INTERNET-DRAFT
draft-ietf-cat-kerberos-pk-init-01.txt
Updates: RFC 1510
expires December 7, 1996

Clifford Neuman
Brian Tung
ISI
John Wray
Digital Equipment Corporation

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-01.txt, and expires December 7, 1996. Please send comments to the authors.

1. Abstract

This document defines extensions to the Kerberos protocol specification (RFC 1510, "The Kerberos Network Authentication Service (V5)", September 1993) to provide a method for using public key cryptography during initial authentication. The method defined specifies the way in which preauthentication data fields and error data fields in Kerberos messages are to be used to transport public key data.

2. Motivation

Public key cryptography presents a means by which a principal may demonstrate possession of a key, without ever having divulged this key to anyone else. In conventional cryptography, the encryption key and decryption key are either identical or can easily be derived from

one another. In public key cryptography, however, neither the public key nor the private key can be derived from the other (although the private key RECORD may include the information required to generate BOTH keys). Hence, a message encrypted with a public key is private, since only the person possessing the private key can decrypt it; similarly, someone possessing the private key can also encrypt a message, thus providing a digital signature.

Furthermore, conventional keys are often derived from passwords, so messages encrypted with these keys are susceptible to dictionary attacks, whereas public key pairs are generated from a pseudo-random number sequence. While it is true that messages encrypted using public key cryptography are actually encrypted with a conventional secret key, which is in turn encrypted using the public key pair, the secret key is also randomly generated and is hence not vulnerable to a dictionary attack.

The advantages provided by public key cryptography have produced a demand for its integration into the Kerberos authentication protocol. The primary advantage of registering public keys with the KDC lies in the ease of recovery in case the KDC is compromised. With Kerberos as it currently stands, compromise of the KDC is disastrous. All keys become known by the attacker and all keys must be changed.

If users register public keys, compromise of the KDC does not divulge their private key. Compromise of security on the KDC is still a problem, since an attacker can impersonate any user by certifying a bogus key with the KDC's private key. However, all bogus certificates can be invalidated by revoking and changing the KDC's public key. Legitimate users have to re-certify their public keys with the new KDC key, but the users's keys themselves do not need to be changed. Keys for application servers are conventional symmetric keys and must be changed.

Note: If a user stores his private key, in an encrypted form, on the KDC, then he does have to change the key pair, since the private key is encrypted using a symmetric key derived from a password (as described below), and is therefore vulnerable to dictionary attack. Assuming good password policy, however, legitimate users may be allowed to use the old password for a limited time, solely for the purpose of changing the key pair. The realm administrator is then not forced to re-key all users.

There are two important areas where public key cryptography will have immediate use: in the initial authentication of users registered with the KDC or using public key certificates from outside authorities, and to establish inter-realm keys for cross-realm authentication. This memo describes a method by which the first of these can be done. The second case will be the topic for a separate proposal.

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 are drawn from the DASS system, and similar extensions have been discussed for use in DCE. These changes are by no means endorsed by these groups. This is an attempt to revive some of the goals of that group, and the proposal approaches those goals primarily from the Kerberos perspective.

3. Initial authentication of users with public keys

This section describes the extensions to Version 5 of the Kerberos protocol that will support the use of public key cryptography by users in the initial request for a ticket granting ticket. This proposal is based on the implementation already made available; nevertheless, we solicit any comments on modifications or additions to the protocol description below.

Roughly speaking, the following changes to RFC 1510 are proposed:

a. The KDC's response is encrypted using a random nonce key, rather than the user's secret key.

b. This random key accompanies the response in a preauthentication field, encrypted and signed using the public key pairs of the user and the KDC.

Certificate and message formats are also defined in this section.

This proposal will allow users either to use keys registered directly with the KDC, or to use keys already registered for use with X.509, PEM, or PGP, to obtain Kerberos credentials. These credentials can then be used, as before, with application servers supporting Kerberos. Use of public key cryptography will not be a requirement for Kerberos, but if one's organization runs a KDC supporting public key, then users may choose to be registered with a public key pair, instead of the current secret key.

The application request and response between Kerberos clients and application servers will continue to be based on conventional cryptography, or will be converted to use user-to-user authentication. There are performance issues and other reasons that servers may be better off using conventional cryptography. For this proposal, we feel that 80 percent of the benefits of integrating public key with Kerberos can be attained for 20 percent of the effort, by addressing only initial authentication. This proposal does not preclude separate extensions.

