Министерство образования Республики Беларусь Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Кафедра информатики

ОТЧЕТ

по лабораторной работе №2

**Симметричная криптография. СТБ 34.101.31-2011**

Выполнил: Проверил:

студент гр. 653503 Артемьев В.С.

Лисковец Б.Н.

Минск 2019

**Введение**

СТБ 34.101.31 – блочный шифр с 256-битным ключом и 8 циклами криптопреобразований, оперирующий с 128-битными словами. Криптографические алгоритмы стандарта построены на основе базовых режимов шифрования блоков данных. Все алгоритмы стандарта делятся на 8 групп:

* алгоритмы шифрования в режиме простой замены;
* алгоритмы шифрования в режиме сцепления блоков;
* алгоритмы шифрования в режиме гаммирования с обратной связью;
* алгоритмы шифрования в режиме счётчика;
* алгоритм выработки имитовставки;
* алгоритмы одновременного шифрования и имитозащиты данных;
* алгоритмы одновременного шифрования и имитозащиты ключей;
* алгоритм хэширования;

Первые четыре группы предназначены для обеспечения безопасного обмена сообщениями. Каждая группа включает алгоритм зашифрования и алгоритм расшифрования на секретном ключе. Стороны, располагающие общим ключом, могут организовать обмен сообщениями путём их зашифрования перед отправкой и расшифрования после получения. В режимах простой замены и сцепления блоков шифруются сообщения, которые содержат хотя бы один блок, а в режимах гаммирования с обратной связью и счётчика — сообщения произвольной длины.

**Задание:** Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма СТБ 34.101.31-2011 в различных режимах.

**Алгоритм**

**Шифрование блока**

**Входные данные и выходные данные**

Входными данными алгоритмов зашифрования и расшифрования являются блок  {\displaystyle X\in \{0,1\}^{128}}и ключ {\displaystyle \theta \in \{0,1\}^{256}.}

Выходными данными является блок {\displaystyle Y\in \{0,1\}^{128}}— результат зашифрования либо расшифрования слова на ключе  либо 

Входные данные для шифрования подготавливаются следующим образом:

* Слово {\displaystyle X} записывается в виде {\displaystyle X=X\_{1}\|X\_{2}\|X\_{3}\|X\_{4},X\_{i}\in \{0,1\}^{32}.}
* Ключ записывается в виде  и определяются тактовые ключи

**Обозначения и вспомогательные преобразования**

Преобразование  ставит в соответствие слову , слово



* циклический сдвиг влево на r бит.

 операция замены 8-битной входной строки подстановкой с рисунка 1.

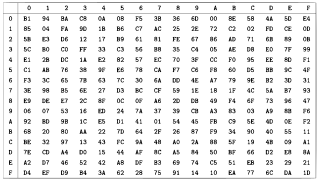


Рисунок 1 – Преобразование Н

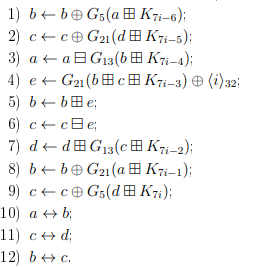
Подстановка {\displaystyle H:\{0,1\}^{8}\rightarrow \{0,1\}^{8}}задается фиксированной таблицей. В таблице используется шестнадцатеричное представление слов {\displaystyle u\in \{0,1\}^{8}.}

{\displaystyle \boxplus } и  {\displaystyle \boxminus -}операции сложения и вычитания по модулю 232{\displaystyle 2^{32}.}

**Зашифрование**

Для зашифрования блока {\displaystyle X} на ключе {\displaystyle \theta } выполняются следующие шаги:

1. Установить {\displaystyle a\leftarrow X\_{1},b\leftarrow X\_{2},c\leftarrow X\_{3},d\leftarrow X\_{4}.}
2. Для {\displaystyle i}i = 1,2,… ,8 выполнить:



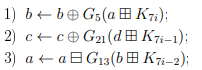
3. Установить {\displaystyle Y\leftarrow b\|d\|a\|c.}

