УДК 004.42

Особенности реализации LFSR средствами языка программирования C#

Ибрагимов Тимур Рашидович
Смоленский государственный университет
магистрант

Аннотация
В статье затронута проблема генерации случайных чисел и задача генерации псевдослучайных ключей, способных обеспечить надежную криптостойкость шифра. Автором рассмотрены особенности реализации регистра сдвига с линейной обратной связью средствами языка C#, предложена конкретная программная реализация данного алгоритма. На примере приложения «Гаммирование» показано как устройство LSFR применяется в криптографии.
Ключевые слова: информатика, программирование, информационно-коммуникационные технологии, информация, криптография, LFSR, РСЛОС, C#, гаммирование, гамма, генератор случайных чисел, случайный ключ.

Features of the implementation of the LFSR by means of the programming language C#

Ibragimov Timur Rashidovich
Smolensk state University
Undergraduate

Abstract
The article raises the problem of generating random numbers and the task of generating pseudo-random keys that can provide reliable cryptographic strength of the cipher. The author considers the features of the implementation of the linear feedback shift register by C # means, a specific program implementation of this algorithm is proposed. The example of the application "Gumming" shows how the device LSFR is used in cryptography.
Keywords: informatics, programming, information and communication technologies, information, cryptography, LFSR, C#, gamming, gamma, random number generator, random key.

Генерация случайных ключей – одна из основных проблем криптографии и защиты информации в целом [1, 2, 3]. Криптостойкость большинства криптографических алгоритмов зависит в первую очередь от свойств ключа, в частности, от его длины и случайности. Однако любой способ генерации случайных чисел, использующий какой-либо алгоритм позволяет получать только псевдослучайные числа. Поэтому главной задачей

становится генерация такого псевдослучайного ключа, который обеспечит надежную криптостойкость шифра.

На практике для генерации последовательности псевдослучайных битов неплохо себя показал регистр сдвига с линейной обратной связью (РСЛОС, англ. linear feedback shift register, LFSR) реализуемый как аппаратно, так и программно [4, 5].

Устройство LFSR состоит из большого количества ячеек памяти, в каждой из которых находится по одному биту информации. Когда устройство начинает работу ячейки содержат секретный, случайный ключ. На каждом этапе работы информация, находящаяся в ячейках преобразуется с помощью функции, называемой функцией обратной связи.

Значение, возвращаемое функцией обратной связи, заносится в крайнюю левую ячейку S_{L-1} регистра, при этом все биты смещаются на одну ячейку вправо, а крайний правый бит выходит из регистра, и именно он является результатом работы (выходным значением) регистра на данном шаге (рис. 1). При этом биты, являющиеся переменными функции обратной связи, называются отводами, а сам регистр называется конфигурацией Фибоначчи [6].

Рисунок 1 – Схема работы LFSR

Текст, зашифрованный с помощью только одного устройства LFSR, имеет низкую криптостойкость. На практике для увеличения криптостойкости используют несколько регистров LFSR. Пусть, например размер секретного ключа равен L. Поделим этот ключ между n регистрами LFSR длин L_i так, чтобы $L_1 + L_2 + \ldots + L_n = L$, и скомбинируем выходы всех n регистров с помощью некоторой булевой функции f (рис. 2)

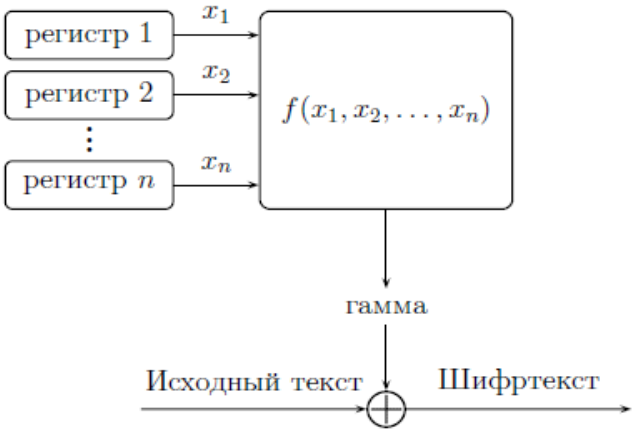

Рисунок 2 – Схема использования нескольких регистров

Такая схема уже соответствует реальному, используемому на практике, механизму шифрования информации [7]. В нем для увеличения криптостойкости можно использовать латентные параметры объектов шифрования информационных систем [8, 9, 10, 11].

Ярким примером применения устройства LFSR в криптографии является шифр гаммирование. При использовании гаммирования процедура шифрования сводится к генерации гаммы шифра (случайной последовательности чисел) и наложении этой гаммы на открытое сообщение обратимым образом, например с использованием операции сложения по модулю 2 [12, 13].

Один из способов реализации LFSR средствами языка C# и пример применения этого устройства в шифровании продемонстрированы в приложении «Гаммирование». Данное приложение создано в интегрированной среде разработки MS Visual Studio. Приложение имеет пользовательский интерфейс (рис. 3) и тип Windows Forms [14].

