Open Software Foundation
Request For Comments: 68.3
January 1997

A. Anderson (HP)
S. Cuti (HP)

DCE 1.2.2 PUBLIC KEY LOGIN — FUNCTIONAL SPECIFICATION

1. INTRODUCTION

This document specifies the functionality required to integrate public key mechanisms into DCE login, that is, into the initial DCE Kerberos Ticket-Granting Ticket protocol. This specification obsoletes [RFC 68.2].

The goal of this effort is to allow users to use a private key rather than a shared-secret password to prove their identity to the DCE Key Distribution Center (KDC) (a.k.a. Key Distribution Server, KDS).

The immediate benefit is that, in the event of a compromise of the KDC, public key users do not have any identifying information exposed to the intruder. If the KDC is compromised, all user secret keys will be revealed to the intruder. This means they become worthless as a proof of identity, and therefore the cell administrator must reissue passwords to all such users before they can be allowed to log-in to the cell. Under the design described in this RFC, public key users prove their identity by knowledge of a private key that is never known to the KDC, and therefore a compromise of the KDC cannot reveal these keys.

The authentication information and protocol are based on the PK-INIT Kerberos protocol [DRAFT-PKINIT] where the user's private key is stored locally.

A DCE Personal Security Module Interface is provided to abstract and hide underlying details of public key algorithm implementations, interfaces, and information storage mechanisms.

A DCE Public Key Module API is provided to abstract and hide the underlying details of KDC and client public key acquisition for use in the Public Key Login protocol.

A DCE Private Key Module Interface is provided to abstract and hide the underlying details of the private key storage mechanism.

There are no changes to existing login APIs (`sec_login`) other than for the addition of some new error status values, and no need for changes to existing login utilities or user interfaces.

1.1. Changes Since Last Publication

Changes since [RFC 68.2]:

(a) The public key login protocol was previously based on [ITU X.509] as described in [RFC 85.0]. The protocol is now one of the protocols specified in [DRAFT-PKINIT].

(b) Support for public key generation has been added.

(c) `dcecp` support for creating or modifying accounts to use the public key login protocol has been added.

(d) Two new interfaces, `sec_pubkey` and `sec_pvtkey`, have been specified.

(e) The `DCEPKModulusLength` ERA has been removed. The modulus length for key-pair generation is now specified as part of the `dcecp` command.

(f) The `DCEPKPrivateKeyStorage` ERA has been added. This ERA is used to identify the private key storage mechanism used by each principal.

(g) The format of public keys as stored in the `DCEPKKeyEncipherment` and `DCEPKAuthentication` ERAs has changed. The keys are now stored as IDL-encoded structures of type `sec_passwd_rec_t`, with `key` of type `sec_passwd_pubkey`. This allows a version number to be associated with each public key.

2. TARGET

This technology is provided for customers who require less reliance on the physical security of DCE Security servers. It may also be of interest to customers moving toward public key based security mechanisms such as smart cards, or public key based ''Single Sign On'' facilities, although this technology is only a starting point for such enhancements, which are not part of this deliverable.

3. GOALS AND NON-GOALS

3.1. Goals

(a) Allow users to use a private key rather than a password to prove their identity to the DCE Key Distribution Center.

(b) Provide a standards-based mutual authentication protocol between the user and the DCE Key Distribution Center.

(c) The protocol must not require private keys to be stored in the DCE Registry or to be transmitted across the wire protected by a password-derived key.

(d) Ease recovery from a compromise of the DCE Key Distribution Center.

(e) Allow for use of public key algorithms that need not be RSA (although the reference implementation uses the RSA BSAFE library).

(f) Allow for integration with public key formats that need not be RSA-format keys (although the reference implementation uses the RSA BSAFE library).

3.2. Non-Goals

(a) It is not a goal to support public key certificates in the reference implementation, or to provide integration with the DCE Certification API [RFC 80.0], or to assume the existence of a Certification Authority. The wire protocol is designed to allow integration of such support in the future.[1]

In the reference implementation, public keys are obtained by the KDC from the DCE Registry. If a client provides a certificate as part of the authentication protocol, the KDC ignores the certificate gracefully.

(b) Server principals that use public key authentication to the KDC must use the OSF DCE 1.2.2 User-to-User Protocol for authenticated communication with DCE clients. This is because non-User-to-User Kerberos Service Tickets continue to be DES-encrypted under the DES key of the server. Extending public key technology into this context is beyond the scope of this project.

(c) It is not a goal of this project to integrate or test the DCE Private Key Storage Server (see [RFC 94.0]) with the functionality provided by this project. Such integration and testing is expected to be done by the provider of the DCE Private Key Storage Server.

(d) There is no specified or supported policy control for public keys (such as control of lifetime, expiration, or length) other than as specified in ''Public Key Policy Control'' under ''FUNCTIONAL DEFINITION''.

4. TERMINOLOGY

The same terminology and notation used in [RFC 85.0] is carried over here, with a few additions:

(a) **PSM** — Personal Security Module; a layer defined to abstract knowledge of public and private key storage mechanisms and public key cryptosystem operations and data types.

(b) **ERA** — OSF DCE 1.1 Extended Registry Attribute. See [RFC 6.0].

---

1. Although it is expected that DCE will support public key certificates eventually, the format of those certificates is still undetermined.

(c) **ASN.1** — Abstract Syntax Notation 1. A notation defined in [ITU X.208] for describing abstract types and values.

(d) **BER** — Basic Encoding Rules. A set of rules defined in [ITU X.209] and used to encode ASN.1 values as strings of octets. A single value can have multiple valid BER encodings.

(e) **DER** — Distinguished Encoding Rules. A restricted form of BER defined in [ITU X.509] to eliminate most of the ambiguities in BER.

(f) **Smart Card** — A multi-purpose, tamper-resistant, portable personal security device, utilizing VLSI chip technology for information storage and processing.

(g) **User** — The human user (and any associated private key storage).

(h) **Client** — An application running on the user's workstation. The login process is an example of a client.

(i) `Cx` — Certificate of X.

(j) `K` — Symmetric (session) key.

(k) `Ktgt` — The session key to be used with a TGT.

(l) `{M}Px` — Message M encrypted with X's public key.

(m) `{M}Sx` — Message M signed by X.

(n) `Nx` — A nonce (e.g., random number) generated by X.

(o) `Tx` — A time stamp generated by X.

(p) **KDC** — The Kerberos Key Distribution Center.[2]

(q) **TGT** — A Kerberos Ticket Granting Ticket.

5. REQUIREMENTS

The technology must support an increase to the overall security of a DCE cell. It must also represent a genuine integration of public key technology with the DCE login process.

6. FUNCTIONAL DEFINITION

6.1. TGT Acquisition Protocol

The DCE Public Key TGT acquisition protocol is a subset of the protocol described in [DRAFT-PKINIT], using the option for user's private key being stored locally.

> *NOTE: The protocol defined by [DRAFT-PKINIT] is very new, and is a candidate for entry onto the IETF standards track. Before it becomes standardized, it may change in an incompatible and non-interoperable way. It is the intent of the authors that the DCE implementation track this protocol as it evolves; however, resource and schedule constraints have prevented us from implementing the full protocol in DCE 1.2.2. Future versions of DCE are expected to change to follow the successors to [DRAFT-PKINIT] as well as maintain interoperability with DCE 1.2.2; however, in the interests of greater interoperability, compliance with [DRAFT-PKINIT] will have priority.*

The DCE login APIs (`sec_login_validate_identity()`, `sec_login_valid_and_cert_ident()`, and `sec_login_validate_first()`) attempt to use this protocol initially by default as long as Public Key authentication information can be constructed. If Public Key authentication information can not be constructed, then the default for the initial attempt is the OSF DCE Third Party protocol. If OSF DCE Third Party

---

2. No distinction is made here between the Authentication Service and the Ticket Granting Service (subservices of the KDC), for reasons of clarity.

authentication information can not be constructed, then the default for the initial attempt is the Timestamps protocol (for which information can always be constructed).

> *NOTE: In the reference implementation, for backwards compatibility with pre-1.2.2 servers, both the* `KRB5_PADATA_ENC_UNIX_TIME` *and* `KRB5_PADATA_ENC_TIMESTAMP` *PADATA are constructed.*

If the KDC is unable to authenticate the user with the supplied public key pre-authentication data, the KDC returns error information.

If the initial public key login attempt fails, then the `sec_login` code falls back to the existing symmetric key password-based authentication, unless the KDC error information indicates that the principal is required to use public key pre-authentication. Sites that do not wish to allow any fall-back must attach an instance of the `pre_auth_req` ERA, with a value of `3` (`PADATA-ENC-PUBLIC-KEY`), to each principal that is required to use public key login. With this ERA attached, the KDC will not accept anything except a public key login for the principal, even if attempted.

A two-message protocol is used to acquire a TGT. This protocol relies, in part, on time stamps to guarantee the freshness of messages. There is no reason to adopt a challenge-response mechanism since the subsequent Kerberos protocols rely on time stamps. Since the TGT session key is encrypted with a random key that is encrypted with the public key of the client, successful use of the TGT implies the ability to decrypt this session key, and therefore possession of the user's private key.

The authentication information is transmitted in the pre-authentication data fields of the standard Kerberos V5 `KRB_AS_REQ` and `KRB_AS_REP` messages [IETF 1510] as new `KRB5_PADATA_ROOT_CERT` and `KRB5_PADATA_PUBLIC_REP` pre-authentication data types.

> *NOTE: As an implementation optimization and for backwards compatibility with pre-1.2.2 servers, the client sends both Third-Party (PADATA-ENC-OSF-DCE) and Public Key (PADATA-ENC-ROOT-CERT) PADATA in the initial TGT request. The Third-Party PADATA is the first PADATA stored in the request. Pre-1.2.2 servers examine and verify the first PADATA, and ignore any remaining PADATA. DCE 1.2.2 servers examine and verify each PADATA type. If the Third-Party PADATA can not be verified, but the Public Key PADATA can, then the KDC returns a TGT to the client using the Public Key reply protocol.*

The protocol usage criteria can be diagrammed as follows.

The ''TP can be built'' column indicates whether a Third-Party PADATA structure can be built by the `sec_login` client code.

The ''PK can be built'' column indicates whether Public Key Protocol information can be built by the `sec_login` client code. This can be built only if the client has a `PSM` and if the supplied passphrase is valid for gaining access to that `PSM`.

The ''PADATA sent'' column indicates which PADATA types are sent in the `KRB_AS_REQ`, and in what order.

The ''PADATA verified'' column indicates which PADATA type must pass verification in order for a TGT to be returned and which protocol will be used for the PADATA in the `KRB_AS_REP`. If there is no possibility of a TGT to be returned, the column indicates ''none''.

TABLE 1: PROTOCOL USAGE CRITERIA

VERSIONS		CASES			PROTOCOLS USED	
client version	server version	TP can be built	PK can be built	password valid	PADATA sent+	PADATA verified+
1.2.2	1.2.2	yes	yes	yes	TP,PK	PK
1.2.2	1.2.2	yes	yes	no	TP,PK	PK
1.2.2	1.2.2	yes	no	yes	TP	TP*
1.2.2	1.2.2	yes	no	no	TP	none
1.2.2	1.2.2	no	yes	yes	TS,PK	PK
1.2.2	1.2.2	no	yes	no	TS,PK	PK
1.2.2	1.2.2	no	no	yes	TS	TS*
1.2.2	1.2.2	no	no	no	TS	none
1.2.2	<1.2.2	yes	yes	yes	TP,PK	TP
1.2.2	<1.2.2	yes	yes	no	TP,PK	none
1.2.2	<1.2.2	yes	no	yes	TP	TP
1.2.2	<1.2.2	yes	no	no	TP	none
1.2.2	<1.2.2	no	yes	yes	TS,PK	TS*
1.2.2	<1.2.2	no	yes	no	TS,PK	none
1.2.2	<1.2.2	no	no	yes	TS	TS*
1.2.2	<1.2.2	no	no	no	TS	none
<1.2.2	1.2.2	yes	N/A	yes	TP	TP*
<1.2.2	1.2.2	yes	N/A	no	TP	none
<1.2.2	1.2.2	no	N/A	yes	TS	TS*
<1.2.2	1.2.2	no	N/A	no	TS	none

```
* PADATA passes verification only if the client's effective
  pre_auth_req value allows the client to use this PADATA type.

+ TS: Timestamps PADATA
      (KRB5_PADATA_ENC_UNIX_TIME from pre-1.2.2 clients,
       KRB5_PADATA_ENC_UNIX_TIME followed by
          KRB5_PADATA_ENC_TIMESTAMP from 1.2.2 clients)
  TP: Third-Party PADATA (KRB5_PADATA_ENC_OSF_DCE)
  PK: Public Key PADATA
      (KRB5_PADATA_ENC_ROOT_CERT,
       KRB5_PADATA_ENC_PUBLIC_REP)
```

6.1.1. Client-to-KDC Message

NOTE: The following protocol description is necessarily a high-level simplification of the actual protocol used. For full details, see [DRAFT-PKINIT].

```
C-->KDC:   {Rc,Cksum,Tc,Nc,KDC}Sc
```

The client process signs a message including an optional list of root certifiers trusted by the client, a checksum, a time stamp, a nonce and the identity of the KDC. The signature is done with the client's private digital signature key. This signed message is sent to the KDC along with the client's (optional) certificate as the contents of the PADATA field of a standard `KRB_AS_REQ` message. The client's identity is part of the existing `KRB_AS_REQ` message.

> *NOTE: In the reference implementation, client processes do not send a certificate, and the list of root certifiers trusted by the client is NULL.*

If the client supplies a certificate, the KDC may choose to ignore the certificate and retrieve public keys directly from the DCE Registry. Alternatively, the KDC may check revocation lists to establish the validity of the certificate.

> *NOTE: In the reference implementation, the KDC ignores any certificates sent by client processes and obtains public keys directly from the DCE Registry.*

6.1.2. KDC-to-Client Message

> *NOTE: The following protocol description is necessarily a high-level simplification of the actual protocol used. For full details, see [DRAFT-PKINIT].*

```
KDC-->C:   Ckdc,{{Kreply,Nc}Skdc}EncReplytmpKey,
           {EncReplytmpKey}Pc
```

The KDC checks the time stamp and signature of the client's message. If the time stamp is sufficiently current and the signature can be verified using the client's public key, the KDC responds with a standard `KRB_AS_REP` reply message in which the PADATA field contains a random symmetric reply key (`Kreply`) and the client's nonce. The reply key and client nonce are first signed using the KDC's private digital signature key, then encrypted using a temporary random symmetric key (`EncReplytmpKey`). This temporary random symmetric key is encrypted with the client's public key-encipherment key [the combination of symmetrically encrypted signed data and asymmetrically encrypted key is called *digital enveloping*]. The reply key is used to encrypt the encrypted portion of the standard `KRB_AS_REP`, which includes the symmetric session key associated with the TGT. The KDC optionally includes its certificate in the PADATA field of the response. This certificate, if included, must be signed by one of the root certifiers trusted by the client.

The TGT is passed in the standard `KRB_AS_REP` ticket field. The TGT is returned without additional encryption (portions of it were encrypted by the KDC) since it is subsequently used in the clear by the client. The symmetric session key used in association with the TGT is returned in the standard `EncKDCRepPart` field of the `KRB_AS_REP` message. This `EncKDCRepPart` field is encrypted using the reply key (`Kreply`) returned in the signed and encrypted authentication data from the KDC.

> *NOTE: In the reference implementation, the KDC does not send its certificate in the response. The public key of the KDC is cached locally on the client node by the* `dced` *daemon, and can be read from the cache by the client.*
>
> *Vendors implementing smart cards may wish to store the public key of the KDC on each user's smart card, and have the client obtain the KDC public key from the smart card.*

By checking the signature on this response, the client can be assured that the reply is from the KDC. The session key can only be decrypted by the legitimate client who possesses the private key needed to decrypt the key encryption key. The TGT and associated session key are then used as normal.

6.1.3. Changes to existing TGT acquisition protocols

When any TGT acquisition request is received, the DCE KDC checks for an instance of the `pre_auth_req` ERA attached to the user principal. If found, the authentication attempt is rejected with the `KDC_PREAUTH_FAILED` error if the value of the ERA is less than the value associated with the protocol used by the TGT acquisition request. The public key login protocol has a higher associated value than any previously specified preauthentication protocol, including the various forms of encrypted timestamps and the DCE 1.1 third-party protocol.

6.2. Public Key Interfaces

6.2.1. Overview

Public Key Login provides a number of layers and APIs. Some of these are provided to help satisfy export requirements. Some are provided to allow easy extension or enhancement by vendors. Only one API — `sec_pubkey`

— is provided for user application use.

The Public Key Login layers and APIs may be diagrammed as follows as they are used internally by the DCE Security client library.

```
        TABLE 2: PUBLIC KEY LOGIN LAYERS

 ___________________________________________________________
|                                                           |
|                         sec_login                         |
|___________________________________________________________|
|                                                           |
|                          sec_psm                          |
|___________________________________________________________|
|                   |                 |                     |
|  sec_pubkey       |  sec_pvtkey     |  crypto library     |
|___________________|_________________|_____________________|
|                   |        |        |
|  rsec_pk_cache    | PKSS   | file   |
|___________________|________|________|
```

Descriptions of the individual interfaces follow.

6.2.2. DCE Personal Security Module Interface

A `sec_psm` layer is defined to abstract knowledge of public and private storage mechanisms and public key cryptosystem functions and data types. This interface is not specified as an application program interface, but rather as a way to allow replacement with a standard personal security module interface as one emerges.

This layer provides access to high-level security operations such as `sign()`, `verify()`, `encrypt()`, and `decrypt()`, hiding the actual implementations in lower layers. The existing `sec_login` API has been made a consumer of this interface.

Only the user principal has the permissions needed to obtain information from this layer, as access to the user's private key requires knowledge of the user's password.

6.2.3. Public Key Module API

A DCE Public Key Module API — `sec_pubkey` — is provided to abstract and hide the details of public key storage and acquisition for use in the Public Key Login protocol. This mechanism is extensible to other storage and acquisition mechanisms and for use by other protocols. This API may be used directly by DCE Applications. It is also used internal to the DCE Public Key protocol implementation.

The mechanisms provided with the reference implementation are:

(a) `sec_pk_domain_dce_general`: retrieval of a principal's public keys from ERAs attached to the principal in the DCE Registry. This mechanism may be used for retrieval by any client for any principal. This mechanism may also be used for public key storage by a cell administrator, or other authorized client.

(b) `sec_pk_domain_kdc_cache`: retrieval of a principal's public keys from a local cache file, backed up by retrieval via request to `dced`. This mechanism may be used only for retrieval of a krbtgt principal's key. It is intended for use by unauthenticated clients as part of the Public Key Login protocol.

(c) `sec_pk_domain_kdc_refresh_refresh`: retrieval of a principal's public keys via request to `dced`. This mechanism may be used only for retrieval of a krbtgt principal's key. This mechanism causes the `dced` daemon to refresh the associated krbtgt principal's local public key cache file.

(d) `sec_pk_domain_kdc_pk_init`: retrieval of a principal's keys directly from the DCE Registry. This mechanism is available only internal to the DCE Security Server.

6.2.4. Private Key Module Interface

A DCE Private Key Module Interface is provided to abstract and hide the underlying details of the private key storage mechanism. This interface is extensible to other acquisition mechanisms and for use by other protocols. The DCE Private Key Module Interface is not provided as an application API, but rather as a means for DCE vendors to supply alternative key storage mechanisms.

