Unit - IV

Hash

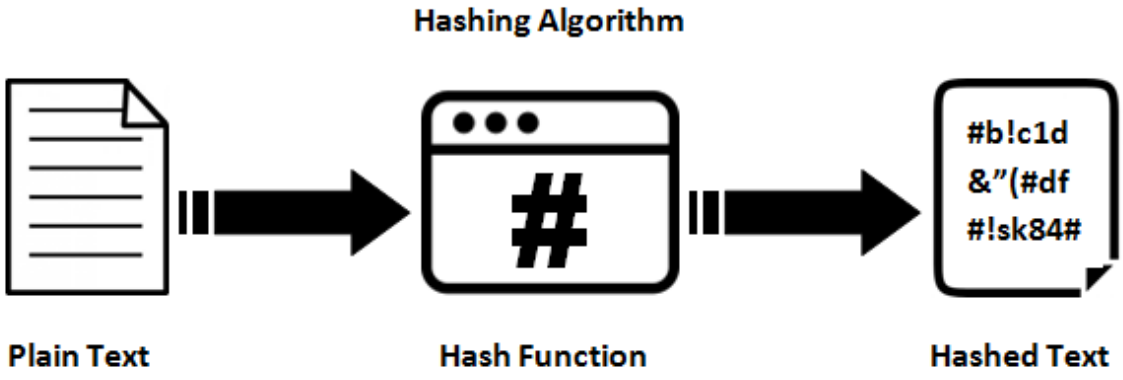
A hashing algorithm is a mathematical function that condenses data to a fixed size. So, for example, if we took the sentence…

"The Quick Brown Fox Jumps Over The Lazy Dog"
…and ran it through a specific hashing algorithm known as **CRC32** we would get:

"07606bb6"
This result is known as a hash or a hash value. Sometimes hashing is referred to as one-way encryption.

Hashes are convenient for situations where computers may want to identify, compare, or otherwise run calculations against files and strings of data. It is easier for the computer to first compute a hash and then compare the values than it would be to compare the original files.

One of the key properties of hashing algorithms is **determinism**. Any computer in the world that understands the hashing algorithm you have chosen can locally compute the hash of our example sentence and get the same answer.



Hashing algorithms are used in all sorts of ways – they are used for storing passwords, in computer vison, in databases, etc.

There are hundreds of hashing algorithms out there and they all have specific purposes – some are optimized for certain types of data, others are for speed, security, etc.

For the sake of today's discussion, all we care about are the SHA algorithms. SHA stands for Secure Hash Algorithm – its name gives away its purpose – it's for cryptographic security.

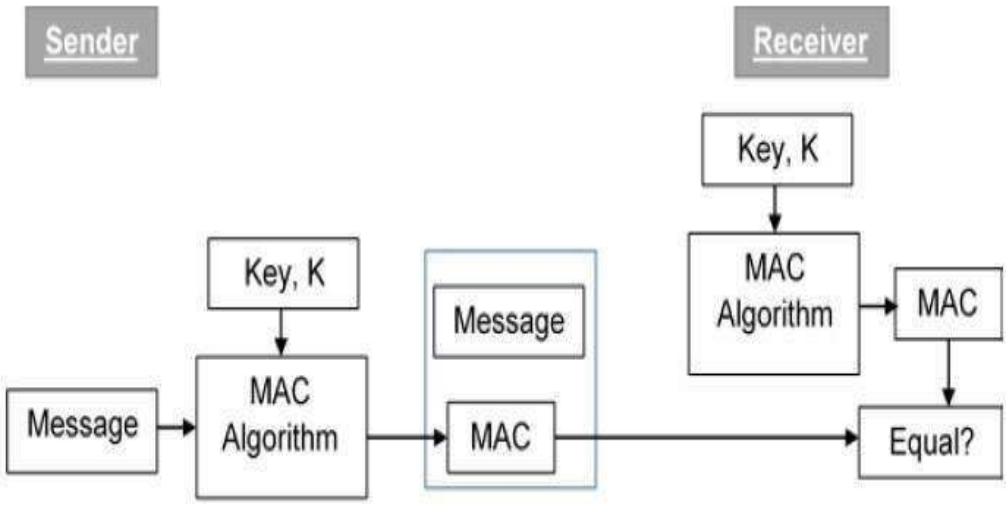
If you only take away one thing from this section, it should be: cryptographic hash algorithms produce **irreversible and unique** hashes. Irreversible meaning that if you only had the hash you couldn't use that to figure out what the original piece of data was, therefore allowing the original data to remain secure and unknown. Unique meaning that two different pieces of data can never produce the same hash – the next section explains why this is so important.

Message Authentication Code (MAC)

MAC algorithm is a symmetric key cryptographic technique to provide message authentication. For establishing MAC process, the sender and receiver share a symmetric key K.

Essentially, a MAC is an encrypted checksum generated on the underlying message that is sent along with a message to ensure message authentication.

The process of using MAC for authentication is depicted in the following illustration −



Let us now try to understand the entire process in detail −

- The sender uses some publicly known MAC algorithm, inputs the message and the secret key K and produces a MAC value.
- Similar to hash, ***MAC function also compresses an arbitrary long input into a fixed length output.*** **The major difference between hash and MAC is that MAC uses secret key during the compression.**
- The sender forwards the message along with the MAC. Here, we assume that the message is sent in the clear, as we are concerned of providing message origin authentication, not confidentiality. If confidentiality is required then the message needs encryption.
- On receipt of the message and the MAC, the receiver feeds the received message and the shared secret key K into the MAC algorithm and re-computes the MAC value.
- The receiver now checks equality of freshly computed MAC with the MAC received from the sender. If they match, then the receiver accepts the message and assures himself that the message has been sent by the intended sender.

- If the computed MAC does not match the MAC sent by the sender, the receiver cannot determine whether it is the message that has been altered or it is the origin that has been falsified. As a bottom-line, a receiver safely assumes that the message is not the genuine.

Limitations of MAC

There are two major limitations of MAC, both due to its symmetric nature of operation –

- **Establishment of Shared Secret.**
  - It can provide message authentication among pre-decided legitimate users who have shared key.
  - This requires establishment of shared secret prior to use of MAC.
- **Inability to Provide Non-Repudiation**
  - Non-repudiation is the assurance that a message originator cannot deny any previously sent messages and commitments or actions.
  - MAC technique does not provide a non-repudiation service. If the sender and receiver get involved in a dispute over message origination, MACs cannot provide a proof that a message was indeed sent by the sender.
  - Though no third party can compute the MAC, still sender could deny having sent the message and claim that the receiver forged it, as it is impossible to determine which of the two parties computed the MAC.

HMAC

HMAC (short for "Keyed-Hash Message Authentication Code") is a cryptographic hash function that uses a secret key as input to the hash function along with the message being hashed. The resulting hash value is unique to the message and the secret key, and can be used to verify the integrity and authenticity of the message.

HMAC is widely used as a secure way to authenticate messages in various communication protocols, including HTTP, SSL, SSH, and many others. It is also commonly used to generate secure hashes for storing passwords, generating unique tokens for session management, and for other security-critical applications.

To compute an HMAC, a message and a secret key are input to a cryptographic hash function along with some additional information, such as the desired length of the hash output and the specific hash algorithm to be used. The hash function then generates a unique hash value based on the message, the secret key, and the additional information.

HMAC is considered a secure and reliable way to authenticate messages because it is resistant to attacks such as dictionary attacks, and because it is difficult to forge without knowing the secret key. It is important to use a strong and unique secret key to get the maximum security benefit from HMAC.

How HMAC Works?

HMAC (short for "Keyed-Hash Message Authentication Code") is a cryptographic hash function that uses a secret key as input to the hash function along with the

message being hashed. The resulting hash value is unique to the message and the secret key, and can be used to verify the integrity and authenticity of the message.

Here's a brief overview of how HMAC works −

- A message is input to the HMAC function along with a secret key.
- The HMAC function applies a cryptographic hash function to the message and the secret key, generating a unique hash value.
- The hash value is output by the HMAC function and can be used to authenticate the message.

To verify the authenticity of a message, the recipient can use the same HMAC function with the same secret key to generate a new hash value for the received message. If the new hash value matches the original hash value, the message is authenticated and considered to be unchanged and unmodified. If the hash values do not match, the message has been tampered with or is otherwise not authentic.

HMAC is widely used as a secure way to authenticate messages in various communication protocols, including HTTP, SSL, SSH, and many others. It is also commonly used to generate secure hashes for storing passwords, generating unique tokens for session management, and for other security-critical applications.

How to Implement HMAC?

To implement HMAC, you will need to use a programming language that provides support for cryptographic hash functions and allows you to specify a secret key. Here is a general outline of the steps you will need to follow −

- Choose a cryptographic hash function to use for the HMAC. Common choices include SHA-1, SHA-256, and SHA-512.
- Choose a secret key to use for the HMAC. The secret key should be a random, unique value that is kept secret from anyone who is not authorized to access the message.
- Input the message and the secret key to the HMAC function along with the desired length of the hash output and the specific hash algorithm to be used.
- The HMAC function will generate a unique hash value based on the message, the secret key, and the additional information.
- The hash value can then be output by the HMAC function and used to authenticate the message.

