**PKI Infra Structures**

**PKI (It is a standard for digital certificates)**

-Related to the idea of asymmetric key cryptography (It includes public key and private key)

-Includes message digests,digital signatures and encryption services.

we can achieve interigity with message digests.

authentication and non repudiation with digital signatures

confidentiality with encryption.

-To enable all the services Digital Certificates are required.

**Digital Certificates** (Standard follow is PKI)

-Small file on computer/electronic device

-File extension is generally (.cer). It must be issued by trusted entity or trusted party

-DC establishes the relation between user and the public key.

**Sample Digital Certificate :**

User name - xyz

public key - <12$aa#>

Serial No - 12345

Other info - email-id

valid from-31 jan 2006

valid to - 31 jab 2016

issuer name - VeriSign/Entrust

**Fields of digital certificate :**

x.509 defines the standard of digital certificate

version :x.509

Signature algo identifier

Issuer U/ID

CA Digital Signature - It is used during digital certificate verification

**What is cerificate Authority (CA)**

- CA are trusted agency that can issue digital certificate

**How CA works -**

when user issue DC to CA, Ca verifies the user and return information.

**Digital certificate creation**

There are several parties/organizations are involved with the process of digital certificate creation.

**(1) Registration Authority (RA)**

- Immediate entity between end users and CA

- Assists the CA in its daily works

**Services of RA :**

- Accepting and verifying registration information about new users

-It can generate keys on behalf of user

- Keys backup/recovery

-certificate revocation

**Certificate Creation Steps:**

**-Key generation -**

(Create private/public keys pairs. The user can create self or by RA)

Private key is kept secret

public key is sent to RA with other information.

**-Registration -**

RA registers the user

**-Verification**

RA verifies user's credentials

RA verifies that users who sends public key contains corresponding private key.

**-certificate creation**

RA pass all details to CA.

CA creates the digital certificate and send back Digital certificate to user

Keeps a copy DC in a directory.

**-----------------------------------------------------**

**Certificate Revocation :**

Withdrawing of digital certificate

**Common reasons -**

- Key (Private key) is stolen

- Mistakes in information

-User leaves the job and organization issues revocation request.

**Digital Certificate Revocation checks :**

1- Offline Checks

maintaining certificate List(CRL)

2- Online Checks

-Online certificate validation protocal status(OCSP)

- Simple (SCVP)

**Symmetric Encryption**

In symmetric encryption, we use a key (obviously kept secret) to encrypt the data that needs to be kept secret. To get back the original text, we need to decrypt it again using the same key. As the same key is used for encryption and decryption, this method is called symmetric encryption/decryption.

I am giving example in java :

For Example

**package** encryption;

**import** java.math.BigInteger;

**import** java.security.InvalidKeyException;

**import** java.security.KeyPair;

**import** java.security.KeyPairGenerator;

**import** java.security.NoSuchAlgorithmException;

**import** java.security.PrivateKey;

**import** java.security.PublicKey;

**import** java.util.Base64;

**import** javax.crypto.BadPaddingException;

**import** javax.crypto.Cipher;

**import** javax.crypto.IllegalBlockSizeException;

**import** javax.crypto.KeyGenerator;

**import** javax.crypto.NoSuchPaddingException;

**import** javax.crypto.SecretKey;

**import** javax.crypto.spec.SecretKeySpec;

**public** **class** EncDecSymmetric

{

// Symmetric encryption algorithms supported - AES, RC4, DES

**protected** **static** String *DEFAULT\_ENCRYPTION\_ALGORITHM* = "AES";

**protected** **static** **int** *DEFAULT\_ENCRYPTION\_KEY\_LENGTH* = 256;

**protected** SecretKey mSecretKey;

**protected** String mEncryptionAlgorithm, mKeyEncryptionAlgorithm, mTransformation;

**protected** **int** mEncryptionKeyLength, mKeyEncryptionKeyLength;

**protected** PublicKey mPublicKey;

**protected** PrivateKey mPrivateKey;

EncDecSymmetric()

{

mSecretKey = **null**;

mEncryptionAlgorithm = EncDecSymmetric.*DEFAULT\_ENCRYPTION\_ALGORITHM*;

mEncryptionKeyLength = EncDecSymmetric.*DEFAULT\_ENCRYPTION\_KEY\_LENGTH*;

}

**public** **static** BigInteger keyToNumber(**byte**[] byteArray)

{

**return** **new** BigInteger(1, byteArray);

}

**public** SecretKey getSecretKey()

{

**return** mSecretKey;

}

**public** **byte**[] getSecretKeyAsByteArray()

{

**return** mSecretKey.getEncoded();

}

**public** String getEncodedPublicKey()

{

String encodedKey = Base64.*getEncoder*().encodeToString(mPublicKey.getEncoded());

**return** encodedKey;

}

// get base64 encoded version of the key

**public** String getEncodedSecretKey()

{

String encodedKey = Base64.*getEncoder*().encodeToString(mSecretKey.getEncoded());

