# Reference links

<https://medium.com/@skshukla.0336/mtls-everything-you-need-to-know-e03804b30804>

Basic concepts

### Terminologies

PKI- means public key infrastructure

Ciphered text – means unreadable text

Cipher – means algorithm which is used to transform plain text to ciphered (encrypted) text

CSR – cert signing request

Man in middle attack – because all the data u send in network can be read by anyone, now if u send the encrypted data, even if middle men reads the ciphered text, without key he cant decrypt the data

Private key - here we will hash the data & generally we will sign the data with only with our private key and we will use the private key for decrypting the encrypted data

(we can get our private key from our certificate)

DSA – means digital signature algorithm

Entity – here entity means server or client who wants the certificate

Important points

* In asymmetric encryption- Public key is used to encrypt the data, private key is used to decrypt the data
* Certificates contains the public keys (hence certificate is mandatory for authentication)

Verifying the data is not tampered

 The data is hashed (e.g., SHA-256) to create a fixed-size digest.

 The hash is encrypted with the sender’s **private key** — that’s the digital signature.

 The recipient uses the sender’s **public key** to verify the signature:

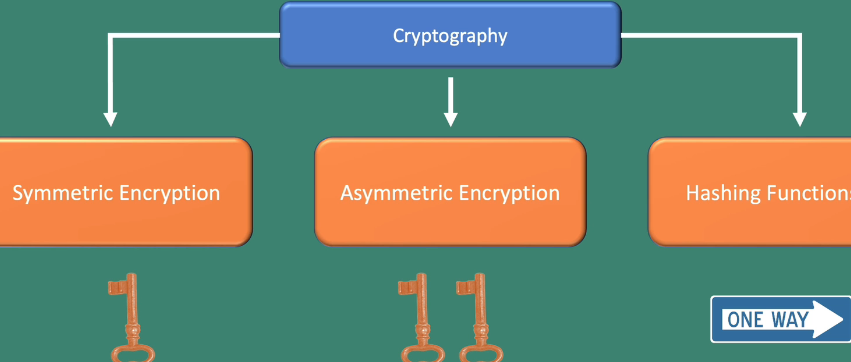
* They decrypt the signature to get the hash.
* They also hash the received data themselves.
* If the two hashes match, the data is authentic and unchanged.

Cryptography

Method of securing data such that is trusted and viewable only by authorized parties

Military people when they send messages they will use cryptography

Cryptograhpy is a art of secure communication -ex:- encrypting, hashing



**Cryptographic keys storage**

* These keys can be stored in PKI certificate
* Or those can be stored on smart card/common access card (smart card= our daily ID card)
* These keys can be stored in a password protected file also
* Token device – hard MFA token or smart android apps

Encoding Vs. encryption vs. hashing

Encoding:- it’s a basic data representation to enable info **which doesn’t involve keeping secrets**

**In encoding security is not the primary concern**

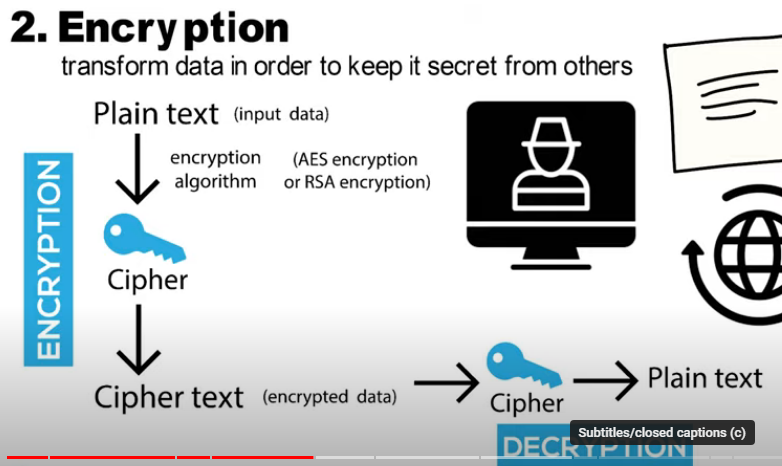
**Encrypting(like cryptography): -** here data can’t be consumed by anyone other than intended recipients

Goals of encryption- confidentialy (only intended people should understand)

