        LIPKEY - A Low Infrastructure Public Key Mechanism Using SPKM

Status of this Memo

Copyright Notice

Abstract

   This memorandum describes a method whereby one can use GSS-API
   [RFC2078] to supply a secure channel between a client and server,
   authenticating the client with a password, and a server with a public
   key certificate. As such, it is analogous to the common low
   infrastructure usage of the Transport Layer Security (TLS) protocol
   [RFC2246].

   The method leverages the existing Simple Public Key Mechanism (SPKM)
   [RFC2025], and is specified as a separate GSS-API mechanism (LIPKEY)
   layered above SPKM.

Table of Contents

1. Introduction

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in [RFC2119].

   This memorandum describes a new security mechanism under the GSS-API
   called the Low Infrastructure Public Key Mechanism (LIPKEY). GSS-API
   provides a way for an application protocol to implement
   authentication, integrity, and privacy. TLS is another way. While TLS

is in many ways simpler for an application to incorporate than GSS-
API, there are situations where GSS-API might be more suitable.
Certainly this is the case with application protocols that run over
connectionless protocols. It is also the case with application
protocols such as ONC RPC [RFC1831] [RFC2203], which have their own
security architecture, and so do not easily mesh with a protocol like
TLS that is implemented as a layer that encapsulates the upper layer
application protocol. GSS-API allows the application protocol to
encapsulate as much of the application protocol as necessary.

Despite the flexibility of GSS-API, it compares unfavorably with TLS
with respect to the perception of the amount of infrastructure
required to deploy it. The better known GSS-API mechanisms, Kerberos
V5 [RFC1964] and SPKM require a great deal of infrastructure to set
up. Compare this to the typical TLS deployment scenario, which
consists of a client with no public key certificate accessing a
server with a public key certificate. The client:

*   obtains the server's certificate,

*   verifies that it was signed by a trusted Certification Authority
        (CA),

*   generates a random session symmetric key,

*   encrypts the session key with the server's public key, and

*   sends the encrypted session key to the server.

At this point, the client and server have a secure channel. The
client can then provide a user name and password to the server to
authenticate the client. For example, when TLS is being used with the
http protocol, once there is a secure channel, the http server will
present the client with an html page that prompts for a user name and
password. This information is then encrypted with the session key and
sent to the server. The server then authenticates the client.

Note that the client is not required to have a certificate for itself
to identify and authenticate it to the server. In addition to a TLS
implementation, the required security infrastructure includes a
public key certificate and password database on the server, and a
list of trusted CAs and their public keys on the client. Most
operating systems that the http server would run on already have a
native password database, so the net additional infrastructure is a
server certificate and CA list. Hence the term "low infrastructure
security model" to identify this typical TLS deployment scenario.

By using unilateral authentication, and using a mechanism resembling the SPKM-1 mechanism type, SPKM can offer many aspects of the previously described low infrastructure security model. An application that uses GSS-API is certainly free to use GSS-API's GSS_Wrap() routine to encrypt a user name and password and send them to the server, for it to decrypt and verify.

Applications often have application protocols associated with them, and there might not be any provision in the protocol to specify a password. Layering a thin GSS-API mechanism over a mechanism resembling SPKM-1 can mitigate this problem. This can be a useful approach to avoid modifying applications that have already bound to GSS-API, assuming the applications are not statically bound to specific GSS-API mechanisms. The remainder of this memorandum defines the thin mechanism: the Low Infrastructure Public Key Mechanism (LIPKEY).

2. LIPKEY's Requirements of SPKM

SPKM-1 with unilateral authentication is close to the desired low infrastructure model described earlier. This section describes some additional changes to how SPKM-1 operates in order to realize the low infrastructure model. These changes include some minor changes in semantics. While it would be possible to implement these semantic changes within an SPKM-1 implementation (including using the same mechanism type Object Identifier (OID) as SPKM-1), the set of changes stretch the interpretation of RFC 2025 to the point where compatibility would be in danger. A new mechanism type, called SPKM-3, is warranted. LIPKEY requires that the SPKM implementation support SPKM-3. SPKM-3 is equivalent to SPKM-1, except as described in the remainder of this section.

2.1. Mechanism Type

SPKM-3 has a different mechanism type OID from SPKM-1.

```
spkm-3 OBJECT IDENTIFIER ::=
   {iso(1)identified-organization(3)dod(6)internet(1)security(5)
   mechanisms(5)spkm(1)spkm-3(3)}
```

2.2. Name Type

RFC 2025 defines no required name types of SPKM. LIPKEY requires that the SPKM-3 implementation support all the mechanism independent name types in RFC 2078.

