Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 2

**Компьютерная реализация СТБ П 34.101.31–2007**

Выполнил студент гр. 953501

Трофимук Г.А

Проверил

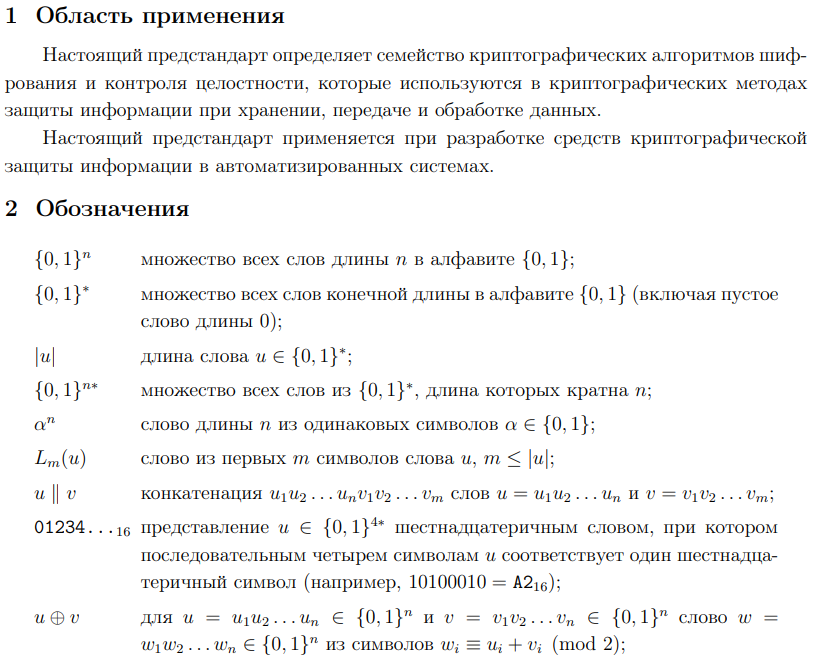
Протько М.И.

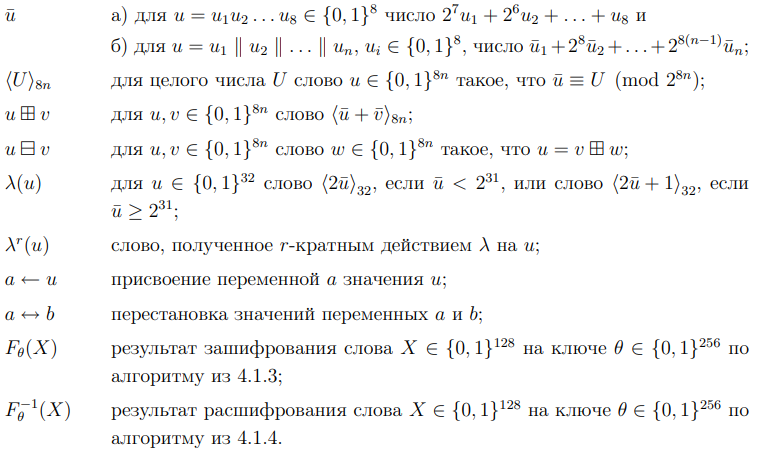
Минск, 2022

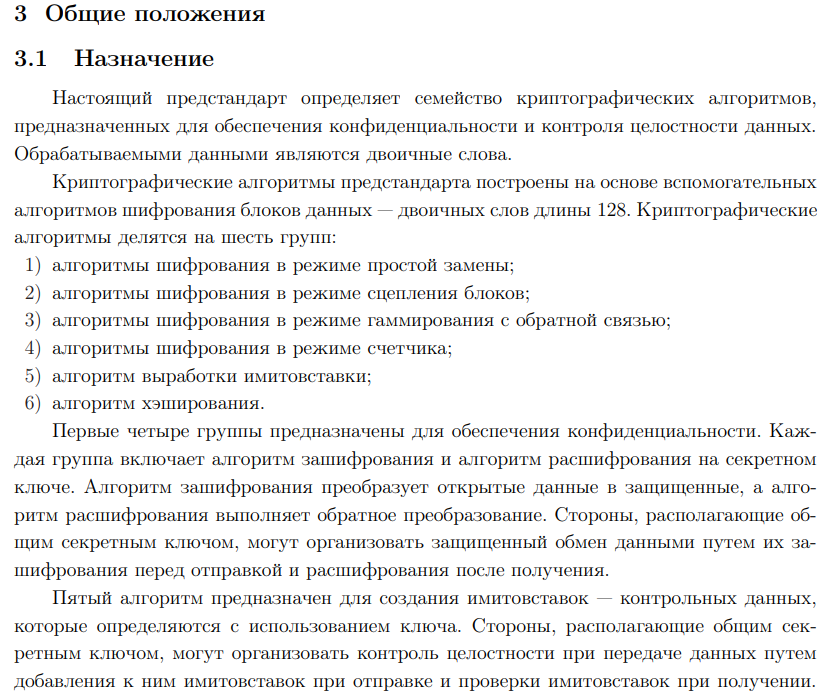
# 1. Введение

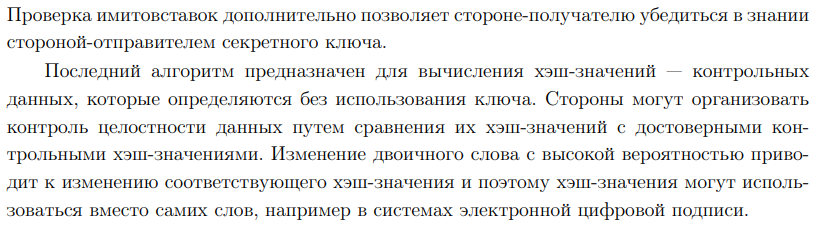
Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма СТБ П 34.101.31–2007.

