1. **Alice has sent a secret message "KKVMIR" to bob which was encrypted using additive cipher with key 2. Identify and Implement a suitable algorithm to find the plain text that was sent by Alice to Bob.**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the encrypted message and the key
        System.out.println("Enter the encrypted message:");
        String encryptedMessage = scanner.next();
        System.out.println("Enter the key (shift value):");
        int key = scanner.nextInt();

        // Decrypt the message
        String decryptedMessage = caesarDecrypt(encryptedMessage, key);

        // Output the decrypted message
        System.out.println("Decrypted message: " + decryptedMessage);

        scanner.close();
    }

    // Caesar Cipher decryption method
    public static String caesarDecrypt(String ciphertext, int shift) {
        StringBuilder plaintext = new StringBuilder();
        for (int i = 0; i < ciphertext.length(); i++) {
            char ch = ciphertext.charAt(i);
            if (Character.isUpperCase(ch)) {
                ch = (char)(((ch - 'A' - shift + 26) % 26) + 'A');
            } else if (Character.isLowerCase(ch)) {
                ch = (char)(((ch - 'a' - shift + 26) % 26) + 'a');
            }
            plaintext.append(ch);
        }
        return plaintext.toString();
    }
}
```

2. **Alice wants to send a message “IITKGP” to bob very secretly. Alice immediately encrypts the message using additive cipher with key 2. Identify and Implement a suitable algorithm to send the message very securely to Bob.**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt for input message
        System.out.println("Enter the message to encrypt:");
        String message = scanner.nextLine();

        // Prompt for the Caesar Cipher key
        System.out.println("Enter the Caesar Cipher key:");
        int key = scanner.nextInt();

        // Encrypt the message using Caesar Cipher
        String encryptedMessage = caesarEncrypt(message, key);

        // Output the encrypted message
        System.out.println("Encrypted message: " + encryptedMessage);
        scanner.close();
    }

    // Caesar Cipher encryption method
    public static String caesarEncrypt(String plaintext, int shift) {
        StringBuilder ciphertext = new StringBuilder();
        for (int i = 0; i < plaintext.length(); i++) {
            char ch = plaintext.charAt(i);
            if (Character.isUpperCase(ch)) {
                ch = (char)(((ch - 'A' + shift) % 26) + 'A');
            } else if (Character.isLowerCase(ch)) {
                ch = (char)(((ch - 'a' + shift) % 26) + 'a');
            }
            ciphertext.append(ch);
        }
        return ciphertext.toString();
    }
}
```

3. Hello, My name is Bob, Lieutenant of Indian Airforce. I wish to attack the terrorist’s secret place. My message is "Bomb destroy at Toga". I want to send this message as an encrypted form with a block cipher, substitution cipher. Construct a 5x5 matrix with the key "attack". Implement a suitable algorithm to decrypt the message.
4. **A secret message encrypted using the polyalphabetic substitution cipher with a keyword of “SECURITY” results in the ciphertext "KRIJY RAVOL TTHJC IRXXE". The keyword is known to contain only uppercase English letters. Decrypt the message and determine the original plaintext.**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
```

```
    Scanner scanner = new Scanner(System.in);

    // Input ciphertext and keyword
    String ciphertext = "KRIJY RAVOL TTHJC IRXXE";
    String keyword = "SECURITY";

    // Decrypt the ciphertext
    String plaintext = vigenereDecrypt(ciphertext, keyword);

    // Output the plaintext
    System.out.println("Decrypted message: " + plaintext);

