Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы Защиты Информации

ОТЧЁТ

к лабораторной работе №2

на тему

**Симметричная криптография.** **СТБ 34.101.31-2011. Режим простой замены**

Выполнил: студент гр. 253505

Сенько Н. С.

Проверил: ассистент кафедры

информатики

Герчик А. В.

Минск 2025

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc207722280)

[2 Ход работы 4](#_Toc207722281)

[Заключение 5](#_Toc207722282)

[Приложение А (обязательное) исходный код программного продукта 6](#_Toc207722283)

# 1 ЦЕЛЬ РАБОТЫ

Целью данной лабораторной работы является изучение принципов симметричного блочного шифрования на примере стандарта шифрования СТБ 34.101.31-2011 и его реализации в режиме простой замены.

В рамках работы требуется разработать программные средство для шифрования и дешифрования текстовых файлов, закрепить навыки работы с

блочными криптографическими алгоритмами и освоить процесс преобразования данных с использованием фиксированных ключей и таблицы преобразования H.

Результатом выполнения работы должен быть скрипт, который позволяет:

– зашифровать текстовый файл при помощи СТБ 34.101.31-2011 в режиме простой замены.

– расшифровать зашифрованный файл обратно в исходный вид.

– работать с ключом 256 бит и встроенной таблицей преобразования H.

– автоматически дополнять данные до кратности 128 бит.

# 2 ХОД РАБОТЫ

Алгоритм был реализован на языке PHP. Для выполнения задачи был создан объект, содержащий все методы, необходимые для шифрования и дешифрования файлов. Алгоритм автоматически генерирует случайный ключ. Далее алгоритм считывает данные из входного файла и применяет алгоритм шифрования. Полученный результат он записывает в бинарный файл 2.bin, ключ записывается в файле key.bin. Далее скрипт считывает данные с файлов, извлекает, ключ и данные и запускает процесс расшифровки.

Результат полного выполнения скрипта можно увидеть на рисунке 1.

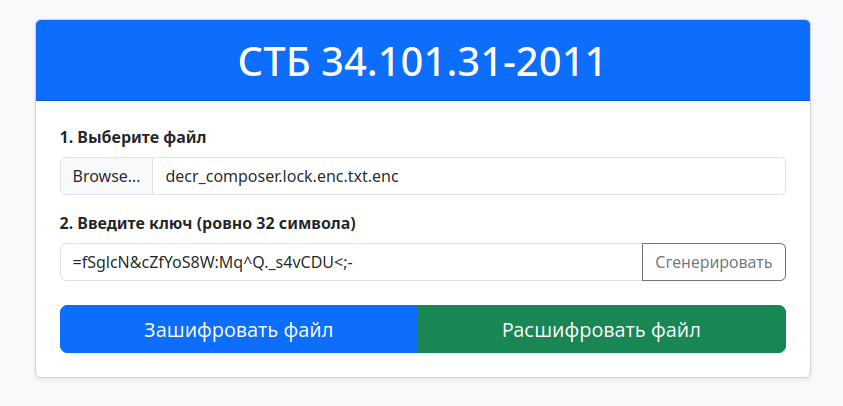


Рисунок 1 – Вывод работы программы

Как видно на рисунке 1, скрипт успешно справился с шифровкой и дешифровкой информации.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были изучены теоретические основы симметричных блочных шифров на примере алгоритма СТБ 34.101.31-2011. Также на PHP был реализован данные гост в режиме простой замены.

Таким образом, поставленные цели были достигнуты: алгоритм был реализован, проведены эксперименты с его использованием, а также закреплены навыки программной реализации симметричных блочных шифров.

# ПРИЛОЖЕНИЕ А

# (ОБЯЗАТЕЛЬНОЕ)

# ИСХОДНЫЙ КОД ПРОГРАММНОГО ПРОДУКТА

<?php

namespace App;

class STB34101312011

{

/\*\*

\* Таблица замен H.