The storage mechanisms provided with the reference implementation are:

(a) `sec_psm_pkss_mechanism`: storage using the Public Key Storage Server [RFC 94.0].

(b) `sec_psm_file_mechanism`: storage using a local file.

(c) `sec_psm_kdc_pk_init_mechanism`: storage in the DCE Registry. This mechanism is available only internal to the DCE Security Server.

6.3. KDC Key Management

The KDC requires a public key pair in order to support the OSF DCE Public Key Login protocol. While the KDC uses the `sec_psm` layer for accessing public key functions, it uses a `sec_pvtkey` mechanism that accesses the DCE Registry directly, both for security and for performance. This mechanism is only available within the `secd` binary image.

6.3.1. KDC Private and Public Key

The KDC private key used for authentication (generating signatures) is stored in the DCE Registry. It is stored, protected, and propagated exactly as the DES symmetric KDC key is now stored, protected, and propagated, by the implementation of a new password type: `sec_passwd_pubkey`, and by the extension of internal mechanisms for storage of keys. This new type is used within the DCE Registry, for propagation to DCE Registry replicas, and for the encoding of public key values inside ERAs. This new type may not be specified by Registry clients.

The KDC public key used for authentication is stored in the `DCEPKAuthentication` ERA attached to the `krbtgt` principal for the cell.

The DCE Registry sets or updates the KDC authentication public key key-pair when the password of the krbtgt principal is set to a keytype of `sec_passwd_genprivkey`. It uses the `modulus_size` parameter in the keytype to determine what modulus size to use.

The `dcecp> account modify krbtgt/<cell> -pkgenprivkey <modulus_size>` user interface can be used by a cell administrator to set the KDC's key to a keytype of `sec_passwd_genprivkey`.

6.3.2. KDC Public Key Caching

During login, a principal needs an authenticated copy of the public key of the local KDC in order to verify that the TGT it obtains is from the correct KDC. However, the principal is not yet authenticated, and so is unable to obtain an authenticated copy of an ERA. Since the `dced` daemon is authenticated (as the client host principal), we make use of this to obtain an authenticated copy of the KDC public key.

The daemon, via the `rsec_pk_cache_kdc_key` remote interface, supports client requests for updates or additions to a local cache of KDC public authentication keys. It retrieves the authentication public key of a cell's KDC from the `DCEPKAuthentication` ERA attached to the `krbtgt` principal for the cell, and writes it to the file `/etc/opt/dce/security/kdc_pk_auth/`*`cell_name`*`/.key` in ASN.1 DER encoding.

The `sec_login` APIs retrieve required public keys of local or foreign KDCs from the cache file for the cell. If a required KDC key file is not found, or if a KDC key fails to decrypt the reply from the KDC successfully, `sec_login` asks the local `dced` daemon to obtain or refresh the required key.

6.4. Public Key Policy Control

The modulus length of the KDC authentication public key is determined by the value of the `-pkgenprivkey` parameter to the `dcecp> account modify krbtgt/<cell>` command. By default, only the cell administrator is authorized to modify this account.

Public keys may be revoked by removing the DCEPKKeyEncipherment and DCEPKAuthentication ERA instances from the principal.

Usage of the public key login protocol can be mandated by attaching a pre_auth_req ERA with a value of PADATA_ENC_PUBLIC_KEY to a principal. See ‘‘Changes to existing TGT acquisition protocols’’ under the ‘‘TGT Acquisition Protocol’’ sub-section under ‘‘FUNCTIONAL DEFINITION’’, and ‘‘Configuring Public Key Login Users’’ under ‘‘MANAGEMENT INTERFACES’’.

Except as stated above, there is no specified or supported policy control for public keys (such as control of lifetime, expiration, or modulus length) included as part of this project.

6.5. Passwords

During login operations, including dce_login and dcecp> login, the string entered as the password value is used first as a passphrase in an attempt to access the PSM. If this fails, and the user is not identified as requiring use of the public key authentication protocol, then the string is used as a DCE shared-secret password.

Except for login operations, the dcecp -password option always refers to a user's DCE shared-secret password, while the oldpassphrase and newpassphrase options always refer to a user's PSM passphrase.

A user's PSM passphrase values may or may not match the DCE shared-secret password value.

7. DATA STRUCTURES

7.1. Public Key Login Base Types

The common base definitions for the DCE Public Key Login APIs are defined in three new files security/idl/sec_pk_base.idl, security/h/sec_pk.h, and security/h/sec_psm_base.h, listed here:

7.1.1. security/idl/sec_pk_base.idl

```
/*
 * @OSF_COPYRIGHT@
 * COPYRIGHT NOTICE
 * Copyright (c) 1990, 1991, 1992, 1993, 1996 Open Software
 *  Foundation, Inc.
 * ALL RIGHTS RESERVED (DCE).  See the file named COPYRIGHT.DCE
 *  in the src directory for the full copyright text.
 */
/*
** Copyright (c) Hewlett-Packard Company 1996
** Unpublished work.  All Rights Reserved.
**
**       Public Key Base Definitions
*/

interface sec_pk_base {

/* s e c _ p k _ d o m a i n _ t
 *
 * A UUID associated with the application domain
 * in which a public or private key is used.
 */
typedef uuid_t sec_pk_domain_t;

/* s e c _ p k _ u s a g e _ f l a g s _ t
 *
 * A set of key usage flags indicating the uses for
```

```
 * a key or key-pair.
 *
 * These correspond to KeyUsage types defined in
 * DAM 1 (Dec 1995) to X.509 (1993)
 */
typedef unsigned32 sec_pk_usage_flags_t;

const unsigned32 sec_pk_usage_digitalSignature  = 0x1;
const unsigned32 sec_pk_usage_nonRepudiation    = 0x2;
const unsigned32 sec_pk_usage_keyEncipherment   = 0x4;
const unsigned32 sec_pk_usage_dataEncipherment  = 0x8;
const unsigned32 sec_pk_usage_keyAgreement      = 0x10;
const unsigned32 sec_pk_usage_keyCertSign       = 0x20;
const unsigned32 sec_pk_usage_offLineCRLSign    = 0x40;

const unsigned32 MAX_USAGES                     = 7;

/* s e c _ p k _ d a t a _ t
 *
 * A structure pointing to an X.509 or X.511
 * ASN.1 DER-encoded value.
 *
 * Rather than using this structure directly, users
 * should use one of the types that follow, which
 * indicate the type of information contained in the
 * particular structure instance.
 */
typedef struct{
    unsigned32       len;
    [ptr, size_is(len)]
        byte         *data;
} sec_pk_data_t;



/* s e c _ p k _ d a t a _ p _ t
 *
 * A pointer to a sec_pk_data_t
 * structure
 */
typedef [ptr] sec_pk_data_t   *sec_pk_data_p_t;

/* s e c _ p k _ g e n _ d a t a _ t
 *
 * A sec_pk_data_t structure containing len and data which are
 * parallel to sec_pk_data_t structure to hold
 * non-ASN.1 DER-encoded data, such as plaintext or ciphertext.
 */
typedef sec_pk_data_t sec_pk_gen_data_t;

/* s e c _ p k _ p u b k e y _ t
 *
 * A sec_pk_data_t structure containing an X.509
 * ASN.1 DER-encoded value of type
 * SubjectPublicKeyInfo
```

```
 *
 * The vendor's public key infrastructure is assumed
 * to provide functions for generating a public key
 * in this format.
 *
 * In the reference implementation, the BSAFE library
 * provides such a function.
 */
typedef sec_pk_data_t sec_pk_pubkey_t;

/* s e c _ p k _ p u b k e y _ p _ t
 *
 * A pointer to a sec_pk_pubkey_t structure
 */
typedef [ptr] sec_pk_pubkey_t *sec_pk_pubkey_p_t;


/* s e c _ p k _ p v t k e y _ t
 *
 * A sec_pk_data_t structure containing an ASN.1
 * DER-encoded private key value.  The
 * format will depend on the public key infrastructure.
 *
 * In the reference implementation, the structure will
 * contain a PKCS#8 private key of type PrivateKeyInfo.
 *
 * The vendor's public key infrastructure is assumed
 * to provide functions for generating a private key
 * in this format.
 */
typedef sec_pk_data_t sec_pk_pvtkey_t;

/* s e c _ p k _ p v t k e y _ p _ t
 *
 * A pointer to a sec_pk_pvtkey_t structure
 */
typedef [ptr] sec_pk_pvtkey_t *sec_pk_pvtkey_p_t;


/* s e c _ p k _ s i g n e d _ t
 *
 * A sec_pk_data_t structure containing an X.509
 * ASN.1 DER-encoded value of type
 * SIGNED.
 *
 * The vendor's public key infrastructure is assumed
 * to provide functions for generating signed data
 * in this format.
 *
 * In the reference implementation, the BSAFE library
 * provides such a function.
 */
typedef sec_pk_data_t sec_pk_signed_t;

/* s e c _ p k _ s i g n e d _ p _ t
```

```
 *
 * A pointer to a sec_pk_signed_t structure
 */
typedef [ptr] sec_pk_signed_t *sec_pk_signed_p_t;


/* s e c _ p k _ e n c r y p t e d _ t
 *
 * A sec_pk_data_t structure containing an X.509
 * ASN.1 DER-encoded value of type
 * ENCRYPTED.
 *
 * The vendor's public key infrastructure is assumed
 * to provide functions for generating encrypted data
 * in this format.
 *
 * In the reference implementation, the BSAFE library
 * provides such a function.
 */
typedef sec_pk_data_t sec_pk_encrypted_t;

/* s e c _ p k _ e n c r y p t e d _ p _ t
 *
 * A pointer to a sec_pk_encrypted_t structure
 */
typedef [ptr] sec_pk_encrypted_t *sec_pk_encrypted_p_t;

/* s e c _ p k _ a l g o r i t h m _ i d _ t
 *
 * A sec_pk_data_t structure containing an X.509
 * ASN.1 DER-encoded value of type
 * AlgorithmIdentifier.
 *
 * The vendor's public key infrastructure is assumed
 * to provide functions for providing
 * this format.
 *
 * In the reference implementation, the BSAFE library
 * provides such a function.
 */
typedef sec_pk_data_t sec_pk_algorithm_id_t;

/* s e c _ p k _ a l g o r i t h m _ i d _ p _ t
 *
 * A pointer to a sec_pk_algorithm_id_t structure
 */
typedef [ptr] sec_pk_algorithm_id_t *sec_pk_algorithm_id_p_t;
}
```

7.1.2. security/h/sec_pk.h

```
/*
 * @OSF_COPYRIGHT@
 * COPYRIGHT NOTICE
 * Copyright (c) 1996 Open Software Foundation, Inc.
 * ALL RIGHTS RESERVED (DCE).  See the file named COPYRIGHT.DCE
```

```
 * in the src directory for the full copyright text.
 */
/*
 * Copyright (c) Hewlett-Packard Company 1996
 * Unpublished work. All Rights Reserved.
 */

#ifndef _SEC_PK_H_
#define _SEC_PK_H_

#include <dce/nbase.h>
#include <dce/sec_pk_base.h>
#include <dce/sec_psm_base.h>
#include <dce/sec_attr_base.h>

/* The following sec_pk_domain_t UUIDs are
 * architectural.  They cannot be changed without
 * destroying interoperability.
 */

/* sec_pk_domain_dce_general: for use by
 * authenticated clients in retrieving and storing
 * public and private keys */
extern uuid_t sec_pk_domain_dce_general;
              /* ae991638-5cbe-11cf-a22d-08000919ebb5 */

/* sec_pk_domain_kdc_cache: for use by
 * unauthenticated clients in retrieving the
 * public key of a krbtgt principal from the
 * dced-maintained krbtgt key cache.
 */
extern uuid_t sec_pk_domain_kdc_cache;
              /* e925f8b4-0a3a-11d0-94f1-08000919ebb5 */

/* sec_pk_domain_pk_kdc_cache_refresh: for use by
 * unauthenticated clients in retrieving a
 * refreshed cache copy of the public key of a
 * krbtgt principal.
 */
extern uuid_t sec_pk_domain_kdc_cache_refresh;
              /* f087b8e0-0a3a-11d0-9113-08000919ebb5 */

/* sec_pk_domain_kdc_pk_init: for use by the KDC
 * in performing sec_psm operations using its own
 * private key.
 */
extern uuid_t sec_pk_domain_kdc_pk_init;
              /* 7d2d9810-e721-11cf-9021-08000919ebb5 */

/* UUIDs of DCE public key-related ERAs
 *
 */

/* DCEPKAuthentication_id: UUID of the
 * DCEPKAuthentication ERA, used to hold a
```

```
 * principal's sec_pk_usage_digitalSignature
 * Public Key for use in the DCE Public Key Login
 * protocol
 */
extern uuid_t DCEPKAuthentication_id;
              /* d44dc60e-6230-11cf-98ed-08000919ebb5 */

/* DCEPKKeyEncipherment_id: UUID of the
 * DCEPKKeyEncipherment ERA, used to hold a
 * principal's sec_pk_usage_keyEncipherment Public
 * Key for use in the DCE Public Key Login
 * protocol
 */
extern uuid_t DCEPKKeyEncipherment_id;
              /* 76251f8c-6230-11cf-af89-08000919ebb5 */

/* DCEPKPrivateKeyStorage_id: UUID of the
 * DCEPKPrivateKeyStorage ERA, used to associate
 * a principal with a private key storage mechanism.
 * The UUIDs of the mechanisms themselves are contained in
 * file sec_psm_base.h
 */
extern uuid_t DCEPKPrivateKeyStorage_id;
              /* 1f79c78e-b026-11cf-b7c8-0800090a5254 */

/* Following format is used in the header of a public
 * or private key keyfile.  A PK keyfile is used both
 * by the dced in caching the public keys of krbtgt
 * principals, and also by dcecp in reading public
 * or private key values from a file supplied by
 * the user.
 *
 * The values used in the header are defined in
 * sec_pk.c
 */
typedef struct {
    char         keyfile_id[4];
    unsigned32   keyfile_fmt_ver;
} sec_pk_keyfile_hdr_t;

extern const sec_pk_keyfile_hdr_t sec_pk_keyfile_hdr;
#define DCEPKKEYFILE_ID sec_pk_keyfile_hdr.keyfile_id;
#define DCEPKKEYFILE_VER sec_pk_keyfile_hdr.keyfile_fmt_ver;

/* s e c _ p k _ d a t a _ i n i t
 *
 * Initializes sec_pk_data_t or its type-specific variants.
 *
 *   In Parameters:
 *       data_p  - pointer to a sec_pk_data_t
 *                 that is to be initialized.
 *                 data_p is returned
 *                 with data_p->len set to 0 and
 *                 data_p->data set to NULL.
 *
```

```
 *   Errors: none.
 */
#define sec_pk_data_init(data_p) \
      do { sec_pk_data_t *__d = \
        (data_p); __d->data = NULL; __d->len = 0; } \
      while (0)

/* s e c _ p k _ d a t a _ c r e a t e
 *
 * Copies data to a sec_pk_data_t or its type-specific variants.
 *
 *   In Parameters:
 *       data, len
 *                pointer to, and length of, memory to be copied
 *   Out Parameters:
 *       data_out_p  - pointer to a sec_pk_data_t
 *                 that receives a pointer to the copied data.
 *
 *   Errors:
 *     error_status_ok    - success
 *     sec_pk_e_no_memory - malloc failed
 */
error_status_t sec_pk_data_create (
    unsigned8       *data,      /* [in] */
    size_t          len,        /* [in] */
    sec_pk_data_t   *data_out_p /* [out] */
);


/* s e c _ p k _ d a t a _ c o p y
 *
 * Copies a sec_pk_data_t or its type-specific variants.
 *
 *   In Parameters:
 *       data_in_p  - pointer to a sec_pk_data_t
 *                 that points to the memory to be copied
 *   Out Parameters:
 *       data_out_p  - pointer to a sec_pk_data_t
 *                 which receives a pointer to the copied data.
 *
 *   Errors:
 *     error_status_ok    - success
 *     sec_pk_e_no_memory - malloc failed
 */
error_status_t sec_pk_data_copy (
    sec_pk_data_t   *data_in_p, /* [in] */
    sec_pk_data_t   *data_out_p /* [out] */
);


/* s e c _ p k _ d a t a _ f r e e
 *
 * Frees memory associated with data of type
 * sec_pk_data_t or its type-specific variants.
 *
```

```
 *   In Parameters:
 *       data_p  - pointer to a sec_pk_data_t
 *                 that points to the memory
 *                 to be reclaimed.
 *                 data_p is returned
 *                 with data_p->data set to NULL and
 *                 data_p->len set to 0.
 *
 *   Errors: None
 */
void sec_pk_data_free (
    sec_pk_data_t   *data_p       /* [in,out] */
);


/* s e c _ p k _ d a t a _ z e r o _ a n d _ f r e e
 *
 * Zeroes out, then frees memory associated with data
 * of type sec_pk_data_t or its type-specific variants.
 *
 * This function, rather than sec_pk_data_free, should
 * be called for structures containing private or
 * secret keys.
 *
 *   In Parameters:
 *       data_p  - pointer to a sec_pk_data_t
 *                 that points to the memory
 *                 to be reclaimed.
 *                 data_p is returned with
 *                 data_p->data set to NULL,
 *                 and data_p->len set to 0.
 *
 *   Errors: none
 */
void sec_pk_data_zero_and_free (
    sec_pk_data_t   *data_p         /* [in,out] */
);

/* sec_pk_get_key_usage_count
 *
 *   In Parameters:
 *       key_usages  - a value of type sec_pk_usage_flags_t
 *           containing key usage flag settings
 *   Return value:
 *       integer count of the number of key usage
 *       flags set in key_usages
 *
 *   Errors: none
 *
extern int sec_pk_get_key_usage_count(
    sec_pk_usage_flags_t     key_usages
);


/* s e c _ p k _ a t t r _ w r i t e _ s e t u p
```

```
 *
 * Function:
 *     Set up an attribute value with public key data
 *     for call to rs_attr_update()
 *      -sets attr->attr_id to id
 *      -IDL-encodes the public key data as a
 *       sec_passwd_rec_t of type sec_passwd_pubkey
 *      -allocates attr->attr_value.tagged_union.bytes
 *      -copies IDL-encoded public key data to
 *       tagged_union.bytes->data
 *
 * In Parameters:
 *     alloc    - allocator (e.g. malloc or rpc_ss_allocate)
 *     dealloc  - deallocator (e.g. free or rpc_ss_free)
 *     attr     - pointer to an attribute to be set up
 *     id       - UUID of attribute to be written
 *     kvno     - pointer to key version number to be
 *                written; must not be 0!
 *     pk_data_p - pointer to a sec_pk_data_t
 *                containing the key value to
 *                be written as an attribute
 * Errors:
 *     error_status_ok
 *     sec_pk_e_no_memory - malloc failed
 *     sec_pk_e_no_key_value - input pk_data is empty
 *     sec_pk_e_key_vers_unsupported - input kvno is
 *         NULL, < 0, or > 255
 * Errors passed through from:
 *     sec_pwd_encode()
 */
error_status_t sec_pk_attr_write_setup(
    idl_void_p_t       (*alloc)(idl_size_t size),
    void               (*dealloc)(idl_void_p_t ptr),
    sec_attr_t         *attr,
    uuid_t             id,
    unsigned32         *kvno,
    sec_pk_data_t      *pk_data_p
);

#endif
```

