### Public Key Infrastructure (PKI)

Many organizations have a need or a desire to manage their own keys especially in situations where paying an external Certificate Authority (CA) their high fees is impractical. This document will attempt to provide some guidance for those who wish to maintain their own PKI.

In most of the cases online where setting up your own keys are presented they are typically a very generic self-signed kind of case. In some cases you may want to add a level of sophistication to ensure greater security. To do this we need to setup a level of delegation that will enable our organization to create any number of keys and to build a trust chain from those key pairs to an organizational wide key pair that may be installed in the organization’s cryptographic software.

The most common case for these kinds of situations are when the only users of a system are internal users, or when you wish to setup dev or qa environments which are only accessed by known client systems. Additionally, you may wish to setup a level of authentication between the nodes of your system that communicate internally only. For example if your web application wishes to pull authentication or authorization information from a central server (Vicki) and you wish that communication to be strongly authenticated you would probably not wish to use usernames and passwords. You can instead use SSL client authentication if you only had access to a set of client certificates. This use case is a perfect example where a home grown PKI can be used to provide production ready credentials to server components.

### Certificate Signing Key Pair (Root CA Cert)

The first step is to create a certificate signing key pair. This key pair will be a root CA certificate and will be used to sign certificates used for SSL, S/MIME, Code Signing, etc. You may be asking yourself what exactly is it that differentiates a CA certificate from any other certificate type? The answer is the X509v3 Extensions that are applied to the certificate. Whether the certificate is a root certificate (self-signed) or not it must identify itself as a CA certificate. It does that by including an attribute in an extension known as the Basic Constraints as well as a set of allowed usages in a separate extension known as Key Usage.

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command (where validity is the number of days before the certificate will expire):

keytool –genkeypair –keyalg RSA –validity 3650 – keysize 2048

–alias cacert –keystore castore.pfx –storetype pkcs12

–ext bc:c=ca:true –ext ku:c=keyCertSign,cRLSign

–storepass password

1. Fill in the prompts for your organization information. When it asks for your first and last name, enter a descriptive name (e.g. BigNow Root CA)

CN = (First & Last Name) = BidNow Root CA

OU = (Organizational Unit) = Manheim Auctions

O = (Organization) = Cox Automotive

L = (City or Locality) = Atlanta

S = (State or Province) = GA

C = (Country Code) = US

1. A CA certificate usually has a much longer validity period then the keys it is used to sign. In the above example we define roughly 10 year period (minus of course the leap days).
2. The –ext option allows you to supply extensions. In this case we add a Basic Constraint (bc) option which is marked as critical and defines this as a CA certificate. We also define a Key usage (ku) extension which is also marked as critical and which defines certificate signing and CRL signing usages.
3. This key pair should be guarded more so than any other as it can be used to create arbitrary key pairs that will be trusted. It is recommended to use a password stronger than the above example.

### Exporting the Root Certificate

Once you have a root certificate you need to export it and import it into the browsers, Java VMs, and other cryptography software that will need to establish a basis of trust in the SSL, Code Signing, and S/MIME keys you will create later.

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command:

keytool –exportcert –alias cacert –storepass password

–keystore castore.pfx –storetype pkcs12

–file cacert.crt

1. You may now import the Root CA certificate into your software’s trust database. Examples of several software (Browser) packages is provided in the README associated with this distribution.

### Creating a Key Pair for SSL Server Authentication

In the vast majority of cases we will be generating SSL Server Certificates for use in non-production environments or for internal use only. A SSL Server certificate is similar to a CA cert above except that its associated extensions are very different.

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command (where validity is the number of days before the certificate will expire):

keytool –genkeypair –keyalg RSA –validity 365 – keysize 2048

–alias prod-ssl –keystore keystore.pfx –storetype pkcs12

–ext san=dns:bidnow.man.com,dns:bidnow.man.org

–storepass password

1. Fill in the prompts for your organization information. When it asks for your first and last name, enter the fully qualified domain name of the server that users will be using to connect to your application (e.g. bidnow.manheim.man-ba.com)

CN = (First & Last Name) = bidnow.man.com

OU = (Organizational Unit) = Manheim Auctions

O = (Organization) = Cox Automotive

L = (City or Locality) = Atlanta

S = (State or Province) = GA

C = (Country Code) = US

1. The Subject Alternative Name (san) extension defines two different DNS names this certificate will be used to authenticate. You may also specify IP addresses in the form ip:192.168.1.1 for example. Each name is simply appended to the end separated by a comma. One of these names should match the one used for CN in step 3.
2. Since this key is used for SSL its validity period should be much smaller. In the above example we use 360 days or 1 year. You can increase this but it should not exceed 3 years.