It is to transform the data in order **to keep** it secret from others

**Encryption algorithm will use a key To encrypt and decrypt** and the encrypted data is called cipher text

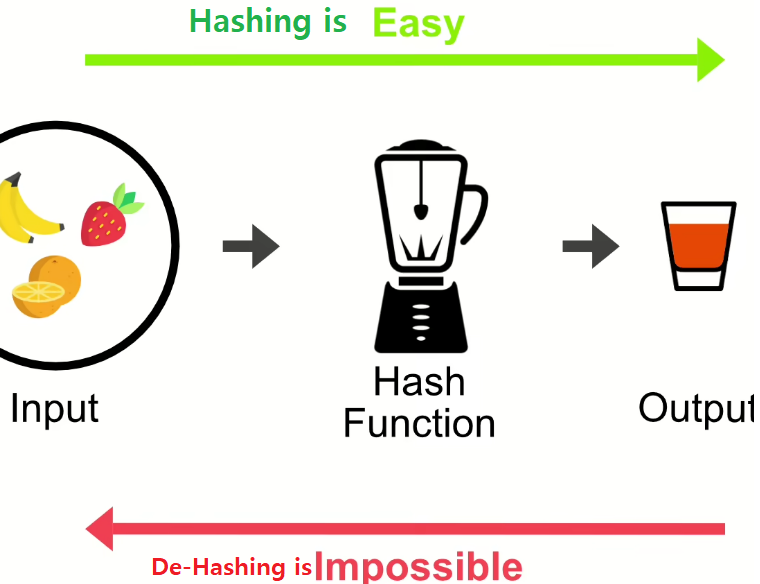
Generally everyone can see the data which is in transmission so we should **encrypt the data before transmitting over wire/ wirelessly**



## Hashing and De hashing

Hashing is a one way cryptography and it is irreversible

SHA-256 is nothing but secure hashing algorithm



Once u hash u cant reverse that, ex:- Hash 256, once passwords are hashed u can’t un hash that

Passwords hashed can’t be de-hashed/reversed/decrypted

### Symmetric vs. asymmetric encryption

Symmetric encryption:- With same key we will encrypt and decrypt (in real time both sender and receiver should use same key for encryption and decryp)

Symmetric encryption Algorithms are – AES (Advanced encryption algorithm), DES, 3DES, Blowfish, RC4, ChaCha20

Asymmetric encryption:

This is mostly used in real time

Every one in the network knows about Public key because all public keys are kept in central repository

and private key is known only to owner

|  |  |
| --- | --- |
| - | * we will encrypt the data with public key before sending (public key cant decrypt the encrytped one) * we can only decrypt the data with private key after receiving   Everybody will have access to public key & only intended user will have private key  Asymmetric encryption algorithms are 🡪 RSA (Max key size = 4096 Bits) |

SSL TLS network security

|  |  |
| --- | --- |
| and this symmetric key will be encrypted and send server decrypts with its own private key and it will get the symmetric key |  |

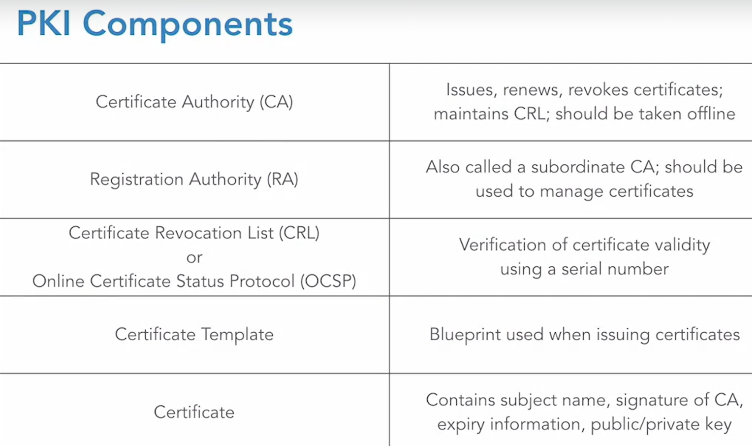
Algorithmns AES, DES,RSA

AES and DES (Data encryption standard) are algorithms

AES(Advanced encryption standard) 128 bit means =block size is 128, algorithm will encrypt into 128 bits (1 block) sets

|  |  |
| --- | --- |
| algorithms are symmetric key | algorithms are Asymmetric key |
| AWS,DES | RSA(rivest shamir, adleman) , ECC (elliptic curve cryptography) |

**Pki components**



|  |  |
| --- | --- |
| sample extracted jks content  Owner: CN=example.com, OU=IT, O=MyCompany, L=City, ST=State, C=IN  Issuer: CN=MyCA, O=CertificateAuthority, C=US | CN = common name it indicates to whom the certificate was issued to |

