**AES and RSA**

**Symmetric vs Asymmetric Encryption**

Symmetric encryption involves using a single key to encrypt and decrypt data, while asymmetric encryption uses two keys - one public and one private - to encrypt and decrypt data.

**Symmetric Key**: A symmetric key is one that may be used to encrypt and decode data. This implies that to decrypt information, the same key that was used to encrypt it must be utilized.

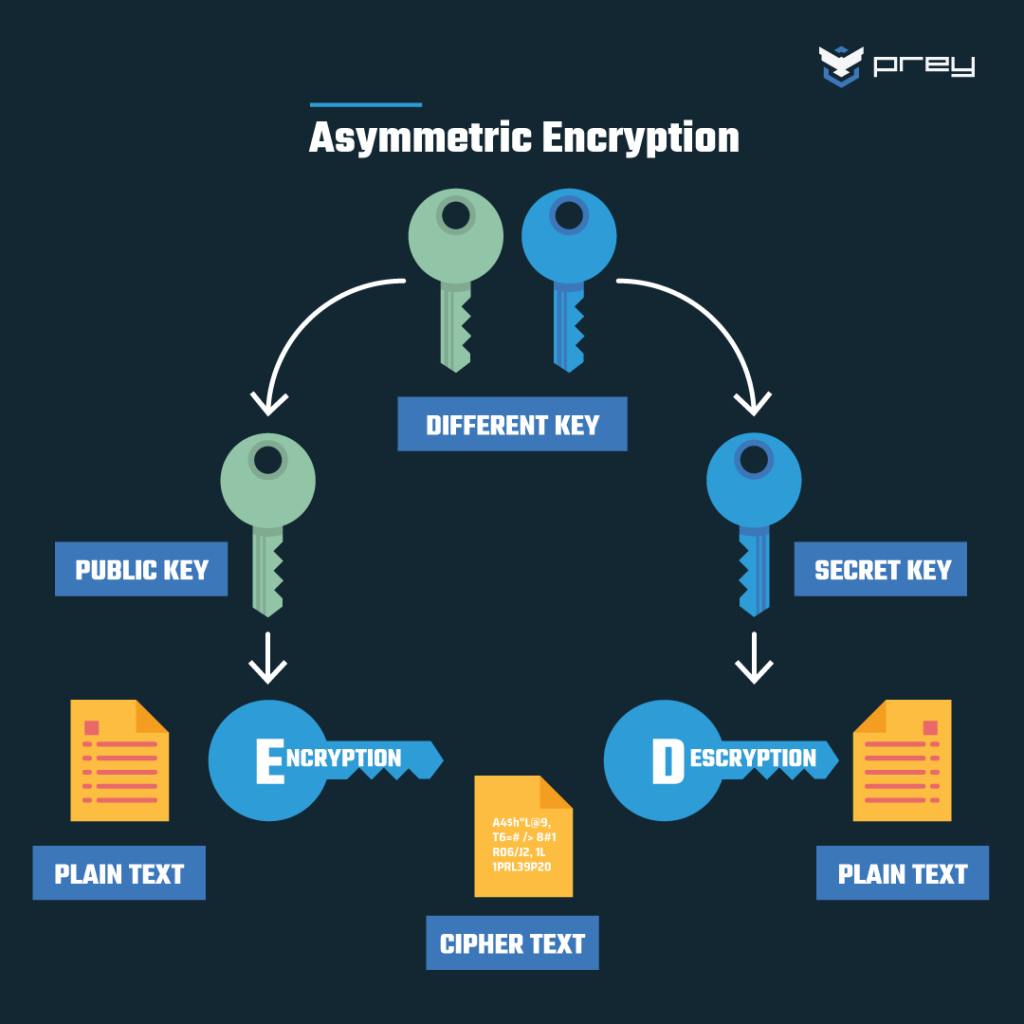
A diagram of keys and folders

Description automatically generated

Examples:

Secure messaging apps (e.g., WhatsApp, Signal): These messaging platforms use symmetric encryption, such as the Signal Protocol, to ensure that messages are encrypted end-to-end, allowing only the intended recipients to read the content.

**Asymmetric Key**: Asymmetric keys are the cornerstone of Public Key Infrastructure (PKI), an encryption technique that requires two keys, one to lock or encrypt the plaintext and another to unlock or decrypt the cyphertext. Neither key performs both functions.



Examples:

Digital signatures for document authenticity and integrity (e.g., Adobe Sign, DocuSign): Asymmetric encryption is also used in digital signature services like Adobe Sign and DocuSign. When a user signs a document digitally, their private key is used to create a unique signature. The recipient can then verify the authenticity and integrity of the document using the sender's public key. This process ensures the document hasn't been tampered with and confirms the identity of the signer.

**AES (Advanced Encryption Standard):**

AES is a symmetric encryption algorithm, meaning the same key is used for both encryption and decryption. It is widely used for securing sensitive data, such as file encryption, data at rest, and communication between devices. AES operates on fixed-size blocks of data (128 bits by default) and supports key sizes of 128, 192, or 256 bits.

Key Points:

Symmetric: Same key for encryption and decryption.

Efficiency: Efficient and fast, making it suitable for large amounts of data.