    scanner.close();
}

// Vigenere Cipher decryption
public static String vigenereDecrypt(String ciphertext, String keyword) {
    StringBuilder plaintext = new StringBuilder();
    for (int i = 0, j = 0; i < ciphertext.length(); i++) {
        char c = ciphertext.charAt(i);
        if (Character.isLetter(c)) {
            char keyChar = keyword.charAt(j % keyword.length());
            int shift = keyChar - 'A';
            if (Character.isLowerCase(c)) {
                plaintext.append((char)('a' + (c - 'a' - shift + 26) % 26));
            } else {
                plaintext.append((char)('A' + (c - 'A' - shift + 26) % 26));
            }
            j++;
        } else {
            plaintext.append(c);
            }
        }
        return plaintext.toString();
    }
}
```

5. **A plaintext was encrypted with a monoalphabetic cipher with a shift of 7 (A maps to H). The resulting ciphertext is:" Kvu'a qbknl h ivvr if paz jvcly". Decrypt the message and determine the plain text.**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the ciphertext
        System.out.println("Enter the ciphertext:");
        String ciphertext = scanner.nextLine();

        // Decrypt the ciphertext using Caesar Cipher with a shift of 7
        String plaintext = caesarDecrypt(ciphertext, 7);

        // Output the decrypted plaintext
        System.out.println("Decrypted plaintext: " + plaintext);

        scanner.close();
    }

    // Caesar Cipher decryption method
    public static String caesarDecrypt(String ciphertext, int shift) {
        StringBuilder plaintext = new StringBuilder();
        for (int i = 0; i < ciphertext.length(); i++) {
            char ch = ciphertext.charAt(i);
            if (Character.isLetter(ch)) {
                int base = Character.isUpperCase(ch) ? 'A' : 'a';
                ch = (char)(((ch - base - shift + 26) % 26) + base);
            }
            plaintext.append(ch);
        }
        return plaintext.toString();
    }
}
```

6. I am a kind of substitution cipher. Use the key “COPYRIGHT” to construct a key matrix of size 5*5 ,identify the type of cipher and encrypt the message “Server is under attack”.
7. **Bob wants to communicate with Alice through the secret matrix to transfer a message “Meet Me”. Bob uses little matrix manipulation to make the encryption more complex. Help Bob to encrypt the message.**

Key matrix = $$\begin{matrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{matrix}$$

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the message and the key matrix
        System.out.println("Enter the message:");
        String message = scanner.nextLine();

        int[][] keyMatrix = {
            {6, 24, 1},
            {13, 16, 10},
            {20, 17, 15}
        };

        // Encrypt the message
        String encryptedMessage = rowColumnarEncrypt(message, keyMatrix);
        System.out.println("Encrypted message: " + encryptedMessage);

        scanner.close();
    }

    public static String rowColumnarEncrypt(String message, int[][] keyMatrix) {
        StringBuilder encryptedMessage = new StringBuilder();

        int messageLength = message.length();

        // Calculate the number of rows needed for the matrix
        int numRows = (int) Math.ceil((double) messageLength /
keyMatrix[0].length);

        // Calculate the actual number of columns needed for the key matrix based on
the message length
        int actualKeyLength = (int) Math.ceil((double) messageLength / numRows);

        char[][] matrix = new char[numRows][actualKeyLength];

        // Fill matrix with message characters
        int messageIndex = 0;
        for (int row = 0; row < numRows; row++) {
            for (int col = 0; col < actualKeyLength; col++) {
                if (messageIndex < messageLength) {
                    matrix[row][col] = message.charAt(messageIndex);
                    messageIndex++;
                } else {
                    matrix[row][col] = ' '; // Pad with space if message is shorter than
matrix
                }
            }
        }

        // Encrypt by column according to key matrix
        for (int col = 0; col < actualKeyLength; col++) {
            int keyIndex = col % keyMatrix[0].length;
            for (int row = 0; row < numRows; row++) {
                encryptedMessage.append(matrix[row][keyIndex]);
            }
        }

        return encryptedMessage.toString();
    }
}
```

8. **I am a transposition cipher. Consider the following plaint text "Attack at Dawn" and encrypt the text using the key "Lemon". Find me and implement this transposition cipher for encryption.**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the message and the key
        System.out.println("Enter the message:");
        String message = scanner.nextLine();

        System.out.println("Enter the key:");
        String key = scanner.nextLine();

        // Encrypt the message
        String encryptedMessage = transpositionEncrypt(message, key);
        System.out.println("Encrypted message: " + encryptedMessage);

        scanner.close();
    }

    public static String transpositionEncrypt(String message, String key) {
        int keyLength = key.length();
        int messageLength = message.length();

        // Calculate number of rows required
        int numRows = (int) Math.ceil((double) messageLength / keyLength);

        // Create a character array to hold the encrypted message
        char[][] matrix = new char[numRows][keyLength];

        // Fill matrix with message characters
        int messageIndex = 0;
        for (int row = 0; row < numRows; row++) {
            for (int col = 0; col < keyLength; col++) {
                if (messageIndex < messageLength) {
                    matrix[row][col] = message.charAt(messageIndex);
                    messageIndex++;
                } else {
                    matrix[row][col] = ' '; // Pad with space if message is shorter than
matrix
                }
            }
        }

        // Create a StringBuilder to store the encrypted message
        StringBuilder encryptedMessage = new StringBuilder();

        // Arrange characters according to key
        for (int i = 0; i < keyLength; i++) {
            int keyIndex = (key.charAt(i) - 'A') % keyLength;
            for (int j = 0; j < numRows; j++) {
                encryptedMessage.append(matrix[j][keyIndex]);
            }
        }

        return encryptedMessage.toString();
    }
}
```

