INTERNET-DRAFT
draft-sirbu-kerb-ext-00.txt
Updates: RFC 1510
May 6, 1996

M. Sirbu
J. Chuang
CMU/INI

Public-Key Based Ticket Granting Service 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-sirbu-kerb-ext-00.txt, and expires November 11, 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 supporting ticket-granting services based on public-key cryptographic algorithms.

2. Motivation

Conventional Kerberos uses a two-level ticket scheme. The client first obtains a Ticket Granting Ticket (TGT) via a request from the Authentication Server (AS) at the Key Distribution Center (KDC). The client then presents the TGT to the Ticket Granting Server (TGS) to request for a service ticket for the application server it wishes to communicate with. Upon verifying the client's identity from the TGT, the TGS establishes a shared symmetric session key between the client and server, with mutual authentication of

the two principals achieved.

The Kerberos KDC/TGS arrangement introduces two significant security and performance concerns. First, because the KDC maintains a shared symmetric cipher key with every principal in the system, it is an attractive target for attack; recovering from compromise of the KDC requires establishing new shared keys with all users of the system. Second, a centralized KDC will be a communications or processing bottleneck if a large number of users present a heavy traffic load.

Both of these problems will be alleviated with this proposed public-key based extension to Kerberos, which is first described in [CTS95].

In the proposed extension, there is no need for a trusted third party beyond the certificate authority, which distributes public-key certificates for the principals. The client and server can authenticate themselves to each other. There is no longer a centralized database of symmetric keys to be compromised.

In addition, the decentralized storage and usage of public-private key pairs distribute the authentication workload across the network to individual client/server pairs. This is especially attractive from a scalability point of view.

3. Public-Key Based Ticket Granting Service (PKTGS)

In the proposed mode of "Public-Key Kerberos" or "PK Kerberos" operation, only the initial authentication between parties will be based on public key cryptography. All subsequent communications, including repeated authentications, will continue to use the more computationally efficient symmetric key methods.

The only message exchanges affected by the proposed extension are those involving the AS and TGS exchanges, as specified in section 5.4 of [rfc1510]. Once a session key is established between the client and the server, normal operation proceeds as per RFC1510.

3.1 PK Kerberos Operation

In Kerberos V5, the client obtains a TGT from the Authentication Service (AS) of the KDC and sends it to the TGS to secure a shared session key between itself and the server it wishes to communicate with. In PK Kerberos, however, the client is allowed to authenticate itself with the server directly. The distribution of public key certificates is performed by either a certificate authority (CA) or the application server itself. The server also assumes the role of the TGS.

In PK Kerberos, the client presents a service ticket request encrypted with a certified public key (we call this a Public-Key based TGS request, or PKTGS-REQ) to the server. Since this request is digitally signed with the client's private key, and encrypted with the server's public key, the server and only the server can authenticate the identity of the client. Conversely, the client is assured of the identity of the server because only the server can decrypt the PKTGS-REQ and construct a valid response.

In addition to the client's identity and other relevant information, a randomly generated one-time key is also included in the service ticket request. This key is not the actual session key, but is instead used by the server to return the service ticket to the client. This key works identically to the symmetric key generated by the KDC to be shared by the client and the TGS in the traditional TGT-REQ/TGS-REP exchange.

The client's public key certificate is also included with this request to facilitate the verification of its signature.

The server, in response to the above PKTGS-REQ message, returns a symmetric-cipher-based service ticket. This service ticket is identical in form to a Kerberos service ticket. Therefore, the message format for PKTGS-REP is very similar to that for TGS-REP. With this service ticket, the client and the server can proceed to communicate normally.

A client application can also choose to present the PKTGS-REQ to a centralized Ticket Granting Server. This may be necessary, especially during the period of migration to public-key based Kerberos, when some of the application servers are not yet equipped to process service ticket requests. This option, described in Section 4.3, is a special case where the server in question is simply a PK-aware TGS.

