Open Software Foundation
Request For Comments: 94.0
April 1996

Mark Heroux (DEC)

A PRIVATE KEY STORAGE SERVER FOR DCE — FUNCTIONAL SPECIFICATION

1. INTRODUCTION

NOTE: This document currently contains a number of <<TBD>>'s. Those will be completed in a subsequent revision.

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 the private key component of 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) Management and auditing capabilities.

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

1.1. Changes Since Last Publication

This is the first revision of this document.

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 the private key component of a user's asymmetric key pair.

(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’’.

(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’’.

(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 the 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 the 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’’.

(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 of an asymmetric key pair.

(d) `Ko` — The public key 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` — Public key of Private Key Store Server; similarly `Kxp`, `Kou`, `Kxu`.

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

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

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

(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`.

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. These are based on the Augmented Encrypted Key Exchange (AEKE) devised by Bellovin and Merritt [BELL 93].

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.

(c) The PKSS will construct and send rejection messages (<<TBD>>) to the client if security checks fail while processing requests for service.

6.1. User Record Format

The user record is a four-tuple:

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

consisting of seven ‘‘atomic’’ items, as follows:

(a) Clear-text `Username`. This is itself a four-tuple consisting of:

(i) A character string containing the name by which the user is known to DCE.

(ii) 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.

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

(iv) 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.

(b) The private key of the user, `Kxu`, encrypted under `H1`.

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

(d) A copy of the user's public key, `Kou`.

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 in 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 contents of this section (<<TBD>>) will describe how the PKSS database gets its initial load of records at configuration time.

6.3. User Private Key Acquisition Protocol

This is a four-step sequence modeled on [BELL 93].

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

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

```
C-->P: Username:[E(Dc)]H2
```

A client initiates an interaction with the PKSS by sending the PKSS a message containing the `Username` in the clear, and an exponentiated Diffie-Hellman seed value `E(Dc)` encrypted under `H2`, where `H2` is a hash of the user's password.

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

(a) Searches the PKSS database for `Username`. If not found, the PKSS discontinues processing.

(b) Decrypts `[E(Dc)]H2` with the `H2` in the user's PKSS database record to obtain the client's exponentiated Diffie-Hellman seed value `Dc`.

(c) Computes a Diffie-Hellman key `D`, using `E(Dc)`.

(d) Generates a nonce, `Np`.

(e) Encrypts `Np` with `D`, to obtain `[Np]D`.

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

(g) Encrypts a copy of `E(Dp)` with `H2`, to obtain `[E(Dp)]H2`.

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

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

```
P-->C: [E(Dp)]H2:[Np]D
```

The PKSS constructs a challenge two-tuple consisting of: a Diffie-Hellman seed value, `E(Dp)` encrypted under `H2`; and a nonce `Np` encrypted with `D`.

Upon receipt of the challenge message, the client:

(a) Decrypts `[E(Dp)]H2` with `H2`, to obtain `E(Dp)`.

(b) Computes a Diffie-Hellman key, `D`, using `E(Dp)`.

(c) Decrypts `[Np]D` with the Diffie-Hellman key `D`, to obtain the PKSS's nonce, `Np`.

(d) 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]D
```

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`. The client sets the operation ID to a pre-defined value (<<TBD>>) that indicates password acquisition. It encrypts the three-tuple with `D` 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 `D`, 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 `Dp` in the challenge message with the password hash `H2` to compute the Diffie-Hellman key D required to produce `[Nc:Np:I]D`.

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]D
```

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 Diffie-Hellman key `D`.

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 `Dc`, `Dp`, `Nc`, and `Np` are nonces, and that they are the only encrypted values at steps one through three. Note further that `E(Dc)` and `E(Dp)` are not ASN.1-encoded before encryption, thus a known-plaintext attack (that is, a search for ASN.1 identifiers) is not possible. 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`, 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 alternative 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 (<<TBD>>) 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]D
```

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 Diffie-Hellman key `D`, 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]D` with the Diffie-Hellman key `D`, 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 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) Replaces `[Kxu]H1:H2:Kou` in the user record with `[Kxu~]H1~:H2~:Kou~`. 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]D
```

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 (codes <<TBD>>). The PKSS encrypts this message message with the Diffie-Hellman key `D`.

