PKI ADMINISTRATION USING EJBCA AND OPENCA

By Ayesha Ishrath Ghori and Asra Parveen

George Mason University-Fall 2006

Abstract:
For secure exchange of information between two entities, there’s a need for some private information (key) to be shared. Even prior to the intended communication, we need a separate, out-of-band secure communication to occur. This requires an “introducer” between the two entities, which had had no relationship in the past.

The idea of having a key that can be revealed publicly without compromising communications security, is the basis of a PKI. The “PKI” provides for services like encryption, digital signatures, data integrity, key establishment, zero knowledge/ minimum knowledge protocols.

This paper presents an analysis of majorly existing Certificate Authorities, which can help people get to know “what’s in.” People who are new to this entire concept can be able to judge, based on the analysis of secure communication.

This paper provides a comparative analysis between two leading certificate Authorities EJBCA and OpenCA. The OpenCA is a popular Linux based certificate authority and EJBCA is a Enterprise JAVA based CA.

General concepts

Certification Authority (CA):
A CA issues certificates to, and vouches for the Authenticity of entities. The level of trust you can assign to a CA is individual, per CA, and depends on the CA’s policy and practices statement.

Certificate revocation list (CRL):
A Certificate revocation list (CRL) is a list of certificates (more accurately: their serial numbers) which have been revoked, are no longer valid, and should not be relied upon by any system user.

RootCA:
A RootCA has a self-signed certificate and is also called Trusted Root. Verification of other certificates in the PKI ends with the RootCAs self-signed certificate. Since the RootCAs certificate is self-signed it must somehow be configured as a trusted root with all clients in the PKI.

SubCA:
A subordinate CA, or SubCA for short, is a CA whose certificate is signed by another CA that can be another SubCA or a RootCA. Since the SubCAs certificate is signed by another CA, it does not have to be configured as a trusted root. It is part of a certificate chain that ends in the RootCA.

Registration Authority (RA):
An RA is an administrative function that registers entities in the PKI. The RA is trusted to identify and authenticate entities according to the CAs policy. There can be one or more RAs connected to each CA in the PKI.

End-entity:
An end-entity is a user, such as an e-mail client, a web server, a web browser or a VPN-gateway. End-entities are not allowed to issue certificates to other entities; they make up the leaf nodes in the PKI.
Hierarchical PKI:
As part of our project, we have implemented the following hierarchical PKI using two open source PKI Implementations, EJBCA and OpenCA.

Digital Certificates:
A digital certificate is an electronic means of establishing your credentials when doing business or other transactions on the Web. It is issued by a certification authority (CA). User’s distinguished name, User’s public key, User’s Credentials, Serial number, Issuer name, Expiration date, copy of the certificate holder’s public key (used for encrypting and decrypting messages and digital signatures), and the digital signature of the certificate – issuing authority so that a recipient can verify that the certificate is real.

Digital signature which meets ITU (International Telecommunication Union) Telecommunication Standardization (ITU-T) PKIX X.509 version 3 [RFC 2459] standard is generated based on:
- Detailed information about the key holder.
- An expiration date, after which the certificate is placed on the CA’s CRL(Certificate Revocation List).

The operating system stores a certificate locally on the computer or device that requested it or, in the case of a user, on the computer or device that the user used to request it. The storage location is called the certificate store. A certificate store will often have numerous certificates, possibly issued from a number of different certification authorities.

How digital Certificates Are Structured:
For a digital certificate to be useful, it has to be structured in an understandable and reliable way so that the information within the certificate can be easily retrieved and understood. For example, passport follows a similar structure allowing people to easily understand the information in a type of passport that they may never have seen before. In the same way, as long as digital certificates are standardized, they can be read and understood regardless of who issued the certificate.

The EJBCA and OpenCA standards specify that digital certificates used for them conform to the International Telecommunications Union (ITU) X.509 standard. Because both the CA’s rely on an established, recognized standard for the structure of digital certificates, thus increases their acceptance.

