МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Исследование потоковых шифров

Студент: Евсеенко В. П.

ФИТ 3 курс 4 группа

Преподаватель: Савельева М. Г.

Минск 2025

Содержание

[Введение 3](#_Toc195772734)

[1 Теоретические сведения 4](#_Toc195772735)

[2 Практическое задание 6](#_Toc195772736)

[Задание 1 6](#_Toc195772737)

[Задание 1 8](#_Toc195772738)

[Оценка времени 10](#_Toc195772739)

[Оценка криптостойкости 11](#_Toc195772740)

[Вывод 12](#_Toc195772741)

# **Введение**

Цель лабораторной работы:

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

Задачи:

* закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости потоковых шифров;
* разработать приложение для реализации указанных преподавателем методов генерации ключевой информации и ее использования для потокового зашифрования/расшифрования;
* выполнить анализ криптостойкости потоковых шифров;
* оценить скорость зашифрования/расшифрования реализованных шифров.

# **1 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.

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.

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.

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

В самосинхронизирующихся потоковых шифрах символы ключевой гаммы зависят от исходного секретного ключа шифра и от конечного числа последних знаков зашифрованного текста. Основная идея заключается в том, что внутреннее состояние генератора потока ключей является функцией фиксированного числа предыдущих битов шифртекста. Поэтому генератор потока ключей на приемной стороне, приняв фиксированное число битов, автоматически синхронизируется с генератором гаммы.

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

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

*xt*+1 ≡ (*a*\**xx* + *c*) mod *n*,

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*.

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 *x*0 генератора вычисляется на основе соотношения

*x*0 ≡ *x*2 mod *n*

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. Выходом генератора на *t*-м шаге является младший бит числа *xt*:

*xt* ≡ (*xt*– 1)2 mod *n*

Алгоритм RC4 представляет собой потоковый шифр с переменным размером ключа. Здесь гамма не зависит от открытого текста. 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. В основе алгоритма – размер блока или слова, определяемый параметром n. Обычно n = 8, но можно использовать и другие значения. Внутренне состояние шифра определяется массивом слов (S-блоком) размером 2n. При n = 8 элементы блока представляют собой перестановку чисел от 0 до 255, а сама перестановка зависит от ключа переменной длины. Другими элементами внутреннего состояния являются 2 счетчика с нулевыми начальными значениями. В основе вычислений лежит операция по mod2n.

# **2 Практическое задание**

# **Задание 1**

Разработать авторское приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться готовыми библиотеками либо программными кодами, реализующими некоторые блочные алгоритмы. Приложение должно реализовывать генерацию ПСП. Исследуемый алгоритм генерации ПСП и параметры – в соответствии с вариантом из табл. 2.1.

Таблица 2.1 – Варианты задания

|  |  |  |
| --- | --- | --- |
| Вариант | Алгоритм генерации ПСП | Параметры |
| 1 | RSA | *p*, *q*, *e*–512-разрядные числа (обосновать выбор)\* |
| 2 | RSA | *p*, *q*, *e*–256-разрядные числа (обосновать выбор)\* |
| 3 | BBS | *n* = 256; *p*, *q*–обосновать выбор по согласованию с преподавателем |
| 4 | BBS | *n* = 512; *p*, *q*–обосновать выбор по согласованию с преподавателем |
| 5 | Линейный конгруэнтный генератор | *а* = 421, *с* = 1663, *n* = 7875 |
| 6 | Линейный конгруэнтный генератор | *а* = 430, *с* = 2531, *n* = 11 979 |

Выбранный вариант 4, соответственно в приложении необходимо реализовать

Алгоритм BBS.

Алгоритм Блюма-Блюма-Шуба (BBS) — это криптографически стойкий генератор псевдослучайной последовательности битов. Он основан на том, что возведение числа в квадрат по модулю большого составного числа даёт значения, которые сложно предсказать без знания исходных параметров. Этот алгоритм считается надёжным, потому что его устойчивость опирается на сложность факторизации больших чисел, аналогично алгоритму RSA.

Сначала выбираются два простых числа, которые при делении на 4 дают в остатке 3. Это числа 19 и 31. Проверим: 19 mod 4 = 3 и 31 mod 4 = 3, значит они подходят. Далее вычисляется произведение этих чисел: 19 \* 31 = 589. Это число и есть модуль, по которому будут проводиться все вычисления — оно обозначается как *n*.

Следующим шагом выбирается начальное значение (seed), которое должно быть взаимно простым с *n*, то есть не иметь с ним общих делителей кроме 1. В коде перед этим есть проверка с помощью функции gcd приведенной в листинге 2.1, которая вычисляет наибольший общий делитель. Если результат не равен 1, значит seed не подходит, и выбрасывается ошибка.

|  |
| --- |
| const gcd = (a: number, b: number): number => {      while (b !== 0) {          const temp = b;          b = a % b;          a = temp;      }      return a;  } |

Листинг 2.1 – Код функции gcd

После этого начинается генерация последовательности. Первое внутреннее значение (обозначается как x₀) вычисляется как квадрат начального значения по модулю n, то есть *x*₀ = seed² mod *n*. Далее каждый следующий элемент получается по формуле *xᵢ* = *xᵢ*₋₁² mod *n*. То есть каждое новое значение — это квадрат предыдущего по модулю *n*. Это значение само по себе не сохраняется в последовательности. Вместо него берётся только его младший бит — это последний бит в двоичном представлении числа, который равен 1, если число нечётное, и 0, если чётное.

