Open Software Foundation
Request For Comments: 94.1
November 1996

Mark Heroux (DEC)

A PRIVATE KEY STORAGE SERVER FOR DCE — FUNCTIONAL SPECIFICATION

1. INTRODUCTION

This document specifies:

(a) A Private Key Store Server (PKSS) for DCE, as referred to in [RFC 68.2].

(b) A secure protocol by which a client obtains an asymmetric key pair from the PKSS.

(c) A secure protocol by which a client can change a password and/or asymmetric key pair.

(d) Secure management and auditing capabilities.

(e) The data structures and interfaces required to implement the above.

1.1. Changes Since Last Publication

This document supercedes [RFC 94.0]. After [RFC 94.0] was published, we learned that one of the secure network communication protocols described therein was protected by a patent. When our attempts to reach a license agreement with the patent-holder failed, we devised and implemented an alternative protocol. This document describes PKSS functionality using that alternative protocol.

2. TARGET

This service is intended for customers who want to use public key encryption for secure logins per [RFC 68.2], but who may not want to invest in the equipment and infrastructure required to deploy smart-card-based public key security enterprise-wide, or who may want to phase in hardware-based public key mechanisms slowly over time.

3. GOALS AND NON-GOALS

3.1. Goals

(a) Provide a secure server to store, retrieve, and manage asymmetric key pairs on behalf of users.

(b) Integrate seamlessly with the public-key-based login procedure specified in [RFC 68.2].

3.2. Non-Goals

(a) Provide a secure means to replicate the PKSS.

(b) Provide a secure means for users to use the private key component of their asymmetric key pairs for any operation other than login as specified in [RFC 68.2].

(c) Provide a means to enforce use of strong passwords through integration with a password strength server.

4. TERMINOLOGY

4.1. Definitions

(a) **Password** — A secret string of characters that one uses to identify oneself securely. A ‘‘strong’’ password may not be compromised by a dictionary attack (see below).

(b) **Key** — A bit string that an encryption algorithm uses for encryption or decryption. A key may be based on a password.

(c) **Secret key encryption** — A method of encryption that uses one key for both encryption and decryption. Also called ‘‘symmetric key encryption’’. The Data Encryption Standard (DES) is an example of a secret key encryption algorithm.

(d) **Public key encryption** — A method of encryption that uses a pair of keys, one for encryption and another for decryption. An important property of public key encryption is that it is infeasible to recover one key given only the other. Also called ‘‘asymmetric key encryption’’. The Rivest- Shamir-Adelman (RSA) algorithm is an example of a public key encryption algorithm.

(e) **Diffie-Hellman key** — A secret key which two communicating principals may derive and share without having previously agreed on a shared secret.

(f) **Hash function** — A mathematical function that takes as input data of arbitrary length and computes a fixed length output. A one-way hash is one whose input is infeasible to recover given only the output. A one-way hash may also be called a ‘‘checksum’’ or ‘‘message digest’’.

(g) **Dictionary attack** — An attack on passwords limited to using only such words or phrases as might be found in a dictionary.

(h) **Exhaustive search attack** — An attack on passwords or keys that systematically tries every possible combination. An exhaustive search on an eight-byte US-ASCII password space would probe every combination between 0x2020202020202020 and 0x7E7E7E7E7E7E7E7E. An exhaustive search on an eight-byte key space would probe every combination between 0x0000000000000000 and 0xFFFFFFFFFFFFFFFF.

(i) **Forward search attack** — An attack on small amounts of encrypted or hashed data (such as passwords) in which an attacker, knowing the algorithm with which the data was transformed, uses a dictionary or exhaustive search attack to produce trial encrypted or hashed output that matches the data under attack. On UNIX systems, `/etc/password` files that contain hashed passwords are susceptible to this attack.

(j) **Known-plaintext attack** — An attack on encrypted data in which an attacker searches for certain material known to be in the data stream. ASN.1-encoded data is subject to known-plaintext attacks (one searches for the ASN.1 identifiers).

(k) **Spoof attack** — An attack on system or network resources in which one party attempts to masquerade as another.

(l) **Replay attack** — An attack in which a message originally transmitted by a valid user is retransmitted by another at a later time for the purpose of gaining access to resources to which the party mounting the attack is not legitimately entitled. This is a form of spoof attack.

(m) **Bucket Brigade attack** — An attack in which an attacker intercepts and retransmits messages between communicating principals but makes it appear as if they are communicating directly with one another. Also called a ‘‘Man-in-the-middle attack’’. An unprotected Diffie- Hellman key negotiation is susceptible to this form of atttack.

(n) **Denial of Service attack** — An attack in which the attacker prevents valid users from gaining access to resources to which they are legitimately entitled.

(o) **Message Stream Modification attack** — An attack in which the contents of a message are changed for the purpose of spoofing or denying service.

4.2. Symbols

In these symbol definitions, case is irrelevant, with the exception that lower-case letters may imply subscripting.

(a) `N` — A nonce. `Np` is a nonce generated by the PKSS. `Nc` is a nonce generated by a PKSS client.

(b) `D` — A symmetric key used for secure communication between the PKSS and the client, jointly generated using the Diffie-Hellman algorithm. The client and PKSS each generate nonces `Xc` and `Xp`, and calculate `E(Xc)` and `E(Xp)` respectively, where `E(x)` is defined as `(P**x) mod g`; `P` is a prime number; and `g` is a number less than `P`. Each party sends its calculated value to the other. Thus the PKSS receives `E(Xc)`, and the client receives `E(Xp)`. The client calculates `(E(Xp)**Xc) mod g`, while the PKSS calculates `(E(Xc)**Xp) mod g`. These two values are the same, and are used as the communications key `D`. An observer of the network traffic sees only `E(Xc)` and `E(Xp)`, from which it is infeasible to derive `D`.

(c) `Kx` — The private key component of an asymmetric key pair.

(d) `Ko` — The public key component of an asymmetric key pair.

(e) `U` — A person (''User'') wanting to use or currently using network resources.

(f) `C` — A PKSS client application under the control of a User.

(g) `P` — Private Key Store Server (PKSS).

(h) `Kop` — The public component key of the Private Key Store Server's asymmetric key pair; similarly `Kxp`, `Kou`, `Kxu`.

(i) `[Z]Y` — `Z` encrypted using key `Y`.

(j) `{Z}Y` — `Z` signed using key `Y`.

(k) `V` — A version number of an asymmetric key pair.

(l) `X-->Y: Z` — Principal `X` sends message `Z` to principal `Y`.

(m) `H1` and `H2` — A pair of hashes that are the results of a password-hashing function `H`, where `H(password) = {H1,H2}`. This hash function must have the properties that: (1) obtaining the password given either or both hashes is intractable; and (2) obtaining one hash given the other is intractable. See [RFC 1507, 4.2.2] for the algorithm by which the PKSS server and its clients will derive `H1` and `H2`.

(n) `A` — An encryption algorithm. `As` is a symmetric key encryption algorithm and `Aa` is an asymmetric key encryption algorithm.

(o) `As...` — A list of secret key encryption algorithms.

(p) `As...[i]` — An index into a list of secret key encryption algorithms.

(q) `H(As...)` — A hash of a list of secret key encryption algorithms.

(r) `L` — The bit length of a public key.

5. REQUIREMENTS

This technology must give customers who want to use public-key encryption for authentication flexibility with respect to whether or when to use and subsequently deploy a hardware-based public-key security infrastructure.

6. FUNCTIONAL DEFINITION

This section defines a PKSS record format, login protocol, and password/key-change protocols.

The record format and protocols specified in this document assume:

(a) Physical and organizational security similar to that required by the DCE Key Distribution Center (KDC).

(b) Use of the DCE Auditing Subsystem.

6.1. User Record and Lookup Key Formats

The PKSS stores records in a database using lookup keys as described below. In this document, the word ''user-key'' means a PKSS database user record lookup key.

The user record lookup key consists of four items, as follows:

(a) A variable-length character array containing the name by which the user is known to DCE.

(b) A domain ID. This specifies the application domain for which use of this key is defined. In this revision, a key pair may only be used in the context of a DCE login.

(c) A key version number. This is an integer that specifies a key's generation.

(d) A set of key usage flags. This is a bitmask which specifies how the key may be used. In this revision, a key pair may only be used for authentication. In a future revision, encryption may be added as an option.

Note that the length field of the username, domain ID, key version number, and usage flags are all integers. To make PKSS databases transportable between different machine architectures, the PKSS converts all lookup key integer data to network byte order before using.

The user record is a four-tuple:

```
[Kxu]H1:H2:Kou:V
```

consisting of four items as follows:

(a) The private key component of the user's asymmetric key pair, `Kxu`, encrypted under `H1`.

(b) A copy of the user's password hash, `H2`.

(c) A copy of the public key component of the user's asymmetric key pair, `Kou`.

(d) The version number `V` of the asymmetric key pair given by `Kxu` and `Kou`. This must match the version number in the lookup key.

Note that the PKSS stores the record in ASN.1 format. Note further that before inserting the record, the PKSS will convert the key version number to network byte order to match the byte order of the key version number in the lookup key.

6.1.1. Observations on the record format

Although strong physical and organizational security measures are assumed, if an attacker compromises these measures, what attacks on the database are possible?

An attacker may mount a forward search attack on passwords to find hashes `H2` that match some of the `H2`'s stored in the database. Note that compromising `H2` reveals `H1` at the same time, and that users' private keys are stored in the database encrypted under H1. Use of strong passwords, therefore, is very desirable, but a means to enforce their use is beyond the scope of this document. (Note that DCE 1.1 already supports some strong password controls.)

6.2. PKSS Database

The PKSS database shall consist of a set of user records as described above. In the first revision, the database will be protected with a default ACL granting the `sec_admin` user permissions to add, delete, and modify user records. The modify permission shall mean the ability to change a user's password and asymmetric key pair. In a future revision, individual user records may be protected by individual ACL's or sets of ACL's but such a discussion is beyond the scope of this document.

6.2.1. Initializing the database

The PKSS database gets its initial ACL's when it is first created. It is created the first time the PKSS in a cell comes up. There are no default user records in a PKSS database.

6.3. User Private Key Acquisition Protocol

This is a four-step sequence.

6.3.1. Client-to-PKSS initial inquiry (step one)

This section describes the client-to-PKSS initial inquiry message:

```
C-->P: userkey:[E(Dc)]:As...
```

A client initiates an interaction with the PKSS by sending the PKSS a message containing the `userkey` in the clear, an exponentiated Diffie-Hellman seed value `E(Dc)`, and a list of symmetric key encryption algorithms `As...` that the client supports.

Upon receipt of an initial inquiry from a client, the PKSS:

(a) Searches the PKSS database for a record matching `userkey`. If not found, the PKSS discontinues processing and returns an error code to the client. NOTE: All PKSS error and status codes are listed and described in the ''REMOTE INTERFACES'' section.

(b) Searches the list of symmetric key encryption algorithms the client supports `As...` for one that the PKSS supports. If the PKSS does not find one that it supports, the PKSS discontinues processing and returns an error code to the client. If the PKSS does find one that it supports, it returns an index `As...[i]` into the the client's list of supported symmetric key encryption algorithms.

(c) Generates its own exponentiated Diffie-Hellman seed value, `E(Dp)`.

(d) Computes a Diffie-Hellman key `D`, using `E(Dc)` suitable for use with the encryption algorithm given by `As...[i]`.