With these changes, users will be able to register public keys, only in realms that support public key, and they will then only be able to perform initial authentication from a client that supports public key, although they will be able to use services registered in any realm. Furthermore, users registered with conventional keys will be able to use any client.

This proposal addresses three ways in which users may use public key cryptography for initial authentication with Kerberos, with minimal change to the existing protocol. Users may register keys directly with the KDC, or they may present certificates by outside certification authorities (or certifications by other users) attesting to the association of the public key with the named user. In both cases, the end result is that the user obtains a conventional ticket granting ticket or conventional server ticket that may be used for subsequent authentication, with such subsequent authentication using only conventional cryptography.

Additionally, users may also register a digital signature key with the KDC. We provide this option for the licensing benefits, as well as a simpler variant of the initial authentication exchange. However, this option relies on the client to generate random keys.

We first consider the case where the user's key is registered with the KDC.

3.1 Definitions

Before we proceed, we will lay some groundwork definitions for encryption and signatures. We propose the following definitions of signature and encryption modes (and their corresponding values on the wire):

```
    #define ENCTYPE_SIGN_MD5_RSA      0x0011

    #define ENCTYPE_ENCRYPT_RSA_PRIV  0x0021
    #define ENCTYPE_ENCRYPT_RSA_PUB   0x0022
```

allowing further modes to be defined accordingly.

In the exposition below, we will use the notation E (T, K) to denote the encryption of data T, with key (or parameters) K.

If E is ENCTYPE_SIGN_MD5_RSA, then

```
    E (T, K) = {T, RSAEncryptPrivate (MD5Hash (T), K)}
```

If E is ENCTYPE_ENCRYPT_RSA_PRIV, then

```
    E (T, K) = RSAEncryptPrivate (T, K)
```

Correspondingly, if E is ENCTYPE_ENCRYPT_RSA_PUB, then

```
    E (T, K) = RSAEncryptPublic (T, K)
```

3.2 Initial request for user registered with public key on KDC

In this scenario it is assumed that the user is registered with a public key on the KDC. The user's private key may be held by the user, or it may be stored on the KDC, encrypted so that it cannot be used by the KDC.

3.2.1 User's private key is stored locally

If the user stores his private key locally, the initial request to the KDC for a ticket granting ticket proceeds according to RFC 1510, except that a preauthentication field containing a nonce signed by the user's private key is included. The preauthentication field may also include a list of the root certifiers trusted by the user.

```
PA-PK-AS-ROOT ::= SEQUENCE {
        rootCert[0]         SEQUENCE OF OCTET STRING,
        signedAuth[1]       SignedPKAuthenticator
}

SignedPKAuthenticator ::= SEQUENCE {
        authent[0]          PKAuthenticator,
        authentSig[1]       Signature
}

PKAuthenticator ::= SEQUENCE {
        cksum[0]            Checksum OPTIONAL,
        cusec[1]            INTEGER,
        ctime[2]            KerberosTime,
        nonce[3]            INTEGER,
        kdcRealm[4]         Realm,
        kdcName[5]          PrincipalName
}

Signature ::= SEQUENCE {
        sigType[0]          INTEGER,
        kvno[1]             INTEGER OPTIONAL,
        sigHash[2]          OCTET STRING
}
```

Notationally, sigHash is then

```
    sigType (authent, userPrivateKey)
```

where userPrivateKey is the user's private key (corresponding to the public key held in the user's database record). Valid sigTypes are thus far limited to the above-listed ENCTYPE_SIGN_MD5_RSA; we expect that other types may be listed (and given on-the-wire values between 0x0011 and 0x001f).

The format of each certificate depends on the particular

service used. (Alternatively, the KDC could send, with its reply, a sequence of certifications (see below), but since the KDC is likely to have more certifications than users have trusted root certifiers, we have chosen the first method.) In the event that the client believes it already possesses the current public key of the KDC, a zero-length root-cert field is sent.

The fields in the signed authenticator are the same as those in the Kerberos authenticator; in addition, we include a client-generated nonce, and the name of the KDC. The structure is itself signed using the user's private key corresponding to the public key registered with the KDC.

Typically, preauthentication using a secret key would not be included, but if included it may be ignored by the KDC. (We recommend that it not be included: even if the KDC should ignore the preauthentication, an attacker may not, and use an intercepted message to guess the password off-line.)