Demo Project was done to illustrate AES Encryption

**Dependencies used for the project:**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

AesEncrypt Class:

Cipher object is created for encryption and decryption in AES. The cipher is initialized to ENCRYPT\_MODE using the public key. The .doFinal method will encrypt the plain text and stores it in byte. The encrypted byte is later converted to String(cipherText)

public static String encrypt (byte [] plaintext, SecretKey key) throws Exception{

Cipher cipher = Cipher.getInstance("AES");

cipher.init(Cipher.ENCRYPT\_MODE, key);

byte [] encryptedByte= cipher.doFinal(plaintext);

String str= Base64.getEncoder().encodeToString(encryptedByte);

return str;

}

AesDecrypt Class:

Again, Cipher object is used for decryption of data. The cipher is initialized to DECRYPT\_MODE using the same public key. The encrypted cipherText is again converted to PlainText using doFinal.

public static String decrypt (String ciphertext, SecretKey key) throws Exception

{

Cipher cipher = Cipher.getInstance("AES");

cipher.init(Cipher.DECRYPT\_MODE, key);

return new String (cipher.doFinal( Base64.getDecoder().decode(ciphertext)));

}

Main class:

Generate the 128 bit key for Encryption and Decryption of data.Convert the plaintext to bytes and pass in the AesEncrypt.encrypt() method and store it in ciphertext. Thus, we got the encrypted String(cipherText). Pass the cipherText in AesDecrypt.decrypt() method. The cipherText will be again decrypted. We will get the plain text at final.

public static void main(String[] args) throws NoSuchAlgorithmException, Exception {

KeyGenerator keyGenerator = KeyGenerator.getInstance("AES"); //AES alogotithm

keyGenerator.init(128, new SecureRandom()); //128 bits key generation(only 128,192,256 bit key can be generated)

SecretKey key = keyGenerator.generateKey(); //generate the key

byte [] plaintext = "This is a secret Message”. getBytes();

String ciphertext = AesEncrypt.encrypt(plaintext, key);

String decryptedText = AesDecrypt.decrypt(ciphertext, key);

System.out.println("\nPlaintext: " + new String(plaintext));

System.out.println("\nCiphertext: " + new String(ciphertext));

System.out.println("\nDecrypted text: " + new String(decryptedText));

}

**RSA (Rivest–Shamir–Adleman):**

RSA is an asymmetric encryption algorithm, meaning it uses a pair of keys: a public key for encryption and a private key for decryption. It is commonly used for securing communication channels, especially during the key exchange phase, and for digital signatures. RSA relies on the mathematical difficulty of factoring large prime numbers.

Key Points:

Asymmetric: Uses a pair of public and private keys.

Key Exchange: Commonly used in key exchange protocols, such as HTTPS.

Digital Signatures: Used for creating and verifying digital signatures.

Key Sizes: Typically operates with key sizes of 2048, 3072, or 4096 bits.

RSA class:

Generate the key pair using the KeyPairGenerator where each key is of 1024 bits. The key pair contains the privatekey and a public key. Initialize the keys to “publicKey” and “privateKey” using getPrivate() and getPublic() methods. The exceptions which are occurring are ignored.

private PrivateKey privateKey;

private PublicKey publicKey;

public RSA () {

try {

KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");

generator.initialize(1024);

KeyPair pair = generator.generateKeyPair();

privateKey = pair.getPrivate();

publicKey = pair.getPublic();

} catch (Exception ignored) {

}

}

The String is passed to encrypt the data. The string is coverted to bytes.An instance of Cipher is used for encrypting the data. Initialize the cipher to ENCRYPT\_MODE using the public key. The .doFinal method will encrypt the data and gives the encrypted byte. The encrypted byte is further converted to String using encode () method.

public String encrypt (String message) throws Exception {

byte [] messageToBytes = message.getBytes();

Cipher cipher = Cipher.getInstance("RSA");

cipher.init(Cipher.ENCRYPT\_MODE, publicKey);

byte [] encryptedBytes = cipher.doFinal(messageToBytes);

return encode(encryptedBytes);

}

public String encode(byte [] data) {

return Base64.getEncoder().encodeToString(data); //binarydata(byte array) to string

}

The EncryptedString(cipherText) is coverted to bytes using decode () method. The cipher object is used for decrypting data. The cipher is initialized to DECRYPT\_MODE using the private key. The decrypted Message is converted to bytes and assigned to decryptedMessage and it is converted to String using UTF8 algorithm.

public String decrypt (String encryptedMessage) throws Exception {

byte [] encryptedBytes = decode(encryptedMessage);

Cipher cipher = Cipher.getInstance("RSA");

cipher.init(Cipher.DECRYPT\_MODE, privateKey);

byte [] decryptedMessage = cipher.doFinal(encryptedBytes);

return new String (decryptedMessage, "UTF8"); //Each byte will be encoded to character.

}

public byte [] decode (String data) {

return Base64.getDecoder(). decode(data);

}

Main Class

RSA class’s object is created and encrypt method is called where the given string is encrypted and the encryptedMessage is passed while calling decrypt method. The exceptions which are occurring are ignored.

public static void main (String [] args) {

RSA rsa = new RSA ();

try {

String encryptedMessage = rsa.encrypt("Hello World");

String decryptedMessage = rsa.decrypt(encryptedMessage);

System.out.println("Encrypted: " + encryptedMessage);

System.out.println("Decrypted: " + decryptedMessage);

//rsa.showKeys();

} catch (Exception ignored) {

}

}