2.3. Algorithms

2.3.1. MANDATORY Algorithms

   RFC 2025 defines various algorithms for integrity, confidentiality,
   key establishment, and subkey derivation. Except for
   md5WithRSAEncryption, the REQUIRED Key Establishment (K-ALG),
   Integrity (I-ALG) and One-Way Functions for Subkey Derivation (O-ALG)
   algorithms listed in RFC 2025 continue to be REQUIRED.

   SPKM is designed to be extensible with regard to new algorithms. In
   order for LIPKEY to work correctly and securely, the following
   algorithms MUST be implemented in SPKM-3:

   *   Integrity algorithms (I-ALG)

      NULL-MAC
           Because the initiator may not have a certificate for itself,
           nor for the target, it is not possible for it to calculate an
           Integrity value in the initiator's REQ-TOKEN that is sent to
           the target. So we define, in ASN.1 [CCITT] syntax, a null I-
           ALG that returns a zero length bit string regardless of the
           input passed to it:

      NULL-MAC OBJECT IDENTIFIER ::=
         {iso(1)identified-organization(3)dod(6)internet(1)security(5)
         integrity(3)NULL-MAC(3)}

      id-dsa-with-sha1
           This is the signature algorithm as defined in Section 7.2.2
           of [RFC2459]. As noted in RFC 2459, the ASN.1 OID used to
           identify this signature algorithm is:

              id-dsa-with-sha1 OBJECT IDENTIFIER ::= {
                      iso(1) member-body(2) us(840) x9-57(10040)
                            x9cm(4) 3
              }

           Note that there is a work-in-progress [PKIX] to obsolete RFC
           2459. However that work-in-progress does not change the
           definition of id-dsa-with-sha1.

      HMAC-MD5
           A consequence of the SPKM-3 initiator not having a
           certificate is that it cannot use a digital signature
           algorithm like md5WithRSAEncryption, id-dsa-with-sha1, or
           sha1WithRSAEncryption once the context is established.
           Instead, a message authentication code (MAC) algorithm is

required. DES-MAC is specified as recommended in [RFC2025].
Since the security of 56 bit DES has been shown to be
inadequate [EFF], SPKM-3 needs a stronger MAC. Thus, SPKM-3
MUST support the HMAC-MD5 algorithm [RFC2104], with this OID:

```
HMAC-MD5 OBJECT IDENTIFIER ::= {
        iso(1) org(3) dod(6) internet(1) security(5)
               mechanisms(5) ipsec(8) isakmpOakley(1)
               1
}
```

The reference for the algorithm OID of HMAC-MD5 is [IANA].
The reference for the HMAC-MD5 algorithm is [RFC2104].

The HMAC-SHA1 algorithm is not a mandatory SPKM-3 I-ALG MAC
because SHA-1 is about half the speed of MD5 [Young]. A MAC
based on an encryption algorithm like cast5CBC, DES EDE3, or
RC4 is not mandatory because MD5 is 31 percent faster than
the fastest of the three encryption algorithms [Young].

   *   Confidentiality algorithm (C-ALG).

RFC 2025 does not have a MANDATORY confidentiality algorithm,
and instead has RECOMMENDED a 56 bit DES algorithm. Since the
LIPKEY initiator needs to send a password to the target, and
since 56 bit DES has been demonstrated as inadequate [EFF],
LIPKEY needs stronger encryption. Thus, SPKM-3 MUST support this
algorithm:

```
cast5CBC OBJECT IDENTIFIER ::= {
        iso(1) memberBody(2) usa(840) nt(113533) nsn(7)
               algorithms(66) 10
}

Parameters ::= SEQUENCE {
        iv OCTET STRING DEFAULT 0, -- Initialization vector
        keyLength INTEGER          -- Key length, in bits
}
```

The reference for the OID and description of the cast5CBC
algorithm is [RFC2144]. The keyLength in the Parameters MUST be
set to 128 bits.

A triple DES (DES EDE3) algorithm is not a mandatory SPKM-3 C-
ALG because it is much slower than cast5CBC. One set of
measurements [Young] on a Pentium Pro 200 megahertz processor
using the SSLeay code, showed that DES EDE3 performed as high as
1,646,210 bytes per second, using 1024 byte blocks. The same

test bed yielded performance of 7,147,760 bytes per second for
cast5CBC, and 22,419,840 bytes per second for RC4. Most TLS
sessions negotiate the RC4 cipher. Given that LIPKEY is targeted
at environments similar to that where TLS is deployed, selecting
a cipher that is over 13 times slower (and over 13 times more
CPU intensive) than RC4 would severely impede the usefulness of
LIPKEY. For performance reasons, RC4 would be the preferred
mandatory algorithm for SPKM-3. Due to intellectual property
considerations with RC4 [Schneier], the combination of
cast5CBC's reasonable performance, and its royalty-free
licensing terms [RFC2144] make cast5CBC the optimal choice among
DES EDE3, RC4, and cast5CBC.