4. Message Exchanges

In Kerberos V5, a normally executed authentication procedure begins with the following five message exchanges:

1. client to KDC: AS-REQ
2. KDC to client: AS-REP
3. client to TGS: TGS-REQ
4. TGS to client: TGS-REP
5. client to server: AP-REQ

In Public key Kerberos, the five step exchange for initial authentication is maintained, but the first four steps are replaced by PK-equivalent versions of the messages. Normal operations proceed from step 5 onwards.

1. client to server/CA: SCERT-REQ
2. server/CA to client: SCERT-REP
3. client to server: PKTGS-REQ
4. server to client: PKTGS-REP
5. client to server: AP-REQ

Each of the new steps is described in detail in the following sub-sections. A sample specification of the protocol, based upon the Interface Specification Language (ISL) [Bra96], can be found in Appendix A.

4.1 Obtaining the Server's Public Key Certificate

The construction of a PKTGS-REQ requires encryption using the recipient's (server's) public key. Therefore, the client must obtain the server's public key certificate before it can generate the PKTGS-REQ message. This request may either be serviced by a certificate authority (CA), or the server itself.

If the client has certificate caching capabilities, steps 1 and 2 may be bypassed for subsequent authentication attempts with a server. It is the responsibility of the client to check with the CRL for any revoked certificates. If the client obtains the server's certificate from the CA, it can be sure that the certificate has not been revoked.

4.1.1 Generation of SCERT-REQ

The client initiates the authentication exchange by generating a simple request message, which consists of the principal name and realm of the server it wishes to communicate with. This message can be transmitted over any available channel, such as an unsecured remote procedure call.

4.1.2 Generation of SCERT-REP

In response to a certificate request, the server or the CA returns the certificate, which contains the public key information. Again, this message can be transmitted via an unprotected channel.

4.2 Client/Server Authentication Using Public Key Cryptography

4.2.1 Generation of PKTGS-REQ

Once the client has obtained and verified the server's public key certificate, it can proceed to generate the ticket request. The PKTGS-REQ message contains information fields similar to those in KDC-REQ messages, except that the authorization fields are now encrypted using the server's public key instead of the shared symmetric key shared by the KDC and the client in traditional

Kerberos. In addition, these fields are also signed by the client, allowing the server to verify the identity of the client.

In traditional Kerberos, the KDC generates a random symmetric key for the client to use in communicating with the TGS. The client now generates this random key, which will be referred to in the rest of this document as "Kr", and sends it in encrypted form to the server. The server will then retrieve and use this key to encrypt the responding message PKTGS-REP.

The generation of this random key "Kr" does not impose any additional security requirements on the client. The same random number generator used to generate the 'nonce' field in KRB-KDC-REQ can be used to generate this key "Kr", as long as a proper key length is used. In fact, the inclusion of this one-time random key in the message eliminates the need for a separate 'nonce' field.

An object containing all the necessary authorization information is first constructed. This object, named 'auth-data' (Section 5.3.2), includes the random key "Kr", the server's identity (to prevent replay attacks first addressed by Denning-Sacco [Sch95]), the server's public key information (to avert "man-in-the-middle" attacks), ticket life-time information, and a timestamp (to prevent replay attacks).

The 'auth-data' field is signed with the client's private key using the 'SignedData' construct defined in PKCS #7. The resulting message digest, but not the content itself, is then encrypted with the server's public key using the 'EnvelopedData' construct. The omission of the 'auth-data' content field results in a shorter message length for PKTGS-REQ. This omission is possible because all the information needed to construct and verify the hash of 'auth-data' is available to the server through other means.

4.2.2 Receipt of PKTGS-REQ

Upon receipt of the PKTGS-REQ message, the server first decrypts and retrieves the content-encryption key using its private key. Using this key, it can decrypt the actual enveloped content. The server retrieves the client's public key, which is included in the 'certificate' field of the signed content. Using this public key, the client's signature (and authenticity of the request) is verified by comparing the retrieved message digest with an independently constructed message digest of 'auth-data'.

While the 'auth-data' content field is omitted from the PKTGS-REQ message, the server can retrieve all the necessary information. Specifically, the field 'sPKeyInfo' can be retrieved from the server's own copy of its public key certificate. All the other fields present in 'auth-data' can and should be retrieved from the plaintext portion of PKTGS-REQ.

