Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа №8

**ИССЛЕДОВАНИЕ ПОТОКОВЫХ ШИФРОВ**

Студент: Сосновец М.И.

ФИТ 3 курс 4 группа

Преподаватель: Нистюк О.А.

Минск 2025

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации потоковых шифров.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости потоковых шифров.
2. Разработать приложение для реализации указанных преподавателем методов генерации ключевой информации и ее использования для потокового зашифрования/расшифрования.
3. Выполнить анализ криптостойкости потоковых шифров.
4. Оценить скорость зашифрования/расшифрования реализованных шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

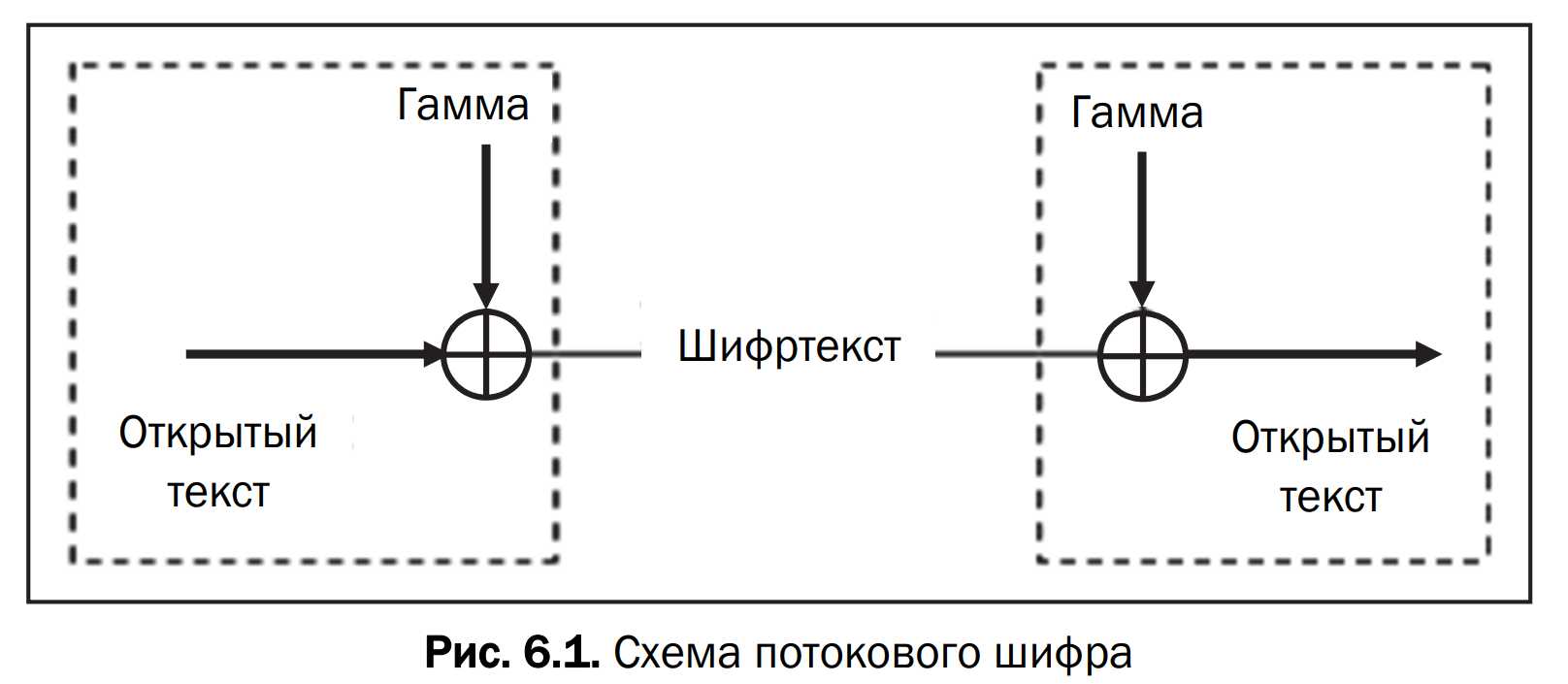
**Теоретические сведения**

Потоковый шифр (иногда говорят «поточный») – симметричный шифр, преобразующий каждый символ mi открытого текста в символ шифрованного ci, зависящий от ключа и расположения символа в тексте.

Термин «потоковый шифр» обычно используется в том случае, когда шифруемые символы открытого текста представляются одной буквой, битом или реже – байтом.

Все потоковые шифры делятся на 2 класса: синхронные и асинхронные (или самосинхронизирующиеся).