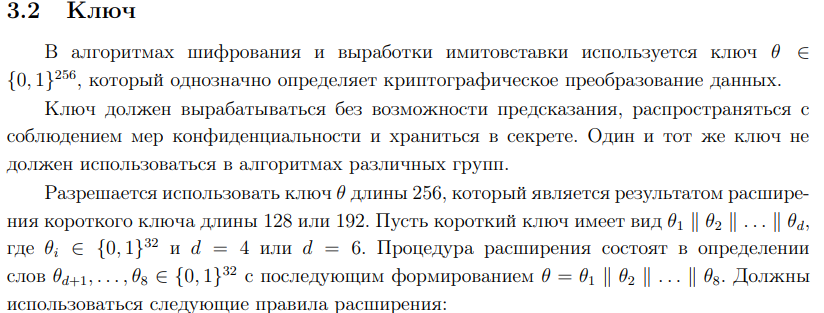
**2. Теоретические сведения**



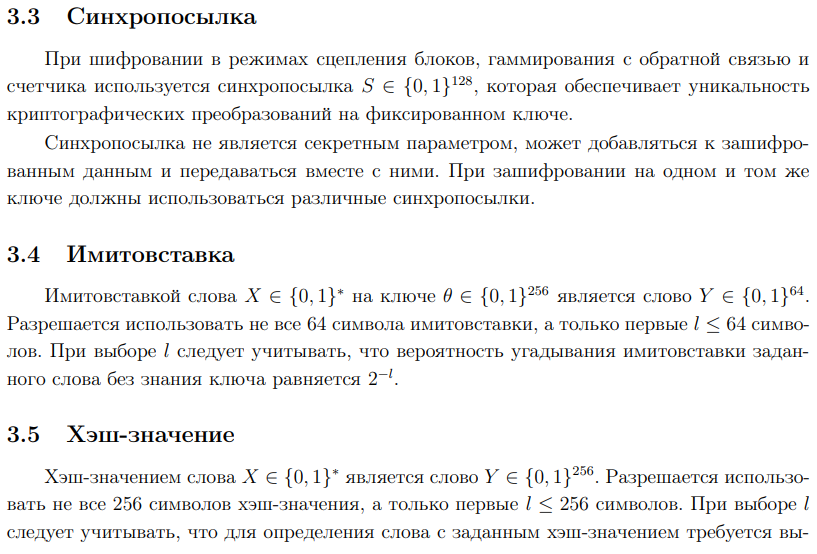


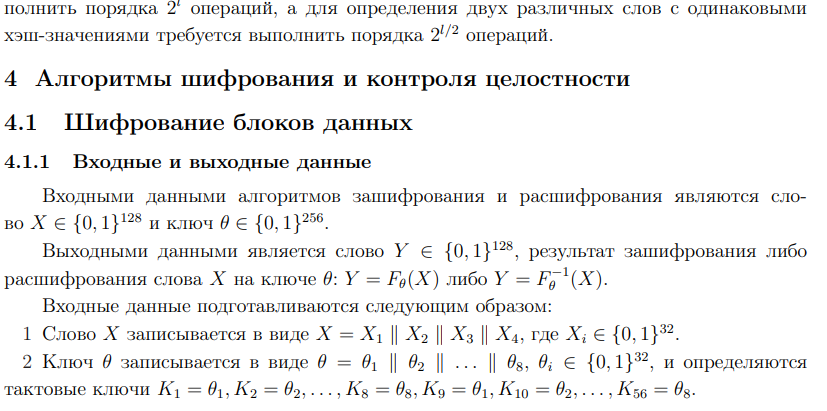


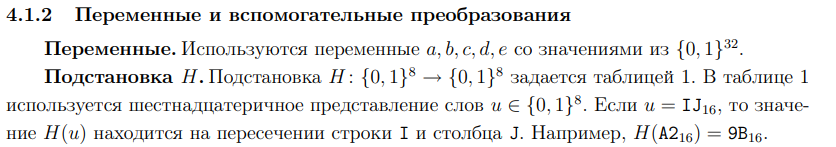


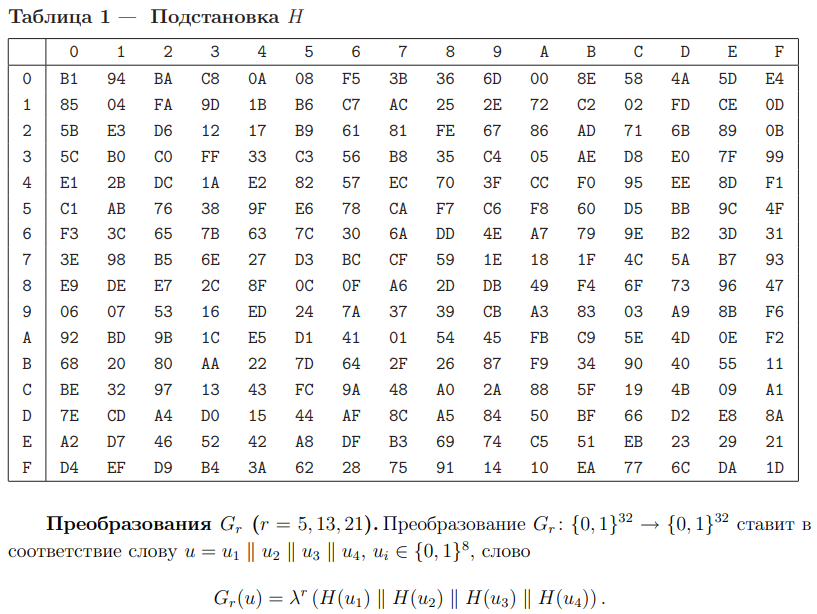


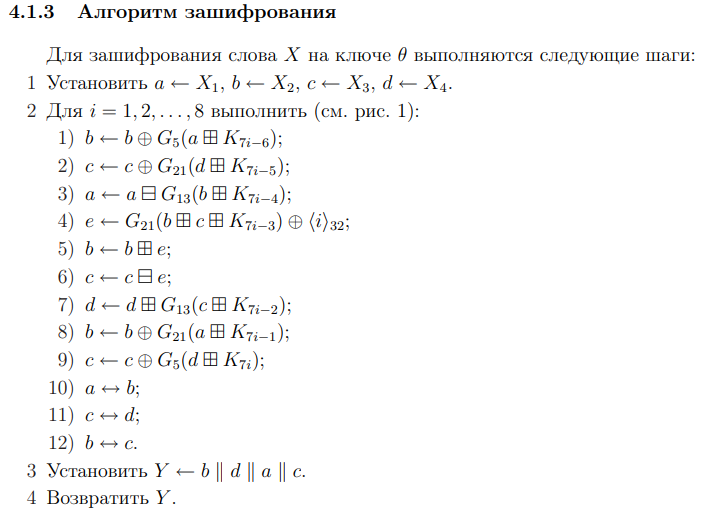


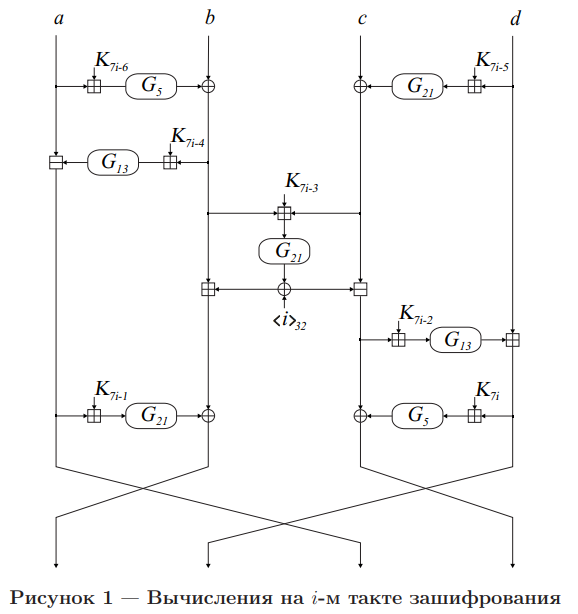


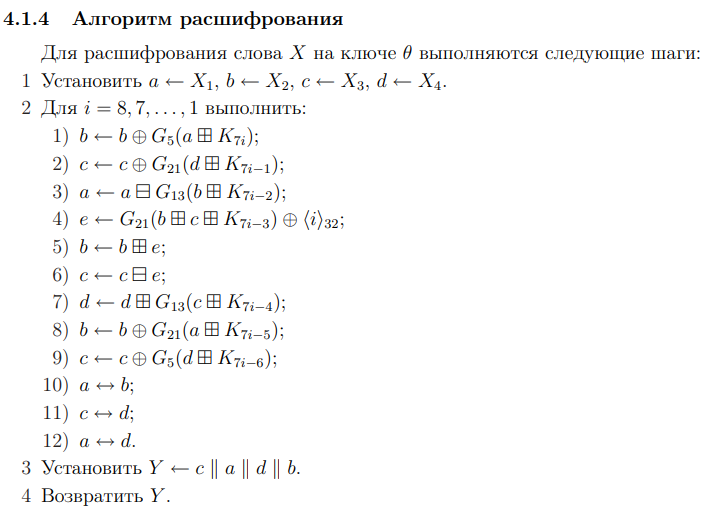


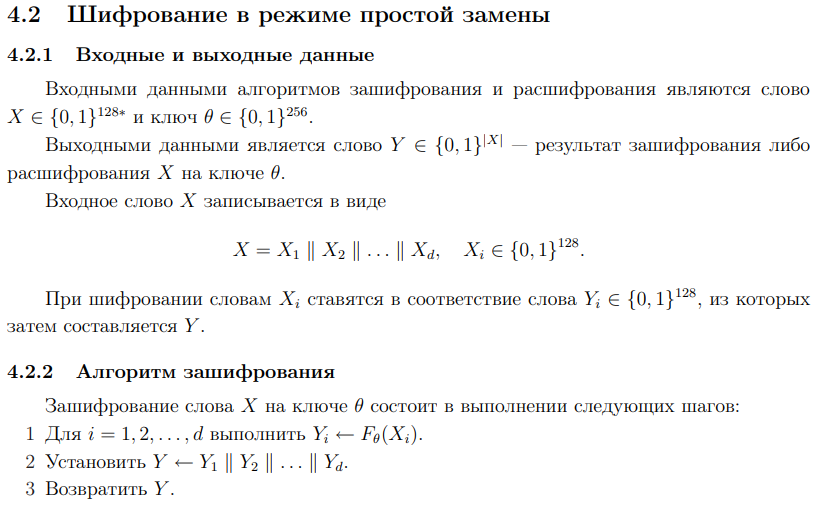


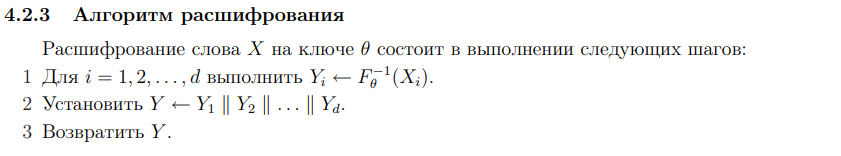


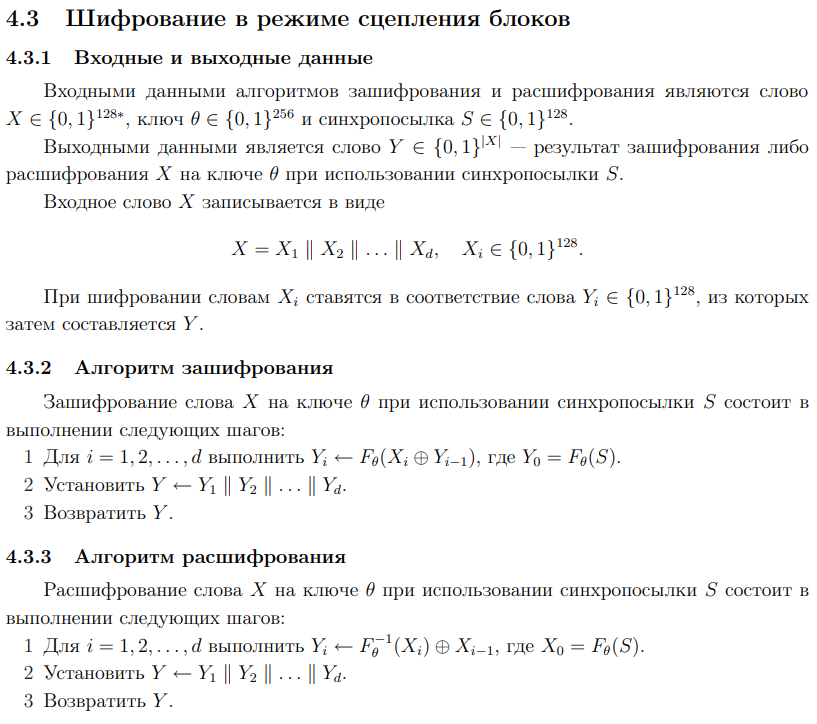


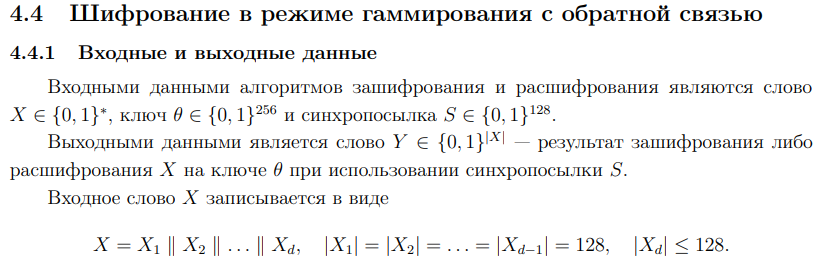


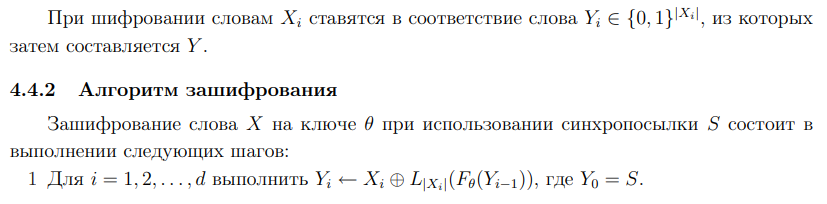


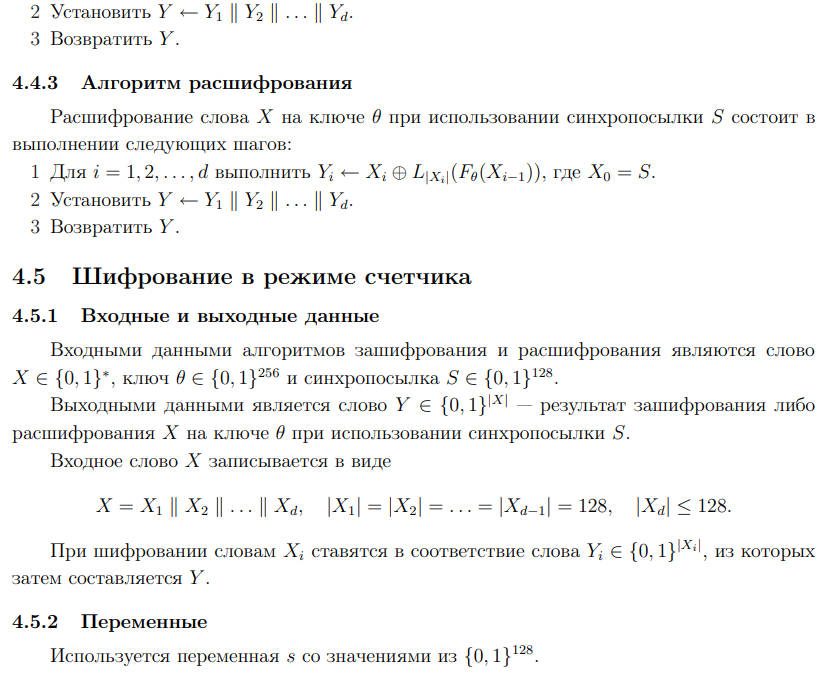


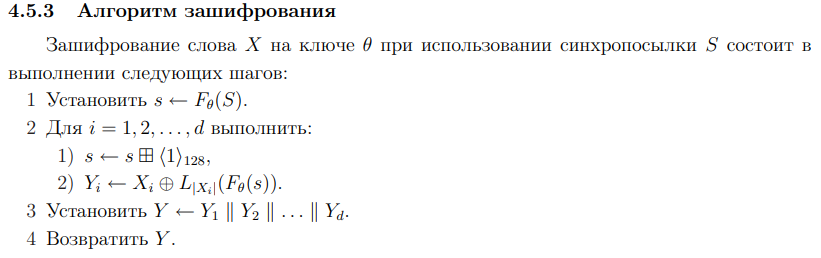


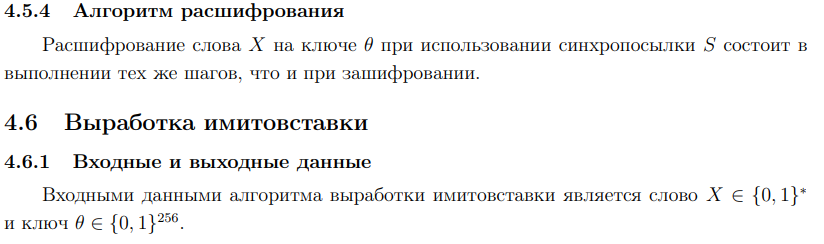


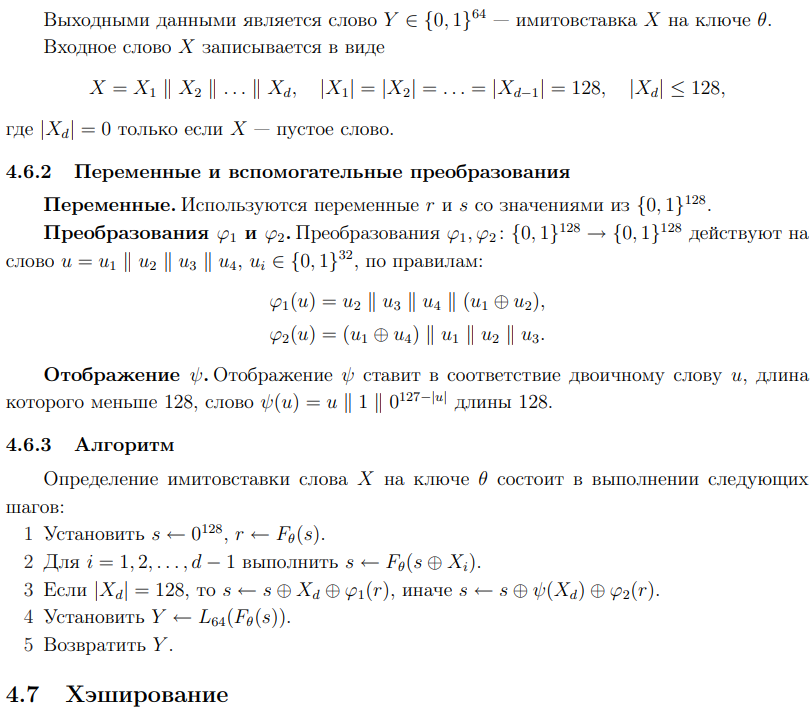


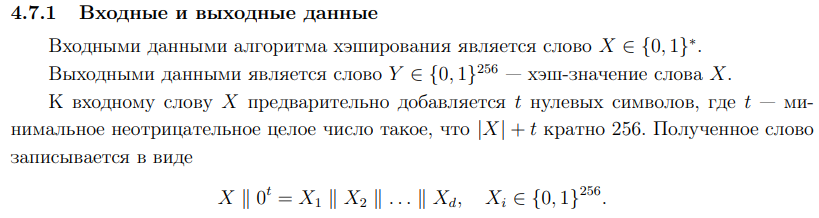