\* @var array

\*/

private static $H\_TABLE = [

[0xB1, 0x94, 0xBA, 0xC8, 0x0A, 0x08, 0xF5, 0x3B, 0x36, 0x6D, 0x00, 0x8E, 0x58, 0x4A, 0x5D, 0xE4],

[0x85, 0x04, 0xFA, 0x9D, 0x1B, 0xB6, 0xC7, 0xAC, 0x25, 0x2E, 0x72, 0xC2, 0x02, 0xFD, 0xCE, 0x0D],

[0x5B, 0xE3, 0xD6, 0x12, 0x17, 0xB9, 0x61, 0x81, 0xFE, 0x67, 0x86, 0xAD, 0x71, 0x6B, 0x89, 0x0B],

[0x5C, 0xB0, 0xC0, 0xFF, 0x33, 0xC3, 0x56, 0xB8, 0x35, 0xC4, 0x05, 0xAE, 0xD8, 0xE0, 0x7F, 0x99],

[0xE1, 0x2B, 0xDC, 0x1A, 0xE2, 0x82, 0x57, 0xEC, 0x70, 0x3F, 0xCC, 0xF0, 0x95, 0xEE, 0x8D, 0xF1],

[0xC1, 0xAB, 0x76, 0x38, 0x9F, 0xE6, 0x78, 0xCA, 0xF7, 0xC6, 0xF8, 0x60, 0xD5, 0xBB, 0x9C, 0x4F],

[0xF3, 0x3C, 0x65, 0x7B, 0x63, 0x7C, 0x30, 0x6A, 0xDD, 0x4E, 0xA7, 0x79, 0x9E, 0xB2, 0x3D, 0x31],

[0x3E, 0x98, 0xB5, 0x6E, 0x27, 0xD3, 0xBC, 0xCF, 0x59, 0x1E, 0x18, 0x1F, 0x4C, 0x5A, 0xB7, 0x93],

[0xE9, 0xDE, 0xE7, 0x2C, 0x8F, 0x0C, 0x0F, 0xA6, 0x2D, 0xDB, 0x49, 0xF4, 0x6F, 0x73, 0x96, 0x47],

[0x06, 0x07, 0x53, 0x16, 0xED, 0x24, 0x7A, 0x37, 0x39, 0xCB, 0xA3, 0x83, 0x03, 0xA9, 0x8B, 0xF6],

[0x92, 0xBD, 0x9B, 0x1C, 0xE5, 0xD1, 0x41, 0x01, 0x54, 0x45, 0xFB, 0xC9, 0x5E, 0x4D, 0x0E, 0xF2],

[0x68, 0x20, 0x80, 0xAA, 0x22, 0x7D, 0x64, 0x2F, 0x26, 0x87, 0xF9, 0x34, 0x90, 0x40, 0x55, 0x11],

[0xBE, 0x32, 0x97, 0x13, 0x43, 0xFC, 0x9A, 0x48, 0xA0, 0x2A, 0x88, 0x5F, 0x19, 0x4B, 0x09, 0xA1],

[0x7E, 0xCD, 0xA4, 0xD0, 0x15, 0x44, 0xAF, 0x8C, 0xA5, 0x84, 0x50, 0xBF, 0x66, 0xD2, 0xE8, 0x8A],

[0xA2, 0xD7, 0x46, 0x52, 0x42, 0xA8, 0xDF, 0xB3, 0x69, 0x74, 0xC5, 0x51, 0xEB, 0x23, 0x29, 0x21],

[0xD4, 0xEF, 0xD9, 0xB4, 0x3A, 0x62, 0x28, 0x75, 0x91, 0x14, 0x10, 0xEA, 0x77, 0x6C, 0xDA, 0x1D],

];

private static function h(int $x): int

{

$line = ($x >> 4) & 0x0F;

$column = $x & 0x0F;

return self::$H\_TABLE[$line][$column];

}

public static function generateKey(): array

{

$key = [];

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

$key[$i] = random\_int(0, 0xFFFFFFFF);

}

return $key;

}

private static function rotateLeft(int $value, int $count): int

{

return (($value << $count) | ($value >> (32 - $count))) & 0xFFFFFFFF;

}

private static function g(int $r, int $block): int

{

$u1 = self::h($block & 0xFF);

$u2 = self::h(($block >> 8) & 0xFF);

$u3 = self::h(($block >> 16) & 0xFF);

$u4 = self::h(($block >> 24) & 0xFF);

$ans = ($u4 << 24) | ($u3 << 16) | ($u2 << 8) | $u1;

return self::rotateLeft($ans, $r);

}

private static function k(int $i, array $key): int

{

return $key[($i - 1) % 8];

}

private static function add(int $a, int $b): int

{

return ($a + $b) & 0xFFFFFFFF;

}

private static function sub(int $a, int $b): int

{

return ($a - $b) & 0xFFFFFFFF;