7.1.3. security/h/sec_psm_base.h

```
/*
 * @OSF_COPYRIGHT@
 * COPYRIGHT NOTICE
 * Copyright (c) 1996 Open Software Foundation, Inc.
 * ALL RIGHTS RESERVED (DCE).  See the file named COPYRIGHT.DCE
 * in the src directory for the full copyright text.
 */
/*
 * Copyright (c) Hewlett-Packard Company 1996
 * Unpublished work.  All Rights Reserved.
 */

#ifndef _SEC_PSM_BASE_H_
```

```
#define _SEC_PSM_BASE_H_

/* The following private key storage mechanism are
 * architectural.  They cannot be changed without
 * destroying interoperability.
 *
 * The UUIDs are defined in sec_psm.c
 */

extern uuid_t sec_psm_pkss_mechanism;
    /* 72053e72-b01a-11cf-8bf5-0800090a5254 */

extern uuid_t sec_psm_file_mechanism;
    /* 8687c5b8-b01a-11cf-b137-0800090a5254 */

extern uuid_t sec_psm_kdc_pk_init_mechanism;
    /* adb48ed4-e94d-11cf-ab4b-08000919ebb5 */

extern sec_pk_algorithm_id_t rsa_pkcs;
extern sec_pk_algorithm_id_t md5_rsa;
extern sec_pk_algorithm_id_t rsa_enc;

#define RSA_PKCS &rsa_pkcs
#define MD5RSA &md5_rsa
#define RSAENC &rsa_enc

#endif
```

7.2. Password types

In order to allow a cell administrator to request the KDC to update its authentication public key key-pair, two new password types are added to the existing `sec_passwd_type_t enum`, and definitions are added to the existing `sec_passwd_rec_t` structure. These modified types are specified in the file `security/idl/passwd.idl`, listed here:

```
/*
 * @OSF_COPYRIGHT@
 * COPYRIGHT NOTICE
 * Copyright (c) 1990, 1991, 1992, 1993, 1996 Open Software
 *   Foundation, Inc.
 * ALL RIGHTS RESERVED (DCE).  See the file named COPYRIGHT.DCE
 * in the src directory for the full copyright text.
 */
/*
** Copyright (c) Hewlett-Packard Company 1991, 1993, 1996
** Unpublished work. All Rights Reserved.
**
*/
/*
 * User Registry - Base password datatypes
 */

interface passwd

{
```

```
import "dce/sec_base.idl";
import "dce/sec_pk_base.idl";

const    unsigned32    sec_passwd_c_des_key_size      = 8;
typedef  byte sec_passwd_des_key_t[sec_passwd_c_des_key_size];

const    unsigned32    sec_passwd_str_max_len = 512;
const    unsigned32    sec_passwd_str_t_size  = 513;
typedef [string] char  sec_passwd_str_t[sec_passwd_str_t_size];

/* BSAFE1.2.1 limits modulus to 1024 bits, but we will support
 * up to 4096 bits. Public key is stored with private key.  The
 * extra 384 bytes of structure is accurate for BSAFE1.2.1, but
 * may not be sufficient for revisions supporting larger moduli.
 * This value should be checked if new BSAFE (or other RSA
 * encryption implementation) is used.
 */

const    unsigned32    sec_passwd_c_max_pk_modulus        = 4096;
                                                            /*bits*/
const    unsigned32    sec_passwd_c_max_pk_overhead       = 384;
                                                            /*bytes*/
const    unsigned32    sec_passwd_c_max_pk_key_size = \
                (2*sec_passwd_c_max_pk_modulus/8) \
                + sec_passwd_c_max_pk_overhead;

/* currently supported key types */
typedef enum {
    sec_passwd_none,
    sec_passwd_plain,
    sec_passwd_des,
    sec_passwd_pubkey,
    sec_passwd_genprivkey
} sec_passwd_type_t;


typedef  struct {
    sec_passwd_version_t  version_number;
    [string, ptr] char    *pepper;

    union switch (sec_passwd_type_t key_type) {

        case sec_passwd_plain:
            [string, ptr] char    *plain;

        case sec_passwd_des:
            sec_passwd_des_key_t  des_key;

        case sec_passwd_pubkey:
            sec_pk_data_t         pub_key;

        case sec_passwd_genprivkey:
            unsigned32            modulus_size;

    } key;
```

```
} sec_passwd_rec_t;
}
```

7.3. pre_auth_req ERA

The existing pre_auth_req ERA is extended by the addition of one more value. The semantics remain consistent: pre-authentication protocols associated with values lower than the value of the pre_auth_req ERA are rejected by the KDC.

```
Attribute Name: pre_auth_req
Attribute UUID: 6c9d0ec8-dd2d-11cc-abdd-080009353559
Attribute Encoding: sec_attr_enc_integer
ACL Manager Type: 06ab9320-0191-11ca-a9e8-08001e039d7d
Query Permissions Set:  sec_acl_perm_mgmt_info,
Update Permissions Set: sec_acl_perm_mgmt_info
Test Permissions Set:   sec_acl_perm_mgmt_info
Delete Permissions Set: sec_acl_perm_mgmt_info
Unique: TRUE
Reserved: TRUE
Intercell Action: REJECT
Trigger Type: NONE
Trigger Binding: NULL
Scope: ""
Multi-valued: FALSE
Comment: "values: {0=NONE, 1=PADATA_ENC_TIMESTAMPS,
          2=PADATA_ENC_THIRD_PARTY, 3=PADATA_ENC_PUBLIC_KEY}"
```

7.4. DCEPKKeyEncipherment ERA

The DCEPKKeyEncipherment ERA is a reserved ERA that may be attached to any principal object to hold the public key used in the DCE authentication protocol to encrypt DCE session (conversation) keys for that principal.

The format of the DCEPKKeyEncipherment ERA is an IDL-encoded byte-string of type sec_passwd_rec_t with key type sec_passwd_pubkey.

```
Attribute Name: DCEPKKeyEncipherment
Attribute UUID: 76251f8c-6230-11cf-af89-08000919ebb5
Attribute Encoding: sec_attr_enc_bytes
ACL Manager Type: 06ab9320-0191-11ca-a9e8-08001e039d7d
Query Permissions Set:  sec_acl_perm_read
Update Permissions Set: sec_acl_perm_mgmt_info
Test Permissions Set:   sec_acl_perm_read
Delete Permissions Set: sec_acl_perm_mgmt_info
Unique: TRUE
Reserved: TRUE
Intercell Action: ACCEPT
Trigger Type: NONE
Trigger Binding: NULL
Scope: ""
Multi-valued: FALSE
Comment: "The principal's DCE key encryption public key value; an
          X.509 SubjectPublicKeyInfo value in ASN.1 DER format."
```

7.5. DCEPKAuthentication ERA

The DCEPKAuthentication ERA is a reserved ERA that may be attached to any user principal to hold the public key used in the DCE authentication protocol to verify a signature from that user. The DCEPKAuthentication ERA attached to the cell's krbtgt principal is the public key used by the client to verify a signature from the DCE KDC during DCE authentication. The DCEPKAuthentication ERA attached to the client principal is the public key used by the KDC to verify a signature from the client during DCE authentication.

The format of the `DCEPKKeyEncipherment` ERA is an IDL-encoded byte-string of type `sec_passwd_rec_t` with key type `sec_passwd_pubkey`.

```
Attribute Name: DCEPKAuthentication
Attribute UUID: d44dc60e-6230-11cf-98ed-08000919ebb5
Attribute Encoding: sec_attr_enc_bytes
ACL Manager Type: 06ab9320-0191-11ca-a9e8-08001e039d7d
Query Permissions Set:  sec_acl_perm_read
Update Permissions Set: sec_acl_perm_mgmt_info
Test Permissions Set:   sec_acl_perm_read
Delete Permissions Set: sec_acl_perm_mgmt_info
Unique: TRUE
Reserved: TRUE
Intercell Action: ACCEPT
Trigger Type: NONE
Trigger Binding: NULL
Scope: ""
Multi-valued: FALSE
Comment: "The principal's DCE authentication public key value; an
          X.509 SubjectPublicKeyInfo value in ASN.1 DER format."
```

7.6. DCEPKPrivateKeyStorage ERA

The `DCEPKPrivateKeyStorage` ERA is a reserved ERA that may be attached to any user principal to identify the private key storage mechanism of that user.

The format of the `DCEPKPrivateKeyStorage` ERA is a UUID.

```
Attribute Name: DCEPKPrivateKeyStorage
Attribute UUID: 1f79c78e-b026-11cf-b7c8-0800090a5254
Attribute Encoding: sec_attr_enc_uuid
ACL Manager Type: 06ab9320-0191-11ca-a9e8-08001e039d7d
Query Permissions Set:  sec_acl_perm_mgmt_info
Update Permissions Set: sec_acl_perm_mgmt_info
Test Permissions Set:   sec_acl_perm_mgmt_info
Delete Permissions Set: sec_acl_perm_mgmt_info
Unique: TRUE
Reserved: TRUE
Intercell Action: Reject
Trigger Type: NONE
Trigger Binding: NULL
Scope: ""
Multi-valued: FALSE
Comment: "Values: sec_psm_pkss_mechanism,
          sec_psm_file_mechanism,
          sec_psm_kdc_pk_init_mechanism"
```

8. USER INTERFACES

8.1. Login

User interfaces to login utilities have not changed, except that additional new error conditions may be reported.

Login utilities such as `dce_login` invoke the existing `sec_login` API, which changes only by the addition of new error status values that can be returned. Login utilities still need to prompt for a user name and a password.

The ''password'' that the user supplies is first tried by `sec_login` as a passphrase to access the user's Personal Security Module. If there is no Personal Security Module, or if the passphrase fails to unlock the module, the ''password'' is then tried automatically as a DCE password (unless the user is identified as requiring public key). See ''TGT Acquisition Protocol'' under ''FUNCTIONAL DEFINITION'' for more information.

8.2. Key Storage and Retrieval

The existing `dcecp` user interface for reading and writing ERA values may be used to read or write the `DCEPKKeyEncipherment`, `DCEPKAuthentication`, `DCEPKPrivateKeyStorage`, or `pre_auth_req` ERAs.

The `DCEPKAuthentication` and `DCEPKKeyEncipherment` attributes must be IDL-encoded byte-strings of type `sec_passwd_rec_t` with key type `sec_passwd_pubkey`.

New options for manipulating public and private key values are defined for the `dcecp> account create` and `dcecp> account modify` commands. These allow setting a principal's private key value in the Personal Security Module, and setting the corresponding public key value in the DCE Registry.

There is no user interface provided for reading the value of a principal's private key.

8.2.1. Existing dcecp operations for public key

Examples of accessing public key information using existing operations:

```
dcecp> principal show <principal> -xattr
{DCEPKAuthentication {30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d  \
  01 01 01 05 00 03 81 \
  8d 00 30 81 89 02 81 81 00 b3 b2 02 84 85 ea cf 0e b8 3e c6 \
  7f c2 16 ff ea 30 79 cf a2 20 2f c5 ea 43 dc e5 39 f1 01 d2 \
  50 52 7c 54 a9 2e 02 c3 8c 57 0c b5 46 b4 4b 0c 20 1d c6 66 \
  6d 70 16 c2 d3 f2 39 0f 1a 3a 44 f1 ee 35 dd fb 16 77 e6 a7 \
  6c c6 86 ba f5 e9 a2 54 54 60 43 7a 2e cd dd 36 0d 8e 0a 03 \
  4f 12 22 95 ff e1 da 3e 25 b1 fb 0a ce f7 9d 95 a2 89 0b bd \
  a6 9a c9 ec d7 c4 e4 4b 89 26 29 30 67 8f 14 89 b5 02 03 01 \
  00 01}}
{DCEPKKeyEncipherment {30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d \
  01 01 01 05 00 03 81 \
  8d 00 30 81 89 02 81 81 00 b2 c4 c2 b9 cd c3 9b 3b 8f 54 35 \
  88 fc 22 8e 39 cd 3d 25 99 62 87 f0 0c 00 b9 55 cd f4 a4 b5 \
  39 25 94 ee 23 5a 35 c6 da 2f 6f 5a a5 9a a3 b4 22 91 43 bf \
  68 3d e8 51 63 43 d0 56 ba c0 86 a3 b2 10 1d e0 05 7e 34 b8 \
  90 4a ed 06 88 ce 3e 52 08 ad 5a 2a ae 7c 3f 0a ce 3b 40 8f \
  03 48 79 7d bd 6e c2 7e 3d 37 5e 8a 63 ff e8 09 5b 93 ad 04 \
  12 ca d9 f2 2f 74 14 c6 c9 79 c5 34 cb 93 71 f9 5d 02 03 01 \
  00 01}}
{DCEPKPrivateKeyStorage 8687c5b8-b01a-11cf-b137-0800090a5254}
{pre_auth_req 3}

dcecp> principal show krbtgt/<cell> -xattr
{DCEPKAuthentication {30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d  \
  01 01 01 05 00 03 81 \
  8d 00 30 81 89 02 81 81 00 c3 6d 50 5f 28 da 55 e1 fb 94 ac \
  2a 29 16 44 bd 62 d0 61 c5 e1 cd f2 a1 3a f6 1c e7 40 11 0a \
  4d eb 80 38 5a b7 1b bf 25 ae 68 6b bc d7 ad 05 27 67 5b c3 \
  bb f7 66 23 4b 59 a4 3f 9f 74 e1 a1 4a a5 9c 86 9c 23 58 b3 \
  3f a9 45 5d d5 e6 33 21 79 4d 86 dd 35 2f fb a2 f6 59 76 ec \
  06 b9 2d 5c 5c 75 61 c7 b4 ec 50 6d 30 02 6d d5 0a 45 ad 06 \
  54 4e 60 c4 f8 77 3b 6c da 58 5d 99 f6 37 cb cb 3d 02 03 01 \
  00 01}}
{DCEPKPrivateKeyStorage adb48ed4-e94d-11cf-ab4b-08000919ebb5}
```

```
dcecp> principal modify <principal> [ -add | -change ] \
       {pre_auth_req 3}
```

8.2.2. New dcecp operations for public key

New options have been added to `dcecp` to assist cell administrators in setting up accounts to use public key login.

Syntax of new account create and modify options:

```
Attribute name       Value
--------------       -----
-pkmechanism         file | pkss
-pksignatureusage    {<pk_attributes>}
-pkkeycipherusage    {<pk_attributes>}

pk_attributes        Value
-------------        -----
generatekey          Integer: value "default" or desired modulus size
privatekeyfile       File system path to private key
publickeyfile        File system path to public key
oldpassphrase        Passphrase string
newpassphrase        Passphrase string
```

The value of the `generatekey` option is the desired modulus size for the key.

> *NOTE: In the reference implementation, the default value for generatekey, indicated by specifying the parameter value* `default`*, is 1024.*

`{publickeyfile ...}` and `{privatekeyfile ...}`
may not be used together with `{generatekey ...}`.

`{publickeyfile ...}` and `{privatekeyfile ...}` must always be specified together: if you have one, you must have the other.

The `{newpassphrase ...}` for the `-pksignatureusage` option must be the same as the `{newpassphrase ...}` for the `-pkkeycipherusage` option.

Examples:

8.2.2.1. Setting up a public key account

```
dcecp> account create pk_account -group none
       -organization none -password pk_pwd
       -mypwd -dce-
       -pkmechanism file
       -pksignatureusage {
           {generatekey default}
           {newpassphrase pk_pass}
       }
       -pkkeycipherusage {
           {generatekey 512}
           {newpassphrase pk_pass}
       }
```

8.2.2.2. Changing a key-pair obtained from an external source

```
dcecp> account modify pk_account
       -pksignatureusage {
           {privatekeyfile /tmp/pk_account.sigkeypvt}
           {publickeyfile /tmp/pk_account.sigkeypub}
           {oldpassphrase pk_pass}
       }
```

8.2.2.3. Changing passphrase on existing keys

```
dcecp> account modify pk_account
        -pksignatureusage {
            {oldpassphrase pk_pass}
            {newpassphrase pk_pwd}
        }
        -pkkeycipherusage {
            {oldpassphrase pk_pass}
            {newpassphrase pk_pwd}
        }
```

Key values obtained from an external source must be stored in a file with identifying header information. The `sec_pubkey_write_keyfile()` API may be used to write a key value to a file in the correct format.

It is up to the user to ensure that public and private key values supplied to the user interfaces are consistent with each other and are valid public key key-pairs.

8.3. KDC Key Update

In order to allow a cell administrator to update the public key pair of the DCE Security Server (KDC), a new `-pkgenprivkey` option to the `dcecp> account modify` command is provided.

Example:

```
dcecp -c account modify krbtgt/dc.cell.ch.hp.com
     -pkgenprivkey default
```

This option is invalid except when used on the account of the `krbtgt` principal of the local cell. It is also invalid except when executed by a user having cell administrator privileges.

The value of the `-pkgenprivkey` option is the desired modulus size of the key to be generated. It is recommended that KDC keys be at least 1024 bits long.