Pki = public key infrastructure = collection of digital security certificates

Certificate will contain the public keys , private keys can also be stored in separate file

**Certificate Request**

A Client or Server in order to acquire a valid certificate first need to create a **CSR** (Certificate Request) file which they need to submit to the **Certificate Authority** which can sign and issue a valid certificate

## 2.2 Certificate Authority (CA)

|  |  |
| --- | --- |
| Simple, authority means power, here certificate Authority means the power to issue the certificate,  So certificate authority is the one who will Issue the Certificates to the Entities (Servers, Clients, Websites)  CA sometime may include a physical visit by them to the business premises and then issue the public certificate  And certificate authority is the one who can confirm whether it’s the same Website which it claims it is.  A Certificate Authority is a trusted organization who confirm the authenticity of a website (Server). Their primarily responsibilities are,  CA will issue a cert to server by signing the cert with its private key (like how we sign the cheq with our own private signature) |  |
| CA can even Issues,renew,revoke the certificate and it maintains CRL (certificate revocation list )  Registration authority is the one who is a subordinate to CA – he also can do whatever CA can do  CA will issue cert only for limited period of time – after that either they have to renew/ they should revoke old cert and issue a new cert  CA will revoke the old cert based on the serial num of old cert  Few of the Certificate Authorities are [VeriSign](https://www.verisign.com/), [DigiCert](https://www.digicert.com/) |  |

.

## 2.3 Certificate

|  |  |
| --- | --- |
| Why cert- these cert are used to secure network traffic /encrypting the data transfer  Cert can be issued to users, smartphoness, apps,  Since CA issues the certificates the CA can verify whether it’s valid certificate or not or whether it is signed by them or not  This cer is same like an id card that contains (to whom this id card is issued), who issued id card, till when its valid, employee num  Every cert contains this infor🡪 cert serial num,subject name (to whom this cert is issued-web site url for whom this cert is issued), sign of issuing CA(who issued this cert), expiry info, public / private key, cert usage details (for which pur this cert has been issued )   * Here this public key is used to encrypt the data (so that this encrypted data will be transmitted ) * Every certificate will have 1 serial number (CA before issuing a cert it will assign a num to that cert) to uniquely identify a cert (like PAN to each individual) so that later if that cert is expired it will block that cert with based on that cert id * with this num only we will identify whether the cert had been already revoked or not & this serial num is used to revoke the cert * It contain cert usage details – to know for what purpose this cert had been issued   CA will sign every cert with its private key, | above is the cert issued to canara bank |
| how to check if cert is valid or not:- by using its cert serial number  The Online Certificate Status Protocol (OCSP) is an Internet protocol used to check the revocation status of digital certificates, such as those used for SSL/TLS connections. It's an alternative to Certificate Revocation Lists (CRLs) and allows clients to query a CA's OCSP server to determine if a certificate is valid or revoked.  With this OCSP we can query if that cert is valid or revoked |  |
| renewal vs re issuance  generally when u are using venafi cert management portal ,when the old cert is expired, u don’t need to re-issue a new certificate, (re-issuance is needed only when private key is lost )  we just need to renew the old certificate   * Every Cert will have expiry date, Once the cert are expired browsers will detect that we are using expired cert, it will result in loss of belief | |

## 2.3 SSL vs TLS

SSL, Stands for, Server Socket Layer however TLS, Stand for, Transport Layer Security. SSL is just the older name.

SSL and TLS are security protocols that are used to secure the network communication (Data in transit must be encrypted using any ssl certificate / key algorithm)

SSL is almost old and deprecated

TLS= more updated version of SSL, but both will do same thing of encrypting the data transfer

SSL (Secure socket layer) is standard technology for securing an internet connection by **encrypting the data** that is being sent between website and browser

It prevents hackers from seeing or stealing any information transferred

TLS (Transport layer security) is a protocol that helps secure communication over computer network, **TLS is also used to encrypt the communication** between web app and server

So data is always encrypted during transit

Both functionalities are same, but internal working implementations are different, like SSL uses different hashing function and TLS uses different hashing function

SSL uses hashing function such as MD5, SDH-1 but these are old sometimes different for diff inputs result hash is same

Hi=110

Bye=110

Whereas TLS uses SHA-256 algorithm, diff inputs always gives diff hash values

|  |  |
| --- | --- |
| SSL | TLS |
| Uses data encryption standards DES, RC4 | Uses AES standard |

We cant call as SSL/ TLS certificate – we should call as PKI cert,

## 2.4 Public/Private keys

* **Public Key** is used to Encrypt the data. (Think of certificates. No harm passing the public key, certificate, to anyone publicly)
* **Private Key** is used to Decrypt the data, encrypted by Public key.
* Private Key is like a password which should never cross the boundary of a system who has generated it. Consider it very secure. Think of it a system password which you don’t even share with your manager or any other person in authority.