The X.509 standard specifies that digital certificates contain standardized information. Specifically, X.509 version 3 certificates contain the following fields:

- **Version number** The version of the X.509 standard to which the certificate conforms.
- **Serial number** A number that uniquely identifies the certificate and is issued by the certification authority.
- **Certificate algorithm identifier** The names of the specific public key algorithms that the certification authority has used to sign the digital certificate.
- **Issuer name** The identity of certification authority who actually issued the certificate.
- **Validity period** The period of time for which a digital certificate is valid and contains both a start date and an expiration date.
- **Subject name** The name of the owner of the digital certificate.
- **Subject public key information** The public key that is associated with the owner of the digital certificate and the specific public key algorithms associated with the public key.
- **Issuer unique identifier** Information that can be used to uniquely identify the issuer of the digital certificate.
- **Subject unique identifier** Information that can be used to uniquely identify the owner of the that certificate.
- **Extensions** Additional information that is related to the use and handling of certificate.
- **Certification authority’s digital signature** The actual digital signature made with the certification authority’s private key using the algorithm specified in the certificate algorithm identifier field.
**EJBCA:**

**Introduction:**

The Enterprise Java Beans (EJB) architecture is a specification developed at Sun Microsystems. It describes a component-based architecture that provides for the development and deployment of distributed applications. The EJB architecture makes enterprise applications scalable, secure, and transactional. EJBCA is a fully functional Certificate Authority built in Java. Based on J2EE technology it constitutes a robust, high performance and component based CA. Both flexible and platform independent, EJBCA can be used standalone or integrated in any J2EE application. Enterprise JavaBeans are components that execute within an “EJB container,” under the supervision of an application server (JBOSS). The application server and EJB container provide system services for EJB’s, such as data persistence, transactions, security, and resource management. The EJB container maintains pools of database connections, as well as pools of EJB instances that can be assigned to clients as needed. The Java 2 Platform, Enterprise Edition (J2EE) is an industry-standard suite of Java APIs for enterprise computing from Sun Microsystems. It includes the Enterprise JavaBeans architecture and a set of related packages that make everything work together.

**EJBCA specific concepts**

**Certificate Profile:**

A certificate profile determines non uses specific content and behavior of certificates. The largest part is extensions and here you decide if a specific extension is present and whether it is critical or not. Some extensions are populated with a value, where it is the same value for all certificates such as CRL Distribution Point. For other extensions only the presence is determined, where the value is user- or cert-specific such as Subject Alternative Name. Here is also determined if these certificates will be published and with which publisher.

**Entity Profiles:**

Entity profiles determine what data can or must be present for users connected with this profile. Some values can also be pre-determined such as the organization, O in the DN. When adding a user in the PKI, the user must be connected with an entity profile. The entity profile specifies one or more certificate profiles used when generating certificates.

**Publishers:**

A publisher stores issued certificates to a central location. EJBCA have implemented support for LDAP and Active Directory but it's also possible to create customized plugins.

**Hard Token Profiles:**

A hard token profile contains information like key length used on cards, certificate profiles that should be used, and templates for the visual graphics printed on the smartcard or on a PIN/PUK letter.

**Hard Token Issuers:**

A hard token issuer represents a physical location running Prime Card where hard tokens can be issued.

**Architecture:**

1) **Data Tier:**

The Data Tier stores the Certificates, CRLs as well as the end users. EJBCA comes with a default database to store the end users. The Certificates are stored in an LDAP repository (Lightweight Directory Access Protocol).

2) **CA Component:**

The Certificate Authority component has the functionality to create Root CAs, Sub CAs, Certificates, CRLs. It also interfaces with the LDAP repository to store the Certificate information.
3) RA Component:
The Registration Authority component has the functionality to create, delete and revoke the Users. It interfaces with the Local database to store the User information.

4) Web Tier:
This is the interface using which the Clients interact with the EJBCA system. It is typically a GUI where different access levels dictate the scope of information a Client can have access to.

5) Client:
The Client is an end-entity. An end-entity is a user, such as an e-mail client, a web server, a web browser or a VPN-gateway. End-entities are not allowed to issue certificates to other entities; rather they make up the leaf nodes in the PKI.

Administration:
The Administration areas include:
- Create and Initialize the Super Administrator
- Creating and Configuring data sources
- Creating Publishers
- Creating Certificate Authorities
- Creating Registration Authorities
- Creating End Entities
- Creating CRLs
- Generating Certificates

Administration Tasks Accomplished:

1) Creating the EJBCA Super Admin Certificate:
4) Administering a Certificate Authority:

5) Publisher for storing the Certificates in the LDAP Format:

6) End Entities Created:

7) Certificates of the End Entity imported into the Browser:
OpenCA

Introduction:

OpenCA is a collaborative effort to create a public–key infrastructure. It is distributed with an Apache–style license. OpenCA started in 1999. It is based on many Open-Source Projects. Among the required software there are OpenLDAP, OpenSSL, Apache Project, Apache mod_ssl.

