**RSA Algorithm:**

**Public-Key Encryption**

Public-key encryption (asymmetric encryption) involves a pair of keys, a **public key** and a **private key**, associated with an entity.

Each public key is published, and the corresponding private key is kept secret.

Data encrypted with a public key and decrypted only with the corresponding private key.

So it is used when sender expect message in encrypted form from outside world (Public)

**Message Digest**

A message digest is a cryptographic hash function containing a string of digits created by a one-way hashing formula. Message digests are designed to protect the integrity of a piece of data or media to detect changes and alterations to any part of a message.

**Message Digest 2** is a hash function used in cryptography.

Developed in 1989 by Ronald Rivest, it is byte-oriented, producing a 128-bit hash value with the help of an arbitrary length message.

It is optimized for 8-bit computers. Message Digest 2 was developed mainly to be used for digital signature applications, which required a secured and compressed large file to be signed with a private key.

It is rarely used as it takes a long time to compute and is no longer considered secure.

**Message Digest 5 (MD5)** is a hash function used in cryptography. Developed by Ronald Rivest in 1991, Message Digest 5 produces a 128-bit resulting hash value. Similar to other message-digest algorithms, it was largely developed for digital signature applications which make use of a large compressed file in a secure fashion.

**Digital Signature:**

Here, private key is used for encryption and the corresponding public key for decryption.

In practice, this type of signature is not used directly, but rather, the message to be signed is first hashed to produce a short digest that is then signed (i.e. encrypted).

The encrypted hash, along with other information such as the hashing algorithm, is known as a digital signature.

Properties of signature:

* Authentication
  + Digital signatures can be used to authenticate the source of messages.
  + When ownership of a digital signature secret key is bound to a specific user, a valid signature shows that the message was sent by that user.
* Non-repudiation
  + By this property, an entity that has signed some information cannot at a later time deny having signed it.
  + Similarly, access to the public key only does not enable a fraudulent party to fake a valid signature.
* Integrity
  + If a message is digitally signed, any change in the message after signature invalidates the signature.
  + Furthermore, there is no efficient way to modify a message and its signature to produce a new message with a valid signature,
  + Because this is still considered to be computationally infeasible by most cryptographic hash functions.

**Certificate**: CA’s certify public key for any entity or server.

To get personal ID (driver's license), a person has to present some other form of identification which confirms that the person is who he claims to be. Certificates work much the same way.

Certificate authorities (CAs) validate identities and issue certificates.

CAs can be either independent third parties or organizations running their own certificate-issuing server software.

The certificate issued by the CA binds a particular public key to the name of the entity the certificate identifies, such as the name of an employee or a server.

Certificates help prevent the use of fake public keys for impersonation.

Only the public key certified by the certificate will work with the corresponding private key possessed by the entity identified by the certificate.

Certificates have a purpose: to establish trust. Their usage varies depending on the kind of trust they are used to ensure.

**trustStore vs keyStore**

keyStore is used to store private key and own identity certificate which program should present to other parties (Server or client) to verify its identity.

trustStore (as name suggest) is used to store certificates from trusted Certificate authorities(CA) which are used to verify certificate presented by Server in SSL Connection.

When you install JDK/JRE or browser on your machine, Java/browser comes with its own truststore (collection of certificate from well known CA like Verisign, goDaddy, thwarte etc. you can find this file inside JAVA\_HOME/JRE/Security/cacerts

keystore is used to store server's own certificate while truststore is used to store the certificate of other parties issued by CA like Verisign or goDaday or even self-signed certificates.

Keystore is needed when you are setting up server side on SSL, it is used to store server's identity certificate, which server will present to a client on the connection while trust store setup on client side must contain to make the connection work.

If your browser to connect to any website over SSL it verifies certificate presented by server against its truststore.

How can you make your server certificate to be trusted by google-chrome.

Google-chrome trusts certificates signed by VeriSign, because VeriSign’s certificate is in google chrom’s trust store.

So you need to follow the below steps:

Step1: Generate certificate for your server.

Step2: get it signed by VeriSign.

* VeriSign will take some input like name, co name, city, country etc.
* Create the digest of above string and encrypt it with it’s private key.
* Both input string and digest will be inserted into certificate
* This certificate is returned to you and used as server certificate.

Step3: VeriSign will sign your certificate and provide you for use.

Step4: When you use this certificate it will be validated by verisign public key (i.e. certificate installed in browser.). Following steps will be done for verification

* During SSL hand-shake server will provide the signed certificate to client (Google chrome)
* Google-chrome will validate this certificate against it’s trust store
* As verisign public key is (i.e. certificate) installed in browser, it will read the message previously added to server’s certificate during signing.
* Using public key digest is decrypted and verified with the input string
* If both are same then certificate is trusted and server’s public key is used for further processing.

**SSL**

The Secure Sockets Layer (SSL) protocol governs

* Server authentication,
* Client authentication, and
* Encrypted communication between servers and clients.

SSL is widely used on the Internet, especially for interactions that involve exchanging confidential information such as credit card numbers.

SSL requires an SSL server certificate. As part of the initial SSL handshake, the server presents its certificate to the client to authenticate the server's identity. The authentication uses public-key encryption and digital signatures to confirm that the server is the server it claims to be. Once the server has been authenticated, the client and server use symmetric-key encryption, which is very fast, to encrypt all the information exchanged for the remainder of the session and to detect any tampering.

HandShake:

* Agreement on how to encrypt
  + Client sends hello message with following info
  + Key exchange method : RSA
  + Cipher :
  + Hash:
* Server Sends certificates
* Client (browser) says start encrypting
* Server says start encrypting
* All message are encrypted

HTTP:

HTTP

The Hypertext Transfer Protocol (HTTP) is a protocol used mainly to access data on the World Wide Web.

Six basic points of HTTP

* Follows client server (request-response) model
* It is stateless protocol
  + Every transaction is independent and not related to any other transaction
* It is application layer protocol
  + Sits above transport layer (TCP) protocol
* Supports
  + Get, Put, Delete
* Status code :
  + It’s programmer responsibility to return proper status code.
* Headers : All headers are custom info of request and response
  + Purpose are to tweak the operations of the request/response.
  + Content-type (request) are type of files expected from sever.
  + Cache-Control (response):

HTTPS