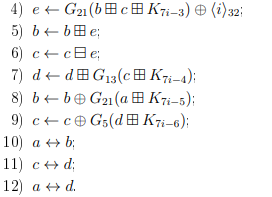
4. Возвратить {\displaystyle Y.}

**Расшифрование**

Для расшифрования блока {\displaystyle X} на ключе {\displaystyle \theta } выполняются следующие шаги:

1. Установить {\displaystyle a\leftarrow X\_{1},b\leftarrow X\_{2},c\leftarrow X\_{3},d\leftarrow X\_{4}.}
2. Для {\displaystyle i}i = 8,7,… ,1 выполнить:





3. Установить {\displaystyle Y\leftarrow c\|a\|d\|b.} 

4. Возвратить {\displaystyle Y.} 

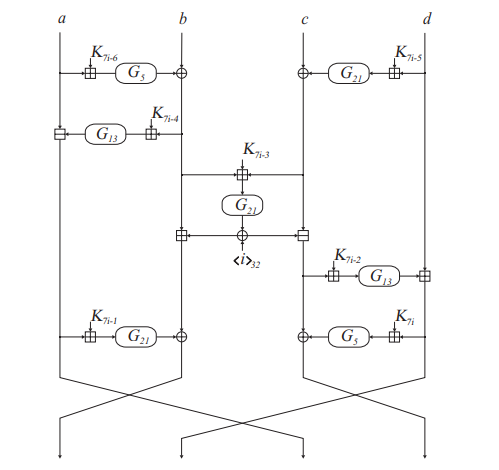


Рисунок 2 – Вычисления на i такте шифрования

**Выработка имитовставки**

**Входные данные**

* Исходное сообщение произвольной длины, представленное в виде битовой последовательности {\displaystyle X\in \{0,1\}^{\*}} . Если {\displaystyle X}  - непустое слово, то записать его в виде:{\displaystyle {\displaystyle X=X\_{1}\|X\_{2}\|...\|X\_{n}},|X\_{1}|=|X\_{2}|=...=|X\_{n-1}|=128,0<|X\_{n}|\leq 128} .   
  Если же  {\displaystyle X}- пустое, то  {\displaystyle n=1}и {\displaystyle |X\_{1}|=0}
* Ключ {\displaystyle \theta \in \{0,1\}^{256}}- битовая последовательность длины 256.

**Вспомогательные преобразования и переменные**

* Преобразования {\displaystyle \phi \_{1},\phi \_{2}}, которые действуют на слово {\displaystyle u=u\_{1}||u\_{2}||u\_{3}||u\_{4},u\_{i}\in \{0,1\}^{32}}- битовая последовательность длины 32 . При этом:



{\displaystyle \phi \_{1}(u)=u\_{2}||u\_{3}||u\_{4}||(u\_{1}\oplus u\_{2}),}Отображение {\displaystyle \psi }, которое ставит в соответствие битовой последовательности длины меньше 128, слово длиной 128. Действует по правилу:



* Вспомогальтельные переменные{\displaystyle r,s\in \{0,1\}^{256}}  - битовые последовательности длины 128.

**Алгоритм выработки имитовставки**

1. Заполнить вспомогательную переменную {\displaystyle s}s нулями: {\displaystyle s\leftarrow 0^{128}}  и установить результат шифрования {\displaystyle s}s на данном ключе {\displaystyle \theta \in \{0,1\}^{256}} в {\displaystyle r}
2. Для каждого блока входного сообщения {\displaystyle i=1,2,...,n-1} выполнить: .
3. Если {\displaystyle |X\_{n}|=128}, то выполняем {\displaystyle s\leftarrow s\oplus X\_{n}\oplus \phi \_{1}(r)}, иначе }.
4. Записать в {\displaystyle T} первые 64 бита слова {\displaystyle F\_{\theta }(s)}.
5. Возвратить {\displaystyle T}.

**Практическая часть**

Зададим 128-битную синхропосылку и 256 битный ключ (рис. 3).