Architecture:

The basic idea of every X.509 PKI (Public Key Infrastructure) is a strong hierarchical organization. This results in a tree of databases if we try to create a distributed PKI architecture. This hierarchy is the backbone of the trust center.

The primary components are:
- Certification Authority (CA)
- Registration Authority (RA)
- Registration Authority Operator (RA Operator)
- End Users

OpenCA supports all these software components via special web interfaces. The web interfaces can be used to configure and manage the components. The following are the web interfaces to manage the workflow of the PKI.

OpenCA supports the following interfaces:

1. **Node (for node management):** This interface manages the database and handles all the export and import functionalities.

2. **CA:** The CA interface has all the functions which are needed to create certificates and Certificate Revocation Lists (CRLs). The CA also includes all the functions which can be used to change the configuration via a web interface.

3. **RA:** RA is able to handle all kinds of requests. This include things like editing requests, approving requests, creating private keys with smart cards, delete wrong requests and email users.

4. **LDAP:** The LDAP interface was implemented to separate the LDAP management completely from the rest of the software. This is necessary because there are many functions which are really specific for LDAP admins, with only a few users needing these features.

5. **Public:** The Public interface includes all the small things which the users need.
   1. generates CSRs (certificate signing request) for Microsoft Internet Explorer
   2. generates CSRs for Mozilla 1.1+ and Netscape Communicator and Navigator
   3. generates client independent requests and private keys
   4. receives PEM-formatted PKCS #10 requests from servers
   5. enrolls certificates
   6. enrolls CRLs
   7. supports two different methods revocation
   8. search certificates
   9. tests user certificates in browsers (Microsoft Internet Explorer and Netscape Communicator and Navigator 4.7x)

Administration:

1. **Initializing the Certification Authority:** This step involves the generation of a private key and creating a certificate request for the initial CA.

2. **Create the initial administrator:** This step is carried out to create the CA administrator whose role is “CA Operator”. A certificate is created using the certificate request of the previous step.

3. **Create the initial RA Certificate:** Create the certificate to identify the RA administrator whose role is “RA Operator”. After the certificate is generated, the RA will be initialized.

4. **Submit a Certificate Request:** This operation is carried out from the public domain and it goes to the notice of the RA Operator of that domain.

5. **Approve the Certificate:** The RA Operator approves/declines certificate requests after verifying the credentials of the end user.

6. **Issue the Certificate:** The CA Operator issues the certificates for the approved requests of the RA Operator.

7. **Importing the Root Certificate:** This step is carried out to make the Root CA, generated by the PKI as one of the trusted domains in the browser.
**Database Layout:**

The database stores all the needed information about the users' cryptographic objects like Certificate Signing Requests (CSRs), Certificates, Certificate Revocation Requests (CRRs) and Certificate Revocation Lists (CRLs).

**Administration Tasks Accomplished:**

1) **OpenCA CA Administrator Interface:** The first screenshot shows the CA Administrative interface. The first tasks would be to initialize the Certificate Authority, then create the Super Administrator, and then create an RA administrator.

2) **Creating the Certificate Authority:** When the Certificate Authority has to be initialized, a secret key is needed to be generated, then a Certificate Sign Request has to be generated. After that, the Certificate will be created for the Top CA.
3) **Creating the CA Administrator:** The Top CA Administrator will then be created for the initial CA. This involves in collecting the profile and then generating a Certificate. The role of this User would be “CA Operator”.
4) **Creating the RA Administrator:** The next step would be to create an RA. This involves in collecting the profile and then generating a Certificate. The role of this User would be “RA Operator”.

5) **Users requesting for Certificates:** Once the PKI has been initialized, the Users can log onto the Public domain and start requesting for Certificates. Once a Certificate is requested, a Serial number is assigned to that request which can be used to track the status of the request.

6) **Approving the Requests by the RA:** The Certificate requests by the User get queued up at the RA’s work pool and he approves or deletes the requests after verifying the credentials of the User requesting the Certificate.
7) Approving the Requests by the CA: The Certificate requests approved by the RA get queued up at the CAs work pool and he is the one who issues the Certificate.

8) User Certificates: The User can now logon to his public domain, download his certificate and install it in his Web browser.
9) Adding the Certificate Authority to Trusted CA List:

The Certificate Authority created can be added to the browser in the list of CAs to trust. This will then install the Root Certificate in that list. They can be viewed in the browser.

