МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ижевский государственный технический университет имени М.Т.Калашникова»

Кафедра «Вычислительная техника»

Отчет

По лабораторной работе №3

по дисциплине

**Методы и средства защиты информации**

по теме **«**Поточное симметричное шифрование**»**

|  |  |
| --- | --- |
| Выполнил | студент гр. Б08-781-1  Суровцева А.С. |
| Принял | к.т.н., доцент кафедры ВТ  Марков Е.М. |

Ижевск 2020

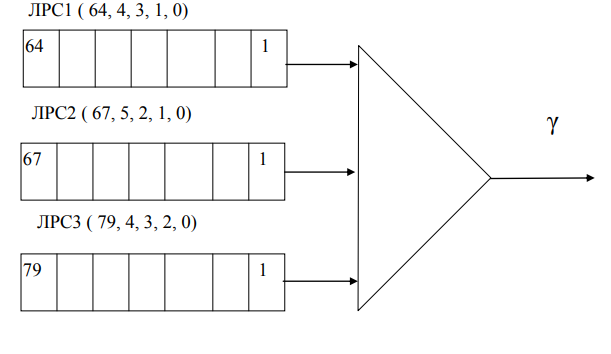
**Цель работы:**изучение структуры и основных принципов работы современных алгоритмов поточного симметричного шифрования, приобретение навыков программной реализации поточных симметричных шифров.

**Задание:** Реализовать в программе поточное кодирование текста, вводимого с

клавиатуры, с помощью заданной нелинейной схемы, использующей

пороговую функцию – если выход двух и более ЛРС 1, то выход гаммы равен

1, иначе – 0.



**Основные сведения:** Поточные шифры характерны тем, что шифруют информацию по одному биту за такт шифрования. Учитывая, что среди операций с битами существуют только две обратимые – сумма по модулю 2 и логическое отрицание, то выбор принципа шифрования очевиден – биты открытого текста должны складываться с битами ключевой последовательности с помощью операции ⊕: *ci = mi* ⊕ *ki.*

Дешифрование происходит аналогичным образом:

*mi = ci* ⊕ *ki.*

Учитывая свойства операции сложения по модулю 2, можно отметить, что выполняется:

*ki = ci* ⊕ *mi,*

поэтому криптостойкость поточных шифров полностью зависит от качества генератора потока ключей. Очевидно, что если поток ключей будет включать в себя только двоичные нули, то шифротекст будет представлять собой точную копию открытого текста. Поток ключей поточных шифров принято обозначать греческой буквой γ (гамма), вследствие чего подобные шифры получили название шифров гаммирования.

**Основные методы формирования γ в современной потоковой криптографии:** Очень популярны для решения этой цели регистры сдвига с обратной связью. Он представляет собой (рис.1) последовательность бит, которая на каждом такте шифрования сдвигается вправо на 1 разряд, при этом выход из крайнего правого бита является выходом генератора, а на вход крайнего левого бита подается значение, вычисляемое как некоторая функция от отдельных битов регистра. Ключ шифрования поточного шифра заносится в регистр перед началом генерации гаммы.

Самым простым способом формирования обратной связи является суммирование по модулю 2 отдельных разрядов регистра. На рис.2 изображен регистр сдвига с линейной обратной связью,

bn

bn-1

bn-2

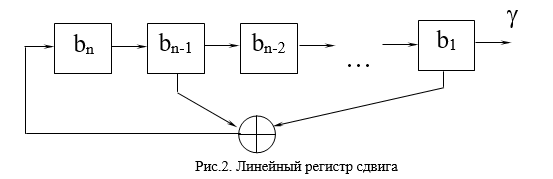
b1

…

γ

Рис.1. Регистр сдвига с обратной связью

Функция обратной связи



Рассмотрим работу ЛРС на примере трехразрядного регистра, структура которого приведена на рис.3

3

2

1

γ

Рис3 Линейный регистр сдвига на три разряда

Занесем в регистр начальное значение 010 и посмотрим, какие значение получим на выходе гаммы (табл.1).

