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**

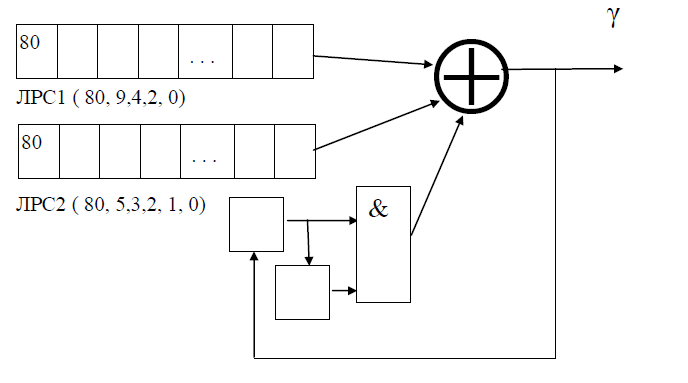
по теме **«**Поточное симметричное шифрование**»**

|  |  |
| --- | --- |
| Выполнил | студент гр. Б08-781-1  Мушегов В.А. |
| 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 дополнительные ячейки памяти РС.



**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

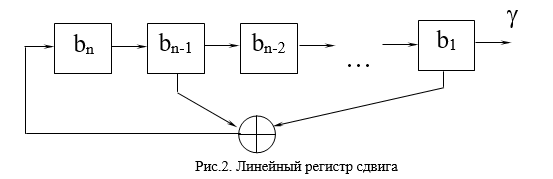
b1

…

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).

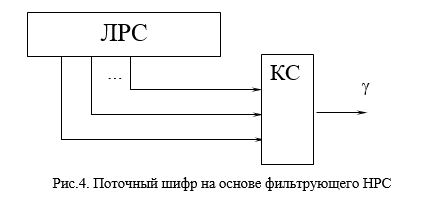
**Построение ЛРС оптимальной структуры** с точки зрения периода повторения гаммы имеет четкую математическую основу в виде теории неприводимых полиномов. Структура ЛРС описывается многочленом вида:

*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

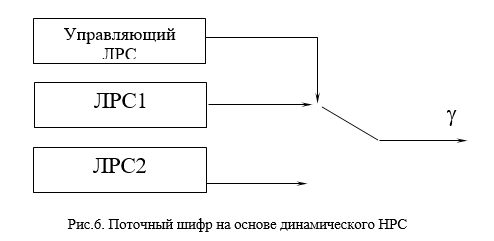
…

KC

γ

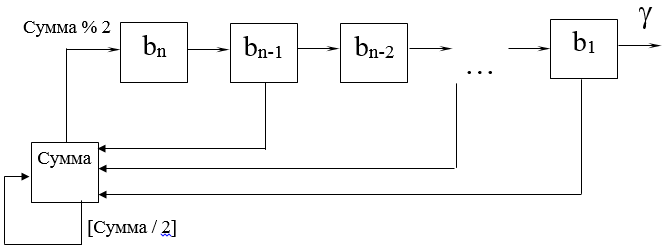
Рис.5. Поточный шифр на основе комбинирующей схемы

*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**

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab3

{

class Program

{

static string pathFile1 = Directory.GetCurrentDirectory() + "\\file1.bmp";

static string pathFile2 = Directory.GetCurrentDirectory() + "\\file2.bmp";

static string pathFile3 = Directory.GetCurrentDirectory() + "\\file3.bmp";

static bool gammaResult;

static BitArray lrs1;

static BitArray lrs2;

static bool buf1 = false, buf2 = false;

static void Main(string[] args)

{

bool exit = true;

while (exit)

{

Console.Clear();

Console.WriteLine("1.Ввести ключ");

Console.WriteLine("2.Зашифровать файл");

Console.WriteLine("3.Расшифровать файл");

Console.WriteLine("0.Выход");

ConsoleKeyInfo cki = Console.ReadKey();

switch (cki.KeyChar)

{

case '1':

GenerateKey();

break;

case '2':

EncryptionFile(pathFile1, pathFile2, true);

break;

case '3':

EncryptionFile(pathFile2, pathFile3, false);

break;

case '0':

exit = false;

break;

default:

break;

}

}

}

/// <summary>

/// Шифрование файла

/// </summary>

static void EncryptionFile(string pathFileIn, string pathFileOut, bool isEcrypt)

