Workshop on Public Key Infrastructure
and
Red Hat Certificate System

Asha Akkiangady, Huzaifa Sidhpurwala, Niranjan Mallapadi

Contents

1   Introduction to Public Key Cryptography

Before we start discussing about public key cryptography, we will in general discuss about how systems communicate and what are the various threat models that are associated with the communication medium and what are the tools to overcome them. [8]

Example1: The most common protocol used to communicate between 2 systems is TCP/IP. TCP/IP allows information to be sent from one system to another system directly or through many intermediate systems. [8]

Below are some of the threat models associated with TCP/IP:

- **Eavesdropping:**

  Information remains intact, but it's privacy is compromised. For example: someone could retrieve the credit card number, record a sensitive conversation or intercept a classified information.

- **Tampering:**

  Information in transit is changed or replaced and then sent to the recipient. For example: Some one could alter an order of goods or change a persons resume.

- **Impersonation:**

  Information passes to a person who poses as intended recipient. Impersonation can take 2 forms:

  - **Spoofing:**

    A person can pretend to be someone else, For example, a person can pretend to have email address *joe@example.net*, or computer can identify itself as a site called www.example.net, when it is not. This type of impersonation is called spoofing.

  - **Misrepresentation:**

    A person or organization can misrepresent itself, For example, suppose the site *www.example.net* pretends to be a furniture store when it's just a site which accepts payments but never sends any goods.

Example2: Consider Alice , Bob and attacker are the parties , where alice and bob want to communicate to each other and Attacker is a threat to the communication. [8]

- Alice can send a postcard to bob to communicate, but this method is very weak as any random eavesdropper could read the postcard.

- Alice could write a letter and put in an envelope and send it to bob, but the attacker could open the envelope and read the letter, both confidentiality and integrity of the message is lost.

- Alice could seal the envelope with wax , but this method too is inefficient as Attacker could read the letter and seal it as it is without bob knowing that it was read by attacker

- Alice could write a letter and put it in a safe which has 2 keys and send one key before to Bob , and send the safe across to bob, this ensures confidentiality, integrity but practically this is not implementable. Considering the number of messages that has to be transferred , it's impractical to implement the mail safe method.

To mitigate above threat models, we will look in to cryptology as one of the tools of the trade.

Cryptology: Cryptology is the theory of designing the various algorithms we use to provide security

Cryptography: Cryptography is the study of using these algorithms to secure systems and protocols.

1.1 Encryption

An encryption algorithm takes some data(called plaintext) and converts it to cipher-text under control of a key. cipher text contains random data which makes no sense without the key.

A key is a short random string (8-24 bytes):

$$\boxed{\text{Plain Text}} \xrightarrow{\text{Encryption}} \boxed{\text{Cipher Text}} \xrightarrow{\text{Decryption}} \boxed{\text{Plain Text}}$$

When a message is encrypted and received , we cannot say if it's not tampered with. An encryption is strong when it can determine the number of possible keys. The attacker tries each key one at a time until he finds a key that produces a plausible decryption. The security of the algorithm should solely depend on the secrecy of the key. The algorithm should not need to be secret.

If the attacker knows the plaintext corresponding to the ciphertext it's called *"known plaintext attack"* .

An attack where attacker doesn't know the plaintext , it's called *"ciphertext-only attack"*. *Ex:* If the attacker knows that plaintext is ASCII , so any decryption which uses non-ascii characters must be using the wrong key.

1.1.1 Symmetric Encryption

When Sender and recipient share the same key(which must be kept secret) is referred to as *Symmetric Key Cryptograph* or also referred to as *Secret Key Cryptography* as opposed to *Public Key Cryptography* [8].
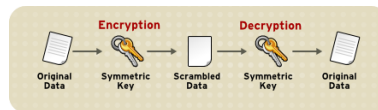
Figure 1: Symmetric Encryption [5]

1.1.2 Assymetric Encryption

- First started in stanford university by Whitfield Diffie and Martin Hellman

- The most commonly used implementation of PKC are based on algorithm based on algorithm patented by RSA data security.

- Each public key is published & private key is kept secret. Data encrypted with public key can be decrypted only with private key.

- In general, to send encrypted data to someone, we encrypt with public key & the person receiving the encrypted data decrypts with private key

- Compared to symmetric key encryption, public key encryption requires more computation & therefore not always appropriate for large amounts of data

- It is possible to use public-key encryption to send a symmetric key, which can then be used to encrypt additional data.

Figure 2: Assymetric Encryption

Figure 3: Digital Signature [2]

- Reverse of the above figure also happens i.e. encrypt with private key and decrypt with public-key. But not useful for sensitive information

- There's a problem with the above method, how would the parties get each other's public-key ? If we send the keys through electronically, then the attacker an tamper, while they are in transit to the receiver.

- When the 2 parties want to communicate, the attacker can intercept the keys & instead send his own key to each other, thus each party encrypts to him and & he re-encrypts it to the real recipient. This is called *man-in-the-middle attack*

1.2 Message Digest

A message digest is simply a function that takes as an input an arbitrary message and outputs a fixed length string which is characteristic of the message. The important property here is irreversibility. It's extremely difficult to compute a message from the given digest. Property of the message digest:

- For a digest to be secure, it must be difficult to generate any of the message that digests to the same value. You have to search a message space of proportional size of the digest in order to find a matching message text

- It should be difficult to produce 2 messages M and M' such that they have the same digest. This property is called collision-resistance. It turns out that the strength of any message digest against finding collision is only half the size of the digest., so a 128-bit digest is only 64 bits strong against collisions.

1.2.1 Message Authentication Code

Consider *Alice* and *Bob* share a key and Alice wants to send a message to *Bob.* The message can be encrypted, and send it across to *Bob*, but we are not sure

that the encrypted message would be tampered and also not sure if *Alice* was
the one who sent the encrypted message. So we use a new tool called *MAC* .
MAC is a digest algorithm, but with a key, So the MAC is dependent on both
the key and the message being MACed [8].

1.3   Algorithms

- **RSA** [8]

    - RSA is the public-key algorithm widely used in Public-key Cryptography

    - Invented by Ron-Rivest, Adi Shamir and Len Adelman (RSA). Each user has a public-key and a private-key

    - The public-key can be freely distributed and the private-key must be kept secret.

    - Brief Explanation:
        * Generate 2 prime numbers (3, 17), call them p and q
        * n = modulus(p * q)
        * p and q are kept secret
        * Difficulty is in factoring m, so that we get p and q
        * Take another number "e" which is called public exponent, which is a prime number , it's usually small prime numers 3, 17 or 65537
        * Compute d = $e^{-1}mod((p-1)(q-1))$
        * Public Key = (e, n)
        * Private Key = (d, n)
        * Message M is encrypted with public key and decrypted with private key
        * RSA assumes that M is a number, So we need a convention to convert strings in to numbers.

- **DSA** [8]

    - National Institute of standards and technology (NIST) published the Digital signature algorithm in the Digital signature standard.

    - Compared to RSA where it can be used for both encryption and digital signature

    - In DSA the signature generation is faster than signature verification

- **RC4**

    - RC4 is a stream cipher.

    - Assume we have a function (f) which produces 1 byte of data. This output is called keystream (ks)

    - The function (f) takes a encryption key as an input to generate keystream

    - Without the key we can't predict keystream

- So combine each byte with one byte of plain text which is our ciphertext
- C[i] denotes the ith unit of ciphertext
- where a unit refers a 1 byte of data.
- ks[i] refers the ith unit of keystream
- M[i] refers to the ith unit of the message.
- C[i] = ks[i] xor M[i]
- m[i] = ks[i] xor C[i]
- Disadvantages:
  * Assume we have used the same key to encrypt the Messages M and M'
  * If the attacker learns M and can compute ks by simply computing M xor C.
  * Once the attacker knows KS he can generate M'
- RC4 was designed by Ron Rivest . RC4 is a variable key length cipher , with key can be anywhere between 8 to 2048 bytes long
- SSL/TLS protocol use RC4 with 128-bit (16 bytes) key length
- RC4 is extremely fast.

- **Block Ciphers** [8]

  - **DES**
    * The data to be encrypted is processed in blocks of bytes (8 or 16).
    * Each possible plain text block corresponds to a row in the table
    * So to encrypt a block you find a column corresponding to the key, run down the table to find the row corresponding to the block you want to encrypt.
    * If the data is huge, it's very difficult to manage, so we define a function that does the computation
    * The idea is to simulate a random table
    * we have a key(k), and a data block . and we define 2 function E for encryption ,and D for decryption
    * C=E(K,M)
    * M=D(k,c)
    * When we have large messages , we do in Electronic code book mode.
      · Break the message up to block sized-chunks.
      · individually encrypt it using encryption algorithm
      · C[i]=E(k,M[i])
      · M[i]=D(K,C[i])
    * DES was desinged by IBM , it's patented but freely available
    * DES is a 64 block cipher with 56-bit key . The data is encrypted in blocks of 8 bytes and a key space of 56 bits.

* DES is used with public-key techniques.
* Disadvantages:
  · if M[i] and M[j] are same, then we get the same C[j] and C[k]. Attacker can get the pattern
  · Cipher block chaining mode, The encryption of each plain text block M[i] depends on cipher text of previous block C[i-1].
  · That can be accomplished by XORing previous Cipher text block C[i-1]] Xor M[i] before encryption

- **3DES** [8]
  * Run the DES algorithm 3 times.
  * It's used in Encrypt-Decrypt-Encrypt (EDE mode).
  * Message is encrypted with key-1, Decrypted with Key-2 and Encrypted with key-3.
  * 3DES is 3 times slower than DES.

- **RC2** [8]
  * RC2 is a block cipher invented by Ron Rivest,
  * RC2 is a variable-length cipher , with variable length key. It uses 64 bit block size.

- **AES** [8]
  * Advanced Encryption Standard uses a minimum of 128 bits and 3 key lengths, 128. 192 and 256 bits

• **Digest Algorithms** [8]

- The two most popular Digest Algorithms are MD5, which was designed by Ron Rivest and SHA-1 by NIST.
- MD5 and SHA share a common ancestor MD4 also designed by Ron Rivest

1.4 Mathematics behind cryptography

For more information on mathematics behind Cryptography please refer to: [11]

1.5 Protocols

1.5.1 SSL

• Secure Socket Layer (SSL) is a protocol that provides a secure channel between machines

• It has facilities for protecting data in transit and identifying machine with which it is begin communicated

• Protocol is transparent , so it can be run on any protocol.

• SSL has gone through many versions and currently is culminating with the adoption by IETF as Transport Layer Security

- SSL was originally designed for world wide web , but also was intended as a unifying solution to all the other communications (web, mail, news traffic)

- The current version of SSL is tlsv2 which fixes a lot of security problems

- **Overview of SSL Protocol**

  - Primary goal of SSL is to provide privacy and reliability between 2 communicating applications
  - The protocol is composed of 2 layers : **Handshake protocol** and **record protocol**
  - In brief Handshake protocol provides
    * allows server & client to authenticate to each other
    * Negotiate encryption algorithm or Cryptography keys.
  - Record protocol provides:
    * Confidentiality
    * Authenticity
    * replay protection

- **Handshake Protocol**

  - Client sends a list of algorithms it's willing to support along with a random number used as input to a key generation proces
  - Server chooses out of that list and sends it back along with a certificate containing servers public-key.
  - certificate also provides the server's identity for authentication purpose and the server supplies a random number which is used as a part of key generation process
  - Client verifies the servers certificate and extracts the server's public-key from the certificate
  - Client then generates a random secret called *pre_master_secret* and encrypts this with server's public-key
  - Client sends this encrypted *pre_master_secret* to the server
  - Client and the Server independently compute the encryption and MAC keys from *the pre_master_secret* and client and server's random values
  - Client sends a MAC of all the handshake messages to the server
  - The server sends a MAC of all the handshake messages to the client
  -
  - Step:1 corresponds to single SSL Handshake message, Client Hello
  - Step:2 corresponds to SSL Handshake messages
    * First is a ServerHello Algorithm preference, Certificate
    * Send ServerHelloDone
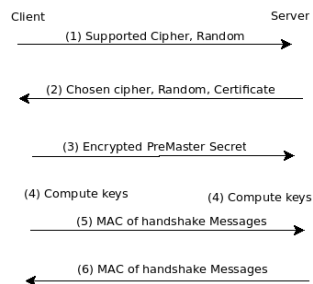 - Step3 correponds to ClientKeyExchange

Figure 4: SSL Handshake

- Step 5 and 6 correspond to the finished message. The Finished message is the first message that's the protected using Just-Negotiated algorithms.
- To protect the handshake from tampering, the content of the message is MAC of all the previous handshake messages
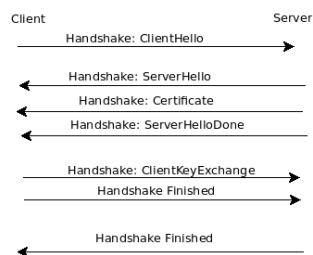

Figure 5: SSL Handshake

-
- **Resuming Session**
  * client sends a clientHello with a Session-ID of the session to be resumed.
  * server check it's session cache for a match
  * If the match is found & the server is willing to re-establish the connection under specified session state , it will send a Server-Hello with same session-ID
  * At this point both client and server must send change Cipher spec messages & proceed directly to finished messages
  * There is no exchange of certificates here.
  * Once the re-establishment is complete the client & server may begin exchange of application data
  * If the Session-ID match is not found the server generates a new session-ID & the client and Server have to perform a full hand-shake

- **SSL Record Protcol**

  - The actual data transfer is accomplished by SSL Record Protocol

- Record Protocol works by breaking up the data stream to be transmitted in to a series of fragments, each of which is independently protected and transmitted.

- On the receiving end , each record is independently decrypted and verified.

- Before transmission , a MAC is computed on each record , This MAC is transferred along with the record

- The concatenated data and MAC are encrypted to form encrypted Payload. We attach a header to that encrypted payload which we refer to as a record
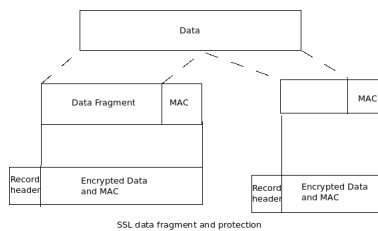
Figure 6: ssl data fragment and protection [8]

-

- The record header provides the information for the other end to interpret the record. It contains:

    * Content Type (Application data, alert messages, handshake message, change cipher spec)
    * length (how many bytes to read off wire)
    * SSL Version

- The *change_cipher_spec* message indicates a change in encryption and authentication of records

- Once the handshake is completed a new set of keys is negotiated, *change_cipher_spec* record is sent to indicate that those keys will now be used.

Figure 7: SSL Overview

2 Introduction to Public Key Infrastructure

Below is a simplified architectural model of Public Key Infrastructure using X.509 (PKIX) Specifications

Figure 8: PKI Architectural Model [9]

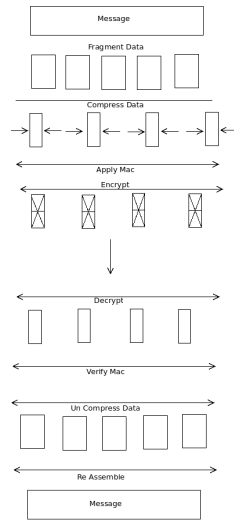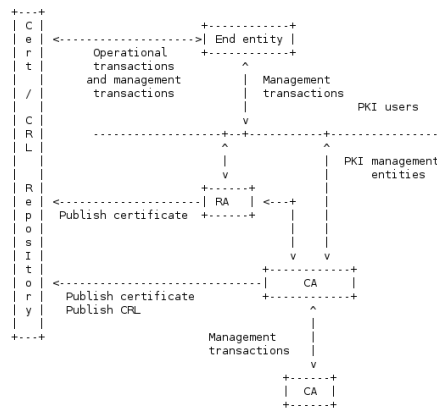## 2.1 Common Terms used in PKI

- End Entity: User of the PKI certificates and/or end user system that is the subject of a certificate

- CA: Certificate Authority

- RA: Registration Authority, Optional System to which CA delegates certain management functions

- CRL issuers: A system that generates CRL

- repository: a system or collection of distributed systems that stores certificates and CRLs and services as a means of distributing these certificates and CRLs to end entities

2.2 Detailed look on certificate/CRL

2.2.1 Certificates

- Certificates are data structures that bind public key values to the subject. This binding is asserted by a trusted Certificate Authority.

- X.509 defines the standard certificate format and v3 is the latest version.

- Internet Privacy Enhanced Mail(PEM) RFC 1421 and 1422 also include specifications for PKI based on X.509 certs.

- Types of Certificates:

  - End-Entity
  - CA

- RFC 1422 defines hierarchial structure of CA's and there are three types of PEM CA

  - IPRA: Internet Policy Registration Authority , acts as Root of Certificate authority. IPRA operates under Internet Society Organization

  - PCA: Policy Certificae Authority ,(Verisign, Digicert, etc) signed by IPRA

  - CA: Certificate Authorities signed by PCA (Organizational CA's)

- Policies used by CA:

  - IPRA certifies only PCA and not CA's or users cert
  - IPRA will make sure that the DN of the PCA is unique and will not issue certificate for PCA's with similar DN
  - Certificates should not be issued to distinct entities under the same distinguished Name.
  - CA's are expected to sign certificates only if the subject DN in the certificates is subordinate to the issuer CA DN.

- Types of Certificate Authorities

  - Cross-Certs: Where Issuer and Subject are different
  - Self-Issue: Where Issuer and Subject are same
    * Self-Signed: Where key bound in to the certificate is same as the key used to sign the certificate

2.2.2 Basic Certificate Fields

- Version:

  - Describes the version of encoded certificate.
  - if extensions are used: **0x2(3)**
  - if extensions are not used but UniqueIdentifier is used: **0x1(2)**
  - if only basic fields are there: **0x0(1)**

  - Values: **0x2(3), 0x1(2), 0x0(1)**

- Serial Number:

  - Positive integer assigned by CA to each certificate
  - It must be unique for each Certificate given by CA
  - Can contain long integers (up to 20 Octets)

  - Values: Integers:
    * **16694152257348400000**
    * **0xe7ad8b07558a1727**
    * **cd:ba:7f:56:f0:df:e4:bc:54:fe:22:ac:b3:72:aa:55**

- Signature:

  - Algorithm used by **CA** to sign the certificate
  - Value:
    * **md2WithRSAEncryption**
    * **sha1WithRSAEncryption**

- Issuer:

  - Identifies the entity that has signed and issued the certificate
  - **MUST** contain non-empty Distinguished Names
  - Names should confirm to X.501 standard
  - Generally contains Country, Organization, Common name, Serial-Number, province, State, title, Surname, Generation Qualifier(Jr, Sr).
  - Values:
    * **C=US, O=VeriSign, Inc., OU=Class 1 Public Primary Certification Authority**
    * **C=DE, ST=Bayern, L=Muenchen, O=Whatever it is, CN=IO::Socket::SSL Demo CA**
    * **C=US, O=VeriSign, Inc., OU=Class 1 Public Primary Certification Authority**

- Validity:

  - The time interval during which the CA warrants that it will maintain status of the certificate

- Consists sequence of 2 dates
    * Date on which certificate validity begins
    * Date on which certificate validity ends
- Validity period of a certificate is period from notBefore to notAfter(inclusive)
- Values:
    * **Not Before: Jan 29 00:00:00 1996 GMT**
    * **Not After : Aug 1 23:59:59 2028 GMT**
    * Special Note:
        · Devices are given certificates where there is no expiration date
        · Value:
        · Certificate is to be used for entire lifetime of the device **Not After: 99991231235959Z.(represented in Generalized Time)**

- Subject:
    - Identifies the entry associated with public key stored in the subject public key field
    - If the subject is CA then it should be populated with same data as issuer
    - Names should confirm to X.501
    - Values:
        * **C=US, ST=North Carolina, O=Fedora Project, OU=Fedora User Cert, CN=mrniranjan/emailAddress=niranjan@ashoo.in**

- Subject Public key Info:
    - This field is used to carry public key and identify the algorithm with which the key is used (RSA, DSA)
    - Supported Cryptographic Algorithms:
        * **Rivest-Shamir-Adelman (RSA)**
        * **Digital Signature Algorithm (DSA)**
        * **Diffie-Hellman (DH)**
        * **Elliptic Curve Digital Signature Algorithm (ECDSA)**
        * **Key Encryption Algorithm (KEA)**
        * **Elliptic Curve Diffie-Hellman (ECDH)**

2.2.3 Extensions

- This field only appears in v3 Certs

- This contains sequence of one or more Certificate Extensions

- Extensions provides method for associating additional attributes with public keys

- Managing relationship between CA's

- Can carry private extensions to carry information unique to their community

- Each Extension is either Critical / Non Critical

- Each Extension includes OID and ASN.1 DER (OCTET)

  - Example: DE:65:01:16:19:2E:51:E0:9A:51:1A:37:50:94:7D:39:29:2A:42:2C

- There cannot be duplicates of Extensions

- Default CRITICAL value is false

- If the Certificate is CA , then they should have below Extensions

  - Basic Key Identifier
  - Authoritative key Identifier
  - Basic Constraints

- **Authority Key Identifier:**

  - Provides a means of identifying the public key corresponding to the private key used to sign the certificate
  - This is required if the issuer has multiple signing keys
  - If the CA certificate is self signed Authority key identifier is skipped
  - Authority key Identifier helps in identifying the issuer certificate.
  - Values:
    * **keyid:48:E6:68:F9:2B:D2:B2:95:D7:47:D8:23:20:10:4F:33:98:90:9F:D4**

- **Subject Key Identifier:**

  - Provides a means of identifying certificates that contain a particular public key
  - This extension is must for CA certificates
  - The value placed in this is same as Authority Key Identifier
  - For End-entity Certificates, this extension provides a means for identifying certificates containing particular key used in application
  - Value:
    * **48:E6:68:F9:2B:D2:B2:95:D7:47:D8:23:20:10:4F:33:98:90:9F:D4**

- **Key Usage:**

  - Defines the purpose of the key contained in the certificate
  - This extension is used to restrict the usage of the key to be used for purpose other than what is defined in Key Usage
    * **digitalSignature**
      · Public key should be used only to verify signatures on objects other than Public key certificates/CRL
    * **nonRepudiation/contentCommitment**

· subject Public key is used to verify digital signatures other than signatures on public key(CRL)

· used to provide nonRepudiation Service that protects against signing entity falsely denying any public action

∗ **keyEncipherment**

· Public key is used for enciphering private or secret keys.

