INTERNET-DRAFT
draft-ietf-cat-kerberos-pk-init-17.txt
Updates: RFC 1510bis
expires May 31, 2004

Brian Tung
Clifford Neuman
USC/ISI
Matthew Hur
Ari Medvinsky
Microsoft Corporation
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 provision 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

The distribution of this memo is unlimited. It is filed as draft-ietf-cat-kerberos-pk-init-17.txt and expires May 31, 2004. Please send comments to the authors.

1. Abstract

This draft describes protocol extensions (hereafter called PKINIT) to the Kerberos protocol specification (RFC 1510bis [1]). These extensions provide a method for integrating public key cryptography into the initial authentication exchange, by passing cryptographic certificates and associated authenticators in preauthentication data fields.

2. 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 draft, 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 user need only explicitly request a ticket and expose his credentials once. The TGT and its associated session key can then be used for any subsequent requests. One implication 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. (Symmetric-key cryptography is typically 10-100 times faster than public-key cryptography, depending on the public-key operations. [c]) One cost of using symmetric-key cryptography is that the keys must be shared, so that before a user can authentication himself, he must already be registered with the KDC.

Conversely, public-key cryptography--in conjunction with an established certification infrastructure--permits authentication without prior registration. Adding it to Kerberos allows the widespread use of Kerberized applications by users without requiring them to register first--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. Another document (PKCROSS) describes a similar protocol for Kerberos cross-realm authentication.

3. 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).

Briefly, the following changes to RFC 1510bis are proposed:

1. If public-key authentication is indicated, the client sends

the user's public-key data and an authenticator in a preauthentication field accompanying the usual request. This authenticator is signed by the user's private signature key.

2. The KDC verifies the client's request against its own policy and certification authorities.

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

   a. a randomly generated key, signed using the KDC's signature key and encrypted using the user's encryption key; or

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

   Any key data 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. Implementation of all specified formats and uses in these sections is REQUIRED for compliance with PKINIT.

3.1. Definitions

3.1.1. Required Algorithms

[What is the current list of required algorithm? --brian]

3.1.2. Defined Message and Encryption Types

PKINIT makes use of the following new preauthentication types:

PA-PK-AS-REQ	TBD
PA-PK-AS-REP	TBD

PKINIT introduces the following new error 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_CLIENT_NAME_MISMATCH                75
```

PKINIT uses the following typed data types for errors:

```
    TD-DH-PARAMETERS                           102
    TD-TRUSTED-CERTIFIERS                      104
    TD-CERTIFICATE-INDEX                       105
```

PKINIT defines the following encryption types, for use in the 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-ENV-OID     (PKCS1 v2.0)        14
    des-ede3-cbc-Env-OID                       15
```

The above encryption types are used (in PKINIT) only within CMS [8] structures within the PKINIT preauthentication fields. Their use within Kerberos EncryptedData structures is unspecified.

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 [11]:

```
    dhpublicnumber
```

PKINIT uses the following signature algorithm identifiers [8, 12]:

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

PKINIT uses the following encryption algorithm identifiers [12] for encrypting the temporary key with a public key:

```
    rsaEncryption          (PKCS1 v1.5)
```

```
    id-RSAES-OAEP           (PKCS1 v2.0)
```

These OIDs are not to be confused with the encryption types listed above.

PKINIT uses the following algorithm identifiers [8] for encrypting the reply key with the temporary key:

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

Again, these OIDs are not to be confused with the encryption types listed above.

3.2. PKINIT Preauthentication Syntax and Use

In this section, we describe the syntax and use of the various preauthentication fields employed to implement PKINIT.

3.2.1. Client Request

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

```
    PA-PK-AS-REQ ::= SEQUENCE {
                                        -- PAType TBD
        signedAuthPack          [0] ContentInfo,
                                    -- Defined in CMS.
                                    -- Type is SignedData.
                                    -- Content is AuthPack
                                    -- (defined below).
        trustedCertifiers       [1] SEQUENCE OF TrustedCAs OPTIONAL,
                                    -- A list of CAs, trusted by
                                    -- the client, used to certify
                                    -- KDCs.
        kdcCert                 [2] IssuerAndSerialNumber OPTIONAL,
                                    -- Defined in CMS.
                                    -- Identifies a particular KDC
                                    -- certificate, if the client
                                    -- already has it.
        encryptionCert          [3] IssuerAndSerialNumber OPTIONAL,
                                    -- May identify the user's
                                    -- Diffie-Hellman certificate,
                                    -- or an RSA encryption key
                                    -- certificate.
        ...
    }
```

```
    TrustedCAs ::= CHOICE {
        caName                  [0] Name,
                                    -- Fully qualified X.500 name
                                    -- as defined in X.509 [11].
        issuerAndSerial         [1] IssuerAndSerialNumber,
                                    -- Identifies a specific CA
                                    -- certificate, if the client
                                    -- only trusts one.
        ...
    }
```