Таблица 1. Результат работы генератора гаммы на основе ЛРС

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер такта | Значения битов ЛРС | | | Бит  гаммы |
| 3 | 2 | 1 |
| нач.сост | 0 | 1 | 0 | - |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 1 | 1 | 1 | 0 |
| 5 | 0 | 1 | 1 | 1 |
| 6 | 1 | 0 | 1 | 1 |
| 7 | 0 | 1 | 0 | 1 |
| 8 | 0 | 0 | 1 | 0 |

Из таблицы видно, что состояние ЛРС повторяется через 7 тактов (начальное состояние ЛРС совпадает с его состоянием на 7-м такте). Повтор состояния ЛРС означает, что и гамма будет периодически повторяться. Повторение гаммы снижает криптостойкость поточных шифров, позволяя криптоаналитику проводить анализ шифротекстов, полученных кодированием на одной и той же гамме. Поэтому при проектировании структуры ЛРС встает проблема достижения максимального периода повтора ЛРС. Для ЛРС длиной *n* бит максимальный период составляет 2*n*-1 тактов (состояние, когда все биты равны нулю, недопустимо, поскольку ЛРС любой структуры не выходит из этого состояния, зацикливаясь в нем).

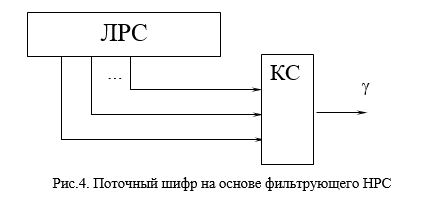
**Построение ЛРС оптимальной структуры** с точки зрения периода повторения гаммы имеет четкую математическую основу в виде теории неприводимых полиномов. Структура ЛРС описывается многочленом вида:

*bn\*xn+bn-1\*xn-1+bn-2\*xn-2+…+b2\*x2+ b1\*x+*1*,* (1)

где *bi*=0, если *i*-й бит не участвует в обратной связи, и *bi*=1, если участвует. ЛРС будет иметь максимально возможный период повторения гаммы, если описывающий его многочлен не раскладывается на произведение многочленов меньшей степени, то есть является примитивным по модулю 2. В контексте (1) структуру ЛРС принято коротко обозначать записью вида (43, 21, 5, 4, 1, 0), что в данном конкретном случае означает построение обратной линейной связи на сорок третьем, двадцать первом, пятом, четвертом и первом разрядах ЛРС.

**Основной проблемой ЛРС является их нестойкость к атаке на основе известного открытого текста.** Даже если неизвестна внутренняя структура ЛРС, криптоаналитик с помощью алгоритма Берлекэмпа-Мэсси по известным 2*N* битам открытого текста и соответствующего шифротекста имеет возможность построить ЛРС, порождающую подобную последовательность (проблема линейной сложности ЛРС). Поэтому **современные поточные шифры строятся на основе нелинейных схемах объединения ЛРС, которые добавляют в структуру нелинейные элементы**: логическое сложение и логическое умножение. Наиболее популярными классами нелинейных схем подключения на сегодня являются фильтрующие, комбинирующие и динамические поточные шифры [4].

*Фильтрующие* схемы строятся с использованием дополнительной комбинационной схемы – фильтра – на выходах некоторых бит ЛРС (рис.4). Выход комбинационной схемы и является гаммой.



*Комбинирующие* схемы также используют комбинационную схему с нелинейными преобразованиями бит, но на вход этой комбинационной схемы подаются выходы нескольких ЛРС (рис.5).

ЛРС1

ЛРС2

ЛРСN

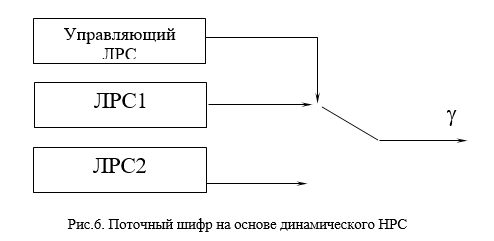
…

KC

γ

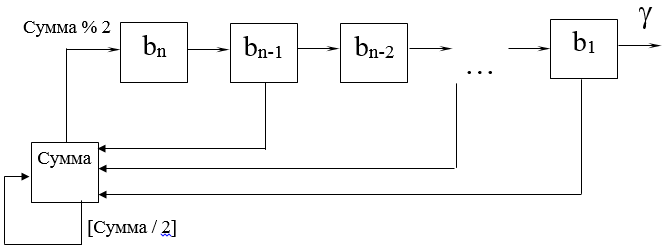
Рис.5. Поточный шифр на основе комбинирующей схемы

*Динамические* схемы объединения ЛРС предполагают отношения «главный-подчиненный» между отдельными регистрами. Например, на схеме рис. 6. зависимости от выхода управляющего ЛРС на общий выход гаммы подается либо выход первого, либо второго ЛРС.



