INTERNET-DRAFT
draft-ietf-cat-kerberos-pk-init-03.txt
Updates: RFC 1510
expires September 30, 1997

Clifford Neuman
Brian Tung
ISI
John Wray
Digital Equipment Corporation
Ari Medvinsky
Matthew Hur
CyberSafe Corporation
Jonathan Trostle
Novell

# Public Key Cryptography for Initial Authentication in Kerberos

## 0. Status Of this Memo

This document is an Internet-Draft. 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."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (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-03.txt, and expires September 30, 1997. Please send comments to the authors.

## 1. Abstract

This document defines extensions (PKINIT) to the Kerberos protocol specification (RFC 1510 [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 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.

One of the guiding principles in the design of PKINIT is that changes should be as minimal as possible. As a result, the basic mechanism of PKINIT is as follows: The user sends a request to the KDC as before, except that if that user is to use public key cryptography in the initial authentication step, his certificate accompanies 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 instead of being encrypted in the user's long-term key (which is derived from a password), it is encrypted in a randomly-generated key. This random key is in turn encrypted using the public key certificate that came with the request and signed using the KDC's signature key, and accompanies the reply, in the preauthentication fields.

PKINIT also allows for users with only digital signature keys to authenticate using those keys, and for users to store and retrieve private keys on the KDC.

The PKINIT specification may also be used for direct peer to peer authentication without contacting a central KDC. This application of PKINIT is described in PKTAPP [4] and is based on concepts introduced in [5, 6]. 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 SSL [7] 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 [8]).

## 3. Proposed Extensions

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

In summary, the following changes to RFC 1510 are 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.
--> Users may store private keys on the KDC for retrieval during Kerberos initial authentication.

This proposal addresses two ways that users may use public key cryptography for initial authentication. Users may present public key certificates, or they may generate their own session key, signed by their digital signature key. In either case, the end result is that the user 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 and 3.3 describe the extensions for the two initial authentication methods. Section 3.3 describes a way for the user to store and retrieve his private key on the KDC.

## 3.1. Definitions

Hash and encryption types will be specified using ENCTYPE tags; we propose the addition of the following types:

```
#define ENCTYPE_SIGN_DSA_GENERATE   0x0011
#define ENCTYPE_SIGN_DSA_VERIFY     0x0012
#define ENCTYPE_ENCRYPT_RSA_PRIV    0x0021
#define ENCTYPE_ENCRYPT_RSA_PUB     0x0022
```

allowing further signature types to be defined in the range 0x0011 through 0x001f, and further encryption types to be defined in the range 0x0021 through 0x002f.

The extensions involve new preauthentication fields. The preauthentication data types are in the range 17 through 21. These values are also specified along with their corresponding ASN.1 definition.

```
#define PA-PK-AS-REQ                17
#define PA-PK-AS-REP                18
#define PA-PK-AS-SIGN               19
#define PA-PK-KEY-REQ               20
#define PA-PK-KEY-REP               21
```

The extensions also involve new error types. The new error types are in the range 227 through 229. They are:

```
    #define KDC_ERROR_CLIENT_NOT_TRUSTED    227
    #define KDC_ERROR_KDC_NOT_TRUSTED       228
    #define KDC_ERROR_INVALID_SIG           229
```

In the exposition below, we use the following terms: encryption key, decryption key, signature key, verification key. It should be understood that encryption and verification keys are essentially public keys, and decryption and signature keys are essentially private keys. The fact that they are logically distinct does not preclude the assignment of bitwise identical keys.

## 3.2. Standard Public Key Authentication

Implementation of the changes in this section is REQUIRED for compliance with pk-init.

