The KX.509 Protocol

William Doster
Marcus Watts
Dan Hyde

University of Michigan

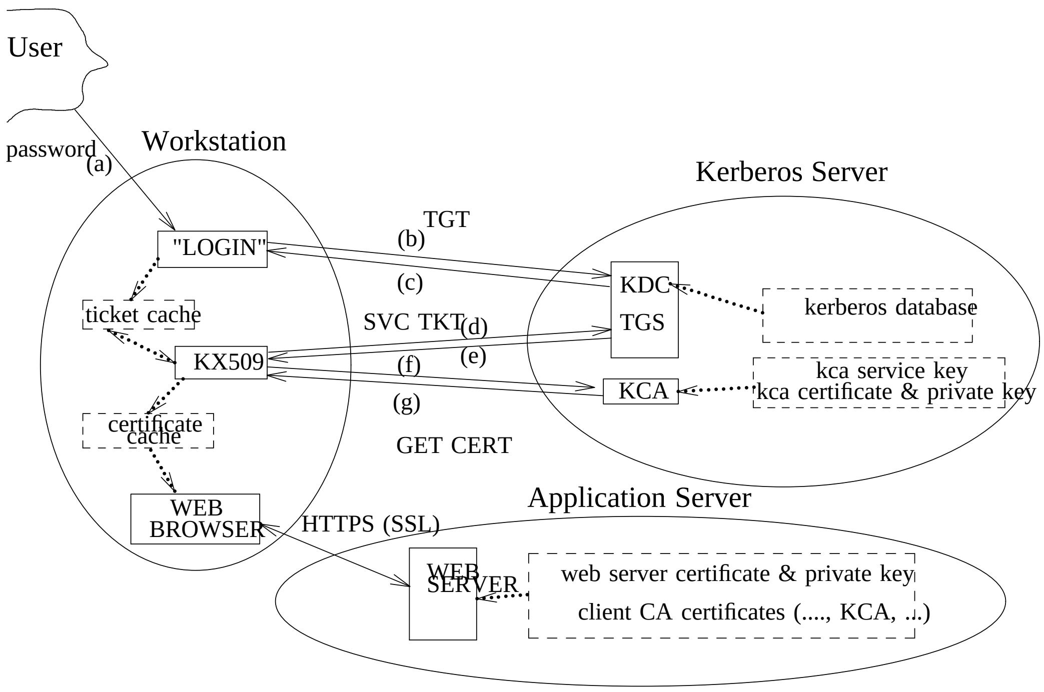
ABSTRACT

This document describes the KX.509 protocol. Using this protocol, a workstation can acquire a temporary (or ‘‘junk’’) X.509 certificate on behalf of a user based on having a kerberos ticket for that user.

1. The Big Picture

In normal use, running kx509 will be an invisible phase of the login process. During ‘‘login’’, the user will supply a password, which will never leave the workstation. The password is used to acquire a K4 ticket granting ticket (TGT), then scrubbed from memory. Once the TGT is acquired by ‘‘login’’, kx509 starts. First, kx509 generates a RSA public/private key-pair. Next, using the TGT it acquires a Kerberos service ticket for the KCA (Kerberized Certificate Authority), and uses this to send the public half of its key-pair. The KCA will decrypt the service ticket, verify the integrity of the incoming packet, determine the identity of the user, and use the session key to send back a short-lived X.509 certificate. If all goes well, kx509 then stores the certificate and key-pair in the local certificate cache.

The full picture looks something like this:



The protocol that is described in this document refers to the lines labelled (f) and (g) in the above picture. This section provides the background needed to understand the roles of kx509 and the KCA. The remainder of this document will concentrate on (f) and (g), the actual protocol. Additional documentation (not yet written) will describe other aspects of this big picture.

On MicroSoft Windows machines, the X.509 certificate is stored in the registry. Web browsers under Windows already know how to get at certificates stored in the registry. Under UNIX, the certificate and key pair are stored as a special entry in the kerberos ticket cache. For Unix and the Macintosh, a pkcs11 shared library is provided that allows the web browser to get at the certificate. This entire process should be invisible to normal users, who will only know what they need to supply their password only once (‘‘single-sign-on,’’) and that this suffices to enable their web browser to identify them to remote services.

A mechanism should be provided to erase the private keys and certificates gotten by kx509 between user login sessions or upon request by the user. Ideally, kx509 certificates and key pairs should be destroyed whenever the kerberos ticket cache is destroyed or recreated.

2. The Protocol

kx509 uses UDP to talk to the kerberos certificate authority. The request packet contains a protocol version identifier, a kerberos 4 ap request, integrity check data on the request, and a public key generated by kx509. The response packet contains a protocol version identifier, and may contain one or more of an error code, integrity check data on the response, a certificate, and an error message.