Таким образом, вся псевдослучайная последовательность состоит из этих младших битов. Если, например, на одном из шагов получилось число 81, то в двоичной системе это 1010001, и его младший бит — 1. Если получилось число 10 (1010), младший бит — 0. Именно эти биты и записываются в итоговую последовательность, длина которой указывается в параметре sequenceLength.

Пусть seed равен 3. Проверяем: наибольший общий делитель с 589 равен 1, значит seed подходит. Тогда:

*x*₀ = 3² mod 589 = 9

*x*₁ = 9² mod 589 = 81

*x*₂ = 81² mod 589 = 6561 mod 589 = 91

*x*₃ = 91² mod 589 = 8281 mod 589 = 27

И так далее. Из каждого значения берём младший бит:

9 в двоичной: 1001 → бит: 1

81 в двоичной: 1010001 → бит: 1

91: 1011011 → бит: 1

27: 11011 → бит: 1

И получается последовательность из единиц. Если на каком-то шаге получалось чётное число, бит был бы 0.

В результате весь алгоритм BBS по примеру последовательно вычисляет цепочку значений и из каждого извлекает один бит, формируя тем самым псевдослучайную битовую строку. Эта строка может использоваться, например, как ключ для шифрования, в генерации случайных чисел или в протоколах защиты информации.

Результат работы приложения с начальным числом 124 приведен на рисунке 2.1.

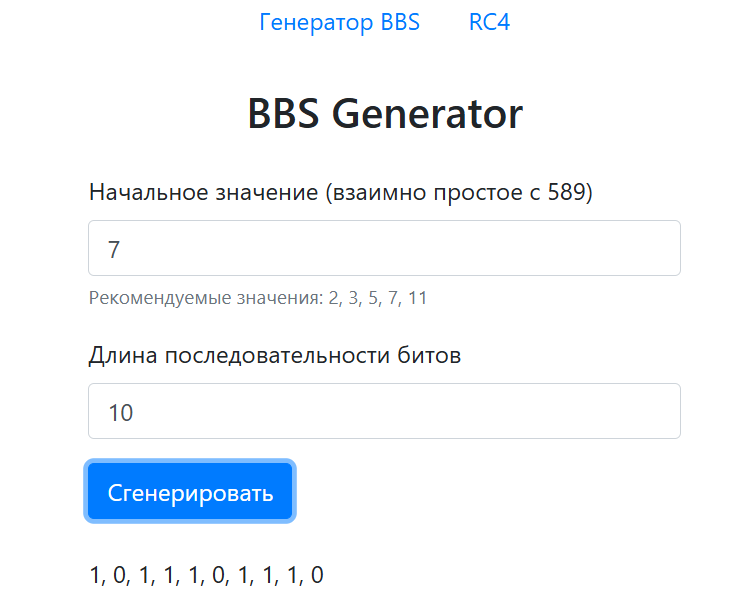


Рисунок 2.1 – Результат работы алгоритма BBS

# **Задание 1**

Далее, нужно было разработать приложение, реализующее алгоритм RC4. Приложение 2 должно реализовывать алгоритм RC4 в соответствии с вариантом из табл. 2.2, а также дополнительно выполнять оценку скорости выполнения операций генерации ПСП.

Таблица 2.1 – Варианты задания

|  |  |  |
| --- | --- | --- |
| Вариант | Алгоритм генерации ПСП | Параметры |
| 1 | 8 | 121, 14, 89, 15 |
| 2 | 8 | 76, 111, 85, 54, 211 |
| 3 | 8 | 43, 45, 100, 21, 1 |
| 4 | 8 | 12, 13, 90, 91, 240 |
| 5 | 8 | 123, 125, 41, 84, 203 |
| 6 | 5 | 1, 11, 21, 31, 41, 51 |
| 7 | 5 | 10, 11, 12, 13, 14, 15 |
| 8 | 5 | 20, 21, 22, 23, 60, 61 |
| 9 | 5 | 61, 60, 23, 22, 21, 20 |
| 10 | 5 | 15, 14, 13, 12, 11, 10 |
| 11 | 8 | 13, 19, 90, 92, 240 |
| 12 | 8 | 122, 125, 48, 84, 201 |
| 13 | 8 | 61, 60, 23, 22, 21, 20 |
| 14 | 8 | 20, 21, 22, 23, 60, 61 |
| 15 | 8 | 1, 11, 21, 31, 41, 51 |

Выбранным вариантом опять будет 4. Функция RC4encrypt принимает исходный текст и шифрует его. Код функции представлен в листинге 2.2.