4.2.3 Generation of PKTGS-REP

The message format for PKTGS-REP is similar to that for TGS-REP in traditional Kerberos [rfc1510], and will be described in Section 5.4 of this document.

The process of generating this message is also identical to that of generating the TGS-REP, with the following three exceptions.

First, this message is now generated by the server (acting in the capacity of the Ticket Granting Service) rather than the TGS.

Second, while the ciphertext portion of the message is encrypted using the session key extracted from the TGT in traditional Kerberos, it is now encrypted using the symmetric 'randomKey' "Kr" extracted from PKTGS-REQ.

Third, while the traditional service ticket is encrypted using a symmetric key shared by the TGS and the server, it is now encrypted using a symmetric key known only to the server. This is consistent with the fact that the server and the TGS are really one entity in this scenario.

4.2.4 Receipt of PKTGS-REP

The client receives and processes this message in the same manner as it would a traditional TGS-REP. As in traditional Kerberos, the client will not be able to decrypt and/or modify the ticket. It will be able to retrieve the session key and use it to generate the appropriate authenticators for the subsequent AP-REQ message.

From this point on, all operations can proceed per normal Kerberos procedures.

4.3 Obtaining Service Tickets From a "PK-Aware" TGS

If the server with whom the client wishes to communicate is not capable of handling service ticket requests, the client has to resort to sending the request to a Ticket Granting Service (TGS). This section describes this scenario, and the appropriate course of action to be taken by the client. This escape mechanism will help preserve the functionality and integrity of the Kerberos Authentication scheme during the transition to PK Kerberos, when there can be a hybrid of "PK-aware" and "non-PK-aware" application servers.

The client initially assumes that the server is "PK-aware" and sends a SCERT-REQ message to the application server as described in Section 4.1. From the lack of a timely SCERT-REP, the client can either assume that the server is unable to handle a certificate

request, or that it is not "PK-aware" at all. In the former case, the client can choose to resend the SCERT-REQ to the certificate authority (CA) and obtain the server's certificate. If the CA is unable to return a certificate for S, then the client knows that the server is not "PK-aware." Else if the client succesfully receives a SCERT-REP from the CA, it can proceed to send a PKTGS-REQ to the server.

Once it is established that the server is not "PK-aware," the client will have to communicate with a TGS to get a traditional TGT and subsequent service ticket for the destination server. This is accomplished by the following 7-step exchange.

1. client to TGS/CA: SCERT-REQ
2. TGS/CA to client: SCERT-REP
3. client to TGS: PKTGS-REQ
4. TGS to client: PKTGS-REP
5. client to TGS: TGS-REQ
6. TGS to client: TGS-REP
7. client to server: AP-REQ

The first four steps of the message exchange are identical to those of PK Kerberos as described in the beginning of Section 4, except that the server being contacted now is a PK-aware TGS. In effect, the client is issuing a PKTGS-REQ to the TGS to request for a traditional TGT. Then, steps 5-7 are really identical to steps 3-5 of the traditional Kerberos exchange, where the client uses the TGT to request for an actual service ticket.

It is worth noting that the client will be communicating with the TGS for the first six steps of the exchange. Therefore, the performance bottleneck associated with a centralized KDC remains. However, the centralized database of symmetric keys will be much smaller in size, since the clients will now authenticate themselves to the TGS using public keys instead. The symmetric keys shared between the KDC and the servers are still required.

5. Message Specifications

5.1 SCERT-REQ

The construction of PKTGS-REQ requires encryption using the recipient's (server's) public key. Therefore, the client must first obtain the server's public key certificate. This request may be serviced by a certificate authority (CA), or the server itself. This simple request message can be transmitted over any available channel, such as an unsecured remote procedure call.

```
SCERT-REQ ::= SEQUENCE{
        pvno                    INTEGER,
```

```
        msg-type              INTEGER,
        srealm                Realm,
        sname                 PrincipalName
}
```

pvno This field is included in each message, and specifies the protocol version number (version '5' for this document).

msg-type This field indicates the type of a protocol message, and is as described in Section 5.4.1 of [rfc1510].

srealm This field specifies the realm part of the server's principal identifier.

sname This field specifies the name part of the server's principal identifier.

