Open Software Foundation
Request For Comments: 68.2
February 1996

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 68.1].

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 will 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 re-issue 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 protocol is based on [ITU X.509] as described in [RFC 85.0].

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

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

Basic changes since [RFC 68.1]:

(a) The public key login protocol is now X.509-based.

(b) Private key storage in the DCE Registry is no longer supported. (That functionality will instead be provided by the DCE Private Key Storage Server, described in [RFC 94.0].)

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 an X.509-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 will use the RSA BSAFE library). The international version of the reference source code will have no-op stubs for each generic public key operation.

(f) Allow for integration with public key formats that need not be RSA-format keys (although the reference implementation will use 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 will be obtained by the KDC from the DCE Registry. If a client provides a certificate as part of the authentication protocol, the KDC will ignore 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 will 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) No public key key-pair generation module is supported. It is assumed that the public key infrastructure provided by the vendor or site will provide this function. The RSA BSAFE library includes such a module, but this will not be tested or supported as part of the reference implementation.

A set of sample key-pairs will be supplied with the reference implementation for testing purposes.

(e) 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’’.

(f) There is no support added to the `dcecp> account create` command to create an account identified as a ‘‘public key account’’. See ‘‘Configuring Public Key Login Users’’ under ‘‘MANAGEMENT INTERFACES’’.

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].

(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.

---

1. In this vein, it is worth noting that the certificates described in [ITU X.509] are ‘‘X.509 version 2 certificates’’. Work on ‘‘X.509 version 3 certificates’’ remains incomplete at the time of this writing, though it is anticipated that those will be supported in DCE eventually.

(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 the same as that described in [RFC 85.0]. The DCE login APIs (`sec_login_validate_identity()`, `sec_login_valid_and_cert_ident()`, and `sec_login_validate_first()`) will attempt to use this protocol initially.

If the KDC is unable to authenticate the user with the supplied public key pre-authentication data, the KDC returns error information. In addition to an error indicating why the authentication failed, this error information will include an indication of whether the user is required to use the public key authentication protocol (which KDC determines from the `pre_auth_req` ERA attached to the user principal).

If the initial public key login attempt fails, then the `sec_login` code will fall 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 `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.

> *NOTE: There may be a significant performance degradation in login operations due to the public key login protocol. See ''Performance'' under ''RESTRICTIONS AND LIMITATIONS''.*

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

---

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

protocols rely on time stamps. Nevertheless, 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 public key authentication information corresponds to X.511 [ITU AM4] **bind tokens**. It is transmitted in the pre-authentication data fields of the standard Kerberos V5 `KRB_AS_REQ` and `KRB_AS_REP` messages [IETF 1510] as a new `OSF_DCE_PUBLIC_KEY` subtype of the existing `KRB5_PADATA_OSF_DCE` pre-authentication data type.

6.1.1. Client-to-KDC Message

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

The client process signs a message including a time stamp, a nonce and the identity of the KDC. This is sent to the KDC along with the client's claimed identity and (optional) certificate.

> *NOTE: In the reference implementation, client processes will not send a certificate.*

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 will ignore any certificates sent by client processes and will obtain public keys directly from the DCE Registry.*

6.1.2. KDC-to-Client Message

```
KDC-->C:        KDC,{Tkdc,C,Nc,{Ktgtkeykey}Pc}Skdc,Ckdc
```

The KDC checks the time stamp and signature of the client's message. If the time stamp is sufficiently current and the nonce new (same criteria as used with current `OSF_DCE_THIRD_PARTY` pre-authentication), the KDC responds with a signed message that includes a time stamp, the client's nonce, the client's identity, and a random symmetric key encryption key (`Ktgtkeykey`). This key encryption key is encrypted using the client's public key. It is used to encrypt the symmetric session key associated with the TGT in the `KRB_AS_REP` message. The KDC optionally includes its certificate in the response.

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 session key is encrypted using the key encryption key returned in the signed authentication data from the KDC.

> *NOTE: In the reference implementation, the KDC will not send its certificate in the response. The public key of the KDC is cached locally on the client node by* `dced`*, 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 will check for an instance of the `pre_auth_req` ERA attached to the user principal. If found, the authentication attempt will be 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 existing protocol.

6.2. DCE Personal Security Module API

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 here 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 will be a consumer of this API.

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.

> *NOTE: In the reference implementation, a trivial implementation of this layer is provided by a UNIX file —* `/opt/dcelocal/var/security/pk_file/`*`principal`* *— with permissions* `600` *and owner* *`principal`*.
>
> *Such an implementation has obvious problems in a distributed environment, and is intended only for testing and prototype usage.*
>
> *Vendors may choose to substitute (or add) an implementation that provides a more configurable interface, uses the DCE Private Key Storage Server ([RFC 94.0]), uses smart cards, or supports an alternative cryptosystem.*

6.3. Password Change

The existing DCE client password change APIs (such as `sec_rgy_acct_add()` or `sec_rgy_acct_replace_all()`) will be modified to update the password used for access to the Personal Security Module where appropriate.