(e) Generates a random symmetric encryption key `S` suitable for use with the algorithm given by `As...[i]` to be used as a session key. Encrypts `S` with `H2` to produce `[S]H2`.

(f) Computes a hash of the client's algorithm list `As...` to produce `H(As...)`.

(g) Encrypts `[S]H2` and `H(As...)` with the Diffie- Hellman key `D` to produce `[[S]H2:H(As...)]D`.

(h) Generates a nonce, `Np`.

(i) Encrypts `Np` with `S`, to obtain `[Np]S`.

6.3.2. PKSS-to-client challenge (step two)

This section describes the PKSS-to-client challenge message:

```
P-->C: E(Dp):[[S]H2:H(As...)]D:[Np]S:As...[i]
```

The PKSS constructs a challenge message consisting of: The PKSS' Diffie-Hellman seed value `E(Dp)`, a two-tuple `[S]H2:H(As...)` encrypted under the Diffie-Hellman key `D`, a nonce `Np` encrypted with the session key `S`, and an index into the symmetric key encryption algorithm list `As...` that is the algorithm the PKSS and the PKSS client will use for secure communication.

Upon receipt of the challenge message, the client:

(a) Computes a Diffie-Hellman key, `D`, using `E(Dp)` suitable for use with the encryption algorithm given by `As...[i]`.

(b) Decrypts `[[S]H2:H(As...)]D` with Diffie-Hellman key `D`, to obtain `[S]H2:H(As...)`.

(c) Computes a hash, `H(As...)` of the algorithm list it sent the PKSS in step one.

(d) Compares the algorithm list hash it just computed with the one returned from the PKSS. If they do not match, the PKSS client discontinues processing and returns an error code.

(e) Decrypts `[S]H2` with the password hash `H2` to obtain the session key, `S`.

(f) Decrypts `[Np]S` with the session key `S`, to obtain the PKSS's nonce, `Np`.

(g) Generates a nonce, `Nc`.

6.3.3. Client-to-PKSS response (step three)

This section describes the client-to-PKSS response message:

```
C-->P: [Nc:Np:I]S
```

The client constructs a response three-tuple consisting of: the nonce `Nc` it generated in the previous step; the nonce `Np` it obtained from the PKSS challenge message; and an operation ID, `I`. NOTE: All PKSS operation ID's are listed in the ''REMOTE INTERFACES'' section. The client sets the operation ID to a pre-defined value that indicates password acquisition. It encrypts the three-tuple with `S` and sends this message to the PKSS.

Upon receipt of the client-to-PKSS response message, the PKSS:

(a) Decrypts the client-to-PKSS response message with `S`, to obtain `Nc:Np:I`. If the `Np` the PKSS sent to the client in the challenge message does not match the `Np` in the client's response message, the PKSS discontinues processing because only the real client could have recovered `E(Dp)` in the challenge message with the password hash `H2` to obtain the session key S required to produce `[Nc:Np:I]S`.

6.3.4. PKSS-to-client response (step four)

This section describes the PKSS-to-client response message:

```
P-->C: [[Kxu]H1:Kou:V:Nc]S
```

The PKSS-to-client response message consists of a four-tuple containing: the user's private key `Kxu` encrypted under `H1`; the user's public key `Kou`; the key version number `V` of `Kxu/Kou`; and the nonce `Nc` it obtained from the client in the previous step. It encrypts the four-tuple with the session key `S`.

Upon receipt of the PKSS-to-client response message, the client:

(a) Decrypts the response message with its copy of the Diffie-Hellman key `D`, to obtain `[Kxu]H1:Kou:V:Nc`. If the `Nc` in this message does not match the `Nc` it sent to the PKSS, the client ignores the response because only the real PKSS could have decrypted the response to its challenge message with `D` to obtain the correct `Nc` to insert in this message.

(b) Decrypts `[Kxu]H1` with `H1`, to obtain the user's private key `Kxu`.

DCE login may now proceed per [RFC 68.2].

6.3.5. Observations on the user private key acquisition protocol

Note that `E(Dc)`, `E(Dp)`, `Nc`, `Np`, and the session key `S` are nonces. Assuming a ''good'' random number generator, that is, one that makes it infeasible for an attacker to guess its output, an attacker who intercepted those messages cannot assail the password with a forward search on `H2` nor assail the Diffie-Hellman key `D`, nor assail the session key `S` since it is not possible to detect whether or not trial decryptions were successful. Therefore:

(a) A spoof attack is not possible without knowledge of the password, because any information a spoofer gets at step two is unusable without the password from which to derive `H2`. Replay and bucket brigade attacks using the message at step one fail for the same reason.

(b) Message stream modification attacks at any step result only in denial of service.

Because DCE RPC does not support chaining, the first two and second two steps of the user private key acquisition protocol will be implemented in separate RPC's, but will use context handles to link them together logically. The exported interfaces to client programs, however, will present a single call that will invoke both RPC's on a client's behalf. Note that the protocol does not rely on the context handles for security.

6.4. User Password/Key-Change Protocols

This section describes two key change protocols, one in which the client generates new key pairs, and one in which the server does. Interfaces for both protocols will be provided in this revision. Each approach has distinct advantages and disadvantages, some of which this document attempts to identify. Note that in both protocols, the user specifies new passwords.

Both protocols are six-step (three RPC) exchanges and require authenticated RPC at each step to assure the PKSS client that it is sending new password/key data to the actual PKSS.

The first four steps are the same as the user private key acquisition protocol (section 6.3), except that the operation ID `I` in 6.3.3 (that is, acquisition protocol step three) is set to a value the indicates either ''record update — client generates keys'' or ''record update — server generates keys''. The purpose of this initial exchange is to establish that client knows `H2` (that is, knows the old password), and to establish a conversation key to be used for the final exchange. The last two messages, described below in sections 6.4.1 (client generates keys) and 6.4.2 (PKSS generates keys) perform the password/key change itself, and furnish additional proof that the client knows `Kxu` (thus proving it also knows `H1`). In all cases in either protocol, the PKSS audits change password/key attempts.

6.4.1. Change password/key (client generates keys)

In this version of the password/key change protocol, the client is responsible to generate new asymmetric key pairs and present them to the server securely. Note that either the asymmetric key pair, the password, or both may be changed with the same message.

6.4.1.1. Client-to-PKSS change password/key message (client generates keys)

This section describes the Client-to-PKSS change password/key message (where the client generates the keys):

```
C-->P: [{[Kxu~]H1~:H2~:Kou~}Kxu]S
```

Once the client has successfully decrypted the PKSS-to-client response message and obtained its private key `Kxu` (step four of the user private key acquisition protocol), it constructs the change password/key message as follows:

(a) Computes new hashes `H1~` and `H2~` of the new password. Observe that this step is not required if only the asymmetric key pair is changing and the password is remaining the same.

(b) Generates a new asymmetric key pair `Kxu~` and `Kou~`. Observe that this step is not required if only the password is changing and the asymmetric key pair key is remaining the same. For simplicity's sake, the symbolic notation used in this section assumes that both the password and the asymmetric key pair are changing.

(c) Encrypts `Kxu~` with `H1~`, to obtain `[Kxu~]H1~`.

(d) Adds `H2~` and `Kou~` to the message. Observe that `[Kxu~]H1~:H2~:Kou~` is the new security data for the PKSS database user record.

(e) Signs `[Kxu~]H1~:H2~:Kou~` with the user's private key `Kxu`, to obtain `{[Kxu~]H1~:H2~:Kou~}Kxu`.

(f) Encrypts the signed change password/key message `{[Kxu~]H1~:H2~:Kou~}Kxu` with the session key `S`, to obtain `[{[Kxu~]H1~:H2~:Kou~}Kxu]D`.

Upon receipt of the change password message from the client, the PKSS:

(a) Decrypts the client's change password/key message `[{[Kxu~]H1~:H2~:Kou~}Kxu]S` with the session key `S`, to obtain `{[Kxu~]H1~:H2~:Kou~}Kxu`.

(b) Verifies the user's digital signature `{[Kxu~]H1~:H2~:Kou~}Kxu` with the user's public key `Kou`, by computing the checksum of `[Kxu~]H1~:H2~:Kou~` and comparing it with the unsealed signature. If the two values match, the PKSS is sure this message originated with the real client, since only the real client would know the session key `S` and the password from which to derive the hash `H1` to obtain the user's private key `Kxu`. If the checksums do not match, the PKSS discontinues further processing.

(c) Obtains new key version number `V~` (actually `V+1`).

(d) Replaces `[Kxu]H1:H2:Kou:V` in the user record with `[Kxu~]H1~:H2~:Kou~:V~`. Observe that if only the password is changing, only `[Kxu]H1:H2` is replaced as there will be no `Kou~`.

6.4.1.2. PKSS-to-client change password ACK/NAK message (client generates keys)

This section describes the PKSS-to-client change password ACK/NAK message (where the client generates the keys):

```
P-->C: [Nc:V:Status]S
```

The PKSS sends an ACK/NAK three-tuple to the client in consisting of: the nonce `Nc` the client sent to the PKSS earlier while attempting to obtain the user's private key; the new key's version number; and a status code. The PKSS encrypts this message message with the session key `S`.

6.4.1.3. Observations on the change protocol (client generates keys)

The messages at steps five and six are encrypted with the session key `S`. This makes the messages at these steps impervious to dictionary attacks. This observation is also true for the version of the change protocol in which the server generates keys.

Because DCE RPC does not support chaining, the first, second, and third pairs of steps of the change password/key protocol will be implemented in three separate RPC's. The exported interface to client programs, however, will present a single call that will invoke all three RPC's on a client's behalf. This is also true for the version of the change protocol in which the server generates keys.

6.4.2. Change password/key (server generates keys)

In this version of the password/key change protocol, the client presents a new password to the server and relies on the server to generate new asymmetric key pairs. This version of the protocol is intended for situations where the PKSS is expected to be more capable of generating keys, either due to superior CPU performance or due to the availability of better random numbers (server hardware is more likely to have a dedicated random number generating device).

6.4.2.1. Client-to-PKSS change password/key message (server generates keys)

This section describes the client-to-PKSS change password/key message (where the server generates the keys):

```
C-->P: [{H1:H2:newpassword:key-chg?:Aa:L}Kxu]S
```

Once the client has successfully decrypted the PKSS-to-client response message and obtained its private key `Kxu`, it constructs the change password/key message as follows:

(a) Computes hashes `H1` and `H2` on the old password.

(b) Adds `H1`, `H2`, and the new password to the message.

(c) Sets a flag `key-chg?` which informs the PKSS whether or not to generate a new asymmetric key pair.

(d) Signs `H1:H2:newpassword:key-chg?` with the user's private key `Kxu`, to obtain `{H1:H2:newpassword:key-chg?}Kxu`.

(e) Encrypts `{H1:H2:newpassword:key-chg?}Kxu` with the session key `S` to obtain `[{H1:H2:newpassword:key-chg?}Kxu]S`.

Upon receipt of the change password message from the client, the PKSS:

(a) Decrypts the message using the session key `S`.

(b) Verifies the user's digital signature `{H1:H2:newpassword:key-chg?}Kxu` with the user's public key `Kou`, by computing the checksum of `H1:H2:newpassword:key-chg?` and comparing it with the unsealed signature. If the two values match, the PKSS is sure this message originated with the real client since only the real client would know the Diffie-Hellman key `D` and the password from which to derive the hash `H1` to obtain the user's private key `Kxu`. If the checksums do not match, the PKSS discontinues further processing.