**return** encodedKey;

}

**public** **void** generateSymmetricKey()

{

KeyGenerator generator;

**try** {

generator = KeyGenerator.*getInstance*(mEncryptionAlgorithm);

generator.init(mEncryptionKeyLength);

mSecretKey = generator.generateKey();

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

}

}

**public** **byte**[] encryptText(String textToEncrypt)

{

**byte**[] byteCipherText = **null**;

**try** {

Cipher encCipher = Cipher.*getInstance*(mEncryptionAlgorithm);

encCipher.init(Cipher.*ENCRYPT\_MODE*, mSecretKey);

byteCipherText = encCipher.doFinal(textToEncrypt.getBytes());

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (NoSuchPaddingException e) {

e.printStackTrace();

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

}

**return** byteCipherText;

}

**public** String decryptText(**byte**[] decryptedKey, **byte**[] encryptedText)

{

String decryptedPlainText = **null**;

**try** {

SecretKey originalKey = **new** SecretKeySpec(decryptedKey , 0, decryptedKey.length, mEncryptionAlgorithm);

Cipher aesCipher2 = Cipher.*getInstance*(mEncryptionAlgorithm);

aesCipher2.init(Cipher.*DECRYPT\_MODE*, originalKey);

**byte**[] bytePlainText = aesCipher2.doFinal(encryptedText);

decryptedPlainText = **new** String(bytePlainText);

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (NoSuchPaddingException e) {

e.printStackTrace();

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

}

**return** decryptedPlainText;

}

}

**package** encryption;

**public** **class** SymetricTest {

**public** **static** **void** main(String[] args)

{

EncDecSymmetric sed = **new** EncDecSymmetric();

sed.generateSymmetricKey();

System.*out*.println("Secret Key :"+sed.getSecretKey().getAlgorithm()+" sed.getSecretKey().getAlgorithm()"+sed.getSecretKey().getFormat());

**byte**[] secretKeyByteArray = sed.getSecretKeyAsByteArray();

System.*out*.println("secret key: '" + EncDecSymmetric.*keyToNumber*(secretKeyByteArray).toString() + "'" );

String plainText = "Hello World, Symmetric Encryption style";

System.*out*.println("plainText: '" + plainText + "'");

**byte**[] encryptedText = sed.encryptText(plainText);

System.*out*.println("encrypted text: '" + EncDecSymmetric.*keyToNumber*(encryptedText).toString() + "'" );

String decryptedText = sed.decryptText(secretKeyByteArray, encryptedText);

System.*out*.println("decrypted text: '" + decryptedText + "'" );

}

}

**Asymmetric Encryption**When two keys are used in the encryption-decryption process. One key is used for encryption, while the other key is used for decryption. As the same key is not used for encryption and decryption, this technique is also known as 'asymmetric encryption'.

In the PKE method, when Alice and Bob wish to exchange messages, both will generate two keys — a private key and a public key. As the names suggest, the private key is not disclosed, while the public key is shared with everyone. When Bob wishes to send a message to Alice, he uses Alice's public key to encrypt the message and send the encrypted message to Alice. On getting the message, Alice decrypts the message using her private key, to get the original message.

The best part of the PKE method is that the public key can be used by anyone to send a message, which can then be decrypted using the private key. As long as the private key is not compromised, the encrypted message cannot be decrypted easily, if at all.

For Example :

**package** encryption;

**import** java.math.BigInteger;

**import** java.security.InvalidKeyException;

**import** java.security.KeyPair;

**import** java.security.KeyPairGenerator;

**import** java.security.NoSuchAlgorithmException;

**import** java.security.PrivateKey;

**import** java.security.PublicKey;

**import** java.util.Base64;

**import** javax.crypto.BadPaddingException;

**import** javax.crypto.Cipher;

**import** javax.crypto.IllegalBlockSizeException;

**import** javax.crypto.NoSuchPaddingException;

**public** **class** EncDecPublicKeyPrivateKey

{

// key encryption algorithms supported - RSA, Diffie-Hellman, DSA

// key pair generator - RSA: keyword - RSA, key size: 1024, 2048

// key pair generator - Diffie-Hellman: keyword i DiffieHellman, key size - 1024

// key pair generator - DSA: keyword - DSA, key size: 1024

// NOTE: using asymmetric algorithms other than RSA needs to be worked out

**protected** **static** String *DEFAULT\_ENCRYPTION\_ALGORITHM* = "RSA";

**protected** **static** **int** *DEFAULT\_ENCRYPTION\_KEY\_LENGTH* = 1024;

**protected** **static** String *DEFAULT\_TRANSFORMATION* = "RSA/ECB/PKCS1Padding";

**protected** String mEncryptionAlgorithm, mTransformation;

**protected** **int** mEncryptionKeyLength;

**protected** PublicKey mPublicKey;

**protected** PrivateKey mPrivateKey;

EncDecPublicKeyPrivateKey(){

mEncryptionAlgorithm = EncDecPublicKeyPrivateKey.*DEFAULT\_ENCRYPTION\_ALGORITHM*;

mEncryptionKeyLength = EncDecPublicKeyPrivateKey.*DEFAULT\_ENCRYPTION\_KEY\_LENGTH*;

mTransformation = EncDecPublicKeyPrivateKey.*DEFAULT\_TRANSFORMATION*;

mPublicKey = **null**;

mPrivateKey = **null**;