6.4. KDC Public Key Caching

The `dced` daemon will be modified to support caching of KDC public authentication keys and to support requests for updates or additions to the cache from login processes.

The daemon, after authenticating to the KDC, initially will retrieve the authentication public key of the local cell's KDC from the `DCEPKAuthentication` ERA attached to the `krbtgt` principal for the cell, and write it to the file `/etc/opt/dce/security/kdc_pk_auth/`*`cell_name`* in ASN.1 DER encoding.

The `sec_login` APIs will 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.

The existing mechanism for obtaining third-party pre-authentication by `sec_login` from the `dced` will be extended to support requests from `sec_login` to the `dced` daemon to retrieve or update, and cache the public key for the KDC of a local or foreign cell.

6.5. KDC Public Keys

The initial KDC authentication public key key-pair will be generated and stored automatically in the master DCE Security Server at the time public key login functionality is enabled (see ‘‘Enabling Public Key Login’’ under ‘‘MANAGEMENT INTERFACES’’).

The KDC private key used for authentication (generating signatures) will be stored in the DCE Registry, protected, and propagated exactly as the DES symmetric KDC key is now stored, protected, and propagated, by the implementation of a new propagation operation.

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

6.6. Public Key Policy Control

The modulus length of the KDC authentication public key is determined by the value of the `DCEPKModulusLength` ERA attached to the `krbtgt` principal object in the DCE Registry. If there is no such ERA attached, then a default length of 1024 bits is used.

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.

7. DATA STRUCTURES

7.1. Public Key Base Types

The common base definitions for the DCE Public Key APIs are defined in a new file `security/idl/sec_pk_base.idl`, listed here:

```
/*
** Copyright (c) Hewlett-Packard Company 1996
** Unpublished work.  All Rights Reserved.
**
**       Public Key Base Definitions
**
*/

[
    local
]

interface sec_pk_base {

    import "dce/nbase.idl";    /* for error_status_t, uuid_t */

    /* 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.
     *
     * In the reference implementation, only the
     * "sec_pk_domain_dce_pk_login" domain is supported.
     */
    typedef sec_pk_domain_t uuid_t;

    /* The following sec_pk_domain_t UUID's are
     * architectural.  They cannot be changed without
     * destroying interoperability.
     */

    /* "ae991638-5cbe-11cf-a22d-08000919ebb5" */
    extern uuid_t sec_pk_domain_dce_pk_login;


    /* 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;

/* 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       num_enc_bytes;
    [ref, size_is(num_bytes)]
        byte         *encoded_data_p;
} sec_pk_data_t;


/* s e c _ p k _ d a t a _ t _ p
 *
 * A pointer to a sec_pk_data_t
 * structure
 */
typedef sec_pk_data_t   *sec_pk_data_t_p;


/* 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 _ t _ p
 *
 * A pointer to a sec_pk_pubkey_t structure
 */
```

```
typedef sec_pk_pubkey_t *sec_pk_pubkey_t_p;


/* 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 _ t _ p
 *
 * A pointer to a sec_pk_pvtkey_t structure
 */
typedef sec_pk_pvtkey_t *sec_pk_pvtkey_t_p;


/* 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 _ t _ p
 *
 * A pointer to a sec_pk_signed_t structure
 */
typedef sec_pk_signed_t *sec_pk_signed_t_p;


/* 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 _ t _ p
 *
 * A pointer to a sec_pk_encrypted_t structure
 */
typedef sec_pk_encrypted_t *sec_pk_encrypted_t_p;


/* s e c _ p k _ o b j _ i d _ t
 *
 * A sec_pk_data_t structure containing an X.509
 * ASN.1 DER-encoded value of type
 * OBJECT IDENTIFIER.
 *
 * 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_obj_id_t;

/* s e c _ p k _ o b j _ i d _ t _ p
 *
 * A pointer to a sec_pk_obj_id_t structure
 */
typedef sec_pk_obj_id_t *sec_pk_obj_id_t_p;


/* s e c _ p k _ B i n d K e y I n f o _ t
 *
 * A sec_pk_data_t structure containing an X.509
 * ASN.1 DER-encoded value of type
 * BindKeyInfo
 *
 */
typedef sec_pk_data_t sec_pk_BindKeyInfo_t;

/* s e c _ p k _ B i n d K e y I n f o _ t _ p
 *
 * A pointer to a sec_pk_BindKeyInfo_t structure
 */
typedef sec_pk_BindKeyInfo_t *sec_pk_BindKeyInfo_t_p;


/* 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 _ t _ p
 *
 * A pointer to a sec_pk_algorithm_id_t structure
 */
typedef sec_pk_algorithm_id_t *sec_pk_algorithm_id_t_p;


/* 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_p
 *                 that points to the memory
 *                 to be reclaimed.
 *                 sec_pk_data_t_p is returned
 *                 set to NULL.
 *
 *   Errors: [reflects errors from underlying
 *            deallocation functions]
 */
error_status_t sec_pk_data_free (
    [in,out] sec_pk_data_t_p   *data_p
);


/* 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_p
 *                 that points to the memory
 *                 to be reclaimed.
 *                 sec_pk_data_p is returned
 *                 set to NULL.
 *
```

```
 *   Errors: [reflects errors from underlying
 *            deallocation functions]
 */
error_status_t sec_pk_data_free (
    [in,out] sec_pk_data_t_p   *data_p
);
```