The response from the KDC would be identical to the response in RFC 1510, except that instead of being encrypted in the secret key shared by the client and the KDC, it is encrypted in a random key freshly generated by the KDC (of type ENCTYPE_ENC_CBC_CRC). A preauthentication field (specified below) accompanies the response, optionally containing a certificate with the public key for the KDC (since we do not assume that the client knows this public key), and a package containing the secret key in which the rest of the response is encrypted, along with the same nonce used in the rest of the response, in order to prevent replays. This package is itself encrypted with the private key of the KDC, then encrypted with the public key of the user.

```
PA-PK-AS-REP ::= SEQUENCE {
        kdcCert[0]          SEQUENCE OF Certificate,
        encryptShell[1]     EncryptedData, -- EncPaPkAsRepPartShell
                                           -- encrypted by encReplyTmpKey
        encryptKey[2]       EncryptedData  -- EncPaPkAsRepTmpKey
                                           -- encrypted by userPubliKey
}

EncPaPkAsRepPartShell ::= SEQUENCE {
        encReplyPart[0]     EncPaPkAsRepPart,
        encReplyPartSig[1]  Signature -- encReplyPart
                                      -- signed by kdcPrivateKey
}

EncPaPkAsRepPart ::= SEQUENCE {
        encReplyKey[0]      EncryptionKey,
        nonce[1]            INTEGER
}

EncPaPkAsRepTmpKey ::= SEQUENCE {
```

```
        encReplyTmpKey[0]   EncryptionKey
}
```

Notationally, assume that encryptPack is encrypted (or signed) with algorithm Ak, and that encryptShell is encrypted with algorithm Au. Then, encryptShell is

```
    Au (Ak ({encReplyKey, nonce}, kdcPrivateKey), userPublicKey)
```

where kdcPrivateKey is the KDC's private key, and userPublicKey is the user's public key.

The kdc-cert specification is lifted, with slight modifications, from v3 of the X.509 certificate specification:

```
Certificate ::= SEQUENCE {
        version[0]          Version DEFAULT v1 (1),
        serialNumber[1]     CertificateSerialNumber,
        signature[2]        AlgorithmIdentifier,
        issuer[3]           PrincipalName,
        validity[4]         Validity,
        subjectRealm[5]     Realm,
        subject[6]          PrincipalName,
        subjectPublicKeyInfo[7] SubjectPublicKeyInfo,
        issuerUniqueID[8]   IMPLICIT UniqueIdentifier OPTIONAL,
        subjectUniqueID[9]  IMPLICIT UniqueIdentifier OPTIONAL,
        authentSig[10]      Signature
}
```

The kdc-cert must have as its root certification one of the certifiers sent to the KDC with the original request. If the KDC has no such certification, then it will instead reply with a KRB_ERROR of type KDC_ERROR_PREAUTH_FAILED. If a zero-length root-cert was sent by the client as part of the PA-PK-AS-ROOT, then a correspondingly zero-length kdc-cert may be absent, in which case the client uses its copy of the KDC's public key.

Upon receipt of the response from the KDC, the client will verify the public key for the KDC from PA-PK-AS-REP preauthentication data field, The certificate must certify the key as belonging to a principal whose name can be derived from the realm name. If the certificate checks out, the client then decrypts the EncPaPkAsRepPart using the private key of the user, and verifies the signature of the KDC. It then uses the random key contained therein to decrypt the rest of the response, and continues as per RFC 1510. Because there is direct trust between the user and the KDC, the transited field of the ticket returned by the KDC should remain empty. (Cf. Section 3.3.)

3.2.2. Private key held by KDC

Implementation of the changes in this section is OPTIONAL.

When the user's private key is not carried with the user, the user may encrypt the private key using conventional cryptography, and register the encrypted private key with the KDC. The MD5 hash of the DES key used to encrypt the private key must also be registered with the KDC.

We provide this option with the warning that storing the private key on the KDC carries the risk of exposure in case the KDC is compromised. If a suffiently good password is chosen to encrypt the key, then this password can be used for a limited time to change the private key. If the user wishes to authenticate himself without storing the private key on each local disk, then a safer, albeit possibly less practical, alternative is to use a smart card to store the keys.