**FUNCTIONALITY:**

**EJBCA:**

EJBCA is a very popular Certificate Authority currently in use, one of the most preferred ones today. The basic features of this CA include the choosing of the algorithm we require. It provides an option between the algorithms—SHA1 with RSA and SHA 256 with RSA with different key sizes. The key size ranges are provided as 1024, 2048, and 4096. However, not all applications support the key size 4096.

The EJBCA provides some prominent features of giving Language preferences during the system configuration. We can also select the type of publisher we want such as LDAP or Active Directory (AD) catalog or a custom-made publisher connector, and its type.

The issuance of certificates is always of the standard X509. There’s also an option provided to choose the type of signing keys—soft or hard. The certificate signing can fall into any of the following categories:

1. Self-signed.
2. External CA.
3. Admin CA.

The number of approvals required ranges between 1 and 4. The CA’s at all the levels of hierarchy have a RSA key length of 2048 bits & a validity of 10 years. The certificate enrollment in the EJBCA provides a user with several choices; user can choose the Cryptographic Service Provider he prefers. And also can choose from the different key sizes provided, 512, 1024 and 2048. It also provides the
user with the option of adding the certificate to the Electronic Identity Card.

**OpenCA:**
The OpenCA is one of the best Certificate Authorities available today for the Linux users. It’s basic features include the provision of the choice of algorithms—DES, DES3 and IDEA. The key size provided here are 2048, 1024 and 4096. The algorithm used for the signature is SHA1 with RSA encryption.

**Level of assurance:** Various levels of assurance are provided while requesting for a certificate as—Test, Rudimentary, basic, medium and high.

**Role Provision:** This provides for the creation of the PKI hierarchy at various levels & in different roles.

**Extensions:** The extensions provided by the OpenCA are of types SKI (Subject Key Identifier) and AKI (Authority Key Identifier).

EJBCA and OpenCA are both robust Open source PKI projects. Still a lot of development is taking place on both of them with a lot of scope for expansion. The following are comparisons between the two architectures.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>EJBCA</th>
<th>OpenCA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ease of Configuration</strong></td>
<td>Very Complex</td>
<td>Complex</td>
</tr>
<tr>
<td><strong>Confidentiality</strong></td>
<td>Offers Confidentiality using encryption</td>
<td>Offers Confidentiality using encryption</td>
</tr>
<tr>
<td><strong>Integrity</strong></td>
<td>Offers Integrity by encryption</td>
<td>Offers Integrity by encryption</td>
</tr>
<tr>
<td><strong>Authentication</strong></td>
<td>Offers Authentication by Digital Signature</td>
<td>Offers Authentication by Digital Signature</td>
</tr>
<tr>
<td><strong>Non Repudiation</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Ability to choose the algorithm to use</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>OCSP</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Ability to choose CSP</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>CRL updates</strong></td>
<td>Automatic</td>
<td>Manual</td>
</tr>
<tr>
<td><strong>Support for smart cards</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>Free</td>
<td>Free</td>
</tr>
<tr>
<td><strong>Extensions</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Platform</strong></td>
<td>Java J2EE</td>
<td>Perl CGI on Unix</td>
</tr>
<tr>
<td><strong>Certificate Repositories</strong></td>
<td>HSQL</td>
<td>MySQL</td>
</tr>
<tr>
<td><strong>LDAP Support</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Modules</strong></td>
<td>EJB</td>
<td>Perl Modules</td>
</tr>
<tr>
<td><strong>Components based</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>Well designed and Scalable</td>
<td>Scalability is difficult with increase of Complexity</td>
</tr>
<tr>
<td><strong>Standalone Component</strong></td>
<td>This is present. The PKI can be administered completely through command line</td>
<td>This is NOT present. Only way of administering the PKI is through Web interfaces.</td>
</tr>
<tr>
<td><strong>Supported Browsers</strong></td>
<td>Multiple</td>
<td>Multiple</td>
</tr>
</tbody>
</table>
Conclusion:
The two Certificate Authorities EJBCA & OpenCA are providers for various clients, individual and business clients. The business users require higher levels of security as they, themselves become the providers for various clients.

The Certificate Authorities studied here can be made to work for both, the beginners and the Established clients, by varying the levels of security. This can be explained by quoting the example of a Certificate Authority — Verisign, which provides certificates for various classes at different levels.