}

**public** **static** BigInteger keyToNumber(**byte**[] byteArray){

**return** **new** BigInteger(1, byteArray);

}

**public** String getEncryptionAlgorithm(){

**return** mEncryptionAlgorithm;

}

**public** **int** getEncryptionKeyLength(){

**return** mEncryptionKeyLength;

}

**public** String getTransformation(){

**return** mTransformation;

}

**public** PublicKey getPublicKey(){

**return** mPublicKey;

}

**public** **byte**[] getPublicKeyAsByteArray(){

**return** mPublicKey.getEncoded();

}

**public** String getEncodedPublicKey(){

String encodedKey = Base64.*getEncoder*().encodeToString(mPublicKey.getEncoded());

**return** encodedKey;

}

**public** PrivateKey getPrivateKey(){

**return** mPrivateKey;

}

**public** **byte**[] getPrivateKeyAsByteArray(){

**return** mPrivateKey.getEncoded();

}

**public** String getEncodedPrivateKey(){

String encodedKey = Base64.*getEncoder*().encodeToString(mPrivateKey.getEncoded());

**return** encodedKey;

}

**public** **byte**[] encryptText(String text){

**byte**[] encryptedText = **null**;

**try** {

KeyPairGenerator kpg = KeyPairGenerator.*getInstance*(mEncryptionAlgorithm);

kpg.initialize(mEncryptionKeyLength);

KeyPair keyPair = kpg.generateKeyPair();

mPublicKey = keyPair.getPublic();

mPrivateKey = keyPair.getPrivate();

Cipher cipher = Cipher.*getInstance*(mTransformation);

cipher.init(Cipher.*PUBLIC\_KEY*, mPublicKey);

encryptedText = cipher.doFinal(text.getBytes());

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (NoSuchPaddingException e) {

e.printStackTrace();

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

}

**return** encryptedText;

}

**public** **byte**[] decryptText(**byte**[] encryptedText){

**byte**[] decryptedText = **null**;

**try** {

Cipher cipher = Cipher.*getInstance*(mTransformation);

cipher.init(Cipher.*PRIVATE\_KEY*, mPrivateKey);

decryptedText = cipher.doFinal(encryptedText);

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (NoSuchPaddingException e) {

e.printStackTrace();

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

}

**return** decryptedText;

}

}

**package** encryption;

**public** **class** AsymmetricTest {

**public** **static** **void** encryptDecrypt(String plainText)

{

EncDecPublicKeyPrivateKey edpkpk = **new** EncDecPublicKeyPrivateKey();

//byte[] secretKeyByteArray = sed.getSecretKeyAsByteArray();

//System.out.println("secret key: '" + EncryptDecryptPublicKeyPrivateKey.keyToNumber(secretKeyByteArray).toString() + "'" );

System.*out*.println("plainText: '" + plainText + "'");

System.*out*.println("plainText size: '" + plainText.length() + "'");

System.*out*.println("encryption key length: '" + edpkpk.getEncryptionKeyLength() + "'");

System.*out*.println("encryption algorithm: '" + edpkpk.getEncryptionAlgorithm() + "'");

System.*out*.println("encryption transform: '" + edpkpk.getTransformation() + "'");

**byte**[] encryptedText = edpkpk.encryptText(plainText);

System.*out*.println("encrypted text: '" + EncDecPublicKeyPrivateKey.*keyToNumber*(encryptedText).toString() + "'" );

System.*out*.println("encrypted text length: '" + EncDecPublicKeyPrivateKey.*keyToNumber*(encryptedText).toString().length() + "'" );

System.*out*.println("public key: '" + EncDecPublicKeyPrivateKey.*keyToNumber*(edpkpk.getPublicKeyAsByteArray()).toString() + "'" );

System.*out*.println("public key length: '" + EncDecPublicKeyPrivateKey.*keyToNumber*(edpkpk.getPublicKeyAsByteArray()).toString().length() + "'" );

System.*out*.println("private key: '" + EncDecPublicKeyPrivateKey.*keyToNumber*(edpkpk.getPrivateKeyAsByteArray()).toString() + "'" );

System.*out*.println("private key length: '" + EncDecPublicKeyPrivateKey.*keyToNumber*(edpkpk.getPrivateKeyAsByteArray()).toString().length() + "'" );

String decryptedText = **new** String(edpkpk.decryptText(encryptedText));

System.*out*.println("decrypted text: '" + decryptedText + "'" );

System.*out*.println("decrypted text length: '" + decryptedText.length() + "'");

}

**public** **static** **void** main(String[] args)

{

String plainText1 = "Hello World, Public Key / Private Key style";