Основной задачей потоковых шифров является выработка некоторой последовательности (гаммы) для зашифрования, т. е. выходная гамма является ключевым потоком (ключом) для сообщения. В общем виде схема потокового шифра изображена на рис. 6.1



Синхронные потоковые шифры (СПШ) характеризуются тем, что поток ключей генерируется независимо от открытого текста и шифртекста. Главное свойство СПШ – нераспространение ошибок. Ошибки отсутствуют, пока работают синхронно шифровальное и дешифровальное устройства отправителя и получателя информации. Один из методов борьбы с рассинхронизацией – разбить отрытый текст на отрезки, начало и конец которых выделить вставкой контрольных меток (специальных маркеров).

Синхронные потоковые шифры уязвимы к атакам на основе изменения отдельных битов шифртекста.

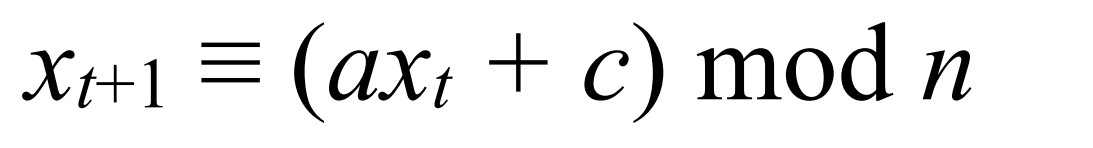
Недостаток этих потоковых шифров – распространение ошибок, так как искажение одного бита в процессе передачи шифртекста приведет к искажению нескольких битов гаммы и, соответственно, расшифрованного сообщения.

Ключевые последовательности (случайные последовательности (СП), либо псевдослучайные последовательности (ПСП)) вырабатываются специальными блоками систем потокового шифрования – генераторами.

Указанный стандарт определяет базовые понятия в рассматриваемой предметной области:

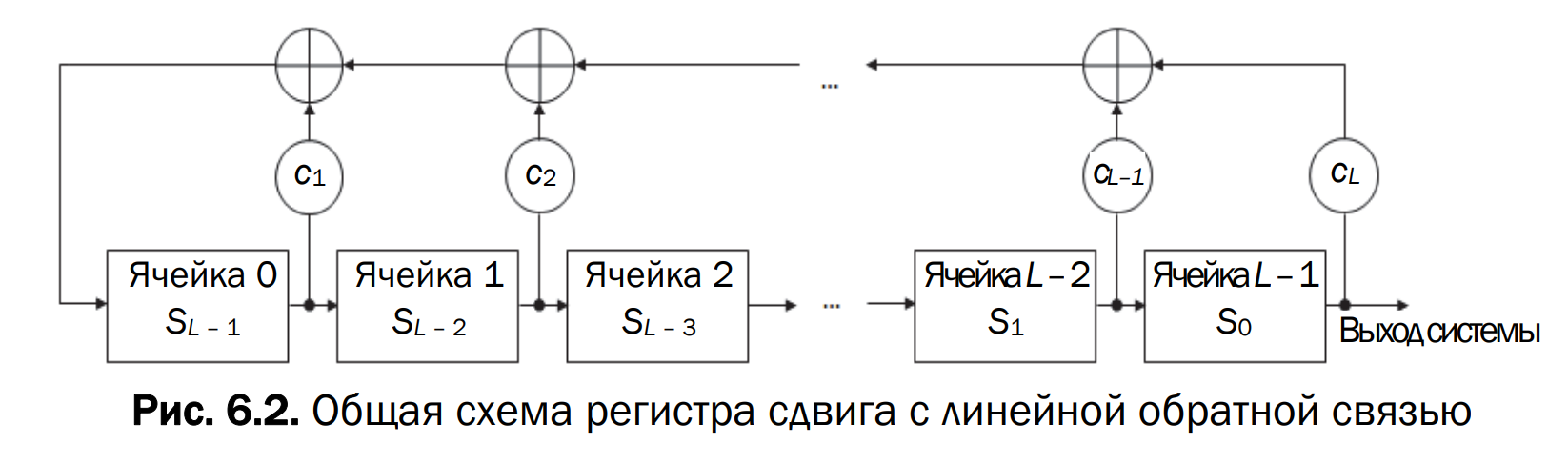
* случайные числа (последовательности) – последовательность элементов, каждый из которых не может быть предсказан (вычислен) только на основе знания предшествующих ему элементов данной последовательности;
* псевдослучайные числа – последовательность элементов, полученная в результате выполнения некоторого алгоритма и используемая в конкретном случае вместо последовательности случайных чисел.

