Cryptography means convert text into unreadable form and again convert it to readable form.

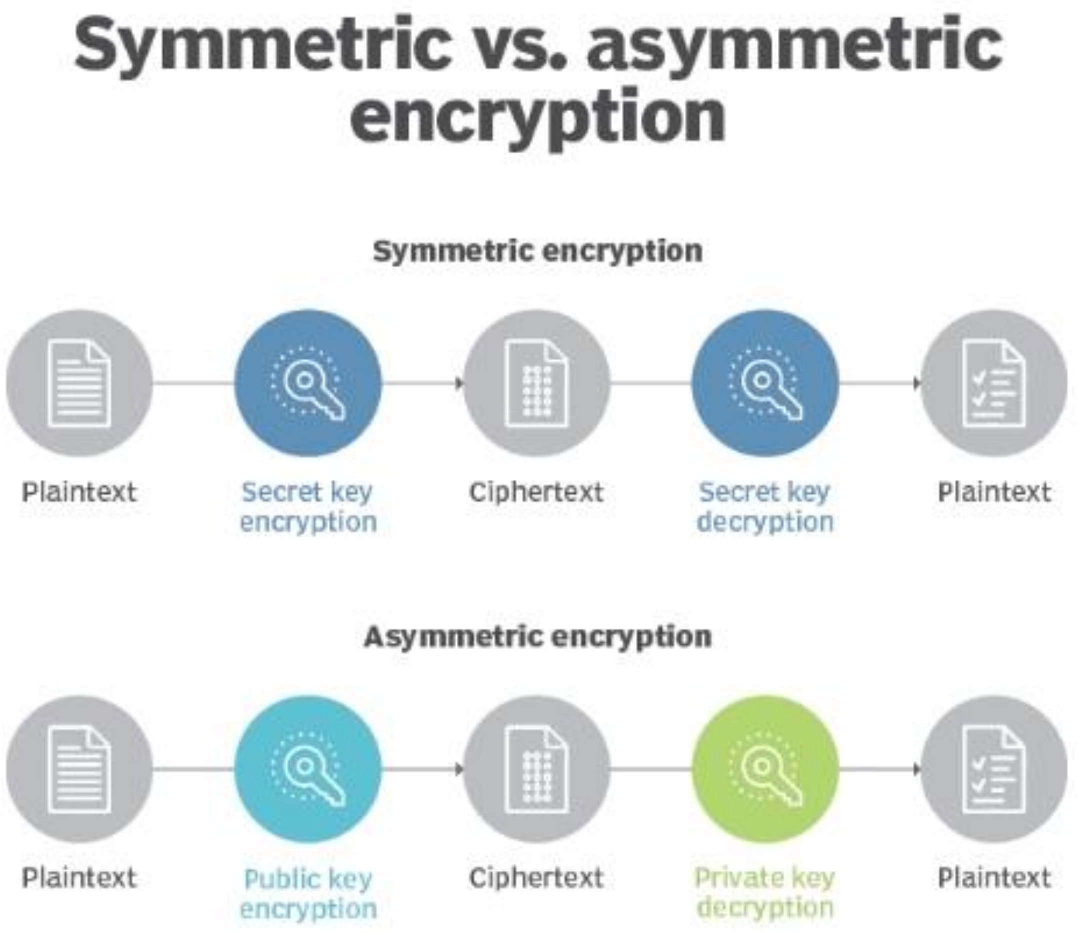
Cryptography has two parts :

Encryption – Convert original text to unreadable form.

Decryption – Convert unreadable form text or encrypted text to original text.

There are two types of Cryptography Algorithms.

1. Symmetric or Single Key Encryption Algorithm
2. Asymmetric or Two Way Encryption Algorithm which has two keys (Private Key and Public Key)



Example of Symmetric or Single Key Encryption Technique is AES Algorithm

AES means Advanced Encryption Standard.