> *NOTE: In the reference implementation, the default value for* `-pkgenprivkey`*, indicated by specifying the parameter value* `default`*, is 1024.*

9. APIs and Interfaces

9.1. DCE Personal Security Module Interface

The interface to the DCE personal Security Module (PSM) Interface is defined in a new file `security/idl/sec_psm.idl`, listed here:

```
/*
 * @OSF_COPYRIGHT@
 * COPYRIGHT NOTICE
 * Copyright (c) 1996 Open Software Foundation, Inc.
 * ALL RIGHTS RESERVED (DCE).  See the file named COPYRIGHT.DCE
 * in the src directory for the full copyright text.
 */
/*
 * Copyright (c) Hewlett-Packard Company 1996
 * Unpublished work.  All Rights Reserved.
 *
 *       Personal Security Module (PSM) Interface
 *
 */

[
    local
```

```
]

interface sec_psm

{

    import "dce/sec_pk_base.idl";

    typedef void *sec_psm_handle_t;

/* PSM API */

/* sec_psm_open
 *
 * Open the personal security mechanism using password.
 *
 * Return status
 *   error_status_ok:    Success.
 *   other (non-zero):   sec_pvtkey_privileged
 *                       sec_pvtkey_no_more_memory
 *                       sec_psm_no_more_memory
 *
 * Input
 *
 *   name: Pointer to the user's canonical name within the
 *         specified domain.
 *   pwd: Pointer to the user's password.
 *   domain_id: Pointer to the application domain the user
 *              is operating on.
 *
 * Output
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.
 *
 */

error_status_t sec_psm_open(
    [in]  void              *name,
    [in]  char              *pwd,
    [in]  sec_pk_domain_t   *domain_id,
    [out] sec_psm_handle_t  *psm_handle
);


/* sec_psm_close
 *
 * Close the personal security mechanism and cleanup the
 * personal security context data.  It also ensures any
 * confidential information such as passwords or private
 * key are zeroed.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  sec_psm_not_init
```

```
 *                       sec_psm_invalid_handle
 *                       sec_psm_internal_error
 *                       sec_pvtkey_invalid_handle
 *                       sec_pvtkey_mechanism_not_init
 *
 * Input
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.  This handle should be
 *               obtained through sec_psm_open().
 *
 */

error_status_t sec_psm_close(
    [in]  sec_psm_handle_t    psm_handle
);

/* sec_psm_sign_data
 *
 * Compute the signature of the input data using the
 * signature algorithm specified in the arguments.
 *
 * The routine allocates memory for the data returned in
 * the signature parameter.  Users should call
 * sec_pk_data_free() to deallocate that memory.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  sec_psm_not_init
 *                       sec_psm_invalid_handle
 *                       sec_psm_unsupported_algorithm_id
 *                       sec_pvtkey_invalid_handle
 *                       sec_pvtkey_mechanism_not_init
 *                       sec_pvtkey_internal_error
 *                       sec_pvtkey_invalid_password
 *                       sec_pvtkey_multiple_key_usages
 *
 * Input
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.  This handle should be
 *               obtained through sec_psm_open().
 *   signature_alg_id: The ASN.1 DER-encoded object ID of
 *               the signature algorithm, such as
 *               MD5WithRSAEncryption.
 *   key_usage: The usage of the private key which should be
 *              picked for this operation.
 *   data: Pointer to the ASN.1 DER-encoded data to be signed.
 *
 * Output
 *
 *   kvno: Key version number of the key used.
 *   signature: Pointer to a signature buffer pointer.
 *
 */
```

```
error_status_t sec_psm_sign_data(
    [in]  sec_psm_handle_t        psm_handle,
    [in]  sec_pk_algorithm_id_t   *signature_alg_id,
    [in]  sec_pk_usage_flags_t    key_usage,
    [in]  sec_pk_gen_data_t       *data,
    [out] unsigned32              *kvno,
    [out] sec_pk_signed_t         *signature
);


/* sec_psm_verify_data
 *
 * Verify the data.  Usually, you verify other people's signature.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  sec_psm_not_init
 *                      sec_psm_invalid_handle
 *                      sec_psm_unsupported_algorithm_id
 *                      sec_bsafe_decryption_failure
 *                      sec_pk_e_domain_unsupported,
 *                      sec_pk_e_usage_unsupported,
 *                      sec_rgy_object_not_found,
 *                      sec_rgy_not_authorized,
 *                      sec_attr_unauthorized
 *
 * Input
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.  This handle should be
 *               obtained through sec_psm_open().
 *   signer_domain: pointer to the application domain of the
 *              principal; [see the sec_pubkey interface for more
 *              details.]
 *   signer_name: Pointer to the name of the principal which had
 *               signedthe data.
 *   kvno: Key version number of the key used.
 *   signature_alg_id: The ASN.1 DER-encoded object ID of the
 *              signature algorithm, such as MD5WithRSAEncryption.
 *   key_usage: The usage of the ;public key which should be
 *              picked for this operation.
 *   data: Pointer to the data to be verified
 *   signature: Pointer to the signature to be verified.
 *
 */

error_status_t sec_psm_verify_data(
    [in]  sec_psm_handle_t        psm_handle,
    [in]  sec_pk_domain_t         *signer_domain_id,
    [in]  void                    *signer_name,
    [in]  unsigned32              kvno,
    [in]  sec_pk_algorithm_id_t   *signature_alg_id,
    [in]  sec_pk_usage_flags_t    key_usage,
    [in]  sec_pk_gen_data_t       *data,
    [in]  sec_pk_signed_t         *signature
```

```
);


/* sec_psm_encrypt_data
 *
 * Encrypt the data in the algorithm specified.  The routine
 * allocates memory for the data returned in the cipher_data
 * parameter.  Users should call sec_pk_data_free() to
 * deallocate that memory.
 *
 * For reference implementation, only keyEncipherment key_usage
 * will be implemented.
 *
 * Usually you encrypt the data with other's public key.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  sec_psm_not_init
 *                      sec_psm_invalid_handle
 *                      sec_psm_unsupported_algorithm_id
 *                      sec_pk_e_domain_unsupported,
 *                      sec_pk_e_usage_unsupported,
 *                      sec_rgy_object_not_found,
 *                      sec_rgy_not_authorized,
 *                      sec_attr_unauthorized
 *
 * Input
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.  This handle should be
 *               obtained through sec_psm_open().
 *   encryptee_domain: pointer to the application domain
 *               of the principal;
 *              [see the sec_pubkey interface for more details.]
 *   encryptee_name: Pointer to the name of the principal this
 *              data is encrypted for.
 *   encryption_alg_id: The ASN.1 DER-encoded object ID of
 *              encryption algorithm, such as RSA.
 *   key_usage: The usage of the public key this key pair
 *              belongs to.
 *   clear_data: Pointer to the ASN.1 DER-encoded data to be
 *               encrypted.
 *
 * Input, output
 *
 *   kvno: Key version number of the key used.
 *
 * Output
 *
 *   cipher_data: Pointer to the encrypted output buffer.
 */

error_status_t sec_psm_encrypt_data(
    [in]  sec_psm_handle_t              psm_handle,
    [in]  sec_pk_domain_t               *encryptee_domain,
```

```
    [in]  void                           *encryptee_name,
    [in,out]  unsigned32                 *kvno,
    [in]  sec_pk_algorithm_id_t          *encryption_alg_id,
    [in]  sec_pk_usage_flags_t           key_usage,
    [in]  sec_pk_gen_data_t              *clear_data,
    [out] sec_pk_encrypted_t             *cipher_data
);

/* sec_psm_decrypt_data
 *
 * Decrypt the mechanism-specific encrypted data. The routine
 * allocates memory for the data returned in the clear_data
 * parameter.  Users should call sec_pk_data_free() to
 * deallocate that memory.
 *
 * For reference implementation, only keyEncipherment key_usage
 * will be implemented.
 *
 * Usually you decrypted the data with your own private key.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  sec_psm_not_init
 *                      sec_psm_invalid_handle
 *                      sec_psm_unsupported_algorithm_id
 *                      sec_bsafe_encryption_failure
 *                      sec_pvtkey_invalid_handle
 *                      sec_pvtkey_mechanism_not_init
 *                      sec_pvtkey_internal_error
 *                      sec_pvtkey_invalid_password
 *                      sec_pvtkey_multiple_key_usages
 *
 * Input
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.  This handle should be
 *               obtained through sec_psm_open().
 *   kvno: Key version number of the key used.
 *   encryption_alg_id: The ASN.1 DER-encoded object ID of
 *               encryption algorithm, such as RSA.
 *   key_usage: The usage of the private key which should be
 *              picked for this operation.
 *   cipher_data: Pointer to encrypted cipher buffer.
 *
 * Output
 *
 *   clear_data: Pointer to decrypted clear text
 *                buffer pointer.
 */

error_status_t sec_psm_decrypt_data(
    [in]  sec_psm_handle_t               psm_handle,
    [in]  unsigned32                     kvno,
    [in]  sec_pk_algorithm_id_t          *encryption_alg_id,
    [in]  sec_pk_usage_flags_t           key_usage,
```

```
    [in]  sec_pk_encrypted_t            *cipher_data,
    [out] sec_pk_gen_data_t             *clear_data
);


/* sec_psm_gen_pub_key
 *
 * Generate public key according to key type.  The routine
 * allocates memory for the data returned in the key
 * parameters.  Users should call sec_pk_data_free() to
 * deallocate that memory.
 *
 * Currently it only supports RSADSI's public key.
 *
 * Return status
 *   error_status_ok:    Success.
 *   other (non-zero):   sec_psm_wrong_pub_key_type
 *                       sec_bsafe_alloc
 *
 * Input
 *
 *    key_type: Only RSA_PKCS is supported.
 *
 *    modulus_bit_size: Desired key-length.  The length of the
 *                       desired key.  Interpretation of this
 *                       parameter is algorithm-dependent; For RSA,
 *                       it shall be interpreted as the bit-length of
 *                       the key.
 *
 * Output
 *
 *    private_key:  Pointer to the encoded private key structure of
 *                  the newly-generated key.
 *    public_key: Pointer to the encoded public key structure of the
 *                 newly-generated key.
 *
 */

error_status_t sec_psm_gen_pub_key(
    [in]  sec_pk_algorithm_id_t   *key_type,
    [in]  unsigned32              modulus_bit_size,
    [in]  sec_pk_gen_data_t       *seed,
    [out] sec_pk_data_t           *public_key,
    [out] sec_pk_data_t           *private_key
);

/* ADMINISTRATIVE INTERFACES */

/* sec_psm_put_pub_key
 *
 * Store the public key pair with associated data into the personal
 * security mechanism.
 *
 * Return status
 *   error_status_ok:   Success.
```

```
 *   other (non-zero):  sec_psm_not_init
 *                      sec_psm_invalid_handle
 *                      sec_pk_e_domain_unsupported,
 *                      sec_pk_e_usage_unsupported,
 *                      sec_rgy_object_not_found,
 *                      sec_rgy_not_authorized,
 *                      sec_attr_unauthorized
 *                      sec_pvtkey_invalid_handle
 *                      sec_pvtkey_mechanism_not_init
 *                      sec_pvtkey_no_more_memory
 *                      sec_pvtkey_internal_error
 *      sec_pvtkey_same_domain_and_usage_key_already_exists.
 *
 * Input
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.  This handle should be
 *               obtained through sec_psm_open().
 *   pwd:  Pointer to the user's password.
 *   key_usage: The usage of the public key this key pair
 *              belongs to.
 *   pvtkey: Pointer to the ASN.1 DER-encoded private key buffer.
 *   pubkey: Pointer to the ASN.1 DER-encoded public key buffer.
 *
 */

error_status_t sec_psm_put_pub_key(
    [in]  sec_psm_handle_t       psm_handle,
    [in]  char                   *pwd,
    [in]  sec_pk_usage_flags_t   key_usage,
    [in]  sec_pk_pubkey_t        *public_key,
    [in]  sec_pk_pvtkey_t        *private_key
);

/* sec_psm_update_pub_key
 *
 * Update the user's own public key pair or passphrase. The
 * oldpassphrase is specified to authenticate the user updating the
 * key.
 *
 * In the reference implementation, only a single version of a key
 * with a given key usage will be maintained.  Any old key version
 * will be overwritten.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  sec_psm_not_init
 *                      sec_psm_invalid_handle
 *                      sec_pk_e_domain_unsupported,
 *                      sec_pk_e_usage_unsupported,
 *                      sec_rgy_object_not_found,
 *                      sec_rgy_not_authorized,
 *                      sec_attr_unauthorized
 *                      sec_pvtkey_invalid_handle
 *                      sec_pvtkey_mechanism_not_init
```

```
 *                       sec_pvtkey_private_key_is_not_supplied
 *                       sec_pvtkey_new_password_required
 *                       sec_pvtkey_no_more_memory
 *                       sec_pvtkey_internal_error
 *                       sec_pvtkey_no_matched_private_key
 *                       sec_pvtkey_Invalid_password.
 *
 * Input
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.  This handle should be
 *               obtained through sec_psm_open().
 *   oldpwd:  Pointer to the user's current password.
 *   newpwd:  Pointer to the user's new password.
 *   key_usage: The usage of the public key this key pair
 *              belongs to.
 *   pvtkey: Pointer to the ASN.1 DER-encoded private key buffer.
 *   pubkey: Pointer to the ASN.1 DER-encoded public key buffer.
 *
 */

error_status_t sec_psm_update_pub_key(
    [in]  sec_psm_handle_t       psm_handle,
    [in]  char                   *oldpwd,
    [in]  char                   *newpwd,
    [in]  sec_pk_usage_flags_t   key_usage,
    [in]  sec_pk_pubkey_t        *public_key,
    [in]  sec_pk_pvtkey_t        *private_key
);

}
```

9.2. DCE Public Key Module API

The interface to the DCE Public Key Module API is defined in a new file `security/idl/sec_pubkey.idl`, listed here:

```
/*
 * @OSF_COPYRIGHT@
 * COPYRIGHT NOTICE
 * Copyright (c) 1996 Open Software Foundation, Inc.
 * ALL RIGHTS RESERVED (DCE).  See the file named COPYRIGHT.DCE
 * for the full copyright text.
 */
/*
** Copyright (c) Hewlett-Packard Company 1996
** Unpublished work.  All Rights Reserved.
**
**   Public Key Storage and Retrieval Interface
*/

[
    local
]

interface sec_pubkey {
```

```
import "dce/rgynbase.idl";
import "dce/sec_pk_base.idl";

typedef error_status_t (*sec_pk_key_retrieval_fn_t)(
            [in] void *principal,
            [in] sec_pk_domain_t *domain,
            [in] sec_pk_usage_flags_t usage,
            [in,out] unsigned32 *kvno,
            [out] sec_pk_pubkey_t *pubkey_p
);

typedef error_status_t (*sec_pk_key_storage_fn_t)(
            [in] void *principal,
            [in] sec_pk_domain_t *domain,
            [in] sec_pk_usage_flags_t usage,
            [in,out] unsigned32 *kvno,
            [in] sec_pk_pubkey_t *pubkey_p
);

typedef struct
{
    sec_pk_domain_t domain;
    sec_pk_key_retrieval_fn_t r;
    sec_pk_key_storage_fn_t s;
} sec_pk_domain_def;

/*
 * s e c _ p u b k e y _ d e f i n e _ d o m a i n
 *
 * Define a public key storage domain.
 *
 * This extends the set of domains supported by the other
 * sec_pubkey functions within the current process.
 *
 * This is used to define "new" domains within the KDC
 * and within the pk-init client which fetch public keys directly
 * from known trusted sources.
 *
 * domain -- the new domain to define.
 *
 * retrieve -- pointer to a function which retrieves a public key,
 *     or NULL if retrievals are not possible in this domain.
 *
 * store -- pointer to a function which stores a public key,
 *     or NULL if storage is not possible in this domain.
 */
error_status_t sec_pubkey_define_domain (
    sec_pk_domain_t *domain,
    sec_pk_key_retrieval_fn_t retrieve,
    sec_pk_key_retrieval_fn_t store
);

/*
 * s e c _ p u b k e y _ f i n d _ d o m a i n
 *
```

```
 * find the functions implementing a public key storage domain.
 *
 * domain -- the domain to find.
 *
 * Returns NULL if domain is not implemented.
 */
sec_pk_domain_def *sec_pubkey_find_domain(
    sec_pk_domain_t *domain
);

/* PUBLIC KEY STORAGE AND RETRIEVAL OPERATIONS
 *
 * These interfaces are used within the sec_psm interface,
 * and the sec_login interface, but may also be used
 * directly by user applications.
 *
 */

/* s e c _ p u b k e y _ s t o r e
 *
 * Function:
 *     Store a public key value for a principal
 *
 *     Stores an X.509 DER-encoded public key value of type
 *     SubjectPublicKeyInfo in the public key storage
 *     facility.  This operation overwrites any
 *     existing public key storage for <principal>, and
 *     creates public key storage if none exists.
 *
 *     In the reference implementation:
 *     - only the sec_pk_domain_dce_pk_login domain
 *       supports the sec_pubkey_store operation
 *     - usage must be either sec_pk_usage_digitalSignature,
 *       sec_pk_usage_keyEncipherment, or both.
 *     - If usage is sec_pk_usage_digitalSignature the public
 *       key is stored in the DCEPKAuthentication ERA attached
 *       to the principal.
 *     - If usage is sec_pk_usage_keyEncipherment, the public key is
 *       stored in the DCEPKKeyEncipherment ERA.
 *
 *     Default ACLs for the reference implementation on the
 *     underlying ERA storage locations will allow only a
 *     principal with sec_admin privileges to perform
 *     this operation.  Sites may choose to modify the ACLs
 *     on the underlying ERAs to allow the owner principal
 *     to modify the public key as well, although sites
 *     should be aware that ability to modify a principal's
 *     public key equates to the ability to impersonate the
 *     principal.
 *
 * In Parameters:
 *     principal  - the canonical name in the specified
 *                  domain of the principal whose public
 *                  key is being stored (sec_rgy_name_t
 *                  containing /.../cell/principal in
```

```
 *                  reference implementation)
 *     domain     - a UUID identifying the domain in which
 *                  the key is used
 *                  (sec_pk_domain_dce_pk_login in
 *                  reference implementation)
 *     usage      - usage(s) of the key being stored
 *                  (sec_pk_usage_digitalSignature,
 *                  sec_pk_usage_keyEncipherment, or both
 *                  in reference implementation)
 *     kvno       - key version; if 0,
 *                  retrieves newest version.
 *     pubkey_p   - pointer to an X.509 DER-encoded public key
 *                  value of type SubjectPublicKeyInfo
 *
 * Out Parameters:
 *     None
 *
 * Errors:
 *     error_status_ok
 *     sec_pk_e_domain_unsupported,
 *     sec_pk_e_usage_unsupported,
 *     sec_pk_e_key_vers_unsupported
 *     sec_pk_e_no_memory
 *
 * Errors passed through from:
 *     uuid_equal
 *     sec_rgy_attr_update
 */
error_status_t sec_pubkey_store (
    [in]  void                  *principal,
    [in]  sec_pk_domain_t       *domain,
    [in]  sec_pk_usage_flags_t  usage,
    [in,out]  unsigned32            *kvno,
    [in]  sec_pk_pubkey_t     *pubkey_p
);

/* s e c _ p u b k e y _ r e t r i e v e
 *
 * Function:
 *     Retrieves an X.509 DER-encoded public key value with
 *     type SubjectPublicKeyInfo from the public key
 *     storage facility for a given principal.
 *
 *     In the reference implementation:
 *     - usage must be either sec_pk_usage_digitalSignature or
 *       sec_pk_usage_keyEncipherment
 *     - If usage is sec_pk_usage_digitalSignature the
 *       public key is retrieved from the
 *       DCEPKAuthentication ERA attached to the
 *       principal.
 *     - If usage is sec_pk_usage_keyEncipherment, the
 *       public key is retrieved from the
 *       DCEPKKeyEncipherment ERA.
 *
 *     Default ACLs for the reference implementation allow
```

```
 *     any principal to perform this operation.
 *
 * In Parameters:
 *     principal  - the canonical name in the specified
 *                  domain of the principal whose public
 *                  key is being retrieved
 *                  (sec_rgy_name_t in reference
 *                   implementation)
 *     domain     - a UUID identifying the domain in which
 *                  the key is used
 *     usage      - usage of the key being retrieved
 *                  (sec_pk_usage_digitalSignature, or
 *                  sec_pk_usage_keyEncipherment, but not
 *                   both, in reference implementation)
 *     kvno       - key version to be retrieved; if 0,
 *                  retrieves newest version.
 *
 * Out Parameters:
 *     pubkey_p   - pointer to an X.509 DER-encoded
 *                  public key value of type
 *                  SubjectPublicKeyInfo; storage for this
 *                  structure must be freed using the
 *                  sec_pk_data_free() function.
 *
 * Errors:
 *     error_status_ok
 *     sec_pk_e_key_vers_unsupported
 *     sec_pk_e_usage_unsupported
 *     sec_pk_e_key_attr_read_failed
 *     sec_pk_e_no_memory
 *
 * Errors passed through from:
 *     sec_id_global_parse_name
 *     sec_rgy_cell_bind
 *     sec_rgy_attr_cursor_alloc
 *     sec_rgy_attr_lookup_by_id
 *     uuid_equal
 */
error_status_t sec_pubkey_retrieve (
    [in]  void                  *principal,
    [in]  sec_pk_domain_t       *domain,
    [in]  sec_pk_usage_flags_t   usage,
    [in,out]  unsigned32        *kvno,
    [out] sec_pk_pubkey_t       *pubkey_p
);

/* s e c _ p u b k e y _ r e a d _ k e y f i l e
 *
 * Function:
 *    Read a key from a keyfile.  Keyfile is assumed
 *    to have format as follows:
 *    unsigned32  'P' 'K' 'E' 'Y'
 *    unsigned32  file format version# (must be 1 or 2)
 *    unsigned32  key_version (file format version 2 only)
 *    unsigned32  key_length
```

```
 *    byte        key[key_length]
 *
 * In Parameters:
 *    keyfilepath      -path to keyfile
 *    key_version_p    -NULL or pointer to key version number
 *                      desired; if 0, any key version accepted
 *
 * Out Parameters:
 *    key_version_p    -NULL or pointer to key version number
 *                      being returned
 *    file_fmt_ver_p   -address of file format version (NULL if
 *                      no output file format desired).  This will
 *                      be returned if it can be read, even if
 *                      version does not match or key can't be read.
 *    key_ret_p        -address of sec_pk_data_t key; caller must
 *                      allocate the sec_pk_data_t before calling
 *                      this function.  The caller must
 *                      deallocate key.data (and, if malloc'd,
 *                      key) when finished.  In the case of a
 *                      private key, key.data should be zeroed
 *                      before being deallocated.
 *
 * Errors:
 *     sec_pk_e_kf_param_not_alloc - A parameter other than
 *                                    file_fmt_ver_p NULL
 *     sec_pk_e_kf_open_err        - error opening the keyfile
 *     sec_pk_e_kf_read_err        - error reading the keyfile
 *     sec_pk_e_kf_format_err      - keyfile did not start with
 *                                    'P''K''E''Y'
 *     sec_pk_e_kf_version_err     - keyfile version not 1 or 2
 *     sec_pk_e_no_memory          - unable to allocate memory for
 *                                    key data
 */
error_status_t sec_pubkey_read_keyfile(
    [in]     char          *keyfilepath,
    [in]     unsigned32    *file_fmt_ver_p,
    [in,out] unsigned32    *key_version_p,
    [out]    sec_pk_data_t *key_ret_p
);

/* s e c _ p u b k e y _ w r i t e _ k e y f i l e
 *
 * Function:
 *    Write a key value to a keyfile.  Keyfile
 *    has following format:
 *    unsigned32  'P' 'K' 'E' 'Y'
 *    unsigned32  file format version# (2)
 *    unsigned32  key_version
 *    unsigned32  key_length
 *    byte        key[key_length]
 *
 * In Parameters:
 *    keyfilepath      -path to keyfile
 *    key_version_p    -pointer to key version to be written
 *    key_p            -address of sec_pk_data_t structure holding
```

```
 *                      the key to be written
 *
 * Out Parameters:
 *    key_version_p    -pointer to key version written
 *    file_fmt_ver_p   -address of file format version written
 *                      (NULL if you don't want this)
 *
 * Errors:
 *     sec_pk_e_kf_param_not_alloc - parameter other than
 *                                   file_fmt_ver_p NULL
 *     sec_pk_e_kf_open_err        - error opening the keyfile
 *                                   for writing
 *     sec_pk_e_kf_write_err       - error writing the keyfile
 */
error_status_t sec_pubkey_write_keyfile(
    [in]  char            *keyfilepath,
    [in]  sec_pk_data_t   *key_p,
    [in,out] unsigned32   *key_version_p,
    [out] unsigned32      *file_fmt_ver_p
);

}
```