AsymmetricTest.*encryptDecrypt*(plainText1);

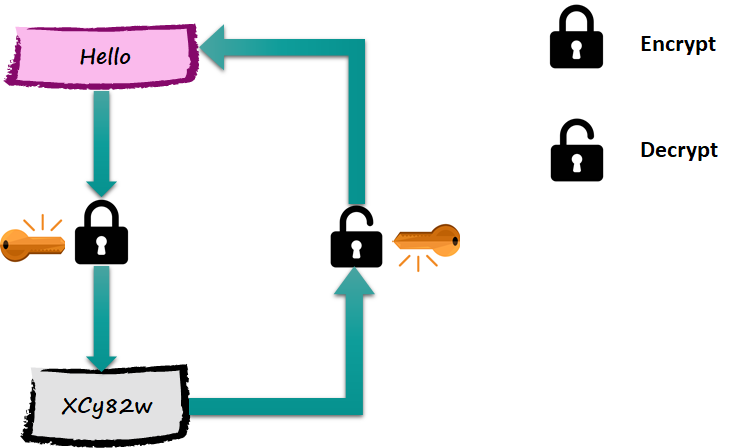
}

}

**Encryption and Signing**

**Symmetric Encryption**

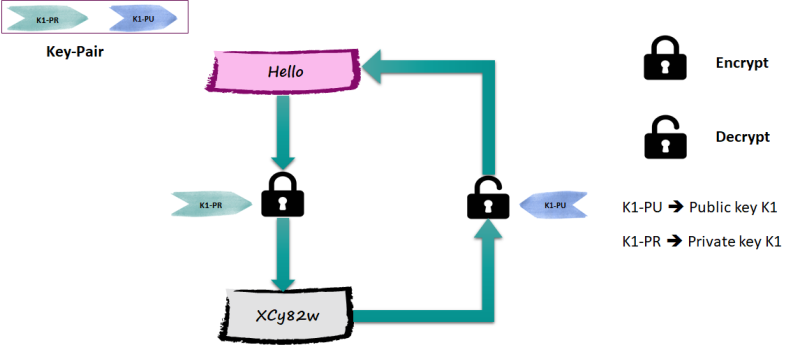
Symmetric encryption means that the same key is used to encrypt and decrypt:

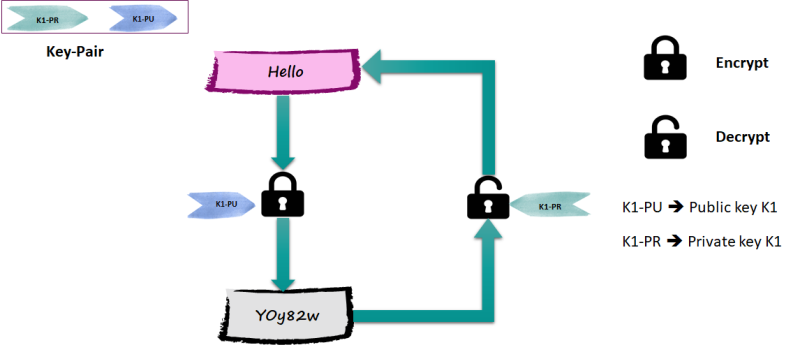


**Public Key Infrastructure (PKI)**

**Asymmetric Encryption**

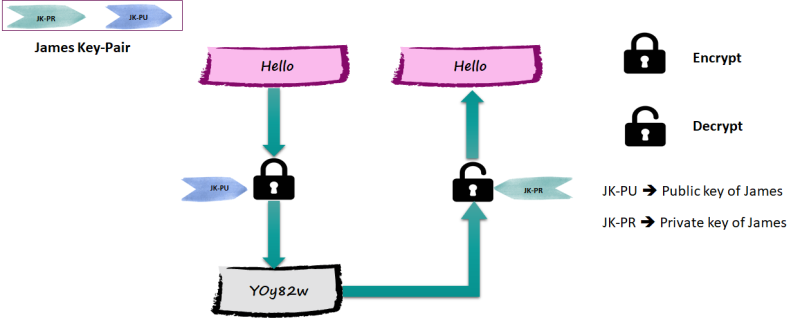
To solve the problem of negotiating 100 keys, if you want to send something privately to 100 individuals/system, a public/private key is used. If a message is encrypted using public key then it can be decrypted only with private key, and on the same line, if a message is encrypted using a private key, then it can decrypted using only a public key (not even with a private key)





* Key-pair (private/public) is generated by owner.
* Private key is kept confidential, while public key is shared with every one.
* It is difficult (next to impossible) to derive private key from public key.

Suppose you are planning to send a message to James privately;, you would use his public key to encrypt the message. Even though other people know the public, they can't decrypt it — only James knows the private key, so he can decrypt it.

