TokenKey (Enrollment) Archival/Recovery Design

Note: This document is informational only. Numerous changes have been made to the underlying source code since this document was originally written. Although every effort has been taken to ensure that changes have been reflected in this document, some discrepancies may exist. Please refer to the source code and headers in the code repository for authoritative information.

0. Introduction

This design covers the following requirements for Certificate System:

- A user should be able to easily get a usable replacement if his token was lost, stolen, broken.
- A user should be able to temporarily get a usable replacement if the token was left at home

The design accomplishes these goals by:

- Allowing the administrator to specify the location where each private key is generated (either on the token, or on the server).
- Automatically archiving private keys
- Automatically generating new signing keys for a new token
- Blocking a new token enrollment until the administrator approves the recovery
- Allowing the user to select keys to recover

Detailed task list

TPS (Token Processing Service)

- Enrollment processor options (config params and logic) for
  - Server-side key generation
  - Key archival for encryption keys
  - Key recovery for encryption keys
- TPS interation with DRM
  - **connection configuration**
  - **protocol**
- New Token Database attributes (for recovery use later)
  - lost - set by admin when a token is reported is lost
  - recoverable - the key generated for this token can be recovered
  - certmetainfo - information about the certs for this token
- new PDUs to inject server-generated keys back to token
- new op = recovery (addon to the existing ops)
- new protocol for the recovery service

TKS (Token Key Service)

- Kek session key generation JNI function
- configuration to retrieve DRM-transport-key with a nickname.
- use DRM-transport-key to wrap kek session key
- change of ComputeSessionKey and TPS-TKS protocol to return DRM-transport-key wrapped kek session key.

DRM (Data Recovery Manager)

- new servlet to handle encryption key generation and archival on DRM
  - key archival support
  - kek session key unwrapping
  - wrapping of user private key with kek session key
  - protocol definition of what to send back (wrapped user private key, user public key, etc.)
- new servlet to handle encryption key recovery on DRM

CA (Certificate Authority)

- store cuid with certs in the internal databas (?>

1. Archival

High-level Design

Configuration parameters

The TPS administrator can set parameters for each token key. New parameters will be added to the configuration to allow the option to generate a key on the server. A separate option will be provided to enable key archival

Note: In this first phase of token key management system, we will not support multiple DRMs that do not share the same database (cloned instances for fail-over are ok).
An example configuration for encryption keys follows. The format is: op.enroll.<TokenType>.keyGen.<KeyType>.<Parameters>

```
############

# Server-Side Keygen and Archival

#   key archival can be
turned on if server-side keygen is enabled

#   No key archival option
is provided if user keys are not generated on server

############


#only encryption certs would get the
serverKeygen option branch

#keygen, archival and recovery have to be on the same drm instance

# *** There is a suggestion that the "recovery" option should be put on
drm

op.enroll.userKey.keyGen.encryption.serverKeygen.enable=on

op.enroll.userKey.keyGen.encryption.serverKeygen.archive=true

op.enroll.userKey.keyGen.encryption.serverKeygen.drm=drm1

op.enroll.userKey.keyGen.encryption.serverKeygen.drmNickname=?DRM1transportCert
```

New configuration parameters for DRM connection pool are detailed below:

```
############

# DRM (handles keygen, archival, and recovery on the same machine)

############

conn.drm.totalConns=1

conn.drm1.hostport=paw.sfbay.redhat.com:8104

conn.drm1.clientNickname=Server-Cert

conn.drm1.servlet.?GenerateKeyPair=/kra/?GenerateKeyPair


conn.drm1.retryConnect=3

conn.drm1.?SSLOn=true

conn.drm1.keepAlive=false
```

TPS is the sole server contact for tokens. Among all CS servers, it is the only one that understands PDUs and speaks to the tokens. TPS talks to TKS, DRM, and CA, while TKS, DRM and CA do not speak to each other in this Token Management system.

Here is the general logic flow of an enrollment with server-side keygen and key archival:

