# Custom Cipher Based on DES

## encrypter.java

import java.util.\*;

import java.io.\*;

import java.nio.file.Files;

import java.nio.file.Paths;

class Block{

private Integer blockData;

private Long preBlockData;

private Long postBlockData;

public Long getPreBlockData() {

return preBlockData;

}

public void setPreBlockData(Long preBlockData) {

this.preBlockData = preBlockData;

}

public Long getPostBlockData() {

return postBlockData;

}

public void setPostBlockData(Long postBlockData) {

this.postBlockData = postBlockData;

}

public Block(List<Byte> blockBytes) {

this.blockData = this.bytesToInt(blockBytes.get(0), blockBytes.get(1),

blockBytes.get(2), blockBytes.get(3));

}

private Integer bytesToInt(Byte b1, Byte b2, Byte b3, Byte b4) {

return ((b1 & 0xFF) << 24) |

((b2 & 0xFF) << 16) |

((b3 & 0xFF) << 8) |

(b4 & 0xFF);

}

public Integer getBlockData() {

return blockData;

}

public void setBlockData(Integer blockData) {

this.blockData = blockData;

}

}

class Round{

final boolean isDebug = true;

final List<Integer> arrayOf32s = new ArrayList<Integer>(Arrays.asList(29, 18, 3, 0, 21, 20, 5, 19

, 13, 2, 25, 4, 8, 23, 14, 1, 6, 15, 12, 17, 28, 26, 24, 11, 27, 30, 9, 7, 16, 10, 22, 31));

Block rBlock;

Integer key;

public Round(Block nBlock, Integer nKey){

this.rBlock = nBlock;

this.key = nKey;

this.startRound();

}

private void startRound(){

this.performInitialPermutation();

this.performBitsSubstitution();

this.performKeyXOR();

this.performRailfenceTransposition();

this.performRandomness();

}

private void performInitialPermutation(){

Integer preShift = this.rBlock.getBlockData();

if (isDebug == true)

System.out.println("Performing Initial Permutation");

if (isDebug == true)

System.out.println("Pre\_\_shift: " + Integer.toString(preShift, 2) + " Length: " + Integer.toString(preShift, 2).length());

Integer afterShift = 0;

for (int i = 0; i <= 31; i++){

Integer bit = (preShift >> i) & 1;

int newPosition = arrayOf32s.get(i);

afterShift |= (bit << newPosition);

}

if (isDebug == true)

System.out.println("Post\_shift: " + Integer.toString(afterShift, 2) + " Length: " + Integer.toString(afterShift, 2).length());

rBlock.setBlockData(afterShift);

}

private void performBitsSubstitution(){

Dictionary<String, String> subTable = new Hashtable<>();

//can be derived from key

subTable.put("00", "01");

subTable.put("01", "11");

subTable.put("10", "00");

subTable.put("11", "10");

StringBuilder sb = new StringBuilder();

StringBuilder subString = new StringBuilder();

Integer beforeSub = rBlock.getBlockData();

for (int i = 0; i < 31; i += 2){

Integer bit\_r = (beforeSub >> i) & 1;

Integer bit\_l = (beforeSub >> (i + 1)) & 1;

String concat = Integer.toString(bit\_l) + Integer.toString(bit\_r);

//System.out.print(concat + " ");

sb.insert(0, concat);

subString.insert(0, subTable.get(concat));

}

//System.out.println("This is reversed");

if (isDebug == true) {

System.out.println("\nPerforming Substitution");

System.out.println("Before Subs: " + sb + " Length: " + sb.length());

System.out.println("After\_ Subs: : " + subString + " Length: " + subString.length());

}

rBlock.setBlockData(Integer.parseUnsignedInt(subString.toString(), 2));

}

private void performKeyXOR(){

Integer randomKey = 353452343;

if (isDebug == true)

System.out.println("\nPerforming Key XOR");

if (isDebug == true)

System.out.println("Before XOR: " + Integer.toString(rBlock.getBlockData(), 2) + " Length: " + Integer.toString(rBlock.getBlockData(), 2).length());

Integer xorVal = rBlock.getBlockData() ^ this.key;

if (isDebug == true)

System.out.println("After\_ XOR: " + Integer.toString(xorVal, 2) + " Length: " + Integer.toString(xorVal, 2).length());

rBlock.setBlockData(xorVal);

}

private void performRailfenceTransposition(){

Integer prefill = rBlock.getBlockData();