5.2 SCERT-REP

In response to a certificate request, the server or the CA returns the certificate, which contains the public key information. Again, this message can be transmitted via an unprotected channel.

```
SCERT-REP ::= SEQUENCE{
        pvno                  INTEGER,
        msg-type              INTEGER,
        scert                 Certificate
}
```

pvno and msg-type These fields are described above in Section 5.1

scert server's certificate (and public key), as defined in X.509 or other certificate standards. The X.509 certificate in ASN.1 notation can be found in Annex G of [X509]. It is excerpted as Appendix B of this document.

5.3 PKTGS-REQ

PKTGS-REQ is sent to the server directly, rather than to the AS or the TGS in the conventional Kerberos protocol. This message contains similar information as conventional ticket requests. The authorization fields, which includes the proposed one-time random key "Kr", is signed with the client's private key and enveloped with the server's public key. This is the basic mechanism underlying authentication and key exchange without the KDC serving as the trusted intermediary.

```
PKTGS-REQ ::= SEQUENCE {
        pvno                    INTEGER,
        msg-type                INTEGER,
        padata                  SEQUENCE OF PA-DATA OPTIONAL,
        req-body                PKTGS-REQ-BODY
}

PKTGS-REQ-BODY ::= SEQUENCE {
        kdc-options             KDCOptions,
        srealm                  Realm,
        sname                   PrincipalName,
        from                    KerberosTime OPTIONAL,
        till                    KerberosTime,
        rtime                   KerberosTime OPTIONAL,
        authtime                KerberosTime,
        etype                   SEQUENCE OF INTEGER,
        addresses               HostAddresses OPTIONAL,
        envelopedContent        ContentInfo, -- type 'envelopedData'
        additional-tickets      SEQUENCE OF Ticket OPTIONAL
}
```

Other than the following two fields, all the other fields are as described for KDC-REQ in Section 5.4.1 of [rfc1510].

authtime This field indicates the time of initial authentication request.

envelopedContent This field contains the enveloped portions of the ticket request message. It is defined to be of type 'ContentInfo', which is a generic data-type exported from RSA Lab's PKCS #7 specification [PKCS7].

The message format differs from that of KDC-REQ in the following four ways:

First, the encrypted portion of the request body is no longer an 'EncryptedData' object as defined in [rfc1510], but rather a 'ContentInfo' container of type 'envelopedData'. This is because public key encryption is used here, using the PKCS specification.

Second, 'cname' is excluded from the cleartext portion of the message. Instead, the client's identity is available in its public key certificate, which is transmitted, in encrypted form, within 'envelopedContent'.

Third, the 'nonce' field is removed. This is possible because its function can be subsumed by the 'randomKey' field in Section 5.3.2. The 'randomKey' can serve the role of a nonce since it is also randomly generated by the client, and used only once.

Fourth, the field 'authtime' is added. Since the client generates this timestamp, the server will have to verify the time elapsed between this timestamp and when it receives this message. By refusing to service a ticket request that occurred "too far" in the past, i.e., beyond the acceptable clock skew, the server can prevent replay attacks. While this field is transmitted in the clear, its integrity is assured by its inclusion in the signed 'auth-data' field.

5.3.1 EnvelopedData

'ContentInfo' is a generic data-type exported from RSA Lab's PKCS #7 specification [PKCS7]. This data type can take on one of several content-types as enumerated in PKCS #7. In this instance, it takes on the type of 'envelopedData', which is essentially the encryption of the content using a randomly generated symmetric key, followed by the encryption of the symmetric key using the recipient's public key. Syntatically, however, the encrypted key preceeds the encrypted content, such that the content can be decrypted with one pass of the message stream. The syntax relevant to this document is shown below. Appendix C provides a brief discussion of the usage of 'ContentInfo', and its ASN.1 notation. A complete specification of the 'EnvelopedData' type can be found in PKCS #7.

```
-- ContentInfo of type 'envelopedData'
ContentInfo ::= SEQUENCE {
        contentType           envelopedData,
        content               EnvelopedData }

EnvelopedData ::= SEQUENCE {
        version               Version,
        recipientInfos        RecipientInfos,
        encryptedContentInfo  EncryptedContentInfo }

RecipientInfos ::= SET OF RecipientInfo

RecipientInfo ::= SEQUENCE {
        version               Version,
        issuerAndSerialNumber IssuerAndSerialNumber,
        keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
        encryptedKey          EncryptedKey }

EncryptedContentInfo ::= SEQUENCE {
        contentType           ContentType,
        contentEncryptionAlgorithm
                              ContentEncryptionAlgorithmIdentifier,
        encryptedContent
                        [0] IMPLICIT     EncryptedContent OPTIONAL }
```

```
EncryptedKey ::= OCTET STRING      -- content encryption key

EncryptedContent ::= OCTET STRING  -- encryption of Protected-body
```

