Open Software Foundation
Request For Comments: 68.1
February 1995

A. Anderson (HP)
J. Wray (DEC)

DCE 1.2 PUBLIC-KEY LOGIN — FUNCTIONAL SPECIFICATION

1. INTRODUCTION

This document specifies the functionality required to integrate public-key mechanisms into DCE login and DCE Registry operations, including the associated protocols, API's, and user interfaces. This specification obsoletes [RFC 68.0].

The goal of this effort is to allow users to choose to use public-key cryptography instead of symmetric-key cryptography to prove their identity to the system.

The immediate benefit of such a change is that, in the event of a KDC compromise public-key users will suffer less exposure than normal users. If the KDC is compromised, all user secret-keys will be revealed to the intruder, which means that 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 KDC compromise cannot reveal these keys.

For public-key users who do not have smart-cards for secure storage of their private-keys, the registry will provide a storage service; although the storage key used for a given user is not known to the registry, a registry compromise will reveal the user's encrypted private-key, enabling a dictionary attack on this key. Therefore, in the event of a registry compromise, public-key users whose private-keys are maintained by the registry should also change their key and password; however (depending on cell policy) they may be permitted to log-in using their current passwords/keys to do so.

1.1. Changes Since Last Publication

The changes that have been made since [RFC 68.0] are massive, and it would be impractical to list them all here. The present revision should be considered the first (nearly) complete release of this document.

A number of ''TBD's'' and editorial ''NOTE's'' remain in this revision. These are meant as *informal* indications of some of the issues still to be addressed. Some of the NOTE's also supply rationale for decisions made.

2. TARGET

This technology is provided for customers who require less reliance on the physical security of DCE Security servers, or who wish to use the asymmetric-key API's or the alternative asymmetric-key authentication mechanism.

3. GOALS AND NON-GOALS

3.1. Goals

(a) Return credentials from login that will support allowing principals to use existing Kerberos-based DCE API's as well as asymmetric-key API's.

(b) Allow reduced reliance on security of the KDC's storage of principal keys.

(c) Allow both interactive user principals and server principals to use asymmetric-key methods.

(d) Provide support for both primary and secondary asymmetric-keys.

(e) Support public-keys that need not be RSA keys.

3.2. Non-Goals

It is not a goal of this functionality to attempt to strengthen the confidentiality of internal DCE wire protocols beyond the existing DES-key protection.

It is not a goal of this functionality to specify password change or key-pair change protocols that do not require possession of a valid TGT (such as would be required to change an expired primary key when expiration policies are being enforced); this functionality is already being examined for current DCE 1.1 primary passwords, and any solution must encompass both primary passwords and primary keys.

3.3. Future Goals

The following functionality described in this specification will not be implemented for OSF DCE 1.2. It is specified here to allow future compatibility and migration:

(a) The `PADATA-ENC-PUBLIC-KEY-KDC` protocol.

(b) Functionality and API's that do not require storage of any keys derived from the principal's primary password in the DCE Registry (i.e., functionality specified as ''Using asymmetric-key methods'').

The following functionality is not committed, but will be implemented for OSF DCE 1.2 if schedule and resources permit:

(a) The `PADATA-ENC-PRIVATE-KEY` login protocol.

(b) Use of `KDC-PubK` to augment the existing primary password change protocol.

4. TERMINOLOGY

It is easy to confuse the term for a public-key cryptosystem or mechanism with the term for the public-key half of a public-key cryptosystem key-pair. To avoid this, this document uses the term ''asymmetric-key'' to denote the public-key cryptographic system or mechanism, and uses ''public-key'' to denote the public-key half of a public-key / private-key pair.

(a) **AS**

Authentication Service. A Kerberos service used in the DCE Security Service for authenticating principals and issuing initial TGT's.

(b) **ERA**

Extended Registry Attribute. A DCE 1.1 facility for attaching attributes to objects in the DCE Registry dynamically [HOPK 93].

(c) **KDC**

Key Distribution Center. A Kerberos facility used in the DCE Security Service to distribute session keys to clients.

(d) **KRB_AS_REQ**

Kerberos Authentication Service Request. A standard Kerberos message used in the DCE Security Service to send a message from a client to the Authentication Service requesting a Ticket Granting Ticket.

(e) **KRB_AS_REP**

Kerberos Authentication Service Reply. A standard Kerberos message used in the DCE Security Service to send a reply to a client from the Authentication Service in response to a `KRB_AS_REQ`.

(f) **TGT**

Ticket-Granting Ticket. A Kerberos feature used in the DCE Security Service. A TGT is a data structure that contains authentication and Kerberos ticket information for a principal, encrypted in the key of the KDC. The encrypted portion of the TGT also contains a session key associated with the TGT.

(g) **pre-authentication**

A protocol used in requesting Kerberos Ticket-Granting Tickets in which the client must supply some proof of identity as part of the request. Two such protocols were implemented in DCE 1.1 [RFC 26.0], [HOND 94].

(h) **symmetric-key**

In this specification, an encryption/decryption system or mechanism using a secret-key known only to the sender and the receiver. In the DCE, the DES symmetric-key mechanism is used.

(i) **secret-key**

In this specification, a key known to two parties and used in a symmetric-key cryptosystem. For privacy and authentication, a sender can encrypt a message in a secret-key shared with the receiver. If the receiver can decrypt the message using the shared secret-key, then the receiver knows that the message was encrypted by the sender and that the message has not been modified.

(j) **asymmetric-key**

In this specification, an encryption/decryption mechanism using a private-key known only to the owner and a public-key that is published. For privacy, messages directed to a principal can be encrypted in the public-key of the principal. Only the principal can decrypt these, using the principal's private-key. For authentication, a principal can encrypt information in the principal's private-key. Other principals can verify that the information originated from the first principal if the information can be decrypted using the first principal's public-key.

(k) **key-pair**

In this specification, an asymmetric-key public-key and its associated private-key.

(l) **primary key-pair**

In this specification, a key-pair associated with a DCE principal that is used for authentication to the KDC. A principal may have only one primary key-pair.

(m) **secondary key-pair**

In this specification, a key-pair associated with a DCE principal other than the primary key-pair used for authentication to the KDC. A principal may have any number of secondary key-pairs.

(n) **private-key**

In this specification, the private-key half of a key-pair. A key known only to the owner.

(o) **public-key**

In this specification, the public-key half of a key-pair. A key associated with a particular principal that is known publicly.

(p) **session key**

A random secret-key generated for use in a particular conversation session. A conversation key.

(q) **password**

A secret sequence of characters entered by a principal to authenticate to another entity that shares the secret (or a value derived from the secret). In DCE, a principal's long-term secret-key used for authentication to the KDC is derived from a one-way hash of the principal's password. Anyone who learns the primary password of another principal can impersonate the principal.

(r) **primary password**

The password used to authenticate to the KDC (i.e., the existing DCE 1.1 principal ''password''). This is also the password used to encrypt a principal's primary private-key, if it is stored in the DCE Registry.

(s) **secondary password**

A password used to encrypt a private-key other than the primary private-key in the DCE Registry.

(t) **DES-key attack**

A cryptographic attack on a message encrypted in a DES-key. Recent research [WIEN 93] suggests that a successful DES-key attack can be made in three hours, using a custom computer currently costing roughly $1 million to build. A $10 million machine can make a successful DES-key attack in 21 minutes.

(u) **password-guessing attack**

A cryptographic attack on a message encrypted in a password, or an attempt at authentication by guessing the password of an authorized user.

Passwords usually fall into one of the following categories: a set of commonly-used words (such as ''root'', user's login name, user's host machine name), a trivial word (such as a short sequence of letters or numbers), or a word that can be found in a dictionary. Even ''random'' passwords are usually limited to the set of printable ASCII characters. This limited selection space makes password-guessing attacks much cheaper and faster than DES-key attacks.

The time required for a successful password-guessing attack depends on the strength of the password, but is a fraction of the time required for a DES-key attack. Good authentication schemes (such as DCE 1.1 Pre-Authentication [HOND 94]) can prevent most password-guessing attacks by limiting the number of invalid login attempts and by further encrypting password-encrypted values with strong DES-keys.

(v) **dictionary attack**

A form of automated password-guessing attack using a word list to supply successive candidates for the password. The word list can be an on-line English (or other language) dictionary, plus other commonly used password choices.

(w) **asymmetric-key attack**

A cryptographic attack on a message encrypted using a public-key, in order to obtain the corresponding private-key. The time required for a successful asymmetric-key attack is exponential in the number of bits in the modulus used to generate the keys. In October 1992, an easy-case 523-bit number was factored (equivalent to computing the private-key) in about three weeks using a commercially-available super-computer. Using *currently-known* factorization methods, a 2048-bit RSA key will ''always be safe from factorization'' [FAQ 94].

(x) **off-line attack**

A cryptographic attack on a copy of an encrypted message. An example is an unauthorized user who taps a network, records an encrypted message, then mounts an cryptographic attack on the message on another computer system using cryptanalytic software.

(y) **host machine principal**

The DCE Registry identity of the machine on which a DCE client is executing. This is the identity under which the machine's security client daemon or the security client functionality of dced executes.

(z) **strong DES-key**

In this specification, a DES-key that is neither a *weak* nor *semi-weak* key (well-known DES-key values that produce easily decipherable encryptions), nor a password-derived DES-key.

(aa) **password-derived DES-key**

In this specification, a DES-key that is generated from a password using a one-way cryptographic hashing scheme. Such keys are subject to dictionary attacks, and thus produce encryptions that are more easily deciphered than those produced from strong DES-keys.

5. REQUIREMENTS

This functionality must not increase the vulnerability of any existing aspect of DCE Security.

This functionality must decrease vulnerability of DCE Security in areas that are currently the easiest points of entry in at least some environments.

See the ‘‘Threat Analysis’’ section under the ‘‘RESTRICTIONS AND LIMITATIONS’’ section for an assessment of how functionality meets this requirement.

The implementation must allow easy generation of sources will null-crypto algorithm for export.

6. FUNCTIONAL DEFINITION

The following notation is used in describing the protocols:

(a) **{x}[y]**

Denotes a block of data `x` that is encrypted using the value `y`. If `y` is an asymmetric public- or private-key, then this notation actually means `{SK}[y],{x}[SK]` whenever `y` is longer than a 64-bit DES-key.

`{TGT}[y]` is somewhat different. It means `TGT,{SK}[y]`, where `SK` is the session key associated with the TGT. In this case, the `SK` is embedded inside the TGT, and is encrypted along with other trusted information in the long-term key of the KDC.