Random rdn = new Random(this.key);

Integer fenceKey = rdn.nextInt(8);

//System.out.println("Fence Key: " + fenceKey.toString() + Integer.toString(fenceKey, 2));

String[][] fence = new String[32][fenceKey];

for (int i = 0; i < 32; i++){

for (int j = 0; j < fenceKey; j++){

fence[i][j] = ".";

}

}

int c = 0;

for (int i = 31; i >= 0; i--){

Integer bit = (prefill >> i) & 1;

fence[31 - i][c] = Integer.toString(bit);

c++;

if (c == fenceKey){ c = 0; };

}

/\*System.out.println("After railfence");

for (int i = 0; i < fenceKey; i++){

for (int j = 0; j < 32; j++){

System.out.print(fence[j][i] + " ");

}

System.out.println("\n");

}\*/

StringBuilder rfStr = new StringBuilder();

for (int j = 0; j < fenceKey; j++){

for (int i = 0; i < 32; i++){

if (fence[i][j].equals(".") == false){

rfStr.append(fence[i][j]);

}

}

}

if (isDebug == true) {

System.out.println("\nPerforming RailFence Transposition");

System.out.println("After RailFence: "+rfStr.toString());

}

rBlock.setBlockData(Integer.parseUnsignedInt(rfStr.toString(), 2));

}

private String[] generateBitArray(long seed) {

Random random = new Random(seed);

String[] bitArray = new String[8];

for (int i = 0; i < 8; i++) {

int bit = random.nextInt(2);

bitArray[i] = Integer.toString(bit);

}

return bitArray;

}

private List<Integer> generateBitLocations(long seed){

List<Integer> pos = new ArrayList<Integer>(Arrays.asList(0 ,1, 2, 3, 4, 5, 6, 7));

//shuffle here

Collections.shuffle(pos, new Random(seed));

return pos;

}

private void performRandomness(){

StringBuilder randomizeStr = new StringBuilder();

Integer beforeAdd = rBlock.getBlockData();

for (int i = 31; i >= 0; i--){

Integer bit = (beforeAdd >> i) & 1;

randomizeStr.append(bit);

}

if (isDebug == true) {

System.out.println("\nPerforming Randomness:");

System.out.println("Before randomness: " + randomizeStr + " Length: " + randomizeStr.length());

}

List<Integer> randomizePos = this.generateBitLocations(this.key);

for (int i = 0; i < 8; i++){

//System.out.println(randomizePos.get(i) \* 4);

randomizeStr.insert(randomizePos.get(i) \* 4, "1");

}

if (isDebug == true)

System.out.println("After\_ randomness: " + randomizeStr.toString() + " Length: " + randomizeStr.toString().length());

this.finishRound(randomizeStr.toString());

}

private void finishRound(String longStr){

Long finalData = Long.parseLong(longStr, 2);

rBlock.setPostBlockData(finalData);

}

}

class Encryption{

private Integer key = 0;

private String plaintext = "";

private List<Block> pBlocks = new ArrayList<Block>();

public Encryption(String nKey, String nPlaintext){

this.plaintext = nPlaintext;

this.key = this.parseKey(nKey);

this.startAlgorithm();

}

private void startAlgorithm(){

this.createBlocks();

this.performRounds();

this.writeCipherDataToFile();

}

private void createBlocks(){

// byte[] bytesStream = this.plaintext.getBytes();

byte[] bytesStream = null;

try {

bytesStream = Files.readAllBytes(Paths.get("plaintextData.txt"));

}

catch (Exception e) { }

List<Byte> mainStream = new ArrayList<Byte>();

for (byte each: bytesStream){

mainStream.add((Byte)each);

}

for (int i = 0; i < mainStream.size(); i += 4){

List<Byte> blockBytes = new ArrayList<Byte>();

int end = Math.min(mainStream.size(), i + 4);

blockBytes = new ArrayList<>(mainStream.subList(i, end));

if (blockBytes.size() < 4){

while (blockBytes.size() < 4) {

Byte zeroByte = 0;

blockBytes.add(zeroByte);

}

}

Block nBlock = new Block(blockBytes);

this.pBlocks.add(nBlock);

}

}

private void performRounds(){

int ctr = 0;

for (Block iBlock: this.pBlocks){

System.out.println("Initiating Block Processing ID: " + Integer.toString(ctr) + "...\n");

Round iRound = new Round(iBlock, this.key);

