        -- certificate with invalid signature.
```

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

The KDC MAY also check whether any certificates in the client's chain 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 certificate chain, the KDC MUST also check that the certificate properly maps to the client's principal name as specified in the KRB_AS_REQ as follows:

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

2. Otherwise, if the 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 is:

```
krb5PrincipalName OBJECT IDENTIFIER ::= iso (1) org (3) dod (6)
internet (1) security (5) kerberosv5 (2) 2
```

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

Even if the certificate chain is validated, and the names in the certificate and the request match, the KDC may decide to reject requests on the basis of the absence or presence of specific EKU OIDs. For example, the certificate may include an Extended Key Usage (EKU) OID of id-pkekuoid in the extensions field:

```
{ iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2)
  pkinit(3) pkekuoid(4) }
```

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

If the client's signature on the signedAuthPack 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 ephemeral-ephemeral Diffie-Hellman, the KDC checks to see if the 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 a DomainParameters (see [RFC3279]), including appropriate Diffie-Hellman parameters with which 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 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] Name,
    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). 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.) 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 KRB_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 generated encryption key, or a key derived from a Diffie-Hellman exchange. The contents of the PA-PK-AS-REP indicate the type of encryption key that was used:


```
PA-PK-AS-REP ::= CHOICE {
    dhInfo                  [0] DHRepInfo,
    encKeyPack              [1] IMPLICIT WrapContentInfo,
                                -- Type is EnvelopedData.
                                -- Content is SignedData over
                                -- ReplyKeyPack (defined below).
    ...
}
```

```
DHRepInfo ::= SEQUENCE {
    dhSignedData            [0] ContentInfo,
                                -- Type is SignedData.
                                -- Content is KDCDHKeyInfo
                                -- (defined below).
    serverDHNonce           [1] DHNonce OPTIONAL
}

KDCDHKeyInfo ::= SEQUENCE {
    subjectPublicKey        [0] BIT STRING,
                                -- Equals public exponent
                                -- (g^a mod p).
                                -- INTEGER encoded as payload
                                -- of BIT STRING.
    nonce                   [1] INTEGER (0..4294967295),
    dhKeyExpiration         [2] KerberosTime OPTIONAL,
                                -- Expiration time for KDC's
                                -- cached values.  If this field
                                -- is omitted then the
                                -- serverDHNonce field MUST also
                                -- be omitted.
    ...
}
```

The fields of the ContentInfo for dhSignedData are to be filled in as follows:

1. The eContent field contains data of type KDCDHKeyInfo.

2. The eContentType field contains the OID value for id-pkdhkeydata: { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2) pkinit(3) pkdhkeydata(2) }.

3. The signerInfos field contains a single signerInfo, which is the signature of the KDCDHKeyInfo.

4. The certificates field contains a signature verification certificate chain that the client will use to verify the KDC's signature over the KDCDHKeyInfo. This field may only be left

empty if the client did include a kdcCert field in the

PA-PK-AS-REQ, indicating that it has the KDC's certificate. The certificate chain MUST NOT contain the root CA certificate.

5. If the client included the clientDHNonce field, then the KDC may choose to reuse its DH parameters. If the server reuses DH parameters then it MUST include an expiration time in the dhKeyExperiation field. Past the point of the expiration time, the signature of the DHRepInfo is considered invalid. When the server reuses DH parameters 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. Clients MUST NOT reuse DH parameters unless the response includes the serverDHNonce field.

If the Diffie-Hellman key exchange is used, the KDC reply key [CLAR] is derived as follows:

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

```
DHKey = g^(ab) mod p
```

where a and b are the client's and KDC'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 protocolkey from a bitstring of length K; and K-truncate truncates its input to 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 parameters 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 KDC reply key k is:

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

If the Diffie-Hellman key exchange is not used, the KDC reply key [CLAR] is encrypted in the encKeyPack, which contains data of type ReplyKeyPack:

```
ReplyKeyPack ::= SEQUENCE {
    replyKey                [0] EncryptionKey,
                                -- Defined in [CLAR].
                                -- Used to encrypt main reply.
                                -- MUST be at least as strong
                                -- as session key.  (Using the
                                -- same enctype and a strong
                                -- prng should suffice, if no
                                -- stronger encryption system
                                -- is available.)
    nonce                   [1] INTEGER (0..4294967295),
                                -- Contains the nonce in
                                -- the KDCDHKeyInfo.
    ...
}
```

The fields of the ContentInfo for encKeyPack MUST be filled in as follows:

1. The content is of type SignedData. The eContent for the SignedData is of type ReplyKeyPack.

2. The eContentType for the SignedData contains the OID value for id-pkrkeydata: { iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2) pkinit(3) pkrkeydata(3) }.

3. The signerInfos field contains a single signerInfo, which is the signature of the ReplyKeyPack.

4. The certificates field contains a signature verification certificate chain that the client will use to verify the KDC's signature over the ReplyKeyPack. This field may only be left empty if the client included a kdcCert field in the PA-PK-AS-REQ, indicating that it has the KDC's certificate. The certificate chain MUST NOT contain the root CA certificate.

5. The contentType for the EnvelopedData contains the OID value for id-signedData: { iso (1) member-body (2) us (840) rsadsi (113549) pkcs (1) pkcs7 (7) signedData (2) }.

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

7. The unprotectedAttrs or originatorInfo fields MAY be present.

3.2.4 Validation of KDC Reply

Upon receipt of the KDC's reply, the client proceeds as follows. If the PA-PK-AS-REP contains a dhSignedData, the client obtains and verifies the Diffie-Hellman parameters, and obtains the shared key as described above. Otherwise, the message contains an encKeyPack, and the client decrypts and verifies the temporary encryption key.

In either case, the client MUST check to see if the included certificate contains a subjectAltName extension of type dNSName or iPAddress (if the KDC is specified by IP address instead of name). If it does, it MUST check to see if that extension matches the KDC it believes it is communicating with, with matching rules specified in RFC 2459. Exception: If the client has some external information as to the identity of the KDC, this check MAY be omitted.

The client also MUST check that the KDC's certificate contains an extendedKeyUsage OID of id-pkkdcekuoid:

```
{ iso(1) org(3) dod(6) internet(1) security(5) kerberosv5(2)
  pkinit(3) pkkdcekuoid(5) }
```

If all applicable checks are satisfied, the client then decrypts the main reply with the resulting key, and then proceeds as described in [1].

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 preauthentication 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.

The padata-value for the PA-PK-AS-REQ entry in the METHOD-DATA object is an empty octet string and SHOULD be ignored otherwise.

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 allows the use of a zero nonce in the PKAuthenticator when cached Diffie-Hellman keys are used. In this case, message binding is performed using the nonce in the main request in the same way as it is done for ordinary KRB_AS_REQs (without the PKINIT pre-authenticator). The nonce field in the KDC request body is

signed through the checksum in the PKAuthenticator, which cryptographically binds the PKINIT pre-authenticator to the main body of the AS Request and also provides message integrity for the full AS Request.

However, when a PKINIT pre-authenticator in the KRB_AS_REP has a zero-nonce, and an attacker has somehow recorded this pre-authenticator and discovered the corresponding Diffie-Hellman private key (e.g., with a brute-force attack), the attacker will be able to fabricate his own KRB_AS_REP messages that impersonate the KDC with this same pre-authenticator. This compromised pre-authenticator will remain valid as long as its expiration time has not been reached and it is therefore important for clients to check this expiration time and for the expiration time to be reasonably short, which depends on the size of the Diffie-Hellman group.

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. [This seems to reflect the consensus of the Kerberos working group.]

5. Acknowledgements

The following people have made significant contributions to this draft: Paul Leach, Sam Hartman, Love Hornquist Astrand, Ken Raeburn, Nicolas Williams, John Wray, Jonathan Trostle, Tom Yu and Jeff Hutzelman.

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 Normative References

[CLAR] Neuman, B., Yu, Y., Hartman, S. and K. Raeburn, "The Kerberos Network Authentication Service (V5)", draft-ietf-krb-wg-kerberos-clarifications, work in progress.

[FIPS74] NIST, Guidelines for Implementing and Using the NBS Encryption Standard, April 1981. FIPS PUB 74.

[KCRYPTO] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", December 2004.

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

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[RFC2437] Kaliski, B. and J. Staddon, "PKCS #1: RSA Cryptography Specifications Version 2.0", RFC 2437, October 1998.

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

[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.

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

Clifford Neuman
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

Matt Hur
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

EMail: matthur@microsoft.com

Sasha Medvinsky
Motorola, Inc.
6450 Sequence Drive
San Diego, CA 92121
US

EMail: smedvinsky@motorola.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(3)
   } DEFINITIONS EXPLICIT TAGS ::= BEGIN


       IMPORTS
            SubjectPublicKeyInfo, AlgorithmIdentifier, Name
               FROM PKIX1Explicit88 { iso (1)
                 identified-organization (3) dod (6) internet (1)
                 security (5) mechanisms (5) pkix (7) id-mod (0)
                 id-pkix1-explicit (18) }


           ContentInfo, IssuerAndSerialNumber
               FROM CryptographicMessageSyntax { iso(1) member-body(2)
                 us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
                 modules(0) cms(1) }


           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-pkdhkeydata  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
```



```
WrapContentInfo ::= OCTET STRING (CONSTRAINED BY {
        -- Contains a BER encoding of ContentInfo.
})


WrapIssuerAndSerial ::= OCTET STRING (CONSTRAINED BY {
        -- Contains a BER encoding of IssuerAndSerialNumber.
})


PA-PK-AS-REQ ::= SEQUENCE {
   signedAuthPack          [0] IMPLICIT WrapContentInfo,
                               -- Type is SignedData.
                               -- Content is AuthPack
                               -- (defined below).
   trustedCertifiers       [1] SEQUENCE OF TrustedCA OPTIONAL,
                               -- A list of CAs, trusted by
                               -- the client, used to certify
                               -- KDCs.
   kdcCert                 [2] IMPLICIT WrapIssuerAndSerial
                               OPTIONAL,
                               -- Identifies a particular KDC
                               -- certificate, if the client
                               -- already has it.
   clientDHNonce           [3] DHNonce OPTIONAL,
   ...
}


TrustedCA ::= CHOICE {
   caName                  [1] Name,
                               -- Fully qualified X.500 name
                               -- as defined in [RFC3280].
   issuerAndSerial         [2] IMPLICIT WrapIssuerAndSerial,
                               -- Identifies a specific CA
```

```
                                    -- certificate.
        ...
    }


    AuthPack ::= SEQUENCE {
        pkAuthenticator         [0] PKAuthenticator,
        clientPublicValue       [1] SubjectPublicKeyInfo OPTIONAL,
                                    -- Defined in [RFC3280].
                                    -- Present only if the client
                                    -- is using ephemeral-ephemeral
                                    -- Diffie-Hellman.
        supportedCMSTypes       [2] SEQUENCE OF AlgorithmIdentifier
```



```
                                    OPTIONAL,
                                    -- List of CMS encryption types
                                    -- supported by client in order
                                    -- of (decreasing) preference.
        ...
    }


    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),
        paChecksum              [3] OCTET STRING,
                                    -- Contains the SHA1 checksum,
                                    -- performed over KDC-REQ-BODY.
        ...
    }


    TrustedCertifiers ::= SEQUENCE OF Name


    CertificateIndex ::= IssuerAndSerialNumber
```