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

The messages at steps five and six are encrypted with the Diffie-Hellman key `D`. 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?}Kxu]D
```

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 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`.

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

(a) Decrypts the message using the Diffie-Hellman key `D`.

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

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~]D
```

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 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. A review of the Diffie-Hellman key exchange protocol follows. The first step is:

```
C-->P: Dc
```

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

Upon receipt of the PKSS management client's `Dc`, the PKSS:

(a) Generates a random number `Xp`.

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

The second step of the Diffie-Hellman protocol is:

```
P-->C: Dp
```

The PKSS sends `Dp` to the PKSS management client. Upon receipt of `Dp`, the PKSS management client:

(a) Computes a Diffie-Hellman key `D = (Dp**Xc) mod g`.

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: [Username:[Kxu]H1:H2:Kou:I]D
```

The PKSS management client constructs the user record four-tuple indicated (recall that the `Username` is itself a four-tuple). It appends an operation ID `I` set to a pre-defined value (<<TBD>>) that indicates ''add a user'' (note that in 6.5.3, the PKSS client will set `I` to a pre-defined value (<<TBD>>) that indicates ''modify a user''.

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 (<<TBD>>).

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: [Username:password:I]D
```

The PKSS management client constructs an add user message three-tuple consisting of: the Username; the new password; and an operation ID `I` set to a pre-defined value (<<TBD>>) that indicates ''add a user'' (note that in 6.5.3, the PKSS client will set `I` to a pre-defined value (<<TBD>>) that indicates ''modify a user''.

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 (<<TBD>>).

6.5.2. PKSS management delete user

This section describes the PKSS management delete user message:

```
C-->P: [Username]D
```

The PKSS management client sends the PKSS a Username four-tuple 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]D
```

The PKSS sends the PKSS client a status code (<<TBD>>) 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 Username four-tuple will be deleted before the new record is created, and the operation ID `I` will be set to a pre-defined value (<<TBD>>) 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) The Diffie-Hellman key established in the first RPC of all protocols.

(b) A state flag which allows the PKSS to track at what step in which protocol it currently is.

(c) A timer ID which allows the PKSS to timeout a long-running request (possibly as a result of a broken connection).

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

<<TBD>>.

8. USER INTERFACES

A management user interface is <<TBD>>.

9. API'S

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

/*
 * Retrieve user's private key for use in [RFC 68.2] login
 *
 * Input
 *
 *    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 key pair.
 *    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 call sec_pkss_key_free() to
 *               release the storage allocated by this function to
 *               contain the encoded private key component.
 *
 *    pub_key_p: Pointer to the encoded 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 storage allocated by this function to
 *               contain the encoded public key component.
 *
 *    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_bellmerr_diffhell
 *              sec_pkss_bellmerr_pkey_get
 *
 */
error_status_t sec_pkss_get_key
(
  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
 *
 *    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 key pair.
 *    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:  a buffer containing the private key component of
 *                  the new asymmetric key pair.
 *    new_pub_key:  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_bellmerr_diffhell
 *              sec_pkss_bellmerr_pkey_get
 *              sec_pkss_chg_key_clt
 */
error_status_t sec_pkss_key_chg_clt
(
  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_pvt_key_t new_pvt_key,
  sec_pk_pub_key_t new_pub_key,
  unsigned32 *new_key_vno_p
);

/*
 * Change user's password and/or asymmetric key pair (server
 * generates new asymmetric key pair).
 *
 * Input
 *
 *    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 key pair.
 *    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.
 *
 * 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 to
 *                    provide the storage to contain the actual key
 *                    version number.
 *
 * Calls:       sec_pkss_bellmerr_diffhell
```

```
 *              sec_pkss_bellmerr_pkey_get
 *              sec_pkss_chg_key_svr
 */
error_status_t sec_pkss_key_chg_svr
(
  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_pub_key_t **new_pub_key_p,
  unsigned32 *new_key_vno_p
);

/*
 * Release asymmetric key storage allocated by other functions
 * in this module.  This function will explicitly zero the contents
 * of the key buffers before releasing the storage.
 *
 * Input
 *
 *    pvt_key_p: Pointer to the encoded private key component of
 *               the user's asymmetric key pair.
 *
 *    pub_key_p: Pointer to the encoded public key component of
 *               the user's asymmetric key pair.
 */
error_status_t sec_pkss_key_free
(
  sec_pk_pvt_key_t *pvt_key_p
  sec_pk_pub_key_t *pub_key_p
);
```

10. REMOTE INTERFACES

A future version of this document will also include the ASN.1 required for the messages.