It is assumed that all public keys are signed by some certification authority (CA). The initial authentication request is sent as per RFC 1510, except that a preauthentication field containing data signed by the user's signature key accompanies the request:

```
PA-PK-AS-REQ ::- SEQUENCE {
                                -- PA TYPE 17
    signedPKAuth            [0] SignedPKAuthenticator,
    userCert                [1] SEQUENCE OF Certificate OPTIONAL,
                                -- the user's certificate
                                -- optionally followed by that
                                -- certificate's certifier chain
    trustedCertifiers       [2] SEQUENCE OF PrincipalName OPTIONAL
                                -- CAs that the client trusts
}

SignedPKAuthenticator ::= SEQUENCE {
    pkAuth                  [0] PKAuthenticator,
    pkAuthSig               [1] Signature,
                                -- of pkAuth
                                -- using user's signature key
}

PKAuthenticator ::= SEQUENCE {
    cusec                   [0] INTEGER,
                                -- for replay prevention
    ctime                   [1] KerberosTime,
                                -- for replay prevention
    nonce                   [2] INTEGER,
                                -- binds response to this request
    kdcName                 [3] PrincipalName,
    clientPubValue          [4] SubjectPublicKeyInfo OPTIONAL,
                                -- for Diffie-Hellman algorithm
}
```

```
Signature ::= SEQUENCE {
    signedHash              [0] EncryptedData
                                -- of type Checksum
                                -- encrypted under signature key
}

Checksum ::=   SEQUENCE {
    cksumtype               [0] INTEGER,
    checksum                [1] OCTET STRING
}   -- as specified by RFC 1510

SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm               [0] algorithmIdentifier,
    subjectPublicKey        [1] BIT STRING
}   -- as specified by the X.509 recommendation [9]

Certificate ::= SEQUENCE {
    CertType                [0] INTEGER,
                                -- type of certificate
                                -- 1 = X.509v3 (DER encoding)
                                -- 2 = PGP (per PGP draft)
    CertData                [1] OCTET STRING
                                -- actual certificate
                                -- type determined by CertType
}
```

Note: If the signature uses RSA keys, then it is to be performed as per PKCS #1.

The PKAuthenticator carries information to foil replay attacks, to bind the request and response, and to optionally pass the client's Diffie-Hellman public value (i.e. for using DSA in combination with Diffie-Hellman). The PKAuthenticator is signed with the private key corresponding to the public key in the certificate found in userCert (or cached by the KDC).

In the PKAuthenticator, the client may specify the KDC name in one of two ways: 1) a Kerberos principal name, or 2) the name in the KDC's certificate (e.g., an X.500 name, or a PGP name). Note that case #1 requires that the certificate name and the Kerberos principal name be bound together (e.g., via an X.509v3 extension).

The userCert field is a sequence of certificates, the first of which must be the user's public key certificate. Any subsequent certificates will be certificates of the certifiers of the user's certificate. These cerificates may be used by the KDC to verify the user's public key. This field is empty if the KDC already has the user's certifcate.

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.

Upon receipt of the AS_REQ with PA-PK-AS-REQ pre-authentication type, the KDC attempts to verify the user's certificate chain (userCert), if one is provided in the request. This is done by verifying the certification path against the KDC's policy of legitimate certifiers. This may be based on a certification hierarchy, or it may be simply a list of recognized certifiers in a system like PGP. If the certification path does not match one of the KDC's trusted certifiers, the KDC sends back an error message of type KDC_ERROR_CLIENT_NOT_TRUSTED, and it includes in the error data field a list of its own trusted certifiers, upon which the client resends the request.

If trustedCertifiers is provided in the PA-PK-AS-REQ, the KDC verifies that it has a certificate issued by one of the certifiers trusted by the client. If it does not have a suitable certificate, the KDC returns an error message of type KDC_ERROR_KDC_NOT_TRUSTED to the client.