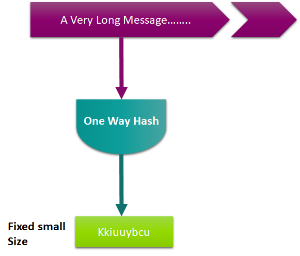


**Digital Signature**

Let's assume that the message you send to James is not confidential; however, James should know the message is really from you. You can use your private key to encrypt the message, and then, James can use your public key to decrypt the message. However, this is not feasible if the message is too long, since the encrypted message would get doubled and it is a time-consuming operation. To solve this, you can feed the message to a **one way hash function** to produce a result (message digest) of same size always (depending upon the algorithm)

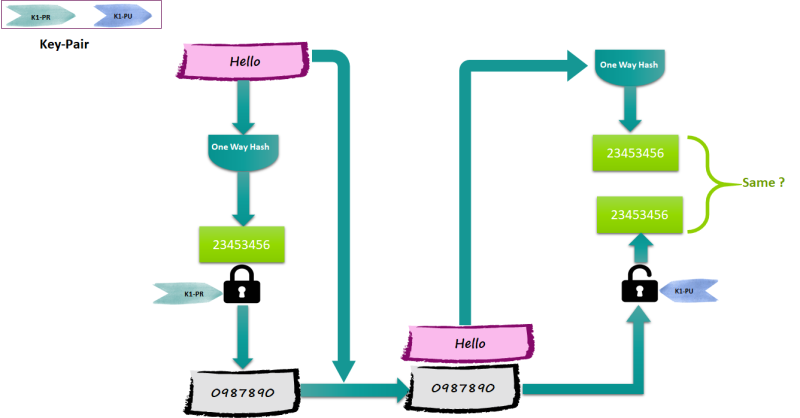
Features of one way hash function:

* Extremely fast
* Impossible to create message from a digest



Now to prove to James that you are who you are claiming to be, you can use your private key to encrypt the message digest and then send both the message and the digest to James.

James can decrypt the digest using your public key, calculate the digest from the original message, and finally compare the message digests.

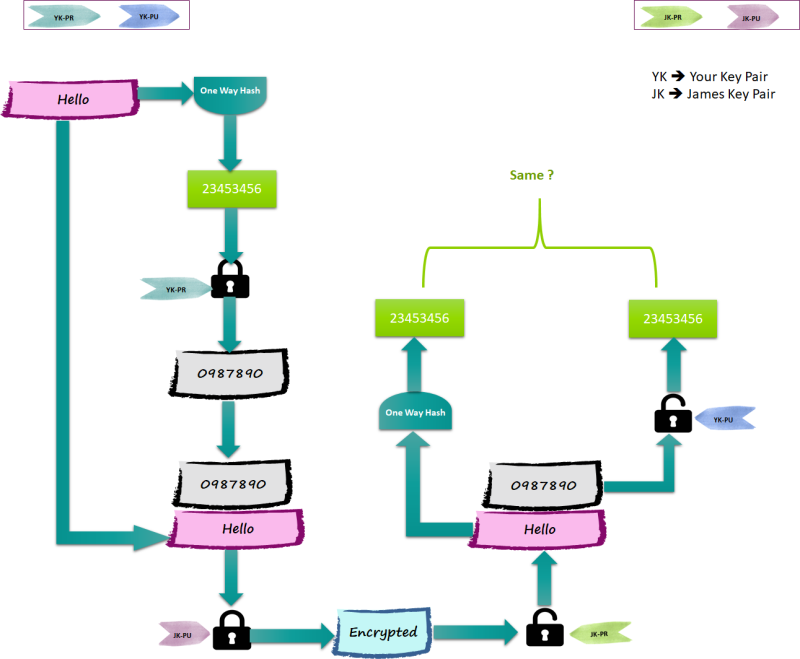


The encrypted digest is called the “**Digital Signature,**” and  the whole process of calculating the digest and then encrypting it is called “**signing the message."**

**Signing and Encrypting**

What if you would like to prove to James that you are who you are claiming to be, and at the same time, you want to make sure message is received only by James?

Follow the diagram.

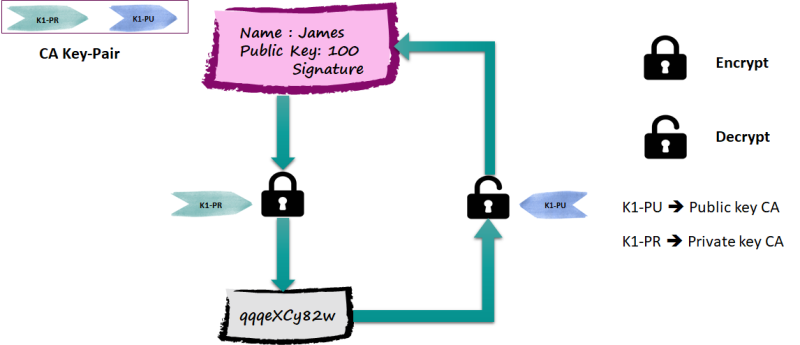


**Certificate and CA**

The problem of how we get one's certificate (without being trapped) and, most importantly, how do we distribute to many individuals is solved by Certificate and CA.