   *   Key Establishment Algorithm (K-ALG)

        RFC 2025 lists dhKeyAgreement [PKCS-3] as an apparently optional
        algorithm. As will be described later, the RSAEncryption key
        establishment algorithm is of no use for a low infrastructure
        security mechanism as defined by this memorandum. Hence, in
        SPKM-3, dhKeyAgreement is a REQUIRED key establishment
        algorithm:

            dhKeyAgreement OBJECT IDENTIFIER ::= {
                    iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1)
                    pkcs-3(3) 1
            }

   *   One-Way Function for Subkey Derivation Algorithm (O-ALG)

        RFC 2025 lists MD5 as a mandatory algorithm. Since MD5 has been
        found to have weaknesses when used as a hash [Dobbertin], id-
        sha1 is a MANDATORY O-ALG in SPKM-3:

            id-sha1 OBJECT IDENTIFIER ::= {
                    iso(1) identified-organization(3) oiw(14)
                    secsig(3) algorithms(2) 26
            }

        The reference for the algorithm OID of id-sha1 is [RFC2437].
        The reference for SHA-1 algorithm corresponding to id-sha1 is
        [FIPS].

2.3.2. RECOMMENDED Integrity Algorithms (I-ALG)

   md5WithRSAEncryption
        The md5WithRSAEncryption integrity algorithm is listed in
        [RFC2025] as mandatory. Due to intellectual property
        considerations [RSA-IP], SPKM-3 implementations cannot be

required to implement it. However, given the proliferation of
certificates using RSA public keys, md5WithRSAEncryption is
strongly RECOMMENDED. Otherwise, the opportunities for LIPKEY to
leverage existing public key infrastructure will be limited.

sha1WithRSAEncryption
     For reasons similar to that for md5WithRSAEncryption,
     sha1WithRSAEncryption is a RECOMMENDED algorithm. The
     sha1WithRSAEncryption algorithm is listed in addition to
     md5WithRSAEncryption due to weaknesses in the MD5 hash algorithm
     [Dobbertin]. The OID for sha1WithRSAEncryption is:

          sha1WithRSAEncryption OBJECT IDENTIFIER ::= {
                  iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1)
                  pkcs-1(1) 5
          }

     The reference for the algorithm OID and description of
     sha1WithRSAEncryption is [RFC2437].

2.4. Context Establishment Tokens

   RFC 2025 sets up a context with an initiator first token (REQ-TOKEN),
   a target reply (REP-TI-TOKEN), and finally an initiator second token
   (REP-IT-TOKEN) to reply to the target's reply. Since LIPKEY uses
   SPKM-3 with unilateral authentication, the REP-IT-TOKEN is not used.
   LIPKEY has certain requirements on the contents of the REQ-TOKEN and
   REP-TI-TOKEN, but the syntax of the SPKM-3 tokens is not different
   from RFC 2025's SPKM-1 tokens.

2.4.1. REQ-TOKEN Content Requirements

2.4.1.1. algId and req-integrity

   If the SPKM-3 initiator cannot calculate a req-integrity field due to
   the lack of a target certificate, it MUST use the NULL-MAC I-ALG
   described earlier in this memorandum. This will produce a zero length
   bit string in the Integrity field.

2.4.1.2. Req-contents

   Because RFC 2025 requires that the RSAEncryption K-ALG be present,
   SPKM-1 must be able to map the target (targ-name) to its public key
   certificate, and thus SPKM can use the RSAEncryption algorithm to
   fill in the key-estb-req field. Because LIPKEY assumes a low
   infrastructure deployment, SPKM-3 MUST be prepared to be unable to
   map the targ-name field of the Req-contents field. This is a
   contradiction which is resolved by requiring SPKM-3 to support the

dhKeyAgreement algorithm. Note that if an SPKM-3 implementation tries
to map the target to a certificate, and succeeds, it is free to use
the RSAEncryption K-ALG algorithm. It is also free to use an algID
other than NULL-MAC in the REQ-TOKEN type.

2.4.1.2.1. Options

SPKM-3 implementations MUST set the target-certif-data-required bit
to 1 if the only K-ALG in the key-estb-set field of Req-contents is
dhKeyAgreement. This would normally occur if the SPKM-3
implementation cannot resolve the target name to a certificate.

2.4.1.2.2. Conf-Algs

If the SPKM-3 implementation supports an algorithm weaker than
cast5CBC, cast5CBC MUST be listed before the weaker algorithm to
encourage the target to negotiate the stronger algorithm.

2.4.1.2.3. Intg-Algs

Because the initiator will be anonymous (at the SPKM-3 level) and
will not have a certificate for itself, the initiator cannot use an
integrity algorithm that supports non-repudiation; it must use a MAC
algorithm. If the SPKM-3 implementation supports an algorithm weaker
than HMAC-MD5, HMAC-MD5 MUST be listed before the weaker algorithm to
encourage the target to negotiate the stronger algorithm.