(c) Inspects the `key-chg?` flag to determine whether or not to generate a new asymmetric key pair. If set, generates the new keys `Kxu˜`, `Kou˜` suitable for use with algorithm `Aa` of length `L`. If not set, decrypts [Kxu]H1 in the user's PKSS database record with the `H1` the client sent the PKSS in the client-to-PKSS change password/key message.

(d) Computes new hashes `H1˜` and `H2˜` on the new password. Observe that `H1˜ = H1` and `H2˜ = H2` if only the key pair is changing but the password is not changing.

(e) Encrypts `Kxu˜` with `H1˜` to obtain `[Kxu˜]H1˜` and concatenates `H2˜` and `Kou˜`. This is the new security data for the user's PKSS database record. Observe that `Kxu˜ = Kxu` and `Kou˜ = Kou` if only the password is changing but the key pair is not changing.

(f) Obtains new key version number `V˜` (actually `V+1`).

(g) Overwrites `[Kxu]H1:H2:Kxu:V` in the user's PKSS database record with `[Kxu˜]H1˜:H2˜:Kou˜:V˜`.

6.4.2.2. PKSS-to-client change password ACK/NAK message (server generates keys)

This section describes the PKSS-to-client change password ACK/NAK message (where the server generates the keys):

```
P-->C: [Nc:V:status:Kou~]S
```

The ACK/NAK message has the same content and semantics in this version of the password/key change protocol as it did in the version in which the client generated new keys, with one exception: If the `key-chg?` flag was set in the change message, the PKSS will also return the client's new public key. Thus, the return message will be either a three-tuple or a four-tuple depending on the contents of the `key-chg?` flag.

6.4.2.3. Observations on the change protocol with server generating keys

Because the PKSS client presents the user's password to the PKSS, this version also allows for PKSS integration with a password strength server. Note, however, that the user's password is exposed briefly on the PKSS.

6.5. PKSS Management Protocols

The `sec_admin` user will be able to add, delete, and modify user records in the PKSS database. The modify operation shall mean to change both password and key pair at the same time. Note that to change password alone requires knowledge of the current password, thus `sec_admin` may only change both at once.

The PKSS management protocols are all four-step (two RPC) exchanges that begin with a modified Diffie-Hellman key exchange [DIFF 76] over a secure RPC to establish a session key to encrypt password/key change data. Authentication and packet-integrity are required to thwart bucket-brigade attacks.

Step one:

```
C-->P: I:E(Dc):As...
```

The PKSS management client initiates an interaction with the PKSS by generating a random number `Xc` and computing the exponentiated Diffie-Hellman seed value `E(Dc) = (P**Xc) mod g`. The PKSS management client sends `E(Dc)` to the PKSS.

The PKSS management client adds to this a PKSS management protocol ID `I` and a list of symmetric key encryption algorithms `As...` that it supports.

Upon receipt of the PKSS management client's message `I:E(Dc):As...`, the PKSS:

(a) Generates a random number `Xp`.

(b) Computes a Diffie-Hellman key `D = (E(Dc)**Xp) mod g`.

Step two:

```
P-->C: E(Dp):H(As...):As...[i]
```

The PKSS sends `E(Dp):H(As...):As...[i]` to the PKSS management client. Upon receipt of, the PKSS management client:

(a) Computes a Diffie-Hellman key `D = (E(Dp)**Xc) mod g` suitable for use with the symmetric key encryption algorithm given by `As...[i]`.

(b) Computes is own hash `H(As...)` and compares it with the `H(As...)` that came back from the PKSS. If they do not match, the client discontinues processing, and returns an error code to the client.

The PKSS and its management client now have between them a session key D for private communication.

6.5.1. PKSS management add user

The ''add user'' management function will have two versions, one in which the client generates asymmetric keys and one in which the server does.

6.5.1.1. PKSS management add user (client generates keys)

This section describes the PKSS management add user message (where the client generates the keys):

```
C-->P: [userkey:[Kxu]H1:H2:Kou]D
```

The PKSS management client constructs the security information of the user record and a PKSS database lookup key. It will encrypt this with the Diffie-Hellman key `D`

Upon receipt by the PKSS, it verifies the ACL against the user attempting to perform the add operation, and if the ACL permits the user to perform the operation, the PKSS inserts this record into the database.

Then:

```
P-->C: [V:Status]D
```

The PKSS constructs a response message two-tuple consisting in the actual key version number and a status code.

6.5.1.2. PKSS management add user (server generates keys)

This section describes the PKSS management add user message (where the server generates the keys):

```
C-->P: [userkey:password:Aa:L]D
```

The PKSS management client constructs an add user message four-tuple consisting of: the userkey; the new password; the asymmetric key encryption algorithm to be used, and the bit length of the asymmetric key to be generated.

Upon receipt of this message, the PKSS:

(a) Generates a new key pair.

(b) Hashes the new password.

(c) Constructs a new user record and inserts it into the database.

Then:

```
P-->C: [V:Kou:Status]D
```

The PKSS constructs a response message three-tuple consisting in the actual key version number, the public key component of the key pair it generated on the client's behalf, and a status code.

6.5.2. PKSS management delete user

This section describes the PKSS management delete user message:

```
C-->P: userkey
```

The PKSS management client sends the PKSS a userkey containing the username of the principal whose record is to be deleted. Upon receipt by the PKSS, it verifies the ACL against the user attempting to perform the delete operation, and if the ACL permits the user to perform the operation, the PKSS deletes the indicated record from the database.

Then:

```
P-->C: Status
```

The PKSS sends the PKSS client a status code specifying whether the operation succeeded or failed.

6.5.3. PKSS management modify user

The modify user functions (two versions: client generates keys, and server generates keys) have the same behavior and syntax as the add user functions with the exception that the record given by the userkey will be deleted before the new record is inserted, and the operation ID `I` will be set to a pre-defined value that indicates ‘‘modify a user’’.

6.6. PKSS Context Management

The protocols by which users interact with the PKSS are all multi-step exchanges requiring two or three separate RPC's in order to fulfill requests for service. To make this work, the PKSS must maintain selected data across individual RPC's. It will do this with user-defined data in the context handle.

The user-defined data in the context handle shall contain:

(a) An encryption key established in the first RPC of all protocols. This will be the randomly generated session key for user protocols or the Diffie-Hellman key in the case of management protocols.

(b) A state flag which allows the PKSS to track at what step in which protocol it currently is. (Not used in the current version, but reserved for future use).

(c) A copy of the symmetric key algorithm ID to be used in conjunction with the encryption key above.

(d) A copy of the PKSS nonce.

(e) A copy of the client's nonce.

(f) A copy of the database record lookup key.

(g) A mutex so that the context handle may be locked while in use.

(h) A flag indicating whether the context has been deleted.

6.7. PKSS Event Auditing

All PKSS events are auditable. The PKSS will use the DCE Auditing Subsystem for secure, reliable auditing.

