INTERNET-DRAFT
draft-ietf-cat-kerberos-pk-cross-07.txt
Updates: RFC 1510
expires May 15, 2001

Matthew Hur
Cisco Systems
Brian Tung
Tatyana Ryutov
Clifford Neuman
ISI
Ari Medvinsky
Keen.com
Gene Tsudik
UC Irvine
Bill Sommerfeld
Sun Microsystems

Public Key Cryptography for Cross-Realm Authentication in Kerberos

0. Status Of this Memo

This document is an Internet-Draft and is in full conformance with all provisions 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.

To learn the current status of any Internet-Draft, please check the ``1id-abstracts.txt'' listing contained in the Internet-Drafts Shadow Directories on ftp.ietf.org (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-cross-07.txt, and expires May 15, 2001. Please send comments to the authors.

1. Abstract

This document defines extensions to the Kerberos protocol

specification [1] to provide a method for using public key cryptography to enable cross-realm authentication. The methods defined here specify the way in which message exchanges are to be used to transport cross-realm secret keys protected by encryption under public keys certified as belonging to KDCs.

2. Introduction

Symmetric and asymmetric key systems may co-exist within hybrid architectures in order to leverage the advantages and mitigiate issues within the respective systems. An example of a hybrid solution that may employ both symmetric and asymmetric technologies is Kerberos ciphersuires in TLS [KERBTLS] which utilizes the Kerberos protocol [KERB] [KERB94] in conjunction with TLS [TLS] which has commonly been thought of as a public key protocol.

The Kerberos can leverage the advantages provided by public key cryptography. PKINIT [PKINIT] describes the use of public key cryptography in the initial authentication exchange in Kerberos. PKTAPP [PKTAPP] describes how an application service can essentially issue a kerberos ticket to itself after utilizing public key cryptography for authentication. This specification describes the use of public key crpytography in cross-realm authentication.

Without the use of public key cryptography, administrators must maintain separate keys for every realm which wishes to exchange authentication information with another realm (which implies n(n-1) keys), or they must utilize a hierachichal arrangement of realms, which may increase network traffic and complicate the trust model by requiring evaluation of transited realms.

Even with the multi-hop cross-realm authentication, there must be some way to locate the path by which separate realms are to be transited. The current method, which makes use of the DNS-like realm names typical to Kerberos, requires trust of the intermediate KDCs.

PKCROSS utilizes a public key infrastructure (PKI) [X509] to simplify the administrative burden of maintaining cross-realm keys. Such usage leverages a PKI for a non-centrally-administratable environment (namely, inter-realm). Thus, a shared key for cross-realm authentication can be established for a set period of time, and a remote realm is able to issue policy information that is returned to itself when a client requests cross-realm authentication. Such policy information may be in the form of restrictions [NEUMAN]. Furthermore, these methods are transparent to the client; therefore, only the KDCs need to be modified to use them. In this way, we take advantage of the the distributed trust management capabilities of public key crypography while maintaining the advantages of localized trust management provided by Kerberos.

Although this specification utilizes the protocol specfied in the PKINIT specification, it is not necessary to implement client changes in order to make use of the changes in this document.

3. Objectives

The objectives of this specification are as follows:

1. Simplify the administration required to establish Kerberos cross-realm keys.

2. Avoid modification of clients and application servers.

3. Allow remote KDC to control its policy on cross-realm keys shared between KDCs, and on cross-realm tickets presented by clients.

4. Remove any need for KDCs to maintain state about keys shared with other KDCs.

5. Leverage the work done for PKINIT to provide the public key protocol for establishing symmetric cross realm keys.

4. Definitions

The following notation is used throughout this specification:

```
KDC_l ........... local KDC
KDC_r ........... remote KDC
XTKT_(l,r) ...... PKCROSS ticket that the remote KDC issues to the
                  local KDC
TGT_(c,r) ....... cross-realm TGT that the local KDC issues to the
                  client for presentation to the remote KDC
```

This specification defines the following new types to be added to the Kerberos specification:

```
  PKCROSS kdc-options field in the AS_REQ is bit 9
  TE-TYPE-PKCROSS-KDC       2
  TE-TYPE-PKCROSS-CLIENT    3
```