2.4.2. REP-TI-TOKEN Content Requirements

With the previously described requirements on REQ-TOKEN, the contents
of SPKM-3's REP-TI-TOKEN can for the most part be derived from the
specification in RFC 2025. The exceptions are the algId and rep-ti-
integ fields.

2.4.2.1. algId

The SPKM-3 target MUST NOT use a NULL-MAC I-ALG; it MUST use a
signature algorithm like id-dsa-with-sha1, md5WithRSAEncryption, or
sha1WithRSAEncryption.

2.4.2.2. rep-ti-integ

If the req-token has an algId of NULL-MAC, then the target MUST
compute the rep-ti-integ on the concatenation of the req-contents and
rep-ti-contents.

2.5. Quality of Protection (QOP)

   The SPKM-3 initiator and target negotiate the set of algorithms they
   mutually support, using the procedure defined in Section 5.2 of RFC
   2025. If a QOP of zero is specified, then the initiator and target
   will use the first C-ALG (privacy), and I-ALG (integrity) algorithms
   negotiated.

   SPKM breaks the QOP into several fields, as reproduced here from
   Section 5.2 of RFC 2025:

```
    Confidentiality                       Integrity
    31 (MSB)                              16 15                 (LSB) 0
    -------------------------------|-------------------------------
    | TS(5) | U(3) | IA(4) | MA(4) | TS(5) | U(3) | IA(4) | MA(4) |
    -------------------------------|-------------------------------
```

   The MA subfields enumerate mechanism-defined algorithms. Since this
   memorandum introduces a new mechanism, SPKM-3, within the SPKM
   family, it is appropriate to add algorithms to the MA subfields of
   the respective Confidentiality and Integrity fields.

   The complete set of Confidentiality MA algorithms is thus:

```
    0001 (1) = DES-CBC
    0010 (2) = cast5CBC
```

   Where "0001" and "0010" are in base 2. An SPKM peer that negotiates
   a confidentiality MA algorithm value of "0010" MUST use a 128 bit
   key, i.e. set the keyLength values in the cast5CBC Parameters to 128
   bits.

   The complete set of Integrity MA algorithms is thus:

```
    0001 (1) = md5WithRSAEncryption
    0010 (2) = DES-MAC
    0011 (3) = id-dsa-with-sha1
    0100 (4) = HMAC-MD5
    0101 (5) = sha1WithRSAEncryption
```

   Where "0001" through "0101" are in base 2.

   Adding support for cast5CBC, id-dsa-with-sha1, HMAC-MD5, and
   sha1WithRSAEncryption in the above manner to SPKM-1 and SPKM-2 does
   not impair SPKM-1 and SPKM-2 backward compatibility because, as noted
   previously, SPKM negotiates algorithms. An older SPKM-1 or SPKM-2
   that does not recognize MA values for cast5CBC, id-dsa-with-sha1,
   HMAC-MD5, or sha1WithRSAEncryption will not select them.

3. How LIPKEY Uses SPKM

3.1. Tokens

   LIPKEY will invoke SPKM-3 to produce SPKM tokens. Since the mechanism
   that the application uses is LIPKEY, LIPKEY will wrap some of the
   SPKM-3 tokens with LIPKEY prefixes. The exact definition of the
   tokens is described later in this memorandum.

3.2. Initiator

3.2.1. GSS_Import_name

   The initiator uses GSS_Import_name to import the target's name,
   typically, but not necessarily, using the GSS_C_NT_HOSTBASED_SERVICE
   name type. Ultimately, the output of GSS_Import_name will apply to
   an SPKM-3 mechanism type because a LIPKEY target is an SPKM-3 target.

3.2.2. GSS_Acquire_cred

   The initiator calls GSS_Acquire_cred. The credentials that are
   acquired are LIPKEY credentials, a user name and password. How the
   user name and password is acquired is dependent upon the operating
   environment. A application that invokes GSS_Acquire_cred() while the
   application's user has a graphical user interface running might
   trigger the appearance of a pop up window that prompts for the
   information. A application embedded into the operating system, such
   as an NFS [Sandberg] client implemented as a native file system might
   broadcast a message to the user's terminals telling him to invoke a
   command that prompts for the information.

   Because the credentials will not be used until GSS_Init_sec_context
   is called, the LIPKEY implementation will need to safeguard the
   credentials. If this is a problem, the implementation may instead
   defer actual acquisition of the user name and password until
   GSS_init_sec_context is ready to send the user name and password to
   the target. In that event, the output_cred_handle argument of
   GSS_Acquire_cred would simply be a reference that mapped to the
   principal corresponding to the desired_name argument. A subsequent
   GSS_Init_sec_context call would consider the mapping of
   claimant_cred_handle to principal when it acquires the user name and
   password. For example, the aforementioned pop up window might fill in
   the user name portion of the dialog with a default value that maps to
   the principal referred to in claimant_cred_handle.