Example

Here is an example of how to implement HMAC in Python using the hmac module −

```
import hmac

import hashlib


# Choose a message and a secret key

message = b"This is a message to be authenticated"

secret_key = b"this is a secret key"
```

```
# Choose a hash function and the desired length of the hash output

hash_function = hashlib.sha256

hash_length = 32


# Calculate the HMAC

hmac_value = hmac.new(secret_key, message, hash_function).hexdigest()


# Output the HMAC

print(hmac_value)
```

Output

This example will output a hexadecimal string representing the HMAC value for the message and secret key using the SHA-256 hash function.

ba5e5aa5bdc27d02973da9f5a8630d56da634b8bb5483c0ea126890ea9477c8b

Keep in mind that it is important to use a strong and unique secret key to get the maximum security benefit from HMAC. It is also important to choose a cryptographic hash function that is suitable for your application and that has a sufficient level of security for your needs.

When Should You Use HMAC?

HMAC (short for "Keyed-Hash Message Authentication Code") is a cryptographic hash function that uses a secret key as input to the hash function along with the message being hashed. The resulting hash value is unique to the message and the secret key, and can be used to verify the integrity and authenticity of the message.

HMAC is commonly used in situations where it is important to ensure the authenticity and integrity of a message, such as when transmitting sensitive data over an insecure network or when storing password hashes in a database. Some specific examples of when you might use HMAC include −

- **Authenticating HTTP requests** − HMAC can be used to authenticate HTTP requests sent between a client and a server. This can help to prevent man-in-the-middle attacks and other types of tampering.
- **Generating secure tokens** − HMAC can be used to generate unique tokens for session management or other purposes. The tokens can be verified using the same secret key to ensure that they have not been tampered with or forged.
- **Storing password hashes** − HMAC can be used to generate secure hashes for storing passwords in a database. The hashes can be verified using the same secret key when a user attempts to log in, to ensure that the password they enter is correct.
- **Authenticating data transmissions** − HMAC can be used to authenticate data transmissions in protocols such as SSL and SSH, to ensure that the data has not been tampered with during transmission.

HMAC is a widely used and trusted technique for authenticating messages, and can be a valuable tool for ensuring the security and integrity of your data. It is important to use a strong and unique secret key to get the maximum security benefit from HMAC.

Advantages

- HMACs are ideal for high-performance systems like routers due to the use of hash functions which are calculated and verified quickly unlike the public key systems.
- Digital signatures are larger than HMACs, yet the HMACs provide comparably higher security.
- HMACs are used in administrations where public key systems are prohibited.

Disadvantages

- HMACs uses shared key which may lead to non-repudiation. If either sender or receiver's key is compromised then it will be easy for attackers to create unauthorized messages.

Digital Signature

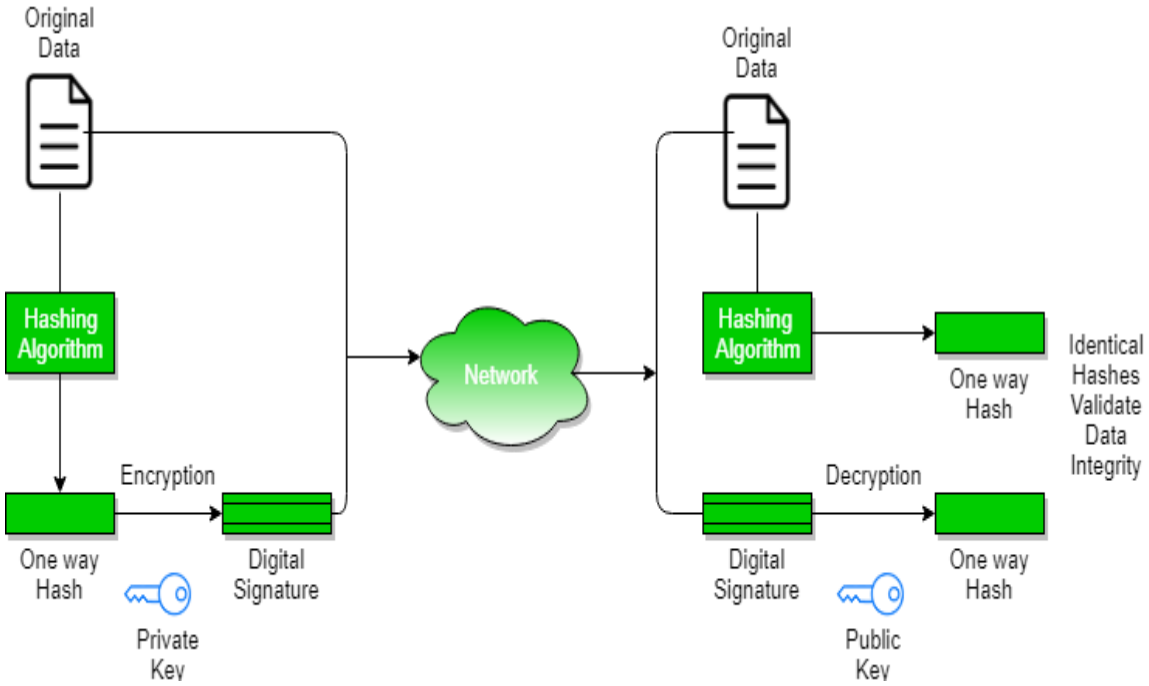
A digital signature is a mathematical technique used to validate the authenticity and integrity of a message, software, or digital document.

1. **Key Generation Algorithms**: Digital signature is electronic signatures, which assure 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 function 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.



Benefits of Digital Signatures

- **Legal documents and contracts:** Digital signatures are legally binding. This makes them ideal for any legal document that requires a signature authenticated by one or more parties and guarantees that the record has not been altered.
- **Sales contracts:** Digital signing of contracts and sales contracts authenticates the identity of the seller and the buyer, and both parties can

be sure that the signatures are legally binding and that the terms of the agreement have not been changed.

- **Financial Documents:** Finance departments digitally sign invoices so customers can trust that the payment request is from the right seller, not from a bad actor trying to trick the buyer into sending payments to a fraudulent account.
- **Health Data:** In the healthcare industry, privacy is paramount for both patient records and research data. Digital signatures ensure that this confidential information was not modified when it was transmitted between the consenting parties.
- Federal, state, and local government agencies have stricter policies and regulations than many private sector companies. From approving permits to stamping them on a timesheet, digital signatures can optimize productivity by ensuring the right person is involved with the proper approvals.
- **Shipping Documents:** Helps manufacturers avoid costly shipping errors by ensuring cargo manifests or bills of lading are always correct. However, physical papers are cumbersome, not always easily accessible during transport, and can be lost. By digitally signing shipping documents, the sender and recipient can quickly access a file, check that the signature is up to date, and ensure that no tampering has occurred.

Drawbacks of Digital Signatures

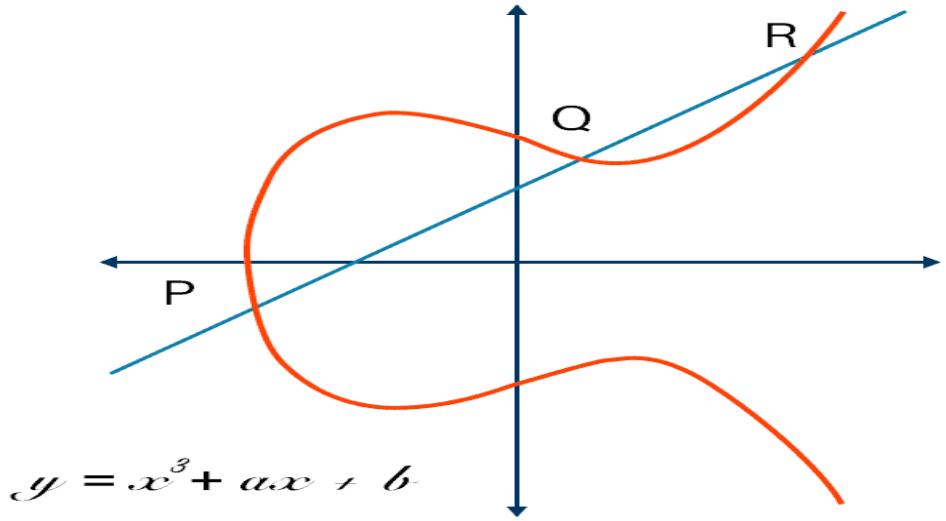
- **Dependence on Key Management:** Digital signatures rely on the secure management of cryptographic keys. This means that the sender must keep their private key safe and secure from unauthorized access, while the recipient must verify the sender's public key to ensure its authenticity. Any failure in key management can compromise the security of the digital signature.
- **Complexity:** Digital signatures require a complex process of key generation, signing, and verification. This can make them difficult to implement and use for non-technical users.
- **Compatibility:** Different digital signature algorithms and formats may not be compatible with each other, making it difficult to exchange signed messages across different systems and applications.
- **Legal Recognition:** Although digital signatures have legal recognition in many countries, their legal status may not be clear in all jurisdictions. This can limit their usefulness in legal or regulatory contexts.
- **Revocation:** In case of key compromise or other security issues, digital signatures must be revoked to prevent their misuse. However, the revocation process can be complex and may not be effective in all cases.
- **Cost:** Digital signatures may involve additional costs for key management, certificate issuance, and other related services, which can make them expensive for some users or organizations.
- **Limited Scope:** Digital signatures provide authentication and integrity protection for a message, but they do not provide confidentiality or

protection against other types of attacks, such as denial-of-service attacks or malware.