Часто используемый алгоритм генерирования (программно или аппаратно) ПСП реализуется на основе так называемого линейного конгруэнтного генератора, описываемого следующим рекуррентным соотношением:

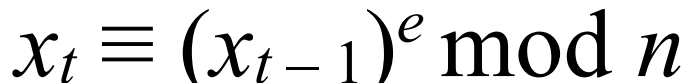


где xt и xt + 1 – соответственно t-й (предыдущий) и (t + 1)-й (текущий, вычисляемый) члены числовой последовательности; а, с и n – константы. Период такого генератора (период ПСП) не превышает n.

Достаточно распространенным является использование регистров сдвига (РС) в качестве генераторов ПСП в силу простоты реализации на основе цифровой логики. РС с линейной обратной связью (РСЛОС) состоит из двух частей: собственно РС и функции обратной связи. На рис. 6.2 представлена общая схема РС с линейной обратной связью. Функция обратной связи реализуется с помощью сумматоров сложения по модулю два (элементы XOR; на рис. 6.2 обозначены в виде кружочков со знаком сложения).

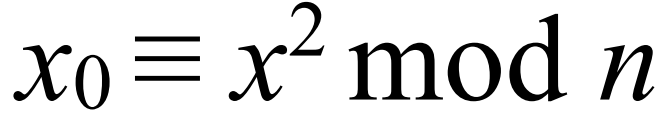


Алгоритм RSA разработан для систем асимметричного зашифрования/расшифрования и будет более детально рассмотрен с практической точки зрения ниже. Генератор же ПСП на основе RSA устроен следующим образом. Последовательность генерируется с использованием соотношения.



Широкое распространение получил алгоритм генерации ПСП, называемый алгоритмом BBS (от фамилий авторов: L. Blum, M. Blum, M. Shub) или генератором на основе квадратичных вычетов. Для целей криптографии этот метод предложен в 1986 г.

Начальное значение x0 генератора вычисляется на основе соотношения.



где n, как и в генераторе на основе RSA, является произведением простых чисел p и q, однако в нашем случае эти простые числа должны быть сравнимы с числом 3 по модулю 4, т. е. при делении p и q на 4 должен получаться одинаковый остаток 3; число x должно быть взаимно простым с n; число n называют числом Блюма.

Алгоритм RC4 разработан Р. Ривестом в 1987 г. Он представляет собой потоковый шифр с переменным размером ключа. Здесь гамма не зависит от открытого текста [5].

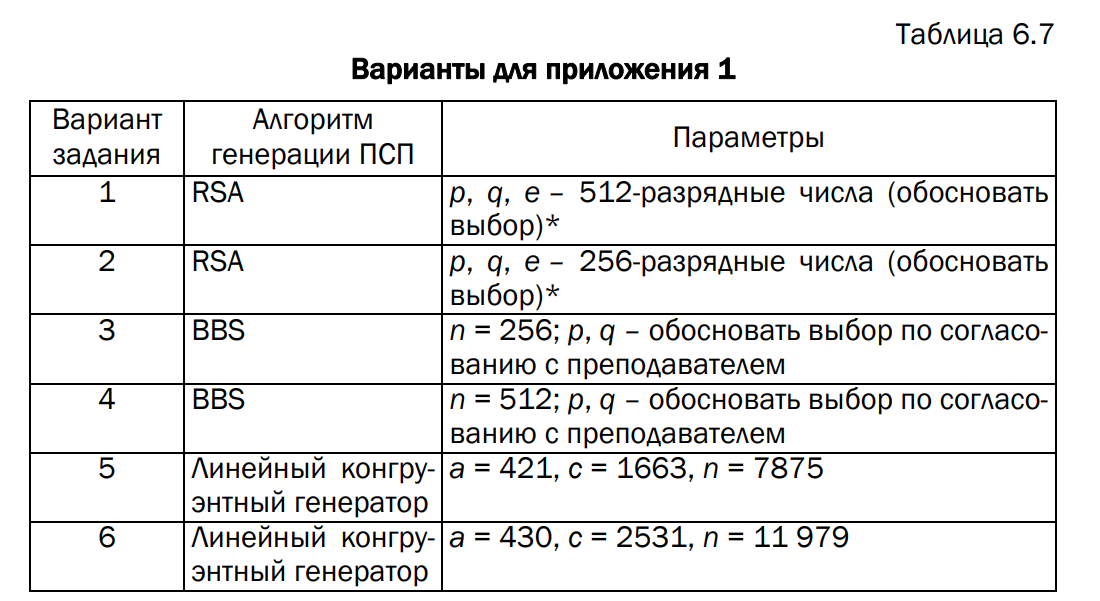
Алгоритм RC4, как и любой потоковый шифр, строится на основе генератора псевдослучайных битов (генератора ПСП). На вход генератора записывается ключ, а на выходе читаются псевдослучайные биты. Длина ключа может составлять от 40 до 2048 битов.