9.3. DCE Private Key Module Interface

The interface to the DCE Private Key Module Interface is defined in two new files `security/idl/sec_pvtkey.idl` and `security/psm/domestic/sec_pvtkey_switch.h`, listed here. `sec_pvtkey_switch.h` defines an interface for registering a private key storage mechanism. `sec_pvtkey.idl` defines the interface between the `sec_psm` layer and the `sec_pvtkey` layer.

9.3.1. security/idl/sec_pvtkey.idl

```
/*
 * @OSF_COPYRIGHT@
 * COPYRIGHT NOTICE
 * Copyright (c) 1996 Open Software Foundation, Inc.
 * ALL RIGHTS RESERVED (DCE).  See the file named COPYRIGHT.DCE
 * in the src directory for the full copyright text.
 */
/*
 * Copyright (c) Hewlett-Packard Company 1996
 * Unpublished work.  All Rights Reserved.
 *
 *       Private Key Storage Interface
 */

[
    local
]

interface sec_pvtkey

{

    import "dce/rgybase.idl";
    import "dce/sec_pk_base.idl";
```

```
    typedef void *sec_pk_mechanism_handle_t;

    /* Capability should be uuid */

    typedef uuid_t sec_pk_pvtkey_capability;

/* sec_pvtkey_open
 *
 * Open (connect to) the private key storage service.  For file-based
 * keystore implementation, it creates the file for the follow up key
 * store.  It opens the file for the follow up key retrieval or key
 * update.  It can involve a password check (please note, password
 * check is not applicable to sys admin's operation).  For a
 * server-based implementation, it binds to the server.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):
 *             for PKSS:
 *                      Can't initialize keystore container.
 *                      Can't locate keystore.
 *                      Can't communicate with keystore.
 *                      Access-control failure.
 *             for file base:
 *                      sec_pvtkey_privileged
 *                      sec_pvtkey_no_more_memory
 *
 * Input
 *
 *    Name: Pointer to the user's canonical name within the
 *          specified domain.
 *    domain_id: The application domain this key belongs to.
 *
 * Output
 *    handle: A pointer to an opaque handle to the private key
 *            context data.  The context data should contain the
 *            domain_id of the operation.
 */

error_status_t sec_pvtkey_open(
        [in]  char                      *name,
        [in]  sec_pk_domain_t           *domain_id,
        [out] sec_pk_mechanism_handle_t *handle
);

/* sec_pvtkey_close
 *
 * Close the connection to the private key storage service.  It also
 * frees the handle.  If there are sensitive data, the data should
 * be zeroed first.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  Can't close keystore container.
 *                      Can't locate keystore.
```

```
 *                      Can't communicate with keystore.
 *                      Access-control failure.
 *              for file base:
 *                      sec_pvtkey_invalid_handle
 *                      sec_pvtkey_mechanism_not_init
 *
 * Input
 *
 *    handle: A pointer to an opaque handle to the private key
 *            context data.  The context data should contain the
 *            domain_id of the operation.  The handle should be
 *            obtained throught sec_pvtkey_open().
 */

error_status_t sec_pvtkey_close(
        [in]  sec_pk_mechanism_handle_t handle
);

/* sec_pvtkey_capabilities
 *
 * Inquire about the capabilities of a key storage service.  Multiple
 * storage services may be supported by a client system, selected
 * according to username and/or domain_id.  If no keystore service
 * is defined for a given username and domain_id, all sec_pvtkey
 * operations for that name/domain will return the error
 * "Can't locate keystore".  Some services may not
 * support all storage service operations, and the
 * sec_pvtkey_capabilities function allows the PSM module to
 * interrogate a particular storage service to determine whether a
 * given feature or operation is supported.
 *
 * Return status
 *   error_status_ok:    Success.
 *   other (non-zero):   Can't locate keystore.
 *                       Can't communicate with keystore.
 *                  for file base:
 *                       sec_pvtkey_capability_not_supported
 *
 * Input
 *    Name: Pointer to the user's canonical name within the
 *          specified domain.
 *    capability_parameter: Pointer to the parameter specific to the
 *              capability specified.  In KEYGEN capability, it is
 *              public-key algorithm.  If specified as
 *              NULL, the default public-key algorithm (RSA)
 *              is implied.  Many capability queries will
 *              ignore this parameter.
 *    Feature:  Pointer to a uuid code identifying a service feature.
 *
 *              The following features are currently defined:
 *              Feature code            Feature
 *              ------------            -------
 *              SEC_PVTKEY_CAP_KEYGEN   Service can generate keys for
 *                                      the specified algorithm.
 *              SEC_PVTKEY_CAP_ANYALG   Service can store keys for
```

```
 *                                     any algorithm.
 *              SEC_PVTKEY_CAP_KEYALG   Service can store keys for
 *                                     the specified algorithm.
 *              SEC_PVTKEY_CAP_KEYDEL   Service can delete user's
 *                                     key.
 *
 * Output
 *    Supported: true if the feature is available, false if not.
 *
 */

error_status_t sec_pvtkey_capabilities(
        [in]  sec_pk_mechanism_handle_t handle,
        [in]  void *                    capability_parameter,
        [in]  sec_pk_pvtkey_capability  *feature,
        [out] boolean32                 *supported
   );

/* sec_pvtkey_store
 *
 * Store the user's keypair encrypted using the user's password in
 * the storage service.  The encryption mechanism is determined by
 * the service provider.  The public-key algorithm ID (such as
 * RSA, DSA) should be included in both the private_key and
 * public_key structures which are the final [in] parameters.  The
 * keys will be indexed by name, domain_id, key_usage and
 * key_version in the private key storage service in order to
 * allow retrieval via the sec_pvtkey_get API.
 *
 * If the specified key already exists in the keystore, the
 * routine will leave it unchanged and return a failure status.
 * To replace a key in a keystore, either use sec_pvtkey_update or
 * sec_pvtkey_update_generate, or call sec_pvtkey_delete followed
 * by sec_pvtkey_store.
 *
 * A keystore may enforce its own access-control requirements, and
 * an access-control failure will be indicated by an appropriate
 * status code, with the contents of the keystore unchanged.  For
 * example, a file-based keystore implementation may require that
 * the caller has write-access to the keystore file; a
 * server-based implementation may require that the caller has
 * appropriate ACL-based permission to create a new entry; a
 * memory-card implementation may require that the card be
 * write-enabled.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  Can't initialize keystore container.
 *                      Can't locate keystore.
 *                      Can't communicate with keystore.
 *                      Access-control failure.
 *                      Same domain and usage's key already exists.
 *                for file base:
 *                      sec_pvtkey_invalid_handle
 *                      sec_pvtkey_mechanism_not_init
```

```
 *                       sec_pvtkey_no_more_memory
 *                       sec_pvtkey_internal_error
 *                sec_pvtkey_same_domain_and_usage_key_already_exists.
 *
 * Input
 *
 *    handle: A pointer to an opaque handle to the private key
 *            context data.  The handle should be obtained through
 *            sec_pvtkey_open()

 *    pwd:  Pointer to the user's password.
 *    key_usage: The usage permitted of this keypair.  One key_usage
 *               at a time.
 *    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 increases 1 for each update.)
 *    private_key: Pointer to the encoded private key
 *                 structure.
 *    public_key: Pointer to the encoded public key structure.
 *
 */

error_status_t sec_pvtkey_store(
        [in]  sec_pk_mechanism_handle_t handle,
        [in]  char                      *pwd,
        [in]  sec_pk_usage_flags_t      key_usage,
        [in]  unsigned32                key_vno,
        [in]  sec_pk_data_t             *private_key,
        [in]  sec_pk_data_t             *public_key
   );

/* sec_pvtkey_delete
 *
 * Delete the user's keypair from the storage device.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  Can't initialize keystore container.
 *                      Can't locate keystore.
 *                      Can't communicate with keystore.
 *                      Access-control failure.
 *                 for file base:
 *                      sec_pvtkey_key_deletion_not_supported
 *
 * Input
 *
 *    handle: A pointer to an opaque handle to the private key
 *            context data.  The handle should be obtained through
 *            sec_pvtkey_open()
 *    pwd:    Pointer to the user's password.
 *    key_usage: The usage of the key pair of the desired key. One
 *               key_usage at a time or NULL key usage.
 *               NULL value means "delete all the key_usages
```

```
 *                related to the user.
 *    key_vno: The version number of the desired key.  If 0 (zero) is
 *             specified, the function will select the most recent
 *             key version number.
 *
 */

error_status_t sec_pvtkey_delete(
        [in]  sec_pk_mechanism_handle_t handle,
        [in]  char                      *pwd,
        [in]  sec_pk_usage_flags_t      key_usage,
        [in]  unsigned32                key_vno
   );

/* sec_pvtkey_generate
 *

 * Generate a new keypair and store it in the keystore encrypted
 * using the user's password in the storage service.  The encryption
 * mechanism is determined by the service provider.  The keys will
 * be indexed by name, domain_id, key_usage and key_version in the
 * private key storage service in order to allow retrieval via the
 * sec_pvtkey_get API.
 *
 * If the specified key already exists in the keystore, the routine
 * will leave it unchanged and return a failure status.  To replace
 * a key in a keystore, either use sec_pvtkey_update or
 * sec_pvtkey_update_generate, or call sec_pvtkey_delete followed by
 * sec_pvtkey_store.
 *
 * A keystore may enforce its own access-control requirements, and
 * an access-control failure will be indicated by an appropriate
 * status code, with the contents of the keystore unchanged.  For
 * example, a file-based keystore implementation may require that
 * the caller has write-access to the keystore file; a server-based
 * implementation may require that the caller has appropriate
 * ACL-based permission to create a new entry; a memory-card
 * implementation may require that the card be write-enabled.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  Can't initialize keystore container.
 *                      Can't locate keystore.
 *                      Can't communicate with keystore.
 *                      Access-control failure.
 *                      Same domain and usage's key already exits.
 *                      Key generation not supported.
 *   for file base:
 *                      sec_pvtkey_key_generation_not_supported
 *
 * Input
 *
 *     handle: A pointer to an opaque handle to the private key
 *             context data.  The handle should be obtained through
 *             sec_pvtkey_open()
```

```
 *     pwd:    Pointer to the user's password.
 *     alg_id: Desired public-key algorithm.  If specified as
 *             NULL, the default public-key algorithm (RSA)
 *             shall be used.
 *     key_length: Desired key-length.  The length of the desired
 *                 key.  Interpretation of this parameter is
 *                 algorithm- dependent; For RSA, it shall be
 *                 interpreted as the bit-length of the key.  For
 *                 any algorithm, a value of 0xffffffff shall mean
 *                 the algorithm-specific default.
 *     key_usage: The usage of the public key this key pair
 *                 shall belongs to.
 *     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 increases 1 for each update.)
 *
 * Output
 *
 *     public_key: Pointer to the encoded public key structure of the
 *     newly-generated key.
 *
 */

error_status_t sec_pvtkey_generate(
        [in]  sec_pk_mechanism_handle_t handle,
        [in]  char                      *pwd,
        [in]  sec_pk_algorithm_id_t     *alg_id,
        [in]  unsigned32                key_length,
        [in]  sec_pk_usage_flags_t      key_usage,
        [in]  unsigned32                key_vno,
        [out] sec_pk_data_t             *public_key
   );

/* sec_pvtkey_get
 *
 * Retrieve the user's keypair from the storage device.
 * The routine allocates memory for the data returned in the
 * private key and public key parameters.  Users should call
 * sec_pvtkey_free() to deallocate that memory.
 *
 * The keypair returned will be the first one found with a
 * stored key_usage value containing the input "key_usage" value.
 * i.e. the input value (which will typically have a single bit
 * set) will be "AND"d with the stored value (which may have many
 * bits set), and the will be returned if the result is not 0.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  Access control failure.
 *                      Can't locate keystore.
 *                      Can't communicate with keystore.
 *                      Can't find specified key.
 *                      Invalid password.
```

```
 *                for file base:
 *                      sec_pvtkey_invalid_handle
 *                      sec_pvtkey_mechanism_not_init
 *                      sec_pvtkey_internal_error
 *                      sec_pvtkey_invalid_password
 *                      sec_pvtkey_multiple_key_usages
 *
 * Input
 *
 *    handle: A pointer to an opaque handle to the private key
 *            context data.  The handle should be obtained through
 *            sec_pvtkey_open()
 *    pwd:  Pointer to the user's password.
 *    domain_id: The application domain of the desired key.
 *    key_usage: The usage of the public key of the desired key. One
 *               key_usage at a time.
 *
 * Input/Output
 *
 *    key_vno: The key version number. (0 may be specified to
 *             indicate that the routine should select the
 *             newest key version number).  The return value is
 *             the key version of the key returned.
 *
 * Output
 *
 *    private_key: Pointer to the encoded private key
 *                 structure.
 *    public_key   Pointer to the encoded public key
 *                 structure.
 *
 */

error_status_t sec_pvtkey_get(
       [in]  sec_pk_mechanism_handle_t handle,
       [in]  char                      *pwd,
       [in]  sec_pk_usage_flags_t      key_usage,
       [in, out]  unsigned32           *key_vno,
       [out] sec_pk_data_t             *private_key,
       [out] sec_pk_data_t             *public_key
   );

/* sec_pvtkey_update
 *
 * Update the user's keypair record in a keystore.  This API may be
 * used to change either the password, or the keypair, or both.
 * It may be invoked either by the user to change her own
 * keypair record (in which case the current_pwd parameter must
 * be supplied), or by a system administrator to change another
 * user's keypair record (in which case the current_pwd parameter
 * should be specified as NULL.
 *
 * Keystore implementations may impose their own access-control
 * requirements.  In particular, some keystore implementations
 * may not support administrative updates, or may require that
```

```
 * the caller have valid administrative DCE credentials.
 *
 * Return status
 *   error_status_ok:   Success.
 *   other (non-zero):  Access control failure.
 *                      Can't locate keystore.
 *                      Can't communicate with keystore.
 *                      Can't find specified record.
 *                      Invalid password.
 *                 for file base:
 *                      sec_pvtkey_invalid_handle
 *                      sec_pvtkey_mechanism_not_init
 *                      sec_pvtkey_private_key_is_not_supplied
 *                      sec_pvtkey_new_password_required
 *                      sec_pvtkey_no_more_memory
 *                      sec_pvtkey_internal_error
 *                      sec_pvtkey_no_matched_private_key
 *                      sec_pvtkey_Invalid_password.
 *
 * Input
 *
 *    handle: A pointer to an opaque handle to the private key
 *            context data.  The handle should be obtained through
 *            sec_pvtkey_open()
 *    current_pwd: Pointer to the user's current password.  Used to
 *                 authenticate the user when changing her own
 *                 password or keypair.  If this parameter is NULL,
 *                 the keystore will assume that the update is an
 *                 administrative action, and will enforce
 *                 administrative access-control
 *    new_pwd: Pointer to the user's new password.  If the user
 *             is changing her own key, and does not wish to change
 *             the associated password, this argument may be NULL
 *             (subject to any restrictions imposed by the keystore).
 *             If this is an administrative action, a valid new_pwd
 *             must be supplied.
 *    key_usage: The allowed usage for this keypair.  One
 *               key_usage at a time.
 *    private_key: Pointer to the encoded private key structure.
 *    public_key:  Pointer to the encoded public key structure.
 *                 If the user is changing her own password, and does
 *                 not wish to change the corresponding keypair, this
 *                 parameter may be NULL (subject to any restrictions
 *                 imposed by the keystore).  If this is an
 *                 administrative action, valid private and public
 *                 keys must be supplied.
 *
 * Output
 *    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 increases 1 for each update.)
 *
 */
```

```
error_status_t sec_pvtkey_update(
        [in] sec_pk_mechanism_handle_t handle,
        [in] char                     *current_pwd,
        [in] char                     *new_pwd,
        [in] sec_pk_usage_flags_t   key_usage,
        [out] unsigned32             *key_vno,
        [in] sec_pk_data_t           *private_key,
        [in] sec_pk_data_t           *public_key
  );

/* sec_pvtkey_update_generate
 *
 * Update the user's keypair record for a keystore that supports
 * key generation.  This API may be used to change
 * either the keypair alone, or both the keypair and the password.
 * It may be invoked either by the user to change her own keypair
 * record (in which case the current_pwd parameter must be supplied),
 * or by a system administrator to change another user's keypair
 * record (in which case the current_pwd parameter should be
 * specified as NULL.
 *
 * Keystore implementations may impose their own access-control
 * requirements.  In particular, some keystore implementations
 * may not support administrative updates, or may require that
 * the caller have valid administrative DCE credentials.
 *
 *
 * Return status
 *   error_status_ok:    Success.
 *   other (non-zero):  Access control failure.
 *                      Can't locate keystore.
 *                      Can't communicate with keystore.
 *                      Can't find specified record.
 *                      Key generation not supported.
 *                      Invalid password.
 *                  for file base:
 *                      sec_pvtkey_key_generation_not_supported
 *
 * Input
 *
 *    handle: A pointer to an opaque handle to the private key
 *            context data.  The handle should be obtained through
 *            sec_pvtkey_open()
 *    current_pwd: Pointer to the user's current password.
 *                 Used to authenticate the user when changing her
 *                 own password or keypair.  If this parameter is
 *                 NULL, the keystore will assume that the update is
 *                 an administrative action, and will enforce
 *                 administrative access-control
 *    new_pwd: Pointer to the user's new password.  If the user
 *             is changing her own key, and does not wish to change
 *             the associated password, this argument may be NULL
 *             (subject to any restrictions imposed by the keystore).
 *             If this is an administrative action, a valid new_pwd
 *             must be supplied.
```

```
 *     alg_id: Desired public-key algorithm.  If specified as
 *             NULL, the default public-key algorithm (RSA)
 *             shall be used.
 *     key_length: Desired key-length.  The length of the desired
 *                 key.  Interpretation of this parameter is
 *                 algorithm-dependent; For RSA, it shall be
 *                 interpreted as the bit-length of the key.  For any
 *                 algorithm, a value of 0xffffffff shall mean the
 *                 algorithm-specific default.

 *     key_usage: The allowed usage for this keypair.  One at a time.
 *
 *
 * Output
 *
 *    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 increases 1 for each update.)
 *    public_key:  The newly-generated public-key.
 *
 */

error_status_t sec_pvtkey_update_generate(
        [in] sec_pk_mechanism_handle_t handle,
        [in] char                      *current_pwd,
        [in] char                      *new_pwd,
        [in] sec_pk_algorithm_id_t     *alg_id,
        [in] unsigned32                key_length,
        [in] sec_pk_usage_flags_t      key_usages,
        [out] unsigned32               *key_vno,
        [out] sec_pk_data_t            *public_key
  );

}
```