## 2.5 MTLS (Mutual TLS)/ 2 way authn

Mutual means both, so MTLS means both client and server will mutually verify others identity /other certificates,

Like client will verify server certificate and server will verify client certificate with the help of CA who can check the certificate

So in postman when u are hitting an MTLS URL u need to attach certificate

Because both server and client needs to verify others certificate

**How Client will validate server certificate???**

Simple client will go and ask the certificate authority (the one who issued the certificate) saying whether it had issued that certificate to that website or not

Generally in one way communication,

1. Client will verify the servers certificate to confirm whether its talking to right client or not
2. And server accepts all clients, whereas in MTLS server will also validate the client certificate

Whereas in MTLS, not only client, whereas server also will verify the client authenticity

1. While making a call to the server, Client uses its ***Certificate*** and ***Client*** ***Key*** for the communication which is validated at the Server side to ensure the identity of client. Server consults Trust Store whether it’s a valid client to be given access or not. Please note, ***Client*** ***Key*** does not travel over network, it’s a private key and used to decrypt the data over the TCP tunnel.

## 2.6 key store and trust store

Key store contains our key and certificate

What is Trust store – trust store is used to trust only some websites

Importing other party appln to trust store

Assume if our appln want to allow 1dstr application, then we should import the 1dstr app certificate (we need to import the certificate that has been issued to 1dstr) into our server,

jKS file (java key store)

Java key store file is a repository of security/ cryptographic keys & certificates and private keys (**means that jks file is already having private keys and public keys** ) and other cryptgraphic artifacts

with these public keys we will encrypt the data so that always data in transit will be encrypted , nobody can tap and see the infor without public/private keys

this jks file is used to manage and protect these sensitive items (even if u want to open jks file u need passwords)

ensuring they are securely stored and used for operations like authentication, encryption, and digital signatures

* **Finally jks file will have private key and the trust store doesn’t contain any kind of private keys**

Keystores are often used in applications like Java, Android, and web servers to handle secure communication and data protection

* Inside jks file only certficate will be present

Key Features of a Keystore:

* **Secure Storage:**

Keystores provide a secure environment for storing cryptographic keys, making it difficult to extract them from the device or application. (without password u cant even open the jks file)

* **Key Management:**

They allow for the management of different types of keys, including private keys, public keys, and certificates.

* **Authentication and Encryption:**

Keystores contain public and private keys which are used to encrypt data in transit, ensuring that data and communication are secure.

* **Digital Signatures:**

They enable the creation and verification of digital signatures, ensuring the integrity and authenticity of data.

* **Key Restriction:**

Keystores can restrict when and how keys can be used, such as requiring user authentication for key use or restricting keys to use only in certain cryptographic modes.

Cacert (trust store)

Cacert means – Certificate authority’s cert (this is the cert of the ca /certificate authority)

This cacert needs to be present at both server and client side

|  |  |
| --- | --- |
|  | The cacerts file is a trusted keystore used by Java to store the certificates of Certificate Authority (CA).  It helps Java verify the authenticity of SSL/TLS connections by verifying the certificates presented by servers.  Think of it as a list of trusted entities that Java can rely on to verify the identity of a server.  ex:- assume we are client, while talking with server, server will give some cert to us, to verify if that cert is valid or not,  now client should talk to CA, CA can confirm whether the cert is original or not, he can confirm bec he only has issued that  we will compare that certificate serial id with the serial ids present in that trust store  It contains a collection of **trusted Certificate Authority (CA) certificates**. Think of CAs as trusted third parties that issue digital certificates to websites and other entities to verify their identity. |

Java codes

The security package really provides two APIs: one for users of security algorithms(developers), and another for the implementors or providers of these algorithms.