}

private static function readLong(string $bytes, int $offset): array

{

$l1 = unpack('N', substr($bytes, $offset, 4))[1];

$l2 = unpack('N', substr($bytes, $offset + 4, 4))[1];

$l3 = unpack('N', substr($bytes, $offset + 8, 4))[1];

$l4 = unpack('N', substr($bytes, $offset + 12, 4))[1];

return [$l1, $l2, $l3, $l4];

}

private static function writeLong(array $longs): string

{

return pack('NNNN', $longs[0], $longs[1], $longs[2], $longs[3]);

}

private static function encryptBlock(array $block, array $key): array

{

$a = $block[0];

$b = $block[1];

$c = $block[2];

$d = $block[3];

for ($i = 1; $i <= 8; $i++) {

$b = $b ^ self::g(5, self::add($a, self::k(7 \* $i - 6, $key)));

$c = $c ^ self::g(21, self::add($d, self::k(7 \* $i - 5, $key)));

$a = self::sub($a, self::g(13, self::add($b, self::k(7 \* $i - 4, $key))));

$e = self::g(21, self::add(self::add($b, $c), self::k(7 \* $i - 3, $key))) ^ $i;

$b = self::add($b, $e);

$c = self::sub($c, $e);

$d = self::add($d, self::g(13, self::add($c, self::k(7 \* $i - 2, $key))));

$b = $b ^ self::g(21, self::add($a, self::k(7 \* $i - 1, $key)));

$c = $c ^ self::g(5, self::add($d, self::k(7 \* $i, $key)));

// Swaps

$temp = $a; $a = $b; $b = $temp;

$temp = $c; $c = $d; $d = $temp;

$temp = $b; $b = $c; $c = $temp;

}

return [$b, $d, $a, $c];

}

private static function decryptBlock(array $block, array $key): array

{

$a = $block[0];

$b = $block[1];

$c = $block[2];

$d = $block[3];

for ($i = 8; $i >= 1; $i--) {

// Swaps are reversed

$temp = $b; $b = $c; $c = $temp;

$temp = $c; $c = $d; $d = $temp;

$temp = $a; $a = $b; $b = $temp;

$c = $c ^ self::g(5, self::add($d, self::k(7 \* $i, $key)));

$b = $b ^ self::g(21, self::add($a, self::k(7 \* $i - 1, $key)));

$d = self::sub($d, self::g(13, self::add($c, self::k(7 \* $i - 2, $key))));

$e = self::g(21, self::add(self::add($b, $c), self::k(7 \* $i - 3, $key))) ^ $i;

$c = self::add($c, $e);

$b = self::sub($b, $e);

$a = self::add($a, self::g(13, self::add($b, self::k(7 \* $i - 4, $key))));

$c = $c ^ self::g(21, self::add($d, self::k(7 \* $i - 5, $key)));

$b = $b ^ self::g(5, self::add($a, self::k(7 \* $i - 6, $key)));

}

return [$a, $b, $c, $d];

}

public static function encrypt(string $input, array $key): string

{

$out = '';

$blocks = str\_split($input, 16);

foreach ($blocks as $block) {

$longs = self::readLong($block, 0);

$encryptedLongs = self::encryptBlock($longs, $key);

$out .= self::writeLong($encryptedLongs);

}

return $out;

}

public static function decrypt(string $input, array $key): string

{

$out = '';

$blocks = str\_split($input, 16);

foreach ($blocks as $block) {

$longs = self::readLong($block, 0);

// The output of encryption is Y = b, d, a, c.

// We need to pass it to decryption in the correct order for the variables a, b, c, d.

$decryption\_input = [$longs[2], $longs[0], $longs[3], $longs[1]];

$decryptedLongs = self::decryptBlock($decryption\_input, $key);

$out .= self::writeLong($decryptedLongs);

}

return $out;

}

/\*\*

\* @param string $key\_string Ключ в виде строки.

\* @return array Массив из 8 целых чисел.

\* @throws Exception Если длина ключа не равна 32 байтам.

\*/

public static function keyFromString(string $key\_string): array

{

if (strlen($key\_string) !== 32) {

throw new Exception("Длина ключа должна быть ровно 32 байта.");

}

$chunks = str\_split($key\_string, 4);

$numeric\_key = [];

foreach ($chunks as $chunk) {

$numeric\_key[] = unpack('N', $chunk)[1];

}

return $numeric\_key;

}

}