NOTE: [John Wray] Whenever any object is encrypted under a public- or private-key, we want to introduce a random symmetrical key, unless we can guarantee that the object is small enough. In practice, this means that the notation:

`{Data}[PublicKey]`

really means:

`{RandomSK}[PublicKey], {Data}[RandomSK]`

whenever `<Data>` *is larger than a DES-key.*

We should make sure that the functioinal spec is consistent, either by stating the above rule, or by making the use of the random DES-key explicit in the protocols.

In some ways I favor just stating the rule, because then we can treat PK encryption very much like symmetric encryption, both in the spec and in the implementation. For example, consider encrypting a Kerberos TGT using the RSA algorithm. There are two notations for writing this:

(i) `{SK}[PubK], {TGT}[SK]` *(explicitly showing the random session-key)*

(ii) `{TGT}[PubK]`

We can implement the encryption either by using a regular DES-encrypted Kerberos TGT, and sending extra data along with it containing {SK}[PubK], or we can implement a new algorithm (‘‘RSA with random DES-key when necessary’’), teach the Kerberos libraries this algorithm, and then just generate a TGT encrypted under the algorithm. That saves us having to modify the Kerberos protocol — all we're doing is defining a new encryption algorithm.

Note that the notation (ii) above sort-of implies that we're making the first choice, whereas notation (i) implies the use of a combined ‘‘RSA with random DES-key’’ algorithm, which is why I prefer notation (ii). It treats the use of the random DES-key as a detail of the encrpytion algorithm, that isn't important at this level of spec. Of course the notations are strictly equivalent, so it doesn't really matter which you pick.

(b) **SKx**

Denotes a session key (a ‘‘strong’’ DES-key).

(c) **H1**

Denotes one of two keys derived from a password, the other being `H2`. `H1` is derived using a new (to DCE) hashing algorithm. `H1` is itself stored in the DCE Registry, but is used only to encrypt the stored value of a private-key in the DCE Registry. Ability to decrypt a value encrypted in `H1` provides proof of knowledge of the password.

(d) **H2**

Denotes one of two keys derived from a password, the other being `H1`. `H2` is derived using the existing DCE str_to_key() function. `H2` is stored in the DCE Registry and is retrievable by the KDC. Knowledge of `H2` provides evidence, but not proof, of knowledge of the password, as `H2` could be stolen from a compromised DCE Registry.

(e) **M-TGT**

Denotes the current Ticket-Granting Ticket of the client's host machine principal.

(f) **M-SK**

Denotes the session key associated with the current TGT of client's host machine principal.

(g) **PrivK**

Denotes the private-key half of the principal's key-pair.

(h) **PubK**

Denotes the public-key half of the principal's key-pair.

(i) **PK-UUID**

Denotes the UUID of the ERA containing a public-key. Also used to identify any associated secondary password-encrypted private-key.

(j) **PW**

Denotes the principal's primary password.

(k) **KDC-PubK**

Denotes the public-key of the KDC.

(l) **TGT**

Denotes a Kerberos V5 Ticket-Granting Ticket.

(m) **TS**

Denotes a timestamp.

6.1. Storage of Asymmetric Keys in the DCE Registry

Asymmetric keys stored in the DCE Registry are always identified with an ERA. They are identified either by the name of the ERA or by the UUID of the ERA.

6.1.1. Public-keys

Public-keys alone can be stored as ERA instances using schema entries with an encoding type of `bytes`.

They may also be stored as instances of an ERA having the new `sec_attr_enc_public_key` encoding type described below.

6.1.2. Key-pairs

Private-keys, if stored in the DCE Registry, need to be stored in encrypted form, and need to be re-encrypted whenever their encrypting key changes. They also need to be changed (and re-encrypted) whenever their corresponding public-key changes.

The properties of key-pairs stored in the DCE Registry are:

(a) Key-pairs may be attached only to principal nodes.

(b) The private-key half of any key-pair may be `NULL` (i.e., not contained in the DCE Registry).

(c) The private-key half of a key-pair, if non-`NULL`, is always encrypted using an associated password (see `Encryption of private-keys` section below).

(d) All encryption and decryption of private-keys is done on the client side.

(e) Only the encrypted form of the private-key is ever transmitted over the wire.

(f) The protocols for changing a key-pair password, or for changing or retrieving the private-key half of a key-pair, require proof of knowledge of one of the following:

  (i) the owner principal's primary password,

  (ii) the owner principal's primary private-key,

  (iii) the password associated with the private-key,

  (iv) or the private-key half of the key-pair.

  This prevents applications running under the user's identity from retrieving private-keys other than the one(s) associated with a specific password. This also prevents a compromise of the KDC from immediately compromising private-keys.

(g) The Registry maintains the linkage between a public-keys, any associated private-key, and the password used to encrypt the private-key. All API's that change one associated element require update of all elements that depend on it.

(h) When a key-pair is changed, the date and time of the change are recorded for use by any key-expiration policy controls.

(i) Key-pair change may be subject to policy controls on minimum modulus length, minimum lifetime, or maximum lifetime.

(j) Key-pair password change may be subject to policy controls on password strength, password minimum lifetime, or maximum lifetime.

To provide these semantics, new asymmetric-key storage functionality is provided that associates three elements: a public-key, a private-key, and a password used to encrypt the private-key. The encrypted private-key element and the associated password may be `NULL`.

The public-key element is projected into a read-only ERA with a new encoding type of `sec_attr_enc_public_key`, in order to provide access via the existing ERA API's, and also to identify the key-pair via the ERA UUID. Updates to any of the associated elements, or retrieval of the private-key element, however, must be done through a new set of API's, `sec_rgy_pk_...()`, that implement new protocols providing protection for private-keys and consistency between associated elements.

> *NOTE: Other options, if projection into an ERA is too difficult: allow any principal to use the* `sec_rgy_pk_get_pair()` *API, but return only the public-key; require manual creation of a byte-encoded ERA to hold any public-keys that other principals will need to retrieve.*

6.1.3. Encryption of private-keys

Any private-key stored in the DCE Registry must be stored encrypted using some value that the DCE Registry does not have access to in order to provide protection in case of KDC compromise.

To provide this protection, each private-key stored in the DCE Registry is associated with a password. In the case of the primary private-key, this password is the user's existing long-term password. In the case of secondary private-keys, a new password is specified when the key-pair is created.

A value `H1` is generated from the password using a password hashing algorithm (described below). DES is then used to encrypt the private-key under `H1`.

Another value, `H2`, is generated using a different hashing algorithm. The algorithm used to generate `H2` is the same algorithm currently used to generate a user's key from the user's long-term password. This value is used to verify evidence that a principal knows the password.

```
Client generates:              KDC contains:

         +-----> H1            PubK, H2, {PrivK}[H1]
         |
PW ------+
         |
         +-----> H2

PubK, PrivK
{PrivK}[H1]
```

`H1` and `H2` are susceptible to dictionary attacks since they are derived from a password, which uses only a fraction of the 56-bit DES-key space. For this reason, they should never be transmitted unless further encrypted by a strong DES-key (or public-key). A compromise of the KDC exposes any private-keys stored encrypted under `H1` in the DCE Registry to dictionary attacks. The `H2` derived from the principal's primary password is directly usable for non-public-key authentication by a user capable of generating the wire protocol from a client machine, so a compromise of the KDC immediately compromises any principal with a stored `H2` value if that principal is allowed to log in using only `H2` for authentication.

Use of a Password Management Server can increase the resistance of `H1` to dictionary attacks, and enforced use of any of the public-key login protocols described here (via the `pre_auth_set` ERA) eliminates the use of only `H2` for authentication.

> *NOTE: [IETF 1507] describes such a password hashing algorithm. The following is a modification of that algorithm to fit the DCE context and to use MD5 rather than MD2. The algorithm is still preliminary. Comments are invited.*

Private-keys are encrypted using DES. The key for that encryption is derived from the associated password and the principal's name by the following algorithm:

(a) Put the cell-relative principal name in canonical form.

> *NOTE: The canonical principal name is used to add ''salt'' to the hash calculation so that someone can't precompute the hash of all the words in a dictionary and then apply them against all names. Deriving the salt from the canonical principal name is a compromise. If it were derived from the whole name, all encrypted keys would be obsoleted when a branch of the namespace was renamed. If it were independent of name, login would take two extra messages to retrieve the salt. With this scheme, encrypted keys are obsoleted by a change in the cell-relative principal name and if a cell-relative principal name is common to a large number of users, dictionary attacks against them are easier; but the common case works as desired.*

(b) Compute `TEMP` as the MD5 message digest of the concatenation of the password and the canonical principal name computed above.

(c) Repeat the following 40 times: Use the first 64 bits of `TEMP` as a DES-key to encrypt the second 64 bits; XOR the result with the first 64 bits of `TEMP`; and compute a new `TEMP` as MD5 of the 128 bit result.

(d) Take the final 64 bits of the result (called `hash1`), correct the parity, and use as the key to decrypt the encrypted credentials. The first 64 bits are discarded.

> *NOTE: Bill Sommerfeld recommends using iterated MD5 with a large seed for the* `PW` → `H1` *transform, rather than the [RFC 1507] algorithm:*
>
> *I'd suggest using either MD5 or SHA instead of MD2, since, to the best of my knowledge, both have received somewhat more attention from cryptographers.*

In step (c), the use of DES is ‘‘cute’’ but perhaps is overkill.

For ‘‘iterated MD5’’, I’d just take the 128-bit output of the function and run it through MD5 again the appropriate number of times (40 to 64 sounds about right).

I would assume that the key parity needs to be fixed up before the first 64 bits of `TEMP` *are used as a key. [Some DES libraries do ‘‘unexpected’’ things when passed a key with bad parity.]*

NOTE: [John Wray] In step (c), I think we want to retain the use of DES here, from the [RFC 1507] hashing algorithm. MD5 is too fast. It’s OK to replace MD2 with MD5 in step (b), but I think we need to retain the multiple rounds of DES in step (c) to keep the hashing time high.

6.1.4. Primary key

A principal’s primary public-key is the one used for authentication to DCE.

The primary key is distinguished using a reserved ERA schema entry with the name `pk_primary`. The encoding type may be either `sec_attr_enc_bytes` or `sec_attr_enc_public_key`, depending on whether the private-key half is stored in the DCE Registry.

NOTE: [John Wray, in response to: ‘‘When an account moves from secret-key to public-key, should it get a new password for public-key login, that’s different to its secret-key login password (the one that forms the current DES-key)?’’] I think that the passwords should be the same, so that the user doesn’t have to work out whether they’re sitting at a 1.1 or a 1.2 client system. Also, it should be possible for an administrator to migrate a site to PK use transparently, without having to involve the users.

6.1.5. Secondary keys

Principals may have more than one key-pair, such as keys for encryption of personal email versus keys used to sign official company documents. These additional key-pairs are called ‘‘secondary keys’’. Like primary private-keys, the private-keys for these secondary key-pairs, if stored in the DCE Registry, are stored in an encrypted form that the DCE Registry does not have access to.