· Is used for encrypting symmetric content-decryption key or an assymetric private key

∗ **dataEncipherment**

· is used for enciphering the raw data without the use any cipher (very rarely used)

∗ **keyAgreement**

· is used for key Agreement,

· when used Diffie-Hellman key is to be used for key management

∗ **keyCertSign**

· Public key is used for verifying the signatures on public keys , if this is true then

∗ **cRLSign**

· when the subject public key is used for verifying signatures of CRL

∗ **encipherOnly**

· subject's public key may be used only for enciphering data while performing key agreement

∗ **decipherOnly**

· subject's public key may be used only for deciphering data while performing key agreement
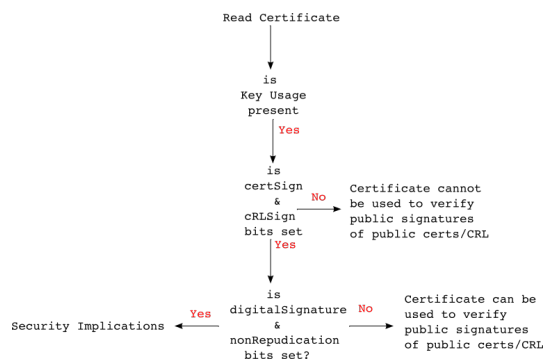
∗ Important Note:

Figure 9: Key Usage restrictions

·

• **Certificate Policies**

   – Contains sequence of one or more policy information terms each of which consits of OID, optional Queries

- OID Should not appear more than once
- for End-entity policy information terms indicate the policy under which the certificate has been issued and purposes for which the certificate must be used
- For CA the policy limits the policies for certificate paths that include this certificate
- if CA does not want to limit the set of policies for certification paths then it may assert **anyPolicy** with value **2.5.29.32.0**
- Policy Qualifiers:
  * CPSNotice: Points URL to *certificate practice statement* document that describes the policy under which the subject was issued
  * userNotice: text that describes the policy(not more than 200 characters)

- **Policy Mappings**
  - This extension is used by CA certificates
  - Lists one or more pairs of OID, which indicate that the corresponding policies of one CA are equivalent to policies of another CA [used in context of Cross pair certificates]
  - **OID: 2.5.29.33**

- **Subject Alternative Name**
  - Allows alternate names to be bound to subject of the certificate.
  - The names specified in subjectAltName may be included in addition to or in place of the identity in the subject field of the certificate
  - Defined subject names used:
    * rfc822Name: IA5String
    * otherName: OtherName
    * dNSName: IA5String
    * directoryName: Name
    * URI: IA5string
    * iPAddress: OCTET String
    * UniformResourceIdentifier: IA5String

- **Issuer Alternative Name**
  - Allows alternate names to be bound to certificate issuer
  - Issuer alternative names are not processed as part of certification path validation
  - Issuer Alternative name extension when present should be non-critical

- **Subject Directory Attributes**
  - This extension is used to convey identification attributes of the subject

- Value:
    * nationality of the subject

- **Basic Constraints**

    - This extension identifies whether the subject of the certificate is CA
    - It also identifies Maximum depth of valid certification paths that include this certificate
    - Values:
        * cA: (boolean): indicates whether certificate is CA cert
        * pathLenConstraint: applicable if cA boolean is true. if prsent asserts keyCertSign bit.
        * It gives maximum number of non-self-issued intermediate Certificates that may follows this certificate in a valid certification path. The last cert is End-entity certificate
        * Ex: PathLength is 2:
            · RootCA - EE Cert
            · RootCA - subca1-EE Cert
            · RootCA - subca1-subca2-EE Cert
            · RootCA - subca1-subca2-subca3-EE Cert

- **Name Constraints**

    - Used only in CA certificate
    - Specifies name space within which all subject names in subsequent certificates ina certification path MUST be located.
    - Restrictions apply to Subject Dishtinguished Name and subject alternative names.
    - Restrictions do not apply for self-issued certs
    - Restrictions are defined in terms of permitted or excluded name subtrees
    - if name matches a restricted excluded subtrees, it 's invalid eventhough the name matches permittedsubtrees.
    - Examples:
        * URI: constraint applies to host part of the name
        * emailAddress: Example .example.org [indicates all emails with domain .example.com]
        * DNS : pki.example.org [www.host1.pki.example.org would satisfy but host1.example.org will not]
        * directoryName: Compares the DN attributes.

- **Policy Constraints**

    - policy constraints extension can be used in certificates issued to CAs.
    - Asserts policy related constraints
        * inhibitPolicyMapping: if present policy mapping is to be inhibited while processing subsequent certificates

* requireExplicitPolicy: if present indicates subsequent certificates need to include an acceptable policy identifier

- **Extended Key Usage**:
  - This extension is used to indicate one or more purposes for which certificates public key can be used:
    * id-kp-serverAuth: TLS WWW server authentication
    * id-kp-clientAuth: TLS WWW client authentication
    * id-kp-codeSigning: Signing of downloadable executable code
    * id-kp-emailProtection: Email protection
    * id-kp-timeStamping: Binding the hash of an object to a time
    * id-kp-OCSPSigning: Signing OCSP responses

- **CRL Distribution Points**
  - cRLDistributionPoints extension is combination:
    * distributionPoint
    * reasons
    * cRLIssuer

- **Authority Information Access**
  - Indicates how to access information and services for the issuer of the certificate in which the extension appears
  - Information and services include, On-line validation services, CA policy data
  - This extension is added in both EE and CA cert

- **Subject Information Access**
  - This extension indicates how to access information and services for the subject of the certificate in which the extension appears
  - If the subject is CA, informaiton and services may include certificate validation and CA policy data.
  - If the subject is EE, information describes the type of services offerred and how to access them

2.2.4   Revocation

When a certificate is issued, it is expected to be used for it's entire validity period. Due to various circumstances, certificate can be invalidated like:

- Change of name

- Change of association with subject and CA (Employee left)

- Compromise or private key

- CA wants to revoke the certificate

- X.509 defines one method of revoking certificates, where CA periodically issues a signed data structure called Certificate revocation list (CRL).

- CRL is a time-stamped list identifying revoked certificates that is signed by CA or CRL issuer.

- This list is freely available through public repositories

- It is expected that the certificate system user not only verifies certificate validity, signature but also aquires latest CRL from public repositories and check against CRL

- CRL's are issued periodically (hourly, daily or weekly).

- CRLissuers or CA issue CRL

- CAs publish CRLs to provide status information about the certificates they issued

- CA may delegate this responsibility to another trusted authority

- Details of CRL

  - Each CRL ahas particular scope
  - CRL scope is the ste of certificates that could appear in a given CRL
  - Examples:
    * all certificates issued by CA x
    * all CA certificates that has been revoked by key compromise or CA compromise
    * set of certificates base on arbitrary local information (all certificate issued to employees at location X)
  - CRL list all **unexpired** certificates, within it's scope that have been revoked for one or other reason
  - If the scope of the CRL includes one or more certificates issued by an entity other the CRL issuer it's called **indirect CRL**
  - CRL issuer may also generate delta CRL.
  - **Delta CRL** only lists those certificates whose revocation status has changed since the issuance of referenced complete CRL.
  - Referenced complete CRL is called **complete CRL**
  - Scope of the delta crl should be same as base CRL that it references
  - When CRL's are issued CRLs must be version 2 CRLs.

2.2.5 CRL Fields

- **Version** All CRLs must be Version 2 CRLs

- **Signature Algorithm**

  - Contains algorithm identifier for the algorithm used by the CRL issuer to sign the certificatesList

- Values:
  * **SHA256withRSA - 1.2.840.113549.1.1.11**

- **Issuer Name**

  – Issuer Name identifies the entity that has signed and issued the CRL

  – Alternative name forms may also appear in the issuerAltName Extension

  – The issuer must contain a non-empty X.500 DN.

- **This Update**

  – This field indicates th issue date of this CRL.

  – Example:
    * **This Update: Sunday, January 17, 2016 8:06:03 AM IST Asia/Kolkata**

- **Next Update**

  – This field indicates the date by which the next CRL will be issued.

  – The next CRL may be issued before the indicated date, but it will not be issued later than indicated date.

  – CRL issuers(CA) **SHOULD** issue CRLs with nextUpdate time equal to or later thean all previous CRLs.

- **Revoked Certificates**

  – Contains list of certificates revoked by CA

  – Certificates revoked by CA are uniquely identified by their certificate serial Number.

  – Date on which revocation occurred is specified.

2.2.6 CRL Extensions

- **Authority Key Identifier**

  – This extension provides a means of identifying the public key correspondign to the private key used to sign the CRL. This identifier can be based on either the key identifier or issuer's name and serial number.

  – Example:
    * Authority Key Identifier Example:

Figure 10: CRL Authority Key identifier

- **Issuer Alternative Name**

  - This extension allows additional identities to be associated with the issuer or CRL.

- **CRL Number**

  - This is a non critical extension that specifies sequence number for a given CRL scope and issuer.
  - This extension allows users to determine if a particular CRL supersedes another CRL.
  - if a CRL issuer generates delta CRLs in addition to complete CRLs for a given scope, the complete CRLs and delta CRLs must share one numbering sequence.

- **Delta CRL Indicator**

  - Delta CRL indicator is a critical CRL extension that identifies a CRL beging a delta CRL.
  - Delta CRLs contain updates to the revocation information previously distributed, rather than all the information that would appear in complete CRL.
  - This helps in reducing network load and processing time in certain environments
  - Example:

Figure 11: Delta CRL Indicator Extension

    *

- **Issuing Distribution Point**

  - This is a critical CRL extension that identifies the CRL distribution point and scope for a particular CRL.
  - It indicates whether CRL covers revocation of EE Certificates only, CA certificates etc.
  - Example:
    *

Figure 12: Issuing Distribution Point extension

2.3 Mozilla NSS

2.3.1 Introduction

- RHEL has 3 sets of security libraries:

  - OpenSSL
  - NSS
  - GnuTLS

- Network Security Services(NSS) is a project by Mozilla which provides crypto API majorly for Mozilla's products like *Firefox, Thunderbird.*

- Mozilla NSS provides all the Public Key cryptography stands which other implementations provide.

2.3.2 Architecture

- With default installation of RHEL, a NSS directory **/etc/pki/nssdb** is created for system wide usage.

- Unlike openssl where certificates are created and stored in text files, certificates created by nss-tools are stored in nss database

- All certificates and keys are stored in berkelely database(or SQLite). There are 3 major files generally seen in a nss directory are:

  - **key3.db** file contains a key used to encrypt and decrypt saved passwords
  - **cert8.db** contains certificates
  - **secmod.db** contains pathnames of pkcs#11 modules

2.4 Exercises

1. Setup: Create PKI Infrastructure using OpenSSL

```
$ mkdir -p ~/pki/ca/{certs,crl,newcerts,private}
cd ~/pki/ca
chmod 700 private
touch index.txt
echo 01 > serial
```

```
wget \
https://raw.githubusercontent.com/mrniranjan/pkiworkshop/master/openssl.cnf
# Edit openssl.cnf and replace the directory mentioned under
# dir parameter with the actual directory where ~pki/ca was
#created
```

2. Create a CA cert using Openssl

```
$ cd ~/pki/ca
# Generate Key Pair
$ openssl genrsa -out private/cakey.pem 2048 -config openssl.cnf

# set permissions
$ chmod 400 private/cakey.pem

#Generate Self signed cert
$ openssl req -config openssl.cnf -key private/cakey.pem \
    -new -x509 -days 365 -extensions v3_ca \
    -out certs/cacert.pem
#view the certificate
$ openssl x509 -in certs/cacert.pem -noout -text
```

```
Output:
You are about to be asked to enter information that will be
    incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
    or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [IN]:
State or Province Name (full name) [Maharashtra]:
Locality Name (eg, city) [Pune]:
Organization Name (eg, company) [Example Org]:
Organizational Unit Name (eg, section) [PKI]:
Common Name (eg, your name or your server's hostname) []:Example
    RootCA
Email Address []:
```

3. Create a End User cert with CN=Pki User1

```
# Create a private key

$ cd ~/pki/ca

$ openssl genrsa -out private/pkiuser1.key 2048

# Create Certificate Request

$ openssl req -new -key private/pkiuser1.key \
```

```
    -out pkiuser1.csr -config openssl.cnf

# Approve the request by signing with CA certificate
$ openssl ca -config openssl.cnf -extensions usr_cert \
    -days 375 -in pkiuser1.csr -out certs/pkiuser1.cert.pem
# view the certificate
$ openssl x509 -in certs/pkiuser1.cert.pem -noout -text
```

4. Create a User cert *cn=Pki User2* with adding extension "Authority Key Identifier" where extension adds the hash of CA public key

   Add the below lines to **usr_cert** section of openssl.cnf

   ```
   subjectKeyIdentifier=hash
   authorityKeyIdentifier=keyid
   ```

   ```
   # Generate Key pair
   $openssl genrsa -out private/pkiuser2.key 2048

   # Generate Certificate Request
   $ openssl req -new -key private/pkiuser2.key -out pkiuser2.csr
       -config openssl.cnf
   -----
   Country Name (2 letter code) [IN]:
   State or Province Name (full name) [Maharashtra]:
   Locality Name (eg, city) [Pune]:
   Organization Name (eg, company) [Example Org]:
   Organizational Unit Name (eg, section) [PKI]:
   Common Name (eg, your name or your server's hostname) []:pkiuser2
   Email Address []:pkiuser2@example.org

   # Sign/Approve the request by signing with CA Cert
   $openssl ca -config openssl.cnf -extensions usr_cert \
       -days 375 -in pkiuser2.csr -out certs/pkiuser2.cert.pem

   # view the certificate
   $ openssl x509 -in certs/pkiuser2.cert.pem -noout -text
   ```

5. Create a user cert by adding followin extensions digitalSignature, nonRepudiation

   Add the below lines to **usr_cert** section of openssl.cnf

   ```
   keyUsage = nonRepudiation, digitalSignature
   ```

   ```
   # Generate Key Pair
   $ openssl genrsa -out private/pkiuser3.key 2048

   # Generate Certificate Request
   $ openssl req -new -key private/pkiuser3.key -out pkiuser3.csr
       -config openssl.cnf
   -----
   ```

```
Country Name (2 letter code) [IN]:
State or Province Name (full name) [Maharashtra]:
Locality Name (eg, city) [Pune]:
Organization Name (eg, company) [Example Org]:
Organizational Unit Name (eg, section) [PKI]:
Common Name (eg, your name or your server's hostname) []:pkiuser3
Email Address []:pkiuser3@example.org

# CA sign the Certificate request
$openssl ca -config openssl.cnf -extensions usr_cert -days 375 -in
    pkiuser3.csr -out certs/pkiuser3.cert.pem
#view the certificate
$ openssl x509 -in certs/pkiuser3.cert.pem -noout -text
```

6. create a server cert by adding following extensions:

   create a new section **server_cert** section of openssl.cnf and add below lines:

   ```
   keyUsage = keyAgreement, digitalSignature
   ```

   ```
   # Generate key pair
   $ openssl genrsa -out private/server1.key 2048

   #Generate certificate request
   $ openssl req -new -key private/server1.key -out server1.csr
       -config openssl.cnf
   -----
   Country Name (2 letter code) [IN]:
   State or Province Name (full name) [Maharashtra]:
   Locality Name (eg, city) [Pune]:
   Organization Name (eg, company) [Example Org]:
   Organizational Unit Name (eg, section) [PKI]:
   Common Name (eg, your name or your server's hostname)
       []:server1.example.org

   # Approve the request by CA signing the request
   $ openssl ca -config openssl.cnf -extensions server_cert -days 375
       -in server1.csr -out certs/server1.cert.pem

   #view the certificate
   $ openssl x509 -in certs/server1.cert.pem -noout -text
   ```

7. Create a user cert with subjectAltname having an email address

   Add the below lines to **usr_cert** section of openssl.cnf

   ```
   subjectAltName = email:copy
   ```

   ```
   # Generate Key Pair
   $ openssl genrsa -out private/pkiuser4.key 2048
   ```

```
# Generate Certificate Request
$ openssl req -new -key private/pkiuser4.key -out pkiuser4.csr
    -config openssl.cnf
-----
Country Name (2 letter code) [IN]:
State or Province Name (full name) [Maharashtra]:
Locality Name (eg, city) [Pune]:
Organization Name (eg, company) [Example Org]:
Organizational Unit Name (eg, section) [PKI]:
Common Name (eg, your name or your server's hostname) []:pkiuser4
Email Address []:pkiuser4@example.org

# CA sign the Certificate request
$openssl ca -config openssl.cnf -extensions usr_cert -days 375 -in
    pkiuser4.csr -out certs/pkiuser4.cert.pem

#view the certificate
$ openssl x509 -in certs/pkiuser4.cert.pem -noout -text
```

8. create a CA cert with pathlength of 2 . Verify path validation by creating multiple CAs

Create a new section **v3_usr_ca** in openssl.cnf and create certs

```
basicConstraints=CA:TRUE
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
basicConstraints = critical,CA:true,pathlen:2
keyUsage = cRLSign, keyCertSign
```

```
# RootCA->SubCA1
# Generate key pair for SubCA1
$ openssl genrsa -out private/subca1.key 2048

# Generate certificate request
$ openssl req -new -key private/subca1.key -out subca1.csr -config
    openssl.cnf
-----
Country Name (2 letter code) [IN]:
State or Province Name (full name) [Maharashtra]:
Locality Name (eg, city) [Pune]:
Organization Name (eg, company) [Example Org]:
Organizational Unit Name (eg, section) [PKI]:
Common Name (eg, your name or your server's hostname) []:Example
    SubCA1
Email Address []:

# Approve the certificate request by RootCA signing the request
$ openssl ca -config openssl.cnf -extensions v3_usr_ca -days 375
    -in subca1.csr -out certs/subca1.cert.pem

# view the certificate
$ openssl x509 -in certs/subca1.cert.pem -noout -text
```

