Workshop on Public Key Infrastructure and Red Hat Certificate System

Niranjan M.R, Huzaifa Sidhpurwala, Asha Akkiangady

Contents

1 Introduction to Public Key Cryptography

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

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.

Below are some of the threat models associated with above communcation:

- **Eavesdropping:**

  Information remains intact, But it's privacy is compromised, For example: some one could learn 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.

- 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):

Plain Text $\xrightarrow{\text{Encryption}}$ Cipher Text $\xrightarrow{\text{Decryption}}$ 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*citation required?.

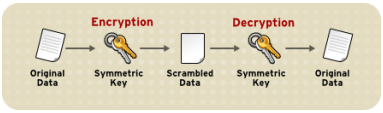

Figure 1: Symmetric Encryption

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 encrypt 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.

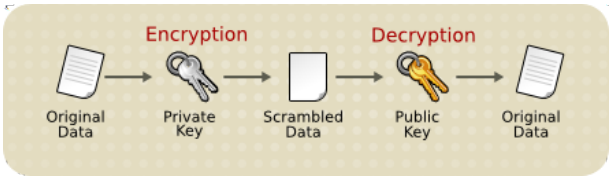

Figure 2: Assymetric Encryption

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

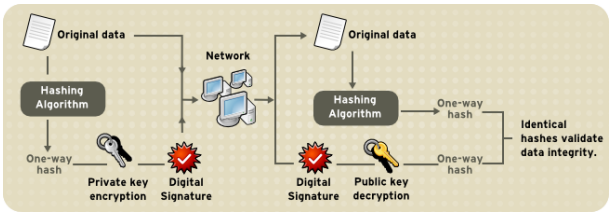

Figure 3: Digital Signature

- 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

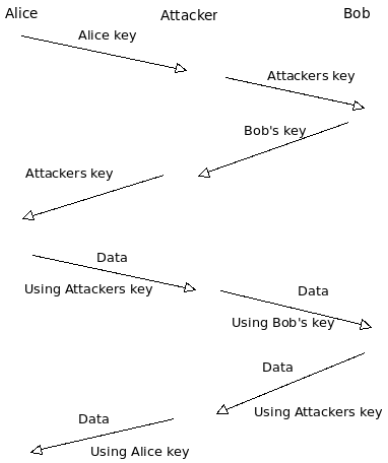

Figure 4: Man-in-the-middle attack

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.

1.3 Protocols

1.4 Exercises

2 Introduction to Public Key Infrastructure

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

2.1 Common Terms used in PKI

- End Entiy: 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

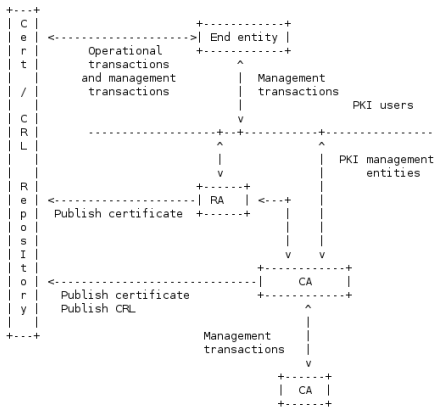

Figure 1 - PKI Entities

Figure 5: PKI Architectural Model

- 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 certificate PCA's with similar DN
  - Certificates should not be issued to distinct entities under the same distinguished Name.
  - IPRA should not certifiy two PCA's with same DN
  - PCA's should not certify two CA's with same DN
  - 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 sgin the certificate

2.2.2 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).

2.2.3 Basic Certificate Fileds

- 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**

2.3 Exercises

3 Red Hat Certificate System

3.1 Certificate Manager

3.1.1 Introduction

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

3.1.2 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.

3.1.3 Key Features

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

Figure 6: Configuring CA subsystem using pkispawn

- 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

3.1.4 Architecture

3.1.5 Interfaces

- End User Interface(Browser/CLI)
- Agent Interface(Browser/CLI)
- Admin interface(java console)

3.1.6 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:

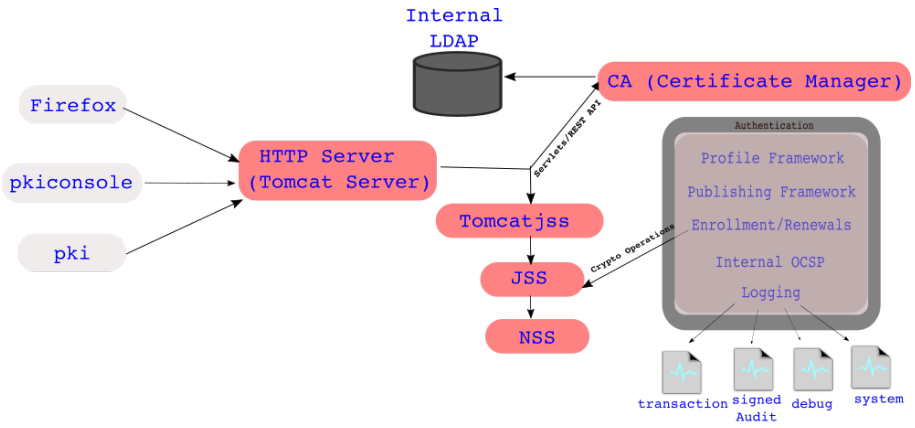

Figure 7: CA Subsystem Architecture

- 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.

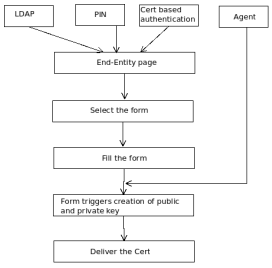

Figure 8: Certificate Enrollment

- 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

  Profile Workflow:

  Each profile are defined in **.cfg** file located at **/var/lib/instance_name/profiles/ca** directory

  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.