### Creating a Certificate Signing Request for an SSL Server

In the previous step we created a key pair identifying a particular server. We included in the creation of the key pair a single X509v3 extension which identified the aliases (CNAMEs) that the server might go by. We now need to create a signing request which indicates our desired usage:

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command:

keytool –certreq –alias prod-ssl –storepass password

–keystore keystore.pfx –storetype pkcs12

–ext ku:c=keyEncipherment

–ext bc:c=ca:false –ext eku=serverAuth

-sigalg SHA256withRSA -file prod-ssl.csr

1. The Basic Constraint (bc) extension identifies this key as NOT a CA key.
2. The Key Usage (ku) extension allows this key to be used to do key encryption which is necessary for an SSL server.
3. The extended Key usage (eku) defines the key as usable for SSL Server Authentication.
4. The Subject Alternative Name (san) defined when we created the key will be included in the certificate signing request as well without the need for us to indicate that here.
5. The prod-ssl.csr file is now ready to be sent to a CA for signing or signed with your CA key pair if you are running your own certificate authority.

### Creating a Key Pair for SSL Client Authentication

In some cases we may want a client user or system to authenticate itself more securely than is possible with simple usernames and passwords. To do that we will generate a SSL Client Authentication Certificate which will securely identify our subject:

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command (where validity is the number of days before the certificate will expire):

keytool –genkeypair –keyalg RSA –validity 365 – keysize 2048

–keystore keystore.pfx –storetype pkcs12

–storepass password –alias node-client

1. Fill in the prompts for your organization information. When it asks for your first and last name, enter the username your server will use to authenticate the client:

CN = (First & Last Name) = node-server

OU = (Organizational Unit) = Manheim Auctions

O = (Organization) = Cox Automotive

L = (City or Locality) = Atlanta

S = (State or Province) = GA

C = (Country Code) = US

1. Since this key is used for client authentication its validity period should be much smaller. In the above example we use 360 days or 1 year. You can increase this but it should not exceed 3 years.

### Creating a Certificate Signing Request for a SSL Client

In the previous step we created a key pair identifying a particular user. We now need to create a signing request which indicates our desired usage:

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command:

keytool –certreq –alias node-client –storepass password

–keystore keystore.pfx –storetype pkcs12

–ext ku:c=digitalSignature

–ext bc:c=ca:false –ext eku=clientAuth

-sigalg SHA256withRSA -file node-client.csr

1. The Basic Constraint (bc) extension identifies this key as NOT a CA key.
2. The Key Usage (ku) extension allows this key to be used to do digital signatures which is required to prove identity in an authentication challenge.
3. The extended Key usage (eku) defines the key as usable for SSL Client Authentication.
4. There are no subject alternative names associated with a SSL Client Auth certificate. The subject name must match what the server is expecting.
5. The node-client.csr file is now ready to be sent to a CA for signing or signed with your CA key pair if you are running your own certificate authority.

### Creating a Key Pair for S/MIME

S/MIME is a form of secure email communication. If two parties have S/MIME enabled key pairs and certificates they can encrypt communication over SMTP and apply strong authentication to incoming messages. To do this both parties must create S/MIME key pairs:

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command (where validity is the number of days before the certificate will expire):

keytool –genkeypair –keyalg RSA –validity 365 – keysize 2048

–keystore keystore.pfx –storetype pkcs12

–storepass password –alias cfloersch

-ext san=email:cfloersch@manheim.com,email:cfloersch@ove.com

1. Fill in the prompts for your organization information:

CN = (First & Last Name) = Chris Floersch

OU = (Organizational Unit) = Manheim Auctions

O = (Organization) = Cox Automotive

L = (City or Locality) = Atlanta

S = (State or Province) = GA

C = (Country Code) = US

1. Since this key will be used for encrypting and authenticating email we need to include the subject alternative names extension which identifies each of the email addresses from which we can send/receive email. These names are used by the client software to associate an incoming/outgoing email with the appropriate key pair.
2. Since this key is used for email encryption and authentication its validity period should be much smaller. In the above example we use 360 days or 1 year. You can increase this but it should not exceed 3 years.

### Creating a Certificate Signing Request for S/MIME

In the previous step we created a key pair identifying a particular person and their associated email addresses. We now need to create a signing request which indicates our desired usage:

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command:

keytool –certreq –alias cfloersch –storepass password

–keystore keystore.pfx –storetype pkcs12

–ext ku:c=digitalSignature ,keyEncipherment

–ext bc:c=ca:false –ext eku=emailProtection

-sigalg SHA256withRSA -file cfloersch.csr

1. The Basic Constraint (bc) extension identifies this key as NOT a CA key.
2. The Key Usage (ku) extension allows this key to be used to do digital signatures and key encryption which are required for complete S/MIME functionality.
3. The extended Key usage (eku) defines the key as usable for E-Mail protection.
4. The Subject Alternative Names (san) extension identifies all of the email addresses that belong to the individual identified as the subject of this certificate.
5. The cfloersch.csr file is now ready to be sent to a CA for signing or signed with your CA key pair if you are running your own certificate authority.