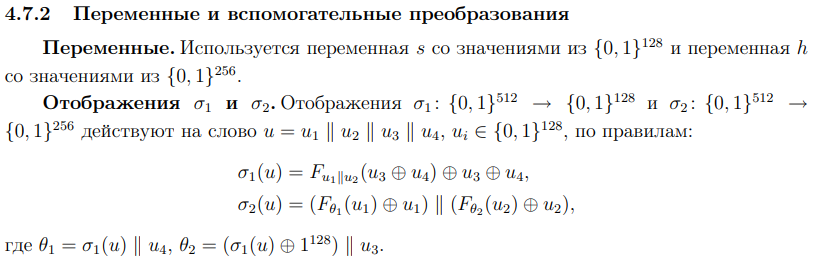


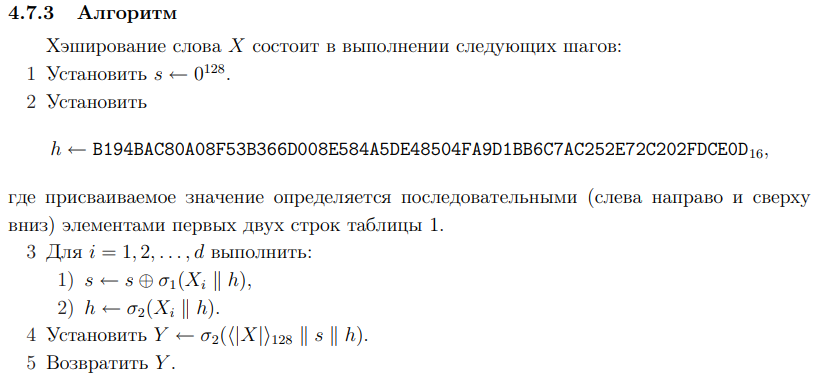




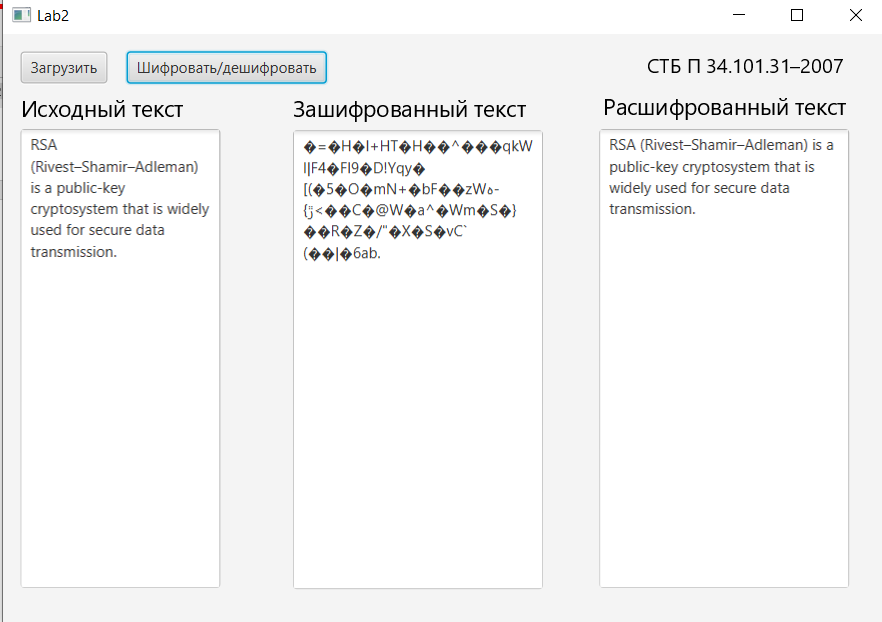


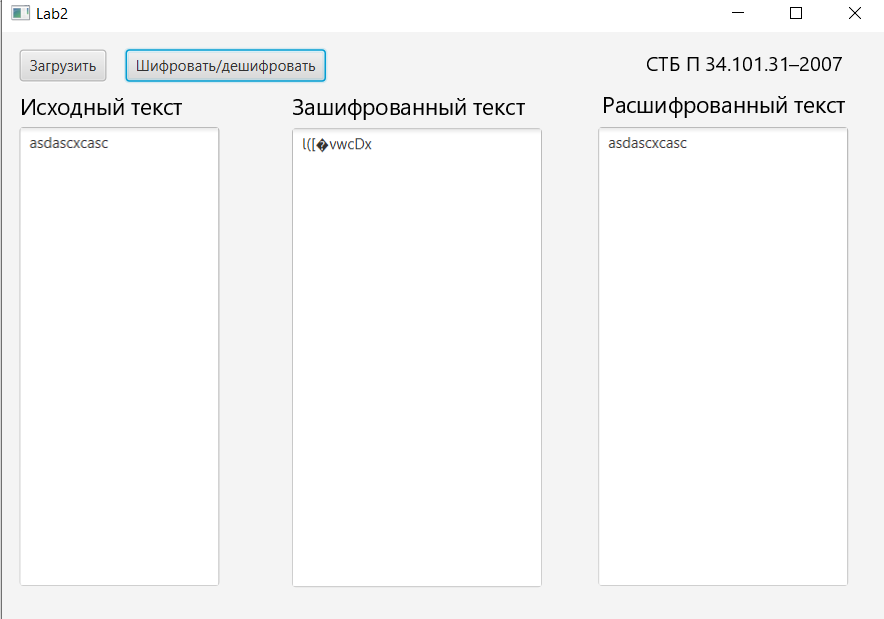


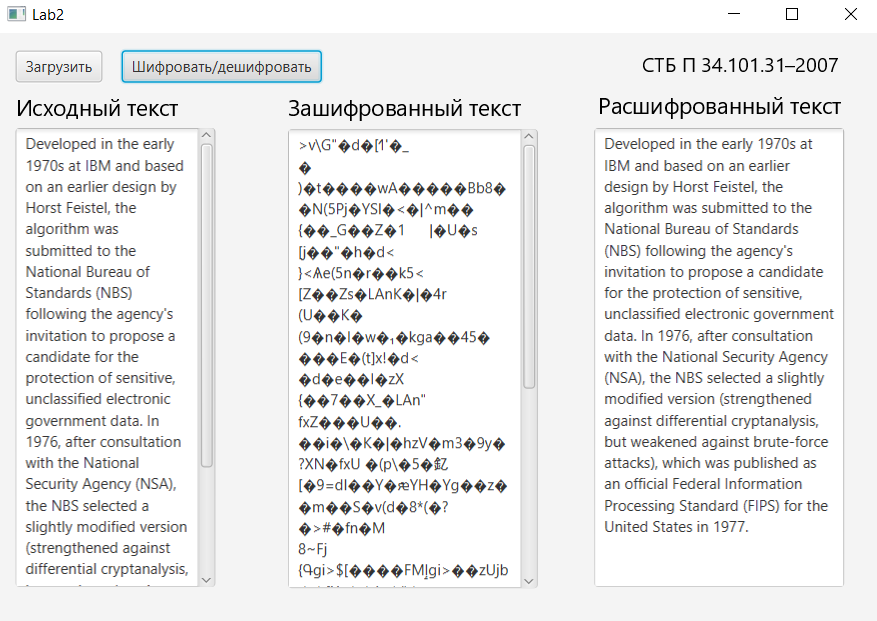


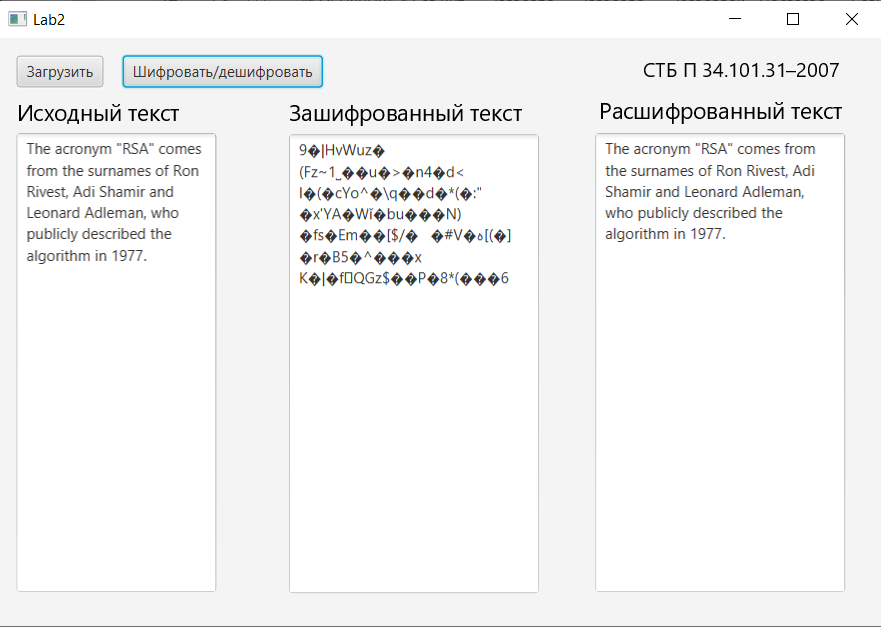


# 4. Результаты выполнения программы









# 5. Выводы

BelT — государственный стандарт симметричного шифрования и контроля целостности Республики Беларусь. Полное название стандарта — СТБ 34.101.31-2007 «Информационные технологии и безопасность. Криптографические алгоритмы шифрования и контроля целостности». Принят в качестве предварительного стандарта в 2007 году. Введён в действие в качестве окончательного стандарта в 2011 году.