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

Assuming no errors, the KDC replies as per RFC 1510, except that it encrypts the reply not with the user's key, but with a random key generated only for this particular response. This random key is sealed in the preauthentication field:

```
PA-PK-AS-REP ::= SEQUENCE {
                              -- PA TYPE 18
    kdcCert                    [0] SEQUENCE OF Certificate OPTIONAL,
                                   -- the KDC's certificate
                                   -- optionally followed by that
                                   -- certificate's certifier chain
    encPaReply                 [1] EncryptedData,
                                   -- of type PaReply
                                   -- using either the client public
                                   -- key or the Diffie-Hellman key
                                   -- specified by SignedDHPublicValue
    signedDHPublicValue        [2] SignedDHPublicValue OPTIONAL
}


PaReply ::= SEQUENCE {
    replyEncKeyPack            [0] ReplyEncKeyPack,
    replyEncKeyPackSig         [1] Signature,
```

```
                                    -- of replyEncKeyPack
                                    -- using KDC's signature key
}

ReplyEncKeyPack ::= SEQUENCE {
    replyEncKey             [0] EncryptionKey,
                                -- used to encrypt main reply
    nonce                   [1] INTEGER
                                -- binds response to the request
                                -- passed in the PKAuthenticator
}

SignedDHPublicValue ::= SEQUENCE {
    dhPublicValue           [0] SubjectPublicKeyInfo,
    dhPublicValueSig        [1] Signature
                                -- of dhPublicValue
                                -- using KDC's signature key
}
```

The kdcCert field is a sequence of certificates, the first of which must have as its root certifier one of the certifiers sent to the KDC in the PA-PK-AS-REQ. Any subsequent certificates will be certificates of the certifiers of the KDC's certificate. These cerificates may be used by the client to verify the KDC's public key. This field is empty if the client did not send to the KDC a list of trusted certifiers (the trustedCertifiers field was empty).

Since each certifier in the certification path of a user's certificate is essentially a separate realm, the name of each certifier shall be added to the transited field of the ticket. The format of these realm names shall follow the naming constraints set forth in RFC 1510 (sections 7.1 and 3.3.3.1). Note that this will require new nametypes to be defined for PGP certifiers and other types of realms as they arise.

The KDC's certificate must bind the public key to a name derivable from the name of the realm for that KDC. 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.

## 3.3. Digital Signature

Implementation of the changes in this section are OPTIONAL for compliance with pk-init.

We offer this option with the warning that it requires the client to generate a random key; the client may not be able to guarantee the same level of randomness as the KDC.

If the user registered a digital signature key with the KDC instead of an encryption key, then a separate exchange must be used. The client sends a request for a TGT as usual, except that it (rather than the KDC) generates the random key that will be used to encrypt the KDC response. This key is sent to the KDC along with the request in a preauthentication field:

```
PA-PK-AS-SIGN ::= SEQUENCE {
                              -- PA TYPE 19
    encSignedKeyPack          [0] EncryptedData
                                  -- of SignedKeyPack
                                  -- using the KDC's public key
}

SignedKeyPack ::= SEQUENCE {
    signedKey                 [0] KeyPack,
    signedKeyAuth             [1] PKAuthenticator,
    signedKeySig              [2] Signature
                                  -- of signedKey.signedKeyAuth
                                  -- using user's signature key
}

KeyPack ::= SEQUENCE {
    randomKey                 [0] EncryptionKey,
                                  -- will be used to encrypt reply
    nonce                     [1] INTEGER
}
```

where the nonce is copied from the request.

Upon receipt of the PA-PK-AS-SIGN, the KDC decrypts then verifies the randomKey. It then replies as per RFC 1510, except that the reply is encrypted not with a password-derived user key, but with the randomKey sent in the request. Since the client already knows this key, there is no need to accompany the reply with an extra preauthentication field. The transited field of the ticket should specify the certification path as described in Section 3.2.

## 3.4. Retrieving the Private Key From the KDC

Implementation of the changes in this section is RECOMMENDED for compliance with pk-init.

When the user's private key is not stored local to the user, he may choose to store the private key (normally encrypted using a password-derived key) on the KDC. We provide this option to present the user with an alternative to storing the private key on local disk at each machine where he expects to authenticate himself using pk-init. It should be noted that it replaces the added risk of

long-term storage of the private key on possibly many workstations with the added risk of storing the private key on the KDC in a form vulnerable to brute-force attack.