Существуют также схемы динамического подключения ЛРС с использованием управляемого тактирования, когда сдвиг управляемого ЛРС зависит от состояния некоторого бита управляющего ЛРС.

Еще одной разновидностью сдвиговых регистров, использующихся для генерации потока ключей в потоковых шифрах, являются **сдвиговые регистры с обратной связью по переносу (РОСП)**.



В регистрах данного типа значение младшего бита формируется после суммированием всех бит обратной связи и содержимого регистра переноса. Остаток от деления на 2 получившейся суммы записывается в младший бит регистра, а результат деления нацело – в регистр переноса. Размер регистра переноса в битах должен быть равен [log2 *t* ], где *t* – количество ответвлений обратной связи. Максимальный период РОСП равен *q*-1, где *q* – целое число связи, его значение вычисляется по отводам обратной связи:

*q*= *q1\*2+ q2\*22+ q3\*23+…+ qn-1\*2n*-1*,*

где *qi* отсчитываются от левого края РОСП.

**WAKE** - сокращение от Word Auto Key Encryption (Автоматическое шифрование слов ключом). Алгоритм выдает поток 32-битовых слов, которые с помощью XOR могут быть использованы для получения шифротекста из открытого текста или открытого текста из шифротекста.

WAKE работает в режиме CFB, для генерации следующего слова ключа используется предыдущее слово шифротекста. Алгоритм также использует *S*-блок из 256 32-битовых значений. Содержимое *S*-блока наполняется по следующему принципу: старший байт всех элементов представляет собой перестановку всех возможных байтов, а в 3 младших байта заносятся случайные значения.

**RC4** — это потоковый шифр, широко применяющийся в различных системах защиты информации в компьютерных сетях (например, в протоколе SSL и для шифрования паролей в Windows NT). Алгоритм RC4 строится, как и любой потоковый шифр, на основе параметризованного ключом генератора псевдослучайных битов с равномерным распределением. Основные преимущества шифра — высокая скорость работы и переменный размер ключа. Типичная реализация выполняет 19 машинных команд на каждый байт текста.

Ядро алгоритма состоит из функции генерации ключевого потока. Эта функция генерирует последовательность битов, которая затем объединяется с открытым текстом посредством суммирования по модулю два. Дешифрация состоит из регенерации этого ключевого потока и суммирования его с шифрограммой по модулю два, восстанавливая исходный текст. Другая главная часть алгоритма — функция инициализации, которая использует ключ переменной длины для создания начального состояния генератора ключевого потока.

RC4 — фактически класс алгоритмов, определяемых размером его блока. Этот параметр n является размером слова для алгоритма. Обычно, *n* = 8, но в целях анализа можно уменьшить его. Однако для повышения безопасности необходимо увеличить эту величину. Внутреннее состояние RC4 состоит из массива размером 2*n* слов и двух счетчиков, каждый размером в одно слово. Массив известен как *S*-box, и далее будет обозначаться как *S*. Он всегда содержит перестановку 2*n* возможных значений слова. Два счетчика обозначены через *i* и *j*.

Алгоритм инициализации RC4 приведен ниже. Этот алгоритм использует ключ *Key*, имеющий длину l байт. Инициализация начинается с заполнения массива *S*, далее этот массив перемешивается путем перестановок определяемых ключом.

Начальное заполнение массива *S*:

for i = 0 to 2n − 1

S [i] = i

Следующий этап – перестановка элементов *S*, параметризуемая ключом:

j = 0 for i = 0 to 2n − 1:

j=(j+S[i]+Key[i])mod2n   
 Перестановка (S [i] , S [j])

Генератор ключевого потока RC4 переставляет значения, хранящиеся в *S*, и каждый раз выбирает различное значение из *S* в качестве результата. В одном цикле RC4 определяется одно *n*-битное слово *K* из ключевого потока, которое в последующем суммируется с исходным текстом для получения зашифрованного текста.