3.2.3. GSS_Init_sec_context

   When a program invokes GSS_Init_sec_context on the LIPKEY mechanism
   type, if the context handle is NULL, the LIPKEY mechanism will in
   turn invoke GSS_Init_sec_context on an SPKM-3 mechanism implemented
   according to the requirements described previously. This call to
   SPKM-3 MUST have the following attributes:

   *   claimant_cred_handle is NULL

   *   mutual_req_flag is FALSE

   *   anon_req_flag is TRUE

   *   input_token is NULL

   *   mech_type is the OID of the SPKM-3 mechanism

   Keep in mind the above attributes are in the GSS_Init_sec_context
   call from the LIPKEY mechanism down to the SPKM-3 mechanism. There
   are no special restrictions placed on the application invoking
   LIPKEY's GSS_Init_sec_context routine. All other arguments are
   derived from the LIPKEY GSS_Init_sec_context arguments.

   The call to the SPKM-3 GSS_Init_sec_context will create an SPKM-3
   context handle. The remainder of the description of the LIPKEY
   GSS_Init_sec_context call depends on whether the caller of the LIPKEY
   GSS_Init_sec_context sets anon_req_flag to TRUE or FALSE.

3.2.3.1. LIPKEY Caller Specified anon_req_flag as TRUE

   If the caller of LIPKEY's GSS_Init_sec_context sets anon_req_flag to
   TRUE, it MUST return to the LIPKEY caller all the outputs from the
   SPKM-3 GSS_Init_sec_context call, including the
   output_context_handle, output_token, and mech_type. In this way,
   LIPKEY now "gets out of the way" of GSS-API processing between the
   application and SPKM-3, because nothing in the returned outputs
   relates to LIPKEY. This is necessary, because LIPKEY context tokens
   do not have provision for specifying anonymous initiators. This is
   because SPKM-3 is sufficient for purpose of supporting anonymous
   initiators in a low infrastructure environment.

   Clearly, when the LIPKEY caller desires anonymous authentication,
   LIPKEY does not add any value, but it is simpler to support the
   feature, than to insist the caller directly use SPKM-3.

If all goes well, the caller of LIPKEY will be returned a
major_status of GSS_S_CONTINUE_NEEDED via SPKM-3, and so the caller
of LIPKEY will send the output_token to the target. The caller of
LIPKEY then receives the response token from the target, and directly
invokes the SPKM-3 GSS_Init_sec_context. Upon return, the
major_status should be GSS_S_COMPLETE.

3.2.3.2. LIPKEY Caller Specified anon_req_flag as FALSE

The LIPKEY mechanism will need to allocate a context handle for
itself, and record in the LIPKEY context handle the SPKM-3 context
handle that was returned in the output_context_handle parameter from
the call to the SPKM-3 GSS_Init_sec_context routine. The LIPKEY
GSS_Init_sec_context routine will return in output_context_handle the
LIPKEY context handle, and in mech_type, the LIPKEY mechanism type.
The output_token is as defined later in this memorandum, in the
subsection entitled "Context Tokens Prior to SPKM-3 Context
Establishment." All the other returned outputs will be those that
the SPKM-3 GSS_Init_sec_context routine returned to LIPKEY. If all
went well, the SPKM-3 mechanism will have returned a major_status of
GSS_S_CONTINUE_NEEDED.

The caller of the LIPKEY GSS_Init_sec_context routine will see a
major_status of GSS_S_CONTINUE_NEEDED, and so the caller of LIPKEY
will send the output_token to the target. The caller of LIPKEY then
receives the target's response token, and invokes the LIPKEY
GSS_Init_sec_context routine for a second time. LIPKEY then invokes
the SPKM-3 GSS_Init_sec_context for a second time and upon return,
the major_status should be GSS_S_COMPLETE.

While SPKM-3's context establishment is now complete, LIPKEY's
context establishment is not yet complete, because the initiator must
send to the target the user name and password that were passed to it
via the claimant_cred_handle on the first call to the LIPKEY
GSS_Init_sec_context routine. LIPKEY uses the established SPKM-3
context handle as the input to GSS_Wrap (with conf_req_flag set to
TRUE) to encrypt what the claimant_cred_handle refers to (user name
and password), and returns that as the output_token to the caller of
LIPKEY (provided the conf_state output from the call to SPKM-3's
GSS_Wrap is TRUE), along with a major_status of
GSS_S_CONTINUE_NEEDED.