Ядро алгоритма состоит из функции генерации ключевого потока. Другая часть алгоритма – функция инициализации, которая использует ключ переменной длины Ki для создания начального состояния генератора ключевого потока.

Шифр RC4 применяется в некоторых широко распространенных стандартах и протоколах шифрования, таких как WEP, WPA и TLS, а также в Kerberos и др.

**Практическое задание**

Приложение 1 должно реализовывать генерацию ПСП в соответствии с вариантом из табл. 6.7. Был реализован 6 вариант.



Линейный конгруэнтный генератор (ЛКГ) — это способ создания псевдослучайных чисел.

Для реализации алгоритма генерации псевдослучайных последовательностей был создан класс, который принимает два параметра: seed – начальное значение, count – количество чисел, которое надо сгенерировать представленный на листинге 1.1.

|  |
| --- |
| function generateLCG(seed, count) {    const a = 430;    const c = 2531;    const n = 11979;    let current = seed;    const sequence = [];    for (let i = 0; i < count; i++) {      current = (a \* current + c) % n;      sequence.push(current);    }    return sequence;  } |

Листинг 1.1 – Реализация алгоритма генерации псевдослучайных последовательностей

В соответствии с вариантом, на вход алгоритма подаются следующие числа: а = 430, с = 2531, n = 11979. Результат работы алгоритма представлен на рисунке 1.1.

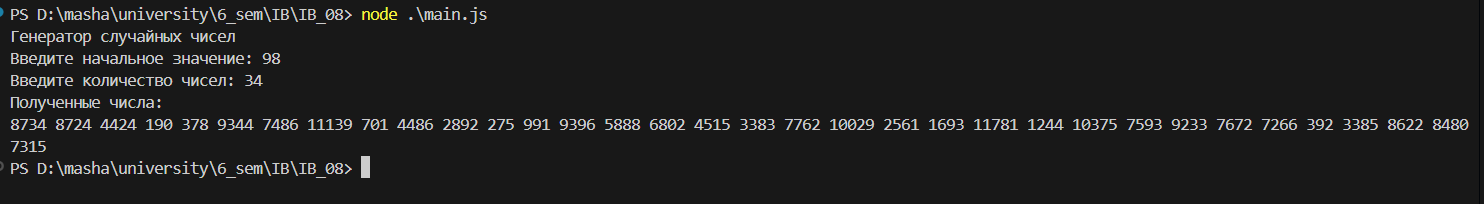
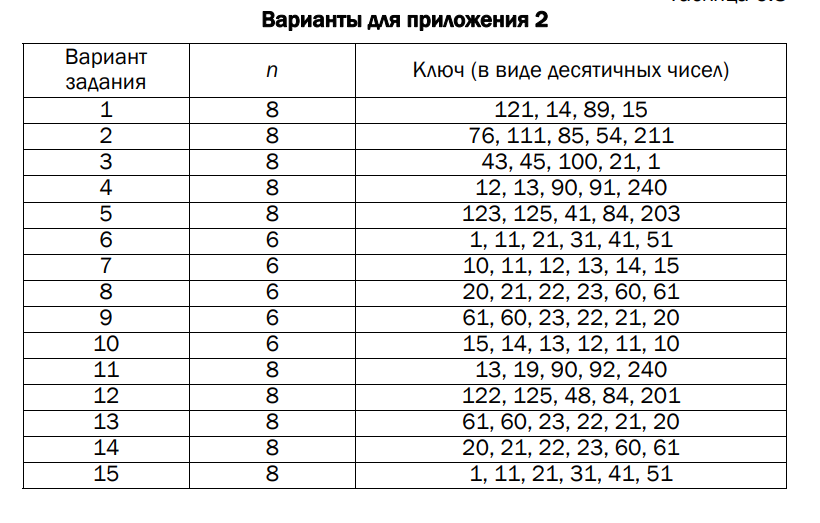


Рисунок 1.1 – Результат работы алгоритма

Приложение 2 должно реализовывать алгоритм RC4 в соответствии с вариантом из табл. 6.8, а также дополнительно выполнять оценку скорости выполнения операций генерации ПСП.



RC4 — это способ шифрования текста, который придумал Рон Ривест в 1987 году. Он создает длинную последовательность чисел, которая выглядит случайной (называется псевдослучайный поток, или ПСП), и использует её, чтобы "замаскировать" текст. RC4 часто применяется там, где нужно быстро шифровать данные, например, в старых версиях Wi-Fi (WEP) или в некоторых протоколах.

