Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Исследование потоковых шифров

Студент: Коктыш Е. С.

ФИТ 3 курс 6 группа

Преподаватель: Нистюк О. А.

Минск 2024

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. 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).

Синхронные потоковые шифры (СПШ) — шифры, в которых поток ключей генерируется независимо от открытого текста и шифротекста.

Самосинхронизирующиеся потоковые шифры (асинхронные потоковые шифры (АПШ)) — шифры, в которых ключевой поток создаётся функцией ключа и фиксированного числа знаков шифротекста.

Основной задачей потоковых шифров является выработка некоторой последовательности (гаммы) для зашифрования, т.е. выходная гамма является ключевым потоком (ключом) для сообщения.

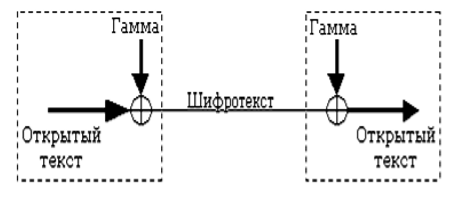


Рисунок 1.1 – Схема потокового шифра

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.

**Генератор ПСП на основе регистров сдвига.** 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).

РСЛОС строятся на основе примитивных порождающих полиномов (многочленов), которые мы подробно анализировали при изучении циклических помехоустойчивых кодов. Если многочлен является неприводимым, то период ПСП при ненулевом начальном условии (ненулевом состоянии) регистра будет максимально возможным: 2L – 1.

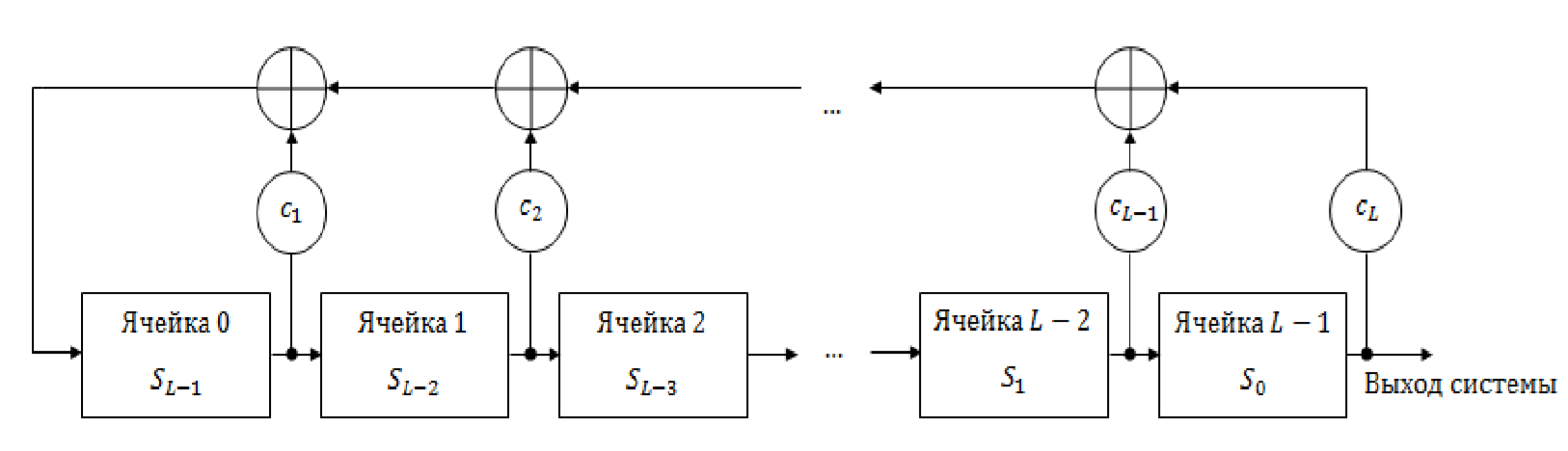


Рисунок 1.2 – Общая схема регистра сдвига с линейной обратной связью

Генератор псевдослучайных чисел на основе **алгоритма RSA** разработан для систем асимметричного зашифрования/расшифрования. Генератор ПСП на основе RSA устроен следующим образом. Последовательность генерируется с использованием соотношения:

xt = (xt-1)е mod n.