The caller of LIPKEY sends its second context establishment token to
the target, and waits for a token provided by the target's
GSS_Accept_sec_context routine. The target's LIPKEY
GSS_Accept_sec_context routine invokes the SPKM-3 GSS_Unwrap routine
on the token, and validates the user name and password. The target
then invokes SPKM-3's GSS_Wrap routine on a boolean indicating

whether or not the user name and password were accepted, and returns
the output_message result from GSS_Wrap as the output_token result
for GSS_Accept_sec_context.

The caller of LIPKEY receives the target's response token, and passes
this via the input_token parameter to the LIPKEY GSS_Init_sec_context
routine. LIPKEY then invokes GSS_Unwrap to get the boolean
acceptance indication, and maps this to a major_status of either
GSS_S_COMPLETE indicating successful (the boolean was TRUE) and
completed LIPKEY context establishment, or GSS_S_FAILURE, indicating
that context establishment failed. GSS_S_CONTINUE_NEEDED will not be
returned.

Note that the mutual_req_flag parameter is ignored because unilateral
authentication is impossible. The initiator must authenticate the
target via SPKM-3 in order to create a secure channel to transmit the
user name and password. The target must authenticate the initiator
when it receives the user name and password.

The SPKM-3 context remains established while the LIPKEY context is
established. If the SPKM-3 context expires before the LIPKEY context
is destroyed, the LIPKEY implementation should expire the LIPKEY
context and return the appropriate error on the next GSS-API
operation.

3.2.4. Other operations

   For other operations, the LIPKEY context acts as a pass through to
   the SPKM-3 context. Operations that affect or inquire context state,
   such as GSS_Delete_sec_context, GSS_Export_sec_context,
   GSS_Import_sec_context, and GSS_Inquire_context will require a pass
   through to the SPKM-3 context and a state modification of the LIPKEY
   context.

3.3. Target

3.3.1. GSS_Import_name

   As with the initiator, the imported name will be that of the target.

3.3.2. GSS_Acquire_cred

   The target calls the LIPKEY GSS_Acquire_cred routine to get a
   credential for an SPKM-3 target, via the SPKM-3 GSS_Acquire_cred
   routine. The desired_name is the output_name from GSS_Import_name.

3.3.3. GSS_Accept_sec_context

   When a program invokes GSS_Accept_sec_context on the LIPKEY mechanism
   type, if the context handle is NULL, the LIPKEY mechanism will in
   turn invoke GSS_Accept_sec_context on an SPKM-3 mechanism implemented
   according the requirements described previously. This call to SPKM-3
   is no different than what one would expect for a layered call to
   GSS_Accept_sec_context.

   If all goes well, the SPKM-3 GSS_Accept_sec_context call succeeds
   with GSS_S_COMPLETE, and the LIPKEY GSS_Accept_sec_context call
   returns the output_token to the caller, but with a major_status of
   GSS_S_CONTINUE_NEEDED because the LIPKEY initiator is still expected
   to send the user name and password.

   Once the SPKM-3 context is in a GSS_S_COMPLETE state, the next token
   the target receives will contain the user name and password, wrapped
   by the output of an SPKM-3 GSS_Wrap call. The target invokes the
   LIPKEY GSS_Accept_sec_context, which in turn invokes the SPKM-3
   GSS_Unwrap routine. The LIPKEY GSS_Accept_sec_context routine then
   compares the user name and password with its user name name and
   password database. If the initiator's user name and password are
   valid, GSS_S_COMPLETE is returned to the caller. Otherwise
   GSS_S_FAILURE is returned. In either case, an output_token - equal to
   the output_message result from an SPKM-3 GSS_Wrap call on a boolean
   value - is returned to the caller. The boolean value is set to TRUE
   if the the user name and password were valid, FALSE otherwise. The
   target expects no more context establishment tokens from caller.

4. LIPKEY Description

4.1. Mechanism Type

   lipkey OBJECT IDENTIFIER ::=
      {iso(1)identified-organization(3)dod(6)internet(1)security(5)
      mechanisms(5)lipkey(9)}

4.2. Name Types

   LIPKEY uses only the mechanism independent name types defined in RFC
   2078. All the name types defined in RFC 2078 are REQUIRED.

4.3. Token Formats

4.3.1. Context Tokens

   GSS-API defines the context tokens as:

```
   InitialContextToken ::=
   -- option indication (delegation, etc.) indicated within
   -- mechanism-specific token
   [APPLICATION 0] IMPLICIT SEQUENCE {
           thisMech MechType,
           innerContextToken ANY DEFINED BY thisMech
               -- contents mechanism-specific
               -- ASN.1 structure not required
   }

   SubsequentContextToken ::= innerContextToken ANY
   -- interpretation based on predecessor InitialContextToken
   -- ASN.1 structure not required
```

   The contents of the innerContextToken depend on whether the SPKM-3
   context is established or not.