7.2. 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}"
```

> *NOTE: The value of* `PADATA_ENC_PUBLIC_KEY` *may change to be greater than 3 if during implementation we discover a need to allow for intermediate protocols.*

7.3. 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 X.509 ASN.1 DER-encoded byte-string of type `SubjectPublicKeyInfo` (`*sec_pk_pubkey_t_p->encoded_data_p`).

```
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."
```

7.4. 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. In particular, the `DCEPKAuthentication` ERA attached to the cell's `krbtgt` principal is the public key used to verify a signature from the DCE KDC during DCE authentication.

The format of the `DCEPKAuthentication` ERA is an X.509 ASN.1 DER-encoded byte-string of type `SubjectPublicKeyInfo` (`*sec_pk_pubkey_t_p->encoded_data_p`).

```
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 in ASN.1 DER format."
```

7.5. DCEPKModulusLength ERA

The `DCEPKModulusLength` ERA is a reserved ERA that may be attached to a principal object to specify the length of the public key modulus to be used when generating a DCE authentication public key for the principal.

> *NOTE: In the reference implementation, only the* `DCEPKModulusLength` *ERA attached to the* `krbtgt` *principal has any effect, as key generation is not supported except for the KDC.*

```
Attribute Name: DCEPKModulusLength
Attribute UUID: dc8be940-6230-11cf-9c5c-08000919ebb5
Attribute Encoding: sec_attr_enc_integer
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 length in bits of the modulus to
          be used when generating a DCE
          authentication public key
          key-pair for the principal.  Default is
          1024 bits."
```

8. USER INTERFACES

8.1. Login

User interfaces to login utilities do not change, 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 will still need to prompt for a user name and a password.

The password that the user supplies will first be tried by `sec_login` as a password to access the user's Personal Security Module. If there is no Personal Security Module, or if the password fails to unlock the module, the password is then tried 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 Extended Registry Attribute values may be used to read or write the `DCEPKKeyEncipherment`, `DCEPKAuthentication`, `pre_auth_req`, or `DCEPKModulusLength` attributes.

The public key attributes must be X.509 ASN.1 DER-encoded byte-strings of type `SubjectPublicKeyInfo` (`*sec_pk_pubkey_t_p->encoded_data_p`).

New `-pkkeyencipherment` and `-pkauthentication` options are defined for the `dcecp> principal modify` command. These allow setting a principal's private key value in the Personal Security Module, and optionally setting the corresponding public key value in the DCE Registry. If no values are supplied, then new key pairs are generated and stored.

> *NOTE: In the reference implementation, values must be specified. There is no key-pair generation module supported.*

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

Examples:

```
dcecp> principal show <principal> -attr
       {DCEPKKeyEncipherment <public key>
       }
       {DCEPKAuthentication  <public key>
       }
       {pre_auth_req 3
       }
dcecp> principal show krbtgt/<cell> -attr
       {DCEPKKeyEncipherment <public key>
       }
       {DCEPKAuthentication  <public key>
       }
       {DCEPKModulusLength 1024
       }
dcecp> principal modify <principal>
       -pkkeyencipherment  [<private key file path>
                            [<public key file path>]]
       -pkauthentication [<private key file path>
                          [<public key file path>]]
       -mypwd <user's password>
dcecp> principal modify <principal>
       -attr
       {DCEPKKeyEncipherment <public key>
       }
       {DCEPKAuthentication  <public key>
```

```
        }
        {pre_auth_req 3
        }
dcecp> principal modify krbtgt/<cell>
        -attr
        {DCEPKModulusLength 2048
        }
```

where `<public key file path>` and `<private key file path>` are paths to files containing ASN.1 DER-encoded byte-string values of type X.509 `SubjectPublicKeyInfo` or a `PrivateKeyInfo` encoding supported by the public key infrastructure, respectively.

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

It is assumed that the public key infrastructure available to the user can generate keys in ASN.1 DER encoded form.

8.3. Password Change

In addition to their previous functionality, the existing `dcecp` password change operations will invoke the `sec_psm_update_pwd()` interface indirectly through the `sec_rgy_acct_add()` and `sec_rgy_acct_replace_all()` interfaces.

9. APIs

9.1. DCE Personal Security Module API

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

```
/*
** Copyright (c) Hewlett-Packard Company 1996
** Unpublished work.  All Rights Reserved.
**
**       Personal Security Module (PSM) Interface
**
*/

[
    local
]

import "dce/sec_pk_base.idl";

typedef void *psm_handle_t;