In order to obtain a private key, the client includes a preauthentication field with the AS-REQ message:

```
PA-PK-KEY-REQ ::= SEQUENCE {
                                -- PA TYPE 20
    patimestamp                 [0] KerberosTime OPTIONAL,
                                    -- used to address replay attacks.
    pausec                      [1] INTEGER OPTIONAL,
                                    -- used to address replay attacks.
    nonce                       [2] INTEGER,
                                    -- binds the reply to this request
    privkeyID                   [3] SEQUENCE OF KeyID OPTIONAL
                                    -- constructed as a hash of
                                    -- public key corresponding to
                                    -- desired private key
}

KeyID ::= SEQUENCE {
    KeyIdentifier               [0] OCTET STRING
}
```

The client may request a specific private key by sending the corresponding ID. If this field is left empty, then all private keys are returned.

If all checks out, the KDC responds as described in the above sections, except that an additional preauthentication field, containing the user's private key, accompanies the reply:

```
PA-PK-KEY-REP ::= SEQUENCE {
                                -- PA TYPE 21
    nonce                       [0] INTEGER,
                                    -- binds the reply to the request
    KeyData                     [1] SEQUENCE OF KeyPair
}

KeyPair ::= SEQUENCE {
    privKeyID                   [0] OCTET STRING,
                                    -- corresponding to encPrivKey
    encPrivKey                  [1] OCTET STRING
}
```

### 3.4.1. Additional Protection of Retrieved Private Keys

We solicit discussion on the following proposal: that the client may optionally include in its request additional data to encrypt the

private key, which is currently only protected by the user's password. One possibility is that the client might generate a random string of bits, encrypt it with the public key of the KDC (as in the SignedKeyPack, but with an ordinary OCTET STRING in place of an EncryptionKey), and include this with the request. The KDC then XORs each returned key with this random bit string. (If the bit string is too short, the KDC could either return an error, or XOR the returned key with a repetition of the bit string.)

In order to make this work, additional means of preauthentication need to be devised in order to prevent attackers from simply inserting their own bit string. One way to do this is to store a hash of the password-derived key (the one used to encrypt the private key). This hash is then used in turn to derive a second key (called the hash-key); the hash-key is used to encrypt an ASN.1 structure containing the generated bit string and a nonce value that binds it to the request.

Since the KDC possesses the hash, it can generate the hash-key and verify this (weaker) preauthentication, and yet cannot reproduce the private key itself, since the hash is a one-way function.

## 4. Logistics and Policy Issues

We solicit discussion on how clients and KDCs should be configured in order to determine which of the options described above (if any) should be used. One possibility is to set the user's database record to indicate that authentication is to use public key cryptography; this will not work, however, in the event that the client needs to know before making the initial request.

## 5. Compatibility with One-Time Passcodes

We solicit discussion on how the protocol changes proposed in this draft will interact with the proposed use of one-time passcodes discussed in draft-ietf-cat-kerberos-passwords-00.txt.

## 6. Strength of Cryptographic Schemes

In light of recent findings on the strength of MD5 and DES, we solicit discussion on which encryption types to incorporate into the protocol changes.

## 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] A. Medvinsky, M. Hur. Addition of Kerberos Cipher Suites to Transport Layer Security (TLS). draft-ietf-tls-kerb-cipher-suites-00.txt

[4] A. Medvinsky, M. Hur, B. Clifford Neuman. Public Key Utilizing Tickets for Application Servers (PKTAPP). draft-ietf-cat-pktapp-00.txt

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

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

[7] Alan O. Freier, Philip Karlton and Paul C. Kocher. The SSL Protocol, Version 3.0 - IETF Draft.

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

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

## 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 September 30, 1997.

## 10. Authors

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

John Wray
Digital Equipment Corporation
550 King Street, LKG2-2/Z7
Littleton, MA 01460
Phone: +1 508 486 5210
E-mail: wray@tuxedo.enet.dec.com

Ari Medvinsky
Matthew Hur
CyberSafe Corporation
1605 NW Sammamish Road Suite 310
Issaquah WA 98027-5378
Phone: +1 206 391 6000
E-mail: {ari.medvinsky, matt.hur}@cybersafe.com

Jonathan Trostle
Novell
E-mail: jonathan.trostle@novell.com