5.3.2 SignedData

The authorization fields, encapsulated in 'auth-data', needs to be signed by the client before it is ready for envelopment. The result of this signing operation is 'Protected-body'. PKCS #7 provides a framework for signing, which is used here. Therefore, we have another instance of 'ContentInfo'; but this time it takes on the content-type of 'SignedData'.

```
Protected-body ::= SEQUENCE {
        randomKey             OCTET STRING,
        contentInfo           ContentInfo -- of type 'signedData'
}
```

randomKey A randomly generated one-time key. This key is used by the server to encrypt the ciphertext portion of PKTGS-REP in section 5.4. This field also serves as the nonce for PKTGS-REQ.

contentInfo This second instance of 'ContentInfo' is used here to declare the object 'SignedData', as required by PKCS #7.

```
-- ContentInfo of type 'signedData'
ContentInfo ::= SEQUENCE {
        contentType           signedData,
        content               SignedData }

SignedData ::= SEQUENCE {
        version               Version,
        digestAlgorithms      DigestAlgorithmIdentifiers,
        contentInfo           ContentInfo, -- of type 'data'
        certificates
             [0] IMPLICIT     ExtendedCertificatesAndCertificates
                 OPTIONAL,    -- MANDATORY in this context
        crls [1] IMPLICIT     CertificateRevocationLists OPTIONAL,
        signerInfos           SignerInfos }
```

A complete description of 'SignedData' can again be found in [PKCS7]. It is worthwhile to note that the actual message digest of this signature can be found under the field 'signerInfos'.

Note that the client's certificate must be included in this

construct, even though it is defined as an optional field in PKCS #7. The client's certificate is mandatory here for the signature verification process. Appendix D offers a brief discussion of the rationale behind this requirement.

The input to be signed is a third instance of 'ContentInfo', this time simply of type 'data'.

```
-- ContentInfo of type 'data'
ContentInfo ::= SEQUENCE {
        contentType           data,
        content               auth-data OPTIONAL -- not present
}

auth-data ::= SEQUENCE {
        randomKey             OCTET STRING,
        etype                 SEQUENCE OF INTEGER,
        srealm                Realm,
        sname                 PrincipalName,
        sPKeyInfo             SubjectPublicKeyInfo,
        from                  KerberosTime OPTIONAL,
        till                  KerberosTime,
        rtime                 KerberosTime OPTIONAL,
        authtime              KerberosTime
}
```

The field 'randomKey' is identical to that found earlier in 'Protected-body'; the fields 'etype', 'srealm', 'sname', 'from', 'till', 'rtime' and 'authtime' are identical to those fields found in the plaintext of PKTGS-REQ. All of the above fields are included here for signing purposes only. The only new field is:

sPKeyInfo the server's public key which is retrieved by the client from the server's certificate found in SCERT-REP.

It is important to note that the 'content' field pointing to 'auth-data' is actually a null field. PKCS #7 allows this field to be not present when the content is supplied through other means. In this case, 'etype', 'srealm', 'sname', 'from', 'till', 'rtime', 'authtime' are available in the plaintext portion of PKTGS-REQ; 'randomKey' can be retrieved from 'Protected-body'; and the server can retrieve 'sPKeyInfo' from its own public-key certificate.