Настоящий предстандарт определяет семейство криптографических алгоритмов шифрования и контроля целостности, которые используются в криптографических методах защиты информации при хранении, передаче и обработке данных.

Настоящий предстандарт применяется при разработке средств криптографической защиты информации в автоматизированных системах.

# 6. КОД ПРОГРАММЫ

**Main.java:**

package com.example.lab2si;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
import org.bouncycastle.jce.provider.BouncyCastleProvider;  
  
import java.io.IOException;  
import java.security.Security;  
  
public class Main extends Application {  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource("view.fxml"));  
 Scene scene = new Scene(fxmlLoader.load());  
 stage.setTitle("Lab2");  
 stage.setScene(scene);  
 stage.show();  
 Controller controller = fxmlLoader.getController();  
 controller.setStage(stage);  
 Security.*addProvider*(new BouncyCastleProvider());  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

**Controller.java:**

package com.example.lab2si;  
  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextArea;  
import javafx.stage.FileChooser;  
import javafx.stage.Stage;  
  
import javax.crypto.\*;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.lang.reflect.Array;  
import java.security.InvalidKeyException;  
import java.security.NoSuchAlgorithmException;  
import java.security.NoSuchProviderException;  
import java.util.ArrayList;  
import java.util.Arrays;  
  
public class Controller {  
 private Stage stage;  
 @FXML  
 private TextArea sourceTextArea;  
 @FXML  
 private TextArea encryptBeltTextArea;  
 @FXML  
 private TextArea decryptBeltTextArea;  
  
 public Stage getStage() {  
 return stage;  
 }  
  
 public void setStage(Stage stage) {  
 this.stage = stage;  
 }  
 @FXML  
 void Load(ActionEvent event) throws IOException {  
 final FileChooser fileChooser = new FileChooser();  
 fileChooser.setInitialDirectory(new File("C:\\Users\\User\\Desktop\\Tests\\Txts"));  
 File file = fileChooser.showOpenDialog(stage);  
 if (file != null) {  
 FileReader reader = new FileReader(file.getAbsolutePath());  
 StringBuilder result = new StringBuilder();  
 int character;  
 while((character=reader.read())!=-1)  
 {  
 result.append((char) character);  
 }  
 sourceTextArea.setText(result.toString());  
 }  
 }  
 @FXML  
 void EncryptDecrypt(ActionEvent event) {  
  