Инициализация: i = 0 j = 0

Цикл генерации: i = (i + 1) mod 2n

j = (j + S [i] ) mod 2n

Перестановка (S [i], S [j])

Результат: K = S [(S [i] + S [j] ) mod 2n]

Алгоритм RC4 устойчив к дифференциальному и линейному криптоанализу, в нем нет коротких циклов, он нелинеен. При *n*=8 RC4 может находиться в примерно 21700(256! \* 2562) различных состояниях.

Известно, что шифр безопасен, если размер ключа — по крайней мере, 128 битов ( 16 байтов). Это подтверждается сообщениями о некоторых атаках для малых размеров ключей (меньше, чем 5 байтов). Протоколы, которые сегодня использует *RC4*, устанавливают размеры ключей, которые делают *RC4* безопасным. Однако, как и для многих других шифров, рекомендуется, чтобы для различных сеансов использовались различные ключи.

**Безопасность:** в отличие от современных шифров, RC4 не использует отдельной оказии (англ. Nonce) наряду с ключом. Это значит, что если один ключ должен использоваться в течение долгого времени для шифрования нескольких потоков, сама криптосистема, использующая RC4, должна комбинировать оказию и долгосрочный ключ для получения потокового ключа для RC4. Один из возможных выходов – генерировать новый ключ для RC4 с помощью хэш-функции от долгосрочного ключа и оказии.

**Листинг программы**

#include <fstream>

#include <iostream>

#include <bitset>

using namespace std;

bitset<65> LRC1(6525926952262);

bitset<68> LRC2(226523215896315556);

bitset<80> LRC3(7356231968533);

void Init\_LRC()

{

LRC1.reset(); LRC1 |= 6525926952262;

LRC2.reset(); LRC2 |= 226523215896315556;

LRC3.reset(); LRC3 |= 7356231968533;

}

char Generator\_LRC() {

char OutGamma = 0;

for (int j = 0; j < 8; j++)

{

char bufGamma = 0;

if ((LRC1[0] == 1 && LRC2[0] == 1) || (LRC2[0] == 1 && LRC3[0] == 1) || (LRC1[0] == 1 && LRC3[0] == 1)) bufGamma = 1;

OutGamma <<= 1;

OutGamma |= bufGamma;

bool buf = LRC1[64] ^ LRC1[4] ^ LRC1[3] ^ LRC1[1];

LRC1 >>= 1; LRC1[64] = buf;

buf = LRC2[67] ^ LRC2[5] ^ LRC2[2] ^ LRC2[1];

LRC2 >>= 1; LRC2[67] = buf;

buf = LRC3[79] ^ LRC3[4] ^ LRC3[3] ^ LRC3[2];

LRC3 >>= 1; LRC3[79] = buf;

}

return OutGamma;

}

int main() {

int kb = 0;

char buf[16];

bool start = true;

setlocale(LC\_ALL, "Russian");

while (start) {

system("cls");

cout << "Введи 1 для шифрования или 2 для расшифрования, а для выхода нажми 3" << endl;

int k = 0; cin >> k;

switch (k)

{

case 1: {

Init\_LRC();

kb = 0;

ifstream inFile("1.bmp", ios\_base::binary);

ofstream outFile("Encrypt.bmp", ios\_base::binary | ios\_base::trunc);

cout << "Шифрование..." << endl;

while (!inFile.eof()) {

inFile.read(buf, 16);

if (kb >= 128)

for (int i = 0; i < inFile.gcount(); i++)

buf[i] = buf[i] ^ Generator\_LRC();

kb++;

outFile.write(buf, inFile.gcount());

}

cout << "Шифрование завершено" << endl;

inFile.close(); outFile.close(); system("pause");

break;

}

case 2: {

Init\_LRC();

kb = 0;

ifstream inFile("Encrypt.bmp", ios\_base::binary);

ofstream outFile("Decrypt.bmp", ios\_base::binary | ios\_base::trunc);

cout << "Расшифровка..." << endl;

while (!inFile.eof()) {

inFile.read(buf, 16);

if (kb >= 128)

for (int i = 0; i < inFile.gcount(); i++)

buf[i] = buf[i] ^ Generator\_LRC();

kb++;

outFile.write(buf, inFile.gcount());

}

cout << "Расшифровка завершена" << endl;

inFile.close(); outFile.close(); system("pause");

break;

}

case 3: //exit

{

start = false;

break;

}

default:

{

cout << "Неверное значение" << endl;

system("pause");

break; }

}

}

return 0;

}

**Результаты работы**