Edit v3_usr_ca section of openssl.cnf and make the pathlength to 1

```
# RootCA->SubCA1->SubCA1.1

#Generate Key pair
$ openssl genrsa -out private/subca1.1.key 2048

# Generate certificate request
$ openssl req -new -key private/subca1.1.key -out subca1.1.csr
    -config openssl.cnf
Country Name (2 letter code) [IN]:
State or Province Name (full name) [Maharashtra]:
Locality Name (eg, city) [Pune]:
Organization Name (eg, company) [Example Org]:
Organizational Unit Name (eg, section) [PKI]:
Common Name (eg, your name or your server's hostname) []:Example
    SubCA1.1
Email Address []:

# Approve the certificate request by subca1 signing the request
$ openssl ca -keyfile private/subca1.key -cert
    certs/subca1.cert.pem \
    -config openssl.cnf -extensions v3_usr_ca -days 375 \
    -in subca1.1.csr -out certs/subca1.1.cert.pem

# View the certificate
$ openssl x509 -in certs/subca1.1.cert.pem -noout -text
```

Edit v3_usr_ca section of openssl.cnf and make the pathlength to 0

```
# RootCA->SubCA1->SubCA1.1->SubCA1.1.1

#Generate Key pair
$ openssl genrsa -out private/subca1.1.1.key 2048

# Generate certificate request
$ openssl req -new -key private/subca1.1.1key -out subca1.1.1.csr
    -config openssl.cnf
Country Name (2 letter code) [IN]:
State or Province Name (full name) [Maharashtra]:
Locality Name (eg, city) [Pune]:
Organization Name (eg, company) [Example Org]:
Organizational Unit Name (eg, section) [PKI]:
Common Name (eg, your name or your server's hostname) []:Example
    SubCA1.1.1
Email Address []:

# Approve the certificate request by subca1.1 signing the request
$ openssl ca -keyfile private/subca1.1.key -cert
    certs/subca1.1.cert.pem \
    -config openssl.cnf -extensions v3_usr_ca -days 375 \
    -in subca1.1.1.csr -out certs/subca1.1.1.cert.pem
```

```
# View the certificate
$ openssl x509 -in certs/subca1.1.1.cert.pem -noout -text
```

There should not be any Subordinate CA of **Example SubCA1.1.1**.
Generate a EE Certificate request and sign using Example SubCA1.1.1

```
# Generate Key pair
$ openssl genrsa -out private/pkiuser4.key

# Generate Certificate request
$ openssl req -new -key private/pkiuser4.key -out pkiuser4.csr
    -config openssl.cnf

# Approve the certificate request by Example subca1.1.1 signing
    the request
$ openssl ca -keyfile private/subca1.1.1.key \
    -cert certs/subca1.1.1.cert.pem -config openssl.cnf \
    -extensions usr_cert -days 375 -in pkiuser4.csr \
    -out certs/pkiuser4.cert.pem

# View the certificate
$ openssl x509 -in certs/pkiuser4.cert.pem -noout -text
```

9. Create a server cert with TLS server and client authentication

   Edit section **server_cert** in openssl.cnf and add below

```
[ server_cert ]
basicConstraints=CA:FALSE
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
keyUsage = digitalSignature, keyAgreement
extendedKeyUsage = serverAuth,clientAuth
```

```
# Generate Key Pair
$ openssl genrsa -out private/server2.key

# Generate certificate request
$ openssl req -new -key private/server2.key -out server2.csr
    -config openssl.cnf
-----
Country Name (2 letter code) [IN]:
State or Province Name (full name) [Maharashtra]:
Locality Name (eg, city) [Pune]:
Organization Name (eg, company) [Example Org]:
Organizational Unit Name (eg, section) [PKI]:
Common Name (eg, your name or your server's hostname)
    []:server2.example.org

# Approve the certificate request by RootCA signing the request
$ openssl ca -config openssl.cnf -extensions server_cert -days 375
    -in server2.csr -out certs/server2.cert.pem
```

```
# View the certificate
$ openssl x509 -in certs/server2.cert.pem -noout -text
```

10. create a server cert with subject CN=server3.example.org having subjectAltName *.example.org

    Add below line to **server_cert** section and create a new section **alt_names**

```
[ server_cert ]
subjectAltName = @alt_names

[alt_names]
DNS.1 = *.example.org
```

```
#Generate key pair
$ openssl genrsa -out private/server3.key

# Generate Certificate request
$ openssl req -new -key private/server3.key -out server3.csr
    -config openssl.cnf
-----
Country Name (2 letter code) [IN]:
State or Province Name (full name) [Maharashtra]:
Locality Name (eg, city) [Pune]:
Organization Name (eg, company) [Example Org]:
Organizational Unit Name (eg, section) [PKI]:
Common Name (eg, your name or your server's hostname)
    []:server3.example.org

# Approve the certificate request by RootCA signing the request
$ openssl ca -config openssl.cnf -extensions server_cert -days 375
    -in server3.csr -out certs/server3.cert.pem

# View the certificate
$ openssl x509 -in certs/server3.cert.pem -noout -text
```

11. Create a Name constraint where permitted DN attribute is "CN=pki*,OU=PKI,O=Example Org,C=US"

    Add below lines to **v3_usr_ca**

```
[ v3_usr_ca ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
basicConstraints = critical,CA:true,pathlen:0
keyUsage = cRLSign, keyCertSign
nameConstraints = permitted;dirName:dir_sect

[dir_sect]
C=IN
O=Example Org
OU=PKI
CN=*
```

```
# Generate key pair
$ openssl genrsa -out private/subca2.key

# Generate certificate request
$ openssl req -new -key private/subca2.key -out subca2.csr -config
    openssl.cnf
-----
Country Name (2 letter code) [IN]:
State or Province Name (full name) [Maharashtra]:
Locality Name (eg, city) [Pune]:
Organization Name (eg, company) [Example Org]:
Organizational Unit Name (eg, section) [PKI]:
Common Name (eg, your name or your server's hostname) []:Example
    SubCA2

# Approve the certificate request using RootCA
$ openssl ca -config openssl.cnf -extensions v3_usr_ca -days 375
    -in subca2.csr -out certs/subca2.cert.pem

# View the certificate
$ openssl x509 -in certs/subca2.cert.pem -noout -text
```

Generate a EE Cert which matches the Name constraint

```
# Generate key pair
$ openssl genrsa -out private/pkiuser5.key

# Generate Certificate requeset
$ openssl req -new -key private/pkiuser5.key -out pkiuser5.csr
    -config openssl.cnf
-----
Country Name (2 letter code) [IN]:
State or Province Name (full name) [Maharashtra]:
Locality Name (eg, city) [Pune]:
Organization Name (eg, company) [Example Org]:
Organizational Unit Name (eg, section) [PKI]:
Common Name (eg, your name or your server's hostname) []:pkiuser5
Email Address []:pkiuser5@example.org

# Approve the certificate request using "Example SubCA2"
$ openssl ca -keyfile private/subca2.key -cert
    certs/subca2.cert.pem \
   -config openssl.cnf -extensions usr_cert \
   -days 375 -in pkiuser5.csr -out certs/pkiuser5.cert.pem
```

Generate a EE Cert which Violates the Name constraint set and Example
SubCA2 will not be able to sign the cert

```
# Generate key pair
$ openssl genrsa -out private/cracker.key

# Generate certificate request
openssl req -new -key private/cracker.key -out cracker.csr -config
```

```
    openssl.cnf
-----
Country Name (2 letter code) [IN]:
State or Province Name (full name) [Maharashtra]:
Locality Name (eg, city) [Pune]:
Organization Name (eg, company) [Example Org]:Cracker Org
Organizational Unit Name (eg, section) [PKI]:HACKING
Common Name (eg, your name or your server's hostname)
    []:server1.cracker.org

# Try approving the certificate request by signing by "Example
    SubCA2"
$ openssl ca -keyfile private/subca2.key \
    -cert certs/subca2.cert.pem -config openssl.cnf \
    -extensions usr_cert -days 375 -in cracker.csr \
    -out certs/cracker.cert.pem
--- output---
Using configuration from openssl.cnf
Check that the request matches the signature
Signature ok
The organizationName field needed to be the same in the
CA certificate (Example Org) and the request (Cracker Org)
```

12. Creating Certificate and key databases using NSS

   - Create the below structure of directories to store CA, Server and user
     certificates
       - **/etc/pki/nssdb/CA_db** (for storing root CA certificates)
       - **/etc/pki/nssdb/server_db** (for storing certificates of servers)
       - **/etc/pki/nssdb/client_db** (for storing certificates of users)
   - create a new certificate database CA_db

```
$ cd /etc/pki/nssdb
$ mkdir CA_db
$ certutil -N -d CA_db
```

   - Create a Self signed Root CA certificate, specify the subject name

```
$ certutil -s -d CA_db -n "Example RootCA" -s "CN=Example
    RootCA,OU=IDMQE,O=Example Org,L=Pune,ST=Maharashtra,C=IN"
    -t "CT,," -x -2
```

   Explanation of the flags:

```
-S specifies create a individual certificate and add it to the
    database
-d specifies the directory of the nss database ( -d .
    specifies current directory)
-n "Name of the certificate"
-t "CT,," specifies that this certificate is a Trusted CA
    which can issue server certficates
```

```
-x use the certutil tool to generate the signature for
   certificate being created or added to the database rather
   than obtaining from separate CA
-2 specifies that add basic constraint extensions, like the
   certificate can be used to sign specific operations like
   SSL, opbject signing, SMIME
```

- Extract the CA certificate from CA's cert database to a file

```
$ certutil -L -d CA_db -n "Example RootCA" -a -o rootca.crt
```

- Display the contents fot the CA's Certificate database

```
$ certutil -L -d CA_db
```

- Creating Certificates for Servers, For servers SSL servers or clients

```
$ mkdir /etc/pki/nssdb/server_db
$ certutil -N -d server_db
```

- Import the new CA certificate in the Server certificate's database and we mark it as trusted CA

```
$ certutil -A -d server_db -n "Example CA" -t "TC,," -a -i
    CA_db/rootca.crt
```

Explanation of the flags:

```
-A Add an existing certificate to a certificate database
-d server_db (directory of certificate database)
-t "TC", specifies that this certificate is a Trusted CA which
   can issue server certficates, client certificates. ",,"
   specifies uu i.e "CT,u,u" , where u specifies certificate
   can be used
-a is ascii format
-i input file
```

- Create the server certificate request, specifying the subject name for the server certificate. Generally CN should be identical to the hostname of the server

```
$ certutil -R -d server_db -s
    "CN=pki1.example.org,OU=IDMQE,O=Example
    Org,L=Pune,ST=Maharashtra,C=IN" -a -o server_db/pki1.req
    -v 12
```

- Sign the Certificate from CA , and issues a new certificate in ascii format

```
$ certutil -C -d CA\_db -c "Example RootCA" -a -i
    server_db/pki1.req -o server_db/pki1.pem -2 -6
```

- Import (Add) the new server certificate to the server's certificate database in the server_db directory with appropriate nickname

```
$ certutil -A -d server_db -n pki1 -a -i server_db\pki1.pem -t
    ",,"
```

- View the certificate

```
$ certutil -V -d server_db -u V -n pki1
```

- By default the certificates that have been created using the above procedure are in pkcs12 standard to use the certificates on Applications, You would require the following
  - RootCA's Certificate (public key)
  - Server Certificate (Certificate and key). The Server's participating in SSL communication should have the certificates and key file
- nss-tools comes with pk12util command , pk12util is a pkcs#12 Tool allowing to share certificates. The tool supports importing certificates and keys from pkcs#12 files in to NSS or export them and also list certificates and keys in such files
- Exporting the Server certificate pki1.example.org in to file pki1.p12

```
$ pk12util -d server_db -o pki1.pk12 -n pki1
```

- The Above pki1.pk12 file has both the certificate and the key file encrypted
- For Applications which require certificate and key file to be in pem format we can use Openssl command to convert the file from p12 to pem format.

```
$ openssl pkcs12 -clcerts -nokeys -in pki1.pk12 -out pki1.pem
$ openssl pkcs12 -nocerts -in pki1.pk12 -out pki1.key.pem
    -nodes
```

3   Common Terms Used in RHCS

- Enrollment The point when the certificate is issued to the User/client is called enrollment

- Renewal When certificate is about to be expired or already expired and has CA requires to renew the certificate

- Dual Keys Conventional hierarchical PKI uses a single key pair one public key and one private key. According to VeriSign, Key Pairs are used for one or more of three basic purposes:

  - Encryption

- Authentication

- non-repudiation

A single key used for multiple purposes violates non-repudiation. It is recommended that you assign two key pairs per person – one key pair for signing messages providing authentication and non-repudiation, and a second key pair for encryption. This allows someone to recover the encryption key and decrypt documents that were encrypted using it, without their gaining the ability to sign documents with that users private key as well (which could lead to forgery and violation of non-repudiation)

- ECC is a cryptographic system that uses elliptic curves to create keys for encrypting data. ECC creates cryptographically-stronger keys with shorter key lengths than RSA, which makes it faster and more efficient to implement. But it is not widely supported as RSA

- CRMF Certificate Request Message Format is a syntax use to convey a request for a certificate to Certificate Authority (CA). This request contains a public key, Proof of Possession(POP), Additional registration informtion combined with pop and public key.[10]

- PKCS10 is a standard which specifies syntax for Certificate Request. A certificate request consists of: [6]

  - Dishtinguished Name

  - Public Key

  - Optional Set of attributes

- FIPS Federal Information Processing Standards (FIPS) are standards to be followed by US Federal Govt for use in system by non-military govt. agencies and contractors.

- FIPS-140 standard specifies the security requirements for a cryptographic module utilized within a security system protecting sensitive information in computer and telecommunication systems. US national Institute of Standards and Technology(NIST) publishes FIPS series of standards for the implementation of Cryptographic modules. Cryptographic Module Validation Program (CMVP) validates cryptographic modules to Federal Information Processing Standard (FIPS) 140-2 and other cryptography based standards.

  FIPS 140-2 is primarily of interest to U.S., Canadian, and UK government agencies which have formal policies requiring use of FIPS 140 validated cryptographic software. [7]

  Products that have received a NIST/CSE validation are listed on the Cryptographic Module Validation List : http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm

- HSM is a physical computing device that safeguards and manages digital keys for strong authentication and provides cryptoprocessing. HSM's are used in PKI environment. HSM's are designed to protect keys(private). A private key in HSM cannot be extracted.

4 Red Hat Certificate System

- RHCS is an Enterprise PKI implementation providing various PKI operations like Enrollment, renewal, revocation

- Provides Certificate Lifecycle Management like CA, Key archival, smartcard management.

- Key Features:

  - Certificate issuance, revocation, and retrieval
  - Certificate Revocation List (CRL) generation and publishing
  - Certificate profiles
  - Encryption key archival and recovery
  - Smartcard lifecycle management

4.1 Overview

RHCS Comprises of 5 subsystems each providing a set of PKI operations:

- Certificate Manager:Also called Certificate Authority is the core of PKI.

  - Issues certs
  - Revokes Certs
  - Publishes CRL
  - Establishes Security Domain of trusted subsystems

- Key Recovery Authority(KRA):It's also called Data Recovery Manager(DRM)

  - where private key is archived for certificates which have been created based on unique pair.
  - KRA archives the key pair and can later be retrieved if the private key of the certificate is lost

- Online Certificate Status Protcol(OCSP):

  - OCSP Verifies certificate is valid or not expired

- Token Processing System(TPS)

  - TPS interacts with smartcard and manages keys and certificates on those tokens. TPS to interact with Smartcard it takes help of Token Key service which creates keys to interact with smartcard securely

- Token key System(TKS)

  - TKS derives keys based on the token CCID, private information,and a defined algorithm. These derived keys are used by the TPS to format tokens and enroll, or process, certificates on the token.

- Enterprise Security Client(ESC)

- ESC is software installed on client systems which have tokens connected to it.
- ESC is used to enroll, format smartcard
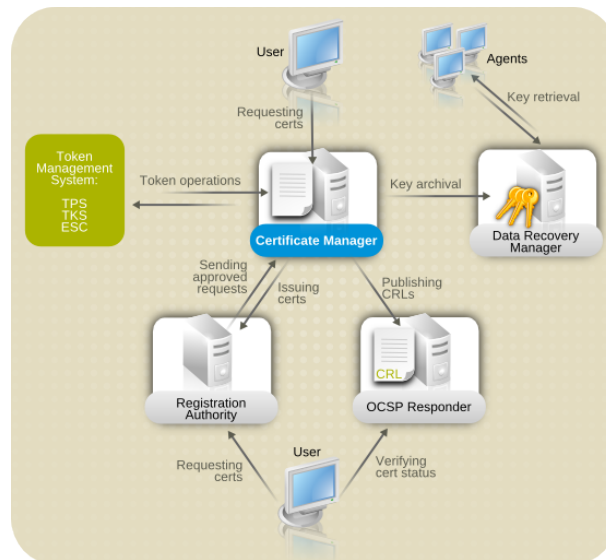- TPS talks to ESC to insert certificates in to smartcards.

Figure 13: RHCS Subsystems [2]

4.2   Version

4.2.1   Older Versions

- Red Hat Certificate System 7.2 (2006)

- Red Hat Certificate System 7.3 (2007)

4.2.2   Current Version

- Red Hat Certificate System 8 (2009)

  - Support Status: Maintenance Support (July 22, 2014 to July 03, 2017

- Red Hat Certificate System 9 (2015)

  - Full support (including hardware updates): September 02, 2015 to September 02, 2018

4.3   Builds

4.3.1   Architecture

- RHCS8 is supported in below architectures:

- x86_64
  - i686

- RHCS9 is supported in x86_64 only

4.3.2 Builds

- RHCS8 Brew Tags:

  - certsys-8-advanced-rhel-5-candidate

- RHCS9 Brew Tags:

  - pki-9-rhel-7-candidate [QE Build]

- RHCS9 Builds

  - pki-core-10.2.6.9.el7pki , which provides:
    * pki-ca
    * pki-kra
    * pki-ocsp
    * pki-tks
    * pki-tps
    * pki-symkey
    * pki-base
    * pki-javadoc
    * pki-tools
    * pki-server
  - pki-console-10.2.6-1.el7pki
  - redhat-pki-10.2.6-1.el7pki
  - redhat-pki-theme-10.2.6-1.el7pki

4.4 Features

Some of Key Features of RHCS9 are:

- RHCS 9 continues to support separate PKI instances for all subsystems

- RHCS 9 introduces the notion of a shared PKI instance:

- *pkispawn* CLI to install and configure new PKI subsystems, support both interactive and batch mode or combination of both.

- *pkidestroy* CLI to remove PKI instance

- RHCS9 services are managed using systemd

  - systemctl [start—status—stop—restart] pki-tomcatd@<instance>.service

- RHCS9 Process management can be done using cli *pki-server* and *pkidaemon*

- Support for REST Web Services APIs

- Web User Interface (UI) Customization

- New Command-Line Interface (CLI) Utilities

4.5 Certificate Manager

4.5.1 Introduction

Certificate Manager is the first subsystem that needs to be configured in PKI Environment, Certificate Manager can be configured as RootCA, Subordinate CA

4.5.2 Installation

- RPM: **pki-ca**

- configuration: CA subsystem is configured using utility **pkispawn**.

  pkispawn provides both interactive configuration or silent configuration by reading a configuration file.

  pkispawn first reads **default.cfg** first and gets other deployment specific information through interactive method or through batch mode by reading a file.

  pkispawn then passes this information to a java servlet which performs the configuration.

```
[root@pki2 ~]# pkispawn

IMPORTANT:

    Interactive installation currently only exists for very basic deployments!

    For example, deployments intent upon using advanced features such as:

        * Cloning,
        * Elliptic Curve Cryptography (ECC),
        * External CA,
        * Hardware Security Module (HSM),
        * Subordinate CA,
        * etc.,

    must provide the necessary override parameters in a separate
    configuration file.

    Run 'man pkispawn' for details.

Subsystem (CA/KRA/OCSP/TKS/TPS) [CA]:

Tomcat:
  Instance [pki-tomcat]:
  HTTP port [8080]:
  Secure HTTP port [8443]:
  AJP port [8009]:
  Management port [8005]:

Administrator:
  Username [caadmin]:
  Password:
  Verify password:
  Import certificate (Yes/No) [N]?
  Export certificate to [/root/.dogtag/pki-tomcat/ca_admin.cert]:
Directory Server:
  Hostname [pki2.example.org]:
  Use a secure LDAPS connection (Yes/No/Quit) [N]?
  LDAP Port [389]:
  Bind DN [cn=Directory Manager]:
  Password:
  Base DN [o=pki-tomcat-CA]:

Security Domain:
  Name [example.org Security Domain]:

Begin installation (Yes/No/Quit)? Yes
```

Figure 14: Configuring CA subsystem using pkispawn

4.5.3 Installation Layout

- **/var/lib/pki/<instanceName>**: ¡Instance¿ here is the tomcat Instance name given during configuring subsystem

- **/var/lib/pki/<instance>/alias**: Is the NSS Database of the system where the subsystem system certs are stored. When a subsystem is configured, Certain certificates are created for certain tasks like audit signing, OCSP Signing, CA Signing cert. These system certificates reside in alias directory. List of system certs are mentioned below, All the below system certs are issued by CA during configuration of CA.