{

Console.Clear();

FileStream fileIn = null;

FileStream fileOut = null;

try

{

fileIn = new FileStream(pathFileIn, FileMode.Open, FileAccess.Read);

fileOut = new FileStream(pathFileOut, FileMode.Create, FileAccess.Write);

byte[] t = new byte[40];

fileIn.Read(t, 0, 40);

fileOut.Write(t, 0, 40);

byte[] temp = new byte[1];

byte gamma = 0;

while (fileIn.Length > fileIn.Position)

{

gamma = 0;

fileIn.Read(temp, 0, 1);

if (GammaNext())

gamma = (byte)(gamma ^ 1);

for (int i = 0; i < 7; i++)

{

if(GammaNext())

gamma = (byte)(gamma ^ 1);

gamma = (byte)(gamma << 1);

}

temp[0] = (byte)(temp[0] ^ gamma);

fileOut.Write(temp, 0, 1);

}

if (isEcrypt)

{

Console.WriteLine("Файл зашифрован " + pathFileOut);

}

else

{

Console.WriteLine("Файл расшифрован " + pathFileOut);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.WriteLine("Ошибка! Не удалось найти или прочитать файл!");

}

finally

{

if (fileIn != null)

fileIn.Close();

if (fileOut != null)

fileOut.Close();

}

Console.ReadKey();

}

public static bool GammaNext()

{

bool temp1 = lrs1[80] ^ lrs1[9] ^ lrs1[4] ^ lrs1[2] ^ lrs1[0];

bool temp2 = lrs2[80] ^ lrs2[5] ^ lrs2[3] ^ lrs2[2] ^ lrs2[1] ^ lrs2[0];

gammaResult = lrs1[0] ^ lrs2[0] ^ (buf1 & buf2);

buf2 = buf1;

buf1 = gammaResult;

for(int i = 0; i < 80; i++)

{

lrs1[i] = lrs1[i + 1];

lrs2[i] = lrs2[i + 1];

}

lrs1[80] = temp1;

lrs2[80] = temp2;

return gammaResult;

}

static void GenerateKey()

{

Console.Clear();

gammaResult = false;

lrs1 = new BitArray(81);

lrs1[0] = true;

lrs2 = new BitArray(81);

lrs2[80] = true;

buf1 = false;

buf2 = false;

Console.WriteLine("Начальные состояния ЛРС заданны");

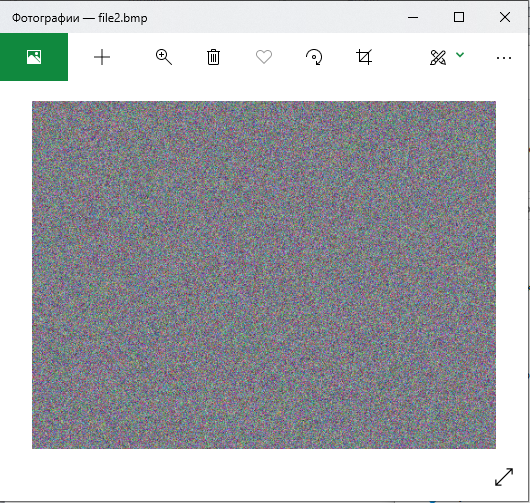
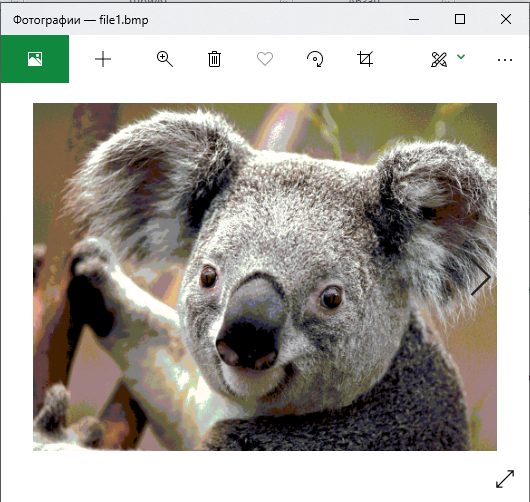
Console.ReadKey();

}

}

}

**Результаты работы**



**Ответы на вопросы**

* 1. Для чего был разработан, где применяется.