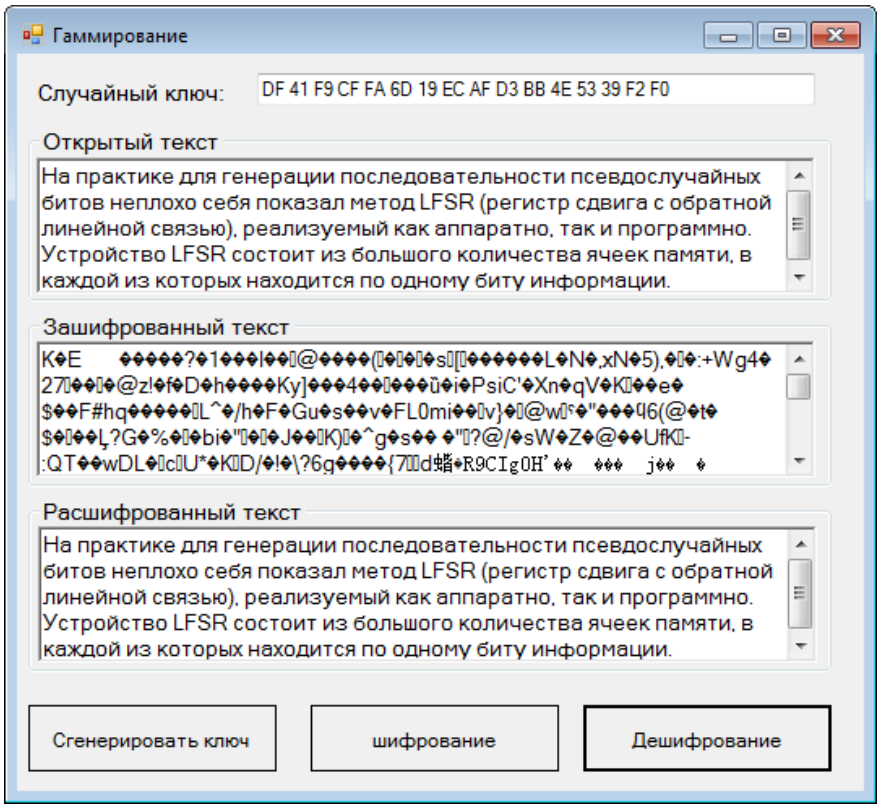

Рисунок 3 – пользовательский интерфейс

Каждый бит гаммы генерируется при помощи комбинирования результатов работы четырех устройств LFSR, каждое из которых использует свой секретный ключ. Данная реализация соответствует схеме, которая изображена выше (рис. 2).

Ключ, который определяет начальное состояние всех четырех регистров генерируется длинной 128 битов (16 байтов) и далее делится на 4 равные части по 32 бита для регистра каждого устройства. Для генерации ключа используется класс RNGCryptoServiceProvider пространства имен

System.Security.Cryptography [15]. Полученный ключ хранится в виде массива байтов. Задачу генерации ключа выполняет метод CreateKey(), который вызывается при нажатии на кнопку «Сгенерировать ключ».

```
public void CreateKey()
{
   key = new byte[lengthKey];
   RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
   rng.GetBytes(key);
}
```

Затем ключ делится на четыре равные части, каждая из которых помещается в коллекцию типа BitArray. Полученные четыре коллекции являются начальными состояниями каждого регистра и имеют имена initialStateR1, initialStateR2 и т.д.

Устройство LFSR реализовано в одноименной функции, которая в качестве параметра принимает коллекцию типа BitArray и возвращает значение булевого типа. В переменную outBit помещается выходное значение регистра на данном шаге. Так как данное устройство реализует конфигурацию Фибоначчи, то для генерации нового бита, с использованием 32-битового сдвигового регистра при помощи операции XOR (сложение по модулю 2) суммируются 0-й, 25-й, 27-й, 29-й, 30-й, 31-й биты. Построенный таким образом NFSR имеет максимальный период $2^{32} – 1$.

```
public bool LFSR(BitArray initialStateRegister)
{
   bool outBit = initialStateRegister[0];
         bool   newBit   =   initialStateRegister[0]   ^   initialStateRegister[25]   ^
   initialStateRegister[27] ^ initialStateRegister[29] ^ initialStateRegister[30] ^
   initialStateRegister[31];
        //цикл выполняет сдвиг битов в ячейках на одну позицию
   for (int i = 1; i < initialStateRegister.Length; i++)
     initialStateRegister[i - 1] = initialStateRegister[i];

   initialStateRegister[31] = newBit;
   return outBit;
}
```

Гамма для шифра генерируется путем суммирования (при помощи операции XOR) на каждом шаге битов, полученных в результате работы четырех устройств LFSR. Эти операции выполняет метод GenGamma().

```
public bool GenGamma()
{
         bool    value    =    LFSR(initialStateR1)    ^    LFSR(initialStateR2)    ^
   LFSR(initialStateR3) ^
     LFSR(initialStateR4);
   return value;
}
```

Шифрование выполняет функция Cipher(String plainText), которая вызывается при нажатии на кнопку «Шифрование». Параметром для функции служит текст, считываемый из поля «Открытый текст».