7. DATA STRUCTURES

```
//
// Class definitions for PKSS ASN.1 messages
//

//
// PKSS Application Tags
//

#define PKSS_LOGIN_P1_MSG_TAG           (1)
#define PKSS_LOGIN_P2_MSG_TAG           (2)
#define PKSS_LOGIN_P3_MSG_TAG           (3)
#define PKSS_LOGIN_P4_MSG_TAG           (4)

#define PKSS_CLT_CHG_P1_MSG_TAG         (11)
#define PKSS_CLT_CHG_P2_MSG_TAG         (12)
#define PKSS_CLT_CHG_P3_MSG_TAG         (13)
#define PKSS_CLT_CHG_P4_MSG_TAG         (14)
#define PKSS_CLT_CHG_P5_MSG_TAG         (15)
#define PKSS_CLT_CHG_P6_MSG_TAG         (16)

#define PKSS_SVR_CHG_P1_MSG_TAG         (21)
#define PKSS_SVR_CHG_P2_MSG_TAG         (22)
#define PKSS_SVR_CHG_P3_MSG_TAG         (23)
#define PKSS_SVR_CHG_P4_MSG_TAG         (24)
#define PKSS_SVR_CHG_P5_MSG_TAG         (25)
#define PKSS_SVR_CHG_P6_MSG_TAG         (26)

#define PKSS_MGMT_P1_MSG_TAG            (31)
#define PKSS_MGMT_P2_MSG_TAG            (32)
```

```
#define PKSS_MGMT_CLT_ADD_P3_MSG_TAG    (43)
#define PKSS_MGMT_CLT_ADD_P4_MSG_TAG    (44)

#define PKSS_MGMT_SVR_ADD_P3_MSG_TAG    (53)
#define PKSS_MGMT_SVR_ADD_P4_MSG_TAG    (54)

#define PKSS_MGMT_DEL_P3_MSG_TAG        (63)
#define PKSS_MGMT_DEL_P4_MSG_TAG        (64)
#define PKSS_MGMT_CLT_MOD_P3_MSG_TAG    (73)
#define PKSS_MGMT_CLT_MOD_P4_MSG_TAG    (74)

#define PKSS_MGMT_SVR_MOD_P3_MSG_TAG    (83)
#define PKSS_MGMT_SVR_MOD_P4_MSG_TAG    (84)


//------------------------------------------------------------------+
// class definition for PKSS database key (that is, db search key) |
//------------------------------------------------------------------+

class PkssUserKeyC : public asn_sequence
{
public:
  asn_IA5string uname;
  asn_IA5string domain;  // must convert uuid_t to/from string
  asn_integer   usages;
  asn_integer   version;

  PkssUserKeyC (void)
  {
    register_child (&uname);
    register_child (&domain);
    register_child (&usages);
    register_child (&version);
  }

}; // End class definition for PkssUserKeyC


//------------------------------------------+
// class definition for PKSS database record |
//------------------------------------------+

class PkssDBrecC : public asn_sequence // [Kxu]H1:H2:Kou:V
{
public:
  asn_octetstring KxuH1;
  asn_octetstring H2;
  SubjectPublicKeyInfo Kou;
  asn_integer version;

  PkssDBrecC (void):asn_sequence(SECURE)
  {
    register_child (&KxuH1);
    register_child (&H2);
    register_child (&Kou);
```

```
    register_child (&version);
  }

}; // End class definition for PkssDBrecC


//-------------------------------------------------------+
// class definition for PKSS client algorithm choice list |
//-------------------------------------------------------+

class PkssAlgListC : public asn_sequenceof<asn_integer>
{

}; // End class definition for PkssAlgListC


//-------------------------------------------------+
// class definition for PKSS user login phase 1 msg |
//-------------------------------------------------+

class PkssUsrGetKeyP1MsgBaseC : public asn_sequence
  // C-->P: userKey:E(Dc):As...
{
public:
  PkssUserKeyC userKey;
  asn_integer eDc;
  PkssAlgListC algList;

  PkssUsrGetKeyP1MsgBaseC (void)
  {
    register_child (&userKey);
    register_child (&eDc);
    register_child (&algList);
  }

}; // End class definition for PkssUsrGetKeyP1MsgBaseC


class PkssLoginP1MsgC : public PkssUsrGetKeyP1MsgBaseC
{
public:
  PkssLoginP1MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_LOGIN_P1_MSG_TAG);
  }

}; // End class definition for PkssLoginP1MsgC


//--------------------------------------------------+
// class definition for PKSS user login phase 2 data |
//--------------------------------------------------+

class PkssUsrGetKeyP2DataF2C : public asn_sequence // [[S]H2:H(As...)
```

```
{
public:
  asn_octetstring sH2;
  asn_octetstring hAlgs;

  PkssUsrGetKeyP2DataF2C (void):asn_sequence(SECURE)
  {
    register_child (&sH2);
    register_child (&hAlgs);
  }

}; // End class definition for PkssUsrGetKeyP2DataF2C


//-------------------------------------------------+
// class definition for PKSS user login phase 2 msg |
//-------------------------------------------------+

class PkssUsrGetKeyP2MsgBaseC : public asn_sequence
      // P-->C: E(Dp):[[S]H2:H(As...)]D:[Np]S:As...[i]
{
public:
  asn_integer eDp;
  asn_octetstring sH2HalgsD;
  asn_octetstring nps;
  asn_integer algIdx

  PkssUsrGetKeyP2MsgBaseC (void)
  {
    register_child (&eDp);
    register_child (&sH2HalgsD);
    register_child (&nps);
    register_child (&algIdx);
  }

}; // End class definition for PkssUsrGetKeyP2MsgBaseC


class PkssLoginP2MsgC : public PkssUsrGetKeyP2MsgBaseC
{
public:
  PkssLoginP2MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_LOGIN_P2_MSG_TAG);
  }

}; // End class definition for PkssLoginP2MsgC


//--------------------------------------------------+
// class definition for PKSS user login phase 3 data |
//--------------------------------------------------+

class PkssUsrGetKeyP3DataC : public asn_sequence
```

```
{
public:
  asn_integer protoID;
  asn_integer np;
  asn_integer nc;

  PkssUsrGetKeyP3DataC (void):asn_sequence(SECURE)
  {
    register_child (&protoID);
    register_child (&np);
    register_child (&nc);
  }

}; // End class definition for PkssUsrGetKeyP3DataC


//--------------------------------------------------+
// class definition for PKSS user login phase 3 msg |
//--------------------------------------------------+

class PkssUsrGetKeyP3MsgBaseC : public asn_octetstring
                                       // C-->P: [I:Np:Nc]S
{

}; // End class definition for PkssUsrGetKeyP3MsgBaseC


class PkssLoginP3MsgC : public PkssUsrGetKeyP3MsgBaseC
{
public:
  PkssLoginP3MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_LOGIN_P3_MSG_TAG);
  }

}; // End class definition for PkssLoginP3MsgC


//---------------------------------------------------+
// class definition for PKSS user login phase 4 data |
//---------------------------------------------------+

class PkssUsrGetKeyP4DataC : public asn_sequence
{
public:
  asn_octetstring kxuH1;
  SubjectPublicKeyInfo kou;
  asn_integer version;
  asn_integer nc;

  PkssUsrGetKeyP4DataC (void):asn_sequence(SECURE)
  {
    register_child (&kxuH1);
    register_child (&kou);
```

```
    register_child (&version);
    register_child (&nc);
  }

}; // End class definition for PkssUsrGetKeyP4DataC


//---------------------------------------------------+
// class definition for PKSS user login phase 4 msg |
//---------------------------------------------------+

class PkssUsrGetKeyP4MsgBaseC : public asn_octetstring
  // P-->C: [[Kxu]H1:Kou:V:Nc]S
{

}; // End class definition for PkssUsrGetKeyP4MsgBaseC


class PkssLoginP4MsgC : public PkssUsrGetKeyP4MsgBaseC
{
public:
  PkssLoginP4MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_LOGIN_P4_MSG_TAG);
  }

}; // End class definition for PkssLoginP4MsgC


//------------------------------------------------------------+
// class defs for user change key/password msgs (clt gens keys) |
//------------------------------------------------------------+

class PkssCltChgP1MsgC : public PkssUsrGetKeyP1MsgBaseC
{
public:
  PkssCltChgP1MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_CLT_CHG_P1_MSG_TAG);
  }

}; // End class definition for PkssCltChgP1MsgC


class PkssCltChgP2MsgC : public PkssUsrGetKeyP2MsgBaseC
{
public:
  PkssCltChgP2MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_CLT_CHG_P2_MSG_TAG);
  }
```

```
}; // End class definition for PkssCltChgP2MsgC


class PkssCltChgP3MsgC : public PkssUsrGetKeyP3MsgBaseC
{
public:
  PkssCltChgP3MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_CLT_CHG_P3_MSG_TAG);
  }

}; // End class definition for PkssCltChgP3MsgC


class PkssCltChgP4MsgC : public PkssUsrGetKeyP4MsgBaseC
{
public:
  PkssCltChgP4MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_CLT_CHG_P4_MSG_TAG);
  }

}; // End class definition for PkssCltChgP4MsgC


//-------------------------------------------------------------------+
// class def for PKSS user chg key/pwd phase 5 data (clt gens keys) |
//-------------------------------------------------------------------+

class PkssCltChgP5DataC : public asn_sequence // [Kxu~]H1~:H2~:Kou~
{
public:
  asn_octetstring newKxuH1;
  asn_octetstring newH2;
  SubjectPublicKeyInfo newKou;

  PkssCltChgP5DataC (void):asn_sequence(SECURE)
  {
    register_child (&newKxuH1);
    register_child (&newH2);
    register_child (&newKou);
  }

}; // End class definition for PkssCltChgP5DataC


class PkssCltChgP5SignedC : public asn_sequence
                                   // {[Kxu~]H1~:H2~:Kou~}Kxu
{
public:
  PkssCltChgP5DataC dataToBeSigned;
  asn_integer sigAlg;
  asn_octetstring signature;
```

```
  PkssCltChgP5SignedC (void)
  {
    register_child (&dataToBeSigned);
    register_child (&sigAlg);
    register_child (&signature);
  }

}; // End class definition for PkssCltChgP5SignedC


//-----------------------------------------------------------------+
// class def for PKSS user chg key/pwd phase 5 msg (clt gens keys) |
//-----------------------------------------------------------------+

class PkssCltChgP5MsgC : public asn_octetstring
{
public:

  PkssCltChgP5MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_CLT_CHG_P5_MSG_TAG);
  }

}; // End class definition for PkssCltChgP5MsgC


//------------------------------------------------------------------+
// class def for PKSS user chg key/pwd phase 6 data (clt gens keys) |
//------------------------------------------------------------------+

class PkssCltChgP6DataC : public asn_sequence
{
public:
  asn_integer nc;
  asn_integer version;
  asn_integer status;

  PkssCltChgP6DataC (void):asn_sequence (SECURE)
  {
    register_child (&nc);
    register_child (&version);
    register_child (&status);
  }

}; // End class definition for PkssCltChgP6DataC


//-----------------------------------------------------------------+
// class def for PKSS user chg key/pwd phase 6 msg (clt gens keys) |
//-----------------------------------------------------------------+

class PkssCltChgP6MsgC : public asn_octetstring
{
public:
```

```
  PkssCltChgP6MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_CLT_CHG_P6_MSG_TAG);
  }

}; // End class definition for PkssCltChgP6MsgC


//---------------------------------------------------------------+
// class defs for user change key/password msgs (svr gens keys) |
//---------------------------------------------------------------+


class PkssSvrChgP1MsgC : public PkssUsrGetKeyP1MsgBaseC
{
public:
  PkssSvrChgP1MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_SVR_CHG_P1_MSG_TAG);
  }

}; // End class definition for PkssSvrChgP1MsgC


class PkssSvrChgP2MsgC : public PkssUsrGetKeyP2MsgBaseC
{
public:
  PkssSvrChgP2MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_SVR_CHG_P2_MSG_TAG);
  }

}; // End class definition for PkssSvrChgP2MsgC


class PkssSvrChgP3MsgC : public PkssUsrGetKeyP3MsgBaseC
{
public:
  PkssSvrChgP3MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_SVR_CHG_P3_MSG_TAG);
  }

}; // End class definition for PkssSvrChgP3MsgC


class PkssSvrChgP4MsgC : public PkssUsrGetKeyP4MsgBaseC
{
public:
  PkssSvrChgP4MsgC (void)
  {
```

```
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_SVR_CHG_P4_MSG_TAG);
  }

}; // End class definition for PkssSvrChgP4MsgC


//---------------------------------------------------------------------+
// class def for PKSS user chg key/pwd phase 5 data (svr gens keys) |
//---------------------------------------------------------------------+

class PkssSvrChgP5DataC : public asn_sequence