System.out.println("Block Processed ID: " + Integer.toString(ctr) + "\n\n");

ctr += 1;

}

}

private void writeCipherDataToFile(){

try {

PrintWriter pw = new PrintWriter("encryptedData.txt", "UTF-8");

for (Block iBlock: this.pBlocks){

pw.println(Long.toString(iBlock.getPostBlockData(), 2));

}

pw.close();

System.out.println("Data written to file");

}

catch (Exception e){

}

}

private Integer parseKey(String nKey){

Integer parsedKey = Integer.parseInt(nKey);

return parsedKey;

}

}

@SuppressWarnings("unused")

public class encrypter{

public static void main(String[] args){

takeInput();

}

public static void takeInput(){

//String key = "432435432";

String plaintext = "Hello this is a test to write and read data from the file 1 2 3 4 , . \*) %\*&";

Scanner myScan = new Scanner(System.in);

System.out.println("Enter a Integer (max 32 bits): ");

String key = myScan.nextLine();

System.out.println("Entered key is: " + key);

Encryption encrypt = new Encryption(key, plaintext);

//System.out.println(Integer.parseInt("01100001011000100110001101100101",2));

}

}

## decrypter.java

import java.io.\*;

import java.util.\*;

class DeBlock{

private Integer blockData;

private Long preBlockData;

private Long postBlockData;

public Long getPreBlockData() {

return preBlockData;

}

public void setPreBlockData(Long preBlockData) {

this.preBlockData = preBlockData;

}

public Long getPostBlockData() {

return postBlockData;

}

public void setPostBlockData(Long postBlockData) {

this.postBlockData = postBlockData;

}

public DeBlock() {

super();

}

private Integer bytesToInt(Byte b1, Byte b2, Byte b3, Byte b4) {

return ((b1 & 0xFF) << 24) |

((b2 & 0xFF) << 16) |

((b3 & 0xFF) << 8) |

(b4 & 0xFF);

}

public Integer getBlockData() {

return blockData;

}

public void setBlockData(Integer blockData) {

this.blockData = blockData;

}

}

class DeRound{

final boolean isDebug = false;

final List<Integer> arrayOf32s = new ArrayList<Integer>(Arrays.asList(29, 18, 3, 0, 21, 20, 5, 19

, 13, 2, 25, 4, 8, 23, 14, 1, 6, 15, 12, 17, 28, 26, 24, 11, 27, 30, 9, 7, 16, 10, 22, 31));

DeBlock rBlock;

Integer key;

public DeRound(DeBlock nBlock, Integer nKey){

super();

this.rBlock = nBlock;

this.key = nKey;

this.startRound();

}

private void startRound(){

this.performRandomness();

this.performRailfenceTransposition();

this.performKeyXOR();

this.performBitsSubstitution();

this.performInitialPermutation();

}

private void performInitialPermutation(){

Integer preShift = this.rBlock.getBlockData();

if (isDebug == true)

System.out.println("Reverting Initial Permutation");

if (isDebug == true)

System.out.println("Pre\_\_shift: " + Integer.toString(preShift, 2) + " Length: " + Integer.toString(preShift, 2).length());

Integer oldNumber = 0;

for (int i = 0; i <= 31; i++){

int oldPosition = arrayOf32s.get(i);

Integer bit = (preShift >> oldPosition) & 1;

//System.out.println(i + " " + Integer.toString(bit, 2) + " " + Integer.toString(oldNumber, 2));

oldNumber |= (bit << i);

}

if (isDebug == true)

System.out.println("Post\_shift: " + Integer.toString(oldNumber, 2) + " Length: " + Integer.toString(oldNumber, 2).length() + "\n");

rBlock.setBlockData(oldNumber);

}

private void performBitsSubstitution(){

Dictionary<String, String> subTable = new Hashtable<>();

//can be derived from key

subTable.put("01", "00");

subTable.put("11", "01");

subTable.put("00", "10");

subTable.put("10", "11");

StringBuilder sb = new StringBuilder();

StringBuilder subString = new StringBuilder();

Integer beforeSub = rBlock.getBlockData();

for (int i = 0; i < 31; i += 2){

Integer bit\_r = (beforeSub >> i) & 1;

Integer bit\_l = (beforeSub >> (i + 1)) & 1;

String concat = Integer.toString(bit\_l) + Integer.toString(bit\_r);

//System.out.print(concat + " ");

sb.insert(0, concat);

subString.insert(0, subTable.get(concat));