4.3.1.1. Context Tokens Prior to SPKM-3 Context Establishment

   In a LIPKEY InitialContextToken, thisMech will be the Object
   identifier for LIPKEY. However, as long as LIPKEY has not
   established the SPKM-3 mechanism, the innerContextToken for both the
   InitialContextToken and the SubsequentContextToken will be the output
   of an SPKM-3 GSS_Init_sec_context or GSS_Accept_sec_context. So the
   LIPKEY innerContextToken would be either:

   *   An InitialContextToken, with thisMech set to the object
        identifier for SPKM-3, with innerContextToken defined to be an
        SPKMInnerContextToken, as defined in RFC 2025.

   *   A SubsequentContextToken, with innerContextToken defined to be
        SPKMInnerContextToken

4.3.1.2. Post-SPKM-3 Context Establishment Tokens

   Once the SPKM-3 context is established, there is just one token sent
   from the initiator to the target, and one token returned to
   initiator.

4.3.1.2.1. From LIPKEY Initiator

   The LIPKEY initiator generates a token that is the the result of a
   GSS_Wrap (conf_req is set to TRUE) of a user name and password by the
   SPKM-3 context. The input_message argument of GSS_Wrap refers to an
   instance of the UserName-Password type defined below:

```
    UserName-Password ::= SEQUENCE {
            user-name        OCTET STRING,
                                    -- each octet is an octet of a
                                    -- UTF-8 [RFC2279] string
            password         OCTET STRING
                                    -- each octet is an octet of a
                                    -- UTF-8 [RFC2279] string
    }
```

4.3.1.2.2. From LIPKEY Target

   The target validates the user name and password token from the
   initiator, and generates a response token that is the output_message
   result of an SPKM-3 GSS_Wrap (conf_req may or may not be set to TRUE)
   call on an indication of validation success. The input_message
   argument of GSS_Wrap refers to an instance of the Valid-UNP type
   defined below:

```
    Valid-UNP ::= BOOLEAN
                    -- If TRUE, user name/password pair was valid.
```

4.3.2. Tokens from GSS_GetMIC and GSS_Wrap

   RFC 2078 defines the token emitted by GSS_GetMIC and GSS_Wrap as:
```
            PerMsgToken ::=
            -- as emitted by GSS_GetMIC and processed by GSS_VerifyMIC
            -- ASN.1 structure not required
                    innerMsgToken ANY

            SealedMessage ::=
            -- as emitted by GSS_Wrap and processed by GSS_Unwrap
            -- includes internal, mechanism-defined indicator
            -- of whether or not encrypted
            -- ASN.1 structure not required
                    sealedUserData ANY
```

   As one can see, there are no mechanism independent prefixes in
   PerMSGToken or SealedMessage, and no explicit mechanism specific
   information. Since LIPKEY does not add any value to GSS_GetMIC and

GSS_Wrap other than passing the message to the SPKM-3 GSS_GetMIC and
GSS_Wrap, LIPKEY's PerMsgToken and SealedMessage tokens are exactly
what SPKM-3's GSS_GetMIC and GSS_Wrap routines produce.

4.4. Quality of Protection

LIPKEY, being a pass through for GSS_Wrap and GSS_GetMIC to SPKM-3,
does not interpret or alter the QOPs passed to the aforementioned
routines or received from their complements, GSS_Unwrap, and
GSS_VerifyMIC. Thus, LIPKEY supports the same set of QOPs as SPKM-3.

5. Security Considerations

5.1. Password Management

LIPKEY sends the clear text password encrypted by 128 bit cast5CBC so
the risk in this approach is in how the target manages the password
after it is done with it. The approach should be safe, provided the
target clears the memory (primary and secondary, such as disk)
buffers that contained the password, and any hash of the password
immediately after it has validated the user's password.

5.2. Certification Authorities

The initiator must have a list of trusted Certification Authorities
in order to verify the checksum (rep-ti-integ) on the SPKM-3 target's
context reply token. If it encounters a certificate signed by an
unknown and/or untrusted certificate authority, the initiator MUST
NOT silently accept the certificate. If it does wish to accept the
certificate, it MUST get confirmation from the user running the
application that is using GSS-API.

5.3. HMAC-MD5 and MD5 Weaknesses

While the MD5 hash algorithm has been found to have weaknesses
[Dobbertin], the weaknesses do not impact the security of HMAC-MD5
[Dobbertin].

5.4. Security of cast5CBC

The cast5CBC encryption algorithm is relatively new compared to
established algorithms like triple DES, and RC4. Nonetheless, the
choice of cast5CBC as the MANDATORY C-ALG for SPKM-3 is advisable.
The cast5CBC algorithm is a 128 bit algorithm that the 256 bit
cast6CBC [RFC2612] algorithm is based upon. The cast6CBC algorithm
was judged by the U.S. National Institute of Standards and Technology
(NIST) to have no known major or minor "security gaps," and to have a
"high security margin" [AES]. NIST did note some vulnerabilities

related to smart card implementations, but many other algorithms NIST
analyzed shared the vulnerabilities, and in any case, LIPKEY is by
definition not aimed at smart cards.