/* PSM API */

/* sec_psm_open
 *
 * Open the personal security mechanism using password.
 *
 * Return status
 *   error_status_ok:    Success.
 *   other (non-zero):   No personal security mechanism available.
 *                       Illegal password.
 *
 * 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] 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):   Illegal psm handle.
 *
 * 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]  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):   Illegal psm handle.
 *                       Illegal signature algorithm.
 *                       Out of memory.
 *
 * Input
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.  This handle should be
 *               obtained through sec_psm_open().
 *   signature_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
 *
 *   signature: Pointer to a signature buffer pointer.
 *
 */


error_status_t sec_psm_sign_data(
    [in]  psm_handle_t            psm_handle,
    [in]  sec_pk_algorithm_id_t_p signature_id,
    [in]  sec_key_usage_flags_t   key_usage,
    [in]  sec_pk_data_t_p           data,
    [out] sec_pk_signed_t_p         *signature
);


/* sec_psm_verify_data
 *
 * Verify the data
 *
 * Return status
 *   error_status_ok:    Success.
 *   other (non-zero):   Illegal psm handle.
 *                       Illegal signature.
 *                       No public key available for the user.
 *
 * Input
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.  This handle should be
 *               obtained through sec_psm_open().
 *   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]  psm_handle_t            psm_handle,
    [in]  sec_key_usage_flags_t   key_usage,
    [in]  sec_pk_data_t_p         data,
    [in]  sec_pk_signed_t_p       signature,
);


/* sec_psm_encrypt_data
 *
 * Encrypt the data in the mechanism 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.
 *
 * Return status
 *   error_status_ok:    Success.
 *   other (non-zero):   Error.
 *
 * Input
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.  This handle should be
 *               obtained through sec_psm_open().
 *   encryption_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.
 *
 * Output
 *
 *   cipher_data: Pointer to the encrypted output buffer.
 */

error_status_t sec_psm_encrypt_data(
    [in]  psm_handle                    *psm_handle,
    [in]  sec_pk_algorithm_id_t_p       encryption_id,
    [in]  sec_key_usage_flags_t         key_usage,
    [in]  sec_pk_data_t_p               clear_data,
    [out] sec_pk_encrypted_t_p          *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.
 *
 * Return status
 *   error_status_ok:    Success.
 *   other (non-zero):   Error.
 *
 * Input
 *
 *   psm_handle: Pointer to an opaque handle to the personal
 *               security context data.  This handle should be
 *               obtained through sec_psm_open().
 *   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]  psm_handle                    *psm_handle,
    [in]  sec_key_usage_flags_t         key_usage,
    [in]  sec_pk_encrypted_t_p          cipher_data,
    [out] sec_pk_data_t_p               *clear_data
);


/* admin */

/* sec_psm_put_pub_key
 *
 * Store the public key pair with associated data into the personal
 * security mechanism.
 *
 *
 * NOTE: Deleting key function is not supplied.  Deleting keys is
 * an internal functions which PSM should initiate  to manage its
 * keys.  So is the versioning.
 *
 *
 * Return status
 *   error_status_ok:    Success.
 *   other (non-zero):   Out of 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 this key pair
 *              belongs to.
```

```
 *   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]  void                   *name,
    [in]  char                   *pwd,
    [in]  sec_pk_domain_t        *domain_id,
    [in]  sec_pk_usage_flags_t   key_usage,
    [in]  sec_pk_pvtkey_t_p      pvtkey,
    [in]  sec_pk_pubkey_t_p      pubkey
);

/* sec_psm_update_pub_key
 *
 * Update the user's own public key pair. The password 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):   Out of 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 this key pair
 *              belongs to.
 *   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]  void                   *name,
    [in]  char                   *pwd,
    [in]  sec_pk_domain_t        *domain_id,
    [in]  sec_pk_usage_flags_t   key_usage,
    [in]  sec_pk_pvtkey_t_p      pvtkey,
    [in]  sec_pk_pubkey_t_p      pubkey
);
```

```
/* sec_psm_update_passwd
 *
 * Update the user's password which is used to protect access to the
 * user's private key stored in the private key storage mechanism.
 * The old password is specified to authenticate the user updating
 * the key.
 *
 * Return status
 *   error_status_ok:    Success.
 *   other (non-zero):   Out of memory.
 *
 * Input
 *
 *   name: Pointer to the user's canonical name within the
 *         specified domain.
 *   oldpwd:  Pointer to the user's old password.
 *   newpwd:  Pointer to the user's new password.
 *   domain_id: Pointer to the application domain this key pair
 *              belongs to.
 *   key_usage: The usage of the public key this key pair
 *              belongs to.
 *
 */


error_status_t sec_psm_update_passwd(
    [in]  void                   *name,
    [in]  char                   *oldpwd,
    [in]  char                   *newpwd,
    [in]  sec_pk_domain_t        *domain_id,
    [in]  sec_pk_usage_flags_t   key_usage,
);
```

9.2. DCE Security Login API

The existing DCE Security Login API in file `security/idl/sec_login.idl` will change to return additional error status values for functions that support public key operations or protocols. Those changes are listed here.