Начальными параметрами служат n, большие простые числа p и q (причем n = p\*q), целое число е, взаимно простое с произведением (р – 1)\*( q –1), а также некоторое случайное начальное значение, x0. Выходом генератора является на t-м шаге является младший бит числа xt. Безопасность генератора опирается на сложности взлома алгоритма RSA, т. е. на разложении числа n на простые сомножители.

1. **Практическая часть**

В данной лабораторной работе было необходимо реализовывать генерацию ПСП.

**2.1. Генерация ПСП на основе алгоритма RSA**

Алгоритм RSA разработан для систем асимметричного зашифрования/расшифрования.

Последовательность генерируется с использованием соотношения.



Начальными параметрами служат n, большие простые числа p и q (причем n = pq), целое число е, взаимно простое с произведением (р – 1)(q – 1), а также некоторое случайное начальное значение x0.

Для реализации генерации ПСП на основе алгоритма RSA была написана следующая функция, представленная в листинге 2.1.

|  |
| --- |
| function generatePseudoRandomSequence(p, q, e, x0, t) {      const n = p \* q;      let sequence = [];      let xt = x0;      for (let i = 0; i < t; i++) {          xt = (xt \*\* e) % n;          sequence.push(xt); // Сохраняем значение xt      }      return sequence;  }  app.get('/generate', (req, res) => {      // Получаем параметры из запроса (p, q, e, x0, t)      const { p, q, e, x0, t } = req.query;      // Преобразуем параметры в числа      const pInt = parseInt(p);      const qInt = parseInt(q);      const eInt = parseInt(e);      const x0Int = parseInt(x0);      const tInt = parseInt(t);      const sequence = generatePseudoRandomSequence(pInt, qInt, eInt, x0Int, tInt);      res.send(sequence.join(' '));  }); |

Листинг 2.1 – Функция генерации ПСП

Результат работы алгоритма, создающий псевдослучайные числа, представлен на рисунке 2.1.



Рисунок 2.1 – Результат работы приложения

# 2.1 Шифрование алгоритмом RC4

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.

Нужно было реализовывать алгоритм RC4 в соответствии с вариантом, а также дополнительно выполнять оценку скорости выполнения операций генерации ПСП.

Исходный текс представлен на рисунке 2.2.

|  |
| --- |
|  |

Рисунок 2.2 – Исходный текст

Функция для шифрования принимает исходный текст и шифрует его. Код функции представлен в листинге 2.2.

|  |
| --- |
| function initialize\_S\_block(n, key) {              let S = [];              for (let i = 0; i < Math.pow(2, n); i++) {                  S[i] = i;              }              let j = 0;              for (let i = 0; i < Math.pow(2, n); i++) {                  j = (j + S[i] + key[i % key.length]) % Math.pow(2, n);                  // Swap S[i] and S[j]                  let temp = S[i];                  S[i] = S[j];                  S[j] = temp;              }              return S;          }          function generate\_keystream(S, text) {              let keystream = [];              let i = 0;              let j = 0;              for (let k = 0; k < text.length; k++) {                  i = (i + 1) % S.length;                  j = (j + S[i]) % S.length;                  // Swap S[i] and S[j]                  let temp = S[i];                  S[i] = S[j];                  S[j] = temp;                  let t = (S[i] + S[j]) % S.length;                  let key = S[t];                  keystream.push(key);              }              return keystream;          }          function processText(plaintext, key) {              let S = initialize\_S\_block(8, key);              let keystream = generate\_keystream(S, plaintext);              let result = [];              for (let i = 0; i < plaintext.length; i++) {                  let encryptedChar = plaintext[i] ^ keystream[i];                  result.push(String.fromCharCode(encryptedChar));              }              return result.join('');          } |

Листинг 2.2 – Код функции, реализующей RC4

Вывод работы функции зашифрования и расшифрования и время выполнения представлен на рисунке 2.3.

|  |
| --- |
|  |

Рисунок 2.3 – Вывод затраченного времени и шифрования и расшифрования текста

# Вывод

В данный лабораторной работе были изучены и приобретены практические навыки разработки и использования потоковых шифров.