    - caSigningCert (*Used to sign Certs*)
    - Server-Cert (*Certificate of the server having CA*)
    - auditSigningCert (*Certificate used to sign Audit messages*)
    - subsystemCert (*Subsystem Certificate*)

- **/var/lib/pki/<instance>/conf:** Configuration directory containing tomcat configuration, password file containing internal database(ldap) password, and NSS Database password

- **/var/lib/pki/<instance>/ca/:** Contains CA Subsystem configuration file CS.cfg. This file is the main configuration file for CA subsystem

4.5.4 Key Features

- CA subsystem issues, renews, revokes Certificates, generates Certificat Revocation lists

- Publishes Certificates/CRL in form of files or can publish to LDAP or OCSP responder

- CA also has an inbuilt OCSP responder enabling OCSP-Compliant clients to query CA about revocation status of Certificate

- Some CA's can delegate some of it's responsibility to another Subordinate CA

4.5.5 Architecture
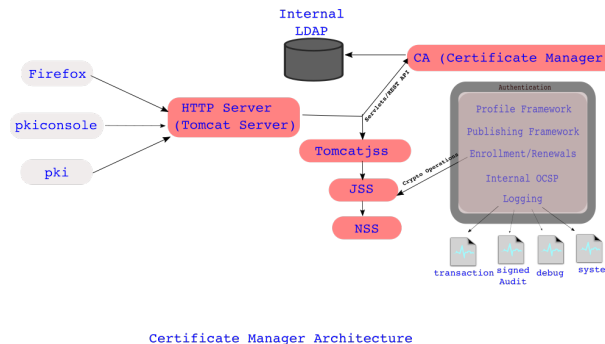

Certificate Manager Architecture

Figure 15: CA Subsystem Architecture

4.5.6 Interfaces

- End User Interface(Browser/CLI)

- Agent Interface(Browser/CLI)

- Admin interface(java console)

4.5.7 Features

- Enrollment:

  End user Enrolls in the PKI infrastructure by submitting a Enrollment(certificate) request through End Entity Interface. This request can be submitted through 2 Methods:

  - Browser
  - CLI

  There can be different kinds of Enrollment(Certificate) Request:

  - Request for User, Server, SMIME, Dual Cert,.. certificate
  - Request certificate if authentication through ldap, pin, cert etc.

  Based on the above types, there are different certificate profiles associated with it. When end-entity(user) enrolls a certificate following events occur:

  - The End-entity provides the information in one of the enrollment forms and submits a request
  - The enrollment forms triggers the creation of public-key and private-key or dual-key pairs
  - The End-entity provides authentication credentials before submitting the request, depending on the authentication type. This can be LDAP authentication, PIN-based authentication or certificate-based authentication.
  - The request is submitted either to an agent-approved enrollment process or an automated process.
    * Agent-approved process requires no end-entity authentication, sends the request to the request queue in the agent-services interface.
    * Automatic notification can be setup so an email can be sent to an agent any time a request appears in the queue
    * The automated process, which involves end-entity authentication,process the certificate as soon as the end-entity successfully authenticates
  - This form collects information about the end entity from the LDAP directory when the form is submitted
  - The profile associated with form determine the aspects of certificate that is issued. Depending upon the certificate profile the request is evaluated to determine if the request meets the constraints set.

– The Certificate request is either rejected because it did not meet the certificate profile or authentication requirement, or a certificate is issued

– The certificate is delivered to end-entity through HTML interface or email or certificate can be retrieved through Agents interface by serial number or request-ID.

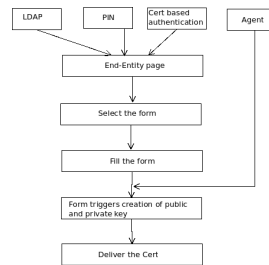– The new certificate is stored in Certificate Managers internal database.

Figure 16: Certificate Enrollment

• Profiles:

Profile determine the content of a certificate. Certificate manager provides customizable framework to apply policies for incoming certificate requests and to control the input requests types and output certificate types

Certificate Profile define the following:

– Authentication Method

– Authorization Method

– Certificate content

– Constraints for the values of content

– Contents of input

– Output
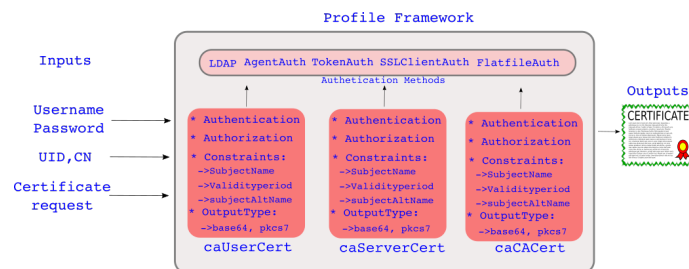
Figure 17: Certificate Profile Architecture

Profile Workflow:

Each profile are defined in **.cfg** file located at **/var/lib/instance_name/profiles/ca** directory
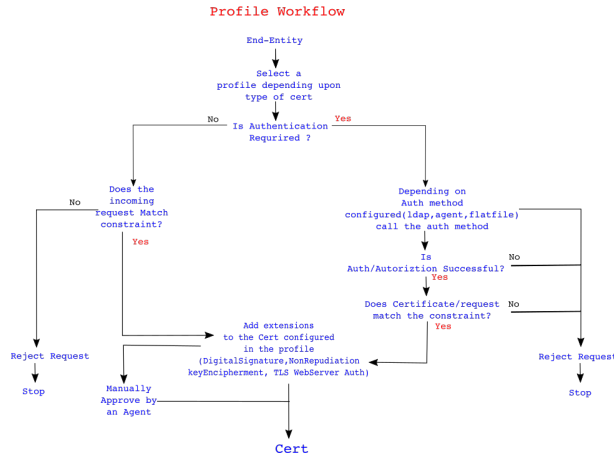
Figure 18: Certificate Profiles Workflow

Example **caUserCert** Profile:

The first part of the profile is the description and specifies whether profile is enabled or disabled, if enabled who enabled it.

Figure 19: Profile Description

The second part of the profile describes the inputs:

Figure 20: Profile inputs

– KeyGenInputImpl: specifies the key pair generation during the request submission, This provides if the request should be of type CRM-F/PKCS10, also provides dropdown specifying the key size.

– subjectNameInputImpl: specifies the subject Distinguished Name(DN) to be used in the cert. The subject DN can be constructed from *UID, Email, Common Name, Organizational Unit, Country*

– submitterInfoImpl: This input specifies three fields: *Requester Name, Requester email, Requester phone*

Third part of the profile is output,

– certOutputImpl: The certificate output format *base64, pkcs7, prettyprint*

44

Figure 21: Profile output

Figure 22: Profile Policies

Last part of the profile is constraints, Policies like:

- validity of the cert
- renewal settings,
- key Usage Extensions
- User supplied extensions

• Publishing Certificate System provides customizable framework from CA's to publish.
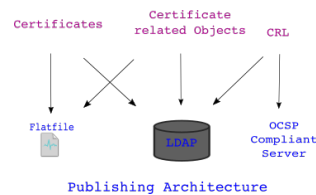

Figure 23: Publishing Architecture

- Key Features
  * Publish to a single repositories or multiple repositories
  * Split locations by certificates/CRL
  * Set individual rules for each type of certs/crl
- Publishing framework consists
  * Publishers
  * Mappers
  * Rules

- Publishers: Publishers specify location to which certificates/CRL's are to be published. Example:
    * To publish to a file, publishers specify the location of the publishing directory.
    * To publish to LDAP, publishers specify the attribute in the directory that srores the cert/CRL.
    * To publish to OCSP, we specify OCSP Server details.
- Rules: Rules define
    * what is to be published and where ?
    * What type of certs can be published to what location
    * Set rules to publish certs to file and LDAP
    * Set individual rules for each type of cert/rule
    * There are rules for Files, LDAP and OCSP
- Mappers: Mappers are only used when publishing to LDAP
    * Mappers construct the DN for an entry based on the information from the certificate or certificate request.
    * Mappers use certificate or certificate request's subject name to construct the DN of the entry to which cert/certificate request/CRL has to be published

4.5.8 Exercises

1. Require 2 RHEL7/Fedora 23 systems , setup hostnames as specified below and make setup hostname resolutions through */etc/hosts*:

```
pki1.example.org
pki2.example.org
```

2. Configure vncserver on both systems

3. Setup CS9 and RHEL7 Optional Repos

```
$ cat /etc/yum.repos.d/rhcs9.repo

[rhcs9]
name=Red Hat Certificate Services 9
baseurl=http://download.eng.bos.redhat.com/rcm-guest/puddles/RHCS/9.0.1/latest/x86_64/os/
enabled=1
gpgcheck=0

$ cat /etc/yum.repos.d/rhel7-optional.repo

[rhel7-optional]
name=Red Hat Enterprise Linux 7.2 Optional
baseurl=http://download.eng.pnq.redhat.com/released/RHEL-7/7.2/Server-optional/x86_64/os/
enabled=1
gpgcheck=0
```

4. Setup Install **redhat-pki** and **389-ds-base** package

```
$ yum install redhat-pki
$ yum install 389-ds-base
```

5. Setup CA Subsystem using pkispawn in interactive mode

   5.1. Setup Directory Server in silent mode by specify config parameters in file **ca-ds.inf**

   ```
    1  [General]
    2  FullMachineName=pki1.example.org
    3  SuiteSpotUserID=nobody
    4  SuiteSpotGroup=nobody
    5
    6  [slapd]
    7  ServerIdentifier=pki-tomcat-RootCA
    8  ServerPort=389
    9  Suffix=dc=example,dc=org
   10  RootDN=cn=Directory Manager
   11  RootDNPwd=Secret123
   12
   13  [admin]
   14  ServerAdminID=admin
   15  ServerAdminPwd=Secret123
   16  SysUser=nobody
   ```

   ```
   $ setup-ds.pl --silent --file=ca-ds.inf --debug
   ```

   5.2. Run **pkispawn** in interactive mode use default ports and specify Directory server information configured above. Use Tomcat Instance name as 'pki-tomcat'

   ```
   Subsystem (CA/KRA/OCSP/TKS/TPS) [CA]:

   Tomcat:
     Instance [pki-tomcat]:
     HTTP port [8080]:
     Secure HTTP port [8443]:
     AJP port [8009]:
     Management port [8005]:

   Administrator:
     Username [caadmin]:
     Password:
     Verify password:
     Import certificate (Yes/No) [N]?
     Export certificate to
          [/root/.dogtag/pki-tomcat/ca_admin.cert]:
   Directory Server:
     Hostname [pki1.example.org]:
     Use a secure LDAPS connection (Yes/No/Quit) [N]?
   ```

```
    LDAP Port [389]:
    Bind DN [cn=Directory Manager]:
    Password:
    Base DN [o=pki-tomcat-CA]:

Security Domain:
    Name [example.org Security Domain]:

Begin installation (Yes/No/Quit)? Yes
```

6. Setup Subordinate CA to the CA configured in 5.2. within the same security domain as RootCA

   6.1. Configure Directory server instance similar to 5.1. but with different port and use instance name as Example1-SubCA

   6.2. Create a config file **subca1.inf** with below information to setup Subordinate CA or RootCA in same security Domain

```
 1  [DEFAULT]
 2  pki_instance_name=Example1-SubCA1
 3  pki_https_port=18443
 4  pki_http_port=18080
 5  pki_admin_password=Secret123
 6  pki_client_database_password=Secret123
 7  pki_client_pkcs12_password=Secret123
 8  pki_ds_password=Secret123
 9  pki_security_domain_password=Secret123
10  pki_security_domain_hostname=pki1.example.org
11  pki_security_domain_https_port=8443
12  pki_security_domain_user=caadmin
13
14  [Tomcat]
15  pki_ajp_port=18009
16  pki_tomcat_server_port=18005
17
18  [CA]
19  pki_subordinate=True
20  pki_issuing_ca=https://pki1.example.org:8443
21  pki_ca_signing_subject_dn=cn=CA Subordinate
        Signing,o=example.org
22  pki_ds_hostname=pki1.example.org
23  pki_ds_ldap_port=1389
24  pki_ds_password=Secret123
25  pki_ds_secure_connection=False
```

   6.3. Run pkispawn with above configuration to configure SubCA.

```
$ pkispawn -s CA -f subca1.inf -vv
```

   6.4. Verify Subsystem status using systemctl and pkidaemon commands

```
$ systemctl status pki-tomcatd@Example1-SubCA1
```

```
$ pkidaemon status tomcat Example1-SubCA1
```

7. Configure External CA signed by RootCA configured in 5.2.

   7.1. Configure Directory server instance similar to 5.1. but with different
        port and use instance name as Example1-SubCA2. Create **ca-ext-
        ds.inf** with below informtion

```
 1  [General]
 2  FullMachineName=pki1.example.org
 3  SuiteSpotUserID=nobody
 4  SuiteSpotGroup=nobody
 5
 6  [slapd]
 7  ServerIdentifier=Example1-ExtCA1
 8  ServerPort=2389
 9  Suffix=dc=example,dc=org
10  RootDN=cn=Directory Manager
11  RootDNPwd=Secret123
12
13  [admin]
14  ServerAdminID=admin
15  ServerAdminPwd=Secret123
16  SysUser=nobody
```

```
$ setup-ds.pl --silent --file=ca-ext-ds.inf --debug
```

   7.2. Setting up External CA is 2 step process. In the first step, a cer-
        tificate signing request (CSR) is generated for the signing certificate.
        Create a config file **ca-ext.inf** with below information:

```
 1  [DEFAULT]
 2  pki_instance_name=Example1-ExtCA1
 3  pki_https_port=30042
 4  pki_http_port=30044
 5  pki_admin_password=Secret123
 6  pki_client_database_password=Secret123
 7  pki_client_pkcs12_password=Secret123
 8  pki_ds_password=Secret123
 9  pki_security_domain_password=Secret123
10
11  [Tomcat]
12  pki_ajp_port=30009
13  pki_tomcat_server_port=30005
14
15  [CA]
16  pki_external=True
17  pki_external_csr_path=/tmp/ca_signing.csr
18  pki_ca_signing_subject_dn=cn=CA Signing,
        ou=External,o=example.org
19  pki_ds_hostname=pki1.example.org
20  pki_ds_ldap_port=2389
```

```
21   pki_ds_password=Secret123
22   pki_ds_secure_connection=False
```

7.3. issue pkispawn command with the above file and this creates a certificate request at location */tmp/ca_signing.csr*

```
$ pkispawn -s CA -f ca-ext.inf -vv
```

7.4. Verify subsystem status using systemctl and pkidaemon commands

```
$ systemctl status pki-tomcatd@Ext1-CA1
$ pkidaemon status tomcat Ext1-CA1
```

7.5. Get the certificate request signed by RootCA configured in 5.2.

7.6. Import CA Admin cert to firefox: [Below procedure is same for any Subsystem Admin Cert]

- Open Firefox
- Click on Preferences (Upper Right hand Corner, check below image):

Figure 24: Firefox Preferences

- Click Preferences
- Select Advanced from left pane
- Select "Certificates" tab
- Click on "View Certificates" to open "Certificate Manager" Window
- From the "Certificate Manager" Windows
- Select "Your Certificates" tab and Click on Import and provide the path of Admin's P12 file
- Should prompt for password, Give the password that is used to protect the P12 file.
- Click on "OK" to close the window
- Close the "preferences" tab

7.7. Approve the request using below procedure

- Access CA EE interface
- Copy the certificate request and submit using the profile: "Manual Certificate Manager Signing Certificate Enrollment"
- From Agent interface approve the request
- Copy the Approved cert and save it to a file /tmp/ca_signing.cert
- Also copy the CA cert chain and save it to file /tmp/ca_cert_chain.cert

7.8. Edit ca-ext.inf as show below

```
 1  [DEFAULT]
 2  pki_instance_name=Example1-ExtCA1
 3  pki_https_port=30042
 4  pki_http_port=30044
 5  pki_admin_password=Secret123
 6  pki_client_database_password=Secret123
 7  pki_client_pkcs12_password=Secret123
 8  pki_ds_password=Secret123
 9  pki_security_domain_password=Secret123
10
11  [Tomcat]
12  pki_ajp_port=30009
13  pki_tomcat_server_port=30005
14
15
16  [CA]
17  pki_external=True
18  pki_external_ca_cert_chain_path=/tmp/ca_cert_chain.cert
19  pki_external_ca_cert_path=/tmp/ca_signing.cert
20  pki_external_step_two=True
21  pki_ca_signing_subject_dn=cn=CA Signing
        Certificate,ou=External,o=example.org
22  pki_ds_hostname=pki1.example.org
23  pki_ds_ldap_port=2389
24  pki_ds_password=Secret123
25  pki_ds_secure_connection=False
```

7.9. Run pkispawn with above modified configuration file

```
$ pkispawn -s CA -f ca-ext.inf -vv
```

7.10. Verify subsystem status using systemctl and pkidaemon commands

```
$ systemctl status pki-tomcatd@Example1-ExtCA1
$ pkidaemon status tomcat Example1-ExtCA1
```

8. Configure Clone of RootCA configured in 5.2.

   8.1. Features of clone
   - A clone of subsystem is a replica of Master subsystem
   - A cloned subsystem has same set of system certs including signing cert
   - Clone acts as a load balancer

   8.2. Create a Backup of RootCA's System Certs in pkcs12 format using PKCS12Export

```
$ PKCS12Export -d /var/lib/pki/pki-tomcat/alias -p
    /tmp/nssdb-pass -w /tmp/pkcs12-pass -o ca_backup_keys.p12
```

   - File /tmp/nssdb-pass should be created with contents of password stored in "Internal" in the file /var/lib/pki-tomcat/conf/password.conf.

51

- File `/tmp/pkcs12-pass` should be craated with the passwod that you want set on the backup file.