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


InitialVerifiedCAs ::= SEQUENCE OF SEQUENCE {
   ca                       [0] Name,
   Validated                [1] BOOLEAN,
   ...
}


PA-PK-AS-REP ::= CHOICE {
   dhInfo                   [0] DHRepInfo,
   encKeyPack               [1] IMPLICIT WrapContentInfo,
                                -- Type is EnvelopedData.
                                -- Content is SignedData over
                                -- ReplyKeyPack (defined below).
   ...
```



```
}

DHRepInfo ::= SEQUENCE {
   dhSignedData             [0] ContentInfo,
                                -- Type is SignedData.
                                -- Content is KDCDHKeyInfo
                                -- (defined below).
   serverDHNonce            [1] DHNonce OPTIONAL
}

KDCDHKeyInfo ::= SEQUENCE {
   subjectPublicKey         [0] BIT STRING,
                                -- Equals public exponent
                                -- (g^a mod p).
                                -- INTEGER encoded as payload
                                -- of BIT STRING.
   nonce                    [1] INTEGER (0..4294967295),
   dhKeyExpiration          [2] KerberosTime OPTIONAL,
                                -- Expiration time for KDC's
                                -- cached values.  If this field
                                -- is omitted then the
```

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


    ReplyKeyPack ::= SEQUENCE {
        replyKey                [0] EncryptionKey,
                                    -- Defined in [CLAR].
                                    -- Used to encrypt main reply.
                                    -- MUST be at least as strong
                                    -- as session key.  (Using the
                                    -- same enctype and a strong
                                    -- prng should suffice, if no
                                    -- stronger encryption system
                                    -- is available.)
        nonce                   [1] INTEGER (0..4294967295),
                                    -- Contains the nonce in
                                    -- the KDCDHKeyInfo.
        ...
    }


END
```


Intellectual Property Statement

such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

Copyright Statement

Acknowledgment

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