Following is the code for AES Algorithm in Java

```
package crypto;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.KeySpec;
import java.util.Base64;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
/**
 *
 * @author raman
 */
public class aes {
     private static final String SECRET_KEY = "123456789";
    private static final String SALTVALUE = "abcdefg";

    /* Encryption Method */
    public static String encrypt(String strToEncrypt)
    {
    try
    {
      /* Declare a byte array. */
      byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
      IvParameterSpec ivspec = new IvParameterSpec(iv);
      /* Create factory for secret keys. */
      SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
      /* PBEKeySpec class implements KeySpec interface. */
      KeySpec spec = new PBEKeySpec(SECRET_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 256);
      SecretKey tmp = factory.generateSecret(spec);
      SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");
      Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
      cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);
      /* Retruns encrypted value. */
      return Base64.getEncoder()
.encodeToString(cipher.doFinal(strToEncrypt.getBytes(StandardCharsets.UTF_8)));
    }
    catch (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException |
InvalidKeySpecException | BadPaddingException | IllegalBlockSizeException |
NoSuchPaddingException e)
    {
      System.out.println("Error occured during encryption: " + e.toString());
    }
    return null;
    }
     public static String decrypt(String strToDecrypt)
    {
    try
    {
      /* Declare a byte array. */
      byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
      IvParameterSpec ivspec = new IvParameterSpec(iv);
      /* Create factory for secret keys. */
      SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
      /* PBEKeySpec class implements KeySpec interface. */
      KeySpec spec = new PBEKeySpec(SECRET_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 256);
      SecretKey tmp = factory.generateSecret(spec);
```

```
      SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");

      Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");

      cipher.init(Cipher.DECRYPT_MODE, secretKey, ivspec);

      /* Retruns decrypted value. */

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

    }

    catch (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException |
InvalidKeySpecException | BadPaddingException | IllegalBlockSizeException |
NoSuchPaddingException e)

    {

      System.out.println("Error occured during decryption: " + e.toString());

    }

    return null;

    }

    public static void main(String args[])

    {

         /* Message to be encrypted. */

        String originalval = "AES Encryption";

        /* Call the encrypt() method and store result of encryption. */

        String encryptedval = encrypt(originalval);

        /* Call the decrypt() method and store result of decryption. */

        String decryptedval = decrypt(encryptedval);

        /* Display the original message, encrypted message and decrypted message on the console. */

        System.out.println("Original value: " + originalval);

        System.out.println("Encrypted value: " + encryptedval);

        System.out.println("Decrypted value: " + decryptedval);

    }

}
```

Asymmetric Key Encryption Algorithm

Asymmetric Key Encryption Algorithms use two key private and public key. Plain Text or Original Text is converted to cipher text or unreadable text using public key and private key is used to converted encrypted text to plain or original text.

Example of Asymmetric Encryption Algorithm is RSA Algorithm or (Rivest-Shamir-Adleman) Algorithm

Java Code for RSA Algorithm

Filename is **EncryptDecryptRSAUtil.java**

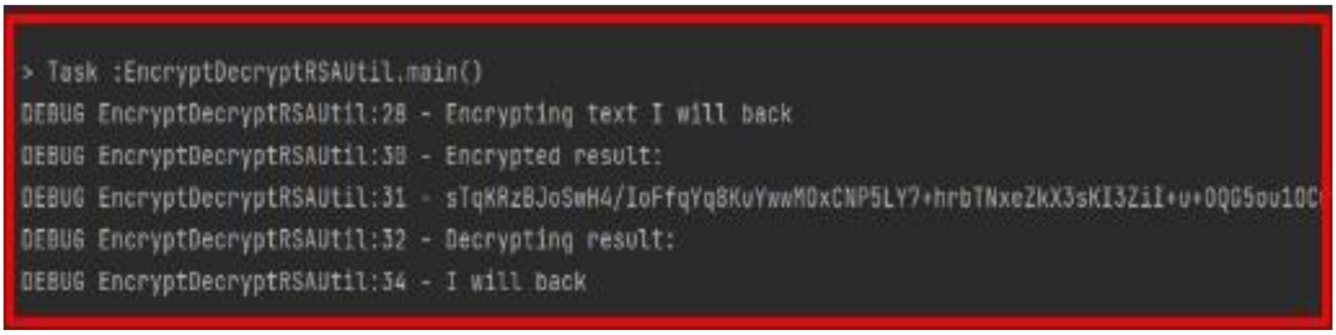
```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.crypto.Cipher;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.EncodedKeySpec;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;

public class EncryptDecryptRSAUtil {

    private static Logger log =
LoggerFactory.getLogger(EncryptDecryptRSAUtil.class);


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

        EncryptDecryptRSAUtil cryptoRSAUtil = new EncryptDecryptRSAUtil();
        String textToEncrypt = "I will back";
        log.debug("Encrypting text {} ", textToEncrypt);
        String encoded = cryptoRSAUtil.encode(textToEncrypt);
        log.debug("Encrypted result:");
        log.debug(encoded);
        log.debug("Decrypting result:");
        String decode = cryptoRSAUtil.decode(encoded);
        log.debug(decode);
    }


    public String encode(String toEncode) throws Exception {

        PublicKey publicKey = loadPublicKey();

        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);

        byte[] bytes =
cipher.doFinal(toEncode.getBytes(StandardCharsets.UTF_8));
        return new String(Base64.getEncoder().encode(bytes));
    }

    public String decode(String toDecode) throws Exception {

        PrivateKey privateKey = loadPrivateKey();

        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] bytes =
cipher.doFinal(Base64.getDecoder().decode(toDecode));
        return new String(bytes);

    }

    private PublicKey loadPublicKey() throws IOException,
NoSuchAlgorithmException, InvalidKeySpecException {

        // reading from resource folder
        byte[] publicKeyBytes =
getClass().getResourceAsStream("/key.pub").readAllBytes();

        KeyFactory publicKeyFactory = KeyFactory.getInstance("RSA");
        EncodedKeySpec publicKeySpec = new
X509EncodedKeySpec(publicKeyBytes);
        PublicKey publicKey =
publicKeyFactory.generatePublic(publicKeySpec);
        return publicKey;
    }

    private PrivateKey loadPrivateKey() throws IOException,
NoSuchAlgorithmException, InvalidKeySpecException {

        // reading from resource folder
        byte[] privateKeyBytes =
getClass().getResourceAsStream("/key.priv").readAllBytes();

        KeyFactory privateKeyFactory = KeyFactory.getInstance("RSA");
        EncodedKeySpec privateKeySpec = new
PKCS8EncodedKeySpec(privateKeyBytes);
        PrivateKey privateKey =
privateKeyFactory.generatePrivate(privateKeySpec);
        return privateKey;
    }


}
```