[Should we even allow principalName as a choice? --brian]

```
    AuthPack ::= SEQUENCE {
        pkAuthenticator         [0] PKAuthenticator,
        clientPublicValue       [1] SubjectPublicKeyInfo OPTIONAL
                                    -- Defined in X.509,
                                    -- reproduced below.
                                    -- Present only if the client
                                    -- is using ephemeral-ephemeral
                                    -- Diffie-Hellman.
    }

    PKAuthenticator ::= SEQUENCE {
        cusec                   [0] INTEGER,
        ctime                   [1] KerberosTime,
                                    -- cusec and ctime are used as
                                    -- in RFC 1510bis, for replay
                                    -- prevention.
        nonce                   [2] INTEGER,
                                    -- Binds reply to request,
                                    -- except is zero when client
                                    -- will accept cached
                                    -- Diffie-Hellman parameters
                                    -- from KDC and MUST NOT be
                                    -- zero otherwise.
        paChecksum              [3] Checksum,
                                    -- Defined in RFC 1510bis.
                                    -- Performed over KDC-REQ-BODY,
                                    -- must be unkeyed.
        ...
    }

    SubjectPublicKeyInfo ::= SEQUENCE {
                                    -- As defined in X.509.
        algorithm                   AlgorithmIdentifier,
                                    -- Equals dhpublicnumber (see
                                    -- AlgorithmIdentifier, below)
                                    -- for PKINIT.
        subjectPublicKey            BIT STRING
```

```
                                    -- Equals public exponent
                                    -- (INTEGER encoded as payload
                                    -- of BIT STRING) for PKINIT.
    }

    AlgorithmIdentifier ::= SEQUENCE {
                                    -- As defined in X.509.
        algorithm                   OBJECT IDENTIFIER,
                                    -- dhpublicnumber is
                                    -- { iso (1) member-body (2)
                                    -- US (840) ansi-x942 (10046)
                                    -- number-type (2) 1 }
                                    -- From RFC 2459 [11].
        parameters                  ANY DEFINED BY algorithm OPTIONAL
                                    -- Content is DomainParameters
                                    -- (see below) for PKINIT.
    }

    DomainParameters ::= SEQUENCE {
                                    -- As defined in RFC 2459.
        p                           INTEGER,
                                    -- p is the odd prime, equals
                                    -- jq+1.
        g                           INTEGER,
                                    -- Generator.
        q                           INTEGER,
                                    -- Divides p-1.
        j                           INTEGER OPTIONAL,
                                    -- Subgroup factor.
        validationParms             ValidationParms OPTIONAL
    }

    ValidationParms ::= SEQUENCE {
                                    -- As defined in RFC 2459.
        seed                        BIT STRING,
                                    -- Seed for the system parameter
                                    -- generation process.
        pgenCounter                 INTEGER
                                    -- Integer value output as part
                                    -- of the system parameter
                                    -- generation process.
    }
```

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 user's Diffie-Hellman public value (clientPublicValue).

2. 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)}

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

4. The certificates field MUST contain at least a signature verification certificate chain that the KDC can use to verify the signature on the AuthPack. Additionally, the client may also insert an encryption certificate chain, if (for example) the client is not using ephemeral-ephemeral Diffie-Hellman.

5. If a Diffie-Hellman key is being used, the parameters SHOULD be chosen from the First or Second defined Oakley Groups. (See RFC 2409 [c].)

6. The KDC may wish to use cached Diffie-Hellman parameters. To indicate acceptance of caching, the client sends zero in the nonce field of the pkAuthenticator. Zero is not a valid value for this field under any other circumstances. Since zero is used to indicate acceptance of cached parameters, message binding in this case is performed instead using the nonce in the main request.