You go to an authority, and by stating your public key with proof of ID, the authority generates an electronic message with your public key that is signed with his (authority) private key.

Such a signed message is called a **certificate**, and such an authority is called the **Certificate Authority** (CA)



# How Code Signing Certificates Work

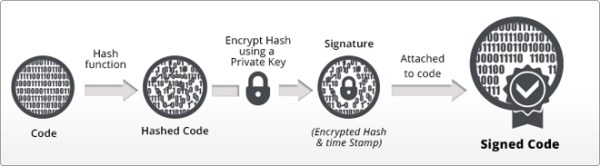
Code signing, as the name implies, is a way for programmers and developers to literally sign their scripts and executables before publishing them. Signing your software/code essentially serves two functions:

1. It provides cryptographic protection against the modification of the code/software
2. It identifies the author of the code/software

## So, How Does Code Signing Work?

Code signing actually works very similarly to SSL in many ways — that’s why many CAs sell both.

We’ll start by looking at the process of obtaining, installing, and using one:

1. **You purchase a certificate**. Depending on whether you’re an organization or an individual, you choose the right code signing option for you. There are even [EV Code Signing Certificates](https://comodosslstore.com/code-signing/comodo-ev-code-signing-certificate) available.
2. **The CA verifies your identity**. The process differs depending on whether you’re a company or an individual, but suffice it to say the CA wants to make sure you are who you say you are and that you’re operating in good faith before they authenticate you.
3. **You install your Code Signing Certificate.** This one is pretty straightforward — you install it on whatever platform you’re using.
4. **You start signing your executables and scripts.** Every platform handles the actual signing process a bit differently. But, one thing is the same — this is where you add your digital signature. As we just discussed, the digital signature really isn’t a signature at all — it’s just a string of data that can be hashed to display your identity and whether or not your code has been altered.
5. **You distribute your signed software**. This is the last step. After you have signed your software/code, you can begin distributing it. Anyone who proceeds to download and run it will be presented with your signature, which, when hashed, will display your identity as a programmer and whether or not the code has been tampered with since you published it.

It’s really as simple as that. Your digital signature contains information on your identity and what exactly it is you signed.

## ****Understanding Public Key Infrastructure (PKI)****

As the name suggests, a Public Key Infrastructure is an infrastructure that uses digital certificates as an authentication mechanism and is designed to manage those certificates and their associated keys.

**Public Key Encryption** is also known as **asymmetric encryption**, and it’s very popular because it is more secure than secret key encryption (also known as symmetric) encryption. In Public Key Encryption, two related keys, one public and one private, work together to with one used for encryption and the other used for decrypting. In this model, the public key — as the name would suggest — is publicly available to anyone who wants to begin encrypted communication with the holder of the private key. The private key is never shared.

## Components of PKI

**Public Key Infrastructures** are not universal — it’s not as if there’s a single PKI that governs all digital certificates. Rather, a PKI can be built for a single organization and implemented only on that organization’s network or it can be a much larger commercial PKI that governs certificates issued to internet users.

**Regardless, all PKIs feature the following four components:**

* A **Certification Authority to issue certificates** – A trusted CA is the only entity that can issue trusted digital certificates. This is extremely important because while PKI manages more of the encryption side of these certificates, authentication is vital to understanding which entities own what keys. Without a trusted CA, anyone can issue their own keys, authentication goes out the window and chaos ensues.
* **Policies that govern the PKI** – Bear in mind that PKI is largely about governance and management of digital CA certificates. In order to achieve both, a set of rules or guidelines must be in place to ensure things go smoothly. For smaller PKIs, these guidelines or often determined in-house by an IT admin or someone knowledgeable. For larger commercial PKIs, they’re determined by a collective of browsers and certificate authorities called the CA/B Forum.
* T**he Digital Certificates themselves** – It’s kind of tough manage a group of digital certificates that don’t exist. In order for a PKI to work and exist properly, it needs to have digital certificates, otherwise—what’s the point?
* **Apps that are written to use the PKI** – This last one may seem abstract, it’s really not. This just means any application that is PKI aware and uses the PKI to facilitate an encrypted connection. Take some of the larger commercial PKIs, this would mean web browser, email clients, etc…

## What Are Certificate Authorities? Why Are Certificate Authorities a Vital Part of PKI?

As we’ve already established, a PKI is a complex system for governing and managing digital certificates. It helps to facilitate encryption while also verifying the owners of the public keys themselves.

This last portion is why the **Certificate Authorities** are so important. If you remove the CAs from PKI you essentially have a large, unverified group of digital CA certificates, many of which are likely viable but some of which could also be used maliciously given that there’s no way to verify ownership of them. For a layman, this means that someone could essentially misrepresent ownership of a given key and then steal encrypted data—or manipulate it.