Filename is **KeyPairRSAGeneratorUtil.java**

```
 import
 java.io.File;
             import java.io.FileOutputStream;
             import java.io.IOException;
             import java.nio.file.Files;
             import java.security.*;
             import java.security.spec.EncodedKeySpec;
             import java.security.spec.InvalidKeySpecException;
             import java.security.spec.PKCS8EncodedKeySpec;
             import java.security.spec.X509EncodedKeySpec;

             public class KeyPairRSAGeneratorUtil {
```

```
    public static void main(String[] args) throws Exception {
        createKeys();
        loadPrivateKey();
        loadPublicKey();
    }

    private static void createKeys() throws NoSuchAlgorithmException,
IOException {

        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(2048);
        KeyPair kp = kpg.generateKeyPair();
        PrivateKey aPrivate = kp.getPrivate();
        PublicKey aPublic = kp.getPublic();

        try (FileOutputStream outPrivate = new
FileOutputStream("key.priv")) {
            outPrivate.write(aPrivate.getEncoded());
        }

        try (FileOutputStream outPublic = new
FileOutputStream("key.pub")) {
            outPublic.write(aPublic.getEncoded());
        }

        System.out.println("Private key: " + aPrivate.getFormat());
        // prints "Private key format: PKCS#8" on my machine

        System.out.println("Public key: " + aPublic.getFormat());
        // prints "Public key format: X.509" on my machine
    }

    private static PrivateKey loadPrivateKey() throws IOException,
NoSuchAlgorithmException, InvalidKeySpecException {

        File privateKeyFile = new File("key.priv");
        byte[] privateKeyBytes =
Files.readAllBytes(privateKeyFile.toPath());

        KeyFactory privateKeyFactory = KeyFactory.getInstance("RSA");
        EncodedKeySpec privateKeySpec = new
PKCS8EncodedKeySpec(privateKeyBytes);
        PrivateKey privateKey =
privateKeyFactory.generatePrivate(privateKeySpec);
        return privateKey;
    }

    private static PublicKey loadPublicKey() throws IOException,
NoSuchAlgorithmException, InvalidKeySpecException {
        File publicKeyFile = new File("key.pub");
        byte[] publicKeyBytes =
Files.readAllBytes(publicKeyFile.toPath());

        KeyFactory publicKeyFactory = KeyFactory.getInstance("RSA");
        EncodedKeySpec publicKeySpec = new
X509EncodedKeySpec(publicKeyBytes);
        PublicKey publicKey =
publicKeyFactory.generatePublic(publicKeySpec);
        return publicKey;
    }

}
```

Output of RSA Algorithm



```
> Task :EncryptDecryptRSAUtil.main()
DEBUG EncryptDecryptRSAUtil:28 - Encrypting text I will back
DEBUG EncryptDecryptRSAUtil:30 - Encrypted result:
DEBUG EncryptDecryptRSAUtil:31 - sTqKRzBJoSwH4/IoFfqYqBKuYwwMOxCNP5LY7+hrbTNxeZkX3sKI3ZiI+u+0QG5ou10C
DEBUG EncryptDecryptRSAUtil:32 - Decrypting result:
DEBUG EncryptDecryptRSAUtil:34 - I will back
```

Hash Functions

Hashing is the process of generating a value from a text or a list of numbers using a mathematical function known as a hash function.