Elliptic Curve Cryptography Definition

Elliptic Curve Cryptography (ECC) is a key-based technique for encrypting data. ECC focuses on pairs of public and private keys for decryption and encryption of web traffic.

ECC is frequently discussed in the context of the Rivest–Shamir–Adleman (RSA) cryptographic algorithm. RSA achieves one-way encryption of things like emails, data, and software using prime factorization.



ECC, an alternative technique to RSA, is a powerful cryptography approach. It generates security between key pairs for public key encryption by using the mathematics of elliptic curves.

RSA does something similar with prime numbers instead of elliptic curves, but ECC has gradually been growing in popularity recently due to its smaller key size and ability to maintain security. This trend will probably continue as the demand on devices to remain secure increases due to the size of keys growing, drawing on scarce mobile resources. This is why it is so important to understand elliptic curve cryptography in context.

In contrast to RSA, ECC bases its approach to public key cryptographic systems on how elliptic curves are structured algebraically over finite fields. Therefore, ECC creates keys that are more difficult, mathematically, to crack. For this reason, ECC is considered to be the next generation implementation of public key cryptography and more secure than RSA.

It also makes sense to adopt ECC to maintain high levels of both performance and security. That’s because ECC is increasingly in wider use as websites strive for greater online security in customer data and greater mobile optimization, simultaneously. More sites using ECC to secure data means a greater need for this kind of quick guide to elliptic curve cryptography.

An elliptic curve for current ECC purposes is a plane curve over a finite field which is made up of the points satisfying the equation:
$y^2=x^3 + ax + b$.

In this elliptic curve cryptography example, any point on the curve can be mirrored over the x-axis and the curve will stay the same. Any non-vertical line will intersect the curve in three places or fewer.

Elliptic Curve Cryptography vs RSA

The difference in size to security yield between RSA and ECC encryption keys is notable. The table below shows the sizes of keys needed to provide the same level of security. In other words, an elliptic curve cryptography key of 384 bit achieves the same level of security as an RSA of 7680 bit.

RSA Key Length (bit)
1024
2048
3072
7680
15360

ECC Key Length (bit)
160
224
256
384
521

There is no linear relationship between the sizes of ECC keys and RSA keys. That is, an RSA key size that is twice as big does not translate into an ECC key size that's doubled. This compelling difference shows that ECC key generation and signing are substantially quicker than for RSA, and also that ECC uses less memory than does RSA.

Also, unlike in RSA, where both are integers, in ECC the private and public keys are not equally exchangeable. Instead, in ECC the public key is a point on the curve, while the private key is still an integer.

A quick comparison of the advantages and disadvantages of ECC and RSA algorithms looks like this:

ECC features smaller ciphertexts, keys, and signatures, and faster generation of keys and signatures. Its decryption and encryption speeds are moderately fast. ECC enables lower latency than inverse throughout by computing signatures in two stages. ECC features strong protocols for authenticated key exchange and support for the tech is strong.

The main disadvantage of ECC is that it isn't easy to securely implement. Compared to RSA, which is much simpler on both the verification and encryption sides, ECC is a steeper learning curve and a bit slower for accumulating actionable results.

However, the disadvantages of RSA catch up with you soon. Key generation is slow with RSA, and so is decryption and signing, which aren't always that easy to implement securely.

Advantages of Elliptic Curve Cryptography

Public-key cryptography works using algorithms that are easy to process in one direction and difficult to process in the reverse direction. For example, RSA relies on the fact that multiplying prime numbers to get a larger number is easy, while factoring huge numbers back to the original primes is much more difficult.

However, to remain secure, RSA needs keys that are 2048 bits or longer. This makes the process slow, and it also means that key size is important.

Size is a serious advantage of elliptic curve cryptography, because it translates into more power for smaller, mobile devices. It's far simpler and requires less energy to factor than it is to solve for an elliptic curve discrete logarithm, so for two keys of the same size, RSA's factoring encryption is more vulnerable.

Using ECC, you can achieve the same security level using smaller keys. In a world where mobile devices must do more and more cryptography with less computational power, ECC offers high security with faster, shorter keys compared to RSA.

How Secure is Elliptic Curve Cryptography?

There are several potential vulnerabilities to elliptic curve cryptography, including side-channel attacks and twist-security attacks. Both types aim to invalidate the ECC's security for private keys.

Side-channel attacks including differential power attacks, fault analysis, simple power attacks, and simple timing attacks, typically result in information leaks. Simple countermeasures exist for all types of side-channel attacks.

An additional type of elliptic curve attack is the twist-security attack or fault attack. Such attacks may include invalid-curve attacks and small-subgroup attacks, and they may result in the private key of the victim leaking out. Twist-security attacks are typically simply mitigated with careful parameter validation and curve choices.

Although there are certain ways to attack ECC, the advantages of elliptic curve cryptography for wireless security mean it remains a more secure option.

What Is an Elliptic Curve Digital Signature?

An Elliptic Curve Digital Signature Algorithm (ECDSA) uses ECC keys to ensure each user is unique and every transaction is secure. Although this kind of digital signing algorithm (DSA) offers a functionally indistinguishable outcome as other DSAs, it uses the smaller keys you'd expect from ECC and therefore is more efficient.

What is Elliptic Curve Cryptography Used For?

ECC is among the most commonly used implementation techniques for digital signatures in cryptocurrencies. Both Bitcoin and Ethereum apply the Elliptic Curve Digital Signature Algorithm (ECDSA) specifically in signing transactions. However, ECC is not used only in cryptocurrencies. It is a standard for encryption that will be used by most web applications going forward due to its shorter key length and efficiency.

What is Elliptic Curve Cryptography Used For?

The VMware NSX Advanced Load Balancer's software load balancer offers an elegant ECC solution. The VMware NSX Advanced Load Balancer fully supports termination of SSL– and TLS-encrypted HTTPS traffic. The VMware NSX Advanced Load Balancer's support for SSL/TLS has included support for both RSA and ECC keys without the need for any proprietary hardware. See documentation for Elliptic Curve versus RSA Certificate Priority within the VMware NSX Advanced Load Balancer.

Read these blog posts to learn more about elliptic curve cryptography:

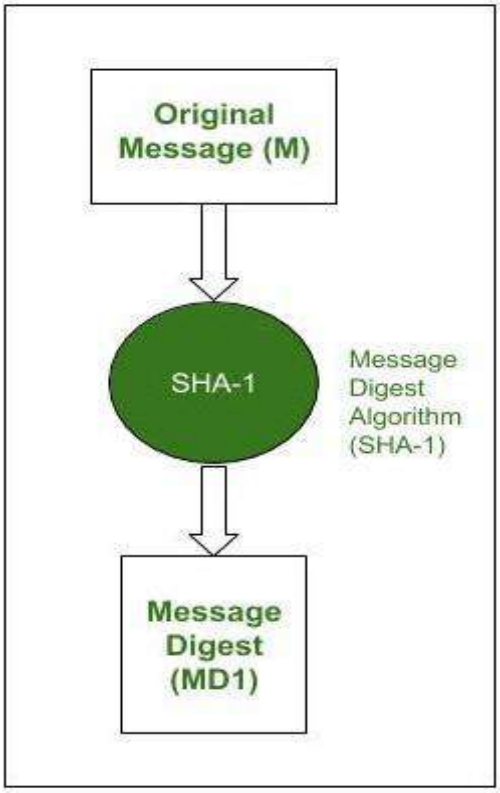
- Five Excuses for Weak SSL Security
- The Hardware Load Balancer Bubble
- The Million Dollar Question – Do Hardware Load Balancers Scale?

For more on the actual implementation of load balancing, security applications and web application firewalls check out our Application Delivery How-To Videos.

RSA-PSS Digital Signature algorithm

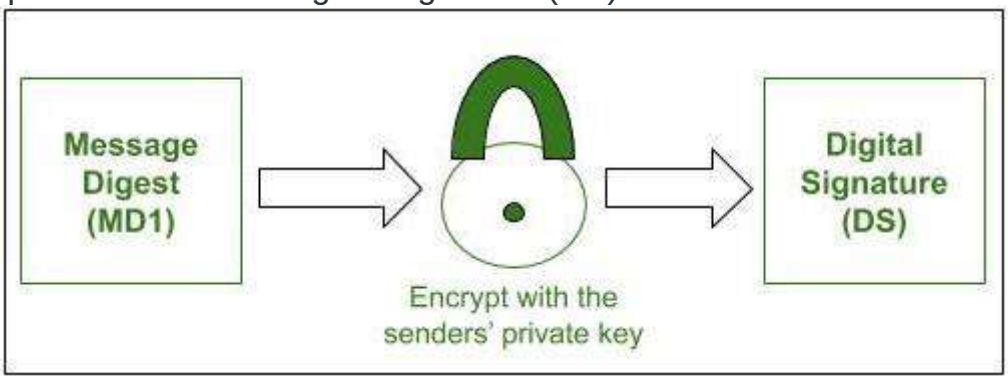
Digital Signature : As the name sounds are the new alternative to sign a document digitally. It ensures that the message is sent by the intended user without any tampering by any third party (attacker). In simple words, digital signatures are used to verify the authenticity of the message sent electronically.
RSA : It is the most popular asymmetric cryptographic algorithm. It is primarily used for encrypting message s but can also be used for performing digital signature over a message. Let us understand how RSA can be used for performing digital signatures step-by-step. Assume that there is a sender (A) and a receiver (B). A wants to send a message (M) to B along with the digital signature (DS) calculated over the message.