```
    /* s e c _ l o g i n _ v a l i d a t e _ i d e n t i t y
     *
     * Validate the login context established via
     * sec_login_setup_identity().  The caller must know the user's
     * password for this operation to succeed.  This operation MUST
     * be invoked before the network credentials will be used.  This
     * operation is intended for use by non-privileged programs
     * performing network logins.  System login programs that set
     * local operating system identity using data extracted from a
     * login context should use sec_login_valid_and_cert_ident()
     * instead of sec_login_validate_identity().
     *
     * This operation destroys the contents of the passwd input
     * parameter.
     *
     * DCE 1.0 Notes
     *
```

```
      * This operation does not consult local passwd data if the
      * network is unavailable.
      *
      * ERRORS:
      *    sec_rgy_registry_unavailable
      *    sec_rgy_object_not_found
      *    sec_rgy_passwd_invalid
      *    sec_login_s_acct_invalid        - account not valid for
      *                                      login
      *    sec_login_s_null_password       - can't log with a
      *                                      zero-length password
      *    sec_login_s_unsupp_passwd_type  - invalid  password type
      *    sec_login_s_already_valid       - context already valid
      *    sec_login_s_refresh_ident_bad   - identity being refreshed
      *                                      is no longer valid
      *    sec_login_s_default_use         - can't validate default
      *                                      context
+     *    sec_pk_e_domain_unsupported     - DCE Login domain not
+     *                                      supported by the
+     *                                      personal security module
+     *    sec_pk_e_device_error           - Personal Security Module
+     *                                      device error
+     *    sec_pk_e_usage_unsupported      - A private key of the
+     *                                      required type was not
+     *                                      located in the PSM
+     *    sec_pk_e_unauthorized           - invalid password for
+     *                                      PSM access
      */
      boolean32 sec_login_validate_identity (
         [in]            sec_login_handle_t   login_context,
         [in, out, ref]  sec_passwd_rec_t     *passwd,
         [out, ref]      boolean32            *reset_passwd,
         [out, ref]      sec_login_auth_src_t *auth_src,
         [out, ref]      error_status_t       *st
     );

     /* s e c _ l o g i n _ v a l i d _ a n d _ c e r t _ i d e n t
      *
      * Validate and certify a login context established via
      * sec_login_setup_identity.  The caller must know the user's
      * password for this operation to succeed.
+     * This operation is intended for use by system login programs
      * that need to extract trustworthy OS credentials for use in
      * setting a process' local identity.  This operation destroys
      * the contents of the passwd input parameter.  If the network
      * security service is unavailable, or the user's password has
      * been overridden on the host, a locally authenticated context
      * is created, and the auth_src parameter is set to
      * sec_login_auth_src_local.  Data extracted from locally
      * authenticated context may be used to set local OS identity,
      * but cannot be used to establish network credentials.
      *
      * sec_login_valid_and_cert_ident() is a privileged operation
      *
      * DCE 1.0 Notes:
```

```
     *
     * Overrides are not implemented
     *
     * ERRORS:
     *    sec_rgy_registry_unavailable
     *    sec_rgy_object_not_found
     *    sec_rgy_passwd_invalid
     *    sec_login_s_acct_invalid        - account not valid for
     *                                      login
     *    sec_login_s_null_password       - can't log with a
     *                                      zero-length password
     *    sec_login_s_unsupp_passwd_type  - invalid  password type
     *    sec_login_s_already_valid       - context already valid
     *    sec_login_s_refresh_ident_bad   - identity being refreshed
     *                                      is not valid
     *    sec_login_s_privileged          - calling process is not
     *                                      privileged
     *    sec_login_s_default_use         - can't validate default
     *                                      context
+    *    sec_pk_e_domain_unsupported     - DCE Login domain not
+    *                                      supported by the PSM
+    *    sec_pk_e_device_error           - PSM device error
+    *    sec_pk_e_usage_unsupported      - A private key of the
+    *                                      required type was not
+    *                                      located in the PSM
+    *    sec_pk_e_unauthorized           - invalid password for
+    *                                      PSM access
     */
     boolean32 sec_login_valid_and_cert_ident (
        [in]            sec_login_handle_t   login_context,
        [in, out, ref]  sec_passwd_rec_t     *passwd,
        [out, ref]      boolean32            *reset_passwd,
        [out, ref]      sec_login_auth_src_t *auth_src,
        [out, ref]      error_status_t       *st
    );

