### Digital Signature

A Digital Signature is a cryptographic primitive that is used to validate the identity of an item’s author or origin. It is a signature in the sense that it is a unique, identifying mark created utilizing encryption and a pair of keys – one of which is used to decrypt something encrypted by the other. This allows anyone in possession of a public key to verify that an item was encrypted by the corresponding private key that is typically kept secure to an author or origin. If the content is altered in transit the decryption will fail indicating that it is not authentic. If an attacker attempts to masquerade as a trusted entity the decryption will fail so long as the attacker can’t deduce or obtain the private key associated with the trusted public key in your possession. For this to work you must trust the public key you are in possession of actually belongs to an identifiable author or origin server.

### Digital Certificate

A Digital Certificate is a cryptographic item that encapsulates the author’s or origin’s public key and identifies that author or origin, a validity period, and some set of extensions which identify in what way the public key may be used. It can be either self-signed in which case you must establish trust in the certificate through an external process or it can be digitally signed by a trusted third party typically referred to as a Certificate Authority. The term public key and public certificate are often used interchangeably.

### Certificate Authority

A Certificate Authority is a trusted organization that is tasked with verifying through back channel means the author or origin of a public key and in turn digitally signing the associated digital certificate so that it may be verified as belonged to that author or origin. The certificate authority will sign the digital certificate using the private key associated with what is known as a root certificate that must be pre-installed in the software that will be doing the verification process. The root certificate or one of its intermediaries must be in the software’s trusted certificates database.

Think of a certificate authority in much the same way as you think of the DMV which issues drivers licenses and other identification documents. They are given a request to identify an entity, they in turn verify the identity of that entity, and then issues a document that is difficult to forge identifying that entity. In the case of digital certificates it uses a cryptographically strong process that is virtually impossible to forge.

### Types of Certificates

There are many types of certificates and their names are most often associated with the role they are used in. For example you will typically hear people request a SSL certificate. This is a simple certificate like any other that uses the server’s fully qualified DNS name as its CN (Common Name) and for which will be granted digital signature usage. The common name represents in this case the origin identifier associated with the certificate which is in turn associated with the public key. A code signing certificate also known as a software publisher’s certificate is exactly the same except the CN (Common Name) identifies the author or company associated with the software. A code signing key is granted an extended usage for the specific use case of code signing. Finally, there are certificate authority certificates also known as trusted or root certificates. These certificates are always self-signed by the certificate authority organization and define the CA role within its key usage extensions. Trust in these keys is always established through back channel means but are typically pre-configured in most cryptographic software like the browers.

### Certificate Chains

### A Certificate Chain is a collection of certificates starting with the certificate to be verified and leading to one of the trusted root certificates. In some cases it may be only two certificates in length. The origin or author’s certificate and the trusted root certificate. In other cases there may be some intermediary certificates. For example most CA (Certificate Authorities) will have a root certificate they then use to certify a certificate used for code signing requests, another used for SSL certificate requests, and finally another used to sign certificates requests used in a CA role within a large organization. These intermediaries allow a delegation model to be established.

### PKI

PKI is an acronym for Public Key Infrastructure and it represents the tools, keys, and process used to establish trust in public keys via the creation of certificates and to destroy trust in compromised keys through the creation of certificate revocation lists (CRLs). Organizations that act as certificate authorities will establish a public key infrastructure to operate their business and to manage that role. Some large companies manage their individual user’s keys and server keys themselves using a PKI and certificate authority keys issued by a larger third party organization such as Verisign.

### Keystore

A Keystore is a file or hardware device which stores private keys and their associated public certificates or a collection of trusted third party public certificates. The most common keystore format is PKCS#12 which is protected by a form of password based encryption so as to prevent unauthorized access to the private key information. Java also provides a proprietary keystore known as the Java Key Store or JKS. It also offers encrypted access control not only at the file level but also on the individual key level making it a better alternative when you want to maintain more than one key pair. However, it is not an open standard and works only with java applications.

### PKCS

PKCS is an acronym for Public Key Cryptography Standard. There are a number of defined standards associated with public key cryptography including file formats, interchange formats, and password based encryption techniques. The most common include PKCS#1 which defines RSA based encryption, PKCS#5 which defines symmetric key encryption and password based encryption, PKCS#7 which defines a general interchange format, PKCS#10 which defines the certificate signing request format, and PKCS#12 which defines a set of standards for securely storing private/public keys and certificates.

### Public Key Process

The basic public key process begins by determining a need. Whether it is software signing, encrypted email, server communication to be protected by SSL, or you want to setup your own PKI we begin by creating a key pair. When the key pair is initially created we often associate the organization, the individual, or the server with that key pair and create what is known as a self-signed certificate.