 String message = sourceTextArea.getText();  
 while (message.length() % 4 != 0){  
 message += " ";  
 }  
 byte[] messageByte = message.getBytes();  
 byte[] encryptByte = new byte[message.length()];  
 byte[] resultBytes = new byte[message.length()];  
 byte[] inBuffer = new byte[4];  
 byte[] outBuffer = new byte[4];  
  
 byte[] ks = {  
 (byte)0xAB, (byte)0xEF, (byte)0x97, 0x25, (byte)0xD4, (byte)0xC5, (byte)0xA8, 0x35,  
 (byte)0x97, (byte)0xA3, 0x67, (byte)0xD1, 0x44, (byte)0x94, (byte)0xCC, 0x25,  
 0x42, (byte)0xF2, 0x0F, 0x65, (byte)0x9D, (byte)0xDF, (byte)0xEC, (byte)0xC9,  
 0x61, (byte)0xA3, (byte)0xEC, 0x55, 0x0C, (byte)0xBA, (byte)0x8C, 0x75  
 };  
  
 Belt belt = new Belt();  
  
 for(int i = 0;i < message.length();i+=4){  
 inBuffer[0] = messageByte[i];  
 inBuffer[1] = messageByte[i+1];  
 inBuffer[2] = messageByte[i+2];  
 inBuffer[3] = messageByte[i+3];  
 belt.decrypt(ks, inBuffer, outBuffer);  
 encryptByte[i] = outBuffer[0];  
 encryptByte[i+1] = outBuffer[1];  
 encryptByte[i+2] = outBuffer[2];  
 encryptByte[i+3] = outBuffer[3];  
 }  
  
 String encrypt = new String(encryptByte);  
 encryptBeltTextArea.setText(encrypt);  
  
 for (int i = 0;i < message.length();i+=4){  
 inBuffer[0] = encryptByte[i];  
 inBuffer[1] = encryptByte[i+1];  
 inBuffer[2] = encryptByte[i+2];  
 inBuffer[3] = encryptByte[i+3];  
 belt.decrypt(ks, inBuffer, outBuffer);  
 resultBytes[i] = outBuffer[0];  
 resultBytes[i+1] = outBuffer[1];  
 resultBytes[i+2] = outBuffer[2];  
 resultBytes[i+3] = outBuffer[3];  
 }  
 String result = new String(resultBytes);  
 decryptBeltTextArea.setText(result);  
 }  
}

**Belt.java:**

package com.example.lab2si;  
  
import java.nio.ByteBuffer;  
import java.nio.ByteOrder;  
import java.nio.IntBuffer;  
  
public class Belt {  
 private static final byte[] *H* = {  
 (byte)0xB1, (byte)0x94, (byte)0xBA, (byte)0xC8, 0x0A, 0x08, (byte)0xF5, 0x3B, 0x36, 0x6D, 0x00, (byte)0x8E, 0x58, 0x4A, 0x5D, (byte)0xE4,  
 (byte)0x85, 0x04, (byte)0xFA, (byte)0x9D, 0x1B, (byte)0xB6, (byte)0xC7, (byte)0xAC, 0x25, 0x2E, 0x72, (byte)0xC2, 0x02, (byte)0xFD, (byte)0xCE, 0x0D,  
 0x5B, (byte)0xE3, (byte)0xD6, (byte)0x12, 0x17, (byte)0xB9, 0x61, (byte)0x81, (byte)0xFE, 0x67, (byte)0x86, (byte)0xAD, 0x71, 0x6B, (byte)0x89, 0x0B,  
 0x5C, (byte)0xB0, (byte)0xC0, (byte)0xFF, 0x33, (byte)0xC3, 0x56, (byte)0xB8, 0x35, (byte)0xC4, 0x05, (byte)0xAE, (byte)0xD8, (byte)0xE0, 0x7F, (byte)0x99,  
 (byte)0xE1, 0x2B, (byte)0xDC, 0x1A, (byte)0xE2, (byte)0x82, 0x57, (byte)0xEC, 0x70, 0x3F, (byte)0xCC, (byte)0xF0, (byte)0x95, (byte)0xEE, (byte)0x8D, (byte)0xF1,  
 (byte)0xC1, (byte)0xAB, 0x76, 0x38, (byte)0x9F, (byte)0xE6, 0x78, (byte)0xCA, (byte)0xF7, (byte)0xC6, (byte)0xF8, 0x60, (byte)0xD5, (byte)0xBB, (byte)0x9C, 0x4F,  
 (byte)0xF3, 0x3C, 0x65, 0x7B, 0x63, 0x7C, 0x30, 0x6A, (byte)0xDD, 0x4E, (byte)0xA7, 0x79, (byte)0x9E, (byte)0xB2, 0x3D, 0x31,  
 0x3E, (byte)0x98, (byte)0xB5, 0x6E, 0x27, (byte)0xD3, (byte)0xBC, (byte)0xCF, 0x59, 0x1E, 0x18, 0x1F, 0x4C, 0x5A, (byte)0xB7, (byte)0x93,  
 (byte)0xE9, (byte)0xDE, (byte)0xE7, 0x2C, (byte)0x8F, 0x0C, 0x0F, (byte)0xA6, 0x2D, (byte)0xDB, 0x49, (byte)0xF4, 0x6F, 0x73, (byte)0x96, 0x47,  
 0x06, 0x07, 0x53, 0x16, (byte)0xED, 0x24, 0x7A, 0x37, 0x39, (byte)0xCB, (byte)0xA3, (byte)0x83, 0x03, (byte)0xA9, (byte)0x8B, (byte)0xF6,  
 (byte)0x92, (byte)0xBD, (byte)0x9B, 0x1C, (byte)0xE5, (byte)0xD1, 0x41, 0x01, 0x54, 0x45, (byte)0xFB, (byte)0xC9, 0x5E, 0x4D, 0x0E, (byte)0xF2,  
 0x68, 0x20, (byte)0x80, (byte)0xAA, 0x22, 0x7D, 0x64, 0x2F, 0x26, (byte)0x87, (byte)0xF9, 0x34, (byte)0x90, 0x40, 0x55, 0x11,  
 (byte)0xBE, 0x32, (byte)0x97, 0x13, 0x43, (byte)0xFC, (byte)0x9A, 0x48, (byte)0xA0, 0x2A, (byte)0x88, 0x5F, 0x19, 0x4B, 0x09, (byte)0xA1,  
 0x7E, (byte)0xCD, (byte)0xA4, (byte)0xD0, 0x15, 0x44, (byte)0xAF, (byte)0x8C, (byte)0xA5, (byte)0x84, 0x50, (byte)0xBF, 0x66, (byte)0xD2, (byte)0xE8, (byte)0x8A,  
 (byte)0xA2, (byte)0xD7, 0x46, 0x52, 0x42, (byte)0xA8, (byte)0xDF, (byte)0xB3, 0x69, 0x74, (byte)0xC5, 0x51, (byte)0xEB, 0x23, 0x29, 0x21,  
 (byte)0xD4, (byte)0xEF, (byte)0xD9, (byte)0xB4, 0x3A, 0x62, 0x28, 0x75, (byte)0x91, 0x14, 0x10, (byte)0xEA, 0x77, 0x6C, (byte)0xDA, 0x1D};  
 private static final int[][] *KeyIndex* = {  
 { 0, 1, 2, 3, 4, 5, 6 },  
 { 7, 0, 1, 2, 3, 4, 5 },  
 { 6, 7, 0, 1, 2, 3, 4 },  
 { 5, 6, 7, 0, 1, 2, 3 },  
 { 4, 5, 6, 7, 0, 1, 2 },  
 { 3, 4, 5, 6, 7, 0, 1 },  
 { 2, 3, 4, 5, 6, 7, 0 },  
 { 1, 2, 3, 4, 5, 6, 7 }  
 };  
  