Gen/Extrct public &private key

|  |  |
| --- | --- |
| In realtime we will get the private key and public key from the jks file | Get the private key from jks file |
| we need to pass the algorithm name so that relative implementation will be given to us  KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");  or  DSAKeyPairGenerator myDSAGen =  (DSAKeyPairGenerator)KeyPairGenerator.getInstance("DSA");  keyGen.initialize(2048); // key size  KeyPair keyPair = keyGen.generateKeyPair();  PrivateKey privateKey = keyPair.getPrivate();  PublicKey publicKey = keyPair.getPublic(); | // Load the JKS file into a KeyStore object  KeyStore ks = KeyStore.getInstance("JKS");  ks.load(new FileInputStream("path/to/jks/file"), "keystorepassword".toCharArray());  // Here keysotre password is the password that is used to open that jks file  // Retrieve the private key from the KeyStore object  PrivateKey key = (PrivateKey) ks.getKey("alias", "privatekeypassword".toCharArray()); |

Verifying the data is not tampered

So for this we should send both original data and hashed data (middle men can change the data , but he cant change the hash, qq- what if he changes both the data and hash? ) and after receiving both orginal and hashed data at end

Since de-hashing is impossible Receiver will hash the orginal data and he will compare with the received hash value, if the both hashes are diff then it means data is not tampered

In real time- we will send the

* original data in payload body and
* a signature in header (which is generated using SHA algorithm and we will sign with private key which is extracted from cert)

here since hacker doesnt have any cert with him, so he cant get original private key and hence he cant sign

ex:- assume hacker tampered the data , in this case he should replace the our signature with new signature for tampered data ,

since he don’t have cert, he cant get original private key , hence he cant get sign (he can hash the data with SHA algorithm but he cant sign)

 The data is hashed (e.g., SHA-256) to create a fixed-size digest.

 The hash is encrypted with the sender’s **private key** — that’s the digital signature. (while signing we will use our private key)

 The recipient uses the sender’s **public key** to verify the signature:

* They decrypt the signature to get the hash.
* They also hash the received data themselves.
* If the two hashes match, the data is authentic and unchanged.

Signing the data

Note:- message can be signed only with private key and we can get the private key from jks file (means sender will use his private key to sign the message)

Digital signatures are used to sign messages and data, so that the receiver can verify the identity of the sender.

Signatures are often implemented using public/private key pairs—a **message is signed using the sender’s private key**, and the signature can be verified on the other end using the corresponding **public key**.

The interfaces provided for generating and using digital signatures include Key, KeyGenerator, KeyPairGenerator, Signature, and MessageDigest.

**Signing** (by the sender)

* Compute a **hash** (e.g., SHA-256) of the message.
* Encrypt the hash with the **private key** → this is the digital signature.
* Send the message + signature to the receiver.

(if u don’t encrypt the hash with private key, then middle men will replace original data with tamepered data and original hash with tampered data hash(hash created for tampered data ) and these tampered data and tam hash will be sent, but if u sign it with private key,since the hacker will not have the private key , he cant sign the data , **so signing with private key is most important** ) the only difference between original sender and hacker is the private key

Why this hashing?- sender can send the data which is encrpted with public key na, since hacker doesn’t have private key he cant decrypt the data right, then why should we hash the original data?

2️⃣ **Verification** (by the receiver)

* Receiver computes the **hash** of the received origianl message.
* Decrypts the signature using the sender’s **public key** → gets the original hash.
* If both hashes match → the signature is valid (data is authentic and untampered).

Why middlemen cant tamper the data

(deep analysis – middle men will also have the public key –he can decrypt the hash with public key and then he will get the decrypted hash and he already original data- now he saw the original data and hash

But he cant tamper – bec assume he changed original data and with tampered data he generated hash but he cant sign bec he don’t have private key,

That’s why we are signing , since the middle men will not have the private key he cant sign, since he cant sign he wont tamper the data at all , hence signing is important

**User will sign the data for Integrity**

* It proves that the data **has not been altered** since it was signed.
* Even a single bit change will cause the signature verification to fail. (ex if data is modified in transit the signature will be changed )
* Example: When you download a file, you can check its signature to ensure no tampering occurred in transit.