Step-1 : Sender A uses SHA-1 Message Digest Algorithm to calculate the message digest (MD1) over the original message M.



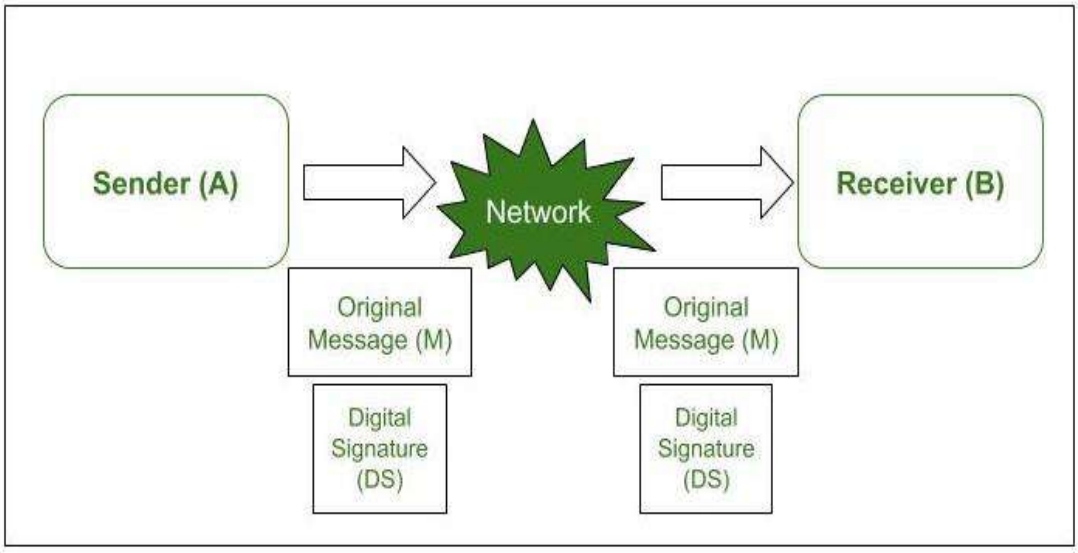
Message digest calculation

Step-2 : A now encrypts the message digest with its private key. The output of this process is called Digital Signature (DS) of A.



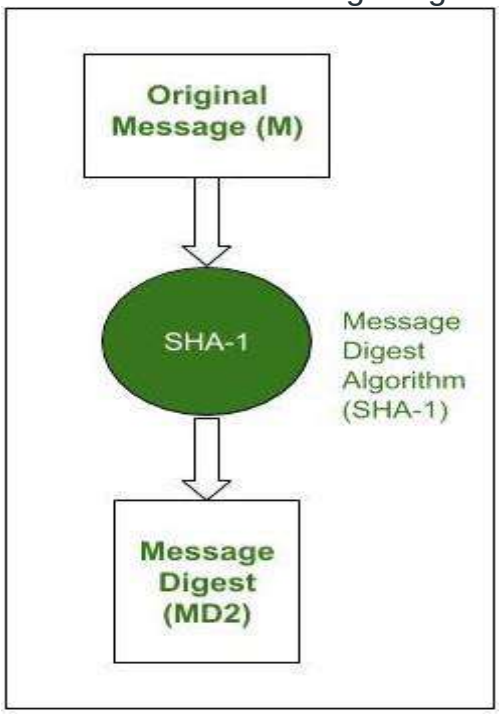
Digital signature creation

Step-3 : Now sender A sends the digital signature (DS) along with the original message (M) to B.



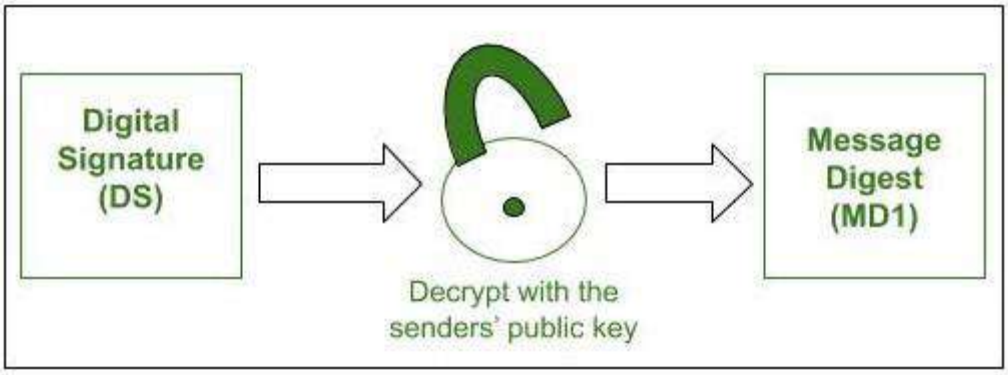
Transmission of original message and digital signature simultaneously

Step-4 : When B receives the Original Message(M) and the Digital Signature(DS) from A, it first uses the same message-digest algorithm as was used by A and calculates its own Message Digest (MD2) for M.



Receiver calculates its own message digest

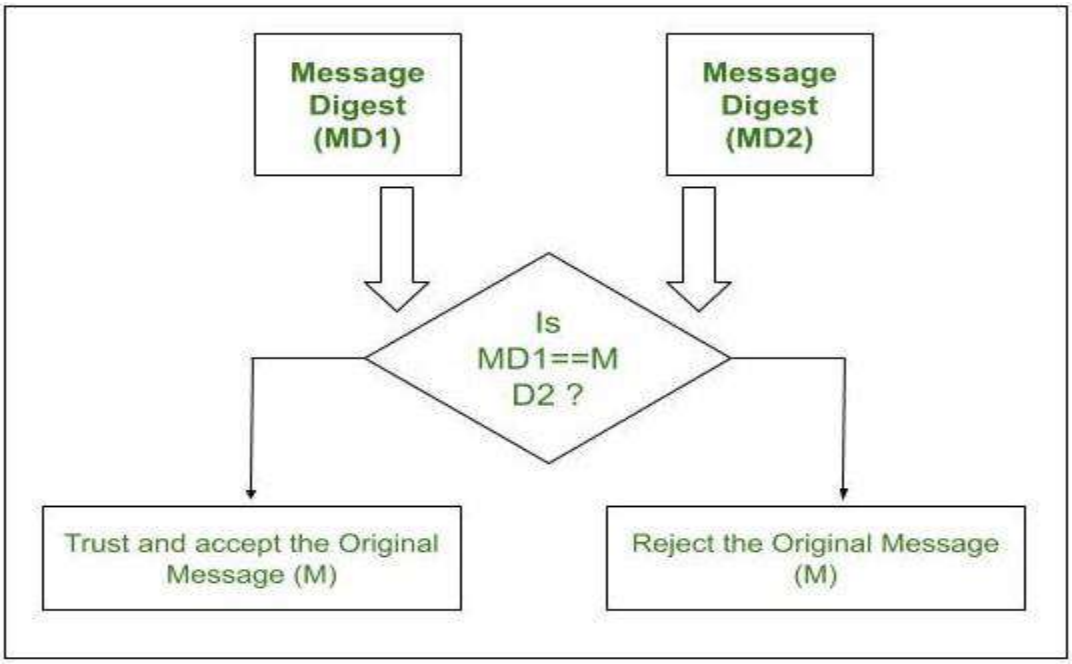
Step-5 : Now B uses A's public key to decrypt the digital signature because it was encrypted by A's private key. The result of this process is the original Message Digest (MD1) which was calculated by A.



Receiver retrieves sender's message digest

Step-6 : If MD1==MD2, the following facts are established as follows.

- B accepts the original message M as the correct, unaltered message from A.
- It also ensures that the message came from A and not someone posing as A.



Digital signature verification

The message digest (MD1) was encrypted using A's private key to produce a digital signature. Therefore, the digital signature can be decrypted using A's public key (due to asymmetric form of RSA). If the receiver B is able to decrypt the digital signature using A's public key, it means that the message is received from A itself and now A cannot deny that he/she has not sent the message. It also proves that the original message did not tamper because when the receiver B tried to find its own message digest MD2, it matched with that of A's MD1. Suppose a malicious user tries to access the original message and perform some alteration. Now he/she will calculate a new message digest over the altered message. It might concern you with data integrity and confidentiality but here's the catch. The attacker will have to

sign the altered message using A's private key in order to pose as A for the receiver B. However, an attacker cannot sign the message with A's private key because it is known to A only. Hence, the RSA signature is quite strong, secure, and reliable. **Attacks on RSA Signature :** There are some attacks that can be attempted by attackers on RSA digital signatures. A few of them are given below as follows.