Пото́чный или Пото́ковый шифр — это симметричный шифр, в котором каждый символ открытого текста преобразуется в символ шифрованного текста в зависимости не только от используемого ключа, но и от его расположения в потоке открытого текста. Поточный шифр реализует другой подход к симметричному шифрованию, нежели блочные шифры.

Потоковые шифры на базе сдвиговых регистров активно использовались в годы войны, ещё задолго до появления электроники. Они были просты в проектировании и реализации.

В 1965 году Эрнст Селмер, главный криптограф норвежского правительства, разработал теорию последовательности сдвиговых регистров. Позже Соломон Голомб, математик Агентства Национальной Безопасности США, написал книгу под названием «Shift Register Sequences» («Последовательности сдвиговых регистров»), в которой изложил свои основные достижения в этой области, а также достижения Селмера.

Большую популярность потоковым шифрам принесла работа Клода Шеннона, опубликованная в 1949 году, в которой Шеннон доказал абсолютную стойкость шифра Вернама (также известного, как одноразовый блокнот). В шифре Вернама ключ имеет длину, равную длине самого передаваемого сообщения. Ключ используется в качестве гаммы, и если каждый бит ключа выбирается случайно, то вскрыть шифр невозможно (так как все возможные открытые тексты будут равновероятны). К настоящему времени создано большое количество алгоритмов потокового шифрования, таких как: A3, A5, A8, MUGI, PIKE, RC4, SEAL.

* 1. Принцип работы и характеристики

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.

* 1. Криптостойкость для алгоритма

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

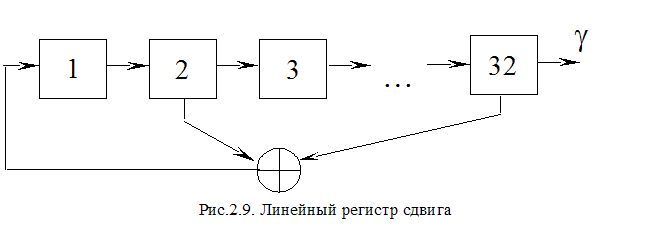
Дешифрование происходит аналогичным образом:

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. Большинство современных генераторов гаммы построено на линейных регистрах сдвига (ЛРС). Он представляет собой (рис.2.9) 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 сумма по модулю 2 нескольких разрядов ЛРС. Ключ шифрования поточного шифра заносится в ЛРС перед началом генерации гаммы.



* 1. Криптоанализ

Силовые атаки

Полный перебор

К этому классу относятся атаки, осуществляющие полный перебор всех возможных вариантов. Сложность полного перебора зависит от количества всех возможных решений задачи (в частности, размера пространства ключей или пространства открытых текстов). Этот класс атак применим ко всем видам систем поточного шифрования. При разработке систем шифрования разработчики стремятся сделать так, чтобы этот вид атак был наиболее эффективным по сравнению с другими существующими методами взлома.

Статистические атаки

Статистические атаки делятся на два подкласса:

- метод криптоанализа статистических свойств шифрующей гаммы: направлен на изучение выходной последовательности криптосистемы; криптоаналитик пытается установить значение следующего бита последовательности с вероятностью выше вероятности случайного выбора с помощью различных статистических тестов.

- метод криптоанализа сложности последовательности: криптоаналитик пытается найти способ генерировать последовательность, аналогичную гамме, но более просто реализуемым способом.

Оба метода используют принцип линейной сложности.

Аналитические атаки

Этот вид атак рассматривается в предположении, что криптоаналитику известны описание генератора, открытый и соответствующий закрытый тексты. Задача криптоаналитика определить использованный ключ (начальное заполнение регистров). Виды аналитических атак, применяемые к синхронным поточным шифрам:

- корреляционные

- компромисс “время-память”

- инверсионная

- “предполагай и определяй”

- на ключевую загрузку и реинициализацию

- XSL-атака

Корреляционные атаки являются наиболее распространёнными атаками для взлома поточных шифров.

Известно, что работа по вскрытию криптосистемы может быть значительно сокращена, если нелинейная функция пропускает на выход информацию о внутренних компонентах генератора. Поэтому для восстановления начального заполнения регистров корреляционные атаки исследуют корреляцию выходной последовательности шифросистемы с выходной последовательностью регистров.