There is no advantage from the point of view of either security or privacy to encrypt the fields 'etype', 'srealm', 'sname', 'from', 'till', 'rtime' or 'authtime', as long as the integrity of these fields are guaranteed by the signature. The 'sname' and 'srealm' fields must be in the clear so that the listener process receiving the PKTGS-REQ message knows for which principal the message is

intended in the event that multiple principals are served from the same server port. Processing speed is enhanced by limiting the enveloped data to the minimum which needs to be protected. The client's certificate is encrypted to protect the privacy of the client who is attempting to communicate with this server.

5.4 PKTGS-REP

The server, in its ticket granting service capacity, returns a PKTGS-REP message. This message is similar to the TGS-REP message of traditional Kerberos, which is defined in Section 5.4.2 of [rfc1510].

This message, like the TGS-REP, consists of the ticket and an encrypted part, the latter of which includes the session key 'key' which will be used by the client to generate the authenticator in AP-REQ.

The fields 'crealm' and 'cname' are moved from the plaintext portion of the message into the encrypted part to protect client privacy. Discussion of the merits of this change is warranted. On the one hand, except for multi-user hosts, client identity can often be inferred from the IP address. Moreover, encrypting 'cname' requires the use of the client port address to match the returning PKTGS-REP to the correct PKTGS-REQ. On the other hand, this change does prevent a network observer from being able to track session requests between identifiable client and server pairs, as is the case with Kerberos V5.

```
PKTGS-REP ::= SEQUENCE {
        pvno                    INTEGER,
        msg-type                INTEGER,
        padata                  SEQUENCE OF PA-DATA OPTIONAL,
        ticket                  Ticket,
        enc-part                EncryptedData
                                -- of instance 'PKTGS-EncPart'
}
```

All of the fields are as described in Section 5.4.2 of [rfc1510].

enc-part This field is a place holder for the ciphertext and related information that forms the encrypted part of a message. The description of the encrypted part of the message follows each appearance of this field. The encrypted part is encoded as described in section 6.1 of [rfc1510]. The key used to encrypt this part is 'randomKey' "Kr" extracted from PKTGS-REQ.

```
PKTGS-EncPart ::= SEQUENCE {
        key                     EncryptionKey,
        last-req                LastReq,
        randomKey               OCTET STRING OPTIONAL,
        key-expiration          KerberosTime OPTIONAL,
        flags                   TicketFlags,
        authtime                KerberosTime,
        starttime               KerberosTime OPTIONAL,
        endtime                 KerberosTime,
        renew-till              KerberosTime OPTIONAL,
        srealm                  Realm,
        sname                   PrincipalName,
        caddr                   HostAddresses OPTIONAL,
        crealm                  Realm,
        cname                   PrincipalName
}
```

All of the fields above are as described in Section 5.4.2 of [rfc1510].

The 'nonce' field found in KDC-REP is substituted by the 'randomKey' field here, which serves the same function. In reality, this field is redundant since the entire PKTGS-EncPart is already encrypted using this one-time 'randomKey'.

From this point on, all operations can proceed per normal Kerberos procedures.

6. Definition of New Message Types

New and appropriate application numbers need to be assigned to the new message types described in this document, namely SCERT-REQ, SCERT-REP, PKTGS-REQ and PKTGS-REP.

7. References

[Bra96] S.H. Brackin: An Interface Specification Language for Cryptographic Protocols and Its Translation into HOL. Submitted to the New Security Paradigm Workshop, Lake Arrowhead, CA, September 16-19, 1996.

[CTS95] B. Cox, J.D. Tygar, M. Sirbu: NetBill Security and Transaction Protocol. Proceedings of the USENIX Workshop on Electronic Commerce, July 1995.

[PKCS7] RSA Laboratories. PKCS #7: Cryptographic Message Syntax Standard. Version 1.5, November 1993.

[rfc1510] J. Kohl, C. Neuman. RFC 1510: The Kerberos

Authentication Service (v5). September 1993.

[Sch95] B. Schneier, Applied Cryptography, 2nd Ed. 1995.

[X509] CCITT. Recommendation X.509: The Directory Authentication Framework. 1988.

8. Security Concerns

Security issues are discussed throughout this document.

9. Expiration

This Internet-Draft expires on November 11, 1996.