1. **Chosen-message Attack –** In the chosen-message attack, the attacker creates two different messages, M1 and M2, and somehow manages to persuade the genuine user to sign both the messages using RSA digital-signature scheme. Let's consider message M1 and message M2. so, the attacker computes a new message M = M1 x M2 and then claims that the genuine user has signed message M.
2. **Key-only Attack –** In this attack, the Assumption is that attacker has access to the genuine user public key and tries to get a message and digital signature. OnlyThe attacker then tries to create another message MM such that the same signature S looks to be valid on MM. However, it is not an easy attack to launch since the mathematical complexity beyond this is quite high.
3. **Known-message Attack –** In a known-message attack, the attacker tries to use a feature of RSA whereby two different messages having two different signatures can be combined so that their signatures also combine. To take an example, let us say that we have two different messages M1 and M2 with respective digital signatures as S1 and S2. Then if M = (M1 x M2) mod n, mathematically S = (S1 ¥ S2) mod n. Hence, the attacker can compute M = (M1 x M2) mod n and then S = (S1 x S2) mod n to forge a signature.

RSA is a widely used algorithm for digital signatures because it provides strong security and efficient performance. Digital signatures are used to verify the authenticity of digital documents and ensure that they have not been tampered with. The process of creating a digital signature involves the following steps:

1. Hashing: The first step in creating a digital signature is to create a hash of the message or document that needs to be signed. This is done using a hash function, which produces a fixed-length output (the hash value) from an input of any size.
2. Signing: The hash value is then encrypted using the private key of the signer. This produces the digital signature, which is attached to the original message or document.
3. Verification: To verify the authenticity of the digital signature, the recipient of the message or document must first decrypt the signature using the public key of the signer. This produces the original hash value. The recipient then calculates the hash value of the received message or document using the same hash function that was used by the signer. If the two hash values match, the signature is valid and the message or document has not been tampered with.

4. RSA is well-suited for digital signatures because it provides strong security and efficient performance. The security of RSA is based on the difficulty of factoring large prime numbers. In RSA, the private key is a pair of prime numbers, and the public key is a product of these primes. Because factoring the public key into its prime factors is considered a computationally difficult problem, it is infeasible for an attacker to deduce the private key from the public key.
5. Furthermore, RSA is efficient because the signing process only involves modular exponentiation, which is a relatively fast operation. This makes it suitable for use in a wide range of applications, including digital certificates, secure email, and electronic commerce.

Authentication Systems

Password based

Passwords are the most common form of authentication, is a string of alphabets, special characters, numbers, which is supposed to be known only to the authentic person that is being authenticated. Behind password authentication, there are great myths. People believe that the use of a password is the simplest and least expensive authentication mechanism, as it does not require any special hardware or software support. This is quite the wrong perception. In this article, we are going to see the mechanism implemented for the password in detail.

1. Clear Text Password

A clear-text password is the simplest password-based mechanism. in this mechanism, the user id and password are assigned to the user. for securing purposes, the user changes the password periodically. the password is stored in the database against the user ID in clear text format.

This password authentication works as follows:

Step 1: Prompt for user ID and password

During the authentication process, the first application prompt a screen for user ID and password.

Step 2: User enters user ID and password

When the application prompt the screen, the user enters his/her user ID and password and press the OK or sign in button. after clicking OK, both user ID and password travel to the server in clear text format.

Step 3: User ID and password validation

In this step, the server checks the user database to see if this particular user ID and password exist or not. this validation is done by the user authenticator program. these programs check the entered user ID and password against the stored user database, and based on the success or failure; it returns a result.

Step 4: Authentication Result

In this step, the user authenticator program returns a result to a server based on the success and failure of the validation process of user ID and password.

Step 5: Inform User Accordingly

Based on the results, the server sends the result to the user. if the authentication results are correct, then the server sends a menu of options to the user, which contains various actions for users to perform. if the authentication results fail, then the server sends an error screen to the user.

2. Problems

There Are Two Major Problems in This Password Authentication Mechanism:

The database contains a password in clear text format: user database contains the user ID and password in clear text format, so if somehow attacker gets access to this user database, an attacker will get the kist of all user ids and passwords. to avoid this, it is advisable to store the user ID and password in an encrypted format

Password travel in clear text from users' computer to server computer: if we store the user ID and password in an encrypted format, the first problem can solve. but what if an attacker gets access to this information while traveling from a users computer to a server computer, a user ID and password travels in clear text format. attackers can easily get access to the password.
Something Derived from a password. to overcome from clear text password mechanism, this mechanism comes into the picture where an algorithm is run on the password and store the output of this algorithm as a password in the database. i.e. when a user enters a password, the users own computer run the algorithm on the password and sends the derived password to the server.

Address based

Address-based authentication doesn't rely on sending passwords around the network, but rather assumes that the identity of the source can be inferred based on the network address from which packets arrive. It was adopted early in the evolution of computer networks by both UNIX and VMS. On UNIX, the Berkeley rtools support such access; on VMS, similar functionality is called PROXY. The general idea can be implemented in various ways.

 Machine B might have a list of network addresses of "equivalent" machines. If machine A is listed, then any account name on A is equivalent to the same account name on B. If a request from A arrives with the name John Smith, then the request will be honored if it for anything that the account John Smith on B was allowed to do. This has the problem that the user has to have the identical account name on all systems.

 Machine B might instead have a list of <address, remote account name, local account name>. If a request arrives from address A with name Jekyll, then the database is scanned for the matching entry, say <A, Jekyll, Hyde>. Then the request is honored provided the local account Hyde is authorized to do the request.

UNIX implements two account mapping schemes:

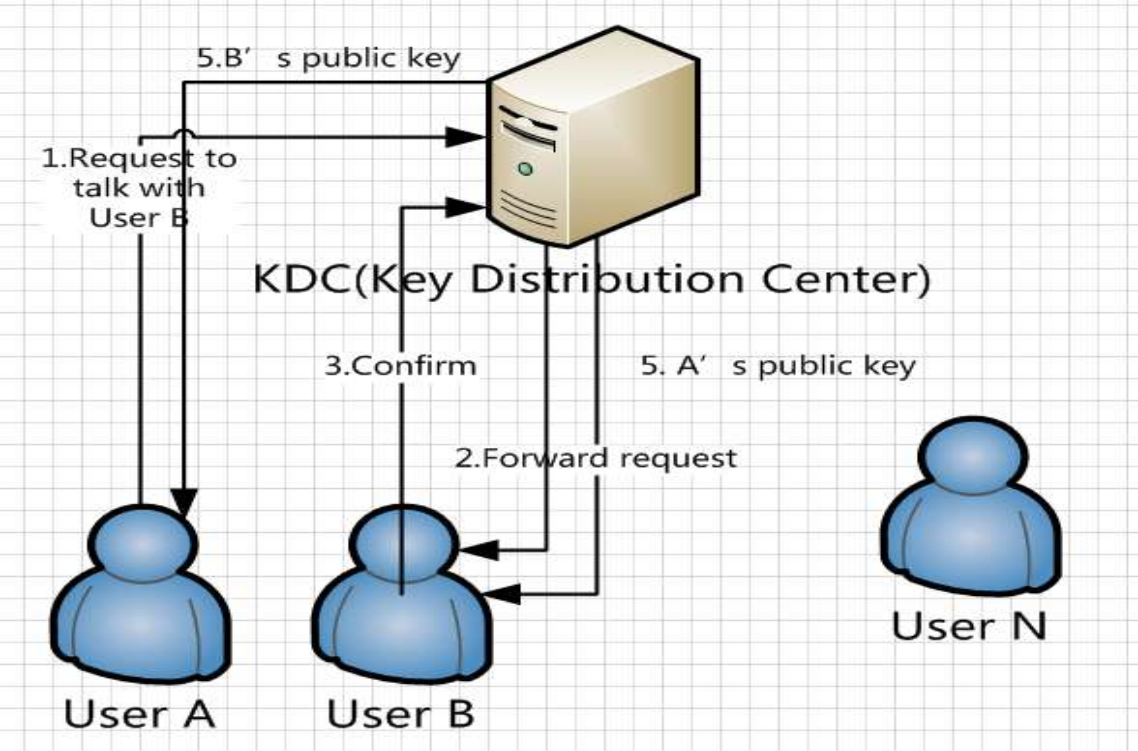
 /etc/hosts.equiv file. There is a global file which implements the first scheme above. The file/ etc/hosts.equiv on a machine A contain a list of computers that have identical user account assignments. Suppose a computer B is listed in /etc/hosts.equiv. Suppose A receives a request with account name Smith, and B's address in the source address field of the network header. If an account with the name smith exists in machine A, the request will be given the same privileges as the local user Smith. The /etc/hosts.eqiv file is useful in managing corresponding accounts in bulk on machines with common accounts and common management.