        // H1:H2:newpassword:key-chg?:Aa:L
{
public:
  asn_octetstring oldH1;
  asn_octetstring oldH2;
  asn_IA5string newPwd;
  asn_boolean keyChg;
  asn_octetstring pkAlgID;
  asn_integer pkKeyLen;

  PkssSvrChgP5DataC (void):asn_sequence (SECURE)
  {
    register_child (&oldH1);
    register_child (&oldH2);
    register_child (&newPwd);
    register_child (&keyChg);
    register_child (&pkAlgID);
    register_child (&pkKeyLen);
  }

}; // End class definition for PkssSvrChgP5DataC


class PkssSvrChgP5SignedC : public asn_sequence
  // {H1:H2:newpassword:key-chg?:A:L}Kxu
{
public:
  PkssSvrChgP5DataC dataToBeSigned;
  asn_integer sigAlg;
  asn_octetstring signature;

  PkssSvrChgP5SignedC (void):asn_sequence (SECURE)
  {
    register_child (&dataToBeSigned);
    register_child (&sigAlg);
    register_child (&signature);
  }

}; // End class definition for PkssSvrChgP5SignedC


//--------------------------------------------------------------------+
// class def for PKSS user chg key/pwd phase 5 msg (svr gens keys) |
```

```
//-----------------------------------------------------------------+

class PkssSvrChgP5MsgC : public asn_octetstring
{
public: // C-->P: [{H1:H2:newpwd:key-chg?:Aa:L}Kxu]S

  PkssSvrChgP5MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_SVR_CHG_P5_MSG_TAG);
  }

}; // End class definition for PkssSvrChgP5MsgC


//-----------------------------------------------------------------+
// class def for PKSS user chg key/pwd phase 6 data (svr gens keys) |
//-----------------------------------------------------------------+

class PkssSvrChgP6DataC : public asn_sequence
{
public:
  asn_integer status;
  asn_integer nc;
  asn_integer version;
  SubjectPublicKeyInfo newKou;

  PkssSvrChgP6DataC (void):asn_sequence (SECURE)
  {
    register_child (&nc);
    register_child (&version);
    register_child (&status);
    register_child (&newKou);
    version.set_optional();
    newKou.set_optional();
  }

}; // End class definition for PkssSvrChgP6DataC


//-----------------------------------------------------------------+
// class def for PKSS user chg key/pwd phase 6 msg (svr gens keys) |
//-----------------------------------------------------------------+

class PkssSvrChgP6MsgC : public asn_octetstring
{
public:

  PkssSvrChgP6MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_SVR_CHG_P6_MSG_TAG);
  }

}; // End class definition for PkssSvrChgP6MsgC
```

```
//-----------------------------------------------+
// PKSS mgmt protocols; PKSS mgmt D-H key exchange |
//-----------------------------------------------+

class PkssMgmtP1MsgC : public asn_sequence  // C-->P:  I:E(Dc):As...
{
public:
  asn_integer protoID;
  asn_integer eDc;
  PkssAlgListC algList;

  PkssMgmtP1MsgC (void)
  {
    register_child (&protoID);
    register_child (&eDc);
    register_child (&algList);
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_P1_MSG_TAG);
  }

}; // End class definition for PkssMgmtP1MsgC


class PkssMgmtP2MsgC : public asn_sequence
    // P-->C:  E(Dp):H(As...):As...[i]
{
public:
  asn_integer eDp;
  asn_octetstring hAlgs;
  asn_integer algIdx;

  PkssMgmtP2MsgC (void)
  {
    register_child (&eDp);
    register_child (&hAlgs);
    register_child (&algIdx);
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_P2_MSG_TAG);
  }

}; // End class definition for PkssMgmtP2MsgC


//-------------------------------------------------------+
// class def for PKSS mgmt phase 3 data (clt gens keys) |
//-------------------------------------------------------+

class PkssMgmtCltP3DataC : public asn_sequence
{
public:
  PkssUserKeyC userKey;
  asn_octetstring kxuH1;
  asn_octetstring H2;
  SubjectPublicKeyInfo kou;
```

```
  PkssMgmtCltP3DataC (void):asn_sequence (SECURE)
  {
    register_child (&userKey);
    register_child (&kxuH1);
    register_child (&H2);
    register_child (&kou);
  }

}; // End class definition for PkssMgmtCltP3DataC


//----------------------------------------------------------+
// class def for PKSS mgmt phase 3 msg base (clt gens keys) |
//----------------------------------------------------------+

class PkssMgmtCltP3MsgBaseC : public asn_octetstring
  // C-->P: [Userkey:[Kxu]H1:H2:Kou]D
{

}; // End class definition for PkssMgmtCltP3MsgBaseC


//--------------------------------------------------------------+
// class def for PKSS mgmt add user phase 3 msg (clt gens keys) |
//--------------------------------------------------------------+

class PkssMgmtCltAddP3MsgC : public PkssMgmtCltP3MsgBaseC
{
public:
  PkssMgmtCltAddP3MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_CLT_ADD_P3_MSG_TAG);
  }

}; // End class definition for PkssMgmtCltAddP3MsgC


//------------------------------------------------------+
// class def for PKSS mgmt phase 4 data (clt gens keys) |
//------------------------------------------------------+

class PkssMgmtCltP4DataC : public asn_sequence
{
public:
  asn_integer version;
  asn_integer status;

  PkssMgmtCltP4DataC (void)
  {
    register_child (&version);
    register_child (&status);
  }

}; // End class definition for PkssMgmtCltP4DataC
```

```
//----------------------------------------------------------+
// class def for PKSS mgmt phase 4 msg base (clt gens keys) |
//----------------------------------------------------------+

class PkssMgmtCltP4MsgBaseC : public asn_octetstring
// P-->C: [V:Status]D
{

}; // End class definition for PkssMgmtCltP4MsgBaseC


//--------------------------------------------------------------+
// class def for PKSS mgmt add user phase 4 msg (clt gens keys) |
//--------------------------------------------------------------+

class PkssMgmtCltAddP4MsgC : public PkssMgmtCltP4MsgBaseC
{
public:
  PkssMgmtCltAddP4MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_CLT_ADD_P4_MSG_TAG);
  }

}; // End class definition for PkssMgmtCltAddP4MsgC


//------------------------------------------------------+
// class def for PKSS mgmt phase 3 data (svr gens keys) |
//------------------------------------------------------+

class PkssMgmtSvrP3DataC : public asn_sequence
{
public:
  PkssUserKeyC userKey;
  asn_IA5string pwd;
  asn_octetstring pkAlgID;
  asn_integer pkKeyLen;

  PkssMgmtSvrP3DataC (void):asn_sequence (SECURE)
  {
    register_child (&userKey);
    register_child (&pwd);
    register_child (&pkAlgID);
    register_child (&pkKeyLen);
  }

}; // End class definition for PkssMgmtSvrP3DataC


//----------------------------------------------------------+
// class def for PKSS mgmt phase 3 msg base (svr gens keys) |
//----------------------------------------------------------+
class PkssMgmtSvrP3MsgBaseC : public asn_octetstring
  // C-->P: [Username:password:Aa:L]D
```

```
{

}; // End class definition for PkssMgmtSvrP3MsgBaseC


//-----------------------------------------------------------------+
// class def for PKSS mgmt add user phase 3 msg (svr gens keys) |
//-----------------------------------------------------------------+

class PkssMgmtSvrAddP3MsgC : public PkssMgmtSvrP3MsgBaseC
{
public:
  PkssMgmtSvrAddP3MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_SVR_ADD_P3_MSG_TAG);
  }

}; // End class definition for PkssMgmtSvrAddP3MsgC


//---------------------------------------------------------+
// class def for PKSS mgmt phase 4 data (svr gens keys) |
//---------------------------------------------------------+

class PkssMgmtSvrP4DataC : public asn_sequence
{
public:
  asn_integer status;
  asn_integer version;
  SubjectPublicKeyInfo kou;

  PkssMgmtSvrP4DataC (void)
  {
    register_child (&status);
    register_child (&version);
    register_child (&kou);
    version.set_optional();
    kou.set_optional();
  }

}; // End class definition for PkssMgmtSvrP4DataC


//-------------------------------------------------------------+
// class def for PKSS mgmt phase 4 msg base (svr gens keys) |
//-------------------------------------------------------------+

class PkssMgmtSvrP4MsgBaseC : public asn_octetstring
{

}; // End class definition for PkssMgmtSvrP4MsgBaseC


//-----------------------------------------------------------------+
```

```
// class def for PKSS mgmt add user phase 4 msg (svr gens keys) |
//--------------------------------------------------------------+

class PkssMgmtSvrAddP4MsgC : public PkssMgmtSvrP4MsgBaseC
  // P-->C: [V:Kou:Status]D
{
public:

  PkssMgmtSvrAddP4MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_SVR_ADD_P4_MSG_TAG);
  }

}; // End class definition for PkssMgmtSvrAddP4MsgC


//----------------------------------------------+
// class def for PKSS mgmt del user phase 3 msg |
//----------------------------------------------+

class PkssMgmtDelP3MsgC : public asn_sequence
{
public:

  PkssUserKeyC userKey;

  PkssMgmtDelP3MsgC (void)
  {
    register_child (&userKey);
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_DEL_P3_MSG_TAG);
  }

}; // End class definition for PkssMgmtDelP3MsgC


//----------------------------------------------+
// class def for PKSS mgmt del user phase 4 msg |
//----------------------------------------------+

class PkssMgmtDelP4MsgC : public asn_sequence
{
public:
  asn_integer status;

  PkssMgmtDelP4MsgC (void)
  {
    register_child (&status);
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_DEL_P4_MSG_TAG);
  }

}; // End class definition for PkssMgmtDelP4MsgC
```

```
//-----------------------------------------------+
// PKSS mgmt modify user (client generates keys) |
//-----------------------------------------------+

class PkssMgmtCltModP3MsgC : public PkssMgmtCltP3MsgBaseC
{
public:
  PkssMgmtCltModP3MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_CLT_MOD_P3_MSG_TAG);
  }

}; // End class definition for PkssMgmtCltModP3MsgC


class PkssMgmtCltModP4MsgC : public PkssMgmtCltP4MsgBaseC
{
public:
  PkssMgmtCltModP4MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_CLT_MOD_P4_MSG_TAG);
  }

}; // End class definition for PkssMgmtCltModP4MsgC


//-----------------------------------------------+
// PKSS mgmt modify user (server generates keys) |
//-----------------------------------------------+

class PkssMgmtSvrModP3MsgC : public PkssMgmtSvrP3MsgBaseC
{
public:
  PkssMgmtSvrModP3MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_SVR_MOD_P3_MSG_TAG);
  }

}; // End class definition for PkssMgmtSvrModP3MsgC


class PkssMgmtSvrModP4MsgC : public PkssMgmtSvrP4MsgBaseC
{
public:
  PkssMgmtSvrModP4MsgC (void)
  {
    set_tag_class (APPLICATION_CLASS);
    set_tag (PKSS_MGMT_SVR_MOD_P4_MSG_TAG);
  }

}; // End class definition for PkssMgmtSvrModP4MsgC
```

```
//
// Client context handle - required by sec_pvtkey_* layer
//
// See [RFC 68.2]
//

typedef struct
{
  char *name_p;
  char *pkss_svc_profile_name_p;
  sec_pk_domain_t domain;
} sec_pkss_clt_ctx_t;


//
// PKSS server context handle block
//

typedef struct
{
  unsigned32 proto;
  unsigned32 sym_alg;
  unsigned32 state;
  unsigned32 deleted;
  pthread_mutex_t mutex;
  // ctx_timer;
  PkssDbKeyC *dbkey_p;
  sec_pk_gen_data_t *svr_nonce_p;
  sec_pk_gen_data_t *clt_nonce_p;
  sec_pk_gen_data_t *crypto_key_p;
} sec_pkss_context_t;


//
// PKSS database record key
//
// contains Username:domain_id:usage_flags:version
// in a buffer consisting in a length field and a
// data field.
//
typedef struct
{
  unsigned32 len;
  unsigned char *data;
} sec_pkss_dbkey_t;

//
// PKSS database record contains ASN.1 encoding of
// [Kxu~]H1~:H2~:Kou~:V in a buffer consisting in a
// length and a data field.
//

typedef struct
{
  unsigned32 asn_rec_len;
```

```
  unsigned char *asn_rec_p;
} sec_pkss_dbrec_t;
```