}

//System.out.println("This is reversed");

if (isDebug == true) {

System.out.println("Reverting Substitution");

System.out.println("Before Subs: " + sb + " Length: " + sb.length());

System.out.println("After\_ Subs: : " + subString + " Length: " + subString.length() + "\n");

}

rBlock.setBlockData(Integer.parseUnsignedInt(subString.toString(), 2));

}

private void performKeyXOR(){

if (isDebug == true)

System.out.println("Performing Key XOR");

if (isDebug == true)

System.out.println("Before XOR: " + Integer.toString(rBlock.getBlockData(), 2) + " Length: " + Integer.toString(rBlock.getBlockData(), 2).length());

Integer xorVal = rBlock.getBlockData() ^ this.key;

if (isDebug == true)

System.out.println("After\_ XOR: " + Integer.toString(xorVal, 2) + " Length: " + Integer.toString(xorVal, 2).length() + "\n");

rBlock.setBlockData(xorVal);

}

private void performRailfenceTransposition(){

Integer prefill = rBlock.getBlockData();

Random rdn = new Random(this.key);

Integer fenceKey = rdn.nextInt(8);

//System.out.println("Fence Key: " + fenceKey.toString());

String[][] fence = new String[32][fenceKey];

for (int i = 0; i < 32; i++){

for (int j = 0; j < fenceKey; j++){

fence[i][j] = ".";

}

}

int c = 0;

for (int i = 31; i >= 0; i--){

fence[31 - i][c] = "0";

c++;

if (c == fenceKey){ c = 0; };

}

/\*for (int i = 0; i < fenceKey; i++){

for (int j = 0; j < 32; j++){

System.out.print(fence[j][i] + " ");

}

System.out.println("\n");

}\*/

Integer shifted = 0;

for (int i = 0; i < fenceKey; i++) {

for (int j = 0; j < 32; j++) {

if (fence[j][i].equals("0") == true) {

int bit = (prefill >> (31 - shifted)) & 1;

fence[j][i] = Integer.toString(bit);

shifted += 1;

}

}

}

int k = 0;

StringBuilder railDecr = new StringBuilder();

for (int i = 31; i >= 0; i--){

railDecr.append(fence[31 - i][k]);

k++;

if (k == fenceKey){ k = 0; };

}

/\*for (int i = 0; i < fenceKey; i++){

for (int j = 0; j < 32; j++){

System.out.print(fence[j][i] + " ");

}

System.out.println("\n");

}\*/

if (isDebug == true) {

System.out.println("Reverting RailFence Transposition");

System.out.println("After RailFence: "+railDecr.toString() + "\n");

}

rBlock.setBlockData(Integer.parseUnsignedInt(railDecr.toString(), 2));

}

private String[] generateBitArray(long seed) {

Random random = new Random(seed);

String[] bitArray = new String[8];

for (int i = 0; i < 8; i++) {

int bit = random.nextInt(2);

bitArray[i] = Integer.toString(bit);

}

return bitArray;

}

private List<Integer> generateBitLocations(long seed){

List<Integer> pos = new ArrayList<Integer>(Arrays.asList(0 ,1, 2, 3, 4, 5, 6, 7));

//shuffle here

Collections.shuffle(pos, new Random(seed));

return pos;

}

private void performRandomness(){

StringBuilder randomizeStr = new StringBuilder();

Long beforeAdd = rBlock.getPreBlockData();

for (int i = 39; i >= 0; i--){

Long bit = (beforeAdd >> i) & 1;

randomizeStr.append(bit);

}

if (isDebug == true) {

System.out.println("Performing Randomness:");

System.out.println("Before randomness: " + randomizeStr + " Length: " + randomizeStr.length());

}

List<Integer> randomizePos = this.generateBitLocations(this.key);

Collections.reverse(randomizePos);

for (int i = 0; i < 8; i++){

//randomizeStr.insert(randomizePos.get(i), "1");

randomizeStr.deleteCharAt(randomizePos.get(i) \* 4);

}

if (isDebug == true)

System.out.println("After\_ randomness: " + randomizeStr.toString() + " Length: " + randomizeStr.toString().length());

rBlock.setBlockData(Integer.parseUnsignedInt(randomizeStr.toString(), 2));

//this.finishRound(randomizeStr.toString());

}

private void finishRound(String longStr){

Long finalData = Long.parseLong(longStr, 2);

rBlock.setPostBlockData(finalData);