 Per-user .rhosts files. In each UNIX user's home directory, there can be a file named .rhosts, which contains a list of <computer, account> pairs that are allowed access to the user's account. Any user Bob can permit remote access to his account by creating a .rhosts file is his home directory. The account names need not be the same on the remote machine as they are on this one, so Bob can handle the case where he has different account names on differentsystems. Because of the way the information is organized,any request that is not for an account named the same as
the source account must include the name of the account that should process the request. For instance, if the local account name on system A is Bob and the request is from computer B, account name Smith, the request has to specify
that it would like to be treated with the privileges given account name Bob.

There is a centrally managed proxy database that says for each remote computer <computer, account> pair what account(s) that pair may access, usually with one of them marked as the default. For example, there might be an entry specifying that account Smith from address B should have access to local accounts Bob and Alice, where the account Bob might be marked as the default. The VMS scheme makes access somewhat more user-friendly than the UNIX scheme in the case where a user has different account names on different systems. Generally (in VMS) the user needn't specify the target account in that case. In the rare case, where a user is authorized to access multiple accounts on the remote computer a target account can be specified in the request to access an account other than the default. Address-based authentication is safe from eavesdropping.

Authentication Protocols

K.D.C - **The key distribution center**



The Key Distribution Center (also known as the “KDC”) is primarily a Central Server that is dedicated solely to the KDC network configuration. It merely consists of a database of all of the end users at the place of business or corporation, as well as their respective passwords and other trusted servers and computers along the network.

It should be noted that the passwords which are stored into a KDC are also encrypted. Now, if one end user wishes to communicate with another end user on a different computer system, the sending party enters their password into the KDC, using a specialized software called the “Kerberos.” When the KDC receives the password, the Kerberos then uses a special mathematical algorithm that adds the receiving party’s information and converts it over to the Cryptographic Key.

Once this Encrypted Key has been established, the KDC then sets up and establishes other keys for the encryption of the communication session between the sending and the receiving parties. These other keys are also referred to as “tickets.”

These tickets have a time expiration associated with them so that it expires at a predetermined point in time to prevent unauthorized usage.

Although the KDC System just described does provide a partial solution to the shortcomings of Symmetric Key Cryptography, the KDC also by nature has some major security flaws, which are as follows:

- If an attack is successful on the KDC, the entire communications channel from within the place of business or organization will completely break down. Also, personnel with access to the KDC can easily decrypt the Ciphertext messages between the sending and receiving parties.
- The KDC process presents a single point of failure for the organization. For example, if the server containing the KDC crashes, then all types of secure communications becomes impossible to have, at least on a temporary basis. Also, since all of the end users will be hitting the KDC at peak times, the processing demands placed onto the KDC can be very great, thus heightening the chances that very slow communications between the sending and the receiving parties or even a complete breakdown in communications can happen.

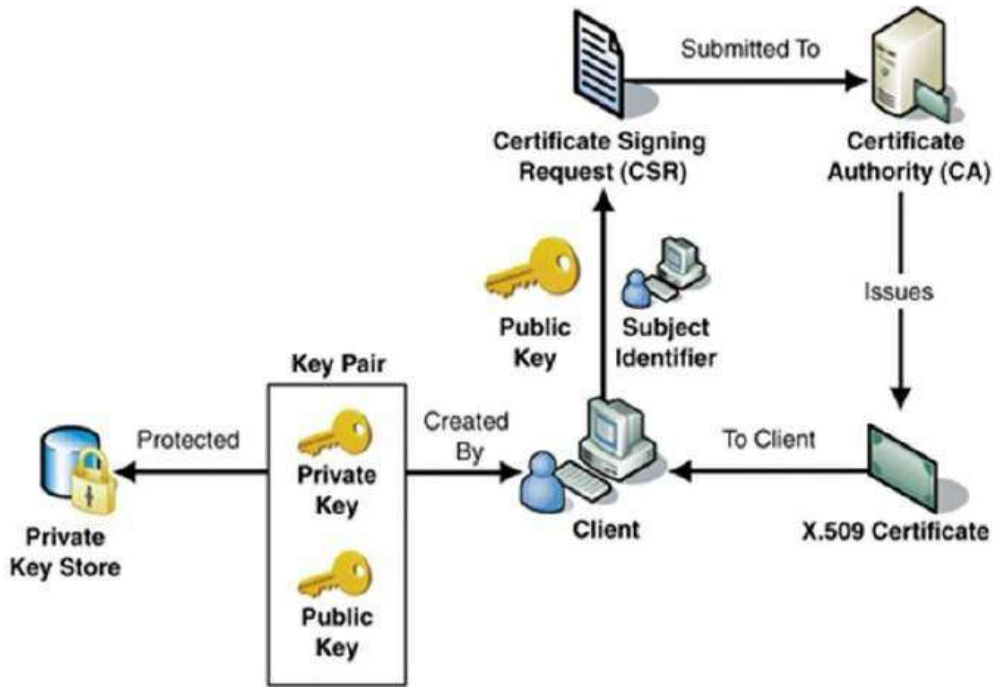
Digital Certificate

For analogy, a certificate can be considered as the ID card issued to the person. People use ID cards such as a driver's license, passport to prove their identity. A digital certificate does the same basic thing in the electronic world, but with one difference.

Digital Certificates are not only issued to people but they can be issued to computers, software packages or anything else that need to prove the identity in the electronic world.

- Digital certificates are based on the ITU standard X.509 which defines a standard certificate format for public key certificates and certification validation. Hence digital certificates are sometimes also referred to as X.509 certificates.
  Public key pertaining to the user client is stored in digital certificates by The Certification Authority (CA) along with other relevant information such as client information, expiration date, usage, issuer etc.
- CA digitally signs this entire information and includes digital signature in the certificate.
- Anyone who needs the assurance about the public key and associated information of client, he carries out the signature validation process using CA’s public key. Successful validation assures that the public key given in the certificate belongs to the person whose details are given in the certificate.

The process of obtaining Digital Certificate by a person/entity is depicted in the following illustration.



As shown in the illustration, the CA accepts the application from a client to certify his public key. The CA, after duly verifying identity of client, issues a digital certificate to that client.

Certifying Authority (CA)

As discussed above, the CA issues certificate to a client and assist other users to verify the certificate. The CA takes responsibility for identifying correctly the identity of the client asking for a certificate to be issued, and ensures that the information contained within the certificate is correct and digitally signs it.

Key Functions of CA

The key functions of a CA are as follows −

- **Generating key pairs** − The CA may generate a key pair independently or jointly with the client.
- **Issuing digital certificates** − The CA could be thought of as the PKI equivalent of a passport agency − the CA issues a certificate after client provides the credentials to confirm his identity. The CA then signs the certificate to prevent modification of the details contained in the certificate.
- **Publishing Certificates** − The CA need to publish certificates so that users can find them. There are two ways of achieving this. One is to publish certificates in the equivalent of an electronic telephone directory. The other is to send your certificate out to those people you think might need it by one means or another.
- **Verifying Certificates** − The CA makes its public key available in environment to assist verification of his signature on clients' digital certificate.
- **Revocation of Certificates** − At times, CA revokes the certificate issued due to some reason such as compromise of private key by user or loss of trust in the client. After revocation, CA maintains the list of all revoked certificate that is available to the environment.

Classes of Certificates

There are four typical classes of certificate −

- **Class 1** − These certificates can be easily acquired by supplying an email address.
- **Class 2** − These certificates require additional personal information to be supplied.
- **Class 3** − These certificates can only be purchased after checks have been made about the requestor's identity.
- **Class 4** − They may be used by governments and financial organizations needing very high levels of trust.

Registration Authority (RA)

CA may use a third-party Registration Authority (RA) to perform the necessary checks on the person or company requesting the certificate to confirm their identity. The RA may appear to the client as a CA, but they do not actually sign the certificate that is issued.

Certificate Management System (CMS)

It is the management system through which certificates are published, temporarily or permanently suspended, renewed, or revoked. Certificate management systems do not normally delete certificates because it may be necessary to prove their status at a point in time, perhaps for legal reasons. A CA along with associated RA runs certificate management systems to be able to track their responsibilities and liabilities.

Private Key Tokens

While the public key of a client is stored on the certificate, the associated secret private key can be stored on the key owner's computer. This method is generally not adopted. If an attacker gains access to the computer, he can easily gain access to private key. For this reason, a private key is stored on secure removable storage token access to which is protected through a password.

Different vendors often use different and sometimes proprietary storage formats for storing keys. For example, Entrust uses the proprietary .epf format, while Verisign, GlobalSign, and Baltimore use the standard .p12 format.

Hierarchy of CA

With vast networks and requirements of global communications, it is practically not feasible to have only one trusted CA from whom all users obtain their certificates. Secondly, availability of only one CA may lead to difficulties if CA is compromised.