8. USER INTERFACES

9. API'S

```
//
// PKSS client API prototypes
//

//
// Retrieve user's private key for use in [RFC 68.2] login
//
// Input
//
//    pkss_svc_profile_name_p: Pointer to entry name in namespace
//                             where the search for compatible
//                             binding handles will begin.
//    name_p: Pointer to the user's canonical name within the
//            specified domain.
//    domain_id: The application domain this key belongs to.
//    key_usages: The usage(s) permitted of this keypair.
//    key_vno: The key version number. (0 may be specified to
//             indicate that the routine should select the next
//             appropriate or newest key version number).
//             (In the reference implementation, key_vno starts with
//             1, then monotonically increase 1 for each update.)
//    pwd_p:  Pointer to the user's password string.
//
// Output
//
//    pvt_key_p: Pointer to the encoded private key component of
//               the user's asymmetric key pair.   It is the caller's
//               responsibility to release the storage allocated for
//               the private key component by this function.
//
//    pub_key_p: Pointer to the encoded public key component of
//               the user's asymmetric key pair.   It is the caller's
//               responsibility to release the storage allocated for
//               the public key component by this function.
//
//    ret_key_vno_p:  Pointer to the actual key version number of
//                    the returned key.  It is the caller's
//                    responsibility to provide the storage to
//                    contain the actual key version number.
// Calls        sec_pkss_clt_diffhell
//              sec_pkss_clt_pkey_get
//
//

#if __cplusplus
extern "C"
#endif
 error_status_t sec_pkss_get_key
```

```
 (
   const char *pkss_svc_profile_name_p,
   const char *name_p,
   sec_pk_domain_t domain_id,
   sec_pk_usage_flags_t key_usages,
   unsigned32 key_vno,
   const char *pwd_p,
   sec_pk_pvtkey_t *pvt_key_p,
   sec_pk_pubkey_t *pub_key_p,
   unsigned32 *ret_key_vno_p
 );

//
// Change user's password and/or asymmetric key pair (client
// generates new asymmetric key pair).
//
// Input
//
//    pkss_svc_profile_name_p: Pointer to entry name in namespace
//                             where the search for compatible
//                             binding handles will begin.
//    name_p: Pointer to the user's canonical name within the
//            specified domain.
//    domain_id: The application domain this key belongs to.
//    key_usages: The usage(s) permitted of this keypair.
//    key_vno: The key version number. (0 may be specified to
//             indicate that the routine should select the next
//             appropriate or newest key version number).
//             (In the reference implementation, key_vno starts with
//             1, then monotonically increase 1 for each update.)
//    old_pwd_p:  Pointer to the user's current password.
//    new_pwd_p:  Pointer to the user's new password.  Callers should
//                set this field to NULL if only the asymmetric key
//                pair is changing, but the password is staying the
//                same.
//    new_pvt_key_p:  a pointer to a buffer containing the private
//                    key component of the new asymmetric key pair.
//    new_pub_key_p:  a pointer to a buffer containing the public key
//                    component of the new asymmetric key pair.
//
// Output
//
//    new_key_vno_p:  Pointer to the actual key version number of
//                    the new key.  It is the caller's responsibility
//                    to provide the storage to contain the actual
//                    key version number.
//
// Calls:        sec_pkss_clt_diffhell
//               sec_pkss_clt_pkey_get
//               sec_pkss_chg_key_clt
//

#if __cplusplus
extern "C"
#endif
```

```
 error_status_t sec_pkss_key_chg_clt
 (
   const char *pkss_svc_profile_name_p,
   const char *name_p,
   sec_pk_domain_t domain_id,
   sec_pk_usage_flags_t key_usages,
   unsigned32 key_vno,
   const char *old_pwd_p,
   const char *new_pwd_p,
   sec_pk_pvtkey_t *new_pvt_key_p,
   sec_pk_pubkey_t *new_pub_key_p,
   unsigned32 *new_key_vno_p
 );

//
// Change user's password and/or asymmetric key pair (server
// generates new asymmetric key pair).
//
// Input
//
//    pkss_svc_profile_name_p: Pointer to entry name in namespace
//                             where the search for compatible
//                             binding handles will begin.
//    name_p: Pointer to the user's canonical name within the
//            specified domain.
//    domain_id: The application domain this key belongs to.
//    key_usages: The usage(s) permitted of this keypair.
//    key_vno: The key version number. (0 may be specified to
//             indicate that the routine should select the next
//             appropriate or newest key version number).
//             (In the reference implementation, key_vno starts with
//             1, then monotonically increase 1 for each update.)
//    old_pwd_p:  Pointer to the user's current password.
//    new_pwd_p:  Pointer to the user's new password.  Callers should
//                set this field to NULL if only the asymmetric key
//                pair is changing, but the password is staying the
//                same.
//    key_change_flag:  If set, instructs PKSS to generate a new
//                      asymmetric key pair.
//    pk_alg_id_p: Pointer to algorithm to use when generating new
//                 public key.
//    pk_len: Key length of public key to be generated by PKSS
//            server.
//
//
// Output
//
//    new_pub_key_p:  Pointer to a buffer to contain the public key
//                    component of the user's asymmetric key pair.
//                    It is the caller's responsibility to call
//                    sec_pkss_key_free() to release the key storage
//                    allocated by this function to contain the key.
//    new_key_vno_p:  Pointer to the actual key version number of
//                    the new key.  It is the caller's responsibility
//                    provide the storage to contain the actual key
```

```
//                     version number.
//
// Calls:        sec_pkss_clt_diffhell
//               sec_pkss_clt_pkey_get
//               sec_pkss_chg_key_svr
//

#if __cplusplus
extern "C"
#endif
 error_status_t sec_pkss_key_chg_svr
 (
   const char *pkss_svc_profile_name_p,
   const char *name_p,
   sec_pk_domain_t domain_id,
   sec_pk_usage_flags_t key_usages,
   unsigned32 key_vno,
   const char *old_pwd_p,
   const char *new_pwd_p,
   unsigned32 key_chg_flag,
   sec_pk_algorithm_id_t *pk_alg_id_p,
   unsigned32 pk_len,
   sec_pk_pubkey_t *new_pub_key_p,
   unsigned32 *new_key_vno_p
 );

#endif // SEC_PKSS_CLT_API_H
```