9.3.2. security/psm/domestic/sec_pvtkey_switch.h

```
/*
 * @OSF_COPYRIGHT@
 * COPYRIGHT NOTICE
 * Copyright (c) 1996 Open Software Foundation, Inc.
 * ALL RIGHTS RESERVED (DCE).  See the file named COPYRIGHT.DCE
 *  in the src directory for the full copyright text.
 */
/*
 * (c) Copyright 1996,
 * Digital Equipment Corporation, Maynard, Massachusetts, USA
 * All Rights Reserved
 */
/*
 * Copyright (c) Hewlett-Packard Company 1996
 * Unpublished work.  All Rights Reserved.
 */
```

```
#ifndef SEC_PVTKEY_SWITCH_H_
#define SEC_PVTKEY_SWITCH_H_

#ifdef __cplusplus
extern "C" {
#endif

#include <dce/sec_pk_base.h>
#include <dce/sec_pvtkey.h>
#include <dce/dcepsmmsg.h>


#define SEC_PVTKEY_SWITCH_FLAG_MATCHALL     1
#define SEC_PVTKEY_SWITCH_FLAG_TERMINATE    2
#define SEC_PVTKEY_SWITCH_FLAG_ENABLED      4

typedef struct {

    unsigned32 magic;

    unsigned32 flags;

    const uuid_t * keystore_uuid;

    char keystore_name[32]; /* Name must be 32 characters or less, */
                            /* and NULL-terminated if less.        */

    error_status_t (* open)(idl_char *name,
                            sec_pk_domain_t *domain_id,
                            sec_pk_mechanism_handle_t *handle);

    error_status_t (* close)(sec_pk_mechanism_handle_t handle);

    error_status_t (* capabilities)(sec_pk_mechanism_handle_t handle,
                                  idl_void_p_t capability_parameter,
                                  sec_pk_pvtkey_capability * feature,
                                  idl_boolean * supported);

    error_status_t (* store)(sec_pk_mechanism_handle_t handle,
                             idl_char *pwd,
                             sec_pk_usage_flags_t key_usages,
                             unsigned32 key_vno,
                             sec_pk_data_t *private_key,
                             sec_pk_data_t *public_key);

    error_status_t (* delete)(sec_pk_mechanism_handle_t handle,
                              idl_char *pwd,
                              sec_pk_usage_flags_t key_usages,
                              unsigned32 key_vno);

    error_status_t (* generate)(sec_pk_mechanism_handle_t handle,
                                idl_char *pwd,
                                sec_pk_algorithm_id_t *alg_id,
                                unsigned32 key_length,
                                sec_pk_usage_flags_t key_usage,
```

```
                                 unsigned32 key_vno,
                                 sec_pk_data_t *public_key);

    error_status_t (* get)(sec_pk_mechanism_handle_t handle,
                            idl_char *pwd,
                            sec_pk_usage_flags_t key_usage,
                            unsigned32 *key_vno,
                            sec_pk_data_t *private_key,
                            sec_pk_data_t *public_key);

    error_status_t (* update)(sec_pk_mechanism_handle_t handle,
                               idl_char *current_pwd,
                               idl_char *new_pwd,
                               sec_pk_usage_flags_t key_usages,
                               unsigned32 *key_vno,
                               sec_pk_data_t *private_key,
                               sec_pk_data_t *public_key);

   error_status_t (* update_generate)(
                                     sec_pk_mechanism_handle_t handle,
                                     idl_char *current_pwd,
                                     idl_char *new_pwd,
                                     sec_pk_algorithm_id_t *alg_id,
                                     unsigned32 key_length,
                                     sec_pk_usage_flags_t key_usages,
                                     unsigned32 *key_vno,
                                     sec_pk_data_t *public_key);

} keystore_epv;

/*
 *  s e c _ p v t k e y _ _ g e t _ u u i d _ e r a _ b y _ u u i d
 *
 * Retrieve a UUID-valued ERA from a specific principal entry in the
 * registry.  The desired ERA is specified by its UUID.
 */

unsigned32 sec_pvtkey__get_uuid_era_by_uuid(
                                  sec_rgy_name_t principal_name,
                                  const uuid_t * uuid,
                                  uuid_t * era_value);

/*
 * s e c _ p v t k e y _ _ l o o k u p _ k e y s t o r e
 *
 * Lookup the keystore associated with the specified user's DCE login
 * key.  Returns a pointer to the internal EPV for that keystore, or
 * NULL if the keystore was not found.
 */

extern unsigned32 sec_pvtkey__lookup_keystore(
                                   const char * username,
                                   const sec_pk_domain_t * domain_id,
                                   const keystore_epv ** keystore);
/*
```

```
 * s e c _ p s m _ _ i n i t _ s v c
 *
 * Initialize PSM serviceability messaging.
 * This routine might be better moved elsewhere; however currently
 * the sec_pvtkey_XXX routines are the only ones that use PSM
 * serviceability messages, so for the moment leave it here.
 *
 * The routine is threadsafe, and can be invoked multiple times
 * without ill effects.
 *
 */
extern void sec_psm__init_svc(void);

/*
 * s e c _ p v t k e y _ _ r e g i s t e r _ k e y s t o r e
 *
 * Register a new keystore implementation for the current image.
 * Keystores are registered either statically (for all images) via
 * sec_pvtkey_registered_keystores.c, or dynamically (for the current
 * image only) by calling this routine.  All parameters are input.
 * The routine adds the new keystore implementation to the list of
 * keystores.  The implementation of this list is fairly limited:
 *
 * 1.  There is a fixed-length list of keystores, and attempts to
 *     register more keystores than will fit in the list will return
 *     an error
 * 2.  Keystore implementations cannot be deleted from the list.
 *     However, apart for releasing the slot within the keystore
 *     list, the effect of keystore deletion can be achieved by
 *     re-registering the keystore you want to delete, setting
 *     replace = true and enabled = false.
 *
 */

extern unsigned32 sec_pvtkey__register_keystore(
    uuid_t * keystore_uuid, /* The UUID to which this pointer
                               points mustn't disappear! */
    const char * keystore_name,
                            /* Name should be 32 characters or less.
                               Used in serviceability messages.  This
                               is copied into the keystore list, so
                               it is OK for it to disappear after the
                               register_keystore call returns. */
    boolean32 enabled,      /* If true, the keystore will be enabled.
                               If false, the keystore will be
                               disabled. */
    boolean32 make_default, /* If true, this keystore will become
                               the new default keystore, to be
                               used if the registry doesn't tell
                               use which one to use. */
    boolean32 replace,      /* If true, this keystore will replace
                               any pre-existing keystore(s) with the
                               same UUID. If false, attempts to
                               re-register an already-existing
                               keystore will fail. */
```

```
    error_status_t (* open)(idl_char *name,
                            sec_pk_domain_t *domain_id,
                            sec_pk_mechanism_handle_t *handle),
    error_status_t (* close)(sec_pk_mechanism_handle_t handle),
    error_status_t (* capabilities)(
                                  sec_pk_mechanism_handle_t handle,
                                  idl_void_p_t capability_parameter,
                                  sec_pk_pvtkey_capability * feature,
                                  idl_boolean * supported),
    error_status_t (* store)(sec_pk_mechanism_handle_t handle,
                             idl_char *pwd,
                             sec_pk_usage_flags_t key_usages,
                             unsigned32 key_vno,
                             sec_pk_data_t *private_key,
                             sec_pk_data_t *public_key),
    error_status_t (* delete)(sec_pk_mechanism_handle_t handle,
                              idl_char *pwd,
                              sec_pk_usage_flags_t key_usages,
                              unsigned32 key_vno),
    error_status_t (* generate)(sec_pk_mechanism_handle_t handle,
                                idl_char *pwd,
                                sec_pk_algorithm_id_t *alg_id,
                                unsigned32 key_length,
                                sec_pk_usage_flags_t key_usage,
                                unsigned32 key_vno,
                                sec_pk_data_t *public_key),
    error_status_t (* get)(sec_pk_mechanism_handle_t handle,
                           idl_char *pwd,
                           sec_pk_usage_flags_t key_usage,
                           unsigned32 *key_vno,
                           sec_pk_data_t *private_key,
                           sec_pk_data_t *public_key),
    error_status_t (* update)(sec_pk_mechanism_handle_t handle,
                              idl_char *current_pwd,
                              idl_char *new_pwd,
                              sec_pk_usage_flags_t key_usages,
                              unsigned32 *key_vno,
                              sec_pk_data_t *private_key,
                              sec_pk_data_t *public_key),
    error_status_t (* update_generate)(
                                    sec_pk_mechanism_handle_t handle,
                                    idl_char *current_pwd,
                                    idl_char *new_pwd,
                                    sec_pk_algorithm_id_t *alg_id,
                                    unsigned32 key_length,
                                    sec_pk_usage_flags_t key_usages,
                                    unsigned32 *key_vno,
                                    sec_pk_data_t *public_key)
);

#ifdef __cplusplus
}
#endif

#endif
```

9.4. DCE Security Login API

The existing DCE Security Login API in file security/idl/sec_login.idl has changed to return additional error status values for functions that support public key operations or protocols. Those changes are listed here. No other changes have been made to the login APIs.

```
*    sec_pk_e_domain_unsupported      - kdc_cache domain not
*                                       supported by sec_pubkey
*    sec_pk_e_usage_unsupported       - a private or public key of
*                                       the required type was not
*                                       located in the PSM
*    sec_pk_e_key_attr_read_failed    - unable to read principal's
*                                       DCEPKAuthentication or
*                                       DCEPKKeyEncipherment ERA
*    sec_pk_e_no_memory               - out of memory
*    sec_pk_e_cell_too_long           - the cell name is too long for
*                                       the public key API
*/
boolean32 sec_login_validate_identity (


*    sec_pk_e_usage_unsupported       - a private or public key of
*                                       the required type was not
*                                       located in the PSM
*    sec_pk_e_key_attr_read_failed    - unable to read principal's
*                                       DCEPKAuthentication or
*                                       DCEPKKeyEncipherment ERA
*    sec_pk_e_no_memory               - out of memory
*    sec_pk_e_cell_too_long           - the cell name is too long for
*                                       the public key API
*/
boolean32 sec_login_valid_and_cert_ident (


*    sec_pk_e_usage_unsupported       - a private or public key of
*                                       the required type was not
*                                       located in the PSM
*    sec_pk_e_key_attr_read_failed    - unable to read principal's
*                                       DCEPKAuthentication or
*                                       DCEPKKeyEncipherment ERA
*    sec_pk_e_no_memory               - out of memory
*    sec_pk_e_cell_too_long           - the cell name is too long for
*                                       the public key API
*/
boolean32 sec_login_validate_first (
```

9.5. sec.sams

Error status values generated by the sec_pubkey interface are created using the SAMS utility from the sec.sams file. Application programs can access the error codes by including the secsts.h file generated by SAMS.

Additions made to the security/idl/sec.sams file follow.

A new sub-component was added for public key errors:

```
sub-component sec_s_pubkey "public_key" sec_i_svc_pubkey

start           !intable undocumented
code            sec_i_svc_pubkey
text            "Public key operations"
end
```

The following new status codes were added for public key errors:

```
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Start of public key messages           %
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

# Files : sec_pubkey.c

start
code            sec_pk_e_domain_unsupported
text            "Public key domain not supported for this
                 operation."
explanation     "The specified public key domain is not supported
                 for this operation."
action          "Login from another node with support for the
                 DCE Login domain; or use a valid public key
                 domain; or use a valid operation for the
                 specified domain."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_usage_unsupported
text            "Public key usage type not supported."
explanation     "The specified public key usage type is not
                 supported."
action          "Login from another node with support for the
                 user's public key login keys, or use a
                 supported public key usage type."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_key_vers_unsupported
text            "Key version not supported."
explanation     "For sec_pubkey_store, the key version must be
                 between 1 and 255; for sec_pubkey_retrieve,
                 the version number must be between 0 and 255."
action          "Call sec_pubkey interfaces with kvno set to
                 valid value."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end
```

```
start
code            sec_pk_e_key_vers_unavailable
text            "Key version requested not found."
explanation     "Specified key version was not found."
action          "Request available key using version 0, or
                 request version of key that exists."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_key_attr_read_failed
text            "Public key attribute read failed."
explanation     "The value of at least one public key was not
                 read successfully."
action          "Make sure the principal has a public key
                 (DCEPKAuthentication ERA or DCEPKKeyEncipherment
                 ERA) attached; make sure sec.1.2.2 has been
                 enabled in the principal's cell; make sure the
                 DCEPK ERA schema entries have not been
                 modified."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_no_memory
text            "Insufficient memory for public key operation."
explanation     "Insufficient memory for public key operation."
action          "Install more memory."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_cell_too_long
text            "target cell name too long for cache file"
explanation     "Attempting to retrieve the public key for the
                 KDC of a cell whose cell name is too long to
                 create a cache file name."
action          "Configure and use a shorter cell name alias."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_no_key_value
text            "the public key value is NULL"
explanation     "Attempting to store a public key whose value
                 is NULL."
action          "Use a valid public key value."
```

```
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_kf_param_not_alloc
text            "Return parameters NULL"
explanation     "The value of a return parameter was null.
                 Return parameters must point to pre-allocated
                 structures."
action          "Allocate the structure for the return value and
                 pass its address to the function."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_kf_open_err
text            "Error opening the keyfile."
explanation     "Specified keyfile could not be opened."
action          "Create keyfile or supply correct keyfile
                 pathname."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_kf_read_err
text            "Error reading the keyfile."
explanation     "Specified keyfile could not be read."
action          "Re-create keyfile or supply correct keyfile
                 pathname; verify permissions on keyfile."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_kf_write_err
text            "Error writing the keyfile."
explanation     "Specified keyfile could not be written."
action          "Supply correct keyfile pathname; create path
                 leading to keyfile; verify permissions on
                 keyfile."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_kf_format_err
text            "Keyfile not recognized."
```

```
explanation     "The keyfile is not a valid public or private
                 key keyfile."
action          "Check file pathname; be sure file is created
                 with correct format (header, version)."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_kf_version_err
text            "Keyfile version not supported."
explanation     "The keyfile version is not supported."
action          "Create keyfile with current software."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

start
code            sec_pk_e_rsdb_keystore_err
text            "Failed to register the rsdb keystore."
explanation     "Error registering the private key keystore
                 mechanism for the krbtgt key."
action          "Ensure libdce is compatible with secd."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# End of public key messages               %
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

start
code            sec_rsdb_acct_bad_keytype
text            "Only DES keys allowed for this account"
explanation     "An attempt was made to set a non-default keytype
                 for a principal where only DES keys are allowed.
                 Only the krbtgt principal is allowed to have
                 non-default keytypes."
action          "Use a DES keytype."
sub-component   sec_s_rs_rsdb
attributes      svc_c_sev_error
engineer        "Anne Anderson"
end

start
code            sec_pk_login_client_fail
text            "Could not acquire public key credentials: %s in %s"
notes           "first string is error message; second is code
                 location where error detected"
explanation     "A failure was detected while the client was
                 attempting to obtain credentials using public
                 key login protocols"
```

```
action          "Make a public key available to the client and
                 known to the KDC"
sub-component   sec_s_pubkey
attributes      svc_c_sev_notice_verbose
engineer        "Bill Sommerfeld"
end

# Files : rs_reserved.c
start
code            sec_res_pk_attr_sch_add_err
text            "Error adding an entry to the PK attribute
                 schema"
explanation     "?"
action          "?"
sub-component   sec_s_rs
attributes      svc_c_sev_error
end
```