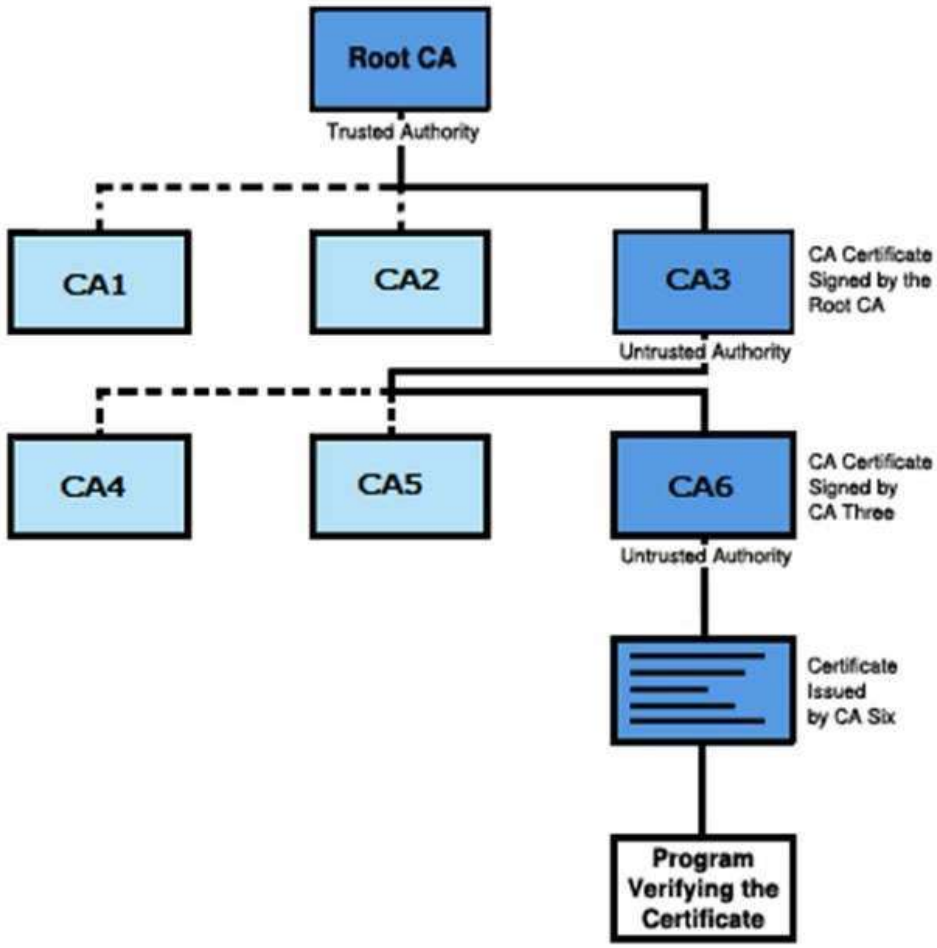
In such case, the hierarchical certification model is of interest since it allows public key certificates to be used in environments where two communicating parties do not have trust relationships with the same CA.

- The root CA is at the top of the CA hierarchy and the root CA's certificate is a self-signed certificate.
- The CAs, which are directly subordinate to the root CA (For example, CA1 and CA2) have CA certificates that are signed by the root CA.

- The CAs under the subordinate CAs in the hierarchy (For example, CA5 and CA6) have their CA certificates signed by the higher-level subordinate CAs.

Certificate authority (CA) hierarchies are reflected in certificate chains. A certificate chain traces a path of certificates from a branch in the hierarchy to the root of the hierarchy.

The following illustration shows a CA hierarchy with a certificate chain leading from an entity certificate through two subordinate CA certificates (CA6 and CA3) to the CA certificate for the root CA.

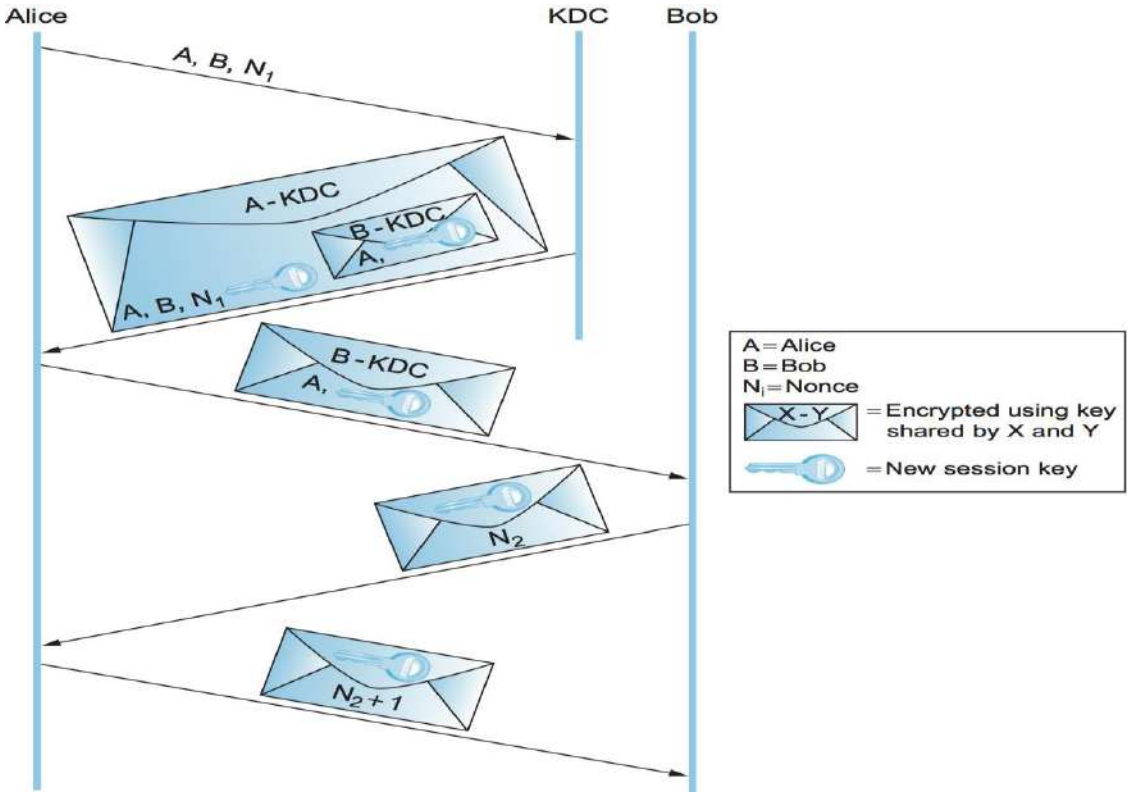


Verifying a certificate chain is the process of ensuring that a specific certificate chain is valid, correctly signed, and trustworthy. The following procedure verifies a certificate chain, beginning with the certificate that is presented for authentication −

- A client whose authenticity is being verified supplies his certificate, generally along with the chain of certificates up to Root CA.
- Verifier takes the certificate and validates by using public key of issuer. The issuer’s public key is found in the issuer’s certificate which is in the chain next to client’s certificate.
- Now if the higher CA who has signed the issuer’s certificate, is trusted by the verifier, verification is successful and stops here.
- Else, the issuer's certificate is verified in a similar manner as done for client in above steps. This process continues till either trusted CA is found in between or else it continues till Root CA

Session Key authentication

Only in fairly small systems is it practical to predistribute secret keys to every pair of entities. We focus here on larger systems, where each entity would have its own *master key* shared only with a Key Distribution Center (KDC). In this case, secret-key-based authentication protocols involve three parties: Alice, Bob, and a KDC. The end product of the authentication protocol is a session key shared between Alice and Bob that they will use to communicate directly, without involving the KDC.



The Needham-Schroeder authentication protocol.

The Needham-Schroeder authentication protocol is illustrated above. Note that the KDC doesn't actually authenticate Alice's initial message and doesn't communicate with Bob at all. Instead, the KDC uses its knowledge of Alice's and Bob's master keys to construct a reply that would be useless to anyone other than Alice (because only Alice can decrypt it) and contains the necessary ingredients for Alice and Bob to perform the rest of the authentication protocol themselves.

The nonce in the first two messages is to assure Alice that the KDC's reply is fresh. The second and third messages include the new session key and Alice's identifier,

encrypted together using Bob's master key. It is a sort of secret-key version of a public-key certificate; it is in effect a signed statement by the KDC (because the KDC is the only entity besides Bob who knows Bob's master key) that the enclosed session key is owned by Alice and Bob. Although the nonce in the last two messages is intended to assure Bob that the third message was fresh, there is a flaw in this reasoning.

Email

Email is a service which allows us to send the message in electronic mode over the internet. It offers an efficient, inexpensive and real time mean of distributing information among people.