9. **ABC organization releases a secret message "TINESAXEOAHTFXHTLTHEYMAUIAIXTAPNGDLOSTNHMX". XYZ is a competitive organization tries to crack the code using "532164". Identify the algorithm and perform decryption for the same.**

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class Main {
    public static void main(String[] args) throws Exception {
        String ciphertext =
"TINESAXEOAHTFXHTLTHEYMAUIAIXTAPNGDLOSTNHMX";
        String key = "532164"; // The secret key

        // Convert the key to bytes
        byte[] keyBytes = key.getBytes("UTF-8");
        SecretKeySpec secretKey = new SecretKeySpec(keyBytes, "AES");

        // Initialize the cipher
        Cipher cipher = Cipher.getInstance("AES/ECB/NoPadding");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);

        // Decode the ciphertext from Base64
        byte[] ciphertextBytes = Base64.getDecoder().decode(ciphertext);

        // Decrypt the ciphertext
        byte[] plaintextBytes = cipher.doFinal(ciphertextBytes);
        String plaintext = new String(plaintextBytes, "UTF-8");

        System.out.println("Decrypted message: " + plaintext);
    }
}
```

10. **ABC organization releases a secret message "ATB RAL H ETFRYU XMLESOOES". XYZ is a competitive organization tries to crack the code taking number of rows as 3. Identify the algorithm and perform decryption for the same**

```
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the ciphertext:");
    String ciphertext = scanner.nextLine();

    System.out.println("Enter the number of rows:");
    int numRows = scanner.nextInt();

    String plaintext = rowColumnarDecrypt(ciphertext, numRows);
    System.out.println("Decrypted message: " + plaintext);

    scanner.close();
  }

  public static String rowColumnarDecrypt(String ciphertext, int numRows) {
    int numCols = (int) Math.ceil((double) ciphertext.length() / numRows);
    int missingChars = numRows * numCols - ciphertext.length();

    StringBuilder plaintext = new StringBuilder();
    int index = 0;
    for (int col = 0; col < numCols; col++) {
      for (int row = 0; row < numRows; row++) {
        if (row < numRows - missingChars || col >= numCols - missingChars) {
          int newIndex = col + row * numCols;
          if (newIndex < ciphertext.length()) { // Check before accessing character
            plaintext.append(ciphertext.charAt(newIndex));
          }
        }
      }
    }

    return plaintext.toString();
  }
```

```
}
```

11. **I am a symmetric key block cipher. I will process 64 – bits plain text with 56 – bits key. I have 16 rounds of processing. Identify who I am. Implement the encryption/ decryption mechanism using the following Cipher text “welcome to los angles”**

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.util.Base64;

public class Main {

  public static void main(String[] args) {
    try {
      String plainText = "welcome to los angles";
      // Shorten the secret key to 16 characters (recommended)
      String secretKey = "mySecretKey12345"; // 128-bit key

      // Encrypt
      String encryptedText = encrypt(plainText, secretKey);
      System.out.println("Encrypted: " + encryptedText);

      // Decrypt
      String decryptedText = decrypt(encryptedText, secretKey);
      System.out.println("Decrypted: " + decryptedText);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public static String encrypt(String plainText, String secretKey) throws Exception
{
    SecretKeySpec keySpec = new
SecretKeySpec(secretKey.getBytes(StandardCharsets.UTF_8), "AES");
    Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, keySpec);
byte[] encryptedBytes
=cipher.doFinal(plainText.getBytes(StandardCharsets.UTF_8));
    return Base64.getEncoder().encodeToString(encryptedBytes);
  }

  public static String decrypt(String encryptedText, String secretKey) throws
Exception {
    SecretKeySpec keySpec = new
SecretKeySpec(secretKey.getBytes(StandardCharsets.UTF_8), "AES");
    Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
    cipher.init(Cipher.DECRYPT_MODE, keySpec);
    byte[] decodedBytes = Base64.getDecoder().decode(encryptedText);
    byte[] decryptedBytes = cipher.doFinal(decodedBytes);
    return new String(decryptedBytes, StandardCharsets.UTF_8);
  }
}
```