8.3. Copy *ca-backup.p12* to the Clone system

8.4. On clone system setup Directory Server using silent mode

ca-clone-ds.inf:

```
 1  [General]
 2  FullMachineName=pki2.example.org
 3  SuiteSpotUserID=nobody
 4  SuiteSpotGroup=nobody
 5
 6  [slapd]
 7  ServerIdentifier=Example1-Clone-CA1
 8  ServerPort=389
 9  Suffix=dc=example,dc=org
10  RootDN=cn=Directory Manager
11  RootDNPwd=Secret123
12
13  [admin]
14  ServerAdminID=admin
15  ServerAdminPwd=Secret123
16  SysUser=nobody
```

8.5. Setup Directory Server using *setup-ds.pl* command

```
$ setup-ds.pl --silent --file=ca-clone-ds.inf --debug
```

8.6. Setup Clone using pkispawn

Note: Clone configuration of RHCS subsystems cannot be done in interactive mode, hence requires an input file containing all configuration parameters to be passed to pkispawn.

ca-clone-inst.inf:

```
 1  [DEFAULT]
 2  pki_instance_name=Example1-CloneCA
 3  pki_https_port=8443
 4  pki_http_port=8080
 5
 6  #Admin
 7  pki_admin_password=Secret123
 8
 9  #client Dir
10  pki_client_database_password=Secret123
11  pki_client_pkcs12_password=Secret123
12  pki_client_dir=/opt/Foobar1-CloneCA
13
14  #Security Domain
15  pki_security_domain_password=Secret123
16  pki_security_domain_hostname=pki1.example.org
17  pki_security_domain_https_port=8443
18  pki_security_domain_user=caadmin
19
```

```
20  [Tomcat]
21  pki_ajp_port=8009
22  pki_tomcat_server_port=8005
23
24  [CA]
25  pki_clone=True
26  pki_clone_pkcs12_password=Secret123
27  pki_clone_pkcs12_path=/tmp/ca_backup_keys.p12
28  pki_clone_replicate_schema=True
29  pki_clone_uri=https://pki1.example.org:8443
30  #Directory Server
31  pki_ds_password=Secret123
32  pki_ds_hostname=pki2.example.org
33  pki_ds_ldap_port=389
34  pki_ds_base_dn=o=pki-tomcat-CA
35  pki_clone_replication_master_port=389
36  pki_clone_replication_clone_port=389
37  pki_ds_secure_connection=False
38  pki_ds_remove_data=True
```

```
$ pkispawn -s CA -f ca-clone-inst.inf -vv
```

9. Configure a Custom profile **caFoobarSanCert** which generates cert with user provided SAN from CSR along with CN from Subject DN

   9.1. Access pkiconsole of RootCA configured in 5.2. using caadmin user

   ```
   $ pkiconsole https://pki1.example.org:8443/ca
   ```

   9.2. From the **Configuration** tab expand **certificate Manager** and then select **certificate profiles**

   9.3. To create a new certificate profile, click **Add**.

   9.4. In the Select Certificate Profile Plugin Implementation window, select **Server Certificate Enrollment Profile**

   9.5. Fill in the profile information in the Certificate Profile Instance Editor.

   - Certificate Profile Instance ID: caFoobarSanCert
   - Certificate Profile Name: Manual Server Certificate with SAN from CSR
   - Certificate Description: This certificate profile is for enrolling server certificate
   - Certificate Profile Authentication: None
     - Note: This sets the authentication method. An automated authentication is set by providing the instance ID for the authentication instance. If this field is blank, the authentication method is agent-approved enrollment; the request is submitted to the request queue of the agent services interface
   - Click OK. The plug-in editor closes, and the new profile is listed in the profiles tab.

53

- Configure the policies, inputs, and outputs for the new profile. Select the new profile from the list, and click **Edit/View**
- Set up policies in the Policies tab of the Certificate Profile Rule Editor window. The Policies tab lists policies that are already set by default for the profile type.
  - To add a policy, click Add.
  - Choose the **User Supplied Extension Default** from the Default field,
  - Select **No Constraint**.
  - In the profile editor
    * Policy Id: p6
    * Under Default tab: userExtOID: 2.5.29.17
    * click Ok
  - Select **Subject Alternative Name Extension Default** and select **No Constraint**
    * Policy Id: p7
    * Under Default tab
      · subjAltNameExtCritical: false
      · subjAltNameNumGNs: 1
      · subjAltExtType_0: DirectoryName
      · subjAltExtPattern_0: $request.req_subject_name.cn$
      · sjbjAltExtGNEnable_0: true
  - Select **Extended Key usage Extension** and select **No Extension**
    * Policy Id: p8
    * Under Default tab
      · exKeyUsageCrtical: false
      · exkeyUsageOIDs: 1.3.6.1.5.5.7.3.2.1,1.3.6.1.5.5.7.3.4
  - Using **openssl** generate CSR for server "www.example.org" with SubjectAltName extensions and specifying alternat host-names "host1.example.org", host2.example.org"
    * Alternatively you could Download CSR from below url

    ```
    https://github.com/mrniranjan/pkiworkshop/blob/master/san.csr
    ```

- Enable the profile from Agent Interface
  - Select Manage Certificate Profiles
  - Select "Manual Server Certificate with SAN from CSR"
  - Click on "Approve"
- From EE page Slect profile "Manual Server Certificate with SAN from CSR" and submit the CSR
- From Agent page Approve the certificate request

10. Use **pki** cli and try different options

10.1. Create a NSS database using pki client cli with password Secret123

```
$ pki -d /opt/rhqa_pki -c Secret123 client-init
```

10.2. Import CA Admin Cert of RootCA(5.2.) to /opt/rhqa_pki

```
$ pki -d /opt/rhqa_pki -c Secret123 client-cert-import
    --pkcs12 /root/.dogtag/pki-tomcat/ca_admin_cert.p12
    --pkcs12-password Secret123
```

10.3. Import CA Cert of RootCA(5.2.)

```
$ pki -d /opt/rhqa_pki1 -c Secret123 -h pki1.example.org -p
    8080 client-cert-import "RootCA" --ca-server
```

10.4. Submit a certificate request for user "foo" with subject DN:

```
UID=foo,E=foo@example.org,CN=Foo User1,OU=IDMQE,O=Example Org
```

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 8080 client-cert-request \
    "UID=foo,E=foo@example.org,CN=Foo User1,O=Example Org"
```

10.5. Approve the Certificate request using Admin Cert

```
$ pki -d /opt/rhqa_pki -c Secret123 -h pki1.example.org\
    -p 8080 -n "PKI Administrator for example.org" \
    cert-request-review 12 --action approve
------------------------------
Approved certificate request 12
------------------------------
  Request ID: 12
  Type: enrollment
  Request Status: complete
  Operation Result: success
  Certificate ID: 0xc
```

10.6. Revoke a certificate with reason "Certificate On Hold"

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 8080 \
    -n "PKI Administrator for example.org" \
    cert-revoke 0xc --reason "Certificate_Hold"
```

10.7. Verify the status of the certificate

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki2.example.org -p 8080 \
    cert-show 0xc
```

11. Remove CA subsystems created above using pkidestroy

```
$ pkidestory -s CA -i Example1-CloneCA
$ pkidestroy -s CA -i Example1-ExtCA1
$ pkidestroy -s CA -i Example1-SubCA1
$ pkidestroy -s CA -i pki-tomcat
```

12. Remove Directory Server Instances

```
$ remove-ds.pl -i slapd-Example1-Clone-CA1
$ remove-ds.pl -i slapd-Example1-ExtCA1
$ remove-ds.pl -i Example1-SubCA
$ remove-ds.pl -i pki-tomcat-RootCA
```

4.6 Key Recovery Authority

4.6.1 Introduction

Key Recovery Authority is an optional subsystem that is configured to archive
private keys. Key Recovery Authority subsystem is by default installed after
CA subsystem is installed and is dependent on CA for certain operations but
can be installed as a standalone subsystem too.

4.6.2 Installation

- RPM: pki-kra

- Configuration: KRA subsystem can be configured through pkispawn using
  interactive or to have a customize

  setup a separate file can be specified with pkispawn.

- KRA subsystem can share the same tomcat instance created by the CA
  subsystem. This allows KRA subsystem to run on the same ports as that
  of CA.

```
[root@pki2 ~]#
[root@pki2 ~]# pkispawn

IMPORTANT:

    Interactive installation currently only exists for very basic deployments!

    For example, deployments intent upon using advanced features such as:

        * Cloning,
        * Elliptic Curve Cryptography (ECC),
        * External CA,
        * Hardware Security Module (HSM),
        * Subordinate CA,
        * etc.,

    must provide the necessary override parameters in a separate
    configuration file.

    Run 'man pkispawn' for details.

Subsystem (CA/KRA/OCSP/TKS/TPS) [CA]: KRA

Tomcat:
  Instance [pki-tomcat]: FoobarKRA
  HTTP port [8080]: 12080
  Secure HTTP port [8443]: 12443
  AJP port [8009]: 12009
  Management port [8005]: 12005

Administrator:
  Username [kraadmin]:
  Password:
  Verify password:
  Import certificate (Yes/No) [Y]? N
  Export certificate to [/root/.dogtag/FoobarKRA/kra_admin.cert]:
Directory Server:
  Hostname [pki2.example.org]:
  Use a secure LDAPS connection (Yes/No/Quit) [N]?
  LDAP Port [389]: 1389
  Bind DN [cn=Directory Manager]:
  Password:
  Base DN [o=FoobarKRA-KRA]:

Security Domain:
  Hostname [pki2.example.org]:
  Secure HTTP port [8443]:
  Name: Foobar Org
  Username [caadmin]:
  Password:

Begin installation (Yes/No/Quit)? Yes
```

Figure 25: pkispawn KRA subsystem

4.6.3 Installation Layout

- **/var/lib/pki/<instance>**: <Instance> here is the tomcat Instance name given during configuring subsystem

- **/var/lib/pki/<instance>/alias**: Is the NSS Database of the system where the subsystem system certs are stored. When a subsystem is configured, Certain certificates are created for certain tasks like Transport Cert for securely transporting private keys, storage cert for storing the private key, These system certificates reside in alias directory. List of system certs are mentioned below, All the below system certs are issued by CA during configuration of KRA.

  - transportCert
  - Server-Cert
  - auditSigningCert
  - storageCert
  - subsystemCert

- **/var/lib/pki/<instance>/conf/**: Configuration directory containing tomcat configuration, password file containing internal database(ldap) password, and NSS Database password

- **/var/lib/pki/<instance>/kra/**: Contains KRA Subsystem configuration file CS.cfg. This file is the main configuration file for KRA subsystem

- **/var/lib/pki/<instance>/lib/**: Contains kra subsystem related jar files

- **/var/lib/pki/<instance>/bin/**: Tomcat related jar files

4.6.4 Key Features

- Key Recovery Authority when configured with with certificate Manager(CA) stores private keys as part of certificate issuance process

- Supports two key recovery modes:

  - Synchronous Recovery(Deprecated)
  - Asynchronous Recovery

- Supports Archiving passwords, symmetric keys, aka acts as a vault to store anything that needs to be stored securely and transported back securely

- Private keys are generally wrapped with a key called storage key and private keys while transporting are wrapped with transport keys, These keys can be rotated
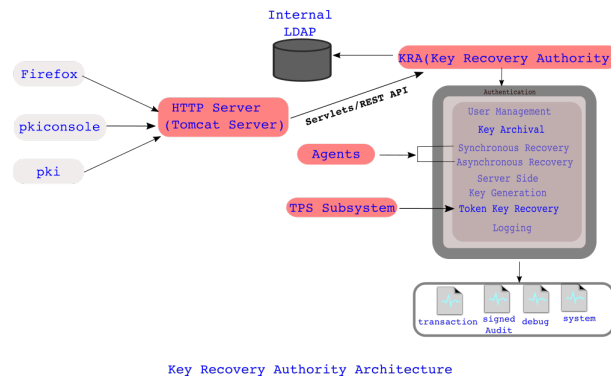
4.6.5 Architecture

Figure 26: KRA Architecture

4.6.6 Interfaces

- End User Interface through **pki** cli

- Agent Interface(Browser/CLI)

- Admin Interface(Java console)

4.6.7 Features

- Key Archival

  - key archival is triggered when a user creates a certificate request through firefox(31.0 or older only)/pki/CRMFPopClient utility
  - The key pair generated as part of the request(public/Private key) is submitted through a defined profile.
  - While submitting the private key is wrapped(encrypted) with transport cert(public key) and submitted to CA
  - CA inturn retrieves the encrypted private key, decripts with transport private key
  - Private key is then stored in interan ldap of kRA by encrypting again with Storage Cert
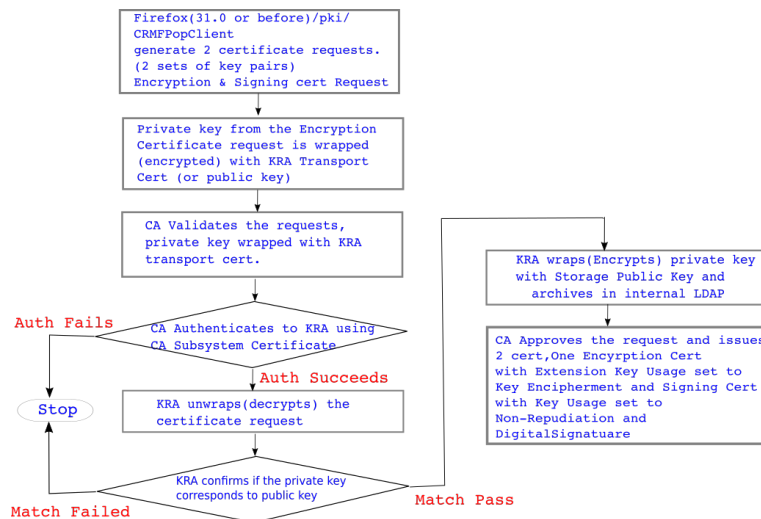 - CA upon successful archival issues certificate
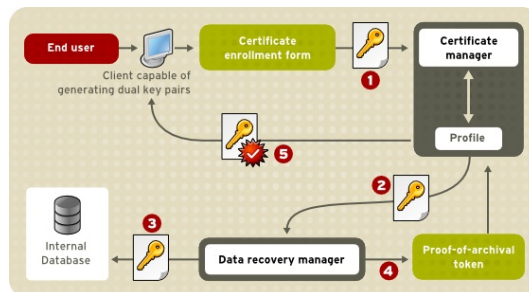

Figure 27: Key Archival Workflow

Figure 28: Key Archival Process

- Key Recovery

- KRA supports Agent initiated key recovery
- Designated Agents use the key recovery form on Agent services page to Approve/Reject key recovery
- CS uses m-of-n ACL-based recovery scheme
- CS uses it's ACL to ensure recovery agents are appropriately authenticated over SSL
- ACL also requires that agents belong to specific recovery agent group
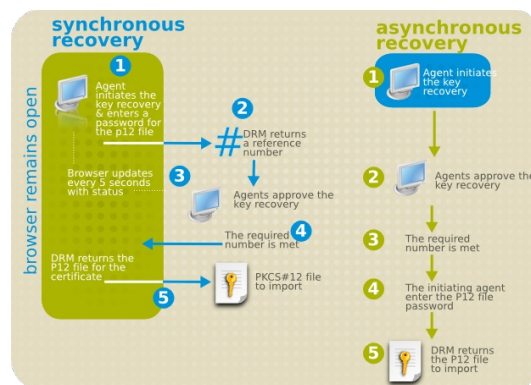- Recovery request is executed only when m of n agents have granted authorization

Figure 29: Key Recovery Process

- Archive Symmetric Keys, passwords, Asymetric keys (Vault)

  - KRA has new REST interface to archive and recover
    * Symmetric Keys
    * Random Security Data
    * Asymmetric Keys
  - This features is implemented through pki key CLI to generate, archive,retrieve keys.

- Transport Key Rotation

  - Transport Certs are created when KRA instance is configured using pkispawn
  - Transport Certs are issued by CA to KRA
  - Transport Certs public key is used to *encrypt* private key while it's being transport by CA to KRA
  - In Large organizations, this transport key needs to be rotated.
  - The feature here adds support for second Transport Key

4.6.8 Exercises

1. Setup required for KRA is same as CA, Refer 1., 2., 3., 4.

2. Setup KRA using pkispawn in separate Tomcat Instance. Use the below configuration with pkispawn

   2.1. Configure CA subsystem using method specified in 5.

   2.2. Setup Directory Server for KRA subsystem using below configuration
       kra-ds.inf:

```
 1  [General]
 2  FullMachineName=pki1.example.org
 3  SuiteSpotUserID=nobody
 4  SuiteSpotGroup=nobody
 5
 6  [slapd]
 7  ServerIdentifier=Example1-RootKRA
 8  ServerPort=1901
 9  Suffix=dc=example,dc=org
10  RootDN=cn=Directory Manager
11  RootDNPwd=Secret123
12
13  [admin]
14  ServerAdminID=admin
15  ServerAdminPwd=Secret123
16  SysUser=nobody
```

```
$ setup-ds.pl --silent --file=kra-ds.inf --debug
```

   2.3. Setup KRA using pkispawn in separate Tomcat Instance. Use the below configuration with pkispawn
       kra-inst.inf:

```
 1  [DEFAULT]
 2  pki_instance_name=Example1-RootKRA
 3  pki_https_port=14443
 4  pki_http_port=14080
 5
 6  #NSS DB Token Password
 7  pki_token_password=Secret123
 8
 9  #RootKRA Admin password
10  pki_admin_password=Secret123
11
12  #Security Domain
13  pki_hostname=pki2.example.org
14  pki_security_domain_hostname=pki1.example.org
15  pki_security_domain_https_port=8443
16  pki_security_domain_user=caadmin
17  pki_security_domain_password=Secret123
18
19  #Client Dir
```

```
20  pki_client_dir=/opt/Example1-RootKRA
21  pki_client_pkcs12_password=Secret123
22  pki_client_database_password=Secret123
23
24  #Backup
25  pki_backup_keys=True
26  pki_backup_password=Secret123
27
28
29  [Tomcat]
30  pki_ajp_port=14009
31  pki_tomcat_server_port=14005
32
33  [KRA]
34  pki_admin_nickname=PKI KRA Administrator for Example Org
35  pki_import_admin_cert=False
36  pki_ds_hostname=pki1.example.org
37  pki_ds_ldap_port=1901
38  pki_ds_bind_dn=cn=Directory Manager
39  pki_ds_password=Secret123
```

```
$ pkispawn -s KRA -f kra-inst.inf -vv
```

3. Enroll a certificate for user Foo1 and archive Private Key in KRA instance

   3.1. Create a NSS Database and Import CA Admin cert , CA cert. Refer:10.1., 10.2., 10.3.

   3.2. Import KRA Admin cert from /opt/Example1-RootKRA

   ```
   $ pki -d /opt/rhqa_pki -c Secret123 \
       client-cert-import --pkcs12 \
       /opt/Example1-RootKRA/kra_admin_cert.p12 \
       --pkcs12-password Secret123
   ```

   ```
   Certificate Nickname                             Trust
       Attributes

                                                    SSL,S/MIME,JAR/XPI

   PKI Administrator for example.org                u,u,u
   Root CA                                          CT,c,
   PKI KRA Administrator for Example Org            u,u,u
   ```

   3.3. Get the transportCert of the KRA instance

   ```
   $ certutil -L -d /var/lib/pki/Example1-RootKRA/alias/ \
       -n "transportCert cert-Example1-RootKRA KRA" \
       -a -o /opt/rhqa_pki/transportCert.pem
   ```

   3.4. Create a certificate request for user Foo1 to a profile in CA which archives private key

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 8080 client-cert-request \
    "uid=foo1,O=Example Org" --profile caDualCert \
    --type crmf --transport /opt/rhqa_pki/transportCert.pem
```

3.5. Approve The certificate request using CA Admin cert

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 8080 \
    -n "PKI Administrator for example.org" \
    cert-request-review <request_id> --action approve
```

3.6. Check Audit logs of KRA instance and see if the Archival request was successful

3.7. Use kra key cli to list the keys archived

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 14080 \
    -n "PKI KRA Administrator for Example Org" kra-key-find
```

4. Archive Symmetric keys and passwords in Key Recovery Authority

   4.1. Setup: Create few role users in KRA subsystem and archive using different users instead of admin

      4.1.1. List all the Groups available for KRA Subsystem

```
$ pki -d /opt/rhqa_pki -c Secret123 -h pki1.example.org \
    -p 14080 -n "PKI KRA Administrator for Example Org" \
    kra-group-find
```

      4.1.2. Create a User "KRA_AgentV" and add him as member of **Data Recovery Manager Agents**

```
$ pki -d /opt/rhqa_pki -c Secret123 \
-h pki1.example.org -p 14080 \
-n "PKI KRA Administrator for Exmaple Org" \
kra-user-add KRA_AgentV --fullName "KRA Agent User" \
--password Secret123
----------------------
Added user "KRA_AgentV"
----------------------
  User ID: KRA_AgentV
  Full name: KRA Agent User
```

      4.1.3. Create a Cert request for "KRA Agent User" using CA Subsystem

```
#This command returns a request id , use the request id to
    approve the request
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 8080 client-cert-request \
```

```
       "UID=KRA_AgentV,E=KRA_AgentV@example.org,CN=KRA Agent
            User,O=Example Org"
```

4.1.4. Approve the certificate request using CA Admin User

```
#This command after approving the request returns
     certificate serial Number
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 8080 \
    -n "PKI Administrator for example.org" \
    cert-request-review <request_id> --action approve
------------------------------
Approved certificate request 29
------------------------------
 Request ID: 29
 Type: enrollment
 Request Status: complete
 Operation Result: success
 Certificate ID: 0x1d
```

4.1.5. Add the certificate created above to KRA Agent User

```
#This will prompt for the CA URL, specify CA's Non secure
     url(8080)
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 14080 \
    -n "PKI KRA Administrator for Example Org" \
    kra-user-cert-add KRA_AgentV --serial \
    <certificate_serial_Number>
```

4.1.6. Make the KRA Agent user member of **"Data Recovery Manager Agents"**

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 14080 \
    -n "PKI KRA Administrator for Example Org" \
    kra-group-member-add "Data Recovery Manager Agents" \
    KRA_AgentV
```

4.1.7. Import the KRA Agent Cert to NSS DB /opt/rhqa_pki Directory

```
$ pki -d /opt/rhqa_pki -c Secret123\
    -h pki1.example.org -p 8080 \
    client-cert-import "KRA_AgentV" \
    --serial \
    <certificate_serial_Number_of Agent_Cert>
```

4.2. Archive a passphrase(password) using Agent User

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 14080 \
    -n "KRA_AgentV" kra-key-archive \
    --clientKeyID beaker_server1 \
```

```
    --passphrase "MySecretSauce"
----------------------
Archival request details
----------------------
  Request ID: 0x2
  Key ID: 0x2
  Type: securityDataEnrollment
  Status: complete
```

4.3. Generate Symmetric keys with AES algo of size 128 and archive it

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 14080 \
    -n "KRA_AgentV" kra-key-generate \
    --key-algorithm AES --key-size 128 \
    --usages wrap
```

5. Retrieve archived symmetric/Asymmetric keys

5.1. Retrieve Archived Symmetric key using **key-retreive**

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 14080 \
    -n "KRA_AgentV" kra-key-retrieve --keyID <KEY_ID>
```

5.2. Retrieve Archived Passwords , Refer 4.2.

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 14080 \
    -n "KRA_AgentV" kra-key-retrieve --keyID <KEY_ID>
```

5.3. Retrieve Asymmetric keys (private Key). Refer Section 3.