E-Mail Address

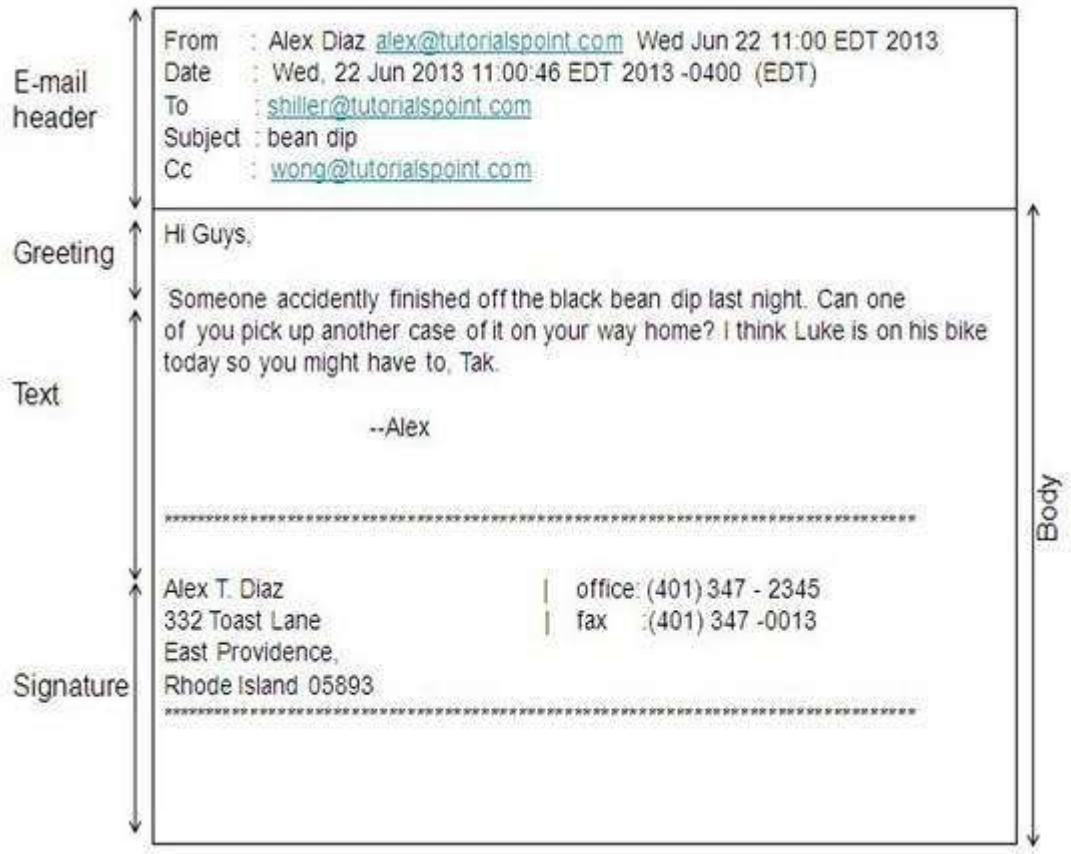
Each user of email is assigned a unique name for his email account. This name is known as E-mail address. Different users can send and receive messages according to the e-mail address.

E-mail is generally of the form username@domainname. For example, webmaster@tutorialspoint.com is an e-mail address where webmaster is username and tutorialspoint.com is domain name.

- The username and the domain name are separated by @ **(at)** symbol.
- E-mail addresses are not case sensitive.
- Spaces are not allowed in e-mail address.
-

E-mail Message Components

E-mail message comprises of different components: E-mail Header, Greeting, Text, and Signature. These components are described in the following diagram:



E-mail Header

The first five lines of an E-mail message is called E-mail header. The header part comprises of following fields:

- From
- Date
- To
- Subject
- CC
- BCC

From

The **From** field indicates the sender's address i.e. who sent the e-mail.

Date

The **Date** field indicates the date when the e-mail was sent.

To

The **To** field indicates the recipient's address i.e. to whom the e-mail is sent.

Subject

The **Subject** field indicates the purpose of e-mail. It should be precise and to the point.

CC

CC stands for Carbon copy. It includes those recipient addresses whom we want to keep informed but not exactly the intended recipient.

BCC

BCC stands for Black Carbon Copy. It is used when we do not want one or more of the recipients to know that someone else was copied on the message.

Greeting

Greeting is the opening of the actual message. Eg. Hi Sir or Hi Guys etc.

Text

It represents the actual content of the message.

Signature

This is the final part of an e-mail message. It includes Name of Sender, Address, and Contact Number.

Advantages

E-mail has prooved to be powerful and reliable medium of commmunication. Here are the benefits of **E-mail:**

- Reliable
- Convenience
- Speed
- Inexpensive
- Printable
- Global
- Generality

Reliable

Many of the mail systems notify the sender if e-mail message was undeliverable.

Convenience

There is no requirement of stationary and stamps. One does not have to go to post office. But all these things are not required for sending or receiving an mail.

Speed

E-mail is very fast. However, the speed also depends upon the underlying network.

Inexpensive

The cost of sending e-mail is very low.

Printable

It is easy to obtain a hardcopy of an e-mail. Also an electronic copy of an e-mail can also be saved for records.

Global

E-mail can be sent and received by a person sitting across the globe.

Generality

It is also possible to send graphics, programs and sounds with an e-mail.

Disadvantages

Apart from several benefits of E-mail, there also exists some disadvantages as discussed below:

- Forgery
- Overload
- Misdirection
- Junk
- No response

Forgery

E-mail doesn't prevent from forgery, that is, someone impersonating the sender, since sender is usually not authenticated in any way.

Overload

Convenience of E-mail may result in a flood of mail.

Misdirection

It is possible that you may send e-mail to an unintended recipient.

Junk

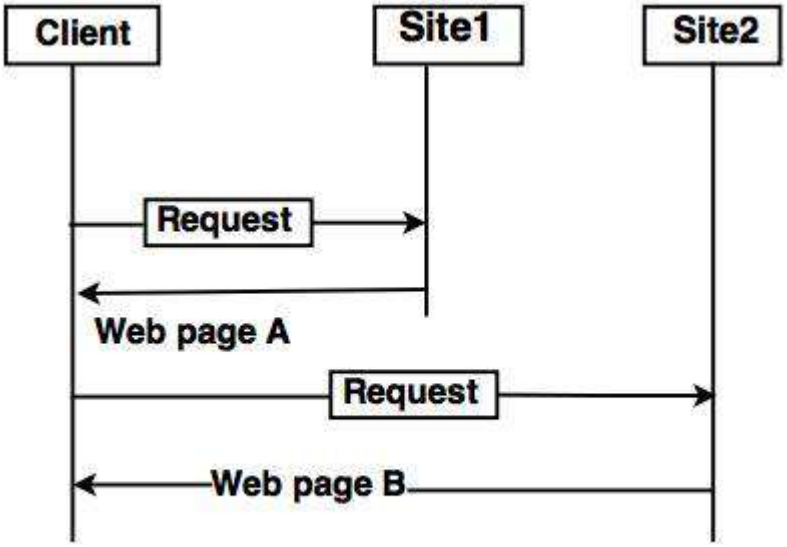
Junk emails are undesirable and inappropriate emails. Junk emails are sometimes referred to as spam.

No Response

It may be frustrating when the recipient does not read the e-mail and respond on a regular basis.

Introduction to World Wide Web

- The World Wide Web (WWW) is a collection of documents and other web resources which are identified by URLs, interlinked by hypertext links, and can be accessed and searched by browsers via the Internet.
- World Wide Web is also called the Web and it was invented by Tim Berners-Lee in 1989.
- Website is a collection of web pages belonging to a particular organization.
- The pages can be retrieved and viewed by using browser.



Architecture of WWW

Let us go through the scenario shown in above fig.

- The client wants to see some information that belongs to site 1.
- It sends a request through its browser to the server at site 2.
- The server at site 1 finds the document and sends it to the client.

Client (Browser):

- Web browser is a program, which is used to communicate with web server on the Internet.
- Each browser consists of three parts: a controller, client protocol and interpreter.
- The controller receives input from input device and use the programs to access the documents.
- After accessing the document, the controller uses one of the interpreters to display the document on the screen.

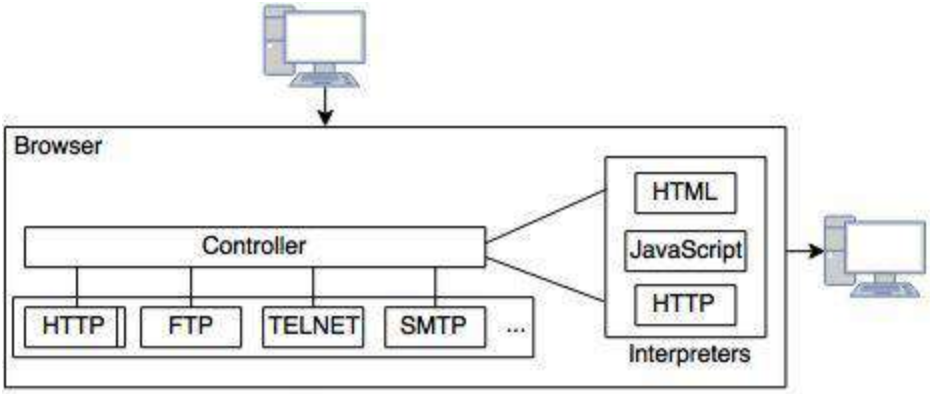


Fig: Cllient (Browser)

Server:

- A computer which is available for the network resources and provides service to the other computer on request is known as server.
- The web pages are stored at the server.
- Server accepts a TCP connection from a client browser.
- It gets the name of the file required.
- Server gets the stored file. Returns the file to the client and releases the top connection.

Uniform Resource Locater (URL)

- The URL is a standard for specifying any kind of information on the Internet.
- The URL consists of four parts: protocol, host computer, port and path.
- The protocol is the client or server program which is used to retrieve the document or file. The protocol can be ftp or http.
- The host is the name of computer on which the information is located.
- The URL can optionally contain the port number and it is separated from the host name by a colon.
- Path is the pathname of the file where the file is stored.