The KCA needs to have a kerberos service identity. This identity is always `cert.x509@@`*REALM*. The identity `cert.kx509@@`*REALM* is also reserved, but not used in the protocol exchange. Some implementations of kx509 use the kerberos-4 ticket cache as the certificate cache, and store the certificate as a ‘‘pseudo-ticket’’ under this identity. The KCA is expected to be run on the kerberos database machines. kx509 client implementations may supply a mechanism to specify the use of an alternate kerberos service identity or KCA hosts, for debugging purposes.

Let v = the version string, the first 4 bytes of the `resquest` or `response`,
c = the client’s identity,
and q = the KCA.
$E_K(P)$ denotes encryption by key K of P,
K_q, the key of KCA,
$K_{c,q}$, the session key,
$T_{c,q}$=q,$E_{K_q}(c$,ipaddr,start,$K_{c,q})$, a kerberos ticket,
a=$A_{c,q}$=$E_{K_{c,q}}(c$,ipaddr,$t)$,$E_{K_q}(T_{c,q})$, the kerberos 4 authenticator,
and p = a public key generated by the client.

The packet described by (f) in the big picture, which goes from kx509 to KCA, is a `request`. It contains

v	version
a	kerberos ap req
$HMAC_{K_{c,q}}(v,p)$	check data
p	public key

The packet described by (g) in the big picture, which goes from the KCA back to kx509, is a `response`. It contains

v	version string
s	error code
$HMAC_{K_{c,q}}(v,x,e,s)$	check data
x	x.509 certificate
e	error message

The only encrypted information in the protocol is that used by kerberos itself; the public key and certificate are transmitted in the clear and HMAC-SHA-1 is used for data integrity checking.

3. Secure Hash Algorithm

HMAC is described in [HMAC]. SHA-1 [SHA] is used for the underlying hash algorithm. All 160 bits are sent. Some uses of HMAC use a truncated version of the hash, this is done when date to be hashed has a predictable, but secret value, to make it harder to guess the data. In this case, the data is easy to predict (as it's been sent in the clear in the same packet), so there is no reason to dicard any hash information.

4. Protocol Version Numbers

The first 4 bytes of the packet contain version identifying information. The first two bytes of data are reserved. They should be set to 0 when sent, and should be ignored when received. The 3rd byte is the protocol version major number, and the 4th byte is the protocol version minor number. The version of the protocol described in this document is 2.0, which is expressed with the bytes 2, 0. So, a full header of all 2.0 packets should be set to look like this:

```
|0 |0 |2 |0 |... |
```

It is expected that the major number should be stepped for incompatible changes in the protocol, and the minor number may be stepped for compatible minor changes in the protocol. A server that receives a packet with a supported major.minor # should attempt to return the exact same version if possible, otherwise it may return a packet with the same major number and a different minor number. In general, clients are only expected to support one protocol version, but may implement an option selected at compile-time or run-time to select a different protocol version. Clients should accept packets that match the same major number as the request, and refuse to process packets with a different major number than the request.

Protocol version 1.0 is not secure, and should not be used except for testing purposes. Production servers should never support version 1.0.

5. ASN.1 Packet Data Structure

The remainder of the packet is packed using DER, which can be described as follows using ASN.1.

```
-- kx509
KX509 DEFINITIONS ::=
BEGIN

kx509Request ::= SEQUENCE {
    ap-req OCTET STRING,
    pk-hash OCTET STRING,
    pk-key OCTET STRING
}

kx509Response ::= SEQUENCE {
    error-code[0]    INTEGER DEFAULT 0,
    hash[1]      OCTET STRING OPTIONAL,
    certificate[2]   OCTET STRING OPTIONAL,
    e-text[3]    VisibleString OPTIONAL
}
END
```

An error decoding the request or response packet indicates that that particular packet was corrupted, and should be ignored. The server should not return any sort of packet in the case of a malformed request. Both request and response packets are prefixed with a 4-byte version string, which is not part of the ASN.1 data structure. Not all combinations of optional data in the response make sense. The legal combinations, and their interpretation, are described later.

6. The Request Packet

The request contains a number of objects that may themselves be DER encoded, stored as opaque vectors using the ASN.1 data type `OCTET STRING`. The reason to handle things this way is to ensure that they can be passed as byte strings of definite length to hash functions, before any further interpretion is made of them by the local ASN.1 library.

The `ap-req` is a kerberos 4 application request, as made by the routine `krb_mk_req` in the kerberos-4 library. The application request includes cleartext information needed by `krb_rd_req` to locate the correct service key, the ticket encrypted by the KCA kerberos-4 service key, and request information encrypted under the session key. The first byte of the application request is always KRB_PROT_VERSION (4), and the second byte is always AUTH_MSG_APPL_REQUEST | HOST_BYTE_ORDER (6 or 7). Future versions of KCA may support kerberos-5 style application requests. It is believed that kerberos-5 style application requests can be uniquely distinguished from kerberos-4 by inspecting the first bytes. The application request also includes a 32-bit ‘‘checksum’’ computed by the sender. This checksum is not checked or used by KCA. It may be set to any constant or random value. If it is set to a random value, care should be taken that it does not leak any secret information from the client machine, including state information used to construct the private/public keypair.