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 user 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 SEQUENCE OF TypedData:

```
TypedData ::= SEQUENCE {
                                -- As defined in RFC 1510bis.
    data-type           [0] INTEGER,
    data-value          [1] OCTET STRING
}
```

For this error, the data-type is TD-TRUSTED-CERTIFIERS, and the data-value is an OCTET STRING containing 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 SEQUENCE OF TypedData,

whose data-type is TD-CERTIFICATE-INDEX, and whose data-value is an OCTET STRING containing the DER encoding of the index into the CertificateSet field, ordered as sent by the client:

```
    CertificateIndex ::= INTEGER
                                    -- 0 = first certificate (in
                                    --     order of encoding),
                                    -- 1 = second certificate, etc.
```

If more than one signature is invalid, the KDC sends one TypedData per invalid signature.

The KDC MAY also check whether any of the certificates in the user's chain have been revoked. If any of them have been revoked, the KDC returns an error of type KDC_ERR_REVOKED_CERTIFICATE; if the KDC attempts to determine the revocation status but is unable to do so, it returns an error of type KDC_ERR_REVOCATION_STATUS_UNKNOWN. In either case, the certificate or certificates affected are identified exactly as for an error of type KDC_ERR_INVALID_CERTIFICATE (see above).

If the certificate chain is successfully validated, but the name in the user's certificate does not match the name given in the request, the KDC returns an error of type KDC_ERR_CLIENT_NAME_MISMATCH. There is no accompanying e-data for this error.

Even if the chain is validated, and the names in the certificate and the request match, the KDC may decide not to trust the client. For example, the certificate may include (or not include) an Enhanced Key Usage (EKU) OID in the extensions field. 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. In this case, the KDC returns an error of type KDC_ERR_CLIENT_NOT_TRUSTED. For the benefit of implementors, we define a PKINIT EKU OID as follows: { iso (1) org (3) dod (6) internet (1) security (5) kerberosv5 (2) pkinit (3) pkekuoid (2) }.

If the certificate chain and usage check out, but the client's signature on the signedAuthPack fails to verify, the KDC returns an error of type KDC_ERR_INVALID_SIG. There is no accompanying e-data for this error.

[What about the case when all this checks out but one or more certificates is rejected for other reasons? For example, perhaps the key is too short for local policy. --DRE]

The KDC must check the timestamp to ensure that the request is not a replay, and that the time skew falls within acceptable limits. If the check fails, the KDC returns an error of type KRB_AP_ERR_REPEAT or KRB_AP_ERR_SKEW, respectively.

Finally, 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 returns an error of type KDC_ERR_KEY_TOO_WEAK. The accompanying e-data is a SEQUENCE OF TypedData, whose data-type is TD-DH-PARAMETERS, and whose data-value is an OCTET STRING containing the DER encoding of a DomainParameters (see above), including appropriate Diffie-Hellman parameters with which to retry the request.

[This makes no sense. For example, maybe the key is too strong for local policy. --DRE]

In order to establish authenticity of the reply, the KDC will sign some key data (either the random key used to encrypt the reply in the case of a KDCDHKeyInfo, or the Diffie-Hellman parameters used to generate the reply-encrypting key in the case of a ReplyKeyPack). The signature certificate to be used is to be selected as follows:

1. If the client included a kdcCert field in the PA-PK-AS-REQ, use the referred-to certificate, if the KDC has it. If it does not, the KDC returns an error of type KDC_ERR_CERTIFICATE_MISMATCH.

2. Otherwise, if the client did not include a kdcCert field, but did include a trustedCertifiers field, and the KDC possesses a certificate issued by one of the listed certifiers, use that certificate. if it does not possess one, it returns an error of type KDC_ERR_KDC_NOT_TRUSTED.

3. Otherwise, if the client included neither a kdcCert field nor a trustedCertifiers field, and the KDC has only one signature certificate, use that certificate. If it has more than one certificate, it returns an error of type KDC_ERR_CERTIFICATE_MISMATCH.

3.2.3. KDC Reply

Assuming that the client's request has been properly validated, the KDC proceeds as per RFC 1510bis, except as follows.

The user's name as represented in the AS-REP must be derived from the certificate provided in the client's request. If the KDC has its own mapping from the name in the certificate to a Kerberos name, it uses that Kerberos name.

Otherwise, if the certificate contains a subjectAltName extension with PrincipalName, it uses that name. In this case, the realm in the ticket is that of the local realm (or some other realm name chosen by that realm). (OID and syntax for this extension to be

specified here.) Otherwise, the KDC returns an error of type KDC_ERR_CLIENT_NAME_MISMATCH.

In addition, the certifiers in the certification path of the user's certificate MUST be added to an authdata (to be specified at a later time).