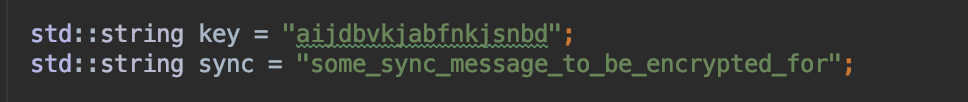


Рисунок 3 – Синхропосылка и ключ шифрования

Создадим файл с начальным текстом (рис. 4).

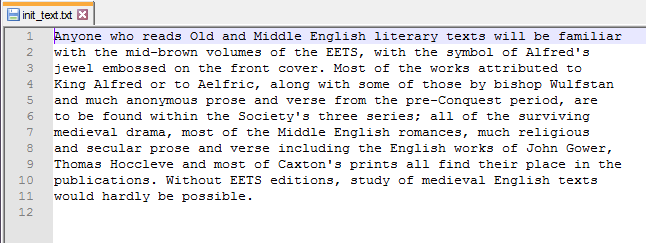


Рисунок 4 – Начальный текст

Далее запустим программу и получим 2 файла: с зашифрованным и расшифрованным сообщениями (рис. 5 и рис. 6).

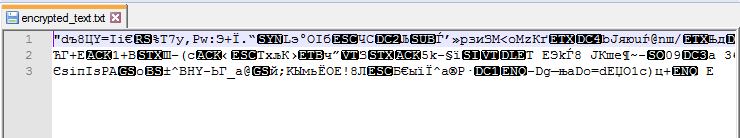


Рисунок 5 – Зашифрованный текст

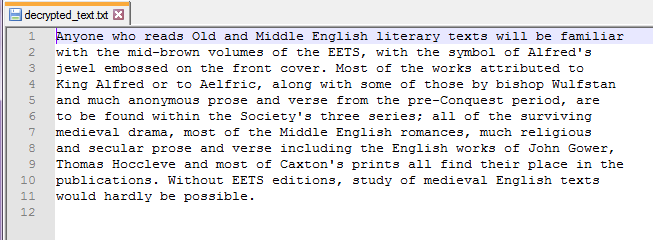


Рисунок 6 –Расшифрованный текст

**Приложение А. Текст программы**

**belt.cpp**

#include "belt.h"  
  
void belt\_init(uint8\_t \*ks, const uint8\_t \*k, std::size\_t klen) {  
 std::size\_t i;  
 switch (klen) {  
 case 16:  
 for (i = 0; i < 16; ++i) {  
 ks[i + 0] = k[i];  
 ks[i + 16] = k[i];  
 }  
 break;  
  
 case 24:  
 for (i = 0; i < 24; ++i) {  
 ks[i] = k[i];  
 }  
 store32(ks + 24, load32(k + 0) ^ load32(k + 4) ^ load32(k + 8));  
 store32(ks + 28, load32(k + 12) ^ load32(k + 16) ^ load32(k + 20));  
 break;  
  
 case 32:  
 for (i = 0; i < 32; ++i) {  
 ks[i] = k[i];  
 }  
 break;  
 }  
}  
  
void belt\_encrypt(uint8\_t \*out, const uint8\_t \*in, const uint8\_t \*ks) {  
 uint32\_t a = load32(in + 0);  
 uint32\_t b = load32(in + 4);  
 uint32\_t c = load32(in + 8);  
 uint32\_t d = load32(in + 12);  
  
 uint32\_t e;  
 std::size\_t i;  
 uint32\_t tmp;  
 uint32\_t key[8] = {0};  
  
 for (i = 0; i < 8; ++i) {  
 key[i] = load32(ks + (4 \* i));  
 }  
  
 for (i = 0; i < 8; ++i) {  
 b ^= G((a + key[KeyIndex[i][0]]), H, 5);  
 c ^= G((d + key[KeyIndex[i][1]]), H, 21);  
 a -= G((b + key[KeyIndex[i][2]]), H, 13);  
 e = (G((b + c + key[KeyIndex[i][3]]), H, 21) ^ (uint32\_t)(i + 1));  
 b += e;  
 c -= e;  
 d += G((c + key[KeyIndex[i][4]]), H, 13);  
 b ^= G((a + key[KeyIndex[i][5]]), H, 21);  
 c ^= G((d + key[KeyIndex[i][6]]), H, 5);  
 SWAP(a, b, tmp);  
 SWAP(c, d, tmp);  
  