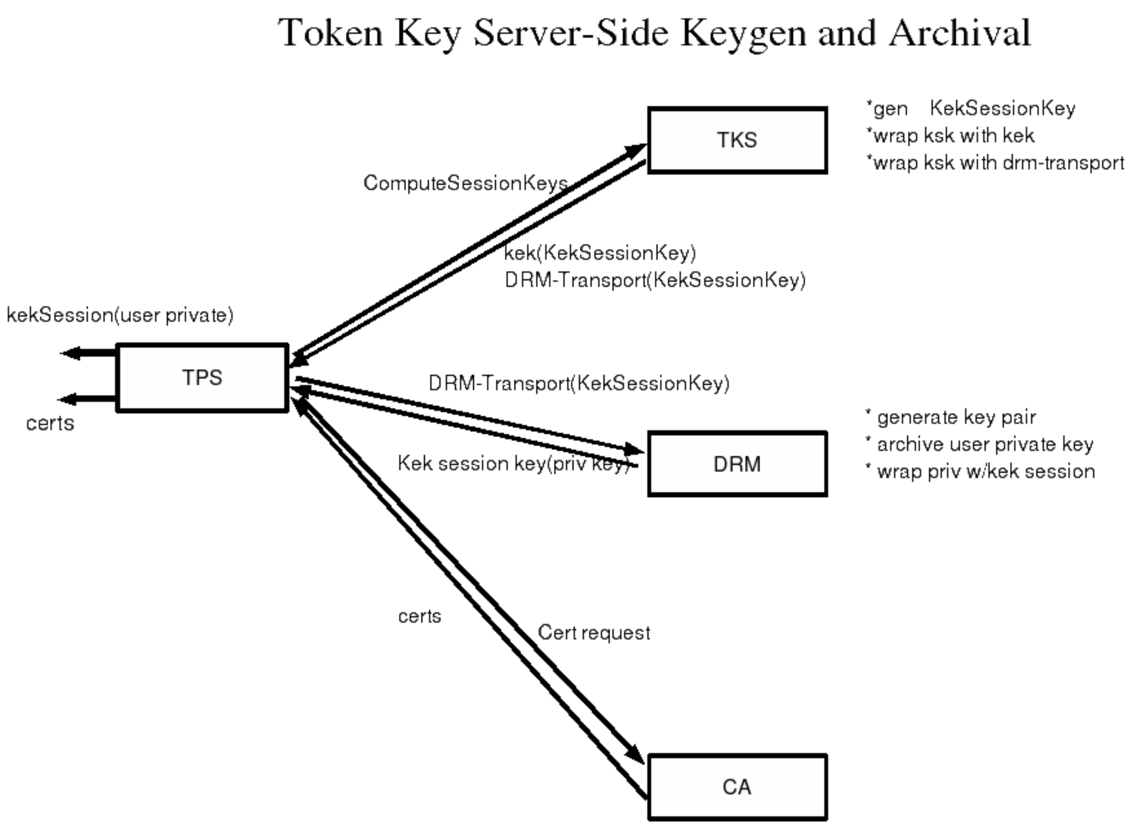
1. TPS sends "ComputeSessionKeys" to TKS (need extend on existing call)
    1. TKS derives KekSessionKey (KSK) and wrap it with KEK; TKS retrieves DRM-transport key, wrap KSK with it.
    2. TKS sends KEK(KSK) and DRM-transport(KSK) back to TPS
2. TPS sends DRM-transport(KSK) and key gen request to DRM (new "keygen/keyArchival" servlet needs to be written)

    1. DRM generates key pair
    2. DRM archives private key
    3. DRM wraps private key with kek session key
    4. DRM sends ksk(private key) and public key to TPS
3. TPS sends KSK(user private key) + user public key + KEK(KSK) back to token
4. TPS forms cert requests and send to CA
    1. CA generates cert and returns to TPS
5. TPS sends cert back to token.

DRM transport cert

DRM transport cert is to be manually added to the TPS' certdb.

Flow


Token Key Server-Side Keygen and Archival
TKS
*gen KekSessionKey
*wrap ksk with kek
*wrap ksk with drm-transport
ComputeSessionKeys
kek(KekSessionKey)
DRM-Transport(KekSessionKey)
kekSession(user private)
TPS
certs
DRM-Transport(KekSessionKey)
Kek session key(priv key)
DRM
* generate key pair
* archive user private key
* wrap priv w/kek session
certs
Cert request
CA

Multiple DRMs sharing the database by ldap cloning is to be supported.

Low-Level Design

Pseudocode in this section that indicates **new code** are in **bold**.

TPS

When an enrollment request comes into the RA_Enroll_Processor(), it does the following to loop through every key type (key types are defined in CMS.cfg with defaults: *auth*, *signing*, and *encryption)*.
The new feature requires to give admins an option to do server-side keygen and optional archival of the encryption keys. It is therefore necessary to put within the DoEnrollment() method code segment to trigger server-side keygen (and archival).

```
in RA_Enroll_Process.cpp
RA_Enroll_Processor()
{
...
    for (every key type) {
       ...Do Enrollment...
          if (server-side keygen desired and keytype is encryption)
             call serverSideKeyGen()
             inject server-generated keys into token with PDUs.
          else
          ...StartEnrollment...
             ...Generate key APDU send to token...

          ...EnrollCertificate...
    }
...
}
```

```
in RA.cpp
/**
* This new function sends a request to the DRM to generate user keys
*/
serverSideKeyGen()
{
...
sends along the kek session key (wrapped in DRM transport key) for DRM to wrap user private key.
}
```

```
in RA.cpp
ComputeSessionKey() {
...
    retrieves kek session key wrapped with DRM transport key (to be used to wrap user private key)
...
}
```