    /* s e c _ l o g i n _ v a l i d a t e _ f i r s t

     * Validate the default login context established via
     * sec_login_setup_first().  Typically, this operation will be
     * called from the sec_clientd helper process to validate the
     * default credentials for the host machine process hierarchy.
     * This operation uses the password for the local host, and
     * therefore does not require a password parameter.
     *
     * ERRORS:
     *    sec_login_s_privileged          - called in unprivileged
     *                                      process
     *    sec_rgy_registry_unavailable
     *    sec_login_s_auth_failure
+    *    sec_pk_e_domain_unsupported     - DCE Login domain not
+    *                                      supported by the PSM
+    *    sec_pk_e_device_error           - PSM device error
+    *    sec_pk_e_usage_unsupported      - A private key of the
+    *                                      required type was not
```

```
+    *                                       located in the PSM
+    *    sec_pk_e_unauthorized            - invalid password for
+    *                                       PSM access
     */
     boolean32 sec_login_validate_first (
        [in]        sec_login_handle_t   init_context,
        [out, ref]  boolean32            *reset_passwd,
        [out, ref]  sec_login_auth_src_t *auth_src,
        [out, ref]  error_status_t       *st
    );
```

9.3. sec.sams

Error status values generated by DCE Public Key Login (with the exception of the Kerberos KDC errors) 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`.

The following new sub-component will be added to the existing `sec.sams` file.

```
sub-component sec_s_pubkey  "public_key"  sec_i_svc_pubkey
```

The following errors will be added to the existing `sec.sams` file.

```
# Files : sec_login_pvt.c
start
code             sec_pk_e_domain_unsupported
text             "DCE Login domain not supported in Personal Security
                 Module"
explanation      "The Personal Security Module does not support the
                 DCE Login domain."
action           "Login from another node with support for the
                 DCE Login 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 login keys not supported"
explanation      "The Personal Security Module does not have access
                 keys of the correct type for public key."
action           "Login from another node with support for the
                 this user's public key login keys."
sub-component    sec_s_pubkey
attributes       "svc_c_sev_fatal | svc_c_action_abort"
engineer         "Anne Anderson"
end

start
code             sec_pk_e_device_error
text             "Personal Security Module device error."
explanation      "The Personal Security Module reported a
                 device error."
action           "Diagnose and fix the device."
sub-component    sec_s_pubkey
attributes       "svc_c_sev_fatal | svc_c_action_abort"
```

```
engineer        "Anne Anderson"
end

start
code            sec_pk_e_unauthorized
text            "Invalid login to Personal Security Module."
explanation     "An incorrect user name, password, or private key
                storage device was supplied while attempting to
                log in."
action          "Supply correct user name, password, and, if
                necessary, the correct private key storage device
                (floppy disk, smart card)."
sub-component   sec_s_pubkey
attributes      "svc_c_sev_fatal | svc_c_action_abort"
engineer        "Anne Anderson"
end
```

10. REMOTE INTERFACES

The DCE Public Key Login X.509 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 a pre-authentication structure of existing type `KRB5_PADATA_OSF_DCE` and new sub-type `OSF_DCE_PUBLIC_KEY`.

10.1. krbdcepk.mvr

The X.509-based credentials used in the DCE public key TGT acquisition protocol are defined in a new file `security/krb5/lib/mvr/krbdcepk.mvr`, listed here.

> *NOTE: In the reference implementation, the* `mavros` *compiler will be 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 will be constructed.*

```
-- Copyright (c) Hewlett-Packard Company 1991, 1994
-- Unpublished work. All Rights Reserved.
--
-- ASN.1 definitions for the DCE Kerberos Public Key
-- Authentication network objects
--

KRBDCEPK DEFINITIONS ::=
BEGIN

   IMPORTS
       UTCTime
           FROM KRB5 ;

DistinguishedName ::= OCTET STRING;
  -- In DCE PK protocol, contains a canonical DCE --
  -- principal name --

-- The following definitions come from X.509 (plus AM1) --
-- and X.511 (1993) (plus AM4).  They define the protocol --
-- used by DCE Public Key Login (OSF_DCE_PUBLIC_KEY --
-- subtype of padata type KRB5_PADATA_OSF_DCE). --

KeyUsage ::= CHOICE {
```

```
    userKeyUsage       [0]  UserKeyUsage,
    cAKeyUsage         [1]  CAKeyUsage
        -- not used --
}

UserKeyUsage ::= BIT STRING {
    digitalSignature        (0),
        -- used in signatures --
    nonRepudiation          (1),
    keyEncipherment         (2),
        -- used in enciphering TGT key --
    dataEncipherment        (3),
    keyAgreement            (4),
    keyCertSign             (5),
    offLineCRLSign          (6)
}

SupportedAlgorithms ::= ATTRIBUTE {
    WITH SYNTAX SupportedAlgorithm
    ID { id-at 52 }
}

SupportedAlgorithm ::= SEQUENCE {
    algorithm         AlgorithmIdentifier,
    usage             [0] KeyUsage OPTIONAL,
    usageRestriction  [1] PolicyInformation OPTIONAL
        -- not used in DCE 1.2.2 --
}

AlgorithmIdentifier ::= SEQUENCE {
    algorithm          ALGORITHM.&id (
        { SupportedAlgorithms }
    ),
    parameters         ALGORITHM.&Type (
        { SupportedAlgorithms}{@algorithm}
    ) OPTIONAL
}