This specification defines the following ASN.1 type for conveying policy information:

```
CrossRealmTktData ::= SEQUENCE OF TypedData
```

This specification defines the following types for policy information conveyed in CrossRealmTktData:

```
  PLC_LIFETIME              1
  PLC_SET_TKT_FLAGS         2
  PLC_NOSET_TKT_FLAGS       3
```

```
TicketExtensions are defined per the Kerberos specification
[KERB-REV]:
TicketExtensions ::= SEQUENCE OF TypedData
  Where
    TypedData ::=   SEQUENCE {
        data-type[0]   INTEGER,
        data-value[1]  OCTET STRING OPTIONAL
    }
```

5. Protocol Specification

We assume that the client has already obtained a TGT. To perform cross-realm authentication, the client does exactly what it does with ordinary (i.e. non-public-key-enabled) Kerberos; the only changes are in the KDC; although the ticket which the client forwards to the remote realm may be changed. This is acceptable since the client treats the ticket as opaque.

5.1. Overview of Protocol

The basic operation of the PKCROSS protocol is as follows:

1. The client submits a request to the local KDC for credentials for the remote realm. This is just a typical cross realm request that may occur with or without PKCROSS.

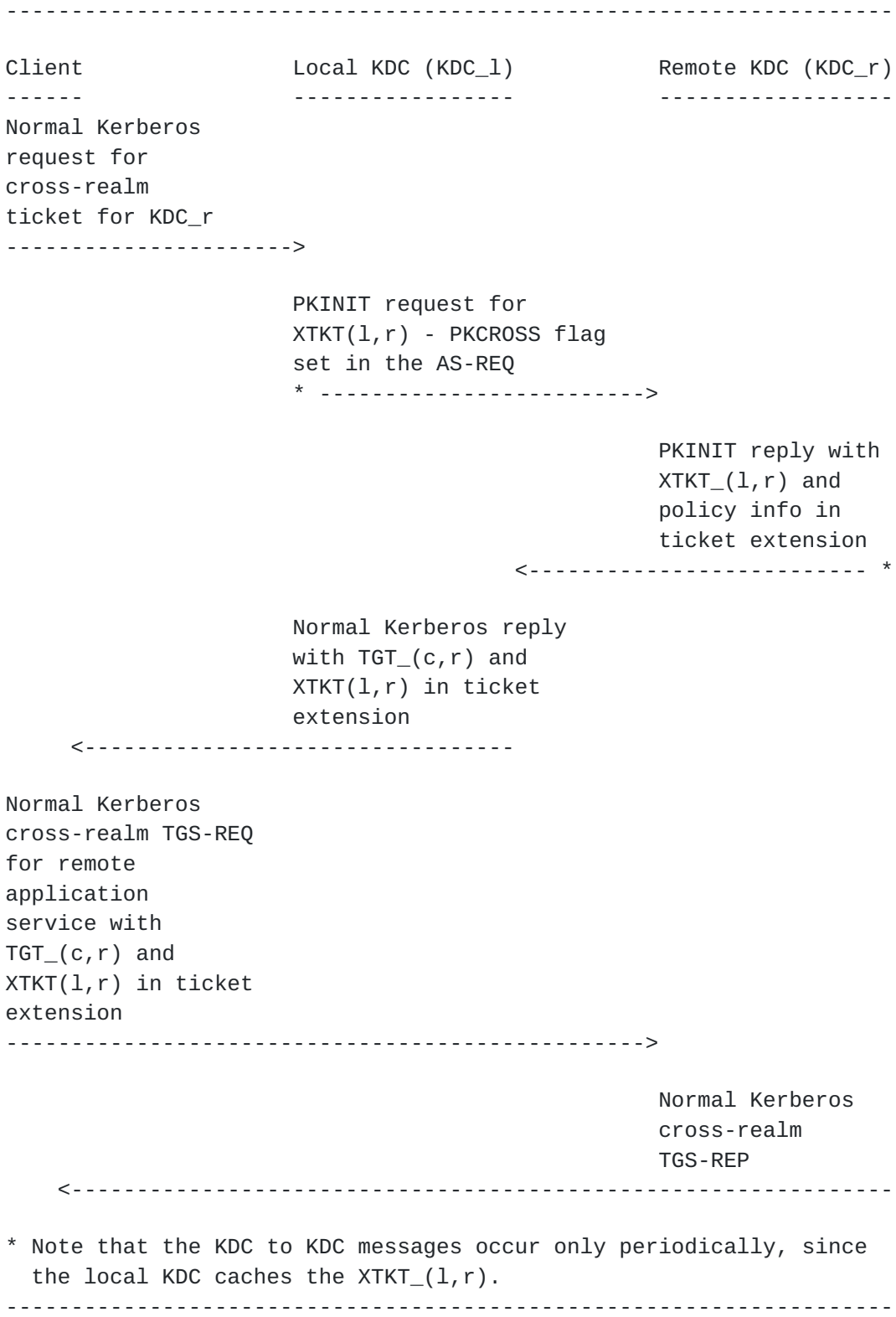
2. The local KDC submits a PKINIT request to the remote KDC to obtain a "special" PKCROSS ticket. This is a standard PKINIT request, except that PKCROSS flag (bit 9) is set in the kdc-options field in the AS_REQ.