DRM

```
ns/cms/classsrc/com/netscape/cms/servlet/connector/GenerateKeyPairServlet.java
    process serverSideKeyGen requests (from TPS).
       retrieves drm-transportkey-wrapped kekSessionKey
          send to request processor (NetkeyKeygenService.java)
```

```
ns/cms/classsrc/com/netscape/cmscore/kra/?NetkeyKeygenService.java
       generates user key pair
       archive private key
       wrap user private key with kekSessionKey (came with request,wrapped with DRM transport key)
```

```
ns/cms/classsrc/com/netscape/cmscore/kra/EncryptionUnit.java
    functionality to handle some token specific wrapping and unwrapping.
```

NOTE: there is a concern that Luna SA does not allow export of private keys, so it is suggested that we make a suggestion to customers to set up a Luna RA for generation of user key pairs (temporary).

TKS

```
ns/cmsutils/jni/com/netscape/cms/servlet/tks
  RASessionKey.cpp
       ComputeKekSessionKey  (jni implementation)
       ComputeKekKey (jni implementation)
```

```
ns/cms/classsrc/com/netscape/cms/servlet/tks
  RASessionKey.java
       ComputeKekSessionKey
```

        ComputeKekKey
  TokenServlet.java
    ProcessComputeSessionKey
        generates kek session key, wrap it with kek key, and separately wrap it with DRM transport key.
        send along with existing params back to TPS

2. Recovery/Re-Enrollment

This section covers re-enrollment. The goal here is to:
- Get new signing keys/certs
- Get new encryption keys/certs
- Recovery old encryption keys/certs

There is a requirement to enforce a policy where the user can only have one active token at a time. If they lose their token, the token processing system must not let them enroll a new token. If the user loses their token, they need to ....
NOTE: During recovery, under the current DRM behavior, the n/m key splitting policy is applied to DRM storage key retrieval in order to retrieve user private keys. There is plan to change the key splitting policy so that the policy of n/m is enforced by ACL instead. In this design, it is our assumption that the application is already logged in to the token on which the Storage Key sits.