Note- anyone can hash the data even hacker, but what a hacker cant do is signature, since hacker doesn’t have cert he cant get private key so he cant do sign ,

He may sign with some random private key but not the original key

<https://learning.oreilly.com/library/view/java-distributed-computing/1565922069/ch05s05.html#JDP-CH-5-SECT-5.2>

read this chapter

note:- In java we will use signature class to generate the signature

|  |  |
| --- | --- |
| import java.security.\*;  public class DigitalSignatureExample {  public static void main(String[] args) throws Exception {  // 1. Generate RSA key pair  KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");  keyGen.initialize(2048); // key size  KeyPair keyPair = keyGen.generateKeyPair();  PrivateKey privateKey = keyPair.getPrivate();  PublicKey publicKey = keyPair.getPublic();  System.out.println("Keys generated!");  // Example data to sign  String data = "This is the data to sign";  byte[] dataBytes = data.getBytes("UTF8");  // 2. Sign the data – here we will hash the data & sign with our private key  here SHA means secure hashing algo, means data will be hashed and we will sign with private key  Signature signer = Signature.getInstance("SHA256withRSA");  signer.initSign(privateKey); //(in real time private key is extracted from certificate)  signer.update(dataBytes);  byte[] signatureBytes = signer.sign();  System.out.println("Data signed!");  // 3. Verify the signature  Signature verifier = Signature.getInstance("SHA256withRSA");  verifier.initVerify(publicKey);  verifier.update(dataBytes);  boolean isVerified = verifier.verify(signatureBytes);  System.out.println("Signature valid? " + isVerified);  }  } |  A digital signature proves **who created or sent the data**.     Example: A software developer signs an app with their private key — users can verify it’s really from that developer.  The KeyPairGenerator instance needs to be initialized before it can generate keys, using the initialize() method with a key size and optionally a source of randomness. The key size is interpreted differently for different algorithms, and the provider may use precomputed parameter sets or create new parameter sets. Using the KeyPair [4](https://learning.oreilly.com/library/view/java-distributed-computing/1565922069/ch05s02.html#source-card-4)[5](https://learning.oreilly.com/library/view/java-distributed-computing/1565922069/ch05s02.html#source-card-5)[6](https://learning.oreilly.com/library/view/java-distributed-computing/1565922069/ch05s02.html#source-card-6) The KeyPair object can be used for cryptographic operations, such as encryption and decryption.  The public key can be used to encrypt data, and the private key can be used to decrypt the data. For example:  Extracting the private key from jks file – in real time jks java keystore file will have many keys mapped to separate alias names  KeyStore keyStore = KeyStore.getInstance("JKS");  keyStore.load(new java.io.FileInputStream(keystorePath), keystorePassword.toCharArray());// here pass is used to open the file  // Retrieve the key  since jks file will have many keys we have to tell which key to extract  Key key = keystore.getKey(alias, keyPassword.toCharArray()); |

Chat gpt simple procedure

|  |  |
| --- | --- |
| PrivateKey myKey = ... // Retrieve my private key  byte[] data = ... // Get the data to be signed  Signature mySig = Signature.getInstance("RSA");  mySig.initSign(myKey);  mySig.update(data);  byte[] signedData = mySig.sign(); |  |

SHA-256

|  |  |
| --- | --- |
| **SHA-256** stands for **Secure Hash Algorithm 256-bit**. It’s a **cryptographic hash function** that takes any input data and produces a fixed-size, unique **256-bit (32-byte)** hash (digest).  **Example:**   * Input: "Hello world" * SHA-256 Hash: a591a6d40bf420404a011733cfb7b190... (64 hex characters, always 256 bits)  ✅ **Key properties of SHA-256**   1. **Deterministic**: The same input always produces the same hash. 2. **Fixed length**: The output is always 256 bits **(32-byte)**, no matter how big or small the input is. 3. **One-way function**: It’s practically impossible to get the original input from the hash. 4. **Collision-resistant**: It’s infeasible to find two different inputs with the same hash. 5. **Avalanche effect**: A small change in input results in a completely different hash.  ✅ **What is SHA-256 used for?** 🔐 **Common uses:**   * Digital signatures → You hash data first, then sign the hash. * Password storage → Store only the hash, not the plain password. * Blockchain → Bitcoin and many cryptocurrencies rely on SHA-256. * File integrity → Verify that a file hasn’t been tampered with. | import java.security.MessageDigest;  public class SHA256Example {  public static void main(String[] args) throws Exception {  String data = "Hello world";  MessageDigest digest = MessageDigest.getInstance("SHA-256");  byte[] hash = digest.digest(data.getBytes("UTF-8"));  // Convert bytes to hex  StringBuilder hexString = new StringBuilder();  for (byte b : hash) {  String hex = Integer.toHexString(0xff & b);  if (hex.length() == 1) hexString.append('0');  hexString.append(hex);  }  System.out.println("SHA-256 hash: " + hexString.toString());  }  } |
|  |  |