 int RotHi(int x,int r){  
 return (((x) << (r)) | ((x) >> (32 - (r))));  
 }  
  
 int U1(int x){  
 return ( (x) >> 24 );  
 }  
 int U2(int x){  
 return (((x) >> 16 ) & 0xff );  
 }  
 int U3(int x){  
 return (((x) >> 8 ) & 0xff );  
 }  
 int U4(int x){  
 return ( (x) & 0xff );  
 }  
  
 int HU1(int x){  
 return ((*H*)[Byte.*toUnsignedInt*((byte)U1((x)))] << 24);  
 }  
 int HU2(int x){  
 return ((*H*)[Byte.*toUnsignedInt*((byte)U2((x)))] << 16);  
 }  
 int HU3(int x){  
 return ((*H*)[Byte.*toUnsignedInt*((byte)U3((x)))] << 8);  
 }  
 int HU4(int x){  
 return (*H*)[Byte.*toUnsignedInt*((byte)U4((x)))];  
 }  
  
 int G(int x, int r){  
 return RotHi(HU4(x) | HU3(x) | HU2(x) | HU1(x), r);  
 }  
 void swap(Object ob1, Object ob2){  
 Object temp;  
 temp = ob1;  
 ob1 = ob2;  
 ob2 = temp;  
 }  
  
 void encrypt(byte[] ks, byte[] inBlock, byte[] outBlock){  
 int a = inBlock[0];  
 int b = inBlock[1];  
 int c = inBlock[2];  
 int d = inBlock[3];  
 int e;  
 IntBuffer intBuf =  
 ByteBuffer.*wrap*(ks)  
 .order(ByteOrder.*BIG\_ENDIAN*)  
 .asIntBuffer();  
 int[] key = new int[intBuf.remaining()];  
 intBuf.get(key);  
 for(int i = 0; i<8; ++i)  
 {  
 b ^= G((a + key[*KeyIndex*[i][0]]), 5);  
 c ^= G((d + key[*KeyIndex*[i][1]]), 21);  
 a = a - G((b + key[*KeyIndex*[i][2]]), 13);  
 e = (G((b + c + key[*KeyIndex*[i][3]]), 21) ^ (i + 1));  
 b += e;  
 c = c - e;  
 d += G((c + key[*KeyIndex*[i][4]]), 13);  
 b ^= G((a + key[*KeyIndex*[i][5]]), 21);  
 c ^= G((d + key[*KeyIndex*[i][6]]), 5);  
 swap(a, b);  
 swap(c, d);  
 swap(b, c);  
 }  
 outBlock[0] = Integer.*valueOf*(b).byteValue();  
 outBlock[1] = Integer.*valueOf*(d).byteValue();  
 outBlock[2] = Integer.*valueOf*(a).byteValue();  
 outBlock[3] = Integer.*valueOf*(c).byteValue();  
 }  
 void decrypt(byte[] ks, byte[] inBlock, byte[] outBlock){  
 int a = inBlock[0];  
 int b = inBlock[1];  
 int c = inBlock[2];  
 int d = inBlock[3];  
 int e;  
 IntBuffer intBuf =  
 ByteBuffer.*wrap*(ks)  
 .order(ByteOrder.*BIG\_ENDIAN*)  
 .asIntBuffer();  
 int[] key = new int[intBuf.remaining()];  
 intBuf.get(key);  
 for(int i = 7; i >= 0; --i)  
 {  
 b ^= G((a + key[*KeyIndex*[i][6]]),5);  
 c ^= G((d + key[*KeyIndex*[i][5]]),21);  
 a = (a - G((b + key[*KeyIndex*[i][4]]),13));  
 e = (G((b + c + key[*KeyIndex*[i][3]]),21) ^ (i + 1));  
 b += e;  
 c = (c - e);  
 d += G((c + key[*KeyIndex*[i][2]]),13);  
 b ^= G((a + key[*KeyIndex*[i][1]]),21);  
 c ^= G((d + key[*KeyIndex*[i][0]]),5);  
 swap(a, b);  
 swap(c, d);  
 swap(a, d);  
 }  
 outBlock[0] = Integer.*valueOf*(b).byteValue();  
 outBlock[1] = Integer.*valueOf*(d).byteValue();  
 outBlock[2] = Integer.*valueOf*(a).byteValue();  
 outBlock[3] = Integer.*valueOf*(c).byteValue();  
 }  
}