10. REMOTE INTERFACES

```
/*
 * PKSS IDL - all buffers containing sensitive data will be
 * explicitly zeroed before their storage is released.
 */

const unsigned long PKSS_PWD_HASH_SIZE = 8;
const unsigned long PKSS_NONCE_SIZE = 8;

//
// PKSS operation ID's
//

const unsigned long PKSS_PROTO_USR_KEY_GET = 1;
const unsigned long PKSS_PROTO_USR_KEY_CHG_CLT = 2;
const unsigned long PKSS_PROTO_USR_KEY_CHG_SVR = 3;

const unsigned long PKSS_PROTO_MGMT_ADD_USR_CLT = 11;
const unsigned long PKSS_PROTO_MGMT_ADD_USR_SVR = 12;
const unsigned long PKSS_PROTO_MGMT_DEL_USR = 13;
const unsigned long PKSS_PROTO_MGMT_MOD_USR_CLT = 14;
const unsigned long PKSS_PROTO_MGMT_MOD_USR_SVR = 15;

//
// PKSS change key flag values
//
```

```
const unsigned long PKSS_SVR_GEN_KEY_NO = 0;
const unsigned long PKSS_SVR_GEN_KEY_YES = 1;


/*
 * PKSS context handle
 */
typedef [context_handle] void *sec_pkss_ctx_hdl_t;


/*
 * Structure for PKSS messages
 */

typedef struct
{
  unsigned32 msg_len;
  [ptr, size_is (msg_len)] byte *msg_p;
} sec_pkss_msg_buf_t;

typedef [ptr] sec_pkss_msg_buf_t *sec_pkss_msg_buf_p_t;


/*
 * Improved Encrypted Key Exchange (steps one and two)
 */

error_status_t sec_pkss_clt_diffhell
(
  [in] handle_t binding,
  [out] sec_pkss_ctx_hdl_t *ctx_handle_p,
  [in]  sec_pkss_msg_buf_t *to_pkss_p,
  [out] sec_pkss_msg_buf_p_t *from_pkss_p,
  [in,out] error_status_t *estatus_p
);

/*
 * Improved Encrypted Key Exchange conclusion (steps three and four)
 */

error_status_t sec_pkss_clt_pkey_get
(
  [in] handle_t binding,
  [in,out] sec_pkss_ctx_hdl_t *ctx_handle_p,
  [in]  sec_pkss_msg_buf_t *to_pkss_p,
  [out] sec_pkss_msg_buf_p_t *from_pkss_p,
  [in,out] error_status_t *estatus_p
);

/*
 * Change user's password/key (client generates new Kxu, Kou)
 */

error_status_t sec_pkss_chg_key_clt
(
```

```
  [in] handle_t binding,
  [in]  sec_pkss_ctx_hdl_t *ctx_handle_p,
  [in]  sec_pkss_msg_buf_t *to_pkss_p,
  [out] sec_pkss_msg_buf_p_t *from_pkss_p,
  [in,out] error_status_t *estatus_p
);

/*
 * Change user's password/key (server generates new Kxu, Kou)
 */

error_status_t sec_pkss_chg_key_svr
(
  [in] handle_t binding,
  [in] sec_pkss_ctx_hdl_t *ctx_handle_p,
  [in]  sec_pkss_msg_buf_t *to_pkss_p,
  [out] sec_pkss_msg_buf_p_t *from_pkss_p,
  [in,out] error_status_t *estatus_p
);

//
// PKSS error/status codes
//

start
code            pks_s_no_memory
text            "Cannot allocate memory"
explanation     "Memory allocation operation failed"
action          "Check memory limits on your machine, and increase
                available memory, if possible."
attributes      svc_c_sev_error
sub-component   pks_s_general
end

start
code            pks_s_no_gen_reply
text            "Cannot generate a reply message, %x"
explanation     "Could not generate ASN.1-encoded reply message"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_general
end

start
code            pks_s_invalid_arg
text            "Bad or wrong arguments"
explanation     "Bad or wrong arguments were supplied."
action          "Reinvoke the command with the correct arguments."
attributes      svc_c_sev_error
sub-component   pks_s_general
end

start
code            pks_s_invalid_arg_1_parm
text            "Bad or wrong argument: %d"
```

```
explanation     "Bad or wrong argument were supplied."
action          "Reinvoke the command with the correct arguments."
attributes      svc_c_sev_error
sub-component   pks_s_general
end


start
code            pks_s_cant_open_storage_file
text            "Failed to open file associated with pks object"
explanation     "When attempting to open a file associated with
                the pks object, a failure occurred."
action          "Ensure that the storage_tag field of the id
                paramater refers to an existing file, and that the
                file is accessible to the pks process."
end

start
code            pks_s_no_config_file
text            "Inaccessible or absent configuration file."
explanation     "The PKSS configuration file could not be opened."
action          "Check that a valid configuration exists and can be
                read."
attributes      svc_c_sev_error
sub-component   pks_s_server
end

start
code            pks_s_no_group_name
text            "Cannot read server group name from config info"
explanation     "Cannot read server group name from config info"
action          "Check that a valid configuration exists and can be
                 read."
attributes      svc_c_sev_error
sub-component   pks_s_server
end

start
code            pks_s_already_initialized
text            "The PKSS has already initialized."
explanation     "An initialization routine was called more than
                once."
action          "None - this is an informational status."
attributes      svc_c_sev_notice
sub-component   pks_s_server
end

start
code            pks_s_cant_remove_storage_file
text            "Failed to remove file belonging to pks object"
explanation     "When attempting to remove a file
                referred to by the pks object, a failure occurred."
action          "Ensure that the storage_tag field of the
                id paramater refers to an existing file, and
                that the file and directory in which it resides is
```

```
                 accessible to the pks process."
end

start
code             pks_s_no_acl
text             "The ACL was not found."
explanation      "The ACL did not exist within the database."
action           "Application-specific"
end

start
code             pks_s_bad_acl
text             "An error was detected in a retrieved ACL."
explanation      "An ACL did not pass consistency checks."
action           "Check the integrity of the database."
end

start
code             pks_s_bad_acl_type
text             "The specified ACL type was not recognized."
explanation      "The specified ACL type was not recognized."
action           "Specify a recognized ACL type."
end

start
code             pks_s_invalid_op
text             "The specified authorization operation was
                 unrecognized."
explanation      "Internal error - the specified authorization
                 operation was unrecognized."
action           "Submit a problem report."
end


start            undocumented intable incatalog
code             pks_s_insufficient_space
sub-component    pks_s_pipe
text             "There is not enough room in the buffer."
explanation      "The data would have overflowed the buffer"
action           "Use a larger buffer"
attributes       svc_c_sev_error
end

start
code             pks_s_pipe_error
sub-component    pks_s_pipe
text             "Can't create pipe."
explanation      "The PKSS split-client can't create a pipe"
action           "Check process resources"
attributes       svc_c_sev_error
end

start
code             sec_pkss_pipe_encrypt_fail
sub-component    pks_s_pipe
```

```
text            "Error encrypting for pipe (%x)"
explanation     "Error returned from crypto routine"
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            sec_pkss_pipe_decrypt_fail
sub-component   pks_s_pipe
text            "Error decrypting data from pipe (%x)."
explanation     "Error returned from crypto routine"
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_pipe_svr_encrypt_fail
sub-component   pks_s_pipe
text            "Error encrypting data in server for pipe (%x)."
explanation     "Error returned from crypto routine"
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_pipe_svr_decrypt_fail
sub-component   pks_s_pipe
text            "Error decrypting data in server from pipe (%x)."
explanation     "Error returned from crypto routine"
action          "submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_fork_error
sub-component   pks_s_pipe
text            "Can't create child process."
explanation     "The PKSS split-client can't fork a child"
action          "Check process resources"
attributes      svc_c_sev_error
end

start
code            pks_s_pipe_data_error
sub-component   pks_s_pipe
text            "Pipe data error."
explanation     "Error communicating with child process."
action          "Check process resources"
attributes      svc_c_sev_error
end

start
code            pks_s_int_length_error
text            "Pipe length error."
explanation     "Error calculating pipe data length."
```

```
action          "Submit a problem report"
end

start
code            pks_s_int_length_errorp
sub-component   pks_s_pipe
text            "Pipe length error (%d)."
explanation     "Error calculating pipe data length."
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_exec_e2big
sub-component   pks_s_pipe
text            "Exec failed (E2BIG)."
explanation     "Exec failed, errno = E2BIG."
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_exec_eacces
sub-component   pks_s_pipe
text            "Exec failed (EACCES)."
explanation     "Exec failed, errno = EACCES."
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_exec_eloop
sub-component   pks_s_pipe
text            "Exec failed (ELOOP)."
explanation     "Exec failed, errno = ELOOP."
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_exec_enametoolong
sub-component   pks_s_pipe
text            "Exec failed (ENAMETOOLONG)."
explanation     "Exec failed, errno = ENAMETOOLONG."
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_exec_enoent
sub-component   pks_s_pipe
text            "Exec failed (ENOENT)."
explanation     "Exec failed, errno = ENOENT."
action          "Submit a problem report"
attributes      svc_c_sev_error
```

```
end

start
code            pks_s_exec_enomem
sub-component   pks_s_pipe
text            "Exec failed (ENOMEM)."
explanation     "Exec failed, errno = ENOMEM."
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_exec_enotdir
sub-component   pks_s_pipe
text            "Exec failed (ENOTDIR)."
explanation     "Exec failed, errno = ENOTDIR."
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_exec_enoexec
sub-component   pks_s_pipe
text            "Exec failed (ENOEXEC)."
explanation     "Exec failed, errno = ENOEXEC."
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_exec_fail
sub-component   pks_s_pipe
text            "Exec failed (%d)."
explanation     "Exec failed."
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_pipsvr_nomem
sub-component   pks_s_pipe
text            "Pipe server can’t allocate memory (%d)."
explanation     "The client-side helper can’t allocate memory."
action          "Check process resources"
attributes      svc_c_sev_fatal
end

start
code            pks_s_pipclt_nomem
sub-component   pks_s_pipe
text            "Pipe client can’t allocate memory (%d)."
explanation     "The client can’t allocate memory."
action          "Check process resources"
attributes      svc_c_sev_fatal
end
```

```
start
code            pks_s_pipclt_bad_msg
sub-component   pks_s_pipe
text            "Unexpected message type returned from pipe helper
                (%d)."
explanation     "A protocol violation occured."
action          "Internal error"
attributes      svc_c_sev_error
end

start
code            pks_s_bad_msg_type
text            "Unexpected message type returned from pipe helper."
explanation     "A protocol violation occured."
action          "Internal error"
end

start
code            pks_s_pipsvr_decode_error
sub-component   pks_s_pipe
text            "Error decoding pipe data."
explanation     "Error decoding pipe data."
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_pipsvr_decode_errorp
sub-component   pks_s_pipe
text            "Error decoding pipe data (%d)."
explanation     "Error decoding pipe data."
action          "Submit a problem report"
attributes      svc_c_sev_error
end

start
code            pks_s_unexpected_termination
sub-component   pks_s_pipe
text            "Unexpected child termination,
                status=<0x%8.8lx|code>."
explanation     "The child process unexpectedly terminated.
                The status code code specifies additional
                information."
action          "Correct the indicated error and retry the
                operation."
attributes      "svc_c_sev_fatal | svc_c_action_exit_bad"
end


start           undocumented intable incatalog
code            pks_s_caught_signal
text            "Caught signal %d. Exiting."
attributes      svc_c_sev_notice
sub-component   pks_s_general
end
```

```
start           undocumented intable incatalog
code            pks_s_start
text            "Server starting.."
attributes      svc_c_sev_notice
sub-component   pks_s_server
end

start
code            pks_s_ready_to_listen
sub-component   pks_s_server
attributes      "svc_c_sev_notice"
text            "Ready to listen on interfaces..."
explanation     "The daemon has started."
action          "None required."
end

start           undocumented intable incatalog
code            pks_s_cant_open_database
text            "Cannot open pks database '%s', errno = %d"
attributes      svc_c_sev_fatal
sub-component   pks_s_server
end

start           undocumented intable incatalog
code            pks_s_storage_close
text            "Storage close, status = %d"
attributes      svc_c_sev_error
sub-component   pks_s_server
end

start           undocumented intable incatalog
code            pks_s_filename
text            "GdbC filename not valid"
attributes      svc_c_sev_fatal
sub-component   pks_s_general
end

start           undocumented intable incatalog
code            pks_s_is_listening
text            "pks is listening..."
attributes      svc_c_sev_notice
sub-component   pks_s_general
end

start           undocumented intable incatalog
code            pks_s_cant_init_something
text            "Initialization (<%s|part>) failed,
                status=<0x%8.8lx|code>."
explanation     "The specified part of server initialization
                failed.  The status code code specifies
                additional information."
action          "Correct the indicated error and restart the server."
attributes      "svc_c_sev_fatal | svc_c_action_exit_bad"
sub-component   pks_s_general
end
```

```
start           undocumented intable incatalog
code            pks_s_cant_export_yet
text            "Cannot export to namespace -- sleeping,
                status=<0x%8.8lx|code>."
explanation     "Unable to export a service to the namespace.
                The status code code specifies additional
                information.  The server will keep retrying until it
                succeeds."
action          "If the failiure is not transient, stop the server,
                correct the indicated error, and restart the server."
attributes      svc_c_sev_notice_verbose
sub-component   pks_s_general
end

start           undocumented intable incatalog
code            pks_s_sec_bsafe
text            "Error on sec_bsafe routine, status=<0x%8.8lx|code>."
explanation     "An unexpected error occured during encryption or
                decryption"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_general
end

start           undocumented intable incatalog
code            pks_s_dbwrite_fail
text            "Error writing ACL to database,
                status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while writing an ACL to the
                database"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end

start           undocumented intable incatalog
code            pks_s_acl_encode_fail1
text            "Error 1 encoding an ACL, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while encoding an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_encode_fail2
text            "Error 2 encoding an ACL, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while encoding an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end
```

```
start           undocumented intable incatalog
code            pks_s_acl_encode_fail3
text            "Error 3 encoding an ACL, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while encoding an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_encode_fail4
text            "Error 4 encoding an ACL, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while encoding an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_encode_fail5
text            "Error 5 encoding an ACL, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while encoding an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_encode_fail6
text            "Error 6 encoding an ACL, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while encoding an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_decode_fail1
text            "Error 1 decoding an ACL, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while decoding an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_decode_fail2
text            "Error 2 decoding an ACL, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while decoding an ACL"
action          "none"
```

```
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_decode_fail3
text            "Error 3 decoding an ACL, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while decoding an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_decode_fail4
text            "Error 4 decoding an ACL, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while decoding an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_decode_fail5
text            "Error 5 decoding an ACL, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while decoding an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_defcon_retrieve_fail1
text            "Error 1 retrieving default container ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_defcon_retrieve_fail2
text            "Error 2 retrieving default container ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end
```

```
start           undocumented intable incatalog
code            pks_s_acl_defcon_retrieve_fail3
text            "Error 3 retrieving default container ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_defcon_retrieve_fail4
text            "Error 4 retrieving default container ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_defcon_retrieve_fail5
text            "Error 5 retrieving default object ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_defcon_retrieve_fail6
text            "Error 6 retrieving default object ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_defcon_retrieve_fail7
text            "Error 7 retrieving default object ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end

start           undocumented intable incatalog
```

```
code            pks_s_acl_defobj_retrieve_fail1
text            "Error 1 retrieving default object ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_defobj_retrieve_fail2
text            "Error 2 retrieving default object ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_defobj_retrieve_fail3
text            "Error 3 retrieving default object ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_defobj_retrieve_fail4
text            "Error 4 retrieving default object ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_defobj_retrieve_fail5
text            "Error 5 retrieving default object ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
```

```
code            pks_s_acl_defobj_retrieve_fail6
text            "Error 6 retrieving default object ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_defobj_retrieve_fail7
text            "Error 7 retrieving default object ACL,
                node <%s>, status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_obj_retrieve_fail1
text            "Error 1 retrieving object ACL, node <%s>,
                status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_obj_retrieve_fail2
text            "Error 2 retrieving object ACL, node <%s>,
                status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_obj_retrieve_fail3
text            "Error 3 retrieving object ACL, node <%s>,
                status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
```

```
code            pks_s_acl_con_retrieve_fail1
text            "Error 1 retrieving container ACL, node <%s>,
                status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           undocumented intable incatalog
code            pks_s_acl_con_retrieve_fail2
text            "Error 2 retrieving container ACL, node <%s>,
                status=<0x%8.8lx|code>."
explanation     "Unexpected error occured while retrieving an ACL"
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end


start           intable incatalog
code            pks_s_acl_not_authorized
text            "Unauthorized add or modify attempt on user '%s'."
explanation     "Caller attempted add or modify operation without
                required privileges."
action          "none"
attributes      svc_c_sev_error
sub-component   pks_s_acl
end

start           !intable undocumented
code            pks_i_svc_general
text            "General PKS facility"
end

start           !intable undocumented
code            pks_i_svc_server
text            "The PKS server"
end

start           !intable undocumented
code            pks_i_svc_acl
text            "PKS ACL manager"
end

start           !intable undocumented
code            pks_i_svc_pipe
text            "Split-client pipe manipulation"
end
```