12. **I am a symmetric key block cipher. I will process with cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits. I have rounds depending on the keys as 10, 12 or 14. Identify who I am. Implement the Decryption/encryption mechanism for the following plain text “welcome to los angles”.**

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
```

```
public class Main {

  public static void main(String[] args) {
    try {
      String plainText = "welcome to los angles";
      // Extend the secret key to 16 characters (not recommended for real use)
      String secretKey = "mySecretKey****"; // 128-bit key

      // Encrypt
      String encryptedText = encrypt(plainText, secretKey);
      System.out.println("Encrypted: " + encryptedText);

      // Decrypt
      String decryptedText = decrypt(encryptedText, secretKey);
      System.out.println("Decrypted: " + decryptedText);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static String encrypt(String plainText, String secretKey) throws Exception
{
    SecretKeySpec keySpec = new
SecretKeySpec(secretKey.getBytes(StandardCharsets.UTF_8), "AES");
    Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, keySpec);
    byte[] encryptedBytes =
cipher.doFinal(plainText.getBytes(StandardCharsets.UTF_8));
    return Base64.getEncoder().encodeToString(encryptedBytes);
  }
  public static String decrypt(String encryptedText, String secretKey) throws
Exception {
    SecretKeySpec keySpec = new
SecretKeySpec(secretKey.getBytes(StandardCharsets.UTF_8), "AES");
    Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
    cipher.init(Cipher.DECRYPT_MODE, keySpec);
    byte[] decodedBytes = Base64.getDecoder().decode(encryptedText);
    byte[] decryptedBytes = cipher.doFinal(decodedBytes);
    return new String(decryptedBytes, StandardCharsets.UTF_8);
  }
}
```

13. **Alice and Bob wants to communicate securely. They want to use block cipher technique with key size 64 bit is converted into 56 bit key. Identify the symmetric encryption method used by them and implement the encryption mechanism using the plain text "We are in danger".**

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.util.Base64;

public class Main {

    public static void main(String[] args) {
        try {
            String plainText = "We are in danger";
            String secretKey = "mySecretKey123456"; // 128-bit key

            // Encrypt
            String encryptedText = encrypt(plainText, secretKey);
            System.out.println("Encrypted: " + encryptedText);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static String encrypt(String plainText, String secretKey) throws
Exception {
        SecretKeySpec keySpec = new
SecretKeySpec(secretKey.getBytes(StandardCharsets.UTF_8), "AES");
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, keySpec);
        byte[] encryptedBytes =
cipher.doFinal(plainText.getBytes(StandardCharsets.UTF_8));
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }
}
```

14. **An asymmetric encryption algorithm is used to encrypt the message 87. Generate the required global parameters, public and private key to perform encryption.**

```
import java.math.BigInteger;
import java.util.Random;

public class RSAExample {

    private BigInteger P;
    private BigInteger Q;
    private BigInteger N;
    private BigInteger PHI;
    private BigInteger e;
    private BigInteger d;
    private int maxLength = 1024;
```

```java
    private Random R;

    public RSAExample() {
        R = new Random();
        P = BigInteger.probablePrime(maxLength, R);
        Q = BigInteger.probablePrime(maxLength, R);
        N = P.multiply(Q);
        PHI = P.subtract(BigInteger.ONE).multiply(Q.subtract(BigInteger.ONE));
        e = BigInteger.probablePrime(maxLength / 2, R);

        while (PHI.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(PHI) < 0) {
            e = e.add(BigInteger.ONE);
        }

        d = e.modInverse(PHI);
    }

    public static void main(String[] args) {
        RSAExample rsa = new RSAExample();
        String plaintext = "87"; // Convert the message to a number

        // Encrypt
        BigInteger ciphertext = rsa.encrypt(new BigInteger(plaintext));
        System.out.println("Ciphertext: " + ciphertext);

        // Decrypt
        BigInteger decrypted = rsa.decrypt(ciphertext);
        System.out.println("Decrypted: " + decrypted);
    }
    public BigInteger encrypt(BigInteger plaintext) {
        return plaintext.modPow(e, N);
    }

    public BigInteger decrypt(BigInteger ciphertext) {
        return ciphertext.modPow(d, N);
    }
}
```