Once we have generated the key pair we then generate what is known as a CSR (certificate signing request) which is a specific file format designed for inter organizational exchange. The CSR includes the public key we generated in the previous step as well as our self-signed certificate which identifies the author or origin. It does not contain the private key which must remain private in the author’s or origin’s possession.

Once we have generated the CSR we submit that CSR to a certificate authority. They in turn verify the identity associated with the request and once done they sign the certificate with their own keys overriding the self-signed nature of the certificate. They will respond with a signed certificate and certification chain.

Once we have obtained our signed certificate and its certification chain we then import that chain back into our keystore that holds our original self-signed certificate and private key overriding the self-signed certificate.

Once we have imported the trusted certificate we can then export the certificate, the certificate chain, or the private key to various formats used by browsers, email clients, http servers, or code signing systems.

Private keys should always be stored in cryptographically secure formats or should be handled with extreme care to ensure they remain private. The security of a public key exchange is only as strong as the trust in the public certificates and the guarantee that the only one with access to the private key is the origin/author identified by the public key. If the private key is compromised the original certificate authority may be directed to issue a CR (certificate revocation) which is added to a list of revocations known as a CRL. Clients verifying a signature always examine the revocation lists to ensure the certificate has not been revoked as part of the verification process.

If we chose to bypass the CSR process we must install the self-signed certificate into all the software that will be verifying public key certificates we use. For example if we are creating a self-signed key to use in the SSL capacity we would need to install that certificate into every browser that would land on a page served by that server. Failure to do so would result in the browser warning the user that the secure connection cannot be authenticated.

Transferring self-signed certificates between parties using unsecured transports is susceptible to man in the middle attacks. Certificate fingerprints may be used to verify the authenticity of these certificates.

### PEM Format

The PEM format is an encoding format used to store cryptographic components in ASCII. PEM certificates usually have extensions such as **.pem, .crt, .key, .p7b, and .cer**. The binary cryptographic contents are encoded using BASE64 encoding and wrapped in "-----BEGIN XXXX-----" and "-----END XXXX-----" statements. Server certificates, certificate chains, and private keys can all be stored in the PEM format.

**Apache, Amazon, and other similar servers** use PEM format certificates. Several PEM certificates, and even the private key, can be included in one file, one below the other, but most platforms, such as Apache, expect the certificates and private key to be in separate files.

Private keys encoded in PEM format are not protected by encryption and must be protected from unauthorized access.

### DER Format

The DER format is simply a binary form of a cryptographic component as opposed to the ASCII PEM format. It sometimes has a file extension of **.der** but it often has a file extension of **.cer, .spc, .crt, .key, or .p7b** so the only way to tell if a file is encoded using DER or PEM is to open it in a text editor and look for the BEGIN/END statements that indicate the PEM format. All types of certificates and private keys can be encoded in DER format.

Private keys encoded in DER format are not protected by encryption and must be protected from unauthorized access.

### PKCS#12 Format

PKCS#12 files are cryptographically secure key stores capable of storing the public certificate, any intermediate certificates, and the private key in one file. PKCS#12 files are binary files and usually have extensions such as **.pfx and .p12**. PKCS#12 files encrypt the private key using a form of password based encryption, protecting it from unauthorized access.

You will in most cases manage your public/private key pairs using this format to keep them secure. Exporting the key or certificates to various formats for use with various tools. This format is standardized and supported by most software and operating systems.

### PKCS#7 Format

The PKCS#7 format is a standard data structure used to store a collection of items, typically a certificate chain. The format may be persisted to a file in binary form encoded in the DER format or in ASCII form encoded in the PEM format. It sometimes has the file extension **.p7b, .p7c, or .spc.** Several platforms utilize PKCS#7 files for certificate chain storage including **Microsoft Windows and Java Tomcat**.

**Generate a Self-Signed Certificate**

Whether you are using a certificate authority or not the first step is to generate a key pair that will be used for code signing, SSL, or email. Generate a self-signed key pair:

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 –alias mykey

–keystore keystore.pfx –storetype pkcs12

–storepass password –validity 365 – keysize 2048

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

CN = (First & Last Name) = [bidnow.manheim.man-ba.com](http://www.manheim.com)

OU = (Organizational Unit) = Manheim Auctions

O = (Organization) = Manheim Remarketing, Inc.

L = (City or Locality) = Atlanta

S = (State or Province) = GA

C = (Country Code) = US

The CN, also known as Common Name, which is referred to as First and Last Name in keytool, must match exactly the URL hostname the user types into their browser. Wildcard names are possible but should be avoided when using self-signed certificates.