11. MANAGEMENT INTERFACES

```
/*
 * PKSS management IDL - all buffers containing sensitive data
```

```
 * will be explicitly zeroed before their storage is released.
 */

/*
 * PKSS Diffie-Hellman key exchange
 */

error_status_t sec_pkss_mgmt_diffhell
(
  [in] handle_t binding,
  [in,out] sec_pkss_ctx_hdl_t *ctx_handle_p,
  [in]  sec_pkss_msg_buf_t *to_pkss_p,
  [out] sec_pkss_msg_buf_p_t *from_pkss_p,
  [in,out] error_status_t *estatus_p
);

/*
 * PKSS management add user (client generates keys)
 */
error_status_t sec_pkss_mgmt_add_user_clt
(
  [in] handle_t binding,
  [in,out] sec_pkss_ctx_hdl_t *ctx_handle_p,
  [in]  sec_pkss_msg_buf_t *to_pkss_p,
  [out] sec_pkss_msg_buf_p_t *from_pkss_p,
  [in,out] error_status_t *estatus_p
);

/*
 * PKSS management add user (server generates keys)
 */
error_status_t sec_pkss_mgmt_add_user_svr
(
  [in] handle_t binding,
  [in,out] sec_pkss_ctx_hdl_t *ctx_handle_p,
  [in]  sec_pkss_msg_buf_t *to_pkss_p,
  [out] sec_pkss_msg_buf_p_t *from_pkss_p,
  [in,out] error_status_t *estatus_p
);

/*
 * PKSS management delete user
 */
error_status_t sec_pkss_mgmt_del_user
(
  [in] handle_t binding,
  [in]  sec_pkss_msg_buf_t *to_pkss_p,
  [out] sec_pkss_msg_buf_p_t *from_pkss_p,
  [in,out] error_status_t *estatus_p
);

/*
 * PKSS management modify user (client generates keys)
 */
error_status_t sec_pkss_mgmt_mod_user_clt
```

```
(
  [in] handle_t binding,
  [in,out] sec_pkss_ctx_hdl_t *ctx_handle_p,
  [in]  sec_pkss_msg_buf_t *to_pkss_p,
  [out] sec_pkss_msg_buf_p_t *from_pkss_p,
  [in,out] error_status_t *estatus_p
);

/*
 * PKSS management modify user (server generates keys)
 */
error_status_t sec_pkss_mgmt_mod_user_svr
(
  [in] handle_t binding,
  [in,out] sec_pkss_ctx_hdl_t *ctx_handle_p,
  [in]  sec_pkss_msg_buf_t *to_pkss_p,
  [out] sec_pkss_msg_buf_p_t *from_pkss_p,
  [in,out] error_status_t *estatus_p
);


//
// PKSS management client API prototypes
//

//
// Add a new user (client generates asymmetric key pair).
//
// Input
//
//    pkss_svc_profile_name_p: Pointer to entry name in namespace
//                             where the search for compatible
//                             binding handles will begin.
//    name_p: Pointer to the user's canonical name within the
//            specified domain.
//    domain_id: The application domain this key belongs to.
//    key_usages: The usage(s) permitted of this keypair.
//    key_vno: The key version number. (0 may be specified to
//             indicate that the routine should select the next
//             appropriate or newest key version number).
//             (In the reference implementation, key_vno starts with
//             1, then monotonically increase 1 for each update.)
//    pvt_key_p:  a pointer to a buffer containing the private key
//              component of the new asymmetric key pair.
//    pub_key_p:  a pointer to a buffer containing the public key
//              component of the new asymmetric key pair.
//    pwd_p:  Pointer to the new user's password.  It is the
//            caller's responsibility to provide the storage to
//            contain the password.
//
// Output
//
//    new_key_vno_p:  Pointer to the actual key version number of
//                    the new key.  It is the caller's responsibility
//                    to provide the storage to contain the actual
```

```
//                    key version number.
//
// Calls:       sec_pkss_mgmt_diffhell
//              sec_pkss_mgmt_add_user_clt
//

#if __cplusplus
extern "C"
#endif
 error_status_t sec_pkss_mgmt_user_add_clt
 (
   const char *pkss_svc_profile_name_p,
   const char *name_p,
   sec_pk_domain_t domain_id,
   sec_pk_usage_flags_t key_usages,
   unsigned32 key_vno,
   sec_pk_pvtkey_t *pvt_key_p,
   sec_pk_pubkey_t *pub_key_p,
   const char *pwd_p,
   unsigned32 *new_key_vno_p
 );

//
// Add a new user (server generates asymmetric key pair).
//
// Input
//
//    pkss_svc_profile_name_p: Pointer to entry name in namespace
//                             where the search for compatible
//                             binding handles will begin.
//    name_p: Pointer to the user's canonical name within the
//            specified domain.
//    domain_id: The application domain this key belongs to.
//    key_usages: The usage(s) permitted of this keypair.
//    key_vno: The key version number. (0 may be specified to
//             indicate that the routine should select the next
//             appropriate or newest key version number).
//             (In the reference implementation, key_vno starts with
//             1, then monotonically increase 1 for each update.)
//    pwd_p:  Pointer to the new user's password.  It is the
//            caller's responsibility to provide the storage to
//            contain the password.
//    alg_id_p:  A pointer to a public key algorithm identifier.
//    key_len:  Bit-length of new public key to be generated by PKSS.
//
// Output
//
//    new_pub_key_p:  Pointer to a buffer to contain the public key
//                    component of the user's asymmetric key pair.
//                    It is the caller's responsibility to to provice
//                    the storage to contain the key.
//    new_key_vno_p:  Pointer to the actual key version number of
//                    the new key.  It is the caller's responsibility
//                    to provice the storage to contain the actual
//                    key version number.
```

```
//
// Calls:       sec_pkss_mgmt_diffhell
//              sec_pkss_mgmt_add_user_svr
//

#if __cplusplus
extern "C"
#endif
 error_status_t sec_pkss_mgmt_user_add_svr
  (
   const char *pkss_svc_profile_name_p,
   const char *name_p,
   sec_pk_domain_t domain_id,
   sec_pk_usage_flags_t key_usages,
   unsigned32 key_vno,
   const char *pwd_p,
   sec_pk_algorithm_id_t *alg_id_p,
   unsigned32 key_len,
   sec_pk_pubkey_t *new_pub_key_p,
   unsigned32 *new_key_vno_p
  );

//
// Delete a user
//
// Input
//
//    pkss_svc_profile_name_p: Pointer to entry name in namespace
//                             where the search for compatible
//                             binding handles will begin.
//    name_p: Pointer to the user's canonical name within the
//            specified domain.
//    domain_id: The application domain this key belongs to.
//    key_usages: The usage(s) permitted of this keypair.
//    key_vno: The key version number. (0 may be specified to
//             indicate that the routine should select the next
//             appropriate or newest key version number).
//             (In the reference implementation, key_vno starts with
//             1, then monotonically increase 1 for each update.)
// Output
//
//    none
//
// Calls:       sec_pkss_mgmt_diffhell
//              sec_pkss_mgmt_del_user
//

#if __cplusplus
extern "C"
#endif
 error_status_t sec_pkss_mgmt_user_del
  (
   const char *pkss_svc_profile_name_p,
   const char *name_p,
   sec_pk_domain_t domain_id,
```

```
   sec_pk_usage_flags_t key_usages,
   unsigned32 key_vno
 );

//
// Modify a user record (client generates asymmetric key pair).
//
// Input
//
//    pkss_svc_profile_name_p: Pointer to entry name in namespace
//                             where the search for compatible
//                             binding handles will begin.
//    name_p: Pointer to the user's canonical name within the
//            specified domain.
//    domain_id: The application domain this key belongs to.
//    key_usages: The usage(s) permitted of this keypair.
//    key_vno: The key version number. (0 may be specified to
//             indicate that the routine should select the next
//             appropriate or newest key version number).
//             (In the reference implementation, key_vno starts with
//             1, then monotonically increase 1 for each update.)
//    pvt_key_p:  a pointer to a buffer containing the private key
//                component of the new asymmetric key pair.
//    pub_key_p:  a pointer to a buffer containing the public key
//                component of the new asymmetric key pair.
//    pwd_p:  Pointer to the new user's password.  It is the
//            caller's responsibility to provide the storage to
//            contain the password.
//
// Output
//
//    new_key_vno_p:  Pointer to the actual key version number of
//                    the new key.  It is the caller's responsibility
//                    to provide the storage to contain the actual
//                    key version number.
//
// Calls:       sec_pkss_mgmt_diffhell
//              sec_pkss_mgmt_mod_user_clt
//

#if __cplusplus
extern "C"
#endif
 error_status_t sec_pkss_mgmt_user_mod_clt
 (
   const char *pkss_svc_profile_name_p,
   const char *name_p,
   sec_pk_domain_t domain_id,
   sec_pk_usage_flags_t key_usages,
   unsigned32 key_vno,
   sec_pk_pvtkey_t *pvt_key_p,
   sec_pk_pubkey_t *pub_key_p,
   const char *pwd_p,
   unsigned32 *new_key_vno_p
 );
```

```
//
// Modify a user record (server generates asymmetric key pair).
//
// Input
//
//    pkss_svc_profile_name_p: Pointer to entry name in namespace
//                             where the search for compatible
//                             binding handles will begin.
//    name_p: Pointer to the user’s canonical name within the
//            specified domain.
//    domain_id: The application domain this key belongs to.
//    key_usages: The usage(s) permitted of this keypair.
//    key_vno: The key version number. (0 may be specified to
//             indicate that the routine should select the next
//             appropriate or newest key version number).
//             (In the reference implementation, key_vno starts with
//             1, then monotonically increase 1 for each update.)
//    pwd_p:  Pointer to the new user’s password.  It is the
//            caller’s responsibility to provide the storage to
//            contain the password.
//    alg_id_p:  A pointer to a public key algorithm identifier.
//    key_len:  Bit-length of new public key to be generated by PKSS.
//
// Output
//
//    new_pub_key_p:  Pointer to a buffer to contain the public key
//                    component of the user’s asymmetric key pair.
//                    It is the caller’s responsibility to provide
//                    the storage to contain the key.
//
//    new_key_vno_p:  Pointer to the actual key version number of
//                    the new key.  It is the caller’s responsibility
//                    to provide the storage to contain the actual
//                    key version number.
//
// Calls:        sec_pkss_mgmt_diffhell
//               sec_pkss_mgmt_mod_user_svr
//

#if __cplusplus
extern "C"
#endif
 error_status_t sec_pkss_mgmt_user_mod_svr
 (
   const char *pkss_svc_profile_name_p,
   const char *name_p,
   sec_pk_domain_t domain_id,
   sec_pk_usage_flags_t key_usages,
   unsigned32 key_vno,
   const char *pwd_p,
   sec_pk_algorithm_id_t *alg_id_p,
   unsigned32 key_len,
   sec_pk_pubkey_t *new_pub_key_p,
   unsigned32 *new_key_vno_p
 );
```

12. RESTRICTIONS AND LIMITATIONS

(a) The current version of the PKSS does not allow the use of alternate Diffie-Hellman blocks during key negotiation.

(b) The current version of the PKSS does not provide a write lock capability on the database. Such a feature would facilitate backups.

REFERENCES

[DIFF 76] Diffie, W. & Hellman, M., ‘‘New Directions in Cryptography’’, IEEE Transactions on Information Theory, Vol. 22 #6, 1976.

[ITU X.509] ITU, ‘‘Final Text of the 1993 Edition of ISO/IEC 9594-8/ITU-T Rec X.509, Information Technology — Open Systems Interconnection — The Directory: Authentication Framework’’, ISO/IEC JTC 1/SC 31 N 8696, 28 June 1994.

[RFC 68.2] Anderson, A. & Cuti, S., ‘‘DCE 1.2.2 Public Key Login — Functional Specification’’, OSF-RFC 68.2, February 1996.

[RFC 1507] Kaufman, C., ‘‘DASS: Distributed Authentication Security Service’’, RFC 1507, September 1993.

AUTHOR’S ADDRESS

Mark Heroux
Digital Equipment Corporation
550 King Street, MS LKG2-2/Z7
Littleton, MA 01460
USA

Internet email: marcoh@lkg.mts.dec.com
Telephone: +1-508-486-7461