Secondary key-pairs may have any ERA schema entry name. The encoding type may be either `sec_attr_enc_bytes` (if the private-key half is not stored in the DCE Registry), or `sec_attr_enc_public_key`.

Users may choose to associate the same password with more than one private-key, however no support for maintaining consistency between such passwords is provided as part of this functionality.

NOTE: If ERA sets are implemented in the future, users will be able to take advantage of these to define sets of key-pair attributes that all share a common password. Tools can then be created to update all members of a set at once.

NOTE: Analysis of options for storage of secondary private-keys:

> *Encrypt under primary public-key.*
- *Requires re-encryption of all secondary private-keys whenever primary private-key is updated.*
- *Cell admin can’t change user password without also changing primary and all secondary keys.*
- *Cell admin can’t change user’s primary public-key without also changing password and all secondary keys.*
- *Cell admin can’t change a secondary key.*
+ *Cell admin can create a new secondary key for a user.*

> *Encrypt under primary private-key’s* `H1`.
- *Requires re-encryption of all secondary private-keys whenever password is updated.*
- *Cell admin can’t change user password without also changing primary and all secondary keys.*
- *Cell admin can’t change user’s primary public-key without also changing password and all secondary keys.*

> *Encrypt under secondary password.*
  − *Multiple passwords per user.*
  − *Cell admin can't change user password without changing associated primary or secondary keys.*
  + *Does not require revealing primary password to an application that uses a secondary password.*
  + *Less linkage (unless principal uses same password for multiple keys, but we might not try to track that).*
  + *Cell admin can change one secondary password without having to change all private-keys.*

6.1.6. KDC public-key

Some of the protocols require the KDC to have a well-known public-key. This is supplied via a new well-known ERA, `pk_KDC`. The value in this ERA is retrieved and copied to a local file as part of `dced secval` startup. It is also copied to a smart card as part of smart card initialization.

The KDC public-key must have a large enough modulus to have a very long lifetime, since changing it requires updating each client node and all smart cards.

> *NOTE: It is OK to cache a foreign* `pk_KDC` *value since these include a* `good_until_date`.

6.2. Login Protocols

Three new pre-authenticated login protocols are proposed in connection with asymmetric-key support, each meeting different goals. All have the common goals of pre-authenticating the principal to the KDC as part of the `KRB_AS_REQ`, and certifying the KDC to the principal as part of the `KRB_AS_REP`.

None of these protocols attempts to protect the TGT in the reply with anything stronger than the current strong DES-key of the KDC. This is because subsequent DCE protocols for requesting service tickets transmit the TGT across the wire encrypted only in that key. Since a network-sniffing DES-key attack can be mounted on any TGT from those frequently-used protocols there is no point in incurring the cost of an extra asymmetric-key encryption to protect the TGT in this one infrequently-used protocol. The defense against all such attacks is periodic change of the strong DES-key of the KDC.

If a principal attempts to log in using a disallowed protocol (see ''Policy Control for Asymmetric-Key'' section), an error is returned that contains identifiers for the set of allowed protocols.

The default initial protocol for DCE 1.2 (as for DCE 1.1) is `PADATA−ENC−THIRD−PARTY` whenever the machine host principal TGT is available, and `PADATA−ENC−TIMESTAMPS` otherwise.

> *NOTE: Encrypting all TGT's in the public-key of the KDC has been suggested to avoid exposure of data on the wire encrypted in the long-term KDC secret-key. Since PTGT's are also encrypted in the long-term KDC secret-key, they would also have to be encrypted in the public-key of the KDC for this to be effective, and this starts to impact performance. We do not plan to do this for DCE 1.2, but it needs to be considered for the future.*
>
> *Analysis:*
>
> + *Removes possibility of network sniffing DES-key attack on KDC long-term secret-key.*
> + *Will not affect clients at all: TGT's are opaque to them.*
> − *PK encryption/decryption is slow; since all PTGT's have to be encrypted, performance will be impacted.*
>
> *NOTE: What issues arise if a server principal is to use public-keys? What threats exist if server principals do not use public-keys, but all interactive principals do?*

6.2.1. PADATA-ENC-THIRD-PARTY variant

This is a variant of the existing DCE 1.1 pre-authentication protocol that returns the principal's primary private-key, encrypted under a hash of the primary password.

Goals:

(a) Do not require the login module to know that the account is required to use asymmetric-key methods when the TGT request is made,

(b) Do not require the login module to have prior access to the principal's primary private-key,

(c) Return the principal's encrypted primary private-key from the DCE Registry as part of the reply for use with certificate or multi-crypto mechanisms.

(d) Require an attacker who has compromised the KDC to mount a dictionary attack on each individual account in order to compromise the account.

In the current DCE 1.1 `PADATA-ENC-THIRD-PARTY` pre-authentication protocol [RFC 26.0], [HOND 94], the client login module generates two random numbers denoted `SK1` — a purely random key — and `SK2` — the ''pre-auth session key''. It then sends a TGT request (`KRB_AS_REQ`) to the KDC with the following PADATA:

```
Login (client)                            KDC
--- M-TGT, {SK1}[M-SK], {{TS,SK2}[H2]}[SK1] --->
```

where `H2` is a hash of the principal's primary password.

If the timestamp can be decrypted and is current, the KDC replies as follows:

```
Login (client)               KDC
<-------- {TGT}[SK2] -----------
```

The proposed modification for use with asymmetric-key methods:

If the client principal has a `pre_auth_set` instance that contains the value `PADATA-ENC-THIRD-PARTY-PK`, and the `KRB_AS_REQ` comes from a DCE 1.2 client, then the KDC replies as follows:

```
Login (client)                            KDC
<--- {TGT}[PubK], {{PrivK}[H1]}[SK2] -------
```

NOTE: The OSF-DCE PADATA sub-type may already contain version# information that will indicate whether the request comes from a DCE 1.1 or DCE 1.2 client. If not, then version# information will need to be added.

No changes to the `KRB_AS_REQ` are required. A new PADATA OSF-DCE sub-type of `PADATA-ENC-THIRD-PARTY-PK` is used in the `KRB_AS_REP` to identify the use of `PubK` for encryption, and the presence of `{PrivK}[H1]` data.

This variant does not work with weaker `KRB_AS_REQ` protocols than `PADATA-ENC-THIRD-PARTY`, since the `{PrivK}[H1]` value needs to be protected by a strong DES-key (`SK2`) for transmission.

The use of `SK2` in the reply serves to certify the KDC, since only the legitimate KDC has access to the principal's `H2` value required to decrypt the `{TS,SK2}` information in the TGT request. The validity of the `{{PrivK}[H1]}[SK2]` information must be verified by the client before using the TGT, because any principal is capable of generating a bogus `{TGT}[PubK]` value.

Use of `PubK` to encrypt the session key associated with the TGT decreases the cost of recovery from a compromise of the KDC, since knowledge of `H1` or the corresponding `PrivK` is required to decrypt the TGT session key, and these are not immediately available upon compromise of the KDC. This decreased cost is obtained only if the principal is not allowed to log in using methods requiring only the knowledge of the primary password's `H2` hash. This can be enforced using appropriate values for `pre_auth_set` ERA instances.

NOTE: Analysis of protocol options for `PADATA-ENC-THIRD-PARTY` *variant:*

> `KRB_AS_REQ`

>> `M-TGT, {SK1}[M-SK], {TS,PW,SK2}[SK1]`
- *Exposes* `PW` *to compromised KDC.*
+ *Proof of* `PW`.

>> `M-TGT, {SK1}[M-SK], {{TS,SK2}[H2]}[SK1]`

0 *Evidence of* `PW`*.*
+ *Does not expose* `PW` *to compromised KDC.*

> `KRB_AS_REP`

>> `{TGT}[PubK], {{PrivK}[H1]}[SK2]`
− *One PK encryption.*
+ *Proof of* `PW`*.*

>> `{TGT}[SK2], {{PrivK}[H1]}[SK2]`
[Assume use of `PW` *in* `KRB_AS_REQ`*.]*
− *Exposes* `PW` *to compromised KDC.*
+ *No PK encryptions.*
+ *Identical to existing protocol, plus encrypted* `PrivK` *info.*

6.2.2. PADATA-ENC-PUBLIC-KEY-KDC

This is a pre-authentication protocol that protects the `KRB_AS_REQ` authentication information using the public-key of the KDC, rather than using the session key of the host machine principal.

Goals:

(a) Do not require prior knowledge of the principal's private-key for authentication,

(b) Do not require use of a host machine principal TGT or session key.

(c) Return the principal's encrypted primary private-key from the DCE Registry as part of the reply.

(d) Require an attacker who has compromised the KDC to mount a dictionary attack on each individual account in order to compromise the account.

This protocol is similar in strength to the `PADATA-ENC-THIRD-PARTY` variant but has the advantage of not requiring use of a host machine principal key.

The client's login module first generates a random reply key `SK`, then issues a TGT request with the following PADATA:

```
Login (client)                          KDC
-------- {{TS,SK}[H2]}[KDC-PubK] ------->
```

If the timestamp can be decrypted and is current, the KDC replies as follows:

```
Login (client)                          KDC
<---- {TGT}[PubK], {{PrivK}[H1]}[SK] ----
```

The `PADATA-ENC-PUBLIC-KEY-KDC` protocol requires definition of a new `PADATA-ENC-PUBLIC-KEY-KDC` sub-type for the OSF-DCE padata-type.

As in `PADATA-ENC-THIRD-PARTY`, the use of `SK` in the reply serves to certify the KDC, since only the legitimate KDC has access to the `H2` value stored in the DCE Registry that is required to decrypt the `TS,SK` information in the TGT request. The validity of the `{{PrivK}[H1]}[SK2]` information must be verified by the client before using the TGT, because any principal is capable of generating a bogus `{TGT}[PubK]` value.

Use of `PubK` to encrypt the session key associated with the TGT decreases the cost of recovery from a compromise of the KDC, since knowledge of `H1` or the corresponding `PrivK` is required, and these are not immediately available upon compromise of the KDC. This decreased cost is obtained only if the principal is not allowed to log in using methods requiring only the knowledge of `H2`. This can be enforced using appropriate values for `pre_auth_set` ERA instances.