The beginners need to get to know the basics first and then can deal with the complex systems. For them anything which is easily installable matters, so EJBCA is the recommended one. EJBCA is the best for them, as the procedure to generate the certificates is very simple and the downloads required too are freely available. Moreover, the CRL update in this CA is Automatic.

For the more established business users, who want to go for Linux, OpenCA is the best. They have to get computer specialists who would create revocations for the certificates manually, for them.

References:
[2] OpenCA PKI: Open Source, out of the Box
Certificate Authority http://pki.openca.org/
[3] A survey of the market, digital certificate providers:
http://www.verisign.com
[6] Public Key Infrastructure, a wiki engine can be found at:
http://en.wikipedia.org/wiki/Public_key_infrastructure
[7] “Managed Public Key Infrastructure: Securing Your Business Applications” (Whitepaper), Verisign
[8] “Test Drive the thawte Starter PKI Program Console” (White Paper), Thawte
[11] A guide to create your own CA for Tomcat at:
http://users.skynet.be/pascalbotte/art/ca.htm
[13] A survey of the market looking for the free digital certificate providers, as a part of the homework assignment 3.
[14] FAQ’s on PKI, Terms & Definitions, JITC : A Peek can be had at http://jtc.fh.hanover.de/jitc/pki/
[15] "JoelWeise ++, Public Key Infrastructure—Overview,
Sun PSSM Global Security Practice
Sun BluePrints™ OnLine - August 2001
http://www.sun.com/blueprints, Part No.: 816-1279-10
Revision 01, August 2001.

APPENDIX A: GLOSSARY OF TERMS USED:

APPENDIX B: EJBCA Installation Instructions:

Prerequisites:
- JDK 1.4.x or 1.5.x
- Unlimited Strength Jurisdiction Policy Files for the JDK (http://java.sun.com/javase/downloads/index.jsp)
- JBoss 4.0.3 (http://labs.jboss.com/portal)
- Ant >=1.6.5 to build (http://jakarta.apache.org/ant/)
- Apache Myfaces 1.1 JSF libraries (included in JBoss >=4.0.3)

Environment Variables:
- ANT_HOME: Set this variable to the directory where Ant has been installed (C:\apache-ant-1.6.5)
- JAVA_HOME: Set this variable to the directory where the JDK has been installed. (C:\Program Files\Java\jdk1.5.0_09)
- JBOSS_HOME: Set this variable to the directory where the JBoss Application Server is installed. (C:\jboss-4.0.5.GA)
- EJBCA_HOME: Set this variable to the directory where EJBCA has been downloaded and extracted (C:\ejbca)

Installation Procedure:
1. Download the EJBCA (http://www.ejbca.org)
2. Copy ejbca.properties.sample to ejbca.properties. (this file is found in EJBCA_HOME directory)
3. Open a console (terminal) and start Jboss
4. Open a console and type 'ant bootstrap', this will compile, jar, war, ear and deploy it to Jboss. JBoss picks up all the changes and deploys the ear without errors. (run this from EJBCA_HOME)
5. Type 'ant install' it will generate all certificates, keys, etc needed to run the JBoss Application Server. (this file is found in EJBCA_HOME directory)
6. Stop JBoss
7. Type 'ant deploy', this will deploy everything again and configure the servlet container with the keystore file
8. Import the certificate from EJBCA_HOME/p12/superadmin.p12 in the web browser. This is the super administrators certificate used to access the admin GUI. Other administrators with specific privileges can be created later on.

APPENDIX C: OpenCA Installation Guide

Prerequisites:
OpenCA is not a complete monolithic system. It uses several software products from other developers of the Open Source community. The following things are used:
- Apache
- mod_ssl
- OpenSSL
4. OpenLDAP
5. Perl
In addition to installing the above software, the following Perl modules are required:
1. Authen::SASL
2. CGI::Session
3. Convert::ASN1
4. Digest::HMAC
5. Digest::MD5
6. Digest::SHA1
7. Encode::Unicode
8. IO::Socket::SSL
9. IO::stringy
10. MIME::Base64
11. MIME::Lite
12. MIME-tools
13. MailTools
14. Net-Server
15. URI
16. XML::Twig

**Installation:** Do the following steps
```
./configure
make
make test
sudo make install-ca
sudo make install-ra
```
This should install the OpenCA on the system. It can be accessed by pointing the browser to [http://localhost/ca](http://localhost/ca) for the CA component and [http://localhost/ra](http://localhost/ra) for the RA component.