9.6. psm.sams

Error status values generated by the `sec_psm` interface are created using the `SAMS` utility from the `psm.sams` file. Application programs can access the error codes by including the `dcepsmmsg.h` file generated by `SAMS`.

```
# @OSF_COPYRIGHT@
# COPYRIGHT NOTICE
# Copyright (c) 1990, 1991, 1992, 1993, 1994, 1996 Open Software
#   Foundation, Inc.
# ALL RIGHTS RESERVED (DCE).  See the file named COPYRIGHT.DCE
# for the full copyright text.
#
# All Rights Reserved
#
# Copyright (c) Hewlett-Packard Company 1996
# Unpublished work. All Rights Reserved.
#

component      psm
table          psm__table
technology     dce

serviceability table psm_svc_table handle psm_svc_handle
start
  sub-component psm_s_user     "psm_user"     psm_i_svc_user
  sub-component psm_s_bsafe    "psm_bsafe"    psm_i_svc_bsafe
  sub-component psm_s_pvtkey   "psm_pvtkey"   psm_i_svc_pvtkey
  sub-component psm_s_psm      "psm_psm"      psm_i_svc_psm
  sub-component psm_s_general  "psm_general"  psm_i_svc_general
end

#
## Message for RSA BSAFE 2.1 error code
#

start
code            sec_s_bsafe_algorithm_already_set
text            "BSAFE2: Algorithm object has already been set
```

```
                 with algorithm info"
explanation     "Algorithm object has already been set with
                 algorithm info"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_algorithm_info
text            "BSAFE2: Invalid algorithm info format"
explanation     "Invalid algorithm info format"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_algorithm_not_initialized
text            "BSAFE2: Algorithm object has not been
                 initialized"
explanation     "Algorithm object has not been initialized"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_algorithm_not_set
text            "Algorithm object has not been set with algorithm
                 info"
explanation     "Algorithm object has not been set with algorithm
                 info"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_algorithm_obj
text            "BSAFE2: Invalid algorithm object"
explanation     "Invalid algorithm object"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_alg_operation_unknown
text            "BSAFE2: Unknown operation for an algorithm or
                 algorithm info type"
explanation     "Unknown operation for an algorithm or algorithm
                 info type"
action          "?"
sub-component   psm_s_bsafe
```

```
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_alloc
text            "BSAFE2: Insufficient memory"
explanation     "Insufficient memory"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_cancel
text            "BSAFE2: Operation was canceled by the surrender
                 function"
explanation     "Operation was canceled by the surrender function"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_data
text            "BSAFE2: Generic data error"
explanation     "Generic data error"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_exponent_even
text            "BSAFE2: Public exponent in key pair generation
                 is not allowed to be even"
explanation     "Invalid even value for public exponent in key
                 pair generation"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_exponent_len
text            "BSAFE2: Invalid exponent length for public
                 exponent in key pair generation"
explanation     "Invalid exponent length for public exponent in
                 key pair generation"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_hardware
```

```
text            "BSAFE2: Cryptographic hardware error"
explanation     "Cryptographic hardware error"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_input_data
text            "BSAFE2: Invalid format for input data"
explanation     "Invalid format for input data"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_input_len
text            "BSAFE2: Invalid length for input data"
explanation     "Invalid length for input data"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_key_already_set
text            "BSAFE2: Key object has already been set with key
                 info"
explanation     "Key object has already been set with key info"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_key_info
text            "BSAFE2: Invalid key info format"
explanation     "Invalid key info format"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_key_len
text            "BSAFE2: Invalid key length"
explanation     "Invalid key length"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_key_not_set
```

```
text            "BSAFE2: Key object has not been set with key
                 info"
explanation     "Key object has not been set with key info"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_key_obj
text            "BSAFE2: Invalid key object"
explanation     "Invalid key object"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_key_operation_unknown
text            "BSAFE2: Unknown operation for a key info type"
explanation     "Unknown operation for a key info type"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_memory_obj
text            "BSAFE2: Invalid internal memory object"
explanation     "Invalid internal memory object"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_modulus_len
text            "BSAFE2: Invalid modulus length in public or
                 private key"
explanation     "Invalid modulus length in public or private key"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_not_initialized
text            "BSAFE2: Algorithm is not properly initialized"
explanation     "Algorithm is not properly initialized"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
```

```
code            sec_s_bsafe_not_supported
text            "BSAFE2: Unsupported operation requested"
explanation     "Unsupported operation requested"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_output_len
text            "BSAFE2: Output data is larger than supplied
                 buffer"
explanation     "Output data is larger than supplied buffer"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_over_32k
text            "BSAFE2: Data block exceeds 32,767 bytes"
explanation     "Data block exceeds 32,767 bytes"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_random_not_initialized
text            "BSAFE2: Random algorithm has not been
                 initialized"
explanation     "Random algorithm has not been initialized"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_random_obj
text            "BSAFE2: Invalid algorithm object for the random
                 algorithm"
explanation     "Invalid algorithm object for the random
                 algorithm"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_signature
text            "BSAFE2: Invalid signature"
explanation     "Invalid signature"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
```

```
end

start
code            sec_s_bsafe_wrong_algorithm_info
text            "BSAFE2: Wrong type of algorithm info"
explanation     "Wrong type of algorithm info"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_wrong_key_info
text            "BSAFE2: Wrong type of key info"
explanation     "Wrong type of key info"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_input_count
text            "BSAFE2: Update called an invalid number of
                 times for inputting data"
explanation     "Update called an invalid number of times for
                 inputting data"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_method_not_in_chooser
text            "Algorithm method required by the specified
                 algorithm not in algorithm chooser"
explanation     "Algorithm method required by the specified
                 algorithm not in algorithm chooser"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_output_count
text            "BSAFE2: Update called an invalid number of
                 times for outputting data"
explanation     "Update called an invalid number of times
                 for outputting data"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_unknown_error
```

```
text            "BSAFE2: Unknown bsafe error"
explanation     "Unknown bsafe error"
action          "?"
sub-component   psm_s_bsafe
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_decryption_failure
text            "BSAFE2: decryption failure"
explanation     "Wrong password or key supplied in decryption or
                 verification"
action          "Get the right password or key"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_s_bsafe_mod_size_out_of_range
text            "BSAFE2: key modulus length should be between 256
                 - 1024"
explanation     "key modulus length is too small or too big for
                 public key generation"
action          "Get the right modulus length"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end



#
## private key accessing mechanism errors.
#

start
code            sec_pvtkey_mechanism_not_init
text            "private key storage mechanism hasn't been
                 initialized"
explanation     "sec_pvtkey_open was not called before this
                 operation."
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_invalid_handle
text            "invalid sec_pvtkey_handle"
explanation     "Pvtkey handle is invalid, could be caused by
                 memory corruption."
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end
```

```
start
code            sec_pvtkey_same_domain_and_usage_key_already_exist
text            "same domain and usage key already exist"
explanation     "There is a private key with same domain, usage
                 existent in the key storage mechanism"
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code         sec_pvtkey_same_domain_and_usage_key_exist_with_diff_pwd
text            "Unable to update
                 same_domain_and_usage_key_already_exist key
                 because of different pwd"
explanation     "Either recreate the user account or use the
                 right password."
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_no_matched_private_key
text            "couldn't find a matched private key in the file"
explanation     "No private key is matched to the requested name,
                 domain, and key usage"
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_key_generation_not_supported
text            "key generation is not supported"
explanation     "private key generation is done in psm level for
                 file based key storage"
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_invalid_pwd
text            "invalid password"
explanation     "illegal password for fetching private key."
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_no_more_memory
text            "ran out of memory"
explanation     "Malloc failed because of no more memory"
```

```
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_key_deletion_not_supported
text            "private key deletion functionality is not
                 supported."
explanation     "private key deletion functionality is not
                 supported."
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_private_key_is_not_supplied
text            "private key is required for admin to update
                 user's private key"
explanation     "For admin to update user's private key record,
                 private key has to supplied."
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_new_password_required
text            "new password is required for updating the
                 password"
explanation     "New password is required for updating the
                 password of the private key"
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_internal_error
text            "sec_pvtkey module internal error.  Please
                 inform the developer."
explanation     "This is a developer coding error."
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_file_locked
text            "pvtkey file is locked by other process."
explanation     "Only one process, one thread can do a write
                 access to the pvtkey file at a time"
action          "Wait a while and try it later"
sub-component   psm_s_pvtkey
```

```
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_file_cannot_open_key_file
text            "pvtkey file cannot be opened."
explanation     "pvtkey file cannot be opened."
action          "Check the permission of the file"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_privileged
text            "Not file owner or root"
explanation     "Must be the file owner or root account to
                 access pvtkey file"
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_capability_not_supported
text            "specified capability not supported"
explanation     "specified capability not supported in
                 this private key storage mechanism"
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_unsupported_capability_name
text            "specified capability name not supported"
explanation     "specified capability name not supported
                 in this private key storage mechanism"
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_multiple_key_usages
text            "too many key usages supplied"
explanation     "Multiple key usages supplied while only
                 one key usage allowed"
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_pvtkey_no_more_data
text            "read reach the end of file"
```

```
explanation     "No more data in the file"
action          "?"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_warning"
end


start
code            sec_pvtkey_wrong_passwd
text            "New passphrase doesn't match  other key pairs'
                 passphrase"
explanation     "The passphrases for all key pairs must be the
                 same"
action          "Change to the same passphrase and try it again"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_warning"
end

start
code            sec_pvtkey_version_not_exist
text            "The correct private key version not found"
explanation     "The correct private key version not found"
action          "Use version 0 to indicate using the most recent
                 one, or use the correct key version number"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_warning"
end

start
code            sec_pvtkey_old_pass_required
text            "Old passphrase required for this update"
explanation     "This is owner updating his key pair.  Old
                 passphrase is required to verify the passphrase"
action          "Supply the correct old passphrase"
sub-component   psm_s_pvtkey
attributes      "svc_c_sev_warning"
end

#
## psm
#

start
code            sec_psm_unable_to_open_pvtkey
text            "Unable to access private key storage mechanism"
explanation     "open a file or make a connection to private key
                 mechanism failed."
action          "?"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end


start
code            sec_psm_not_init
```

```
text            "PSM was not initialized"
explanation     "PSM need to be initialized before calling
                 any psm operations"
action          "Please call sec_psm_open before any other
                 psm operation"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_psm_invalid_handle
text            "Invalid psm handle"
explanation     "psm handle was corrupted"
action          "Check the memory corruption"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_psm_no_more_memory
text            "system ran out of memory"
explanation     "system ran our of memory during malloc"
action          "Increase the swap space or memory"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_psm_wrong_pub_key_type
text            "wrong public key type. The only supported
                 public key type is RSA_PKCS"
explanation     "The only supported public key type is RSA_PKCS"
action          "Use the right public key type"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_psm_internal_error
text            "psm internal error"
explanation     "psm internal error"
action          "Please inform engineer"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_psm_unsupported_algorithm_id
text            "unsupported crypto algorithm"
explanation     "unsupported crypto algorithm"
action          "Please choose supported one"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end
```

```
#
# Messages added by Digital
#

start
code            sec_psm_keystore_not_found
text            "Keystore not found"
explanation     "A keystore could not be found for the
                 specified principal"
action          "Register a keystore for the principal"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_psm_keystore_list_corrupt
text            "Keystore list corrupt"
explanation     "Internal error - the list of registered
                 keystores is corrupt"
action          "Submit a problem report"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end


start
code            sec_psm_invalid_keystore_handle
text            "Invalid keystore handle"
explanation     "The sec_pvtkey_mechanism_handle_t was corrupt"
action          "Check for an error in the calling program"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_psm_keystore_attr_read_failed
text            "Keystore ERA retrieval failed"
explanation     "An attempt to retrieve a principal's keystore
                 ERA failed"
action          "Register a keystore for the principal"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
code            sec_psm_keystore_already_registered
text            "Keystore already registered."
explanation     "The specified keystore is already registered"
action          "To change an existing keystore, set the replace
                 flag"
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

start
```

```
code            sec_psm_keystore_list_full
text            "No space for a new keystore"
explanation     "There is no space left in the keystore list"
action          "None."
sub-component   psm_s_psm
attributes      "svc_c_sev_error | svc_c_action_exit_bad"
end

#
# Digital debugging messages
#

start
code            sec_psm_i_lookup_keystore
text            "Entering sec_pvtkey__lookup_keystore(%s)"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug9
end

start
code            sec_psm_i_bad_domain
text            "sec_pvtkey__lookup_keystore(%s) failed - Bad
                 domain"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug8
end

start
code            sec_psm_i_no_prin_keystore_era
text            "sec_pvtkey__lookup_keystore(%s) - No principal
                 keystore ERA"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug8
end

start
code            sec_psm_i_no_def_keystore_era
text            "sec_pvtkey__lookup_keystore(%s) - No default
                 keystore ERA"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug8
end

start
code            sec_psm_i_bad_keystore_list
text            "Keystore entry %d is corrupt"
explanation     "Internal error - The specified entry in the
```

```
                 registered keystore list is invalid"
action          "Submit a problem report"
sub-component   psm_s_psm
attributes      svc_c_sev_error
end

start
code            sec_psm_i_def_keystore
text            "sec_pvtkey__lookup_keystore(%s) - Using
                 default keystore %d (%.32s)"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug8
end

start
code            sec_psm_i_no_keystore
text            "sec_pvtkey__lookup_keystore(%s) failed - No
                 appropriate keystore"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug8
end

start
code            sec_psm_i_era_lookup_failure
text            "sec_pvtkey__lookup_keystore(%s) failed - Bad ERA
                 lookup (0x%x)"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug8
end

start
code            sec_psm_i_chosen_keystore
text            "sec_pvtkey__lookup_keystore(%s) - Returning
                 keystore %d (%.32s)"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug8
end

start
code            sec_psm_i_matchall_keystore
text            "sec_pvtkey__lookup_keystore(%s) - Returning
                 matchall keystore %d (%.32s)"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug8
end
```

```
start
code            sec_psm_i_searching_explicit
text            "sec_pvtkey__lookup_keystore(%s) - Searching for
                 ERA-match keystore %s"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug9
end

start
code            sec_psm_i_searching_matchall
text            "sec_pvtkey__lookup_keystore(%s) - Searching for
                 matchall keystore"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug9
end

start
code            sec_psm_i_discarded_keystore
text            "sec_pvtkey__lookup_keystore(%s) - Discarded
                 keystore %d (%.32s, uuid %s)"
explanation     "Debugging message"
action          "None - Debugging message"
sub-component   psm_s_psm
attributes      svc_c_debug9
end

start
code            sec_psm_i_bad_keystore_id
text            "sec_pvtkey__lookup_keystore(%s) - Bad UUID in
                 keystore %d (%.32s)"
explanation     "Internal error - The specified entry in the
                 registered keystore list is invalid"
action          "Submit a problem report"
sub-component   psm_s_psm
attributes      svc_c_sev_error
end

start
code            sec_psm_i_bad_keystore_id1
text            "sec_pvtkey__register_keystore - Bad UUID in
                 keystore %d (%.32s)"
explanation     "Internal error - The specified entry in the
                 registered keystore list is invalid"
action          "Submit a problem report"
sub-component   psm_s_psm
attributes      svc_c_sev_error
end

#
## Messages for serviceability table
#
```

```
start           !intable undocumented
code            psm_i_svc_user
text            "PSM user operations0
end

start           !intable undocumented
code            psm_i_svc_bsafe
text            "BSAFE 2.1 (RSA) library operations"
end

start           !intable undocumented
code            psm_i_svc_pvtkey
text            "Private key accessing mechanism"
end

start           !intable undocumented
code            psm_i_svc_psm
text            "Personal security module framework"
end

start           !intable undocumented
code            psm_i_svc_general
text            "Personal security module general"
end
```

9.7. dhd.sams

Error status values generated by the `rsec_pk_cache` interface are created using the `SAMS` utility from the `dhd.sams` file. Application programs can access the error codes by including the `dcedhdmsg.h` file generated by `SAMS`.

The following new sub-component is defined:

```
sub-component dhd_s_pkcache "pkcache" dhd_i_svc_pkcache

start   !intable undocumented
code    dhd_i_svc_pkcache
text    "The dced KDC public key cache service"
end
```

The following new errors are defined:

```
# -------- pkcache-specific
start collection 13

start
code            dced_s_pkcache_usage_unsupp
text            "Public key usage type unsupported."
explanation     "sec_login has requested an invalid
                 usage type for a KDC key.  Usage type must be
                 sec_pk_usage_digitalSignature"
action          "Make sure libdce and dced are from the same
                 release."
attributes      svc_c_sev_error
sub-component   dhd_s_pkcache
end

start
```

```
code            dced_s_pkcache_key_write_failure
text            "Cache of KDC key failed."
explanation     "Some error occurred during the attempt to write
                 the public key of the KDC into the file cache."
action          "None."
attributes      svc_c_sev_error
sub-component   dhd_s_pkcache
end

start
code            dced_s_pkcache_cell_invalid
text            "cell name invalid"
explanation     "The cell name requested is too long to form
                 a valid krbtgt principal name."
action          "Create shorter cellname alias for target cell."
attributes      svc_c_sev_error
sub-component   dhd_s_pkcache
end

start
code            dced_s_pkcache_mkdir_error
text            "directory creation error"
explanation     "Unable to create a directory component
                 of the KDC cache file path."
action          "Check permissions on DCELOCAL/etc/security to
                 ensure root can create directories and files
                 there."
attributes      svc_c_sev_error
sub-component   dhd_s_pkcache
end

start
code            dced_s_pkcache_key_fresh
text            "public key cache is fresh"
explanation     "The cache file for the requested cell's krbtgt
                 public key has been updated within the past 5
                 minutes"
action          "Either wait five minutes and request update
                 again, or read cached key value using the
                 sec_pubkey_retrieve_kdc_key interface."
attributes      svc_c_sev_error
sub-component   dhd_s_pkcache
end
```

10. REMOTE INTERFACES

10.1. KDC Public Key Cache API

Not-yet-authenticated clients using the DCE Public Key Login protocol need access to a trusted copy of the public key of the KDC of the cell to which they belong. The following new remote interface has been added to `dced` to support retrieval of the KDC public key by an authenticated client (`dced`) on behalf of the unauthenticated client.

Applications should not invoke this interface directly, but rather go through the `sec_pubkey_retrieve()` API using the `sec_pk_domain_pk_kdc_cache` domain. This will attempt to read the public key from the cache file, and if that fails, will call `dced` to obtain the key.

If a key is successfully obtained using `sec_pubkey_retrieve()` and the `sec_pk_domain_pk_kdc_cache` domain, but the key obtained does not successfully verify the KDC signature on the returned PADATA, it may be due to the KDC public key pair having been changed since the public key was cached on the local node. In this case, login clients may choose to invoke the `sec_pubkey_retrieve()` API again, this time with the `sec_pk_domain_pk_kdc_cache_refresh` domain. This API will first verify that the cached KDC key is at least 5 minutes old, and if so will invoke the `rsec_pk_cache_kdc_key()` remote interface to obtain a fresh cache copy of the KDC key.

```
/*
 * @OSF_COPYRIGHT@
 */
/*
 * Copyright (c) Hewlett-Packard Company 1996
 * Unpublished work. All Rights Reserved.
 *
 * rsec_pk_cache.idl - Remote request for dced to cache or
 * refresh the public key of a KDC
 *
 */

[
    uuid(d9a38608-688d-11cf-ae3c-08000919ebb5),
    pointer_default(ptr),
    version(1.1)
]

interface rsec_pk_cache {

import "dce/rgynbase.idl";
import "dce/sec_pk_base.idl";

/* r s e c _ p k _ c a c h e _ k d c _ k e y
 *
 * rsec_pk_cache_kdc_key - this routine requests dced to
 * obtain and cache, or refresh cache, of public key of
 * the KDC of a specified cell.  The key is cached in
 * the file ${KDC_PUBKEY_CACHE_DIR}/<cell>/${KDC_PUBKEY_CACHE_FILE}
 * (KDC_PUBKEY_CACHE_DIR/FILE defined in security/h/rca_fileloc.h).
 *
 * Keys may be retrieved from this file via the
 * sec_pubkey_retrieve() interface with domain
 * sec_pk_domain_pk_kdc_cache.  The sec_pubkey interface invokes
 * the rsec_pk_cache_kdc_key interface if the cache is empty or
 * a refresh request is being made.
 *
 * rsec_pk_cache_kdc_key refuses to refresh an existing
 * cache file if it has been written within the past five
 * minutes.  This is intended to stall denial-of-service
 * attacks.
 *
 * In Parameters
 *   cell           - canonical cell name (/.../cell_name)
 *   usage          - must be "sec_pk_usage_digitalSignature" in the
 *                    reference implementation.
 *   kvno           - pointer to key version desired (or NULL)
```

```
 *
 * Out Parameters
 *   kvno           - pointer to key version returned (or NULL)
 *   public_key     - DER-encoded public key value
 *
 * Returns:
 *   dced_s_pkcache_cellname_invalid - bad cellname
 *   dced_s_pkcache_usage_unsupp - usage value unsupported
 *   dced_s_pkcache_key_fresh - key was cached recently
 *            (within KEY_CACHE_STALE_SECS)
 */
error_status_t rsec_pk_cache_kdc_key (
    [in]    handle_t                handle,
    [in]    sec_rgy_name_t          cell,
    [in]    sec_pk_usage_flags_t    usage,
    [in,out] unsigned32             *kvno,
    [out]   sec_pk_pubkey_t         *pubkey_p
);
}
```

10.2. krb.mvr

The DCE Public Key Login credentials are contained within the pre-authentication data fields of the Kerberos V5 `KRB_AS_REQ` and `KRB_AS_REP` protocol messages [IETF 1510]. The credentials are stored in pre-authentication structures of new types `KRB5_PADATA_ROOT_CERT` and `KRB5_PADATA_PUBLIC_REP`.