3. The remote KDC responds as per PKINIT, except that the ticket contains a TicketExtension, which contains policy information such as lifetime of cross realm tickets issued by KDC_l to a client. The local KDC must reflect this policy information in the credentials it forwards to the client. Call this ticket XTKT_(l,r) to indicate that this ticket is used to authenticate the local KDC to the remote KDC.

4. The local KDC passes a ticket, TGT_(c,r) (the cross realm TGT between the client and remote KDC), to the client. This ticket contains in its TicketExtension field the ticket, XTKT_(l,r), which contains the cross-realm key. The TGT_(c,r) ticket is encrypted using the key sealed in XTKT_(l,r). (The TicketExtension field is not encrypted.) The local KDC may optionally include another TicketExtension type that indicates the hostname and/or IP address for the remote KDC.

5. The client submits the request directly to the remote KDC, as before.

6. The remote KDC extracts XTKT_(l,r) from the TicketExtension in order to decrypt the encrypted part of TGT_(c,r).



* Note that the KDC to KDC messages occur only periodically, since the local KDC caches the XTKT_(l,r).

Sections 5.2 through 5.4 describe in detail steps 2 through 4 above. Section 5.6 describes the conditions under which steps 2 and 3 may be skipped.

Note that the mechanism presented above requires infrequent KDC to KDC communication (as dictated by policy - this is discussed later). Without such an exchange, there are the following issues:

1) KDC_l would have to issue a ticket with the expectation that KDC_r will accept it.
2) In the message that the client sends to KDC_r, KDC_l would have to authenticate KDC_r with credentials that KDC_r trusts.
3) There is no way for KDC_r to convey policy information to KDC_l.
4) If, based on local policy, KDC_r does not accept a ticket from KDC_l, then the client gets stuck in the middle. To address such an issue would require modifications to standard client processing behavior.

Therefore, the infreqeunt use of KDC to KDC communication assures that inter-realm KDC keys may be established in accordance with local policies and that clients may continue to operate without modification.

5.2. Local KDC's Request to Remote KDC

When the local KDC receives a request for cross-realm authentication, it first checks its ticket cache to see if it has a valid PKCROSS ticket, XTKT_(l,r). If it has a valid XTKT_(l,r), then it does not need to send a request to the remote KDC (see section 5.5).

If the local KDC does not have a valid XTKT_(l,r), it sends a request to the remote KDC in order to establish a cross realm key and obtain the XTKT_(l,r). This request is in fact a PKINIT request as described in the PKINIT specification; i.e., it consists of an AS-REQ with a PA-PK-AS-REQ included as a preauthentication field. Note, that the AS-REQ MUST have the PKCROSS flag (bit 9) set in the kdc_options field of the AS-REQ. Otherwise, this exchange exactly follows the description given in the PKINIT specification.