```
$ pki -d /opt/rhqa_pki -c Secret123 \
    -h pki1.example.org -p 14080 \
    -n "KRA_AgentV" kra-key-retrieve --keyID <KEY_ID>
```

5.4. Retrieving Asymmetric Keys From Browser(firefox)
   - Import CA Admin Cert and KRA Admin p12 file to Firefox browser. Refer: 7.6.
   - Trust the CA cert
   - Access KRA Agent Page *https://pki1.example.org:14443*
   - Authenticate it with KRA Agent Cert
   - Click "Recover Keys"
   - From the Available Recovery Options select "Certificate"
   - Paste the Certificate foo1 user archived, use CA EE interface to get the certificate
   - Click on Show key
   - From the Search Results , Click on "Recover"

- In the Next screen , Select "Async Recovery", Click on "Recover"
- This should return a Key Recovery Status with reference Number
- Click on The request Number and "Grant" the request
- click on the request id shown on the page
- Specify the PKCS12 Password and click on Retrieve PKCS12

6. Transport Key Rotation

   6.1. Setup CA clone , Refer to steps described in Section: 8.

   6.2. Setup KRA Clone

       6.2.1. While setting up KRA instance (Refer Section: 2.3., we had backed the system keys using parameter `pki_backup_keys=True`. This is equivalent of PKCS12Export ran in section 8.2.

       6.2.2. The backup of system keys of KRA is saved in alias Directory of KRA which is `/var/lib/pki/Example1-RootKRA/alias/`

```
# ls -l /var/lib/pki/Example1-RootKRA/alias/
total 104
-rw-------. 1 pkiuser pkiuser 65536 Jan 31 17:17 cert8.db
-rw-------. 1 pkiuser pkiuser 28672 Feb 1 17:08 key3.db
-rw-r--r--. 1 pkiuser pkiuser 11327 Jan 31 17:17
    kra_backup_keys.p12
-rw-------. 1 pkiuser pkiuser 16384 Jan 31 17:16 secmod.db
```

       6.2.3. Copy the kra_backup_keys.p12 to clone system `pki2.example.org`

       6.2.4. Setup Directory server on clone system for KRA kra-clone-ds.inf

```
1   [General]
2   FullMachineName=pki2.example.org
3   SuiteSpotUserID=nobody
4   SuiteSpotGroup=nobody
5
6   [slapd]
7   ServerIdentifier=Foobar1-CloneKRA
8   ServerPort=1902
9   Suffix=dc=example,dc=org
10  RootDN=cn=Directory Manager
11  RootDNPwd=Secret123
12
13  [admin]
14  ServerAdminID=admin
15  ServerAdminPwd=Secret123
16  SysUser=nobody
```

```
$ setup-ds.pl --silent --file=kra-clone-ds.inf
```

       6.2.5. Setup Clone KRA using below configuration kra-clone.inf :

```
 1  [DEFAULT]
 2  pki_instance_name=Example1-CloneKRA
 3  pki_https_port=14443
 4  pki_http_port=14080
 5
 6  # Token password
 7  pki_token_password=Secret123
 8
 9  #Security Domain
10  pki_security_domain_hostname=pki1.example.org
11  pki_security_domain_https_port=8443
12  pki_security_domain_user=caadmin
13  pki_security_domain_password=Secret123
14
15  # Admin Password
16  pki_admin_password=Secret123
17
18  # client Dir
19  pki_client_pkcs12_password=Secret123
20  pki_client_database_password=Secret123
21  pki_client_dir=/opt/Example1-CloneKRA
22
23
24  [Tomcat]
25  pki_ajp_port=14009
26  pki_tomcat_server_port=14005
27
28  [KRA]
29  pki_clone=True
30  pki_clone_pkcs12_password=Secret123
31  pki_clone_pkcs12_path=/tmp/kra_backup_keys.p12
32  pki_clone_replication_master_port=1901
33  pki_clone_replication_clone_port=1902
34  pki_clone_repicate_schema=True
35  pki_clone_uri=https://pki1.example.org:14443
36  pki_import_admin_cert=False
37  pki_ds_hostname=pki3.example.org
38  pki_ds_ldap_port=1902
39  pki_ds_bind_dn=cn=Directory Manager
40  pki_ds_password=Secret123
41  pki_ds_secure_connection=False
42  pki_ds_remove_data=True
43  pki_ds_base_dn=o=Example1-RootKRA-KRA
```

```
$ pkispawn -s KRA -f kra-clone.inf -vv
```

6.2.6. Request a new DRM Transport Cert

6.2.6.1. Stop the Master KRA instance

```
$ systemctl stop pki-tomcatd@Example1-RootKRA
```

6.2.6.2. Go to KRA's NSS DB directory and take the backup of ex-

isting NSS DB in a subdirectory called backup

```
$ cd /var/lib/pki/Example1-RootKRA/kra/alias
$ mkdir backup
$ cp -a *.db backup
```

6.2.6.3. Create a new pkcs10 request

```
$ PKCS10Client -p <password> -d '.' -o 'req.txt' -n
    'CN=DRM Transport 2 Certificate,O=Example Org'
```

Note: password of NSS Database can be found in password.conf in conf directory of KRA instance

6.2.6.4. Submit the certificate request to CA using EE interface and approve the request using Agent inteface
  • Open the CA EE Page and select profile "Manual Data recovery Manager transport certificate Enrollment"
  • paste the certificate request created in step 6.2.6.3. and submit the request
  • From CA Agent page approve the request

6.2.6.5. Get the approved cert and save the base64 format in a file new-drm-transport.pem

6.2.6.6. Import the DRM Certificate in NSSDB and Update DRM Configuration

```
$ certutil -d /var/lib/pki/Example1-RootKRA/alias \
    -A -n "transportCert-new Cert-Example1-Rootkra
        KRA" \
    -t u,u,u -i new-drm-transport.pem
```

6.2.6.7. Add the new transport certs nick name in KRA's CS.cfg

```
kra.transportUnit.newNickName=transportCert-new
    Cert-Example1-Rootkra KRA
```

6.2.6.8. Update the CA's CS.cfg with new transport Certificate

```
$cp new-drm-transport.pem new-drm-transport-backup.pem

$ unix2dos new-drm-transport-backup.pem
# remove cetificate headers
$ cat new-drm-transport.pem | perl -p -i -e 's/\r\n//'
```

Copy the output of above command and edit CA CS.cfg and replace the exisiting transport Cert

```
ca.connector.KRA.transportCert=<new transport cert> [
    base64 cert which has been trimmed in to one line]
```

6.2.6.9. Restart CA and KRA instances

6.2.6.10. Perform Archival Operation as specified in section: 3. with new transport cert

4.7 Online Certificate Status Protocol

4.7.1 Introduction

- Online Certificate Status Manager is an subsystem external to CA.

- This subsystem acts as an external OCSP which can be publicly accessible.

- Multiple CA's can publish CRL to a single OCSP

- Clients can verify certs using OCSP

4.7.2 Installation

- RPM: **pki-ocsp** OCSP subsystem can be configured through pkispawn using interactive or to have a customize setup a separate file can be specified with pkispawn

  OCSP subsystem can share the same tomcat instance created by the CA subsystem which allows OCSP to use same ports as that of CA.

Figure 30: pkispawn OCSP subsystem

4.7.3 Instance Layout

- **/var/lib/pki/<instance>**: <Instance> here is the tomcat Instance name given during configuring subsystem

- **/var/lib/pki/<instance>/alias**: Is the NSS Database of the system where the subsystem system certs are stored.

    - Server-Cert
    - auditSigningCert
    - ocspSigningCert
    - subsystemCert

- **/var/lib/pki/<instance>/conf/**: Configuration directory containing tomcat configuration, password file containing internal database(ldap) password, and NSS Database password

- **/var/lib/pki/<instance>/OCSP/**: Contains KRA Subsystem configuration file CS.cfg. This file is the main configuration file for KRA subsystem

- **/var/lib/pki/<instance>/ocsp/**: Contains kra subsystem related jar files

4.7.4 Interfaces

- End User Interface through **pki** cli

- Agent Interface(Browser/CLI)

- Admin Interface(Java console)

4.7.5 Exercises

1. Setup OCSP using pkispawn in separate Tomcat Instance. Use the below configuration with pkispawn

    1.1. Configure CA subsystem using method specified in 5.

    1.2. Setup Directory Server for OCSP subsystem using below configuration
    ocsp-ds.inf:

```
1   [General]
2   FullMachineName=pki1.example.org
3   SuiteSpotUserID=nobody
4   SuiteSpotGroup=nobody
5
6   [slapd]
7   ServerIdentifier=Example1-OCSP
8   ServerPort=1902
9   Suffix=dc=example,dc=org
10  RootDN=cn=Directory Manager
11  RootDNPwd=Secret123
```

```
12
13  [admin]
14  ServerAdminID=admin
15  ServerAdminPwd=Secret123
16  SysUser=nobody
```

```
$ setup-ds.pl --silent --file=ocsp-ds.inf --debug
```

1.3. Setup OCSP using pkispawn in separate Tomcat Instance. Use the below configuration with pkispawn

ocsp-inst.inf:

```
 1  [DEFAULT]
 2  pki_instance_name=Example1-RootOCSP
 3  pki_https_port=18443
 4  pki_http_port=18080
 5
 6  #NSS DB Token Password
 7  pki_token_password=Secret123
 8
 9  #OCSP Admin password
10  pki_admin_password=Secret123
11
12  # Security Domain
13  pki_hostname=pki2.example.org
14  pki_security_domain_hostname=pki1.example.org
15  pki_security_domain_https_port=8443
16  pki_security_domain_user=caadmin
17  pki_security_domain_password=Secret123
18
19  #Client Dir
20  pki_client_dir=/opt/Example1-RootOCSP
21  pki_client_pkcs12_password=Secret123
22  pki_client_database_password=Secret123
23
24  # backup
25  pki_backup_keys=True
26  pki_backup_password=Secret123
27
28  [Tomcat]
29  pki_ajp_port=18009
30  pki_tomcat_server_port=18005
31
32  [OCSP]
33  pki_admin_nickname=PKI OCSP Administrator for Example Org
34  pki_import_admin_cert=False
35  pki_ds_hostname=pki1.example.org
36  pki_ds_ldap_port=1902
37  pki_ds_bind_dn=cn=Directory Manager
38  pki_ds_password=Secret123
```

2. Configure Publishing to OCSP with Client Authentication

2.1. Configure publisher on the CA

```
$ pkiconsole https://pki1.example.org:8443/ca
```

2.2. In the Configuration tab, select Certificate Manager from the navigation tree on the left. Select Publishing, and then Publishers.

2.3. Click Add to open the Select Publisher Plug-in Im plem entation window, which lists registered publisher modules

2.4. Select the OCSPPublisher module, then open the editor window. Make sure that the enableClientAuth checkbox is selected.

- Publisher ID: OCSPPublisher1
- host: pki1.example.org
- port: 18443
- path: /ocsp/agent/ocsp/addCRL
- nickName: subsystemCert cert-pki-tomcat

2.5. From the Configuration tab, select Certificate Manager from the navigation tree on the left. Select Publishing, and then Rules.

2.6. Click on Add:

- Rule ID: ocspRule1
- type: crl
- enabled: click on the checkbox
- mapper: NoMap
- publisher: OCSPPublisher1

2.7. Enroll Multiple certificate requests using **pki client-cert-request cli**

```
$ pki -d /opt/rhqa_pki -c 'Secret123' \
-h pki1.example.org -p 8080 \
client-cert-request \
"UID=test1,E=test1@example.org,CN=Test1 User,O=Example Org"
```

2.8. Approve the Certificate request using Agent Cert. Repeat the below command for all the requests created above.

```
$ pki -d -d /opt/rhqa_pki -c 'Secret123' \
    -h pki1.example.org -p 8080 \
    -n "PKI Administrator for example.org" \
    cert-request-review <request_id> --action approve
```

2.9. Revoke the Certs using **pki cert-revoke cli**

```
$ pki -d -d /opt/rhqa_pki -c 'Secret123' \
    -h pki1.example.org -p 8080 \
    -n "PKI Administrator for example.org" \
    cert-revoke <serial_Number> --reason "Key_Compromise"
```

4.8 SmartCard Management

4.8.1 Introduction

- **What:** Smartcard contains an embedded microprocessor. This micropro-cess runs Java Virtual Machine(JVM) and provides environment to run applications on top of it.

  - Smartcard contains a chip with java on it.
  - it contains a JVM, Java Runtime Environment, and some API for applications to access the card.

- **Why:**

  - Smartcards are typically used by enterprise to provide identification and authentication
  - Smartcards provide 2-Factor Authentication which is
    * Something you have - smartcard
    * Something you know - pin(password) to access the smartcard

- **Use cases:**

  - System Authentication using Smartcard (Linux/Windows System Authentication)
  - Secure VPN Access
  - Email Signature and Encryption
  - Disc Encryption

- **How:**

  - For enterprise to use smartcard a digital Certificate is issued to the smartcard
  - This certificate is used to identify the user of the smartcard
  - User uses digital certificate on the smartcard to perform 2-factor authentication

- **Mangement:**

  - We need a management software to manage certificates on the smart-card
    * Enrolling certificate
    * Formatting Smartcard
    * What to do with the certificate if smartcard is lost
    * What to do if the user of the smartcard is no longer in the company
  - To do the above operations, we have Token Management system in Certificate Services which comprises of the following:
    * **TPS**: The Token Processing System (TPS) interacts with smart cards to help them generate and store keys and certificates for a specific entity, such as a user or device.

73

* **TKS**: The Token Key Service (TKS) generates, or derives, symmetric keys used for communication between the TPS and smart card.
* **CA**: The Certificate Authority (CA) creates and revokes user certificates stored on the smart card.
* **KRA**: Optionally, the Key Recovery Authority (KRA) archives and recovers keys for the smart card.
* **Enterprise Security Client**: is the conduit through which TPS communicates with each token over a secure HTTP channel (HTTPS), and, through the TPS, with the Certificate System.

4.8.2 Architecture
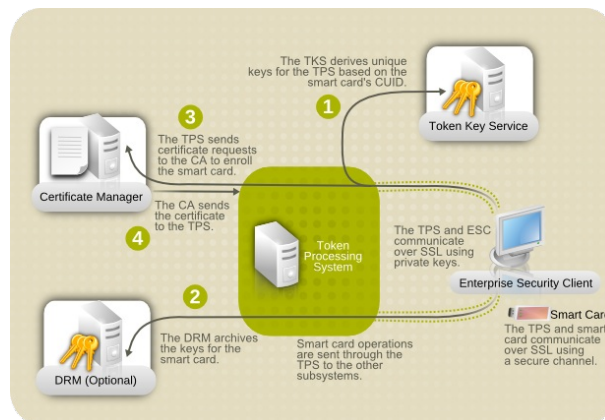
Figure 31: Token Management System

4.8.3 Common Terms Used with regard to Smartcard

- Secure Channel Protocol: A secure communication protocol and set of security protocol used to create a secure channel with token during TPS Operations.

- Applets: Applet is a smartcard application that holds the certs and keys

- APDU(Application Protocol Data Unit): Standard communication messaging protocol between a card accepting device(Reader) and a smart card

- SmartCards come with a cardmanager applet and a vendor applet or with only card manager applet.

- Coolkey: applet is the supported Card Manager applet

- Smartcard has cryptographic functionality supporting symmetric and Assymetric Encryption(like RSA)

- CCID: CCID (chip card interface device) protocol is a USB protocol that allows a smartcard to be connected to a computer via a card reader using a standard USB interface, without the need for each manufacturer of smartcards to provide its own reader or protocol. [12]

4.8.4 Token Key Service

- Introduction

  - A subsystem in the token management system which derives specific, separate keys for every smart card based on the smart card APDUs and other shared information, like the token CUID. [4]
  - The TKS derives keys based on the token CCID, private information, and a defined algorithm. These derived keys are used by the TPS to format tokens and enroll, or process, certificates on the token(smartcard). [4]
  - TKS and TPS have a sharedSecret to wrap sensitive info sent between TKS and TPS

- Features:

  - **Manage shared keys**

    * Token Key Service (TKS) derives keys for the TPS to use.
    * Based on Token CUID, and other token material , Derive keys that encrypt session between TPS and ESC
    * Keys are derived based on Common Master Key which is known to TKS and existent on smartcards

  - **Generate Master Key**

    * Master key is 3DES symmetric Key stored in software/Hardware
    * Transport Key wraps the other keys derived(from master key) and used by TKS and TPS

4.8.5 Token Processing Service

- Introduction

  - TPS interacts with smart cards to help them generate and store keys and certificates for a specific entity
  - Smart card operations go through TPS and are forwarded to the appropriate subsystem for action, such as the Certificate Authority to generate certificates or the Key Recovery Authority to archive and recover key

- Features

  - Secure Channel

    * TPS Interacts with Smartcard over a secure channel. Based on the keys derived by TKS.
    * TPS uses developer key sets(KEK,AUTH,MAC) to initially setup Secure Channel
    * In case smartcard wants to upgrade keys, A new set of keys created by TKS are used to setup Secure Channel

  - Supported TPS Operations

    * Formatting Smartcards

* Resetting PIN on smartcards
* Upgrading the applet for smartcard tokens
* Perform LDAP Authentication
* Managing token database
* Logging Token events

– Token Profiles

* Just like certificate profiles there are different token profiles to format different kind of tokens
* Token profiles define the below:
    · The steps to format and enroll the token
    · The configuration of the final enrolled token
    · Profiles also specify which LDAP server to be used for authentication and which CA should be used for enrolling certificates
    · List of Token profiles:

Table 2.1. Default Token Types

Token Type	Description
cleanToken	For operations for any blank token, without any other applied token types.
soKey	For operations for generating keys for security officer stations.
soCleanSOToken	For operations for blank tokens for security officer stations.
soKeyTemporary	For operations for temporary security officer tokens.
soCleanUserToken	For operations for blank user tokens for security officers.
soUserKey	For operations for security officer user tokens.
tokenKey	For operations for generating keys for uses with servers or devices.
userKey	For operations for regular user tokens.
userKeyTemporary	For operations for temporary user tokens.

Figure 32: List of Token Profiles

– TKS Configuration

* RPM: **pki-tks**
* Configuration: TKS subsystem can be configured through pkispawn using interactive or to have a customize setup a separate file can be specified with pkispawn
* TKS subsystem can share the same tomcat instance created by the CA subsystem which allows TKS to use same ports as that of CA

Figure 33: Interactive installation of TKS using pkispawn

– Creating sharedSecret

  ∗ TKS generates a **sharedSecret** to wrap sensitive information passed over TPS.
  ∗ This sharedSecret is to be created on TKS NSS database using tool called **tkstool**

Figure 34: NSS Internal database password

Figure 35: tkstool Usage

```
Generating the second session key share . . .

    second session key share:      FBCD 5868 AD73 3857
                                   5EF2 CED0 E3DC 34DF

    second session key share KCV:  827F 9805

(1) Write down and save the value for this second session key share.
(2) Write down and save the KCV value for this second session key share.