NOTE: Analysis of protocol options for `PADATA-ENC-THIRD-PARTY` *variant:*

> `KRB_AS_REQ`

>> `M-TGT, {SK1}[M-SK], {TS,PW,SK2}[SK1]`

- *Exposes* `PW` *to compromised KDC.*
+ *Proof of* `PW`*.*

>> `M-TGT, {SK1}[M-SK], {{TS,SK2}[H2]}[SK1]`
0 *Evidence of* `PW`*.*
+ *Does not expose* `PW` *to compromised KDC.*

> `KRB_AS_REP`

>> `{TGT}[PubK], {{PrivK}[H1]}[SK2]`
- *One PK encryption.*
- *Requires client to be able to use asymmetric-key algorithms in order to use this protocol at all, but some clients may need a private-key but do not need to execute the algorithms [???].*
+ *Proof of* `PW`*.*

>> `{TGT}[SK2], {{PrivK}[H1]}[SK2]`
- *Exposes* `PW` *to compromised KDC (since must be used with* `PW` *protocol to avoid* `H2`*-based authentication).*
+ *No PK encryptions.*

6.2.3. PADATA-ENC-PRIVATE-KEY

This protocol is designed for principals who can store their primary private-key on a smart card or similar personal storage device.

Goals:

(a) Require prior knowledge of the principal's primary private-key for authentication.

(b) Do not require storage of the principal's primary private-key in the DCE Registry, not even in encrypted form.

(c) Do not require storage of a hash of the primary password in the DCE Registry.

(d) Do not require use of a host machine principal TGT or session key.

(e) If the KDC is compromised, eliminate compromise of accounts that use this protocol in conjunction with no storage of a hash of the primary password.

(f) Allow a smart card to sign pre-authentication information rather than reveal the private-key.

The client's login module first generates a random reply key, `SK`, then issues a TGT request with the following PADATA:

```
Login (client)                          KDC
---- {{TS}[PrivK],SK}}[KDC-PubK] ---->
```

If the timestamp can be decrypted and is current, the KDC replies as follows:

```
Login (client)                KDC
<------- {TGT}[SK] ------------
```

The `PADATA-ENC-PRIVATE-KEY` protocol requires definition of a new `PADATA-ENC-PRIVATE-KEY` subtype for the OSF-DCE padata-type.

The use of `SK` in the reply serves to certify the KDC, since only the legitimate KDC knows the `KDC-PrivK` required to obtain `SK`.

The public-key of the KDC should be stored on the smart card or other private-key storage device at the time the smart card is initialized.

In association with use of this protocol, policies to prevent use of primary password based protocols must be used in order to gain the additional decreased cost of recovery from a compromise to the KDC.

NOTE: Analysis of protocol options for `PADATA-ENC-PRIVATE-KEY`*:*

> `KRB_AS_REQ`

>> `{{TS}[PrivK],SK}[KDC-PubK]`
- − *Two PK encryptions.*
- + *Short data block to encrypt with PK.*
- 0 *Proof of* `PrivK`*, but not of* `PW`*: presumably the smart card requires proof of* `PW` *before yielding* `PrivK`*.*
- + *Easy to pass* `{TS}` *to a smart card for encryption under* `PrivK`*.*
- + *Easy to pass* `{SK1}` *to a smart card for encryption under* `KDC-PubK`*.*

> `KRB_AS_REP`

>> `{TGT}[SK]`
- + *No PK encryptions.*
- + *Identical to* `PADATA-ENC-THIRD-PARTY`*.*

>> `TGT, {SK3,SK}[PubK]`
[where `SK3` *is session key of TGT]*
- − *One PK encryption.*
- 0 *Proof of* `PrivK`*, but that was provided in the request.*
- + *Easy to pass* `{SK3,SK}[PubK]` *to a smart card for decryption using* `PrivK.`

>> `{TGT}[PubK]`
- −−− *No certification of the KDC!!*
- − *One PK encryption.*
- + *Proof of* `PrivK`*.*
- + *Easy to pass* `{SK3}[PubK]` *to a smart card for decryption using* `PrivK`*.*

6.3. Key-Pair Password Change

Since a principal's private-keys, if stored in the DCE Registry, are stored encrypted under a key derived from a password, the encrypted private-key value in the DCE Registry must be updated whenever the associated password changes.

6.3.1. Using DES-keys

The current DCE 1.1 primary password change protocol, which operates under authenticated RPC, is:

```
Passwd (client)                KDC
--------- {H2'}[H2] ------------>
```

For change of passwords, primary or secondary, associated with private-keys, a modified protocol is used, operating under authenticated RPC:

```
PK-passwd (client)                                  KDC
-- {PK-UUID, {H2', {PrivK}[H1']}[H2]}[KDC-PubK] -->
```

where `PK-UUID` is the ERA UUID of the key-pair whose password is being changed, `H2'` and `H1'` are hashes of the new password, and `H2` is a hash of the current password.

A status is returned indicating whether the key-pair password change was successfully completed or not. This reply is identical to the reply from the existing primary password change protocol.

The new protocol is implemented by the `sec_rgy_pk_passwd()` API.

Note that asymmetric-key login protocol users will have provided proof of knowledge of either the primary password or the primary private-key as part of becoming authenticated at login.

`H2` provides timely evidence of knowledge of the current key-pair password, thus preventing an unauthorized user from using an authenticated client's unattended session to change a key-pair password.

It is the responsibility of the client API to obtain the `PrivK` value prior to making the key-pair password change request, and to encrypt it under the `H1'` hash derived from the new key-pair password.

> *NOTE: Is there any way to provide strength checking on key-pair passwords without sacrificing the additional security provided by the protocols? The existing password strength checking protocol requires transmission of the password from the client to the server, rather than just transmission of a hash of the password.*
>
> *One possibility is to use asymmetric-key methods to Password Management Server (PMS):*
>
> ```
> Passwd (client)         PMS
> --- {PW, SK}[PMS-PubK] --->
> ```
>
> *If PW meets requirements, PMS returns:*
>
> ```
> Passwd (client)                                    PMS
> <-- {{principal-name,TS,"OK",H2}[PMS-PrivK]}[SK] --
> ```
>
> *Client can then include* `{principal-name,TS,"OK",H2}[PMS-PrivK]` *in the key-pair password change request as timely proof that the password meets strength requirements for this principal.*

6.3.2. Using asymmetric-key methods

No special asymmetric-key protocol is provided, as the protocol using DES-keys does not require any primary password information.

6.4. Key-Pair Generation

A `keygen` module is provided on the client that allows the client to generate key-pairs with optional specified minimum modulus length. The default minimum modulus length is 2K bits if no minimum is specified. The API for this module is `sec_rgy_pk_gen_pair()`. This API executes locally.

6.5. Key-Pair Creation

This functionality allows a key-pair to be associated with a principal. It can be used either by the owner principal or by any other principal with `sec-admin` privileges. It also causes the ERA schema entry associated with the key-pair to be created, if it does not already exist.

6.5.1. Using DES-keys

> *NOTE: Details TBD.*

6.5.2. Using asymmetric-key methods

> *NOTE: Details TBD.*

6.6. Key-Pair Modification

This functionality allows a principal to modify a key-pair associated with that principal.

The `keygen` module is called to generate a new key-pair unless key-pair values are specified. If there is a minimum modulus length policy associated with the key-pair, then that length is passed to the `keygen` module. By default a length of 2K bits is used.

The private-key half may then be stored externally (such as on a smart card), or may be encrypted under the `H1` key derived from the key-pair password. The public-key, and optionally the encrypted private-key, are then transmitted to the DCE Registry using one of the protocols described below.

Key-pairs are long-lived objects, and the key-pair halves can be cached in multiple locations unknown to the DCE Registry. If the key-pair being modified is the primary public-key, the DCE Registry determines that the key-pair is eligible for update given any minimum lifetime policies and the date and time the primary key-pair was last updated. For secondary key-pairs, the UI will warn the user prior to attempting a modification that existing cached public-key values will no longer work, and will prompt for continuation of the operation.

It is an error to modify just the public-key element of a key-pair that currently has a private-key element and associated key-pair password. In order to convert a key-pair to contain just a public-key, a principal with `sec-admin` privileges must execute the key-pair reset operation.

The DCE Registry verifies that the key-pair satisfies any associated policy for minimum modulus length.

If request satisfies any policy requirements, then the DCE Registry updates the values of the key-pair and sets the `good_since_date` to the current date/time, and the `good_until_date` to the current date/time plus any associated maximum lifetime policy value.

If the primary key-pair has an associated an maximum lifetime policy, and if the primary key-pair is allowed to expire, then the principal will not be able to log in until some other principal with `sec-admin` privileges resets the principal's primary key-pair.

> *NOTE: When a password change protocol is implemented that does not require a TGT, then this restriction can be lifted.*

Two protocols are specified for this operation.

6.6.1. Using DES-keys

Under authenticated RPC,

```
PK-modify (client)                          DCE Registry
-- {PK-UUID, {PubK', {PrivK'}[H1]}[H2]}[KDC-PubK] -->
```

where `PK-UUID` is the ERA UUID of the key-pair being modified, `PubK'` is the new public-key, `PrivK'` is the new private-key, and `H1` and `H2` are hashes of the current key-pair password. The `{PrivK'}[H1]` value may be `NULL`.

A status is returned indicating whether the key-pair modification was successfully completed or not.

This protocol is implemented by the `sec_rgy_pk_modify_pair()` API.

Note that asymmetric-key login protocol users will have provided proof of knowledge of either the primary password or the primary private-key as part of becoming authenticated at login.

As in the existing primary password change operation, `H2` is provided in order to provide timely evidence of knowledge of the key-pair password, thus preventing an unauthorized user from using an authenticated client's unattended session to change a key-pair password.

6.6.2. Using asymmetric-key methods

This is an alternate protocol for changing the key-pair using strictly asymmetric-key methods for proof of identity and for confidentiality. The advantage is that the user must prove knowledge of `PrivK`, which requires knowledge of the key-pair password (via knowledge of `H1`), rather than just knowledge of `H2` (which could be obtained from a compromise of the DCE Security Registry).

With a `NULL` value for `{PrivK'}[H1]`, this protocol is also suitable for principals who do not have a primary password `H2` value stored in the DCE Registry, and who do not need to update the `{PrivK}[H1]` half of the key-pair.

```
PK-modify (client)                                  DCE Registry
-- {PK-UUID, PubK', {PrivK'}[H1], {TS}[PrivK]}[KDC-PubK] -->
```

In this protocol, the encryption under `[PrivK]` provides timely authentication of the principal, while the encryption under `[KDC-PubK]` serves to protect the confidentiality of the information across the wire.

> *NOTE: In the future, we probably want a way to present a certificate and have the key-pair associated with the certificate be accepted as the key-pair for the account.*

6.7. Key-Pair Reset

This functionality allows the principal owning a key-pair or another principal with `sec-admin` privileges to overwrite a key-pair with new values for the public-key half, hash of the key-pair password, and encrypted value for the private-key half. It is designed for use in the case where a principal has forgotten the key-pair password associated with a key-pair.

This functionality also allows the private-key element of a key-pair to be removed.