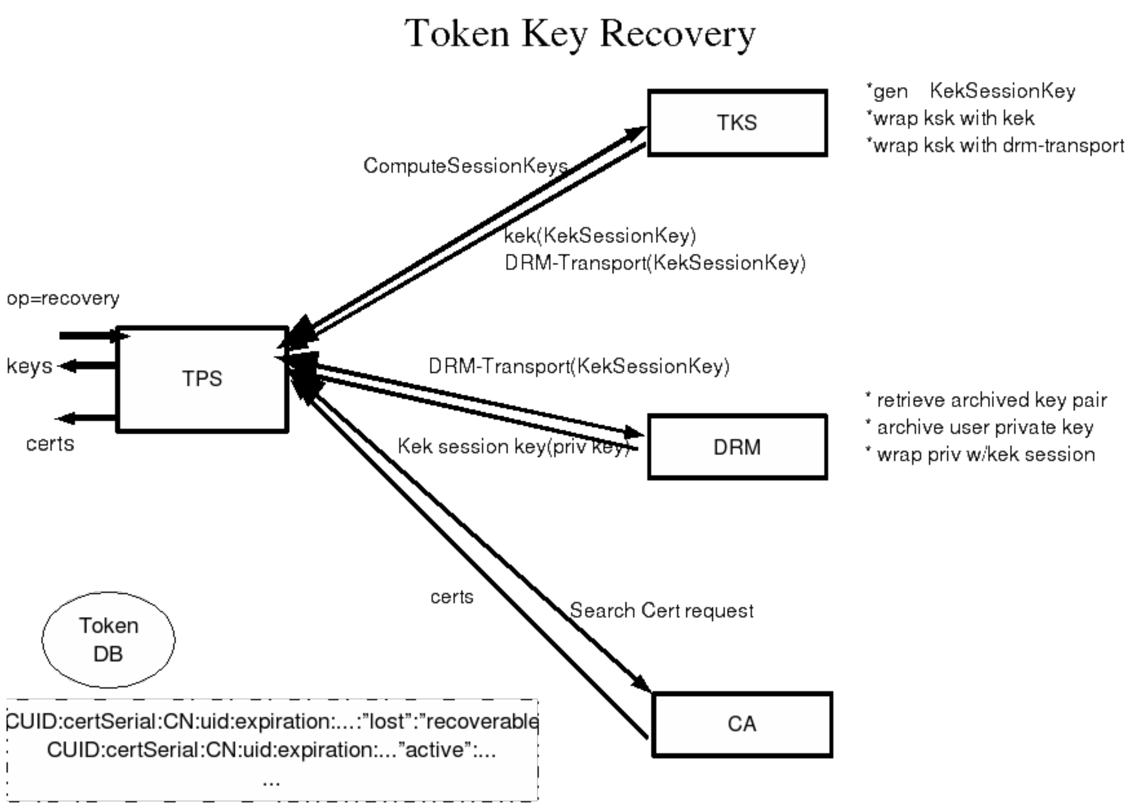
High-level Design

1. lost/stolen key

User case:
- User lost token, report to helpdesk
- helpdesk mark the token entry in token db as "lost" and "recoverable".
- user gets a new token and plugs it in. New enrollment assumes.
- user can choose to "recover" (client provides button), and is presented with a list of recoverable certs/keys.
- user should be able to tell from the client how many keys are permitted to restore (space limitation).

In this case, token db needs to be added with the following fields for each token entry:
- "status" should have an additional (if not yet there) "lost" value.
- new "recoverable" field with "ture/false" values
- one "cert" entry for each cert on each token with information enough to identify a cert (for use by ESC to display to user for recovery selection):
    - "certinfo": serial number, keytype, cn, uid, expiration, etc.
- auto-revoke associated certs if configured

Here is the logic flow of a recovery operation:
1. User clicks "recover key/cert" on client, such as ESC to TPS. Token sends op=recovery, along with authentication data (requested by whichever authentication module that's configured by TPS).
2. TPS authenticates user, then verifies with token db (search by user id) to find the lost token (could there be more than one lost token at a time? Maybe policy should be only one token belonging to the same user can be "recoverable" at a time) entry, and search for all "certinfo" attributes for the token's entry and send them back to client for selection.
3. Upon receival of the list of certs info, user is warned of the # of slots allowed for recovery. For example, if there is only room allowed for two additional keys, then it should be displayed on the client and enforced (e.g. user can only select two out of the list).
4. user selection sent to TPS.
    1. TPS double-checks their "lost" and "recoverable" status and sends "ComputeSessionKeys" to TKS (need extend on existing call)
        1. TKS derives KekSessionKey (KSK) and wrap it with KEK; TKS retrieves DRM-transport key, wrap KSK with it.
        2. TKS sends KEK(KSK) and DRM-transport(KSK) back to TPS
    2. TPS sends "keyRecovery" request (with recovery info and drm-transport(ksk)) to DRM to retrieve keys (new "token key recovery" servlet needs to be written)
        1. DRM retrieves key blob based on recovery info.
        2. DRM unwrap ksk with DRM transport private and wrap key.
        3. DRM sends ksk(user private key) and user public key back to TPS
    3. TPS has retrieved keys injected back to token.
    4. TPS contacts CA to retrieve certs
    5. retrieved certs injected back to token.
5. TPS sends "revocation" request (optional, configurable "revoke at recovery" flag in tdb entry) to DRM for the retrieved certs, once certs revoked, updates token db.


Token Key Recovery
TKS
*gen KekSessionKey
*wrap ksk with kek
*wrap ksk with drm-transport
ComputeSessionKeys
kek(KekSessionKey)
DRM-Transport(KekSessionKey)
op=recovery
keys
certs
TPS
DRM-Transport(KekSessionKey)
DRM
Kek session key(priv key)
* retrieve archived key pair
* archive user private key
* wrap priv w/kek session
certs
Search Cert request
CA
Token DB
CUID:certSerial:CN:uid:expiration:...:"lost":"recoverable
CUID:certSerial:CN:uid:expiration:..."active":...
...

2. temporarily left key at home

The concept of a temporary token is like a temporary badge at work. If you happen to forget your token at home, there should be a mechanism to allow you to have a temporary token that works like your token.

Here is the logic flow:

- user left token at home, reported to security officer and got a replacement blank token
- security officer marks the token as "recoverable" and "validity" (# days is configurable). "revoke at recovery" should be unset.
- user plugs in token, and establishes a new signing cert enrollment, and old encryption key recovery.
- when the "validity" time is up, if the token is inserted afterwards (even without authentication), the data will be zapped.

3. law enforcement

With proper court order, it is possible to override the user authentication.

In this case, the existing (old) DRM functionality of n of m recovery mechanism can be used to recover keys into p12 files.

4. token synchronization (duplicating token/keys) -- feature not in this release

This is not "recovery" per se, however, this feature can tap into the recovery operation. This feature is to allow users to duplicate their keys/certs onto multiple tokens. (not sure if I'll have time to put it into this release, but it's worth mentioning).

Like recovery, this retrieves user's keys/certs (originally on another token) and insert them onto a new token. However, unlike recovery, this should not trigger revocation.