5.3. Remote KDC's Response to Local KDC

When the remote KDC receives the PKINIT/PKCROSS request from the local KDC, it sends back a PKINIT response as described in the PKINIT specification with the following exception: the encrypted part of the Kerberos ticket is not encrypted with the krbtgt key; instead, it is encrypted with the ticket granting server's PKCROSS key. This key, rather than the krbtgt key, is used because it encrypts a ticket used for verifying a cross realm request rather than for issuing an application service ticket. Note that, as a

matter of policy, the session key for the XTKT_(l,r) MAY be of greater strength than that of a session key for a normal PKINIT reply, since the XTKT_(l,r) SHOULD be much longer lived than a normal application service ticket.

In addition, the remote KDC SHOULD include policy information in the XTKT_(l,r). This policy information would then be reflected in the cross-realm TGT, TGT_(c,r). Otherwise, the policy for TGT_(c,r) would be dictated by KDC_l rather than by KDC_r. The local KDC MAY enforce a more restrictive local policy when creating a cross-realm ticket, TGT_(c,r). For example, KDC_r may dictate a lifetime policy of eight hours, but KDC_l may create TKT_(c,r) with a lifetime of four hours, as dictated by local policy. Also, the remote KDC MAY include other information about itself along with the PKCROSS ticket. These items are further discussed in section 6 below.

5.4. Local KDC's Response to Client

Upon receipt of the PKINIT/CROSS response from the remote KDC, the local KDC formulates a response to the client. This reply is constructed exactly as in the Kerberos specification, except for the following:

A) The local KDC places XTKT_(l,r) in the TicketExtension field of the client's cross-realm, ticket, TGT_(c,r), for the remote realm.
   Where
      data-type equals 3 for TE-TYPE-PKCROSS-CLIENT
      data-value is ASN.1 encoding of XTKT_(l,r)

B) The local KDC adds the name of its CA to the transited field of TGT_(c,r).

5.5 Remote KDC's Processing of Client Request

When the remote KDC, KDC_r, receives a cross-realm ticket, TGT_(c,r), and it detects that the ticket contains a ticket extension of type TE-TYPE-PKCROSS-CLIENT, KDC_r must first decrypt the ticket, XTKT_(l,r), that is encoded in the ticket extension. KDC_r uses its PKCROSS key in order to decrypt XTKT_(l,r). KDC_r then uses the key obtained from XTKT_(l,r) in order to decrypt the cross-realm ticket, TGT_(c,r).

KDC_r MUST verify that the cross-realm ticket, TGT_(c,r) is in compliance with any policy information contained in XTKT_(l,r) (see section 6). If the TGT_(c,r) is not in compliance with policy, then the KDC_r responds to the client with a KRB-ERROR message of type KDC_ERR_POLICY.

5.6. Short-Circuiting the KDC-to-KDC Exchange

As we described earlier, the KDC to KDC exchange is required only for establishing a symmetric, inter-realm key. Once this key is established (via the PKINIT exchange), no KDC to KDC communication is required until that key needs to be renewed. This section describes the circumstances under which the KDC to KDC exchange described in Sections 5.2 and 5.3 may be skipped.

The local KDC has a known lifetime for TGT_(c,r). This lifetime may be determined by policy information included in XTKT_(l,r), and/or it may be determined by local KDC policy. If the local KDC already has a ticket XTKT(l,r), and the start time plus the lifetime for TGT_(c,r) does not exceed the expiration time for XTGT_(l,r), then the local KDC may skip the exchange with the remote KDC, and issue a cross-realm ticket to the client as described in Section 5.4.

Since the remote KDC may change its PKCROSS key (referred to in Section 5.2) while there are PKCROSS tickets still active, it SHOULD cache the old PKCROSS keys until the last issued PKCROSS ticket expires. Otherwise, the remote KDC will respond to a client with a KRB-ERROR message of type KDC_ERR_TGT_REVOKED.