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

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


/*
 * Structure for PKSS message buffers
 */
typedef struct
{
  unsigned32 num_bytes;
```

```
  [ref, sizeis (num_bytes)]
  byte *msg_buf_p;
} sec_pkss_msg_buf_t;

/*
 * Bellovin-Merritt Diffie-Hellman exchange (steps one and two)
 */

error_status_t sec_pkss_bellmerr_diffhell
(
  [out] sec_pkss_ctx_t
            pkss_ctx;              /* context handle */
  [in]  sec_pkss_msg_buf_t
            pkss_clt_diffhell_buf, /* Username, domain id,
                                      key usages bitmask,
                                      key version number,
                                      [E(Dc)]H2 */
  [out] sec_pkss_msg_buf_t
            pkss_svr_diffhell_buf; /* [E(Dp)]H2:[Np]D */
);

/*
 * Bellovin-Merritt conclusion (steps three and four)
 */

error_status_t sec_pkss_bellmerr_pkey_get
(
  [in,out] sec_pkss_ctx_t
              pkss_ctx;              /* context handle */
  [in]  sec_pkss_msg_buf_t
              pkss_clt_bellmerr_buf, /* [Nc:Np:I]D */
  [out] sec_pkss_msg_buf_t
              pkss_svr_bellmerr_buf; /* [[Kxu]H1:Kou:V:Nc]D */
);

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

error_status_t sec_pkss_chg_key_clt
(
  [in,out] sec_pkss_ctx_t
              pkss_ctx;               /* context handle */
  [in]  sec_pkss_msg_buf_t
              pkss_chg_key_clt_in,
                                      /* [[Kxu~]H1~:H2~:Kou~:[M]Kxu]D */
  [out] sec_pkss_msg_buf_t
              pkss_chg_key_clt_out;  /* [Nc:V:Status]D */
);

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

error_status_t sec_pkss_chg_key_svr
```

```
(
  [in,out] sec_pkss_ctx_t
              pkss_ctx;               /* context handle */
  [in]  sec_pkss_msg_buf_t
              pkss_chg_key_svr_in,
                          /* [H1:H2:newpassword:key-chg?:[M]Kxu]D */
  [out] sec_pkss_msg_buf_t
              pkss_chg_key_svr_out; /* [Nc:V:Status:Kou~]D */
);
```

11. MANAGEMENT INTERFACES

```
/*
 * PKSS management IDL
 */

/*
 * PKSS Diffie-Hellman key exchange
 */

error_status_t sec_pkss_mgmt_diffhell
(
  [out] sec_pkss_ctx_t
           pkss_ctx;                /* context handle */
  [in]  sec_pkss_msg_buf_t
           pkss_mgmt_diffhell_in,   /* [E(Dc)] */
  [out] sec_pkss_msg_buf_t
           pkss_mgmt_diffhell_out;  /* [E(Dp)] */
);

/*
 * PKSS management add user (client generates keys)
 */
error_status_t sec_pkss_mgmt_add_user_clt
(
  [in,out] sec_pkss_ctx_t
              pkss_ctx;                    /* context handle */
  [in]  sec_pkss_msg_buf_t
              pkss_mgmt_add_user_clt_in,
                                /* [Username:[Kxu]H1:H2:Kou:I]D */
  [out] sec_pkss_msg_buf_t
              pkss_mgmt_add_user_clt_out;  /* [V:Status]D */
);

/*
 * PKSS management add user (server generates keys)
 */
error_status_t sec_pkss_mgmt_add_user_svr
(
  [in,out] sec_pkss_ctx_t
              pkss_ctx;                    /* context handle */
  [in]  sec_pkss_msg_buf_t
              pkss_mgmt_add_user_svr_in,
                                      /* [Username:password:I]D */
  [out] sec_pkss_msg_buf_t
```

```
                pkss_mgmt_add_user_svr_out;  /* [V:Kou:Status]D */
);