Функция Cipher(String plainText) первым делом выполняет перевод строки отрытого текста в массив байтов plainBytes. Далее определяется размер полученного массива и создается пустой массив cipherText аналогичной длины для хранения зашифрованного текста. Текст шифруется побайтно. Первый (внешний) цикл проходит по каждому байту открытого текста и выполняет его шифрование, суммированием с байтом гаммы. Байт гамы вырабатывается во внутреннем цикле путем 8-кратного вызова метода GenGamma(). Результат работы функции – массив байт, который переводится в строковый формат и выводится на форме в поле «Зашифрованный текст».

```
public byte[] Cipher(String plainText)
{
   byte[] plainBytes = Encoding.UTF8.GetBytes(plainText);
   byte[] cipherText = new byte[plainBytes.Length];
   bool[] oneByteKey = new bool[8];

      for (int i = 0; i < plainBytes.Length; i++)
      {
         for (int j = 0; j < oneByteKey.Length; j++)
         {
             oneByteKey[j] = GenGamma();
         }
         cipherText[i] = (byte)(plainBytes[i] ^ ConvertBoolArrayToByte(oneByteKey));
      }
   return cipherText;
}
```

Для дешифрования зашифрованного текста необходимо нажать на кнопку «Дешифрование». Дешифрование выполняется аналогично шифрованию путем накладывания на массив байтов зашифрованного текста гаммы, вырабатываемой из того же секретного ключа тем же алгоритмом.

Результат работы приложения продемонстрирован на рисунке 3.

Для генерации псевдослучайных чисел существует множество алгоритмов. Описанное выше приложение демонстрирует способ реализации средствами языка C# одного из эффективных методов генерации гаммы шифра – LSFR. Также на примере реализации шифра гаммирования показано как устройство LSFR применяется в криптографии. Кроме того, в приложении представлена модификация генератора гаммы путем применения четырех регистров сдвига. Следует отметить, что для увеличения криптостойкости шифра количество используемых регистров и их размер легко увеличивается, а чтобы не получить провал в производительности эффективным и легко реализуемым является применение распараллеливания.

Библиографический список

1. Гатченко Н. А Криптографическая защита информации / Гатченко Н. А., Исаев А. С., Яковлев А. Д. СПб: НИУ ИТМО, 2012. 142 с.
2. Гатчин Ю. А. Основы криптографических алгоритмов. Учебное пособие / Гатчин Ю. А., Коробейников А. Г. СПб: ГИТМО (ТУ), 2002. 29 с.

3. Журавлёва У. С., Баженов Р. И. Нейронные сети в Scilab // Постулат. 2017. № 1 (15). С. 25.
4. Гомес Ж. Математики, шпионы и хакеры. Кодирование и криптография // Мир Математики. М.: Де Агостини, 2014. Т.34. 144 с.
5. Мао В. Современная криптография: Теория и практика. М.: Вильямс, 2005. 247 с.
6. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. М., 2013. 816 с.
7. Яблонский С. В. Введение в дискретную математику. М.: Высшая школа, 2008. 384 с.
8. Козлов С. В. Применение соответствия Галуа для анализа данных в информационных системах // Траектория науки. 2016. Т. 2. № 3 (8). С. 18.
9. Козлов С. В. Интерпретация инвариантов теории графов в контексте применения соответствия Галуа при создании и сопровождении информационных систем // International Journal of Open Information Technologies. 2016. Т. 4. № 7. С. 38-44.
10. Козлов С. В. Использование математического аппарата импликативных матриц при создании и сопровождении информационных систем // International Journal of Open Information Technologies. 2017. Т. 5. № 12. С. 16-23.
11. Козлов С. В. Использование соответствия Галуа как инварианта отбора контента при проектировании информационных систем // Современные информационные технологии и ИТ-образование. 2015. Т. 2. № 11. С. 220-225.
12. Виноградова М.С Булевы функции: методические указания к выполнению типового расчета / Виноградова М. С., Ткачев С. Б. М., 2007. 32 с.
13. Логачев О. А Булевы функции в теории кодирования и криптологии / Логачев О. А., Сальников А. А., Ященко В. В. М.: МЦНМО, 2004. 470 с.
14. Суин И. А., Козлов С. В. Особенности реализации графических решений при разработке пользовательских приложений в объектно-ориентированных средах программирования // Международный студенческий научный вестник. 2017. № 5. С. 53.
15. Ибрагтмов Т. Р., Козлов С. В. Особенности использования алгоритмов DES и 3DES в языке C# // Постулат. 2017. № 12. С. 22.