15. **Implement a suitable Hash algorithm which takes the input of any length and produces the output of fixed length hash. Let the output be 160 bits. Transform the given plain texts "We are in danger " and "We " into equivalent hashed values.**

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Main {

    public static String calculateSHA1(String input) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-1");
            byte[] messageDigest = md.digest(input.getBytes());
            BigInteger no = new BigInteger(1, messageDigest);
            String hashText = no.toString(16);

            // Pad with leading zeros if needed
            while (hashText.length() < 40) {
                hashText = "0" + hashText;
            }
            return hashText;
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }

    public static void main(String[] args) {
        String plainText1 = "We are in danger";
        String plainText2 = "We";

        String hash1 = calculateSHA1(plainText1);
        String hash2 = calculateSHA1(plainText2);

        System.out.println("SHA-1 hash for \"" + plainText1 + "\": " + hash1);
        System.out.println("SHA-1 hash for \"" + plainText2 + "\": " + hash2);
    }
}
```

16. **In an Insecure world, communication can be carried out successfully through shared secret key. To establish a shared secret key, implement a suitable mechanism and exchange the secret keys between Bob and Alice.**

```
public class Main {

    public static void main(String[] args) {
        long P = 23; // Prime number (public key)
        long G = 5;  // Primitive root of P (public key)
        long a = 6;  // Private key for Alice
        long b = 15; // Private key for Bob

        // Calculate public values
        long x = calculatePower(G, a, P); // Alice's public value
        long y = calculatePower(G, b, P); // Bob's public value

        // Exchange public keys (x and y)

        // Calculate symmetric secret keys
        long ka = calculatePower(y, a, P); // Alice's secret key
        long kb = calculatePower(x, b, P); // Bob's secret key

        System.out.println("Shared secret key for Alice: " + ka);
        System.out.println("Shared secret key for Bob: " + kb);
    }

    public static long calculatePower(long base, long exponent, long modulus) {
        if (exponent == 1) {
            return base;
        } else {
            return (long) Math.pow(base, exponent) % modulus;
        }
    }
}
```

17. **Mr. Alex signs a document for a new project that has to be launched on 01.11.2019. Mr. Rohit, CEO needs to verify the document signed by Mr. Alex so that he can approve the project and launch it in the specified date. Suggest a suitable algorithm to Mr. Rohit to verify the signature.**

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.util.Base64;
```

```
public class Main {

    public static void main(String[] args) {
        try {
            // Generate key pair (public key and private key)
            KeyPairGenerator keyPairGenerator =
KeyPairGenerator.getInstance("RSA");
            keyPairGenerator.initialize(2048); // Key size (adjust as needed)
            KeyPair keyPair = keyPairGenerator.generateKeyPair();

            // Example: Mr. Alex signs the document
            String document = "Project details for launch on 01.11.2019";
            byte[] signature = createDigitalSignature(document, keyPair.getPrivate());

            // Example: Mr. Rohit verifies the signature
            boolean isVerified = verifyDigitalSignature(document, signature,
keyPair.getPublic());

            System.out.println("Signature verification result: " + isVerified);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static byte[] createDigitalSignature(String input, PrivateKey privateKey)
throws Exception {
        Signature signature = Signature.getInstance("SHA256withRSA");
        signature.initSign(privateKey);
        signature.update(input.getBytes());
        return signature.sign();
    }

    public static boolean verifyDigitalSignature(String input, byte[] signature,
PublicKey publicKey) throws Exception {
        Signature verifier = Signature.getInstance("SHA256withRSA");
        verifier.initVerify(publicKey);
        verifier.update(input.getBytes());
        return verifier.verify(signature);
    }
}
```