10. Authors' Addresses

Marvin Sirbu
Information Networking Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412)268-3436
Email: sirbu@cmu.edu

John Chung-I Chuang
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412)268-5618
Email: chuang+@cmu.edu

Appendix A: Sample ISL Specification for PK Kerberos Protocol (simplified)

```
DEFINITIONS:
PRINCIPALS:         C, S, CA;
SYMMETRIC KEYS:     Kr, Kcs, Ks;
PUBLIC KEYS:        PKC, PKS, PKCA;
PRIVATE KEYS:       ^PKC, ^PKS, ^PKCA;
OTHER:              Ts1, Ts2, Ts3, Tsc, Tss;
ENCRYPT FUNCTIONS: des, rsa;
HASH FUNCTIONS:     MD5;
des WITH ANYKEY HASINVERSE des WITH ANYKEY;
rsa WITH ^PKCA HASINVERSE rsa WITH PKCA;
rsa WITH ^PKC HASINVERSE rsa WITH PKC;
rsa WITH ^PKS HASINVERSE rsa WITH PKS;
rsa WITH PKCA HASINVERSE rsa WITH ^PKCA;
rsa WITH PKC HASINVERSE rsa WITH ^PKC;
rsa WITH PKS HASINVERSE rsa WITH ^PKS;
```

```
INITIALCONDITIONS:
CA Received C, S, CA, Tsc, Tss, PKC, PKS, ^PKCA, PKCA, rsa, MD5;
CA Believes (PublicKey CA rsa PKCA; PrivateKey CA rsa ^PKCA;
             PublicKey S rsa PKS; PublicKey C rsa PKC;
             Fresh Tss; Fresh Tsc
            );
C Received C, S, CA, Kr, Ts1, Ts3, ^PKC, PKCA, rsa, des, MD5;
C Received [CA, Tsc, C, PKC](MD5,rsa)(^PKCA) ||
           (PublicKey C rsa PKC) From CA;
C Believes (PublicKey CA rsa PKCA; PublicKey C rsa PKC;
            PrivateKey C rsa ^PKC; SharedSecret C S Kr;
            Fresh Ts1; Fresh Ts2; Fresh Ts3; Fresh Tsb;
            C Recognizes C; C Recognizes S; C Recognizes CA;
            Trustworthy CA; Trustworthy S
           );
S Received S, Ks, Kcs, Ts2, PKCA, ^PKS, rsa, des, MD5;
S Received [CA, Tss, S, PKS](MD5,rsa)(^PKCA) ||
           (PublicKey S rsa PKS) From CA;
S Believes (PublicKey S rsa PKS; PrivateKey S rsa ^PKS;
            PublicKey CA rsa PKCA;
            SharedSecret S S Ks; SharedSecret C S Kcs;
            Fresh Ts1; Fresh Ts2; Fresh Ts3;  Fresh Tsc;
            S Recognizes C; S Recognizes S; S Recognizes CA;
            Trustworthy CA; Trustworthy C
           );

PROTOCOL:
1. C -> CA: S;
2. CA -> C: [CA,Tss,S,PKS](MD5,rsa)(^PKCA)||(PublicKey S rsa PKS);
3. C -> S:  S, Ts1,
            {Kr, C,
             [CA,Tsc,C,PKC](MD5,rsa)(^PKCA)||(PublicKey C rsa PKC),
             {S, Ts1, Kr,
              [CA,Tss,S,PKS](MD5,rsa)(^PKCA)||(PublicKey S rsa PKS)
             }(MD5,rsa)(^PKC)
            }rsa(PKS)||(SharedSecret C S Kr);
4. S -> C:  S, {Kcs,C,Ts2}des(Ks), {C,S,Kr,Kcs,Ts2}des(Kr)||
            (SharedSecret C S Kcs);
5. C -> S:  S, {Kcs,C,Ts2}des(Ks), {C,Ts3}des(Kcs)||
            (SharedSecret C S Kcs);

GOALS:
1. C Believes (PublicKey S rsa PKS);
2. S Believes (PublicKey C rsa PKC);
3. S Possesses Kr;
   S Believes (SharedSecret C S Kr);
   S Believes (C Possesses Kr);
   S Believes (C Believes (SharedSecret C S Kr));
4. C Possesses Kcs;
   C Believes (SharedSecret C S Kcs);
```

```
      C Believes (S Possesses Kcs);
      C Believes (S Believes (SharedSecret C S Kcs));
   5. S Believes (C Possesses Kcs);
      S Believes (C Believes (SharedSecret C S Kcs));
```