  The second part of the profile describes the inputs:

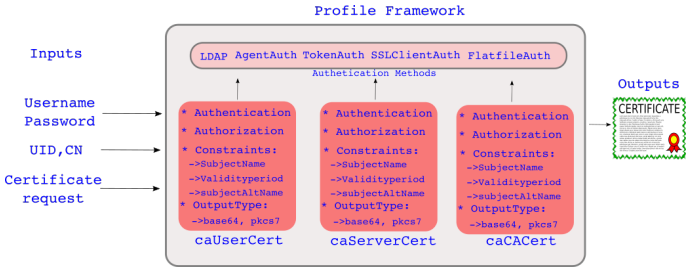

Figure 9: Certificate Profile Architecture

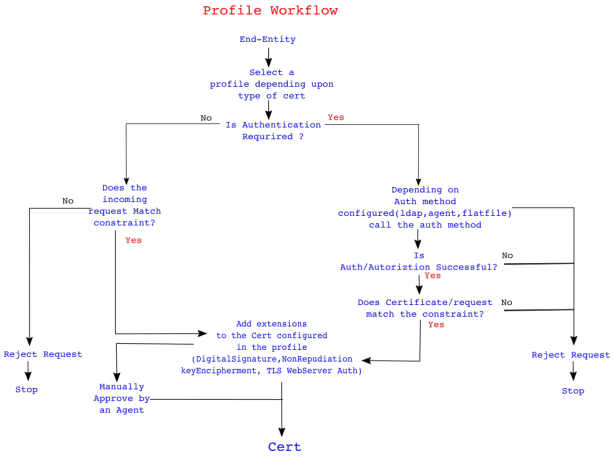

Figure 10: Certificate Profiles Workflow

```
desc=This certificate profile is for enrolling user certificates.
visible=true
enable=true
enableBy=admin
name=Manual User Dual-Use Certificate Enrollment
```

Figure 11: Profile Description

```
auth.class_id=
input.list=i1,i2,i3
input.i1.class_id=keyGenInputImpl
input.i2.class_id=subjectNameInputImpl
input.i3.class_id=submitterInfoInputImpl
```

Figure 12: Profile inputs

- KeyGenInputImpl: specifies the key pair generation during the request submission, This provides if the request should be of type CRMF/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,

```
output.list=o1
output.o1.class_id=certOutputImpl
```

Figure 13: Profile output

– certOutputImpl: The certificate output format *base64, pkcs7, pret-typrint*

```
policyset.userCertSet.list=1,10,2,3,4,5,6,7,8,9
policyset.userCertSet.1.constraint.class_id=subjectNameConstraintImpl
policyset.userCertSet.1.constraint.name=Subject Name Constraint
policyset.userCertSet.1.constraint.params.pattern=UID=.*
policyset.userCertSet.1.constraint.params.accept=true
policyset.userCertSet.1.default.class_id=userSubjectNameDefaultImpl
policyset.userCertSet.1.default.name=Subject Name Default
policyset.userCertSet.1.default.params.name=
policyset.userCertSet.10.constraint.class_id=renewGracePeriodConstraintImpl
policyset.userCertSet.10.constraint.name=Renewal Grace Period Constraint
policyset.userCertSet.10.constraint.params.renewal.graceBefore=30
policyset.userCertSet.10.constraint.params.renewal.graceAfter=30
policyset.userCertSet.10.default.class_id=noDefaultImpl
policyset.userCertSet.10.default.name=No Default
policyset.userCertSet.2.constraint.class_id=validityConstraintImpl
policyset.userCertSet.2.constraint.name=Validity Constraint
policyset.userCertSet.2.constraint.params.range=365
policyset.userCertSet.2.constraint.params.notBeforeCheck=false
policyset.userCertSet.2.constraint.params.notAfterCheck=false
policyset.userCertSet.2.default.class_id=validityDefaultImpl
policyset.userCertSet.2.default.name=Validity Default
policyset.userCertSet.2.default.params.range=180
policyset.userCertSet.2.default.params.startTime=0
policyset.userCertSet.3.constraint.class_id=keyConstraintImpl
policyset.userCertSet.3.constraint.name=Key Constraint
policyset.userCertSet.3.constraint.params.keyType=
policyset.userCertSet.3.constraint.params.keyParameters=1024,2048,3072,4096,nistp256,nistp384,nistp521
policyset.userCertSet.3.default.class_id=keyDefaultImpl
policyset.userCertSet.3.default.name=Key Default
policyset.userCertSet.4.constraint.class_id=noConstraintImpl
policyset.userCertSet.4.constraint.name=No Constraint
policyset.userCertSet.4.default.class_id=authorityKeyIdentifierExtDefaultImpl
policyset.userCertSet.4.default.name=Authority Key Identifier Default
policyset.userCertSet.5.constraint.class_id=noConstraintImpl
policyset.userCertSet.5.constraint.name=No Constraint
policyset.userCertSet.5.default.class_id=authInfoAccessExtDefaultImpl
policyset.userCertSet.5.default.name=AIA Extension Default
```

Figure 14: 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.

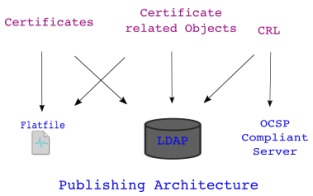

Figure 15: Publishing Architecture

– 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

3.1.7 Exercises

3.2 Key Recovery Authority

3.3 Online Certificate Status Protocol

3.4 Token Key Service & Token Processing System