|  |
| --- |
| export const RC4encrypt = (data: string)  => {      const n: number = 8;      const key: number[] = [12, 13, 90, 91, 240];      const m: number = Math.pow(2, n);      let x: number = 0;      let y: number;      const box: number[] = [...Array(m).keys()];      let startTime: number = performance.now();      for (let i = 0; i < m; i++) {          x = (x + box[i] + key[i % key.length]) % m;          [box[i], box[x]] = [box[x], box[i]];      }      let endTime: number = performance.now();      const generationTime: string = (endTime - startTime).toFixed(4);      x = y = 0;      const out: string[] = []      for (const char of data) {          x = (x + 1) % m;          y = (y + box[x]) % m;          [box[x], box[y]] = [box[y], box[x]];          out.push(String.fromCharCode(char.charCodeAt(0) ^ box[(box[x] + box[y]) % m]));      }      return {result: out.join(''), generationTime};  } |

Листинг 2.2 – Код функции RC4

Результат шифрования текста «EvseenkoViktoriyaPavlovna» представлен на рисунке 2.2.

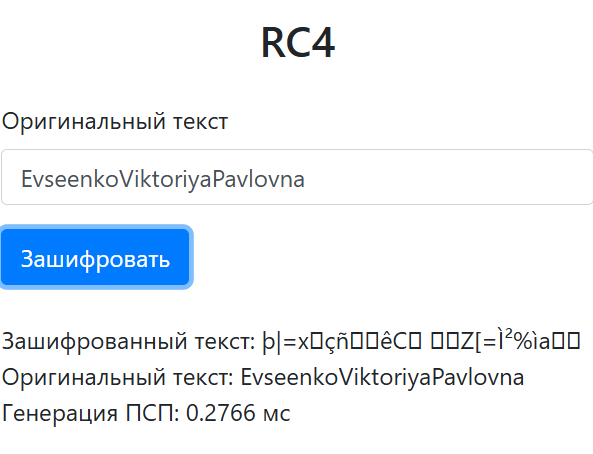


Рисунок 2.2 – Результат алгоритма RC4

# **Оценка времени**

Приложение выполняет оценку скорости генерации ПСП. Результат нескольких попыток генерации ПСП при значении ключа [12, 13, 90, 91, 240] представлен на рисунке 2.3.



Рисунок 2.1 – Оценка времени операций зашифрования/расшифрования

# **Оценка криптостойкости**

Генератор псевдослучайной последовательности Блюма-Блюма-Шуба (BBS) обладает высокой криптостойкостью. Его безопасность основана на математически сложной задаче факторизации больших чисел, аналогичной используемой в алгоритме RSA. Это означает, что при использовании достаточно больших простых чисел (*p* и *q*), восстановить внутреннее состояние генератора или предсказать следующий бит последовательности становится вычислительно неразрешимой задачей. BBS генерирует только по одному биту за итерацию, что делает его относительно медленным, но зато надёжным для криптографических применений, где требуется высокая степень защиты.

Алгоритм RC4, напротив, ориентирован на высокую производительность и простоту реализации. Он был широко распространён и применялся во многих протоколах (например, WEP и SSL/TLS в старых версиях). Однако, несмотря на свою эффективность, RC4 обладает рядом уязвимостей. Основная проблема заключается в слабостях на начальных этапах генерации ключевого потока, из-за чего первые байты шифртекста могут содержа ть статистические зависимости от ключа. Это позволяет проводить атаки, особенно при многократном использовании одного и того же ключа. На сегодняшний день RC4 считается устаревшим и небезопасным для применения в системах, требующих высокой степени защиты.

Таким образом, BBS демонстрирует высокую криптостойкость, но имеет ограничения по скорости, тогда как RC4 обеспечивает высокую производительность, но уступает в надёжности. Выбор между этими алгоритмами зависит от требований к безопасности и производительности в конкретной задаче.

# **Вывод**

В ходе выполнения лабораторной работы были изучены теоретические основы потоковых шифров, рассмотрены алгоритмы их функционирования, а также особенности генерации ключевой информации. Полученные знания были закреплены путём практической реализации генераторов псевдослучайной последовательности и алгоритмов зашифрования/расшифрования данных.

Были разработаны и протестированы программные модули, реализующие генератор BBS и алгоритм RC4. Проведены эксперименты по зашифрованию и расшифрованию информации с использованием этих методов. Осуществлён замер времени генерации ключей и процесса шифрования, что позволило оценить скорость работы реализованных шифров. Также была выполнена базовая оценка криптостойкости реализованных алгоритмов, показавшая, что BBS отличается высокой стойкостью, обусловленной сложностью факторизации, тогда как RC4, несмотря на высокую скорость, требует осторожного применения из-за известных уязвимостей при неправильной инициализации.

Разработанное приложение и методика проведения экспериментов позволяют наглядно исследовать работу потоковых шифров, а также сравнивать их характеристики по скорости и стойкости. Полученные результаты могут быть использованы для дальнейшего анализа эффективности различных методов симметричного шифрования.