Appendix B: ASN.1 Notation for 'Certificate' as specified in Annex G of X.509.

```
   Certificate :: = SIGNED SEQUENCE{
           version[0]              Version DEFAULT 1988,
           serialNumber            SerialNumber,
           signature               AlgorithmIdentifier,
           issuer                  Name,
           validity                Validity,
           subject                 Name,
           subjectPublicKeyInfo    SubjectPublicKeyInfo
   }

   Version ::= INTEGER {1988(0)}

   SerialNumber ::= INTEGER

   Validity ::= SEQUENCE{
           notBefore               UTCTime
           notAfter                UTCTime
   }

   SubjectPublicKeyInfo ::= SEQUENCE{
           algorithm               AlgorithmIdentifier
           subjectPublicKey        BIT STRING
   }

   AlgorithmIdentifier ::= SEQUENCE{
           algorithm               OBJECT IDENTIFIER,
           parameters              ANY DEFINED BY algorithm OPTIONAL
   }

   SIGNED MACRO :: =
   BEGIN
   TYPE NOTATION ::= type (ToBeSigned)
   VALUE NOTATION ::= value(VALUE
                    SEQUENCE{
                    ToBeSigned,
                    AlgorithmIdentifier,
                    -- of the algorithm used to generate the signature
                    ENCRYPTED OCTET STRING
                    -- where the octet string is the result
                    -- of the hashing of the value of
                    -- "ToBeSigned"--
                    }
```

```
          )
   END -- of SIGNED
```

Appendix C: PKCS #7 Definition for 'ContentInfo'

'ContentInfo' is a general data type defined in, and exported by PKCS #7 for use in conjunction with operations such as enveloping and signing. It is used as a "wrapper" for calling specific data types such as 'signedData' and 'envelopedData', which themselves are not exported from PKCS #7. Nesting is permitted explicitly by the recursive nature of the 'ContentInfo' syntax.

```
ContentInfo ::= SEQUENCE{
        caddr[11]                   HostAddresses OPTIONAL
        contentType                 ContentType,
        content[0]     EXPLICIT ANY DEFINED BY contentType OPTIONAL
}
```

contentType
: an object identifier, with six content types defined in PKCS #7, Section 14: data, signedData, envelopedData, signedAndEnvelopedData, digestedData, and encryptedData.

content
: the content field is optional, and if the field is not present, its intended value must be supplied by other means. Its type is defined along with the object identifier for contentType.

Appendix D: Discussion on Mandatory Inclusion of 'ccert' in PKTGS-REQ

In this proposal, inclusion of the client's certificate, ccert, is mandatory. However there are several situations where inclusion of the client certificate might be superfluous.

1. the server intends to check with the CA every time to assure the certificate is fresh and not revoked.
2. the intended server is the CA.

Thus, one could consider making the inclusion of ccert in the PKTGS-REQ optional. If the inclusion of ccert is optional, three additional changes to the protocol are required.

1. Inclusion of a TGS-REP message which says, in effect, "please resubmit this request including the certificate.

2. The inclusion of a flag in the SCERT-REP message indicating the server's preference for receiving or not receiving the certificate. The use of such a flag can dramatically reduce

the number of rejected PKTGS-REQ messages due to failure to include ccert.

3. If ccert is not included in the PKTGS-REQ, than the following fields must be included in its place so that the server can obtain the correct certificate:
   - cname and crealm
   - issuer of the client's certificate
   - serial number of the client's certificate.

Actually, cname and crealm are redundant.

Both for simplicity and because we believe that the costs of sending a superfluous certificate on occasion are outweighed by the costs likely to be incurred due to retransmission of PKTGS-REQ messages because a certificate was not originally included, we have chosen to make inclusion of ccert mandatory. We welcome further discussion on this issue.