References

    [AES]   Nechvatal, J., Barker, E., Dodson, D., Dworkin, M., Foti,
                 J., Roback, E. (Undated, but no later than 1999). "Status
                 Report on the First Round of the Development of the
                 Advanced Encryption Standard."
                 http://csrc.nist.gov/encryption/aes/round1/r1report.htm

    [CCITT]   CCITT (1988). "Recommendation X.208: Specification of
                 Abstract Syntax Notation One (ASN.1)."

    [Dobbertin] Dobbertin, H. (1996). "The Status of Md5 After a Recent
                 Attack," RSA Laboratories' CryptoBytes, Volume 2, Number
                 2.
                 ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf

    [EFF]   Electronic Frontier Foundation, John Gilmore (Editor)
                 (1998). "Cracking Des: Secrets of Encryption Research,
                 Wiretap Politics & Chip Design," O'Reilly & Associates,
                 ISBN 1565925203.

    [FIPS]   National Institute of Standards and Technology (1995).
                 "Secure Hash Standard" (SHA-1).
                 http://www.itl.nist.gov/fipspubs/fip180-1.htm

    [IANA]   Internet Assigned Numbers Authority (1999). "Network
                 Management Parameters." http://www.isi.edu/in-
                 notes/iana/assignments/smi-numbers

    [PKCS-3]   RSA Laboratories (1993). "PKCS #3: Diffie-Hellman Key-
                 Agreement Standard, Version 1.4."
                 ftp://ftp.rsa.com/pub/pkcs/ascii/pkcs-3.asc

    [PKIX]   Housley, R., Ford, W., Polk, W., Solo, D., "Internet
                 X.509 Public Key Infrastructure Certificate and CRL
                 Profile", Work in Progress.

    [RFC1831]   Srinivasan, R., "RPC: Remote Procedure Call Protocol
                 Specification Version 2", RFC 1831, August 1995.

    [RFC1832]   Srinivasan, R., "XDR: External Data Representation
                 Standard", RFC 1832, August 1995.

   [RFC1964]   Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC
                 1964, June 1996.

   [RFC2203]   Eisler, M., Chiu, A. and L. Ling, "RPCSEC_GSS Protocol
                 Specification", RFC 2203, September 1997.

   [RFC2025]   Adams, C., "The Simple Public-Key GSS-API Mechanism
                 (SPKM)", RFC 2025, October 1996.

   [RFC2078]   Linn, J., "Generic Security Service Application Program
                 Interface, Version 2", RFC 2078, January 1997.

   [RFC2104]   Krawczyk, H, Bellare, M. and R. Canetti, "HMAC: Keyed-
                 Hashing for Message Authentication", RFC 2104, February
                 1997.

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
                 Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC2144]   Adams, C., "The CAST-128 Encryption Algorithm", RFC 2144,
                 May 1997.

   [RFC2246]   Dierks, T. and C. Allen, "The TLS Protocol Version 1.0",
                 RFC 2246, January 1999.

   [RFC2279]   Yergeau, F., "UTF-8, a transformation format of ISO
                 10646", RFC 2279, January 1998.

   [RFC2437]   Kaliski, B. and J. Staddon, "PKCS #1: RSA Cryptography
                 Specifications Version 2.0", RFC 2437, October 1998.

   [RFC2459]   Housley, R., Ford, W., Polk, W. and D. Solo, "Internet
                 X.509 Public Key Infrastructure Certificate and CRL
                 Profile", RFC 2459, January 1999.

   [RFC2612]   Adams, C. and J. Gilchrist, "The CAST-256 Encryption
                 Algorithm", RFC 2612, June 1999.

   [RSA-IP]   All statements received by the IETF Secretariat are places
                 on-line in http://www.ietf.org/ipr.html. Please check
                 this web page to see any IPR information received about
                 this and other technology.

   [Sandberg]   Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., Lyon,
                 B. (1985). "Design and Implementation of the Sun Network
                 Filesystem," Proceedings of the 1985 Summer USENIX
                 Technical Conference.

    [Schneier] Schneier, B. (1996). "Applied Cryptography," John Wiley &
                Sons, Inc., ISBN 0-471-11709-9.

    [Young]   Young, E.A. (1997). Collected timing results from the
                SSLeay source code distribution.

Author's Address

    Address comments related to this memorandum to:

    ietf-cat-wg@lists.Stanford.EDU

    Mike Eisler
    Zambeel
    5565 Wilson Road
    Colorado Springs, CO 80919

    Phone: 1-719-599-9026
    EMail: mike@eisler.com

Full Copyright Statement

Acknowledgement