HASHED { ToBeHashed } ::= OCTET STRING (
    CONSTRAINED-BY {
        -- must be the result of applying a hashing --
        -- procedure to the DER-encoded octets --
        -- of a value of --
        ToBeHashed
    }
)

ENCRYPTED { ToBeEnciphered } ::= BIT STRING (
    CONSTRAINED-BY {
        -- must be the result of applying an --
        -- encipherment procedure to the DER-encoded --
        -- octets of a value of --
        ToBeEnciphered
    }
)
```

```
SIGNED { ToBeSigned } ::= SEQUENCE {
    toBeSigned        ToBeSigned,
    COMPONENTS OF     SIGNATURE{ ToBeSigned }
}

SIGNATURE { OfSignature } ::= SEQUENCE {
    algorithmIdentifier   AlgorithmIdentifier,
    encrypted             ENCRYPTED {
        HASHED { OfSignature }
    }
}

BindKeyInfo ::= ENCRYPTED { BIT STRING }

Token ::= SIGNED {
    SEQUENCE {
        algorithm         [0] AlgorithmIdentifier,
            -- authentication signature algorithm --
        name              [1] DistinguishedName,
            -- KDC in KRB_AS_REQ --
            -- client principal in KRB_AS_REP --
        time              [2] UTCTime,
            -- timestamp --
        random            [3] BIT STRING,
            -- nonce --
        response          [4] BIT STRING OPTIONAL,
            -- not used --
        bindIntAlgorithm  [5] SEQUENCE OF AlgorithmIdentifier
                              OPTIONAL,
            -- not used --
        bindIntKeyInfo    [6] BindKeyInfo OPTIONAL,
            -- not used --
        bindConfAlgorithm [7] SEQUENCE OF AlgorithmIdentifier
                              OPTIONAL,
            -- not used in KRB_AS_REP --
            -- TGT key encryption key algorithm id in KRB_AS_REQ --
        bindConfKeyInfo   [8] BindKeyInfo OPTIONAL
            -- not used in KRB_AS_REQ --
            -- TGT key encryption key in KRB_AS_REP --
            -- encrypted using client public key --
        dirqop            [9] OBJECT IDENTIFIER OPTIONAL,
            -- not used --
        attributeCertificate [10] AttributeCertificate OPTIONAL,
            -- not used --
        clearanceCertPath [11] ClearanceCertPath OPTIONAL
            -- not used --
    }
}

StrongCredentials ::= SET {
    certification-path [0]  Certification-Path OPTIONAL,
    bind-token         [1]  Token,
    name               [2]  DistinguishedName OPTIONAL
}
```

```
PAPublicKey ::= {
    credentials         [0] StrongCredentials
}

END
```

10.2. 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.

```
****************************
  #ifdef OSF_DCE
  /*
   * Different subtypes of DCE padata
   *
   * OSF_DCE_THIRD_PARTY - protocol defined
   * in DCE RFC 26.
+  *
+  * OSF_DCE_PUBLIC_KEY - protocol defined
+  * in DCE RFC 68.2.
   */
  #define OSF_DCE_THIRD_PARTY             1
+ #define OSF_DCE_PUBLIC_KEY              2
****************************
  #define OSF_DCE_ERR_ADMIN_VALID         4

+ /*
+  * OSF_DCE_ERR_PK - used to indicate there is additional
+  * information regarding why the public key authentication
+  * failed.  The additional information is in the form of a
+  * DCE error status returned in the "err" field of the
+  * "krb5_dce_pk_err_info" structure.
+  */
+ #define OSF_DCE_ERR_PK                  5
****************************
  /*
   * This structure contains various information
   * regarding why the server could not decrypt the
   * preauthentication data.
   */
  typedef struct _krb5_dce_pa_err_info {
      krb5_error_code   err;            /* internal preauth error */
      krb5_preauthtype  type;           /* min. type of preauth
                                           allowed */
      krb5_preauthtype  subtype;        /* min. subtype of preauth
                                           allowed */
      krb5_data         salt;           /* required salt */
  } krb5_dce_pa_err_info;
+
+/*
+ * This structure contains information regarding
+ * why the server could not accept the DCE public key
+ * preauthentication data.
+ *
+ * These errors are:
```