We can’t have that. So, as a result, the Certificate Authorities are in place to help with authentication. Authentication simply means you’re proving ownership over a given certificate, and by extension that certificate’s key. The CAs are trusted for a reason, they have invested heavily in their own infrastructure and have robust operations in place that are capable of verifying identities and issuing digital certificates properly. They follow guidelines handed down by the browser community and maintain best practices aimed at ensuring optimal web security.

Basically, they’re trusted for a reason. And because of that trust, we can also trust the certificates they issue, which makes management of those certificates via PKI that much easier.

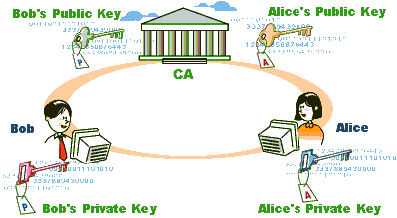
## How Does a Certificate Authority Work? The Role of CA

Well, in order to be a trusted **Certificate Authority**, you must first have made a multi-million dollar annual investment in the infrastructure that it takes to be an active CA. So, there’s already an upfront cost just for doing business. Beyond that, you have to follow guidelines set for by the CA/B forum that govern issuance and authentication practices.

Then, you have to start actually issuing certificates. We won’t drill all the way down into roots and intermediates, etc. We’ll just touch on the process of actually authenticating and issuing a digital CA certificate. After the certificate is ordered, depending on the level of validation required, the CA goes to work verifying the identity of the applicant.

If it’s simply a [Domain Validation certificate](https://cheapsslsecurity.com/sslproducts/domainvalidatedssl.html), the CA just checks ownership over the domain, and then, once this is satisfied, issues the certificate. For Organization Validation and **Extended Validation**, also known as business validation, the Certificate Authority will use business registration and credit reports to vet the organization applying. This can take between 3-5 days and is typically a fairly extensive process. Once it is complete, the certificate can then be issued and will contain critical details about the business itself.

All of this is essential, especially for a PKI, as it allows the true owner of the keys being managed to be verified and makes the entire endeavor safer and more reliable.



# Digital Signatures and Certificates

**Encryption** – Process of converting electronic data into another form, called cipher text, which cannot be easily understood by anyone except the authorized parties.This assures data security.  
**Decryption**– Process of translating code to data.

* Message is encrypted at the sender's side using various encryption algorithms and decrypted at the receiver's end with the help of the decryption algorithms.
* When some message is to be kept secure like username, password, etc., encryption and decryption techniques are used to assure data security.

**Types of Encryption**

1. **Symmetric Encryption**– Data is encrypted using a key and the decryption is also done using the same key.
2. **Asymmetric Encryption**-Asymmetric Cryptography is also known as public key cryptography. It uses public and private keys to encrypt and decrypt data. One key in the pair which can be shared with everyone is called the public key. The other key in the pair which is kept secret and is only known by the owner is called the private key. Either of the keys can be used to encrypt a message; the opposite key from the one used to encrypt the message is used for decryption.

**Public key**– Key which is known to everyone. Ex-public key of A is 7, this information is known to everyone.  
**Private key**– Key which is only known to the person who's private key it is.  
  
**Authentication**-Authentication is any process by which a system verifies the identity of a user who wishes to access it.  
**Non- repudiation**– Non-repudiation means to ensure that a transferred message has been sent and received by the parties claiming to have sent and received the message. Non-repudiation is a way to guarantee that the sender of a message cannot later deny having sent the message and that the recipient cannot deny having received the message.  
**Integrity**– to ensure that the message was not altered during the transmission.  
**Message digest** -The representation of text in the form of a single string of digits, created using a formula called a one way hash function. Encrypting a message digest with a private key creates a digital signature which is an electronic means of authentication

**Digital Signature**

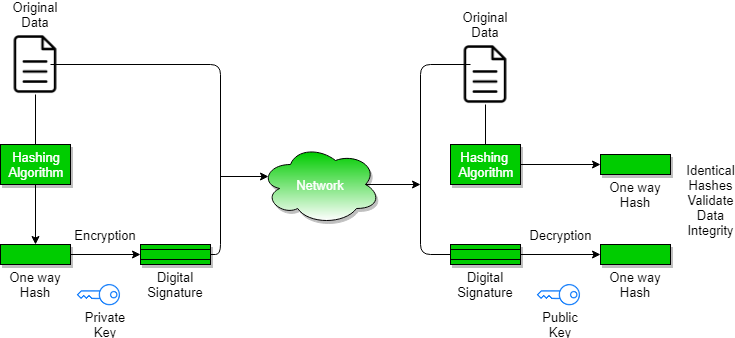
A digital signature is a mathematical technique used to validate the authenticity and integrity of a message, software or digital document.