**Printing the Details of a Key Pair in a Keystore**

Periodically it may be important to output the details of a key pair stored in a keystore to verify a certificate chain was imported or to identify the keys contained within. Each certificate output will include an MD5 and various SHA hashes which can be used as an alternative verification method as well as the validity period of the certificate:

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 –list –alias mykey –keystore keystore.pfx

–storetype pkcs12 –storepass password

1. The abbreviated details of the key pair identified by the specified alias. This will not print the entire certificate chain but only the public certificate associated with the private key.
2. If you include the –v option the entire certificate chain will be printed including the public certificate.
3. Excluding the –alias will dump the entire contents of the keystore which may include more than one key pair.

**Exporting a Certificate to PEM Format**

Amazon Elastic Load Balancing requires the private key, public certificate, and public certificate chain to be in PEM (Privacy Enhanced Mail) format as defined in RFC 1421:

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 mykey –keystore keystore.pfx

–storetype pkcs12 –storepass password –rfc

-file cert.crt

1. The certificate will be written to the specified file. It may now be uploaded to Amazon’s elastic load balancer or F5’s BigIP. For self-signed certificates there is no certificate chain.
2. Excluding the –rfc option will result in a binary DER formatted file being created instead of the PEM format above.

**Exporting a Private Key to PEM Format**

Amazon Elastic Load Balancing requires the private key, public certificate, and public certificate chain to be in PEM (Privacy Enhanced Mail) format as defined in RFC 1421:

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 –exportprikey –alias mykey –keystore keystore.pfx

–storetype pkcs12 –storepass password –rfc

-file my\_private\_key.key

1. The private key will be written to the specified file. It may now be uploaded to Amazon’s elastic load balancer or F5’s BigIP.
2. It is important to note that the private key is very vulnerable in this unencrypted state. Keeping this file private is critically important to the security of the system.
3. Excluding the –rfc option will result in a binary DER formatted file being created instead of the PEM format above.

**Generating a Certificate Signing Request**

Once you have generated a self-signed key pair, for production use it is necessary to send a certificate signing request to a trusted certificate authority to get a real certificate:

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 mykey –keystore keystore.pfx

–storetype pkcs12 –storepass password

-file bidnow.csr

1. The certificate signing request will be written to the specified file. The file may now be submitted to a certificate authority to be signed.
2. Certificate Signing Requests are always created in PEM format.

**Viewing a Certificate Signing Request**

Once you have created a certificate signing request you might want to view it to verify its contents:

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 -file bidnow.csr

1. The certificate signing request will be dumped to the console.

**Viewing the CA Signed Certificate**

Once you have submitted a certificate signing request to a certificate authority such as GoDaddy you will receive back a signed certificate file of some sort. To view the contents of the certificate file:

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 –printcert -v -file bidnow.crt

1. The file from the certificate authority may have any number of file extensions. For code signing certificates it is typically spc. It may also be crt, der or pem.
2. The entire certificate chain (if one exists) will be printed to the screen for your evaluation. You can compare key fingerprints with those in your keystore and identify if the file contains just the signed certificate or both the signed certificate and the certification chain.
3. Excluding the –v option will result in only the user certificate being printed to screen.

**Importing a Signed Certificate**

Once you have submitted a certificate signing request to a certificate authority such as GoDaddy you will receive back a signed certificate file and corresponding certificate chain. To import that signed certificate into your key store:

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 mykey –keystore keystore.pfx

–storetype pkcs12 –storepass password -trustcacerts

-file bidnow.crt

1. The file from the certificate authority may have any number of file extensions. For code signing certificates it is typically spc. It may also be crt, der or pem.
2. In many cases the certificate you receive back from a Certificate Authority (CA) will include your signed certificate as well as a chain of trust to a root certificate that is available in the various browsers. This process will import the entire chain into the keystore. This is the most desirable import format.
3. To utilize the newly signed certificate on Amazon’s Elastic Load Balancer you will need to export the certificate to PEM format and upload it to Amazon. You will also need to upload the certification chain in PEM format as well.

**Exporting a Certificate Chain to PEM Format**

Amazon Elastic Load Balancing requires the certification chain, if one exists, to be in PEM (Privacy Enhanced Mail) format as defined in RFC 1421. They add the additional stipulation that the chain not include the User certificate which they expect in another file:

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 –exportpem –alias mykey –keystore keystore.pfx

–storetype pkcs12 –storepass password

-file cert\_chain.pem

1. The certification chain will be written to the specified file if one exists for the specified alias. It may now be uploaded to Amazon’s elastic load balancer or F5’s BigIP.
2. Certificate chains do not exist for self-signed certificates.
3. To include the User level certificate in the output use the –v option.