 SWAP(b, c, tmp);  
 }  
 store32(out + 0, b);  
 store32(out + 4, d);  
 store32(out + 8, a);  
 store32(out + 12, c);  
}  
  
void belt\_decrypt(uint8\_t \*out, const uint8\_t \*in, const uint8\_t \*ks) {  
 uint32\_t a = load32(in + 0);  
 uint32\_t b = load32(in + 4);  
 uint32\_t c = load32(in + 8);  
 uint32\_t d = load32(in + 12);  
 uint32\_t e;  
 std::size\_t i;  
 uint32\_t tmp;  
 uint32\_t key[8] = {0};  
  
 for (i = 0; i < 8; ++i) {  
 key[i] = load32(ks + (4 \* i));  
 }  
  
 for (i = 0; i < 8; ++i) {  
 b ^= G((a + key[KeyIndex[7 - i][6]]), H, 5);  
 c ^= G((d + key[KeyIndex[7 - i][5]]), H, 21);  
 a -= G((b + key[KeyIndex[7 - i][4]]), H, 13);  
 e = (G((b + c + key[KeyIndex[7 - i][3]]), H, 21) ^ (uint32\_t)(7 - i + 1));  
 b += e;  
 c -= e;  
 d += G((c + key[KeyIndex[7 - i][2]]), H, 13);  
 b ^= G((a + key[KeyIndex[7 - i][1]]), H, 21);  
 c ^= G((d + key[KeyIndex[7 - i][0]]), H, 5);  
 SWAP(a, b, tmp);  
 SWAP(c, d, tmp);  
 SWAP(a, d, tmp);  
 }  
 store32(out + 0, c);  
 store32(out + 4, a);  
 store32(out + 8, d);  
 store32(out + 12, b);  
}

**modes.cpp**

#include "../include/modes.h"  
  
// прямая замена  
void encrypt\_plain(uint8\_t \*outenc, const uint8\_t \*inenc, const uint8\_t \*ks,  
 const int32\_t len) {  
 uint8\_t buffer[16] = {0};  
 uint8\_t outchunk[16] = {0};  
 for (int i = 0; i < (len + 15) / 16; ++i) {  
 store128(buffer, inenc + i \* 16);  
 belt\_encrypt(outchunk, buffer, ks);  
 store128(outenc + i \* 16, outchunk);  
 }  
}  
  
void decrypt\_plain(uint8\_t \*outdec, const uint8\_t \*outenc, const uint8\_t \*ks,  
 const int32\_t len) {  
 uint8\_t buffer[16] = {0};  
 uint8\_t outchunk[16] = {0};  
 for (int i = 0; i < (len + 15) / 16; ++i) {  
 store128(buffer, outenc + i \* 16);  
 belt\_decrypt(outchunk, buffer, ks);  
 store128(outdec + i \* 16, outchunk);  
 }  
}  
  
// сцепление блоков  
void encrypt\_block(uint8\_t \*outenc, const uint8\_t \*inenc, const uint8\_t \*ks,  
 const int32\_t len, const uint8\_t \*s) {  
 uint8\_t buffer[16] = {0};  
 uint8\_t outchunk[16] = {0};  
 uint8\_t synhro[16] = {0};  
 belt\_encrypt(synhro, s, ks);  
 for (int i = 0; i < (len + 15) / 16; ++i) {  
 store128(buffer, inenc + i \* 16);  
  
 for (std::size\_t j = 0; j < 16; ++j) {  
 buffer[j] = buffer[j] ^ synhro[j];  
 }  
 belt\_encrypt(outchunk, buffer, ks);  
  