6. Extensions for the PKCROSS Ticket

As stated in section 5.3, the remote KDC SHOULD include policy information in XTKT_(l,r). This policy information is contained in a TicketExtension, as defined by the Kerberos specification, and the authorization data of the ticket will contain an authorization record of type AD-IN-Ticket-Extensions. The TicketExtension defined for use by PKCROSS is TE-TYPE-PKCROSS-KDC.

Where
  data-type equals 2 for TE-TYPE-PKCROSS-KDC
  data-value is ASN.1 encoding of CrossRealmTktData

CrossRealmTktData ::= SEQUENCE OF TypedData

CrossRealmTktData types and the corresponding data are interpreted as follows:

type	value	interpretation	ASN.1 data encoding
PLC_LIFETIME	1	lifetime (in seconds) for TGT_(c,r) - cross-realm tickets issued for clients by TGT_l	INTEGER

PLC_SET_TKT_FLAGS	2	TicketFlags that must be set - format defined by Kerberos specification	BITSTRING
PLC_NOSET_TKT_FLAGS	3	TicketFlags that must not be set - format defined by Kerberos specification	BITSTRING

Further types may be added to this table.

7. Usage of Certificates

In the cases of PKINIT and PKCROSS, the trust in a certification authority is equivalent to Kerberos cross realm trust. For this reason, an implementation MAY choose to use the same KDC certificate when the KDC is acting in any of the following three roles:

1) KDC is authenticating clients via PKINIT
2) KDC is authenticating another KDC for PKCROSS
3) KDC is the client in a PKCROSS exchange with another KDC

Note that per PKINIT, the KDC X.509 certificate (the server in a PKINIT exchange) MUST contain the principal name of the KDC in the subjectAltName field.

8. Transport Issues

Because the messages between the KDCs involve PKINIT exchanges, and PKINIT recommends TCP as a transport mechanism (due to the length of the messages and the likelihood that they will fragment), the same recommendation for TCP applies to PKCROSS as well.

9. Security Considerations

Since PKCROSS utilizes PKINIT, it is subject to the same security considerations as PKINIT. Administrators should assure adherence to security policy - for example, this affects the PKCROSS policies for cross realm key lifetime and for policy propogation from the PKCROSS ticket, issued from a remote KDC to a local KDC, to cross realm tickets that are issued by a local KDC to a client.

10. Bibliography

[KERBTLS] A. Medvinsky and M. Hur, "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)", RFC 2712,

October 1999.

[KERB] J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

[TLS] T. Dierks and C. Allen, "The TLS Protocol, Version 1.0", RFC 2246, January 1999.

[PKINIT] B. Tung, C. Neuman, M. Hur, A. Medvinsky, S. Medvinsky, J. Wray, J. Trostle. Public Key Cryptography for Initial Authentication in Kerberos. draft-ietf-cat-kerberos-pk-init-12.txt

[PKTAPP] A. Medvinsky, M. Hur, S. Medvinsky, C. Neuman. Public Key Utilizing Tickets for Application Servers (PKTAPP). draft-ietf-cat-kerberos-pk-tapp-03.txt

[X509] ITU-T (formerly CCITT) Information technology - Open Systems Interconnection - The Directory: Authentication Framework Recommendation X.509 ISO/IEC 9594-8

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

[KERB94] B.C. Neuman, Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, IEEE Communications, 32(9):33-38. September 1994.

[KERB-REV] C.Neuman, J. Kohl, T. Ts'o. The Kerberos Network Authentication Service (V5). draft-ietf-cat-kerberos-revisions-07.txt

11. Authors' Addresses

Matthew Hur
Cisco Systems
500 108th Ave. NE, Suite 500
Bellevue, WA 98004
Phone:
EMail: mhur@cisco.com

Brian Tung
Tatyana Ryutov
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, tryutov, bcn}@isi.edu

Ari Medvinsky
Keen.com
2480 Sand Hill Road, Suite 200
Menlo Park, CA 94025
Phone +1 650 289 3134
E-mail: ari@keen.com

Gene Tsudik
ICS Dept, 458 CS Building
Irvine CA 92697-3425
Phone: +1 310 448 9329
E-Mail: gts@ics.uci.edu

Bill Sommerfeld
Sun Microsystems
E-Mail: sommerfeld@east.sun.com