RC4 работает так:

1. Берет ключ (122, 125, 48, 84, 201).
2. Создает на его основе "перемешанную" таблицу чисел.
3. Использует эту таблицу, чтобы генерировать числа (ПСП).
4. Применяет эти числа к тексту, чтобы зашифровать его (или расшифровать).

RC4 состоит из двух главных частей: KSA (подготовка таблицы) и PRGA (генерация чисел для шифрования).

Программная реализация метода генерирования ПСП представленная в листинге 1.2.

|  |
| --- |
| function RC4(key) {    const S = new Array(256); // Таблица S (размер 2^8 = 256)    for (let i = 0; i < 256; i++) {      S[i] = i;    }    // KSA (Key-Scheduling Algorithm)    let j = 0;    for (let i = 0; i < 256; i++) {      j = (j + S[i] + key[i % key.length]) % 256;      [S[i], S[j]] = [S[j], S[i]]; // Swap}    // PRGA (Pseudo-Random Generation Algorithm)    function\* generateKeystream(length) {      let i = 0, j = 0;const S\_copy = [...S]; // Копируем S, чтобы не менять оригинал      for (let k = 0; k < length; k++) {        i = (i + 1) % 256;        j = (j + S\_copy[i]) % 256;        [S\_copy[i], S\_copy[j]] = [S\_copy[j], S\_copy[i]]; // Swap        const t = (S\_copy[i] + S\_copy[j]) % 256;        yield S\_copy[t];}}return {      generateKeystream,      encrypt: (text) => {        const textBytes = Buffer.from(text, 'utf8');   const keystream = generateKeystream(textBytes.length);        const encrypted = new Array(textBytes.length);        for (let i = 0; i < textBytes.length; i++) {    encrypted[i] = textBytes[i] ^ keystream.next().value; // XOR}return Buffer.from(encrypted);},      decrypt: (encrypted) => {  const keystream = generateKeystream(encrypted.length);        const decrypted = new Array(encrypted.length);        for (let i = 0; i < encrypted.length; i++) {decrypted[i] = encrypted[i] ^ keystream.next().value; //XOR}return Buffer.from(decrypted).toString('utf8');}};} |

Листинг 1.2 – Реализация метода генерирования ПСП

Результат работы алгоритма представлен на рисунке 1.2.

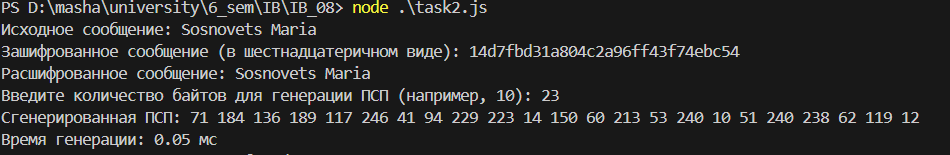


Рисунок 1.2 – Результат работы алгоритма

Оценка скорости выполнения операций генерации ПСП, реализация представлена в листинге 1.3.

|  |
| --- |
| function measureSpeed(generateKeystream, length) {    const start = performance.now();    const stream = [...generateKeystream(length)];    const end = performance.now();    const time = end - start;    const speed = (length / time).toFixed(2);    return { stream, time, speed };  } |

Листинг 1.3 – Оценка скорости выполнения операций генерации ПСП

Результат оценки показан на рисунке 1.3.

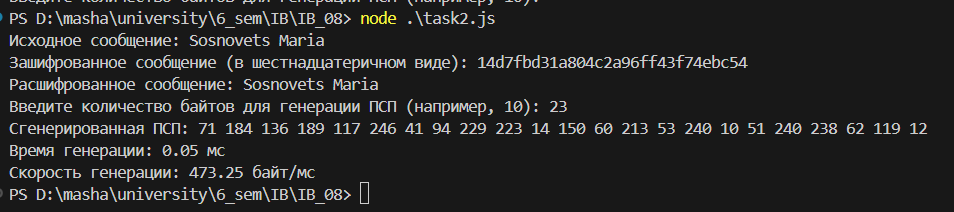


Рисунок 1.3 – Результат работы алгоритма

**Вывод:**

Алгоритм генерации ПСП линейного конгруэнтного генератора очень простой в реализации и понимании. Алгоритм шифрования также очень прост в реализации и понимании, он использует сгенерированную последовательность битов и при помощи операции XOR шифрует текст. Время выполнения генерации псевдослучайной последовательности битов происходит очень быстро поэтому данный алгоритм шифрования достаточно производительный.