 store128(outenc + i \* 16, outchunk);  
 store128(synhro, outchunk);  
 }  
}  
  
void decrypt\_block(uint8\_t \*outdec, const uint8\_t \*outenc, const uint8\_t \*ks,  
 const int32\_t len, const uint8\_t \*s) {  
 uint8\_t buffer[16] = {0};  
 uint8\_t outchunk[16] = {0};  
 uint8\_t synhro[16] = {0};  
 belt\_encrypt(synhro, s, ks);  
 for (int i = 0; i < (len + 15) / 16; ++i) {  
 store128(buffer, outenc + i \* 16);  
  
 belt\_decrypt(outchunk, buffer, ks);  
 for (std::size\_t j = 0; j < 16; ++j) {  
 outchunk[j] = outchunk[j] ^ synhro[j];  
 }  
  
 store128(outdec + i \* 16, outchunk);  
 store128(synhro, buffer);  
 }  
}  
  
// гаммирование  
void encrypt\_gamming(uint8\_t \*outenc, const uint8\_t \*inenc, const uint8\_t \*ks,  
 const int32\_t len, const uint8\_t \*s) {  
 uint8\_t buffer[16] = {0};  
 uint8\_t outchunk[16] = {0};  
 uint8\_t synhro[16] = {0};  
 uint8\_t ns[16] = {0};  
 store128(ns, s);  
 for (int i = 0; i < (len + 15) / 16; ++i) {  
 belt\_encrypt(synhro, ns, ks);  
 store128(buffer, inenc + i \* 16);  
  
 for (std::size\_t j = 0; j < 16; ++j) {  
 outchunk[j] = buffer[j] ^ synhro[j];  
 }  
  
 store128(outenc + i \* 16, outchunk);  
 store128(ns, outchunk);  
 }  
}  
  
void decrypt\_gamming(uint8\_t \*outdec, const uint8\_t \*outenc, const uint8\_t \*ks,  
 const int32\_t len, const uint8\_t \*s) {  
 uint8\_t buffer[16] = {0};  
 uint8\_t outchunk[16] = {0};  
 uint8\_t synhro[16] = {0};  
 uint8\_t ns[16] = {0};  
 store128(ns, s);  
 for (int i = 0; i < (len + 15) / 16; ++i) {  
 belt\_encrypt(synhro, ns, ks);  
 store128(buffer, outenc + i \* 16);  
  
 for (std::size\_t j = 0; j < 16; ++j) {  
 outchunk[j] = buffer[j] ^ synhro[j];  
 }  
  
 store128(outdec + i \* 16, outchunk);  
 store128(ns, buffer);  
 }  
}  
  
// счётчик  
void encrypt\_counter(uint8\_t \*outenc, const uint8\_t \*inenc, const uint8\_t \*ks,  
 const int32\_t len, const uint8\_t \*s) {  
 uint8\_t buffer[16] = {0};  
 uint8\_t outchunk[16] = {0};  
 uint8\_t synhro[16] = {0};  
 uint8\_t ns[16] = {0};  
 belt\_encrypt(ns, s, ks);  
 for (int i = 0; i < (len + 15) / 16; ++i) {  
 belt\_encrypt(synhro, ns, ks);  
 square\_plus(ns, synhro);  
 store128(buffer, inenc + i \* 16);  
  
 for (std::size\_t j = 0; j < 16; ++j) {  
 outchunk[j] = buffer[j] ^ ns[j];  
 }  
  
 store128(outenc + i \* 16, outchunk);  
 }  
}  
  
void decrypt\_counter(uint8\_t \*outdec, const uint8\_t \*outenc, const uint8\_t \*ks,  
 const int32\_t len, const uint8\_t \*s) {  
 encrypt\_counter(outdec, outenc, ks, len, s);  
}