Type the word "proceed" and press enter to continue (or ^C to break):  █
```

Figure 36: First Session Key

```
Generating the third session key share . . .

    third session key share:       FE31 7FD0 831C D64C
                                   D92A 3B54 15F4 917F

    third session key share KCV:   7C0B AB35

(1) Write down and save the value for this third session key share.
(2) Write down and save the KCV value for this third session key share.

Type the word "proceed" and press enter to continue (or ^C to break):  █
```

Figure 37: Second Session Key

```
Generating first symmetric key . . .
Generating second symmetric key . . .
Generating third symmetric key . . .
Extracting transport key from operational token . . .

    transport key KCV:  6F68 C5A0

Storing transport key on final specified token . . .
Naming transport key "sharedSecret" . . .
Successfully generated, stored, and named the transport key!

[root@pki2 alias]# █
```

Figure 38: Third Session Key

* tkstool utility prints out the key shares and KCV values for each
  of the three session keys that are generated. These are necessary
  to import the **sharedSecret** to TPS
– TPS Configuration
  * RPM: **pki-tps**
  * Configuration: TPS subsystem can be configured through pkispawn
    using interactive or to have a customize setup a separate file can
    be specified with pkispawn
  * TPS subsystem can share the same tomcat instance created by
    the CA subsystem which allows TPS to use same ports as that
    of CA

Figure 39: Interactive installation of TKS using pkispawn

Figure 40: Continuation of pkispawn

∗ Import SharedSecret
  · Import the sharedSecret created by TKS so that TKS and
    TPS can wrap sensitive information
  · sharedSecret needs to be imported in to TPS NSS Database

```
[root@pki2 conf]# pwd
/var/lib/pki/FoobarTPS/conf
[root@pki2 conf]# cat password.conf
internal=548782292471
internaldb=Secret123
replicationdb=-352748034
[root@pki2 conf]# tkstool -I -d /var/lib/pki/FoobarTPS/alias/ -n sharedSecret
Enter Password or Pin for "NSS Certificate DB":
```

Figure 41: Get NSS Password of TPS NSS database

```
Enter the first session key share . . .

Type in the first session key share (or ^C to break):

    first session key share:       73FD 68D0 0D08 E6F1
                                   B949 9BD9 EF64 D3B9

Type in the corresponding KCV for the first session key share (or ^C to break):

    first session key share KCV:  1392 8DB8

Verifying that this session key share and KCV correspond to each other . . .

Congratulations, the first session key share KCV value entered CORRESPONDS
to the first session key share value entered!

Type the word "proceed" and press enter to continue (or ^C to break):  
```

Figure 42: tkstool to import first session key

```
Enter the second session key share . . .

Type in the second session key share (or ^C to break):

    second session key share:       FBCD 5868 AD73 3857
                                    5EF2 CED0 E3DC 34DF

Type in the corresponding KCV for the second session key share (or ^C to break):

    second session key share KCV:  827F 9805

Verifying that this session key share and KCV correspond to each other . . .

Congratulations, the second session key share KCV value entered CORRESPONDS
to the second session key share value entered!

Type the word "proceed" and press enter to continue (or ^C to break):  
```

Figure 43: tkstool to import second session key

```
Enter the third session key share . . .

Type in the third session key share (or ^C to break):

    third session key share:       FE31 7FD0 831C D64C
                                   D92A 3B54 15F4 917F

Type in the corresponding KCV for the third session key share (or ^C to break):

    third session key share KCV:  7C0B AB35

Verifying that this session key share and KCV correspond to each other . . .

Congratulations, the third session key share KCV value entered CORRESPONDS
to the third session key share value entered!

Type the word "proceed" and press enter to continue (or ^C to break):  proceed
```

Figure 44: tkstool to import third session key

4.8.6 Enterprise Security Client

- **Features:**

  - Enterprise Security Client is the conduit through which TPS communicates with each token over a secure HTTP channel (HTTPS), and, through the TPS, with the Certificate System.

  - Enterprise Security Client provides the user interface for end entities to enroll tokens.

- **RPM**: esc

- **Platforms**:

  - RHEL5(x86_64,i686)
  - RHEL6(x86_64,i686)
  - RHEL7(x86_64,i686)
  - Windows Vista(x86_64, i686)
  - Windows XP
  - Apple Mac OSx 10.5.8 and higher

- **Supported Smartcards:**

  - Safenet 330J Java smart cards
  - Gemalto 64K V2 tokens, both as a smart card and GemPCKey USB form factor key
  - Safenet 650 Smartcards
  - Personal identity verification (PIV) cards, compliant with FIPS 201

- **Installation:**

```
$ yum install esc coolkey
```

- **Configuration:**

  - Configuring Phone Home

    * The Phone Home feature in the Enterprise Security Client associates information within each smart card with information that points to distinct T PS servers and Enterprise Security Client UI pages.

    * Phone Home retrieves and then caches this information; because the information is cached locally, the TPS subsystem does not have to be contacted each time a formatted smart card is inserted

    * The information can be different for every key or token, which means that different TPS servers and enrollment URLs can be configured for different corporate or customer groups.

* Phone Home makes it possible to configure different TPS servers for different issuers or company units, without having to configure the Enterprise Security Client manually to locate the correct server and URL.
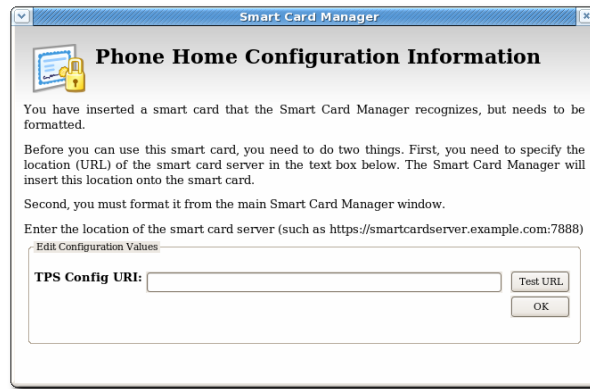
Figure 45: ESC Phone Home URL [3]

– Setting up Users to be enrolled
  * When the Token Processing System is installed, one of its configuration settings is the LDAP directory which contains the users who are allowed to enroll a token
  * Only users who are stored within this authentication directory are allowed to enroll, format, or have a token.
  * The TPS is configured to look at a specific base DN in the LDAP directory. This is configured in the TPS's CS.cfg

```
1  auth.instance.0.baseDN=dc=example,dc=com
2  auth.instance.0.hostport=server.example.com:389
```

  * If the user does not already have an entry, then the administrator must add the user to the specified LDAP directory in the specified base DN before any tokens can be enrolled for the user.

```
$ usr/bin/ldapmodify -a -D "cn=Directory Manager" -w
    secret -p 389 -h server.example.com
dn: uid=jsmith,ou=People, dc=example,dc=com
objectclass: person
objectclass: inetorgperson
objectclass: top
uid: jsmith
cn: John Smith
email: jsmith@example.com
userPassword: secret
```

4.8.7   Smartcard Workflow

- **Secure Channel** [1]

– TPS, TKS and Smartcard need to communicate with each other over a Secure Channel (once TLS is established).

– This secure channel is necessary for certain sensitive operations to be performed upon the token on behalf of the host
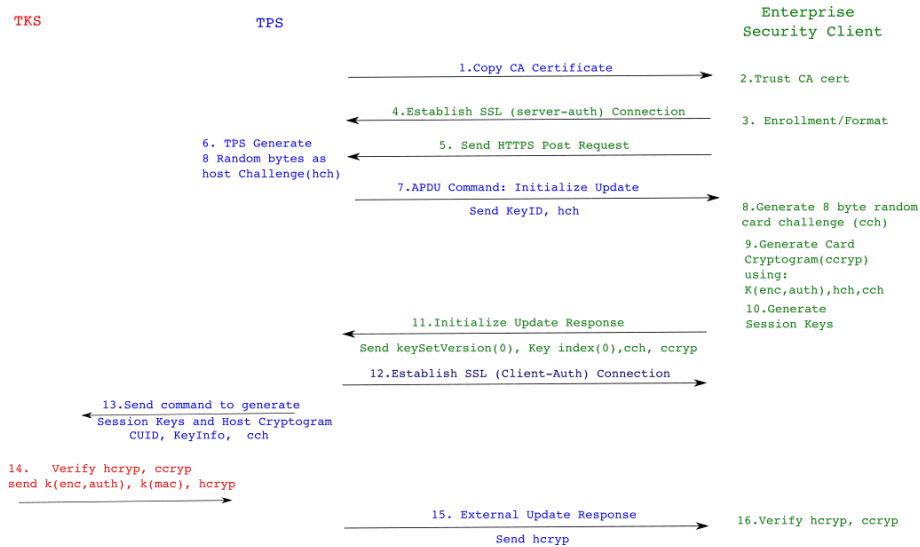
Figure 46: TKS-TPS-Smartcard Securechannel

• **Token Format**
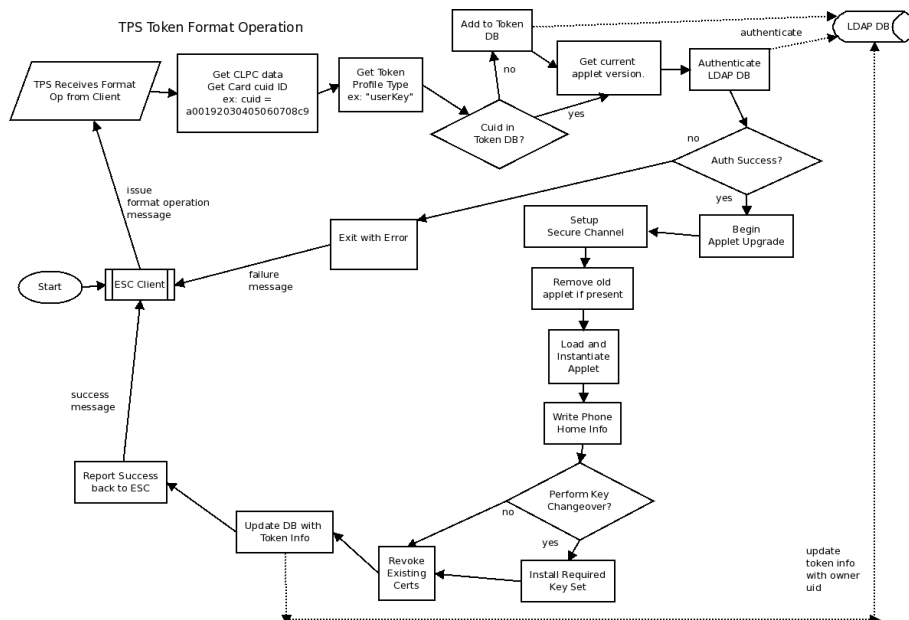
Figure 47: Token Format Operation Workflow. [1]

- **Token Enrollment**
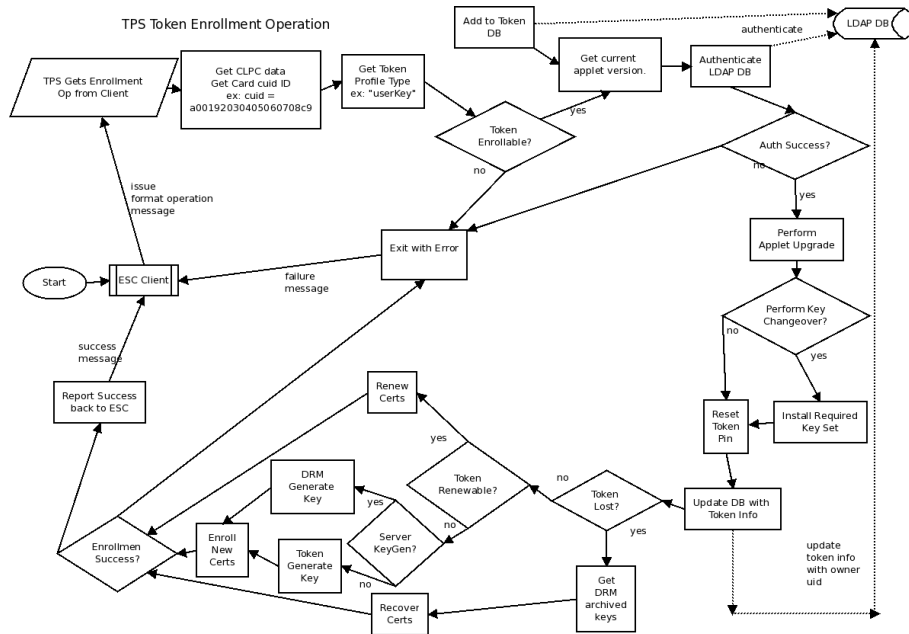
Figure 48: Token Enrollment Operation Workflow. [1]

- **Server Side Key Generation**

Figure 49: Server side key Generation

4.8.8 Exercises

1. Setup TMS Subsystem (Include KRA) all sharing single Tomcat Instance

    1.1. Configure CA subsystem using method specified in 5.

    1.2. Configure KRA subsystem using pkispawn interactive method and make KRA subsystem part of tomcat instance used by RootCA.

    ```
    Subsystem (CA/KRA/OCSP/TKS/TPS) [CA]: KRA

    Tomcat:
      Instance [pki-tomcat]:

    Administrator:
      Username [kraadmin]:
      Password:
      Verify password:
      Import certificate (Yes/No) [Y]?
      Import certificate from
          [/root/.dogtag/pki-tomcat/ca_admin.cert]:
      Export certificate to
          [/root/.dogtag/pki-tomcat/kra_admin.cert]:
    ```

```
Directory Server:
  Hostname [pki1.example.org]:
  Use a secure LDAPS connection (Yes/No/Quit) [N]?
  LDAP Port [389]:
  Bind DN [cn=Directory Manager]:
  Password:
  Base DN [o=pki-tomcat-KRA]:

Security Domain:
  Hostname [pki1.example.org]:
  Secure HTTP port [8443]:
  Name: example.org Security Domain
  Username [caadmin]:
  Password:

Begin installation (Yes/No/Quit)? Yes
```

1.3. Configure TKS subsystem using pkispawn interactive method and join to ROOTCA Tomcat Instance

```
Subsystem (CA/KRA/OCSP/TKS/TPS) [CA]: TKS

Tomcat:
  Instance [pki-tomcat]:

Administrator:
  Username [tksadmin]:
  Password:
  Verify password:
  Import certificate (Yes/No) [Y]?
  Import certificate from
      [/root/.dogtag/pki-tomcat/ca_admin.cert]:
  Export certificate to
      [/root/.dogtag/pki-tomcat/tks_admin.cert]:
Directory Server:
  Hostname [pki1.example.org]:
  Use a secure LDAPS connection (Yes/No/Quit) [N]?
  LDAP Port [389]:
  Bind DN [cn=Directory Manager]:
  Password:
  Base DN [o=pki-tomcat-TKS]:

Security Domain:
  Hostname [pki1.example.org]:
  Secure HTTP port [8443]:
  Name: example.org Security Domain
  Username [caadmin]:
  Password:

Begin installation (Yes/No/Quit)? Yes
```

1.4. Configure TPS subsystem using pkispawn interactive method and join to RootCA Tomcat Instance

```
Subsystem (CA/KRA/OCSP/TKS/TPS) [CA]: TPS

Tomcat:
  Instance [pki-tomcat]:

Administrator:
  Username [tpsadmin]:
  Password:
  Verify password:
  Import certificate (Yes/No) [Y]?
  Import certificate from
      [/root/.dogtag/pki-tomcat/ca_admin.cert]:
  Export certificate to
      [/root/.dogtag/pki-tomcat/tps_admin.cert]:
Directory Server:
  Hostname [pki1.example.org]:
  Use a secure LDAPS connection (Yes/No/Quit) [N]?
  LDAP Port [389]:
  Bind DN [cn=Directory Manager]:
  Password:
  Base DN [o=pki-tomcat-TPS]:

Security Domain:
  Hostname [pki1.example.org]:
  Secure HTTP port [8443]:
  Name: example.org Security Domain
  Username [caadmin]:
  Password:

External Servers:
  CA URL [https://pki2.example.org:8443]:
  TKS URL [https://pki2.example.org:8443]:
  Enable server side key generation (Yes/No) [N]?

Authentication Database:
  Hostname [pki2.example.org]:
  Port [389]:
  Base DN: dc=example,dc=org

Begin installation (Yes/No/Quit)? Yes
```

1.5. Verify the sharedSecret has been created for TPS and TKS with pkispawn

Note: When TPS and TKS share the same tomcat instance, sharedSecret is created as part of pkispawn configuration process.