6.7.1. Using DES-keys

```
PK-reset (client)                              DCE Registry
-- {principal-id, PK-UUID, \
   PubK', {PrivK'}[H1'], {H2'}[H2]}[KDC-PubK] -->
```

where `H2` is a hash of the primary password of the principal executing the operation (either owner principal or other principal with `sec-admin` privileges).

NOTE: How do we tell which password was used to encrypt `H2'` *?*

6.7.2. Using asymmetric-key methods

NOTE: Details TBD.

6.8. Key-Pair Retrieval

This functionality allows a principal to retrieve any private-keys stored in the DCE Registry independent of the login operation, for use by protocols that require the private-key (such as change of key-pair password or application use). This avoids the need to cache private-keys on the client.

The public-key half of a key-pair may be retrieved by any principal using normal ERA API's.

The private-key half may be retrieved only through a new `sec_rgy_pk_get_pair()` API that implements one of the following protocols.

6.8.1. Using DES-keys

```
PK-retrieve (client)              Registry
--- {PK-UUID, {TS,SK}[H2]}[KDC-PubK] --->

<------- {PubK, {PrivK}[H1]}[SK] --------
```

where `H2` is a hash of either the owning principal's primary password or of the key-pair password associated with the key-pair.

6.8.2. Using asymmetric-key methods

```
PK-retrieve (client)                 Registry
-- {PK-UUID, {TS, SK}[PrivK']}[KDC-PubK] -->

<-------{PubK, {PrivK}[H1]}[SK]-------------
```

where `PrivK'` is the principal's primary private-key.

NOTE: Cliff Neuman and Brian Tung are working on some public-key enhancements to MIT Kerberos, some of which are functionally equivalent to the work we're doing. I've agreed to work with them, in the hope of minimizing divergence between DCE and Kerberos. I have a feeling that they won't want their protocols to rely on node-keys, so where we have a choice (like in the case above), we may want to use the PK version rather than the third-party version. [John Wray]

NOTE: In the future, the DCE Registry may retrieve ''user public-keys from certificates, either in GDS or in CDS, so if the code can be structured with this possibility in mind, that's goodness, but I don't think it's essential for 1.2''. [John Wray, 11/18/94]

6.9. Key-Pair Deletion

This functionality allows either the owner principal or another principal with `sec-admin` privileges to delete an instance of a key-pair attached to a principal.

6.9.1. Using DES-keys

```
PK-delete (client)                DCE Registry
-- {principal-id, {PK-UUID}[H2]}[KDC-PubK] -->
```

where `H2` is a hash of the primary password of the principal executing the operation (owner principal or other principal with `sec-admin` privileges).

6.9.2. Using asymmetric-key methods

NOTE: Details TBD.

6.10. Asymmetric-Key Policy Control

Besides the asymmetric-key mechanisms, there are policies relevant to the use of these mechanisms. The following functionality is provided to allow administrators to set such policies.

6.10.1. Pre-authentication protocol control

The existing `pre_auth_req` ERA controls the minimum level of pre-authentication that a principal must use. This ERA is obsoleted by a new, multi-valued `pre_auth_set` ERA that specifies the set of pre-authentication types that a principal is allowed to use. The values in the set include the existing values used by `pre_auth_req`, plus additional values for asymmetric-key protocols.

> *NOTE: Could we just change* `pre_auth_req` *from single-valued to multi-valued, and store any current* `pre_auth_req` *value first? This might aid migration if DCE 1.1 clients are coded to handle* `pre_auth_req` *values they don't understand gracefully.*

The KDC replies using the same protocol as used in the request if that protocol is in the principal's `pre_auth_set`. If the requested protocol is not in the principal's `pre_auth_set`, then an error is returned specifying the set of protocols the principal is allowed to use. If a principal has no instances of `pre_auth_set` attached, then the principal is allowed to use any protocol.

There is one exception to this rule: if a `PADATA-ENC-THIRD-PARTY` request is received from a DCE 1.2 client, AND if the principal has `PADATA-ENC-THIRD-PARTY-PK` in the `pre_auth_set`, then the reply uses `PADATA-ENC-THIRD-PARTY-PK`. This saves the client from having to know whether the principal is an asymmetric-key user or not.

The default initial protocol, whenever a machine host principal TGT is available, is `PADATA-ENC-THIRD-PARTY`. If a machine host principal TGT is not available, then the default initial protocol is `PADATA-ENC-TIMESTAMPS`.

The following values are supported for the `pre_auth_set` attribute. The first three correspond to values used in DCE 1.1 with the `pre_auth_req` attribute.

(a) `NONE`

The principal need not use any pre-authentication. It is an error to set this value for a principal with a `NULL` primary password `H2` value.

(b) `PADATA-ENC-TIMESTAMPS`

The principal may use the `PADATA-ENC-TIMESTAMPS` protocol. It is an error to set this value for a principal with a `NULL` primary password `H2` value.

(c) `PADATA-ENC-THIRD-PARTY`

The principal may use the `PADATA-ENC-THIRD-PARTY` protocol. It is an error to set this value for a principal with a `NULL` primary password `H2` value.

(d) `PADATA-ENC-PASSWORD`

The principal is allowed to use a protocol that was partially implemented in DCE 1.1, where the pre-authentication data contains `{TS,PW,SK2}`, rather than `{TS,SK2}[H1]`. This protocol was designed to allow users to get a ticket to change an expired primary password.

(e) PADATA-ENC-THIRD-PARTY-PK

The principal may use the PADATA-ENC-THIRD-PARTY protocol, but if so, and if the request comes from a DCE 1.2 client, then the reply will be the PADATA-ENC-THIRD-PARTY-PK variant. It is an error to set this value for a principal that does not have an instance of pk_primary with a non-NULL private-key attached. It is an error to set this value for a principal with a NULL primary password H2 value.

(f) PADATA-ENC-PUBLIC-KEY-KDC

The principal may use the new PADATA-ENC-PUBLIC-KEY-KDC protocol. It is an error to set this value for a principal that does not have an instance of pk_primary with a non-NULL private-key attached. It is an error to set this value for a principal with a NULL primary password H2 value.

(g) PADATA-ENC-PRIVATE-KEY

The principal may use the new PADATA-ENC-PRIVATE-KEY protocol. It is an error to set this value for a principal that does not have an instance of pk_primary attached.

6.10.2. Expiration of key-pairs

Since asymmetric-key technology requires that key length increase as computing power required to crack the key becomes cheaper, administrators need to have a way to control the lifetime of key-pairs. This is done with a new pk_max_lifetime ERA that is integrated with the login mechanisms. If there is no ERA attached, then there is no expiration on the primary key-pair.

Whenever a primary key-pair is created or updated, the good_until_date is computed and stored with the key-pair. If the pk_max_lifetime ERA value is changed, the good_until_date is not modified until the next time the key-pair is updated.

Whenever a KRB_AS_REQ is received from a principal using one of the new asymmetric-key protocols, then the KDC checks the good_until_date stored with the primary key-pair. If the good_until_date is earlier than the current date, then an error is returned in the KRB_AS_REP indicating that the primary key-pair has expired.

There is no control or enforcement of expiration of secondary key-pairs, although date and time of last update will be stored.

6.10.3. Minimum lifetime for key-pairs

Most key-pairs should be long-lived values, since the public-keys can be cached off-line by correspondents of the key-pair owner(and the private-key can be cached off-line by the owner). For this reason, administrators need a way to prevent principals from updating their key-pairs more often than necessary.

To provide this functionality, a new pk_min_lifetime ERA that is integrated with the primary key-pair change protocol is provided. If no instance of the ERA is attached, then there is no minimum lifetime for the primary key-pair.

Whenever an attempt is made to update the primary key-pair for an account, the time since the previous update (good_since_date is computed by the DCE Registry. If the time is less than the value of pk_min_lifetime, then an error is reported and the update is rejected.

There is no control or enforcement of minimum lifetime for secondary key-pairs, although date and time of last update will be stored.

Even if the key-pair has not expired, the user interface for changing key-pairs will warn the principal of the consequences, and require positive confirmation before proceeding to make the change.

6.10.4. Minimum length for asymmetric-key modulus

The effort required to compute the private-key half of an key-pair is related to the number of bits in the asymmetric-key modulus. The public-key for a very long-lived principal such as the KDC or a Certification Authority needs to have a longer modulus than the public-key for an interactive principal, which in turn needs to have a longer modulus than a public-key used for a single communications session.

Also, as the cost of a given unit of computing power decreases over time, the length of a public-key needs to increase in order to provide the same security.

For these reasons, administrators need to have a means of controlling the minimum length of the modulus of an key-pair.

To provide this functionality, a new ERA, `pk_min_length`, that is integrated with the key-pair change protocols is provided. Instances of this attribute can be attached to a principal in the DCE Registry to control the minimum length in bits of the modulus used in generating key-pairs for that principal's account. If no ERA is attached, then a default length of 2K bits will be used.

Whenever an attempt is made to update the primary key-pair for an account, the value of the effective `pk_min_length` attribute, if any, attached to the principal's node in the Registry is determined. The DCE Registry will not accept a primary key-pair for storage unless it satisfies the `pk_min_length` value.

Note that existing primary asymmetric-keys are not required to be updated when the value of the `pk_min_length` changes.

There is no enforcement of minimum length for secondary key-pairs, although the key-pair generation API will accept a length specification.

7. DATA STRUCTURES

7.1. Key-Pair ERA Schema Entries

Following are all defined in `security/idl/sec_attr_base.idl`.