**Viewing the Certificate Installed on a SSL Server**

Once you have installed your certificate in the AWS load balancer or BigIP you can verify it with the following command:

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 –printcert -v –sslserver bidnow.manheim.com

1. The tool will connect to <https://bidnow.manheim.com/> and retrieve and display the installed certificate details. This can be useful when transitioning from a self-signed certificate to a CA signed certificate to verify the update.

**Exporting a Certificate Chain to PKCS7 Format**

There are occasions when a certification chain needs to be exported to the PKCS7 format:

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 –exportchain –alias mykey –keystore keystore.pfx

–storetype pkcs12 –storepass password

-file cert\_chain.p7b

1. The above options will encode the cert chain in binary format and will exclude the User certificate from the path.
2. To include the User certificate include –v option.
3. To encode the PKCS7 structure in PEM format include the –rfc option.
4. Certificate chains do not exist for self-signed certificates.

**Changing a Keystore Password**

There may be a need to change the password on your keystore so that it might be shared with others or because the password has become compromised:

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 –storepasswd –alias mykey –keystore keystore.pfx

–storetype pkcs12 –storepass password

-new new-password

1. If you exclude either –storepass or –new it will prompt you to enter your passwords.
2. The new password must be at least 6 characters long.

**Deleting an Alias from a Keystore**

If you have installed a self-signed key into your trusted certs keystore or you simply want to get rid of an old expired key pair you may delete it:

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 –delete –alias mykey –keystore keystore.pfx

–storetype pkcs12 –storepass password

1. If you exclude either –storepass it will prompt you to enter it.
2. The storetype should be omitted and the keystore file changed to <JAVA\_HOME>/jre/lib/security/cacerts for Java’s trusted certs.

**Importing a Certificate into Trusted Certs (Java)**

Java will complain about SSL self-signed certificates. To overcome this we need to import our self-signed certificate into java’s trusted certificates list:

1. Backup your existing cacerts file located in <JAVA\_HOME>/jre/lib/security
2. Open the command console on whatever operating system you are using and navigate to the directory where keytool is installed.
3. Run the following command:

keytool –importcert –alias selfsigned

–keystore %JAVA\_HOME%\jre\lib\security\cacerts

–trustcacerts –file self\_signed.cer

-file cert\_chain.p7b

1. The self-signed certificate file above is one exported in PEM or DER format from your keystore after having created the key pair.
2. The above assumes a windows platform. Use $JAVA\_HOME/jre/lib/security/cacerts instead on other platforms.

**Importing a Certificate into Trusted Certs (Chrome/IE)**

Browsers will complain about SSL self-signed certificates. To overcome this we need to import our self-signed certificate into the browser’s trusted certificates list:

1. If your using Chrome skip to step 4
2. Open Internet Explorer and click the “Options” > “Internet Options” menu. The Internet Options dialog box will be displayed.
3. Click the “Content” tab and the “Certificates” button. The Windows Certificates dialog box is displayed. Skip to step 7.
4. For Chrome you must click the menu icon and then select “Settings” from the popup menu.
5. On the Chrome Settings page click the “Show advanced settings” link at the bottom of the page.
6. Scroll down to the HTTPS/SSL section of the page and click the “Manage certificates…” button. The Windows Certificates dialog box is displayed.
7. Click the “Trusted Root Certification Authorities” tab, and click the “Import…” button. The Certificate Import Wizard will be displayed.
8. Click the “Next” button. The File to Import step is displayed.
9. Use the “Browse” button to find and select the certificate previously exported from keytool. Then click the “Next” button. The Certificate Store step is displayed.
10. Keep the default certificate store selection: “Trusted Root Certificate Authorities”, and click the “Next” button. The confirmation step is displayed.
11. Click the “Finish” button. A warning message will be displayed.
12. Click the “Yes” button. Your self-signed certificate will now be trusted by both Chrome and Internet Explorer.

**Importing a Certificate into Trusted Certs (Firefox)**

Browsers will complain about SSL self-signed certificates. To overcome this we need to import our self-signed certificate into the browser’s trusted certificates list:

1. Click the menu icon and select “Options” from the popup menu.
2. Click the “Advanced” icon on the options dialog and then select the “Certificates” tab.
3. Click the “View Certificates” button on the “Certificates” tab. The Certificate Manager dialog will be displayed.
4. Select the “Authorities” tab and click the “Import…” button.
5. Find and select the certificate previously exported from keytool. Then click the “Open” button. The “Downloading Certificate” dialog is displayed with a warning and some options.
6. Check all the checkboxes and click the “OK” button. Your certificate is now imported into Firefox as a trusted certificate authority.