```
$ cd /var/lib/pki/pki-tomcat/conf
$ cat password.conf
internal=153622747649
internaldb=Secret123
replicationdb=-1899602373
```

Use the password mentioned under ***internal*** for tkstool

```
$ tkstool -L -d /var/lib/pki/pki-tomcat/tps/alias


 slot: NSS User Private Key and Certificate Services
 token: NSS Certificate DB

 Enter Password or Pin for "NSS Certificate DB":
         <0> defKeySet-macKey
         <1> TPS-pki1.example.org-8443 sharedSecret
```

1.6. Run pkidaemon against Tomcat Instance pki-tomcat and find out which is the phone home url for TPS

```
$ pkidaemon status tomcat pki-tomcat
```

1.7. Open TKS CS.cfg file and Verify what keysets information can be seen:

```
$ vim /var/lib/pki/pki-tomcat/tks/conf/CS.cfg
```

```
tks.defKeySet._000=##
tks.defKeySet._001=## Axalto default key set:
tks.defKeySet._002=##
tks.defKeySet._003=##
    tks.defKeySet.mk_mappings.#02#01=<tokenname>:<nickname>
tks.defKeySet._004=##
tks.defKeySet.auth_key=#40#41#42#43#44#45#46#47#48#49#4a#4b#4c#4d#4e#4f
tks.defKeySet.kek_key=#40#41#42#43#44#45#46#47#48#49#4a#4b#4c#4d#4e#4f
tks.defKeySet.mac_key=#40#41#42#43#44#45#46#47#48#49#4a#4b#4c#4d#4e#4f
tks.defKeySet.nistSP800-108KdfOnKeyVersion=00
tks.defKeySet.nistSP800-108KdfUseCuidAsKdd=false
tks.defaultSlot=Internal Key Storage Token
tks.drm_transport_cert_nickname=
tks.jForte._000=##
tks.jForte._001=## SAFLink's jForte default key set:
tks.jForte._002=##
tks.jForte._003=##
    tks.jForte.mk_mappings.#02#01=<tokenname>:<nickname>
tks.jForte._004=##
tks.jForte.auth_key=#30#31#32#33#34#35#36#37#38#39#3a#3b#3c#3d#3e#3f
tks.jForte.kek_key=#50#51#52#53#54#55#56#57#58#59#5a#5b#5c#5d#5e#5f
tks.jForte.mac_key=#40#41#42#43#44#45#46#47#48#49#4a#4b#4c#4d#4e#4f
tks.jForte.nistSP800-108KdfOnKeyVersion=00
tks.jForte.nistSP800-108KdfUseCuidAsKdd=false
tks.master_key_prefix=
```

1.8. Open TPS CS.cfg and Check what authentication Directory is mentioned which will be looked to authenticate users against to enroll tokens

```
$ vim /var/lib/pki/pki-tomcat/tps/conf/CS.cfg
```

```
auths.instance.ldap1.ldap.basedn=dc=example,dc=org
auths.instance.ldap1.ldap.ldapauth.authtype=BasicAuth
auths.instance.ldap1.ldap.ldapauth.bindDN=
auths.instance.ldap1.ldap.ldapauth.bindPWPrompt=ldap1
auths.instance.ldap1.ldap.ldapauth.clientCertNickname=subsystemCert
    cert-pki-tomcat
auths.instance.ldap1.ldap.ldapconn.host=pki2.example.org
auths.instance.ldap1.ldap.ldapconn.port=389
auths.instance.ldap1.ldap.ldapconn.secureConn=False
auths.instance.ldap1.ldap.ldapconn.version=3
auths.instance.ldap1.ldap.maxConns=15
auths.instance.ldap1.ldap.minConns=3
auths.instance.ldap1.ldapByteAttributes=
```

1.9. Go through all the entries starting with op.enroll.userKey. What does the entries tell about

1.10. Go through all the entries starting witn op.format.userKey and find out more about them

1.11. From the entries of userKey, can you find where is CA , TKS and KRA information ?

1.12. Add few users to ldap (Authenticating Directory) used by LDAP to enroll tokens

users.ldif

```
# jdoe, example.org
dn: uid=jdoe,ou=People,dc=example,dc=org
uid: jdoe
givenName: jdoe
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetorgperson
userPassword: redhat
sn: doe
cn: jdoe

# foo1, example.org
dn: uid=foo1,ou=People,dc=example,dc=org
uid: foo1
givenName: foo1
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetorgperson
userPassword: redhat
sn: 1
cn: foo1

# foo2, example.org
dn: uid=foo2,ou=People,dc=example,dc=org
uid: foo2
```

```
                    givenName: foo
                    objectClass: top
                    objectClass: person
                    objectClass: organizationalPerson
                    objectClass: inetorgperson
                    userPassword: redhat
                    sn: 2
                    cn: foo2

                    # foo3, example.org
                    dn: uid=foo3,ou=People,dc=example,dc=org
                    uid: foo3
                    givenName: foo
                    objectClass: top
                    objectClass: person
                    objectClass: organizationalPerson
                    objectClass: inetorgperson
                    userPassword: redhat
                    sn: 3
                    cn: foo3

                    # foo4, example.org
                    dn: uid=foo4,ou=People,dc=example,dc=org
                    uid: foo4
                    givenName: foo4
                    objectClass: top
                    objectClass: person
                    objectClass: organizationalPerson
                    objectClass: inetorgperson
                    userPassword: redhat
                    sn: 4
                    cn: foo4
```

```
$ ldapadd -x -D "cn=Directory Manager" -h pki1.example.org -w
    Secret123 -f users.ldif
```

1.13. Enroll a token with user jdoe with token CUID "40000000000000000001"
using tpsclient

- create a file "enroll.jdoe.tps" with below contents:

```
1   op=var_set name=ra_host value=pki1.example.org
2   op=var_set name=ra_port value=8080
3   op=var_set name=ra_uri value=/tps/tps
4   op=token_set cuid=40000000000000000001 msn=01020304
        app_ver=6FBBC105 key_info=0101 major_ver=0 minor_ver=0
5   op=token_set auth_key=404142434445464748494a4b4c4d4e4f
6   op=token_set mac_key=404142434445464748494a4b4c4d4e4f
7   op=token_set kek_key=404142434445464748494a4b4c4d4e4f
8   op=token_status
9   op=ra_enroll uid=jdoe num_threads=1 pwd=redhat
        new_pin=redhat extensions=tokenType=userKey
10  op=token_status
```

```
11   op=exit
```

- Run tpsclient using the above input file

```
$ tpsclient < enroll.jdoe.tps
```

- Output of tpsclient should look like below

```
-----
Output> Thread (0) status='1' time='104606 msec'
Result> Success - Operation 'ra_enroll' Success (104607
    msec)
Command>op=token_status
Output> life_cycle_state : '0'
Output> pin : 'redhat'
Output> app_ver : '05' (4 bytes)
Output> major_ver : '0'
Output> minor_ver : '0'
Output> cuid : '01' (10 bytes)
Output> msn : '04' (4 bytes)
Output> key_info : '01' (2 bytes)
Output> auth_key : '4f' (16 bytes)
Output> mac_key : '4f' (16 bytes)
Output> kek_key : '4f' (16 bytes)
Result> Success - Operation 'token_status' Success (0
    msec)
Command>op=exit
```

1.14. Import CA Admin Cert and KRA Admin p12 file to Firefox browser. Refer: 7.6.

1.15. From the browser Access CA EE Page and check what certificates were issued

1.16. Access KRA Agent Interface and verify if the Private Key has been archived

1.17. From the TPS UI check the token information

- Access https://pki1.example.org:8443
- Use the Admin cert when prompted
- Open each subsystem on different tabs
- From TPS UI, click on "Login"
  - Click on Tokens , Should see the list of enrolled tokens
  - click on "Certificates" and should see the certificates associated with tokens

1.18. Format tthe token "40000000000000000001" and verify the associated certs are revoked

- Create a file format.jdoe.tps with below contents

```
op=var_set name=ra_host value=pki1.example.org
op=var_set name=ra_port value=8080
op=var_set name=ra_uri value=/tps/tps
```

```
op=token_set cuid=40000000000000000001 msn=01020304
    app_ver=6FBBC105 key_info=0101 major_ver=0 minor_ver=0
op=token_set auth_key=404142434445464748494a4b4c4d4e4f
op=token_set mac_key=404142434445464748494a4b4c4d4e4f
op=token_set kek_key=404142434445464748494a4b4c4d4e4f
op=token_status
op=ra_format uid=jdoe num_threads=1 pwd=redhat
    new_pin=redhat extensions=tokenType=userKey
op=token_status
op=exit
```

```
$ tpsclient < format.jdoe.tps
```

1.19. Check the TPS UI and CA EE interface to verify the associated certs have been revoked.

1.20. Enroll a token with user foo1 with token CUID "40000000000000000002" using tpsclient

1.21. create a file "enroll.foo1.tps" with below contents:

```
1   op=var_set name=ra_host value=pki1.example.org
2   op=var_set name=ra_port value=8080
3   op=var_set name=ra_uri value=/tps/tps
4   op=token_set cuid=40000000000000000002 msn=01020304
        app_ver=6FBBC105 key_info=0101 major_ver=0 minor_ver=0
5   op=token_set auth_key=404142434445464748494a4b4c4d4e4f
6   op=token_set mac_key=404142434445464748494a4b4c4d4e4f
7   op=token_set kek_key=404142434445464748494a4b4c4d4e4f
8   op=token_status
9   op=ra_enroll uid=foo1 num_threads=1 pwd=redhat new_pin=redhat
        extensions=tokenType=userKey
10  op=token_status
11  op=exit
```

1.22. Run tpsclient using the above input file

```
$ tpsclient < enroll.foo1.tps
```

1.23. Login to TPS UI , select the token "40000000000000000002" , On the righ side click on "Change Status", Mark the status to "Temporary Lost". From CA EE page and TPS Certificates page, verify the status of the certificates

1.24. Login to TPS UI and mark the token "40000000000000000002" as "Active" and check the status of certificates from TPS and CA UI

1.25. Login to TPS UI and mark the token "40000000000000000002" as Permanently Lost. Verify the status of Certificates from TPS UI and CA EE pages.

2. Setup TMS Subsystem (Include KRA) on separate Tomcat Instance

   2.1. Configure CA subsystem using below config file.
        Note: We will be using existing Directory Server

ca-inst.inf:

```
 1   [DEFAULT]
 2   pki_instance_name=FoobarCA
 3   pki_https_port=30042
 4   pki_http_port=30044
 5
 6   #NSS DB Token Password
 7   pki_token_password=Secret123
 8
 9   #Admin Password
10   pki_admin_password=Secret123
11
12   #Security Domain
13   pki_hostname=pki1.example.org
14   pki_security_domain_name=Foobar Org
15   pki_security_domain_password=Secret123
16
17   #client Dir
18   pki_client_dir=/opt/FoobarCA
19   pki_client_pkcs12_password=Secret123
20
21   #Backup
22   pki_backup_keys=True
23   pki_backup_password=Secret123
24
25   #DS
26   pki_ds_hostname=pki1.example.org
27   pki_ds_ldap_port=389
28   pki_ds_password=Secret123
29   pki_ds_secure_connection=False
30
31   [Tomcat]
32   pki_ajp_port=30009
33   pki_tomcat_server_port=30005
34
35   [CA]
36   pki_admin_nickname=PKI CA Administrator for Foobar Org
37   pki_import_admin_cert=False
```

```
$ pkispawn -s CA -f ca-inst.inf -vv
```

2.2. Configure KRA subsystem using below config file
     kra-inst.inf:

```
 1   [DEFAULT]
 2   pki_instance_name=FoobarKRA
 3   pki_https_port=14443
 4   pki_http_port=14080
 5
 6   #NSS DB Token Password
 7   pki_token_password=Secret123
 8
```

```
 9   #RootKRA Admin password
10   pki_admin_password=Secret123
11
12   #Security Domain
13   pki_hostname=pki1.example.org
14   pki_security_domain_hostname=pki1.example.org
15   pki_security_domain_https_port=30042
16   pki_security_domain_user=caadmin
17   pki_security_domain_password=Secret123
18
19   #Client Dir
20   pki_client_dir=/opt/FoobarKRA
21   pki_client_pkcs12_password=Secret123
22   pki_client_database_password=Secret123
23
24   #Backup
25   pki_backup_keys=True
26   pki_backup_password=Secret123
27   #DS
28   pki_ds_hostname=pki1.example.org
29   pki_ds_ldap_port=389
30   pki_ds_bind_dn=cn=Directory Manager
31   pki_ds_password=Secret123
32
33
34   [Tomcat]
35   pki_ajp_port=14009
36   pki_tomcat_server_port=14005
37
38   [KRA]
39   pki_admin_nickname=PKI KRA Administrator for Foobar Org
40   pki_import_admin_cert=False
```

```
$ pkispawn -s KRA -f kra-inst.inf -vv
```

2.3. Configure TKS subsystem using below config file
tks-inst.inf:

```
 1   [DEFAULT]
 2   pki_instance_name=FoobarTKS
 3   pki_https_port=18443
 4   pki_http_port=18080
 5
 6   #NSS DB Token Password
 7   pki_token_password=Secret123
 8
 9   #TKS Admin password
10   pki_admin_password=Secret123
11
12   # Security Domain
13   pki_hostname=pki1.example.org
14   pki_security_domain_hostname=pki1.example.org
15   pki_security_domain_https_port=30042
```

```
16   pki_security_domain_user=caadmin
17   pki_security_domain_password=Secret123
18
19   #Client Dir
20   pki_client_dir=/opt/FoobarTKS
21   pki_client_pkcs12_password=Secret123
22   pki_client_database_password=Secret123
23
24   # backup
25   pki_backup_keys=True
26   pki_backup_password=Secret123
27
28   # DS
29   pki_ds_hostname=pki1.example.org
30   pki_ds_ldap_port=389
31   pki_ds_bind_dn=cn=Directory Manager
32   pki_ds_password=Secret123
33
34   [Tomcat]
35   pki_ajp_port=18009
36   pki_tomcat_server_port=18005
37
38   [TKS]
39   pki_admin_nickname=PKI TKS Administrator for Example Org
40   pki_import_admin_cert=False
```

```
$ pkispawn -s TKS -f tks-inst.inf -vv
```

2.4. Stop the TKS subsystem and create shared Secret in TKS

```
$ systemctl stop pki-tomcatd@FoobarTKS
```

```
$ cd /var/lib/pki/FoobarTKS/conf
$ cat password.conf
internal=187041555763
internaldb=Secret123
replicationdb=1433722269
# Use the password mentioned in internal for tkstool
$ cd /var/lib/pki/FoobarTKS/alias
#This creates 3 session keys, #save each session keys
#and KCV values for each session in file
$ tkstool -T -d . -n sharedSecret
$ reset
```

2.5. Start the TKS subsystem

```
$ systemctl start pki-tomcatd@FoobarTKS
```

2.6. Configure TPS subsystem on separate Tomcat instance using below
     config file
     tps-inst.inf:

```
 1  [DEFAULT]
 2  pki_instance_name=FoobarTPS
 3  pki_https_port=20443
 4  pki_http_port=20080
 5
 6  ##NSS DB Token Password
 7  pki_token_password=Secret123
 8
 9  #Security Domain
10  pki_hostname=pki1.example.org
11  pki_security_domain_hostname=pki1.example.org
12  pki_security_domain_https_port=30042
13  pki_security_domain_user=caadmin
14  pki_security_domain_password=Secret123
15
16  #TPS Admin password
17  pki_admin_password=Secret123
18
19  # Client dir
20  pki_client_database_dir=/opt/FoobarTPS
21  pki_client_pkcs12_password=Secret123
22  pki_client_database_password=Secret123
23
24  #backup
25  pki_backup_keys=True
26  pki_backup_password=Secret123
27
28  #DS
29  pki_ds_hostname=pki1.example.org
30  pki_ds_ldap_port=389
31  pki_ds_bind_dn=cn=Directory Manager
32  pki_ds_password=Secret123
33
34  [Tomcat]
35  pki_ajp_port=20009
36  pki_tomcat_server_port=20005
37
38
39  [TPS]
40  pki_admin_nickname=PKI TPS Administrator for Foobar Org
41  pki_import_admin_cert=False
42  pki_ca_uri=https://pki1.example.org:30042
43  pki_kra_uri=https://pki1.example.org:14443
44  pki_tks_uri=https://pki1.example.org:18443
45  pki_authdb_hostname=pki1.example.org
46  pki_authdb_port=389
47  pki_authdb_basedn=dc=example,dc=org
48  pki_authdb_secure_conn=False
49  pki_import_shared_secret=False
```

```
$ pkispawn -s TPS -f tps_inst.inf -vv
```

2.7. Stop TPS Instance and Import the shared Secret created earlier

```
$ systemctl stop pki-tomcatd@FoobarTPS
```

2.8. Import the sharedSecret from TPS

```
$ cd /var/lib/pki/FoobarTPS/conf/
$ cat password.conf
internal=321588268669
internaldb=Secret123
replicationdb=1975043500
# use the password mentioned under "internal" variable
# with tkstool
$ cd /var/lib/pki/FoobarTPS/alias
# when running this command will prompt to enter session
# keys , Enter the session keys and KCV which were saved while
# creating sharedSecret in TKS
$ tkstool -I -d . -n sharedSecret
```

2.9. Edit TPS CS.cfg to modify the below line to specify correct shared-Secret nickname

```
conn.tks1.tksSharedSymKeyName=TPS-pki2.example.org-20443
    sharedSecret
```

```
$ vim /var/lib/pki/FoobarTPS/tps/conf/CS.cfg
conn.tks1.tksSharedSymKeyName=sharedSecret
```

2.10. Edit TKS CS.cfg to remove "TPS-pki2.example.org-20443 sharedSecret" and replace with "sharedSecret"

```
tps.0.nickname=TPS-pki2.example.org-20443 sharedSecret
```

```
$ vim /var/lib/pki/FoobarTPS/tps/conf/CS.cfg
tps.0.nickname=sharedSecret
```

2.11. Restart TKS and TPS subsystems

```
$ systemctl restart pki-tomcatd@FoobarTKS
$ systemctl restart pki-tomcatd@FoobarTPS
```

2.12. Do all the exercises from 1.12. to 1.25.

3. Configure Multiple Master Keys and do a key changeover

3.1. Change to TKS alias directory and create a Master key master_key_v2

```
$ tkstool -M -n master_key_v2 -d /var/lib/pki/FoobarTKS/alias/
    -h internal
```

```
Enter Password or Pin for "NSS Certificate DB":

Generating and storing the master key on the specified token .
    . .

Naming the master key "master_key_v2" . . .

Computing and displaying KCV of the master key on the
    specified token . . .

    master_key_v2 key KCV: 3EA6 B8F3


    Successfully generated, stored, and named the master key
    including computing and displaying its KCV!
```

3.2. List the Keys in NSS database

```
$ tkstool -L -d /var/lib/pki/FoobarTKS/alias/
Enter Password or Pin for "NSS Certificate DB":
        <0> master_key_v2
        <1> TPS-pki2.example.org-20443 sharedSecret
        <2> sharedSecret
```

3.3. Edit TKS CS.cfg and add the below lines

```
tks.mk_mappings.#02#01=internal:master_key_v2
tks.defKeySet.mk_mappings.#02#01=internal:master_key_v2
```

3.4. Enroll a token which will be in the default **#01#01**
     enroll.foo2.tps

```
 1  op=var_set name=ra_host value=pki1.example.org
 2  op=var_set name=ra_port value=8080
 3  op=var_set name=ra_uri value=/tps/tps
 4  op=token_set cuid=40000000000000000003 msn=01020304
        app_ver=6FBBC105 key_info=0101 major_ver=0 minor_ver=0
 5  op=token_set auth_key=404142434445464748494a4b4c4d4e4f
 6  op=token_set mac_key=404142434445464748494a4b4c4d4e4f
 7  op=token_set kek_key=404142434445464748494a4b4c4d4e4f
 8  op=token_status
 9  op=ra_format uid=foo2 num_threads=1 pwd=redhat new_pin=redhat
        extensions=tokenType=userKey
10  op=token_status
11  op=exit
```

```
$ tpsclient < enroll.foo2.tps
```

3.5. Do a Key Change over from #01 to #02 . Mofify the tps CS.cfg to
     upgrade the symmetric Keys from v1 to v2.

```
op.enroll.userKey.update.symmetricKeys.enable=true
op.enroll.userKey.update.symmetricKeys.requiredVersion=2
```

```
$ tpsclient < enroll.foo2.tps
```

3.6. Go to TPS UI and verify in the tokens tab and click on token enrolled should show key info #01#02

3.7. Do a Key Change over from #02 to #01 . Mofify the tps CS.cfg to upgrade the symmetric Keys from v1 to v2.

```
op.enroll.userKey.update.symmetricKeys.enable=true
op.enroll.userKey.update.symmetricKeys.requiredVersion=1
```

```
$ tpsclient < enroll.foo2.tps
```

References

[1] Christina Fu and Jack Magne. Tps module: Secure channel. http://alpha.dsdev.sjc.redhat.com/home/cfu/CCdelivered/RHCS8_CommonCriteria/Evidence_Documents/tmp/en-US/html/Dev-TDS_LLD_TPSMod_SecureChannel.html.

[2] Red Hat Certificate System Installation & Deployment Guide. Red hat certificate system overview. https://access.redhat.com/documentation/en-US/Red_Hat_Certificate_System/8.1/html/Deploy_and_Install_Guide/SubsystemOverview.html, 2015.

[3] Red Hat. Managing single sign-on and smartcards. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Managing_Smart_Cards/Using_the_Enterprise_Security_Client-Phone_Home.html, 2014.

[4] Red Hat. Red hat certificate services 8.0 admin guide. https://access.redhat.com/documentation/en-US/Red_Hat_Certificate_System/8.1/html/Admin_Guide/index.html, 2015.

[5] Red Hat. Symmetric encryption. https://access.redhat.com/documentation/en-US/Red_Hat_Certificate_System/9/html/Planning_Installation_and_Deployment_Guide/Introduction_to_Public_Key_Cryptography.html#Symmetric_Key_Encryption-Symmetric_Key_Encryption, 2015.

[6] B. Kaliski M. Nystrom. PKCS #10: Certification Request Syntax Specification. RFC 2986, RFC Editor, November 2000.

[7] NIST. Fips publication. http://csrc.nist.gov/publications/PubsFIPS.html, 2015.

[8] Eric Rescorla. *SSL and TLS Designing and Building Secure System.* Addison-Wesely, 2001.

[9] S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, RFC Editor, MAY 2008.

[10] J. Schaad. Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF). RFC 4211, RFC Editor, September 2005.

[11] Huzaifa Sidhpurwala. Understanding rsa for people who don't have a phd in maths. http://file.pnq.redhat.com/~huzaifas/RSA.pdf, 2015.

[12] Wikipedia. Ccid (protocol). https://en.wikipedia.org/wiki/CCID_%28protocol%29, 2015.