7.1.1. sec_attr_public_key_t structure

A new structure, `sec_attr_public_key_t` is defined to contain the ERA projection of a public-key that is stored in the DCE Registry.

```
/* sec_attr_public_key_t
 *
 * Used to contain the projection of a public-key value
 * and related information from a sec_passwd_pk_pair_t
 * instance for read-only access via ERA API's.
 *
 * pk_version: version number for this key-pair.
 * good_since_date: date/time when key-pair is first valid.
 * good_until_date: date/time when key-pair expires.
 * public_key_length: length in bytes of the public_key
 *   field.
 * public_key_format: format of the public_key.
 * public_key: array of bytes holding the public-key half
 *   of the key-pair value.
 */
typedef struct {
   sec_key_version_t    pk_version;
   sec_timeval_sec_t    good_since_date;
   sec_timeval_sec_t    good_until_date;
   unsigned32           public_key_length;
   sec_passwd_pk_fmt_t  public_key_format;
   [size_is(public_key_length)] byte public_key[];
} sec_attr_public_key_t;
```

NOTE: Should keys be stored as pickles so that RSA-format keys could be embedded in the key fields in the future? Or is `public_key_format` *sufficient?*

7.1.2. sec_attr_encoding_t enumeration

The existing `sec_attr_encoding_t` is extended with the addition of `sec_attr_enc_public_key`:

```
/* sec_attr_encoding_t
 */
typedef enum {
    sec_attr_enc_any,
    sec_attr_enc_void,
    sec_attr_enc_integer,
    sec_attr_enc_printstring,
    sec_attr_enc_printstring_array,
    sec_attr_enc_bytes,
    sec_attr_enc_confidential_bytes,
    sec_attr_enc_i18n_data,
    sec_attr_enc_uuid,
    sec_attr_enc_attr_set,
    sec_attr_enc_binding,
    sec_attr_enc_trig_binding,
    sec_attr_enc_public_key  /* NEW */
} sec_attr_encoding_t;
```

7.1.3. sec_attr_value_t union

The existing `sec_attr_value_t` union is extended with a case for the encoding type `sec_attr_enc_public_key`:

```
/* s e c _ a t t r _ v a l u e _ t;
 *
 * An attribute value is a union of the various known
 * encodings.
 */
typedef union sec_attr_u
  switch (sec_attr_encoding_t  attr_encoding)
  tagged_union {
    case sec_attr_enc_void:
      ;

    case sec_attr_enc_integer:
      signed32   signed_int;

    case sec_attr_enc_printstring:
      sec_attr_enc_printstring_p_t  printstring;

    case sec_attr_enc_printstring_array:
      [ptr] sec_attr_enc_str_array_t   *string_array;

    case sec_attr_enc_bytes:
    case sec_attr_enc_confidential_bytes:
      [ptr] sec_attr_enc_bytes_t   *bytes;

    case sec_attr_enc_i18n_data:
      [ptr] sec_attr_i18n_data_t   *idata;

    case sec_attr_enc_uuid:
      uuid_t   uuid;
```

```
        case sec_attr_enc_attr_set:
          [ptr] sec_attr_enc_attr_set_t   *attr_set;

        case sec_attr_enc_binding:
          [ptr] sec_attr_bind_info_t   *binding;

        case sec_attr_enc_public_key:       /* NEW */
          [ptr] sec_attr_public_key_t   *public_key;

    } sec_attr_value_t;
```

7.2. Password with Key-Pair Types

Following are all defined in `security/idl/passwd.idl`.

7.2.1. sec_passwd_type_t enumeration

The existing `sec_passwd_type_t` is extended to include a type for a password accompanied by its associated key-pair:

```
/* sec_passwd_type_t
 *
 * ...
 *
 * sec_passwd_pk_pair: a public-key with optional
 *   associated encrypted private-key and hashed
 *   password value.   /* NEW */
 *   Corresponds to type sec_passwd_pk_pair_t.
 */
typedef enum {
    sec_passwd_none,
    sec_passwd_plain,
    sec_passwd_des,
    sec_passwd_pk_pair
} sec_passwd_type_t;
```

7.2.2. sec_passwd_pk_fmt_t enumeration

This enumeration identifies the supported formats for public-key values.

```
/* sec_passwd_pk_fmt_t
 *
 * The supported formats for public-key values.
 *
 * sec_passwd_pk_
 */
typedef enum {
    sec_passwd_pk_fmt_plain,
    sec_passwd_pk_fmt_rsa
} sec_passwd_type_t;
```

7.2.3. sec_passwd_pk_pair_t structure

A new structure, `sec_passwd_pk_pair_t`, is defined to hold a public-key with optional associated encrypted private-key and hash of password used to encrypt the private-key.

```
/* sec_passwd_pk_pair_t
 *
 * Used to contain a public-key value along with
 * an optional encrypted private-key and associated
```

```
 * password hash.
 *
 * Corresponds to sec_passwd_type_t of sec_passwd_pk_pair.
 *
 * pk_uuid: UUID of the ERA that projects the public-key.
 * pk_version: version number for this key-pair.
 *   If 0 (i.e., no value supplied), DCE Registry supplies
 *   current version plus 1 if there is a current version,
 *   and 1 otherwise.
 * good_since_date: date/time when key-pair is first valid.
 * good_until_date: date/time when key-pair expires.
 * public_key_length: length in bytes of the public_key
 *   field.
 * public_key_format: format of the public_key.
 * public_key: array of bytes holding the public-key half
 *   of the key-pair value.
 * private_key_encoding: how private-key is encrypted.
 * private_key_length: length in bytes of the private_key
 *   field.  If 0, then the private_key field is considered
 *   NULL.  This is a valid entry.
 * private_key: array of bytes holding the private-key half
 *   of the key-pair value, encrypted under a
 *   hash of the key owner's passwd (sec_etype_t value of
 *   sec_etype_passwd_hash).
 *
 * DCE Registry records the date and time of the last update
 * to an attribute of this type for use by policies that
 * control minimum or maximum lifetime of asymmetric-key
 * key-pairs.  DCE Registry also uses any pk_max_lifetime
 * ERA value to compute the good_until_date when the
 * key-pair values are updated.
 */
type struct {
   uuid_t              pk_uuid;
   sec_key_version_t   pk_version;
   sec_timeval_sec_t   good_since_date;
   sec_timeval_sec_t   good_until_date;
   unsigned32          public_key_length;
   sec_passwd_pk_fmt_t public_key_format;
   [size_is(public_key_length)] byte public_key[];
   sec_attr_enc_t      private_key_encoding;
   unsigned32          private_key_length;
   [size_is(private_key_length)] byte private_key[];
} sec_passwd_pk_pair_t;
```

7.2.4. sec_passwd_rec_t structure

The existing structure `sec_passwd_rec_t` is extended with a case for a public-key with optional encrypted private-key and associated password hash:

```
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_pk_pair:  /*NEW */
            [ptr] sec_passwd_pk_pair_t
                                 *pk_pair;

    } key;
} sec_passwd_rec_t;
```

NOTE: The primary key-pair should use the existing primary password for the key-pair password. Secondary key-pairs need their own key-pair passwords.

NOTE: [John Wray] I agree that this linkage may complicate the implementation slightly, since secondary keys will have their `H2` *stored with them, whereas primary keys will want to go to the current DES-key object for their* `H2`*, and in all other respects (other than login) primary and secondary keypairs behave the same. Maybe you can create an* `H2` *field for primary keys, but link it to the account DES-key object, so that they both change together. Or better, make the account DES-key field be just a projection of the primary key* `H2` *field for accounts that have a primary key.*

7.3. Asymmetric-Key Encryption Types

Following are all defined in `security/idl/sec_base.idl`.

7.3.1. sec_etype_t enumeration

The existing enumeration type `sec_etype_t` is extended with new types for data encrypted using an asymmetric key.

```
/* sec_etype_t
 *
 * Encryption types supported for data.
 *
 * sec_etype_none: data not encrypted.
 * sec_etype_des_cbc_crc: data encrypted under DES
 *   cipher-block-chaining with
 *   cyclic-redundancy-check.
 * sec_etype_public_key: encrypted using           /*NEW*/
 *   a principal's public-key.
 * sec_etype_private_key: encrypted using
 *   a principal's private-key.
 */
typedef enum {
    sec_etype_none,
    sec_etype_des_cbc_crc,
    sec_etype_public_key,
    sec_etype_private_key
} sec_etype_t;
```

NOTE: How is the associated key used to encrypt an item of data identified? E.g., `SK`, `KDC-PubK`, `PrivK`, `PubK`, `H2`, `H1`, ...

7.4. Well-Known Public-Key ERA's

7.4.1. pk_primary ERA

A new well-known, read-only ERA attribute named `pk_primary` is defined. Instances of this attribute are attached to principal nodes in the DCE Registry database to contain the projection of the principal's primary public-key. The associated key-pair is the one that will always be returned in the reply to a valid TGT request if the `PADATA-ENC-THIRD-PARTY-PK` or `PADATA-ENC-PUBLIC-KEY-KDC` reply protocols are used. The public-key half will be used by the KDC to authenticate the principal if the principal uses the `PADATA-ENC-PRIVATE-KEY` pre-authentication protocol.

It is an error to delete the `H2` value derived from the primary password unless an instance of the `pk_primary` attribute with a `NULL` private-key field is also attached to the principal's node in the DCE Registry. If the principal has a `pre_auth_set` attribute instance containing the values `PADATA-ENC-THIRD-PARTY-PK` or `PADATA-ENC-PUBLIC-KEY-KDC`, then the `pk_primary` attribute must have a non-`NULL` private-key half.

```
attr_name: "pk_primary"
attr_uuid: <TBD>
attr_encoding: sec_attr_enc_public_key
acl_mgr_set: query/update/test/delete
        princ:  <TBD>
        group:  <TBD>
        org:    <TBD>
        policy: <TBD>
default acl settings:
        user_obj:         <TBD>
        group:sec_admin:  <TBD>
        other_obj:        <TBD>
        foreign_other:    <TBD>
        any_other:        <TBD>
unique: FALSE
multi-values: FALSE
reserved: TRUE
apply_defaults: TRUE
intercell_action: accept
trig_types: n/a
comment: "Values: primary public-key."
```

7.4.2. pk_KDC ERA

A well-known ERA named `pk_KDC` is defined. Instances of this attribute are attached to the policy node in the registry database to contain the public-key associated with the KDC.

```
attr_name: "pk_KDC"
attr_uuid: <TBD>
attr_encoding: sec_attr_enc_public_key
acl_mgr_set: Query/Update/Test/Delete
        princ:  <TBD>
        group:  <TBD>
        org:    <TBD>
        policy: <TBD>
default acl settings:
        user_obj:         <TBD>
        group:sec_admin:  <TBD>
        other_obj:        <TBD>
        foreign_other:    <TBD>
        any_other:        <TBD>
unique: FALSE
multi-values: FALSE
reserved: TRUE
```

```
apply_defaults: TRUE
intercell_action: accept
trig_types: N/A
comment: "Values: public-key of KDC."
```

7.5. Asymmetric-Key Policy ERA's

7.5.1. pre_auth_set ERA

To support asymmetric-key methods, a new well-known multi-valued ERA schema entry named `pre_auth_set` is defined to replace the existing `pre_auth_req` ERA. Instances of this attribute are attached to principal nodes in the registry database to indicate pre-authentication protocols allowed to be used by this principal when obtaining a TGT.

If a principal does not have any `pre_auth_set` instances attached, then the principal is allowed to login using any protocol.