When the user's private key is stored on the KDC, the KDC record will also indicate whether preauthentication is required before returning the key (we recommend that it be required). If such preauthentication is required, when the initial request is received, the KDC will respond with a KRB_ERROR message, with msg-type set to KDC_ERR_PREAUTH_REQUIRED, and e-data set to:

```
PA-PK-AS-INFO ::= SEQUENCE {
        kdcCert[0]          SEQUENCE OF Certificate
}
```

The kdc-cert field is identical to that in the PA-PK-AS-REP preauthentication data field returned with the KDC response, and must be validated as belonging to the KDC in the same manner.

Upon receipt of the KRB_ERROR message with a PA-PK-AS-INFO field, the client will prompt the user for the password that was used to encrypt the private key, derive the DES key from that password, and calculate the MD5 hash H1 of the DES key. The client then sends a request to the KDC, which includes a timestamp and a client-generated random secret key that will be used by the KDC to super-encrypt the encrypted private key before it is returned to the client. This information is sent to the KDC in a subsequent AS_REQ message in a preauthentication data field:

```
PA-PK-AS-REQ ::= SEQUENCE {
        encHashShell[0]     EncryptedData -- EncPaPkAsReqShell
}

EncPaPkAsReqShell ::= SEQUENCE {
        encHashPart[0]      EncryptedData -- EncPaPkAsReqPart
}

EncPaPkAsReqPart ::= SEQUENCE {
        encHashKey[0]       EncryptionKey,
        nonce[1]            INTEGER
```

```
}
```

The EncPaPkAsReqPart is first encrypted with a DES key K1, derived by string_to_key from the hash H1 (with null salt), then encrypted again with the KDC's public key from the certificate in the PA-PK-AS-INFO field of the error response.

Notationally, if encryption algorithm A is used for DES encryption, and Ak is used for the public key encryption, then enc-shell is

```
    Ak (A ({encHashKey, nonce}, K1), kdcPublicKey)
```

Upon receipt of the authentication request with the PA-PK-AS-REQ, the KDC verifies the hash of the user's DES encryption key by attempting to decrypt the EncPaPkAsReqPart of the PA-PK-AS-REQ. If decryption is successful, the KDC generates the AS response as defined in RFC 1510, but additionally includes a preauthentication field of type PA-PK-USER-KEY. (This response will also be included in response to the initial request without preauthentication if preauthentication is not required for the user and the user's encrypted private key is stored on the KDC.)

```
PA-PK-USER-KEY ::= SEQUENCE {
        encUserKeyPart[0]   EncryptedData -- EncPaPkUserKeyPart
}

EncPaPkUserKeyPart ::= SEQUENCE {
        encUserKey[0]       OCTET STRING,
        nonce[1]            INTEGER
}
```

Notationally, if encryption algorithm A is used, then enc-key-part is

```
    A ({encUserKey, nonce}, enc-hash-key)
```

(where A could be null encryption).

This message contains the encrypted private key that has been registered with the KDC by the user, as encrypted by the user, optionally super-encrypted with the enc-hash-key from the PA-PK-AS-REQ message if preauthentication using that method was provided (otherwise, the EncryptedData should denote null encryption). Note that since H1 is a one-way hash, it is not possible for one to decrypt the message if one possesses H1 but not the DES key that H1 is derived from. Because there is direct trust between the user and the KDC, the transited field of the ticket returned by the KDC should remain empty. (Cf. Section 3.3.)

3.3. Clients with a public key certified by an outside authority

Implementation of the changes in this section is OPTIONAL.

In the case where the client is not registered with the current KDC, the client is responsible for obtaining the private key on its own. The client will request initial tickets from the KDC using the TGS exchange, but instead of performing pre-authentication using a Kerberos ticket granting ticket, or with the PA-PK-AS-REQ that is used when the public key is known to the KDC, the client performs preauthentication using the preauthentication data field of type PA-PK-AS-EXT-CERT:

```
PA-PK-AS-EXT-CERT ::= SEQUENCE {
        userCert[0]         SEQUENCE OF OCTET STRING,
        signedAuth[1]       SignedPKAuthenticator
}
```

where the user-cert specification depends on the type of certificate that the user possesses. In cases where the service has separate key pairs for digital signature and for encryption, we recommend that the signature keys be used for the purposes of sending the preauthentication (and deciphering the response).

The authenticator is the one used from the exchange in section 3.2.1, except that it is signed using the private key corresponding to the public key in the user-cert.