### Creating a Key Pair for Code Signing

Code Signing is becoming more and more common as user begin to challenge the author of the software they allow to run on their machine. In most cases a code signing certificate belongs to an organization that publishes software but it may also belong to an individual in rare cases. To create a key pair suitable for use in code signing:

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command (where validity is the number of days before the certificate will expire):

keytool –genkeypair –keyalg RSA –validity 365 – keysize 2048

–keystore keystore.pfx –storetype pkcs12

–storepass password –alias mandev

1. Fill in the prompts for your organization information:

CN = (First & Last Name) = Manheim Software

OU = (Organizational Unit) = Manheim Auctions

O = (Organization) = Cox Automotive

L = (City or Locality) = Atlanta

S = (State or Province) = GA

C = (Country Code) = US

1. Since this key is used for email encryption and authentication its validity period should be much smaller. In the above example we use 360 days or 1 year. You can increase this but it should not exceed 3 years.

### Creating a Certificate Signing Request for Code Signing

In the previous step we created a key pair identifying a particular organization. We now need to create a signing request which indicates our desired usage:

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command:

keytool –certreq –alias mandev –storepass password

–keystore keystore.pfx –storetype pkcs12

–ext ku:c=digitalSignature

–ext bc:c=ca:false –ext eku=codeSigning

-sigalg SHA256withRSA -file mandev.csr

1. The Basic Constraint (bc) extension identifies this key as NOT a CA key.
2. The Key Usage (ku) extension allows this key to be used to do digital signatures which is required for signing code.
3. The extended Key usage (eku) defines the key as usable for Code Signing.
4. This certificate does not need additional subject alternative names.
5. The mandev.csr file is now ready to be sent to a CA for signing or signed with your CA key pair if you are running your own certificate authority.

### Validating a Certificate Signing Request

As a Certificate Authority it is your responsibility to verify the certificate signing requests before actually signing them. This will usually include verifying the public key included in the CSR is in fact the public key the subject submitted to you and it was not changed in transit. It also includes verifying the subject is who they claim to be and that they own the email/dns/ip/name identified in the certificate signing request. Finally, it is also your responsibility to check the certificate extensions associated with the request and remove ones that are inappropriate or add ones that may be missing.

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command:

keytool –printcertreq -v –file prod-ssl.csr

1. The above command will output the certificate signing request to the console and allow you to examine the contents. It will include the subject name which you should verify, both MD5 and SHA1 hashes of the contained public key, and the set of extensions they are requesting.
2. The extensions will often indicate the intended use the requester has in mind for the certificate. If those extensions are not present it is the CA’s responsibility to ask the requester through an out of band method what its intended use is and to supply those extensions when signing the certificate.

|  |  |
| --- | --- |
| SSL Server Auth | Basic Constraints: CA=false  Key Usage: Key Encipherment  Extended Key Usage: Server Auth  Subject Alternative Names: A list of IP or DNS entries |
| SSL Client Auth | Basic Constraints: CA=false  Key Usage: Digital Signature  Extended Key Usage: Client Auth |
| S/MIME | Basic Constraints: CA=false  Key Usage: Digital Signature, Key Encipherment  Extended Key Usage: Email Protection  Subject Alternative Names: A list of EMAIL addresses |
| Code Signing | Basic Constraints: CA=false  Key Usage: Digital Signature  Extended Key Usage: Code Signing |
| Certificate Signing (CA) | Basic Constraints: CA=true  Key Usage: Key Cert Sign, CRL Sign |

1. It is possible that a certificate may be used for more than one purpose. In that case the request will likely include a combination of the above. The most typical example of that is a combination of Client and Server SSL auth or a certificate used by a user for both SSL Client Auth and S/MIME.
2. Certificate signing keys should not be used for anything other than certificate and CRL signing.

### Signing a Certificate

Once we have verified the information contained within a certificate signing request we can now sign the certificate and generate a response to the requester:

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command:

keytool –gencert -infile prod-ssl.csr -outfile prod-ssl.p7b

-alias cacert -keystore castore.pfx -storetype pkcs12

-sigalg SHA256withRSA -ext honored=all

1. The special case use of the –ext honored=all above indicates that we will honor all of the X509v3 extensions included in the certificate signing request.
2. It is possible to honor a subset of the request extensions while excluding others and potentially adding yet others:

|  |  |
| --- | --- |
| -ext honored=all | Accept all of the extensions from the request as they are. |
| -ext honored=all,-bc | Honor all the request extensions except the basic constraint (bc) |
| -ext honored=san | Honor only the subject alternative name (san) and exclude all the others. |
| -ext honored=san,ku | Honor only subject alternative name (san) and key usage (ku). Exclude all others. |
| -ext honored=all  -ext bc:c=ca:false | Honor all requested extensions but override/add the basic constraint (bc) with the specified basic constraint. |

1. The prod-ssl.p7b file is now ready to be sent back to the signing requester.
2. The p7b file is a binary encoded PKCS#7 file which contains the requester’s signed certificate and the certificate used to sign it.

### Importing the Signed Certificate

Once the Certificate Authority has delivered your signed certificate, you must import it back into the keystore that contains your private key.

1. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
2. Run the following command:

keytool –importcert –alias prod-ssl –storepass password

–keystore keystore.pfx –storetype pkcs12 -trustcacerts

–file prod-ssl.p7b

1. You now have a key store which contains your private key and its associated and signed public certificate. You may export the keys or certificate or certificate chain as per your target platform needs. Instructions for this are provided in the associated README documentation.