Only a principal with `sec_admin` privileges is allowed to attach or modify a `pre_auth_set` attribute instance, since changing it can reduce the security of the account but can be transparent to the user principal.

```
attr_name: "pre_auth_set"
attr_uuid: <TBD>
attr_encoding: integer
acl_mgr_set: query/update/test/delete
   <existing values for pre_auth_req>
default acl settings:
   <existing values for pre_auth_req>
unique: FALSE
multi-values: TRUE
reserved: TRUE
apply_defaults: TRUE
intercell_action: accept
trig_types: n/a
comment: "Values:
   0-need not use any pre-authentication;
   1-may use Kerberos V5 PADATA-ENC-TIMESTAMPS protocol;
   2-may use PADATA-ENC-THIRD-PARTY protocol;
   3-may use PADATA-ENC-PASSWORD protocol;
   4-may use PADATA-ENC-THIRD-PARTY-PK protocol;
   3-may use PADATA-ENC-PUBLIC-KEY-KDC protocol;
   4-may use PADATA-ENC-PRIVATE-KEY protocol."
```

7.5.2. pk_min_lifetime ERA

A well-known DCE Registry attribute named `pk_min_lifetime` is defined. Instances of this attribute are optionally attached to PGO or Policy nodes in the registry database to indicate the minimum lifetime of a primary key-pair associated with the accounts under that node.

```
attr_name: "pk_min_lifetime"
attr_uuid: <TBD>
attr_encoding: integer
acl_mgr_set: Query/Update/Test/Delete
        princ:  <TBD>
        group:  <TBD>
        org:    <TBD>
        policy: <TBD>
default acl settings:
        user_obj:         <TBD>
        group:sec_admin:  <TBD>
```

```
          other_obj:       <TBD>
          foreign_other:   <TBD>
          any_other:       <TBD>
unique: FALSE
multi-values: FALSE
reserved: TRUE
apply_defaults: TRUE
intercell_action: accept
trig_types: N/A
comment: "Values: minimum number of
       days after primary key-pair has
       been changed before it can be
       changed again."
```

7.5.3. pk_max_lifetime ERA

A well-known DCE Registry attribute named `pk_max_lifetime` is defined. Instances of this attribute are optionally attached to PGO or Policy nodes in the registry database to indicate the maximum lifetime of a primary key-pair associated with the accounts under that node.

```
attr_name: "pk_max_lifetime"
attr_uuid: <TBD>
attr_encoding: integer
acl_mgr_set: Query/Update/Test/Delete
        princ:  <TBD>
        group:  <TBD>
        org:    <TBD>
        policy: <TBD>
default acl settings:
        user_obj:        <TBD>
        group:sec_admin: <TBD>
        other_obj:       <TBD>
        foreign_other:   <TBD>
        any_other:       <TBD>
unique: FALSE
multi-values: FALSE
reserved: TRUE
apply_defaults: TRUE
intercell_action: accept
trig_types: N/A
comment: "Values: maximum number of
        days after primary key-pair has
        been changed before it must be
        changed again."
```

7.5.4. pk_min_length ERA

A well-known DCE Registry attribute named `pk_min_length` is defined. Instances of this attribute are optionally attached to PGO or Policy nodes in the registry database to indicate the minimum length of the modulus of a primary key-pair generated for this principal.

```
attr_name: "pk_min_length"
attr_uuid: <TBD>
attr_encoding: integer
acl_mgr_set: Query/Update/Test/Delete
        princ:  <TBD>
        group:  <TBD>
        org:    <TBD>
```

```
        policy: <TBD>
default acl settings:
        user_obj:         <TBD>
        group:sec_admin:  <TBD>
        other_obj:        <TBD>
        foreign_other:    <TBD>
        any_other:        <TBD>
unique: FALSE
multi-values: FALSE
reserved: TRUE
apply_defaults: TRUE
intercell_action: reject
trig_types: N/A
comment: "Values: minimum number
        of bits required in the modulus of
        a primary key-pair generated for
        this principal."
```

8. USER INTERFACES

NOTE: These user interfaces are all very preliminary.

8.1. Login UI

The user interfaces for login do not change. The user must still supply a principal name and a primary password.

8.2. Key-Pair Password Change UI

If the private-key is not specified explicitly, the private-key is automatically retrieved using the key retrieval API.

```
dcecp> account modify <principal> -change \
           -publickey \
           {pk_primary {{password abcxyz} {oldpassword xyzabc}}
```

8.3. Key-Pair Generation UI

```
dcecp> account generate -publickey {length 2048}
```

8.4. Key-Pair Creation UI

The `keygen` module is invoked automatically if key-pair values are not specified.

Examples:

```
dcecp> account modify <principal> -add \
            -publickey \
            {pk_primary {password xyzabc}}
```

or:

```
dcecp> account modify <principal> -change \
           -publickey {pk_primary \
            {version 5} \
            {publickey "iQCVAwUBLuOfprT+rHlVUG
             pxAQGxKgP/WxcvwKD5EpApwCm31XtEast+
             XPR5h3oJZogOV+5dAR05GFV4RI/c11yg0v
             iQYID+0w2+M5bGOqiAI+ujGK58JxRE5BoHO
             TDw8c5jXVxUnycmC9iRmts8V8Fcu7jinvWs
             IVD15IqmFZZFYyuosUzoEYzltyHt9XdP
             rX+cyQp2c4E==w6Yp"} \
            {privatekey 0}}
```

NOTE: Whitespace embedded in public-key strings in this document is to be ignored (it's included here only for typographical display purposes).

8.5. Key-Pair Modification UI

`Keygen` module is called to generate a new key-pair if no values are specified.

Examples:

```
dcecp> account modify <principal> -change \
          -publickey {pk_primary {password xyzabc}} \
          -mypwd abcxyz
```

or:

```
dcecp> account modify <principal> -change \
          -publickey {pk_primary \
           {version 5} \
           {publickey "iQCVAwUBLuOfprT+rHlVUG
            pxAQGxKgP/WxcvwKD5EpApwCm31XtEast+
            XPR5h3oJZogOV+5dAR05GFV4RI/c11yg0v
            iQYID+0w2+M5bGOqiAI+ujGK58JxRE5BoHO
            TDw8c5jXVxUnycmC9iRmts8V8Fcu7jinvWs
            IVD15IqmFZZFYyuosUzoEYzltyHt9XdP
            rX+cyQp2c4E==w6Yp"} \
           {privatekey 0}} \
         -mypwd abcxyz
```

Since users should not update key-pairs more often than necessary, the following warning is displayed to `stdout`:

```
Public-key attributes are very secure, and should not be
changed often.  If you change your public-key attribute,
other principals that have stored your current public-key
may not be able to communicate with you until they have
updated their key copies.  Do you want to continue [no]?
```

If the user enters `yes`, then the operation proceeds as described in the ''FUNCTIONAL DEFINITION'' section.

8.6. Key-Pair Reset UI

NOTE: Details TBD.

8.7. Key-Pair Retrieval UI

The existing user interfaces for retrieval of ERA instance values may be used to retrieve public-keys from the DCE Registry.

Example:

```
dcecp> principal show <principal> -xattrs
```

Retrieval of the private-key requires:

```
dcecp> account show <principal> \
       -publickey {pk_primary {password xyzabc}}
```

or:

```
dcecp> account show <principal> \
       -publickey {pk_primary} \
       -mypwd abcxyz
```

8.8. Key-Pair Deletion UI

NOTE: Details TBD.

8.9. Asymmetric-Key Policy Control UI

The existing user interfaces for adding, modifying, and deleting ERA instances are used for manipulating instances of the new ERA's associated with asymmetric-key.

For example:

```
dcecp> principal modify <principal> -change \
                {{pre_auth_set 4} {pre_auth_set 5}}
```

9. API'S

NOTE: These are all very preliminary.

9.1. Login API's

The existing DCE login API's are not modified. The login protocol changes are contained entirely inside the existing `sec_login_*` API's.

9.2. Key-Pair Password Change API

This new API must used to change a password associated with a key-pair, either primary or secondary. An error is returned if a principal having a primary private-key stored in the DCE Registry attempts to use the existing DCE 1.1 password change protocol to change the primary password.

9.2.1. sec_rgy_pk_passwd()

NOTE: Details TBD.

9.3. Key-Pair Generation API

This new API, executed locally on the client side, generates a new asymmetric-key key-pair with an optionally specified minimum modulus length.

9.3.1. sec_rgy_pk_gen_pair()

NOTE: Details TBD.

9.4. Key-Pair Creation API

This new API allows a new key-pair associated with a principal to be created in the DCE Registry. An ERA of encoding type `sec_attr_enc_public_key` is created to hold the projection of the public-key.

9.4.1. sec_rgy_pk_create_pair()

NOTE: Details TBD.

9.5. Key-Pair Modification API

This new API allows the principal to change the value of a public-key associated with the principal. If there is an encrypted private-key associated with the public-key, then the encrypted private-key portion must be updated as well.

9.5.1. sec_rgy_pk_modify_pair()

NOTE: Details TBD.

9.6. Key-Pair Reset API

This new API allows an owner principal or a principal with `sec-admin` privileges to overwrite the public-key, password, and encrypted private-key for a key-pair associated with a principal. This is used in cases where the principal has forgotten the password associated with the key-pair, or where the principal wishes to delete password and encrypted private-key from the DCE Registry.

9.6.1. sec_rgy_pk_reset()

NOTE: Details TBD.

9.7. Key_Pair Retrieval API

The existing API's for manipulating ERA instances may be used to retrieve public-key values from the DCE Registry.

Retrieval of private-key information requires use of the following new API.

9.7.1. sec_rgy_pk_get_pair()

NOTE: Details TBD.

9.8. Key-Pair Deletion API

This new API is used by an owner principal or by a principal with `sec-admin` privileges to delete a key-pair associated with the owner principal from the DCE Registry.

9.8.1. sec_rgy_pk_delete_pair()

NOTE: Details TBD.

9.9. Asymmetric-Key Policy Control API's

The existing ERA API's are used to create, retrieve, modify, and delete the ERA's associated with asymmetric-key policies.

10. REMOTE INTERFACES

NOTE: These are all very preliminary.

NOTE: A new error type needs to be defined for return with a `KRB_AS_REP` *that can contain identifiers for the set of permitted login protocols. This error type would be used when the principal used a protocol that was not in the principal's* `pre_auth_set`.

Current error structure:

```
typedef struct _krb5_dce_pa_err_info {
    krb5_error_code   err;      /* internal preauth
                                   error */
    krb5_preauthtype  type;     /* min. type preauth
                                   allowed */
    krb5_preauthtype  subtype;  /* min. subtype preauth
                                   allowed */
    krb5_data         salt;     /* required salt */
} krb5_dce_pa_err_info;
```

10.1. KRB_AS_REQ Remote Interface

For `PADATA-ENC-THIRD-PARTY`, this is unchanged. For `PADATA-ENC-THIRD-PARTY-PK`, `PADATA-ENC-PUBLIC-KEY-KDC` and `PADATA-ENC-PRIVATE-KEY`, new OSF-DCE PADATA sub-types are defined and used as described in the ''FUNCTIONAL DEFINITION'' section.