The KDC will verify the preauthentication authenticator, and check the certification path against its own policy of legitimate certifiers. This may be based on a certification hierarchy, or simply a list of recognized certifiers in a system like PGP.

If all checks out, the KDC will issue Kerberos credentials, as in 3.2, but with the names of all the certifiers in the certification path added to the transited field of the ticket, with a principal name taken from the certificate (this might be a long path for X.509, or a string like "John Q. Public <jqpublic@company.com>" if the certificate was a PGP certificate. The realm will identify the kind of certificate and the final certifier as follows:

```
    cert_type/final_certifier
```

as in PGP/<endorser@company.com>.

3.4. Digital Signature

Implementation of the changes in this section is OPTIONAL.

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

If a user registered a digital signature key pair with the KDC, a separate exchange may be used. The client sends a KRB_AS_REQ as described in section 3.2.2. If the user's database record indicates that a digital signature key is to be used, then the KDC sends back a KRB_ERROR as in section 3.2.2.

It is assumed here that the signature key is stored on local disk. The client generates a random key of enctype ENCTYPE_DES_CBC_CRC, signs it using the signature key (otherwise the signature is performed as described in section 3.2.1), then encrypts the whole with the public key of the KDC. This is returned with a separate KRB_AS_REQ in a preauthentication of type

```
PA-PK-AS-SIGNED ::= SEQUENCE {
        signedKey[0]        EncryptedData -- PaPkAsSignedData
}

PaPkAsSignedData ::= SEQUENCE {
        signedKeyPart[0]    SignedKeyPart,
        signedKeyAuth[1]    PKAuthenticator
}

SignedKeyPart ::= SEQUENCE {
        encSignedKey[0]     EncryptionKey,
        nonce[1]            INTEGER
}
```

where the nonce is the one from the request. Upon receipt of the request, the KDC decrypts, then verifies the random key. It then replies as per RFC 1510, except that instead of being encrypted with the password-derived DES key, the reply is encrypted using the randomKey sent by the client. Since the client already knows this key, there is no need to accompany the reply with an extra preauthentication field. Because there is direct trust between the user and the KDC, the transited field of the ticket returned by the KDC should remain empty. (Cf. Section 3.3.)

4. Preauthentication Data Types

We propose that the following preauthentication types be allocated for the preauthentication data packages described in this draft:

```
    #define KRB5_PADATA_ROOT_CERT     17  /* PA-PK-AS-ROOT */
    #define KRB5_PADATA_PUBLIC_REP    18  /* PA-PK-AS-REP */
    #define KRB5_PADATA_PUBLIC_REQ    19  /* PA-PK-AS-REQ */
    #define KRB5_PADATA_PRIVATE_REP   20  /* PA-PK-USER-KEY */
    #define KRB5_PADATA_PUBLIC_EXT    21  /* PA-PK-AS-EXT-CERT */
    #define KRB5_PADATA_PUBLIC_SIGN   22  /* PA-PK-AS-SIGNED */
```

5. Encryption Information

For the public key cryptography used in direct registration, we used (in our implementation) the RSAREF library supplied with the PGP 2.6.2 release. Encryption and decryption functions were implemented directly on top of the primitives made available therein, rather than the fully sealing operations in the API.

6. Compatibility with One-Time Passcodes

We solicit discussion on how the use of public key cryptography for initial authentication will interact with the proposed use of one time passwords discussed in Internet Draft <draft-ietf-cat-kerberos-passwords-00.txt>.

7. Strength of Encryption and Signature Mechanisms

In light of recent findings on the strengths of MD5 and various DES modes, we solicit discussion on which modes to incorporate into the protocol changes.

8. Expiration

This Internet-Draft expires on December 7, 1996.

9. Authors' Addresses

B. Clifford Neuman
USC/Information Sciences Institute
4676 Admiralty Way Suite 1001
Marina del Rey, CA 90292-6695

Phone: 310-822-1511
EMail: bcn@isi.edu

Brian Tung
USC/Information Sciences Institute
4676 Admiralty Way Suite 1001
Marina del Rey, CA 90292-6695

Phone: 310-822-1511
EMail: brian@isi.edu

John Wray
Digital Equipment Corporation
550 King Street, LKG2-2/Z7
Littleton, MA 01460

Phone: 508-486-5210
EMail: wray@tuxedo.enet.dec.com