A **Hash Function** is a function that converts a given numeric or alphanumeric key to a small practical integer value. The mapped integer value is used as an index in the hash table. In simple terms, a hash function **maps** a significant number or string to a small integer that can be used as the **index** in the hash table.

The pair is of the form **(key, value)**, where for a given key, one can find a value using some kind of a “function” that maps keys to values. The key for a given object can be calculated using a function called a hash function. For example, given an array A, if i is the key, then we can find the value by simply looking up A[i].

Example of Hash Function is SHA-1 Hash Function

Usage of SHA in Java

The Secure Hash Algorithms are used in digital signatures and their related certificates to establish a secure connection between the web server and its clients using explicit or implicit connections like **SSL** and **TSL** cryptographic protocols. Various applications also utilize SHA. They are:

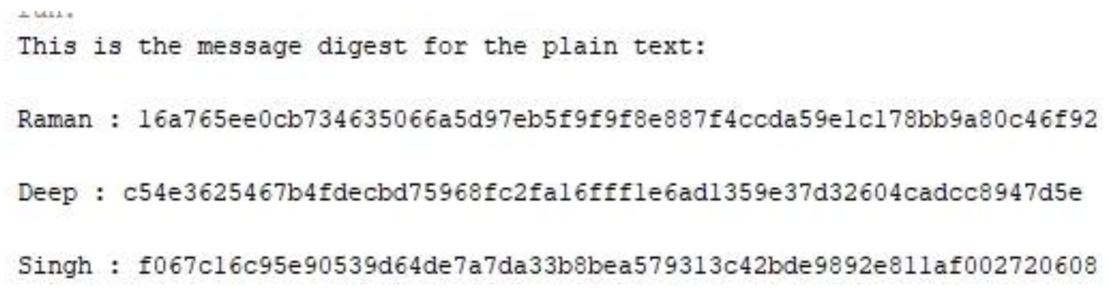
1. Secure Shell Protocol (SSH) applications.
2. Secure Multipurpose Internet Mail Extensions (S-MIME)
3. Intrusion Prevention System (IPS)

Java Code for SHA-1 Hash Function is :

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package crypto;

import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;


// This is a Java program to determine the usage of SHA Hashing.

class ShaExample {
   public static byte[] getSHA(String input) throws     NoSuchAlgorithmException {
 // The Static method " getInstance() " is called to initiate hashing with SHA
       MessageDigest md = MessageDigest.getInstance("SHA-256");

       // The static method called in the JAVA program
       // for calculating the message digest of a given input
```

```java
        // and results in an array of byte
        return md.digest(input.getBytes(StandardCharsets.UTF_8));
    }

    public static String toHexString(byte[] hash) {
        // calling the " BigInteger " function in JAVA programming language.
        BigInteger number = new BigInteger(1, hash);

        // Converting the message digest into a Hexa decimal value.
        StringBuilder hexString = new StringBuilder(number.toString(16));

        while (hexString.length() < 64) {
            hexString.insert(0, '0');
        }

        return hexString.toString();
    }

    // Main code containing the plain text to be converted.
    public static void main(String args[]) {
        try {
            System.out.println("This is the message digest for the plain text:");

            String str1 = "Raman";
            System.out.println("\n" + str1 + " : " + toHexString(getSHA(str1)));

            String str2 = "Deep";
            System.out.println("\n" + str2 + " : " + toHexString(getSHA(str2)));

            String str3 = "Singh";
            System.out.println("\n" + str3 + " : " + toHexString(getSHA(str3)));
        }
        // using the catch keyword for determining the wrong message digest algorithms
        catch (NoSuchAlgorithmException e) {
            System.out.println("The Exception for wrong hashing algorithm: " + e);
        }
    }
}
```

Output



```
This is the message digest for the plain text:

Raman : 16a765ee0cb734635066a5d97eb5f9f9f8e887f4ccda59e1c178bb9a80c46f92

Deep : c54e3625467b4fdecbd75968fc2fa16fff1e6ad1359e37d32604cadcc8947d5e

Singh : f067c16c95e90539d64de7a7da33b8bea579313c42bde9892e811af002720608
```