10.2. KRB_AS_REP Remote Interface

Depending on the PADATA type used in the request, the reply is as described in the ''FUNCTIONAL DEFINITION'' section. For some protocols, as described in the ''FUNCTIONAL DEFINITION'' section, the PADATA field in the reply may contain an encrypted value for the user's primary private-key, encrypted in a one-way hash of the primary password, as described in the ''FUNCTIONAL DEFINITION'' section.

NOTE: The algorithm used for message encryption is already a field. Define a new one for `{PrivK}[H1]`.

10.3. Key-Pair Creation Remote Interface

Used to create a key-pair in the DCE Registry.

> *NOTE: Details TBD.*

10.4. Key-Pair Modification Remote Interface

Used to update a key-pair.

> *NOTE: Details TBD.*

10.5. Key-Pair Reset Remote Interface

Used to overwrite a key-pair and its associated key-pair password.

> *NOTE: Details TBD.*

10.6. Key-Pair Deletion Remote Interface

Used to delete a key-pair from the DCE Registry.

> *NOTE: Details TBD.*

10.7. Asymmetric-Key Policy Control Remote Interfaces

The existing ERA remote interfaces are used to handle asymmetric-key policy control ERA values.

11. MANAGEMENT INTERFACES

No new management interfaces are required by this component.

12. RESTRICTIONS AND LIMITATIONS

12.1. Threat Analysis

This section analyzes the existing threats to DCE 1.1 Security and shows how they are affected by the new functionality.

12.1.1. Administration policy

Setting and maintaining consistent DCE Security policies is a current weak point in DCE Security.

> *NOTE: Details TBD.*

12.1.2. ACL's

Setting and maintaining consistent ACL's on DCE Security policy-related information is a current weak point in DCE Security.

> *NOTE: Details TBD.*

12.1.3. Off-line password-guessing attack

Using the vanilla Kerberos authentication protocol, an unauthorized user with normal login access to a client machine can request a TGT for any authorized user. The session key associated with the returned TGT, which is required to use the TGT for authentication, is encrypted in the long-term key of the authorized user. A user's long-term key is a one-way hash of the user's primary password, so the long-term key is subject to an off-line dictionary attack through the encrypted session key.

This threat is effectively eliminated by requiring the use of the DCE 1.1 Pre-Authentication Protocol, in which a TGT is returned only when the request includes proof of knowledge of the long-term key.

Required use of any of the new asymmetric-key protocols also effectively eliminates this threat. All new asymmetric-key protocols require pre-authentication information, and this information is protected as in the DCE 1.1 Pre-Authentication protocol by a strong random DES-key. In the case of the `PADATA−ENC−PRIVATE−KEY` protocol, the pre-authentication information is protected by an asymmetric key, and can be considered stronger than the DCE 1.1 Pre-Authentication protocol.

This threat can also be greatly reduced by use of a DCE 1.1 Password Management Server to enforce password strength policies. The new protocols that depend on passwords can also benefit from enforcement of stronger passwords by a Password Management Servers.

12.1.4. Active password-guessing attack

An unauthorized user with normal login access to a client machine can request a TGT for any authorized user, supplying a guessed password.

This threat is effectively eliminated by the use of the DCE 1.1 Extended Login mechanism for limiting the number of successive invalid access attempts on an account. The invalid access attempts mechanism also works with all the new asymmetric-key protocols.

This threat can also be greatly reduced by use of a DCE 1.1 Password Management Server to enforce password strength policies. The new protocols that depend on passwords can also benefit from enforcement of stronger passwords by a Password Management Servers.

12.1.5. Network sniffing password-guessing attack

An unauthorized user with access to cell network traffic can capture a vanilla Kerberos V5 `KRB_AS_REP`, which includes a session key encrypted in some authorized user's long-term key, and subject the message to an off-line password-guessing attack as above.

This threat is eliminated by the use of the DCE 1.1 Pre-Authentication Protocol, in which the `KRB_AS_REP` is encrypted not in the authorized user's long-term key derived from a password, but in a strong DES session key.

All new asymmetric-key protocols require pre-authentication information protected by at least strong random DES-key. In the case of the `PADATA-ENC-PRIVATE-KEY` protocol, the pre-authentication information is protected by an asymmetric key, and can be considered effectively immune from this attack.

12.1.6. Network sniffing DES-key attack

An unauthorized user with access to cell network traffic can capture a pre-authenticated `KRB_AS_REQ` and subject the pre-authentication data to a DES-key attack (much more expensive than a dictionary attack), yielding the long-term machine host key. Then, the pre-authentication data can be further subjected to an off-line dictionary attack, yielding the authorized user's long-term key and the authorized user's password.

Alternatively, an unauthorized user with access to cell network traffic can capture any `KRB_AS_REP` and subject the TGT to a DES-key attack, yielding the long-term key of the KDC. With this key, the unauthorized user can generate a bogus TGT in the name of any authorized principal, and can use this TGT to obtain a PTGT or service ticket to any server, including the DCE Registry and the KDC.

Enforced use of the `PADATA-ENC-PRIVATE-KEY` protocol via deletion of the `H2` value derived from the primary password and through `pre_auth_set` attributes eliminates vulnerability to this threat. The other asymmetric-key protocols specified have the same level of vulnerability as the existing DCE 1.1 Pre-Authentication protocol.

12.1.7. KDC DES-key attack

An unauthorized user with root access to the KDC server machine can capture the master key of the DCE Registry. The master key can then be used directly to obtain the long-term keys of every authorized user. In the case of interactive users, these long-term keys be used directly for authentication or can be subjected one-by-one to an off-line dictionary attack, yielding the user's password.

The threat of compromising user's passwords can be reduced by using a Password Management Server to make the off-line dictionary attack much harder, but the difficulty is still considerably less than a DES-key attack, and the long-term keys can still be used directly by an attacker who has access to the low-level protocols.

Direct use of the long-term key can be eliminated by using the `PADATA-ENC-THIRD-PARTY-PK` or `PADATA-ENC-PUBLIC-KEY-KDC` protocols, forcing the attacker to subject each long-term key to an off-line-dictionary attack. The attacker could still attack the cell_admin's long-term key and use it to change other keys, but this eliminates the authorized user's ability to log in, exposing the attack. The cell_admin's long-term key could also be used to add new accounts.

All use of the long-term key can be eliminated, and this threat removed, for principals who delete the `H2` value associated with the primary private-key in conjunction with the `PADATA-ENC-PRIVATE-KEY` protocol. Sites where this protocol is available should enforce its use on any account with `sec-admin` privileges in order to eliminate the threat of changing keys and adding new accounts.

13. OTHER COMPONENT DEPENDENCIES

(a) S0 — Public-Key License

(b) S13 — Smartcard and Token Card Integration. While this did not make the cut for DCE 1.2, the functionality, interfaces, API's, and data structures are closely connected, and should be designed to be consistent.

(c) Sxx — Multi-Crypto, since may want to define new encryption types for keys, etc., that would be used by both projects.

14. COMPATIBILITY

A principal with all `pre_auth_set` attribute instances set to values greater than `PADATA-ENC-THIRD-PARTY` will not be able to log in from pre-DCE 1.2 clients.

A utility will be provided to create new `pre_auth_set` attributes equivalent to all existing `pre_auth_req` attributes, for use by sites migrating from DCE 1.1 to DCE 1.2. Once all cell nodes are at DCE 1.2, the `pre_auth_req` attributes can be removed.

Principals with private-keys stored in the DCE Registry encrypted under their password will not be able to change their passwords from pre-DCE 1.2 client nodes.

Principals will not be able to manipulate attributes using new encoding types for key-pairs from pre-DCE 1.2 client nodes.

15. STANDARDS

(a) IETF RFC 1507 (DASS).

(b) IETF RFC 1510 (Kerberos V5).

(c) DCE Security AES.

16. OPEN ISSUES

(a) DIVISION OF WORK

This work is being shared by HP and DEC. Both companies will work on the specifications and write the overview documentation materials. A tentative division of work is as follows.

DEC will provide:

(i) The actual public-key encryption and public-key generation modules [IETF 1507] that will be called from the `sec_*` API's.

(ii) The ASN.1 specification for the new `KRB_AS_REQ/REP` and other protocols.

(iii) All client-side code, including an implementation specification to be reviewed by HP; first draft to include login API's for purpose of reviewing match to potential smart card interfaces and protocols; due mid-March 1995.

(iv) Documentation related to above work.

HP will provide:

(i) `mavros` processing of ASN.1 specifications of `KRB_AS_REQ/REP`, or hand-coding.

(ii) `secd` glue to use the new `KRB_AS_REQ/REP`

(iii) Modifications to the DCE Registry and `secd`, including an implementation specification. First draft to include encoding types for pk-pairs for purpose of reviewing match to potential smart card interfaces and protocols; due mid-March 1995.

(iv) `dcecp` work to support new asymmetric-key encoding type attributes.

(v) Documentation related to above work.

(b) NEW PASSWORD-HASHING ALGORITHM

Exact algorithm to be used.

(c) PASSWORD STRENGTH CHECKING

Requires plaintext password.

> *NOTE: Use public-key methods?*

(d) STORAGE IN LOGIN CONTEXT

Whether and how to store public-key information locally.

> *NOTE: Sign a short-term certificate with private-key? Leave definition up to multi-crypto and certificate API's? API to store arbitrary typed data in the login context?*

(e) SERVERS AND PUBLIC-KEYS

> *NOTE: What is required for servers to use public-keys? What threats exist if interactive users use public-keys, but servers do not?*

APPENDIX A. LIST OF SECTION TITLES

REFERENCES

[FAQ 93] Usenet sci.crypt newsgroup, ''Cryptography FAQ'', Version 1.0, 1/11/94.

[HOND 94] Hondo, M., ''DCE 1.1 Functional Specification — Extended Login and Password Management'', Revision 1.5, March 4, 1994.

[HOPK 93] Hopkins, A., and Bennett, M., ''DCE 1.1 Functional Specification — Extended Registry Attributes'', Version 2.2, November 19, 1993.

[IETF 1507] Kaufman, C., ''DASS — Distributed Authentication Security Service'', IETF RFC 1507, Network Working Group, September 1993.

[RFC 26.0] Pato, J., ‘‘Using Pre-Authentication to Avoid Password-Guessing Attacks’’, OSF RFC 26.0, June 1993.

[RFC 68.0] Wray, J., ‘‘Public-Key Login for DCE 1.2’’, OSF RFC 68.0, November 1994.

[WIEN 93] Wiener, Michael J., ‘‘Efficient DES Key Search’’, presented at Crypto ’93, Aug 20, 1993.

AUTHORS’ ADDRESSES

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

John Wray
Digital Equipment Corporation
550 King Street, LKG2-2/Z7
Littleton, MA 01460
USA
Internet email: wray@tuxedo.enet.dec.com
Telephone: +1-508-486-5210