The AS-REP is otherwise unchanged from RFC 1510bis. The KDC then encrypts the reply as usual, but not with the user's long-term key. Instead, it encrypts it with either a random encryption key, or a key derived through a Diffie-Hellman exchange. Which is the case is indicated by the contents of the PA-PK-AS-REP (note tags):

```
    PA-PK-AS-REP ::= CHOICE {
                                        -- PAType YY (TBD)
        dhSignedData            [0] ContentInfo,
                                        -- Type is SignedData.
                                        -- Content is KDCDHKeyInfo
                                        -- (defined below).
        encKeyPack              [1] ContentInfo,
                                        -- Type is EnvelopedData.
                                        -- Content is ReplyKeyPack
                                        -- (defined below).
        ...
    }
```

Note that PA-PK-AS-REP is a CHOICE: either a dhSignedData, or an encKeyPack, but not both. The former contains data of type KDCDHKeyInfo, and is used only when the reply is encrypted using a Diffie-Hellman derived key:

```
    KDCDHKeyInfo ::= SEQUENCE {
        subjectPublicKey        [0] BIT STRING,
                                        -- Equals public exponent
                                        -- (g^a mod p).
                                        -- INTEGER encoded as payload
                                        -- of BIT STRING.
        nonce                   [1] INTEGER,
                                        -- Binds reply to request.
                                        -- Exception: A value of zero
                                        -- indicates that the KDC is
                                        -- using cached values.
        dhKeyExpiration         [2] KerberosTime OPTIONAL,
                                        -- Expiration time for KDC's
                                        -- cached values.
        ...
    }
```

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 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 may use to verify the KDC's signature over the KDCDHKeyInfo.) It may only be left empty if the client did not include a trustedCertifiers field in the PA-PK-AS-REQ, indicating that it has the KDC's certificate.

5. If the client and KDC agree to use cached parameters, the KDC SHOULD return a zero in the nonce field and include the expiration time of the cached values 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 the cached parameters.

The key is derived as follows: Both the KDC and the client calculate the value g^(ab) mod p, where a and b are the client and KDC's private exponents, respectively. They both take the first N bits of this secret value and convert it into a reply key, where N depends on the key type.

1. For example, if the key type is DES, N = 64 bits, where some of the bits are replaced with parity bits, according to FIPS PUB 74 [c].

2. If the key type is (three-key) 3DES, N = 192 bits, where some of the bits are replaced with parity bits, again according to FIPS PUB 74.

If the KDC and client are not using Diffie-Hellman, the KDC encrypts the reply with an encryption key, packed in the encKeyPack, which contains data of type ReplyKeyPack:

```
ReplyKeyPack ::= SEQUENCE {
    replyKey                [0] EncryptionKey,
                                -- Defined in RFC 1510bis.
                                -- Used to encrypt main reply.
                                -- MUST be at least as strong as
                                -- enctype of session key.
    nonce                   [1] INTEGER,
                                -- Binds reply to request.
    ...
```

}

[What exactly does "at least as strong" mean? --DRE]

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

1. The innermost data is of type SignedData. The eContent for this data is of type ReplyKeyPack.

2. The eContentType for this data contains the OID value for 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, which the client may use to verify the KDC's signature over the ReplyKeyPack.) It may only be left empty if the client did not include a trustedCertifiers field in the PA-PK-AS-REQ, indicating that it has the KDC's certificate.

5. The outer data is of type EnvelopedData. The encryptedContent for this data is the SignedData described in items 1 through 4, above.

6. The encryptedContentType for this data contains the OID value for id-signedData: { iso (1) member-body (2) us (840) rsadsi (113549) pkcs (1) pkcs7 (7) signedData (2) }

7. 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.

8. Neither the unprotectedAttrs field nor the originatorInfo field is required for PKINIT.

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 then decrypts the main reply with the resulting key, and then proceeds as described in RFC 1510bis.

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. 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 cryptosystems of varying strengths, and this now includes public-key cryptosystems. Many systems, for example, allow the use of 512-bit public keys. Using such keys to wrap data encrypted under strong conventional cryptosystems, such as 3DES, may be inappropriate.

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.

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 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.

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.

5. 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.

6. Expiration Date

This draft expires May 31, 2004.

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

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

[13] 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

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

8. 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
Ari Medvinsky
Microsoft Corporation
One Microsoft Way
Redmond WA 98052
Phone: +1 425 707 3336
E-mail: matthur@microsoft.com, arimed@windows.microsoft.com

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