}

}

class Decryption{

private Integer key = 0;

private String plaintext = "";

private List<DeBlock> pBlocks = new ArrayList<DeBlock>();

public Decryption(String nKey){

this.key = this.parseKey(nKey);

this.startAlgorithm();

}

private void startAlgorithm(){

this.readCipherDataFromFile();

this.performRounds();

this.readDataFromBlocks();

}

private void performRounds(){

int ctr = 0;

for (DeBlock iBlock: this.pBlocks){

System.out.println("Initiating Block Processing ID: " + Integer.toString(ctr) + "...\n");

DeRound iRound = new DeRound(iBlock, this.key);

System.out.println("Block Processed ID: " + Integer.toString(ctr) + "\n\n");

ctr += 1;

}

}

private void readDataFromBlocks() {

StringBuilder plaintext = new StringBuilder();

for (DeBlock iBlock: this.pBlocks){

Integer blockData = iBlock.getBlockData();

Integer byte1 = blockData >> 24;

plaintext.append((char)(int)byte1);

Integer byte2 = (blockData >> 16) & 0xFF;

plaintext.append((char)(int)byte2);

Integer byte3 = (blockData >> 8) & 0xFF;

plaintext.append((char)(int)byte3);

Integer byte4 = blockData & 0xFF;

plaintext.append((char)(int)byte4);

}

System.out.println("100% decrypting data...\n");

System.out.println("Plaintext:\n\n" + plaintext.toString());

}

private void readCipherDataFromFile(){

try {

BufferedReader br = new BufferedReader(new FileReader("encryptedData.txt"));

String line = br.readLine();

while (line != null) {

//System.out.println("Read line: " + line);

DeBlock block = new DeBlock();

block.setPreBlockData(Long.parseLong(line, 2));

this.pBlocks.add(block);

line = br.readLine();

}

System.out.println("Finished reading data from file");

}

catch (Exception e){

}

}

private Integer parseKey(String nKey){

Integer parsedKey = Integer.parseInt(nKey);

return parsedKey;

}

}

public class decrypter {

public static void main(String[] args){

takeInput();

}

public static void takeInput(){

//String key = "432435432";

String plaintext = "abceefg";

Scanner myScan = new Scanner(System.in);

System.out.println("Enter a Integer (max 32 bits): ");

String key = myScan.nextLine();

System.out.println("Entered key is: " + key);

Decryption decrypt = new Decryption(key);

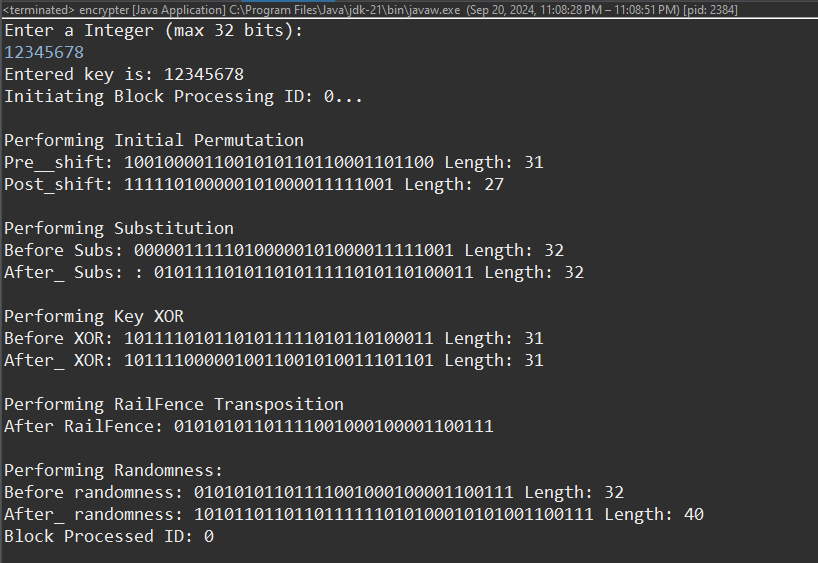
//System.out.println(Integer.parseInt("01100001011000100110001101100101",2));