The credentials used in the DCE public key TGT acquisition protocol are defined in changes and additions to the existing file `security/krb5/lib/mvr/krb.mvr`, listed here.

> *NOTE: In the reference implementation, the* `mavros` *compiler is used to generate ASN.1 DER encode and DER/BER decode stubs for use within the* `sec_login` *and KDC implementations. For those constructs* `mavros` *is unable to handle, hand-generated stubs have been constructed.*

```
EncTicketPart ::=       [APPLICATION 3] SEQUENCE {
        flags[0]        TicketFlags,
        key[1]          EncryptionKey,
        crealm[2]       Realm,
        cname[3]        PrincipalName,
        transited[4]    TransitedEncoding,
        authtime[5]     KerberosTime,
        starttime[6]    KerberosTime OPTIONAL,
        endtime[7]      KerberosTime,
        renew-till[8]   KerberosTime OPTIONAL,
        caddr[9]        HostAddresses OPTIONAL,
        authorization-data[10]  AuthorizationData OPTIONAL
}

--
-- RFC 1510 (as of 6/10/96) currently marks the sname field of a
-- KDC-REQ-BODY as optional. DCE however, does not. Moreover, MIT's
-- current KRB5 implementation does not follow RFC 1510, which states
-- that the sname can only be absent when the ENC-TKT-IN-SKEY option
-- is specified. We have suggested changing the RFC to make sname a
-- mandatory field; this change will probably be in the next
-- revision.
--
```

```
--
-- additions from v5b6
-- !!! asn.1 is approximate..
--

ETYPE-INFO-ENTRY ::= SEQUENCE {
        etype[0]          INTEGER,
        salt[1]           OCTET STRING OPTIONAL
}

ETYPE-INFO ::= SEQUENCE OF ETYPE-INFO-ENTRY

PA-ENC-TS-ENC   ::= SEQUENCE {
        patimestamp[0]    KerberosTime, -- client's time
        pausec[1]         INTEGER OPTIONAL
}

--
-- Additions for Public Key Authentication --
--
PA-PK-AS-ROOT ::= SEQUENCE {
        rootCert[0]          SEQUENCE OF OCTET STRING,
        signedAuth[1]        SignedPKAuthenticator
}

SignedPKAuthenticator ::= SEQUENCE {
        authent[0]           PKAuthenticator,
        authentSig[1]        Signature
}

PKAuthenticator ::= SEQUENCE {
        cksum[0]             Checksum OPTIONAL,
        cusec[1]             INTEGER,
        ctime[2]             KerberosTime,
        nonce[3]             INTEGER,
        kdcRealm[4]          Realm,
        kdcName[5]           PrincipalName
}

Signature ::= SEQUENCE {
        sigType[0]           INTEGER,
        kvno[1]              INTEGER OPTIONAL,
        sigHash[2]           OCTET STRING
}
PA-PK-AS-REP ::= SEQUENCE {
        kdcCert[0]           SEQUENCE OF Certificate,
        encryptShell[1]      EncryptedData, -- EncPaPkAsRepPartShell
                                     -- encrypted by encReplyPartKey
        encryptKey[2]        EncryptedData  -- EncPaPkAsRepTmpKey
                                        -- encrypted by userPublicKey
}

EncPaPkAsRepPartShell ::= SEQUENCE {
        encReplyPart[0]      EncPaPkAsRepPart,
        encReplyPartSig[1]   Signature -- encReplyPart
```

```
                                        -- signed by kdcPrivateKey
}

EncPaPkAsRepPart ::= SEQUENCE {
        encReplyKey[0]      EncryptionKey,
        nonce[1]            INTEGER
}

EncPaPkAsRepTmpKey ::= SEQUENCE {
        encReplyTmpKey[0]   EncryptionKey
}

Certificate ::= SEQUENCE {
-- !!! large chunks temporary; not intended to be permanent..
--      version             [0]  Version, --  DEFAULT v1,
--      serialNumber        CertificateSerialNumber,
--      signature           AlgorithmIdentifier,
        issuer              PrincipalName,
--      validity            Validity,
        subjectRealm        Realm,
        subject             PrincipalName,
--      subjectPublicKeyInfo SubjectPublicKeyInfo,
--      issuerUniqueID   [1]  IMPLICIT UniqueIdentifier OPTIONAL,
--      subjectUniqueID  [2]  IMPLICIT UniqueIdentifier OPTIONAL,
        authentSig          Signature
}

--
-- End of Additions for Public Key Authentication --
--
```

10.3. krb5.h

DCE types corresponding to the ASN.1 data that need to be manipulated by DCE clients are defined in additions to the existing file `security/krb5/include/krb5/krb5.h`, shown here.

```
#define OSF_DCE_ARCHAISM
#define OSF_DCE_FEATURE

typedef struct _krb5_pa_enc_ts {
    krb5_timestamp      patimestamp;
    krb5_int32          pausec;
} krb5_pa_enc_ts;

/********************
 * Public Key Login *
 ********************/

typedef struct _krb5_pk_authenticator {
    krb5_checksum  *cksum; /* checksum, includes type,
                              optional */
    krb5_int32     cusec;  /* client usec portion */
    krb5_timestamp ctime;  /* client sec portion */
    krb5_int32     nonce;  /* nonce */
    krb5_principal kdcname;/* KDC name/realm */
} krb5_pk_authenticator;
```

```
typedef struct _krb5_signed_pk_authenticator {
    krb5_pk_authenticator pk_authenticator;
    krb5_enc_data   authentsig; /* signature of
                                   pk_authenticator */
} krb5_signed_pk_authenticator;

typedef struct _krb5_pa_pk_as_root {
    krb5_data       root_cert;   /* not used in DCE Pubkey */
    krb5_signed_pk_authenticator authent;
} krb5_pa_pk_as_root;

typedef struct _krb5_enc_pa_pk_as_rep_part {
    krb5_keyblock  enc_reply_key; /* used to encrypt reply */
    krb5_int32     nonce;         /* copied from request */
} krb5_enc_pa_pk_as_rep_part;

typedef struct _krb5_enc_pa_pk_as_rep_part_shell {
    krb5_enc_pa_pk_as_rep_part reply;
    krb5_enc_data  replysig; /* signature of reply */
} krb5_enc_pa_pk_as_rep_part_shell;

typedef struct _krb5_pa_pk_as_rep {
    krb5_data     kdc_cert;          /* not used in DCE Pubkey */
    krb5_enc_data enc_tmp_key;       /* krb5_keyblock */
    krb5_enc_data enc_shell; /* krb5_enc_pa_pk_as_rep_part_shell */
} krb5_pa_pk_as_rep;

/***************************
 * End of Public Key Login *
 ***************************/

#endif /* OSF_DCE_FEATURE */


/*
 * This structure is returned in the e-data field of the KRB-ERROR
 * message when the error calling for an alternative form of
 * authentication is returned, KRB_AP_METHOD.
 */
typedef struct _krb5_alt_method {
#ifndef OSF_DCE_ARCHAISM
        krb5_magic      magic;
#endif
        krb5_int32      method;
        int             length;
        krb5_octet      *data;
} krb5_alt_method;

/*
 * A null-terminated array of this structure is returned by the KDC
 * as the data part of the ETYPE_INFO preauth type.  It informs the
 * client which encryption types are supported.
 */
typedef struct _krb5_etype_info_entry {
#ifndef OSF_DCE_ARCHAISM
```

```
        krb5_magic      magic;
#endif
        krb5_enctype    etype;
        int             length;
        krb5_octet      *salt;
} krb5_etype_info_entry;

typedef krb5_etype_info_entry ** krb5_etype_info;
```

11. MANAGEMENT INTERFACES

Minimal management interfaces are provided. If a vendor chooses to add other implementations of the DCE Personal Security Module, then the vendor should expect to add management interfaces for configuring the new mechanism for a cell or principal.

11.1. Installation

Installing the new public key functionality requires only stopping DCE, installing the software upgrades (client, security server), and restarting DCE.

11.2. DCE Security Service Configuration

11.3. Enabling OSF DCE 1.2.2 Features

By default, all OSF DCE 1.2.2 features are disabled in a cell originally configured with a release prior to OSF DCE 1.2.2. Once software supporting DCE Public Key Login has been installed on all DCE Security Server replicas, public key functionality, along with other OSF DCE 1.2.2 functionality, can be enabled using the following `dcecp` command:

```
dcecp> registry modify -version secd.dce.1.2.2
```

When OSF DCE 1.2.2 features are enabled, any DCE Security Server replicas that do not support OSF DCE 1.2.2 features are shut down automatically.

A new cell configured with OSF DCE 1.2.2 release software has OSF DCE 1.2.2 features enabled from the start.

11.4. Enabling Public Key Login

After the OSF DCE 1.2.2 features have been enabled, the cell administrator can request generation of the initial KDC public key key-pair using the following `dcecp` command:

```
dcecp> account modify krbtgt/<cell> -pkgenprivkey <modulus size>
```

In response to this command, the DCE Security Registry generates a key-pair for the KDC, storing the public key portion in the `DCEPKAuthentication` ERA attached to the `krbtgt` principal for the cell and storing the private key in the Registry under existing Registry password-protection encryption.

On each client node supporting public key login, the `dced` daemon retrieves the krbtgt principal's public key when requested by `sec_login`, allowing authorized principals to log in using DCE public keys on that client node. See ‘‘KDC Public Key Caching’’ under ‘‘FUNCTIONAL DEFINITION’’ for more information.

11.5. Configuring Public Key Login Users

The cell administrator must use the `dcecp> account create/modify <user name> ....` command with the `-pkmechanism`, `-pksignatureusage`, and `-pkkeycipherusage` options to create a public key account or modify a regular account to become a public key account. See the ‘‘USER INTERFACES’’ section for more information.

Optionally, the cell administrator can attach an instance of the `pre_auth_req` ERA to the public key account principal with a value of `PADATA-ENC-PUBLIC-KEY` (3) to prevent the principal from logging in with any protocol other than public key.

After the public key account has been created, if the user's private key storage mechanism (`-pkmechanism`) is `file`, the cell administrator must move the user's private key file from the administrator's working node, to the user's workstation and change the ownership of the file to the user. The private key file is located in `/opt/dcelocal/var/security/pk_file/<principal_UUID>`.

By default, only the cell administrator can create or modify a user's public key key-pair.

Optionally, the cell administrator can allow a user to change her own public key key-pair by changing the ACL of two ERAs — `DCEPKAuthentication` and `DCEPKKeyEncipherment`. The following `dcecp` command accomplishes this:

```
dcecp>  xattr modify /.:/sec/xattrschema/DCEPKAuthentication
        -aclmgr {principal {query r} {update u}
                        {test r} {delete m}}
```

12. RESTRICTIONS AND LIMITATIONS

12.1. Exportability

12.1.1. Export of Binary (Executable) Code

The functionality provided by the binary code for the `sec_psm_encrypt_data()` and `sec_psm_decrypt_data()` functions is not exportable unless its use is confined to the authentication process in such a way that users are unable to use the interfaces to encrypt and decrypt arbitrary data.

There may also be export issues with the binary code for the `sec_psm_sign_data()` and `sec_psm_verify_data()` functions, depending on algorithm, usage, and key length.

For U.S. vendors and ISVs, OSF DCE provides partial (source-code level) support for building an internationally exportable version of OSF DCE from the Domestic sources. However, this support does not necessarily extend to full (binary-code level) support for building exportable binaries. It is the responsibility of each vendor/ISV to determine how to build such a product for their platform, and to verify that the resulting product is indeed exportable.

To create an exportable product, each vendor should do the following to handle the BSAFE library:

(a) Make the necessary libdce Makefile change, so libbsafe2.a is linked into the shared libdce in an exportable manner. Please note, we do not recommend linking the BSAFE lib into an archive libdce due to export and licensing issues.

(b) Use "nm libbsafe2.a" to extract all the symbols in the BSAFE lib and do all the necessary steps to hide or scramble all the BSAFE symbols to meet the export requirements and potential licensing issues.

Minimally, the following must be done to ensure that Public Key Login limits its use of encryption:

> Define `DES_HIDDEN` for the build of any component that would support encryption and/or decryption in the Domestic build.
>
> No matter how the above is accomplished, it should be understood that this is limited source-code level support, not full binary-code level support for building the Domestic DCE sources for export. Each vendor remains responsible for making sure that the above steps and any additional work to hide and remove encryption have resulted in a product that complies with the applicable export laws.[3]

12.1.2. Export of Source Code

OSF DCE 1.2.2 source includes no-encryption stubs for the operations in the `sec_psm` layer in the international version, similar to those supplied now for DES.

---

3. This is the same advice that has held for all releases of DCE, not just the present DCE 1.2.2 public key work.

12.2. Size

The public key cryptosystem libraries do not add significantly to the size of a DCE client. The increase rates of the related libraries and binaries are:

```
libdce.sl:    .06 %
libdcecp.sl: .07 %

dced:         .05 %
secd:       14%
```

12.3. Performance

The public key cryptosystem operations degrade the performance of DCE client logins, due to the increased complexity of the arithmetic operations involved. Both public key and non-public key logins are affected, since the clients attempt the public key login protocol first.

The performance result of different login protocols are as follows:

The time it takes to perform a non-public key login in 1.2.2 is roughly 1.8 times the time it takes in 1.2.1. In 1.2.2, public key login takes about 1.3 times the time required for non-public key login.

> *NOTE: In the reference implementation, there has been no optimization done on the login protocols, and it is expected that the time required for non-public key login in 1.2.2 could be reduced significantly.*

13. OTHER COMPONENT DEPENDENCIES

13.1. DCE Certification API

Integration with the DCE Certification API specified in [RFC 80.0] is not included in this project.

13.2. DCE Private Key Storage Server

Integration with the DCE Private Key Storage Server specified in [RFC 94.0] is not included in this project, although some components of the integration such as UUID values are shown in this specification. Integration has been done as part of the overall OSF DCE 1.2.2 project by the PKSS supplier.

13.3. Pluggable Authentication Modules (PAM)

[RFC 86.0] outlines integration of ‘‘Pluggable Authentication Modules’’ (PAM) into DCE. It is expected that the entire DCE Login mechanism, in which the functionality specified here is buried, will be wrapped inside a single PAM, and thus there will be no issues specific to the public key aspects of DCE Login. Since the timeframe for [RFC 86.0] is beyond the timeframe for DCE Public Key Login, no further investigation of integration will be done as part of the Public Key Login project.

14. COMPATIBILITY

14.1. Interoperability

Clients attempting to use the new protocols with a security server that does not support the new protocols will receive an error in the `KRB_AS_REP` indicating that the pre-authentication data supplied is not supported (`KRB5KDC_ERR_PADATA_TYPE_NOSUPP`). The client then (unless the `KRB_AS_REP` error information indicates the client is required to use public key login) attempts to use the existing password-based protocol (which will also fail if the principal does not have a valid DCE password).

Clients attempting to use the new protocols with a pre-DCE 1.1 security server (where PADATA is ignored) will receive a standard pre-DCE 1.1 `KRB_AS_REP` with the TGT protected by the user's long-term DCE key.

Users who have invalid passwords, but valid entries for the `DCEPKKeyEncipherment` and `DCEPKAuthentication` ERAs will be unable to log in from client nodes that do not support the new protocols, or that do not support a Personal Security Module with access to the user's private key.

See table in the ‘‘TGT Acquisition Protocol’’ section for more information on compatibility.

14.2. Migration

Migration code has been provided to convert existing DCE 1.2.1 security server data to DCE 1.2.2 formats. Conversion of the DCE Registry database is triggered automatically when `secd` is started up by checking the database version stored in the database against the version in the binary. If different, the automatic database conversion code in `secd` is invoked.

Migration code has been provided that will allow public key protocols to be enabled in the cell only when all replicas support the protocols. See ‘‘Enabling Public Key Login’’ under ‘‘MANAGEMENT INTERFACES’’. Once the new protocols have been enabled, it is not possible to add a non-public key security server to the cell without reconfiguring all security servers as non-public key servers.

When the public key protocols are enabled, migration code is invoked automatically to add the schema entries for the `DCEPKKeyEncipherment`, `DCEPKAuthentication`, and `DCEPKPrivateKeyStorage` ERAs.

15. STANDARDS

[ITU X.208], [ITU X.209], [IETF 1510].

16. OPEN ISSUES

(a) Size of pre-authentication data in the future when certificates are supported. The entire `KRB_AS_REQ` and `KRB_AS_REP` messages need to fit into a UDP packet, so there is a size limit of 1k bytes or so.

The plan is to transmit the certificates out-of-band. The pre-authentication data could contain the signature of the certificate to be used.

REFERENCES

[DRAFT-PKINIT] IETF, ‘‘draft-ietf-cat-kerberos-pk-init-01.txt’’, by Clifford Neuman, Brian Tung, and John Wray, June 1996. Subsequently replaced by ‘‘draft-ietf-cat-kerberos-pk-init-02.txt’’

[IETF 1510] IETF, ‘‘RFC 1510: The Kerberos Network Authentication Service (V5)’’, by J. Kohl and C. Neuman, September 1993.

[ITU AM1] ITU, ‘‘PDAM 1 to ITU-T X.509 (1993) | ISO/IEC 9594-8:1995, Information Technology — Open Systems Interconnection — The Directory: Authentication Framework’’, ISO/IEC JTC 1/SC 21 N 9214, December 1994.

[ITU AM4] ITU, ‘‘PDAM 4 to ITU-T X.511 (1993) | ISO/IEC 9594-3:1995, Information Technology — Open Systems Interconnections — The Directory: Abstract Service Definition’’, ISO/IEC JTC 1/SC 21 N, 24 November 1995.

[ITU X.208] ITU, ‘‘Recommendation X.208: Specification of abstract syntax notation one (ASN.1)’’, ISO/IEC 8824, 1987.

[ITU X.209] ITU, ‘‘Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).’’, ISO/IEC 8825, 1987.

[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.

[ITU X.511] ITU, ‘‘ISO/IEC 9594-3/ITU-T Rec X.511, Information Technology — Open Systems Interconnection — The Directory: Abstract Service Definition’’, 1993 (E).

[PKCS 8] RSA Laboratories, ‘‘PKCS #8: Private-Key Information Syntax Standard’’, Version 1.2, November 1, 1993.

[RFC 6.0] J. Pato, ‘‘A Generic Interface for Extended Registry Attributes’’, June 1992.

[RFC 68.1] A. Anderson, J. Wray, ‘‘DCE 1.2 Public-Key Login — Functional Specification’’, February 1995.

[RFC 80.0] J. Wray, ''DCE Certification API — Functional Specification'', January 1995.

[RFC 85.0] M. Warner, ''Improved Public Key Login Protocols for DCE'', October 1995.

[RFC 86.0] V. Samar, R. Schemers, ''Unified Login with Pluggable Authentication Modules (PAM)'', October 1995.

[RFC 94.0] J. Wray, ''DCE Private Key Storage Server'', to appear.

AUTHORS' ADDRESSES

Anne H. Anderson
Hewlett-Packard Company
300 Apollo Drive
Chelmsford, MA 01824
USA
Internet e-mail: aha@apollo.hp.com
Telephone: +1-508-436-5707

Sue-Fen Wang Cuti
Hewlett-Packard Company
300 Apollo Drive
Chelmsford, MA 01824
USA
Internet e-mail: cuti@apollo.hp.com
Telephone: +1-508-436-4241