1. **Key Generation Algorithms** : Digital signature are electronic signatures, which assures that the message was sent by a particular sender. While performing digital transactions authenticity and integrity should be assured, otherwise the data can be altered or someone can also act as if he was the sender and expect a reply.
2. **Signing Algorithms**: To create a digital signature, signing algorithms like email programs create a one-way hash of the electronic data which is to be signed. The signing algorithm then encrypts the hash value using the private key (signature key). This encrypted hash along with other information like the hashing algorithm is the digital signature. This digital signature is appended with the data and sent to the verifier. The reason for encrypting the hash instead of the entire message or document is that a hash function converts any arbitrary input into a much shorter fixed length value. This saves time as now instead of signing a long message a shorter hash value has to be signed and moreover hashing is much faster than signing.
3. **Signature Verification Algorithms** : Verifier receives Digital Signature along with the data. It then uses Verification algorithm to process on the digital signature and the public key (verification key) and generates some value. It also applies the same hash function on the received data and generates a hash value. Then the hash value and the output of the verification algorithm are compared. If they both are equal, then the digital signature is valid else it is invalid.

**The steps followed in creating digital signature are :**

1. Message digest is computed by applying hash function on the message and then message digest is encrypted using private key of sender to form the digital signature. (digital signature = encryption (private key of sender, message digest) and message digest = message digest algorithm(message)).
2. Digital signature is then transmitted with the message.(message + digital signature is transmitted)
3. Receiver decrypts the digital signature using the public key of sender.(This assures authenticity,as only sender has his private key so only sender can encrypt using his private key which can thus be decrypted by sender’s public key).
4. The receiver now has the message digest.
5. The receiver can compute the message digest from the message (actual message is sent with the digital signature).
6. The message digest computed by receiver and the message digest (got by decryption on digital signature) need to be same for ensuring integrity.

Message digest is computed using one-way hash function, i.e. a hash fucntion in which computation of hash value of a message is easy but computation of the message from hash value of the message is very difficult.



**Digital Certificate**

Digital certificate is issued by a trusted third party which proves sender's identity to the receiver and receiver’s identity to the sender.  
A digital certificate is a certificate issued by a Certificate Authority (CA) to verify the identity of the certificate holder. The CA issues an encrypted digital certificate containing the applicant’s public key and a variety of other identification information. Digital certificate is used to attach public key with a particular individual or an entity.  
**Digital certificate contains:-**

1. Name of certificate holder.
2. Serial number which is used to uniquely identify a certificate, the individual or the entity identified by the certificate
3. Expiration dates.
4. Copy of certificate holder's public key.(used for decrypting messages and digital signatures)
5. Digital Signature of the certificate issuing authority.

Digital ceritifcate is also sent with the digital signature and the message.  
  
**Digital certificate vs digital signature :**  
Digital signature is used to verify authenticity, integrity, non-repudiation ,i.e. it is assuring that the message is sent by the known user and not modified, while digital certificate is used to verify the identity of the user, maybe sender or receiver. Thus, digital signature and certificate are different kind of things but both are used for security. Most websites use digital certificate to enhance trust of their users.

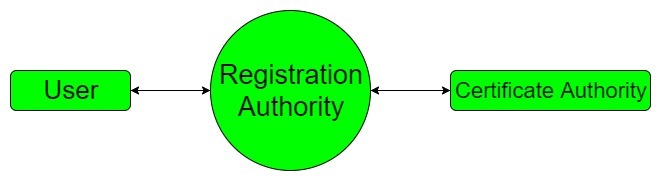
|  |  |  |
| --- | --- | --- |
| **FEATURE** | **DIGITAL SIGNATURE** | **DIGITAL CERTIFICATE** |
| Basics / Definition | Digital signature is like a fingerprint or an attachment to a digital document that ensures its authenticity and integrity. | Digital certificate is a file that ensures holder’s identity and provides security. |
| Process / Steps | Hashed value of original message is encrypted with sender’s secret key to generate the digital signature. | It is generated by CA (Certifying Authority) that involves four steps: Key Generation, Registration, Verification, Creation. |
| Security Services | **Authenticity** of Sender, **integrity** of the document and **non-repudiation**. | It provides security and **authenticity** of certificate holder. |
| Standard | It follows Digital Signature Standard (DSS). | It follows X.509 Standard Format |

# Digital Certificate Creation

The steps required to create a [digital certificate](https://www.geeksforgeeks.org/digital-signatures-certificates/) involves three parties first the end user, second the registration authority and third is certificate authority. The end user request for a digital certificate and the request goes to the registration authority(RA) which then assist the certificate authority(CA) to create the digital certificate. Registration authority act as a intermediate between end user and the certificate authority. It also assist in day to day task of certificate authority.

**Services of Registration Authority:**

* Accepting and verifying the details of new user’s registration.
* User key generation.
* Backups and recovery of key.
* Certificate cancellation.



**Steps for Digital Certificate Creation:**

* **Step-1:** Key generation is done by either user or registration authority. The public key which is generated is sent to the registration authority and private key is kept secret by user.
* **Step-2:** In the next step the registration authority registers the user.
* **Step-3:** Next step is verification which is done by registration authority in which the user’s credentials are being verified by registration authority. It also checks that the user who send the public key have corresponding private key or not.
* **Step-4:** In this step the details and sent to certificate authority by registration authority who creates the digital certificate and give it to users and also keeps a copy to itself.