**Fileio.cpp**

#include "../include/fileio.h"  
#include <iostream>  
  
std::vector<uint8\_t> read\_str\_from\_file(const std::string &input) {  
 std::ifstream t(input);  
 t.seekg(0, std::ios::*end*);  
 size\_t size = t.tellg();  
 std::string buffer(size, ' ');  
 t.seekg(0);  
 t.read(&buffer[0], size);  
 return {buffer.begin(), buffer.end()};  
}  
  
void write\_str\_to\_file(const std::vector<uint8\_t> &text,  
 const std::string &output) {  
 std::ofstream t(output);  
 t << std::string(text.begin(), text.end());  
}  
  
void encrypt\_file(Mode m, const uint8\_t \*keyenc, const uint8\_t \*synhro,  
 const std::string &init\_text\_file,  
 const std::string &encrypted\_text\_file) {  
 std::vector<uint8\_t> inenc = read\_str\_from\_file(init\_text\_file);  
  
 std::vector<uint8\_t> outenc(inenc.size() + 10, 0);  
 uint8\_t ks[32] = {0};  
  
 belt\_init(ks, keyenc, 32);  
 switch (m) {  
 case Mode::*PLAIN*:  
 encrypt\_plain(outenc.data(), inenc.data(), ks, inenc.size());  
 break;  
 case Mode::*BLOCK*:  
 encrypt\_block(outenc.data(), inenc.data(), ks, inenc.size(), synhro);  
 break;  
 case Mode::*GAMMING*:  
 encrypt\_gamming(outenc.data(), inenc.data(), ks, inenc.size(), synhro);  
 break;  
 case Mode::*COUNTER*:  
 encrypt\_counter(outenc.data(), inenc.data(), ks, inenc.size(), synhro);  
 break;  
 }  
  
 write\_str\_to\_file(outenc, encrypted\_text\_file);  
}  
  
void decrypt\_file(Mode m, const uint8\_t \*keyenc, const uint8\_t \*synhro,  
 const std::string &encrypted\_text\_file,  
 const std::string &decrypted\_text\_file) {  
 std::vector<uint8\_t> outenc = read\_str\_from\_file(encrypted\_text\_file);  
  
 std::vector<uint8\_t> outdec(outenc.size(), 0);  
 uint8\_t ks[32] = {0};  
  
 belt\_init(ks, keyenc, 32);  
 switch (m) {  
 case Mode::*PLAIN*:  
 decrypt\_plain(outdec.data(), outenc.data(), ks, outenc.size());  
 break;  
 case Mode::*BLOCK*:  
 decrypt\_block(outdec.data(), outenc.data(), ks, outenc.size(), synhro);  
 break;  
 case Mode::*GAMMING*:  
 decrypt\_gamming(outdec.data(), outenc.data(), ks, outenc.size(), synhro);  
 break;  
 case Mode::*COUNTER*:  
 decrypt\_counter(outdec.data(), outenc.data(), ks, outenc.size(), synhro);  
 break;  
 }  
  
 write\_str\_to\_file(outdec, decrypted\_text\_file);  
}

**utility.cpp**

#include "utility.h"  
  
uint32\_t load32(const void \*in) {  
 const uint8\_t \*p = (const uint8\_t \*)in;  
 return ((uint32\_t)p[0] << 0) | ((uint32\_t)p[1] << 8) |  
 ((uint32\_t)p[2] << 16) | ((uint32\_t)p[3] << 24);  
}  
  
void store32(void \*out, const uint32\_t v) {  
 uint8\_t \*p = (uint8\_t \*)out;  
 p[0] = (uint8\_t)(v >> 0);  
 p[1] = (uint8\_t)(v >> 8);  
 p[2] = (uint8\_t)(v >> 16);  
 p[3] = (uint8\_t)(v >> 24);  
}  
  
void store128(void \*out, const uint8\_t \*ins) {  
 uint8\_t \*p = (uint8\_t \*)out;  
 for (std::size\_t i = 0; i < 16; ++i) {  
 p[i] = ins[i];  
 }  
}  
  
void square\_plus(void \*out, const uint8\_t \*ins) {  
 uint8\_t \*p = (uint8\_t \*)out;  
 for (std::size\_t i = 15;; i = (i + 15) % 16) {  
 if (p[i] == (uint8\_t)((1 << 8) - 1))  
 p[i] = 0;  
 else {  
 ++p[i];  
 break;  
 }  
 }  
}