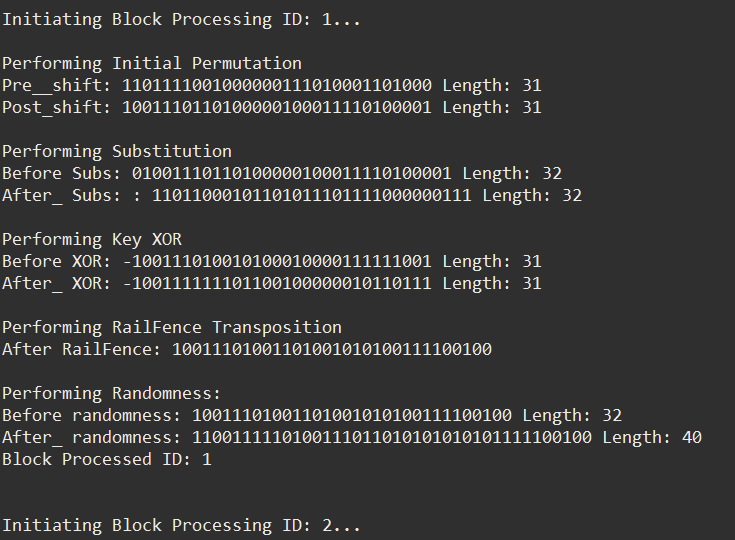
}

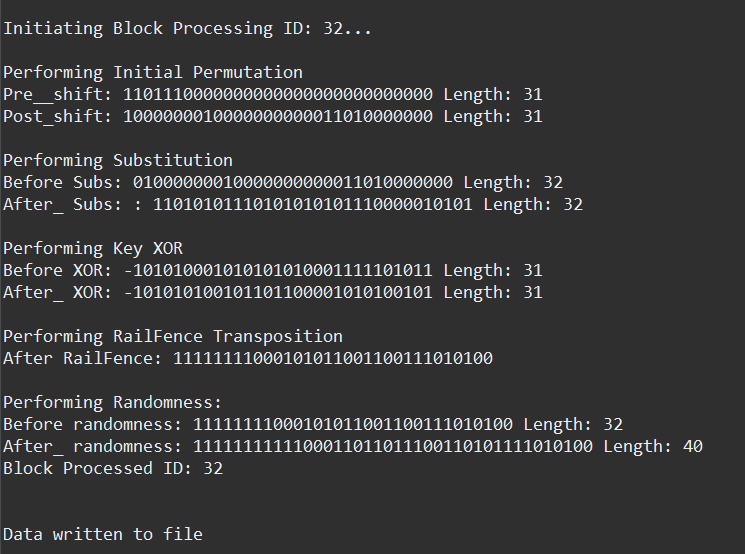
}

## Output

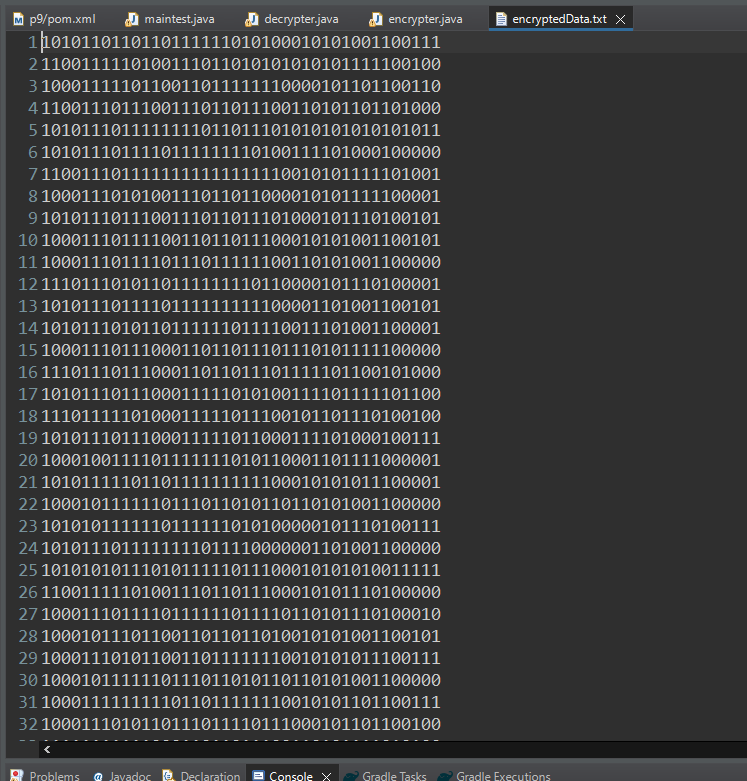
### While encryption







### encryptedData.txt file



### While decryption