The `pk-hash` is an octet-string containing the 20 byte SHA-1 HMAC of the request. SHA-1 is described in FIPS PUB 180-1. HMAC is described in RFC 2104 and RFC 2202. The key used is the session key. Kerberos 4 session keys are always 8 bytes. Kerberos 5 session keys may be of other lengths. The data passed to SHA-1 contists of the 4-byte version string at the start of the packet and the octet string for `pk-key`.

The `pk-key` is an octet string containing a public key. This key and its corresponding private key are generated locally by the kx509 client before the server is contacted. The public key is converted to DER representation and this is then stored in an octet-string in the request.

7. Handling the Request

When the server receives a request, it may make several sanity checks on the request. This includes checking to see if the application request can be decoded, verifying the hash on the request, and imposing a minimum length or other simple sanity checks on the public key. If the application request cannot be decoded, the server may choose to return an unauthentic error response with a status code and error message, but no hash or certificate. The server should contain logic to limit the number of unauthentic errors that can be returned in a given time. If there are no errors in the application request (lifetime within bounds, etc.) the server should always supply a hash on the response packet.

8. The Response Packet

The response contains a number of objects that are optional. Although the actual wire protocol makes no distinction as to which are present and which are not, there are definite logical rules the server and client should respect as to which fields should or should not be present. The server SHOULD NOT transmit a zero error-code. The client MUST treat a missing `error-code` as if it were 0. The following cases are possible for responses:

```
1   certificate                      hash
2   error-code    error-message     hash
3   error-code    error-message
```

The first case is returned when the server successfully generates a certificate for the user.

The 2nd case is returned when the server is able to successfully authenticate the user and extract the session key, but is unable for some reason to generate a certificate.

The 3rd case is allowed if the server is unable to successfully authenticate the user, and is intended to allow the server to return useful but unauthenticated information to the user. (Currently, if the KCA sanity check fails, the client times out and receives no error-code nor error-message).

The `error-code` should be treated as an unsigned 32-bit integer and only the non-zero bytes, in network byte order (most significant byte first) should be processed. Some implementations of ASN.1 may pass very large 32-bit unsigned integers as negative integers; for the purpose of computing the hash the client should ignore any such bogus indication.

The `hash` on a response is computed using SHA-1 HMAC, as for the request. The data that is passed to this function includes, in order: the 4-byte version string at the start of the packet, the `error-code` if present, `certificate` if present, and `error-message` if present.

The `certificate` is the DER encoded representation of the certificate. It is then encapsulated as an opaque vector in the actual packet sent to facilitate computing the hash before unwrapping the certificate.

The `error-message` is sent as a `VisibleString` instead of a `OCTET-STRING` because of its special interpretation as containing printable characters. If the client machine does not use the ASCII character set, it may wish to translate this error message into a different character set. If it does so, the hash function described above MUST be done before the translation. The error message, as shipped, SHOULD NOT contain any NUL characters. The client MUST be capable of properly processing an error message returned without a trailing NUL. The client MAY ignore any part of the error message after the first NUL character for display purposes. The client MUST process the entire `error-message`, including any buried NUL characters, when computing the `hash`.

9. Interpresting Optional Data in kx509 Responses

If the response contains a certificate, it must contain a hash. If the hash is missing, a protocol error has occurred. The client should continue to look for a valid packet, but may report that a protocol error has occured if no valid packet is received, as described below. A response with a certificate should not contain an error-code, but if it does, the error-code must be zero. A response with a certificate may contain an error message. The error message must be considered in the hash calculation, but may be otherwise ignored. Client implementations may choose to treat the error message in a response with a valid certificate and hash as informational text and display it, if this is appropriate.

If the response does not contain a certificate, it must contain an error message and a non-zero error code. In this case, a zero error code is a protocol error, and a missing error message is a protocol error. Error messages may contain a hash. If the hash is there, the client should check it, and if the hash does not check out, a protocol error has occurred. An error message with a non-zero error code and a valid hash should be regarded as authoritative for that server, and the client should not send any additional requests to that server. The client may continue to attempt to do further retries for other servers in the same domain.

In case of a protocol error, the client should always regard the packet as an unauthentic error. When reporting such packets, the client should report the protocol error, and may accompany this report with diagnostic information from the packet. An error message without a valid hash, or upon which a protocol error has occurred, is not authoritative. If no such packet is received, the client should report the error contained in the first unauthentic error response received, and may note that the error message is not authenticate.

10. Areas for Further Research

1. The hash on the request does not include the authenticator. Should it?
2. Should the checksum in the kerberos-4 ap_req be better used?
3. The use of numbered tags for the ASN.1 reply, but not in the request, is weird.
4. It has been suggested that the KCA ought to be willing to supply its public certificate as well. Does this mean it would be useful to have 2 kinds of requests? Would it be useful if requests had an op-code field?

11. References

[HMAC] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[KRB5] Kohl J. and C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

[SHA] NIST, FIPS PUB 180-1: Secure Hash Standard, April 1995.

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