/*
 * PKSS management delete user
 */
error_status_t sec_pkss_mgmt_del_user
(
  [in,out] sec_pkss_ctx_t
                pkss_ctx;               /* context handle */
  [in]  sec_pkss_msg_buf_t
                pkss_mgmt_del_user_in,  /* [Username]D */
  [out] sec_pkss_msg_buf_t
                pkss_mgmt_del_user_out; /* [Status]D */
);

/*
 * PKSS management modify user (client generates keys)
 */
error_status_t sec_pkss_mgmt_mod_user_clt
(
  [in,out] sec_pkss_ctx_t
                pkss_ctx;               /* context handle */
  [in]  sec_pkss_msg_buf_t
                pkss_mgmt_mod_user_clt_in,
                                /* [Username:[Kxu]H1:H2:Kou:I]D */
  [out] sec_pkss_msg_buf_t
                pkss_mgmt_mod_user_clt_out; /* [V:Status]D */
);

/*
 * PKSS management modify user (server generates keys)
 */
error_status_t sec_pkss_mgmt_mod_user_svr
(
  [in,out] sec_pkss_ctx_t
                pkss_ctx;                   /* context handle */
  [in]  sec_pkss_msg_buf_t
                pkss_mgmt_mod_user_svr_in,
                                      /* [Username:password:I]D */
  [out] sec_pkss_msg_buf_t
                pkss_mgmt_mod_user_svr_out; /* [V:Kou:Status]D */
);

/*
 * PKSS management client API prototypes
 */

/*
 * Add a new user (client generates asymmetric key pair).
 *
 * Input
 *
 *    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 key pair.
 *    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:  a buffer containing the private key component of
 *                  the new asymmetric key pair.
 *    pub_key:  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
 */
error_status_t sec_pkss_mgmt_user_add_clt
(
  const char *name_p,
  sec_pk_domain_t domain_id,
  sec_pk_usage_flags_t key_usages,
  unsigned32 key_vno,
  sec_pk_pvt_key_t pvt_key,
  sec_pk_pub_key_t pub_key,
  const char *pwd_p,
  unsigned32 *new_key_vno_p
);

/*
 * Add a new user (server generates asymmetric key pair).
 *
 * Input
 *
 *    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 key pair.
 *    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.
 *
```

```
 * 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 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
 *                    to provide the storage to contain the actual
 *                    key version number.
 *
 * Calls:       sec_pkss_mgmt_diffhell
 *              sec_pkss_mgmt_add_user_svr
 */
error_status_t sec_pkss_mgmt_user_add_svr
(
  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_pub_key_t **new_pub_key_p,
  unsigned32 *new_key_vno_p
);

/*
 * Delete a user
 *
 * Input
 *
 *    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 key pair.
 *    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
 */
error_status_t sec_pkss_mgmt_user_del
(
  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_pub_key_t *new_pub_key_p,
  unsigned32 *new_key_vno_p
);

/*
 * Modify a user record (client generates asymmetric key pair).
 *
 * Input
 *
 *    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 key pair.
 *    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:  a buffer containing the private key component of
 *                  the new asymmetric key pair.
 *    pub_key:  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
 */
error_status_t sec_pkss_mgmt_user_mod_clt
(
  const char *name_p,
  sec_pk_domain_t domain_id,
  sec_pk_usage_flags_t key_usages,
  unsigned32 key_vno,
  sec_pk_pvt_key_t pvt_key,
  sec_pk_pub_key_t pub_key,
  const char *pwd_p,
  unsigned32 *new_key_vno_p
);

/*
 * Modify a user record (server generates asymmetric key pair).
 *
 * Input
 *
 *    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 key pair.
 *    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.
 *
 * 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 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
 *                    to provide the storage to contain the actual
 *                    key version number.
 *
 * Calls:       sec_pkss_mgmt_diffhell
 *              sec_pkss_mgmt_mod_user_svr
 */
error_status_t sec_pkss_mgmt_user_mod_svr
(
  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_pub_key_t  **new_pub_key_p,
  unsigned32 *new_key_vno_p
);
```

12. RESTRICTIONS AND LIMITATIONS

13. OTHER COMPONENT DEPENDENCIES

14. COMPATIBILITY

15. STANDARDS

16. OPEN ISSUES

REFERENCES

[BELL 93] Bellovin, S. & Merritt, M., ‘‘Augmented Encrypted Key Exchange’’, Proceedings of the First ACM Conference on Computer and Communications Security, November, 1993.

[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