```
+ *   KRB5KDC_ERR_ETYPE_NOSUPP  - an unsupported encryption
+ *      algorithm used in the public key pre-authentication
+ *      data.
+ *   KRB5KRB_AP_ERR_MSG_TYPE - the pre-authentication data
+ *      was not a valid public key authentication bind token
+ *   KRB5_PROG_KEYTYPE_NOSUPP - an invalid key usage was
+ *      supplied in a public key authentication bind token
+ *   KRB5_BADMSGTYPE - public key pre-authentication data
+ *      contains an invalid message type specified for encoding
+ *   KRB5_PARSE_ILLCHAR - illegal character in a public key
+ *      principal name field
+ *   KRB5_PARSE_MALFORMED - illegal format for a public key
+ *      principal name field
+ *   KRB5KRB_AP_WRONG_PRINC - Wrong principal in pk request
+ *   KRB5_KDCREP_MODIFIED - KDC reply did not match expectations
+ *   KRB5_CRYPTO_INTERNAL - Cryptosystem internal error
+ *   KRB5_BAD_KEYTYPE - Keytype is incompatible with encryption type
+ *   KRB5_BAD_KEYSIZE - Key size is incompatible with encryption
+ *                      type
+ *
+ * If the client is required to use the public key
+ * login protocol, then the pk_required field is
+ * set to KRB5KRB_AP_ERR_METHOD as an indication that,
+ * despite failure, other protocols should not be
+ * attempted by the client.
+ */
+typedef struct _krb5_dce_pk_err_info {
+    krb5_error_code   err;          /* KRB5 error code */
+    krb5_error_code   pk_required;  /* KRB5KDC_ERR_NONE or
+                                       KRB5KRB_AP_ERR_METHOD
+                                       if pk login required */
+} krb5_dce_pk_err_info;
****************************
   /*
    * This structure holds the three fields of DCE third party
    * preauthentication data. Each field is encoded.
    */
   typedef struct _krb5_pa_third_party {
       krb5_data *machine_tgt_data;
       krb5_data *single_ebuf_data;
       krb5_data *double_ebuf_data;
   } krb5_pa_third_party;
+
+  /*
+   * krb5_pa_pk_data holds the DCE Public Key preauthentication
+   * data.  The data is DER encoded.
+   */
+  typedef krb5_data *krb5_pa_pk_data;
   #endif /* OSF_DCE */
```

11. MANAGEMENT INTERFACES

Minimal management interfaces are provided, and are not intended to be of product-level functionality. If a vendor chooses to add other implementations of the DCE Personal Security Module, then the vendor should expect to add a management interface for configuring the appropriate mechanisms for a cell or principal.

11.1. Installation

Installing the new public key functionality requires only stopping DCE, installing the software upgrades (libdce, secd, dced), and restarting DCE. No changes to `dce_config` are required.

11.2. DCE Security Service Configuration

If a modulus length for the KDC public key other than the default (see `DCEPKModulusLength` ERA) is desired, then an instance of the `DCEPKModulusLength` ERA containing the desired modulus length should be attached to the `krbtgt` principal for the cell. See ''USER INTERFACES'' section.

11.3. Enabling Public Key Login

By default, public key login functionality is disabled. Once software supporting DCE Public Key Login has been installed on a DCE Security Server, public key functionality can be enabled using the `dcecp> registry modify -version secd.dce.1.2` command. This must be set on each replica first, then on the master DCE Security Server.

When public key login is enabled on the master DCE Security Server, any replicas that do not support public key login will be shut down automatically.

When public key login functionality is enabled on the master DCE Security Server, the server will generate a new public key key-pair according to the `DCEPKModulusLength` ERA, and store the public key portion in the `DCEPKAuthentication` ERA attached to the `krbtgt` principal for the cell.

On each client node supporting public key login, the `dced` daemon will retrieve this 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.

There will be a way for a `cell_admin` to request that the DCE Security Service update its public key key-pair, although the interface has not yet been determined.

11.4. Configuring Public Key Login Users

The cell administrator must attach instances of the `DCEPKKeyEncipherment` and `DCEPKAuthentication` ERAs to the principal, then optionally attach an instance of the `pre_auth_req` ERA to the principal with a value of `PADATA-ENC-PUBLIC-KEY`.

See ''USER INTERFACES'' for the supported interfaces for performing these configuration steps.

12. RESTRICTIONS AND LIMITATIONS

12.1. Exportability

The functionality provided by 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 `sec_psm_sign_data()` and `sec_psm_verify_data()` functions depending on algorithm and key length.

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

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.

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

> Define `NOENCRYPTION` 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 lvel 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]

12.2. Size

The public key cryptosystem libraries may add significantly to the size of a DCE client. This will be measured during implementation and will be reported in the final version of the Functional Specification.

12.3. Performance

The public key cryptosystem operations may 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 could be affected, since the clients will attempt the public key login protocol first. This will be measured during implementation and will be reported in the final version of the Functional Specification.

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.

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

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 will then attempt 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.

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

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 to add the schema entries of the `DCEPKKeyEncipherment`,

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.

DCEPKAuthentication, and DCEPKModulusLength ERAs. The necessary code for this conversion to occur will be added as part of this project.

15. STANDARDS

[ITU X.208], [ITU X.209], [ITU X.500], [ITU X.509], [ITU X.511], [IETF 1510].

16. OPEN ISSUES

(a) How should update of the KDC public key key-pair be triggered?

(b) 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.

(c) Possible future version number as part of the public key ERA values, so that version can be compared with the corresponding private key versions.

REFERENCES

[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 email: aha@apollo.hp.com
Telephone: +1-508-436-5707

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