

Министерство образования и науки Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Сыктывкарский государственный университет имени Питирима Сорокина»  
Институт точных наук и информационных технологий  
Кафедра информационной безопасности

Допустить к защите  
Зав. кафедрой, к.ф.-м.н., доцент  
\_\_\_\_\_ Л.С. Носов  
«\_\_\_» июня 2015 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

Разработка криптографической библиотеки поточного шифрования на основе  
проекта eStream  
направление 090900.62 – «Информационная безопасность»

Научный руководитель:

к.ф.-м.н., доцент

\_\_\_\_\_ Ю.В. Гольчевский  
«\_\_\_» июня 2015 г.

Исполнитель:

Студент группы 143

\_\_\_\_\_ Р.А. Гашин  
«\_\_\_» июня 2015 г.

Сыктывкар 2015

## СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ.....	4
ВВЕДЕНИЕ.....	5
ГЛАВА 1 ПРОЕКТ eStream.....	7
1.1 Проект «NESSIE».....	7
1.2 Проект eStream .....	8
1.3 Алгоритм шифрования Salsa.....	10
1.4 Алгоритм шифрования Rabbit .....	13
1.5 Алгоритм шифрования HC128 .....	16
1.6 Алгоритм шифрования Sosemanuk.....	17
1.7 Алгоритм шифрования Grain.....	20
1.8 Алгоритм шифрования Trivium.....	21
1.9 Алгоритм шифрования Mickey.....	23
1.10 Общая схема функции шифрования/расшифровывания .....	25
1.11 Алгоритмы проекта eStream в действующих проектах .....	26
1.12 Безопасность алгоритмов проекта eStream .....	27
ГЛАВА 2 БИБЛИОТЕКА estream.h .....	29
2.1 Структура библиотеки estream.h .....	29
2.2 Механизмы модернизации .....	33
2.2.1 Использование макросов.....	33
2.2.2 Использование битовых операций.....	34
2.3 Модернизация алгоритмов.....	35
2.4 Сравнение библиотеки estream.h и алгоритмов разработчиков.....	38
ГЛАВА 3 ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ БИБЛИОТЕКИ estream.h.....	43
3.1 Рекомендации разработчикам.....	43
3.2 Проверка корректности библиотеки estream.h.....	45
3.3 Программа для шифрования файлов .....	47
3.4 Клиент-серверное приложение.....	49
ЗАКЛЮЧЕНИЕ .....	53

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ И ИСТОЧНИКОВ .....	55
ПРИЛОЖЕНИЕ 1 .....	58
ПРИЛОЖЕНИЕ 2 .....	59
ПРИЛОЖЕНИЕ 3 .....	60
ПРИЛОЖЕНИЕ 4 .....	61

## ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

**Вектор** – одномерный массив данных [1].

**Вектор инициализации (инициализационный вектор)** – вектор, определенный как начальная точка работы функции [2].

**Ключевая последовательность** – набор данных, генерируемый криптографическим алгоритмом, который используется для шифрования/расшифровывания информации [3].

**Потоковый шифр** – шифр, в котором каждый символ открытого текста преобразуется в символ шифрованного текста не только в зависимости от используемого секретного ключа, но и от его расположения в потоке открытого текста [3].

**Секретный ключ** – секретная информация, используемая криптографическим алгоритмом при шифровании/расшифровывании сообщений [1].

**Тестовый вектор** – вектор, подаваемый на вход алгоритма, при котором результат вычислений алгоритма сравнивается с эталонным значением [3].

**Хэш-функция** – функция, отображающая строку бит произвольной длины в строку бит фиксированной длины и удовлетворяющая следующим свойствам:

1. по данному значению функции сложно вычислить исходные данные, отображаемые в это значение;
2. для заданных исходных данных сложно вычислить другие исходные данные, отображаемые в то же значение функции;
3. сложно вычислить какую-либо пару исходных данных, отображаемых в одно и то же значение [2].

## ВВЕДЕНИЕ

Стремительное развитие информационных технологий привело к созданию глобальных компьютерных сетей общего пользования, доступ к которым открыт для всех пользователей по всему миру. По этим вычислительным сетям ежедневно передаются огромные объемы информации, в том числе и ограниченного распространения. Такая информация нуждается в обеспечении конфиденциальности при передаче. Самым надежным способом защиты по праву считается использование криптографических алгоритмов, основанных на математических преобразованиях.

Для защиты конфиденциальной информации, передающейся по вычислительным сетям, самыми эффективными признаны потоковые криптографические алгоритмы. Достоинством потоковых шифров является высокая скорость шифрования, и возможность работы с неограниченными потоками данных в режиме реального времени.

Передовые разработки в области потоковых шифров ведутся в России, США, Европе и Японии. Основой потоковой криптографии в США является алгоритм AES (Advanced Encryption Standard), работающий в режиме «счетчика». Эффективных атак на алгоритм, которые существенно бы понизили его стойкость, на сегодняшний день не выявлено, в результате чего AES признан одним из самых надежных алгоритмов в мире [1, 4].

На территории стран Европейского Союза самым распространенным потоковым шифром считается алгоритм A5, который используется в европейской системе мобильной цифровой связи GSM (Group Special Mobile). Со времени публикации официальной документации (1994 год) в алгоритме были обнаружены некоторые недостатки, понижающие его надежность [5].

В феврале 2003 года в Японии завершился крупный проект по разработке криптографических алгоритмов CRYPTREC. Его победителями стали 3 потоковых криптографических шифра MUGI, MULTI-S01 и 128-bit RC, предназначенные для использования в системе электронного

правительства Японии. Недостатков и эффективных атак на алгоритмы не было выявлено, что позволяет говорить об их надежности и стойкости [6].

Из открытых источников известно, что на территории Российской Федерации в качестве алгоритма поточного шифрования информации используется шифр на основе ГОСТ Р 28147-89, работающий в режиме гаммирования [7].

Таким образом, обеспечение безопасности информации, передаваемой по вычислительным сетям, с помощью криптографических алгоритмов является важным звеном в обеспечении информационной безопасности.

Объектом исследования данной выпускной квалификационной работы является обеспечение безопасности передаваемой информации по компьютерным сетям с помощью потоковых криптографических алгоритмов, а предметом исследования – потоковая криптографическая библиотека, разработанная на основе проекта eStream.

Цель выпускной квалификационной работы: разработка потоковой криптографической библиотеки на основе проекта eStream.

Для достижения цели требуется решить следующие задачи:

1. изучение алгоритмов проекта eStream;
2. поиск и реализация возможностей для оптимизации алгоритмов проекта eStream с целью повышения их производительности;
3. сравнение оптимизированных реализаций алгоритмов с существующими разработками;
4. разработка потоковой криптографической библиотеки `estream.h`;
5. разработка программных приложений для демонстрации возможностей разработанной библиотеки `estream.h`.

## ГЛАВА 1 ПРОЕКТ eStream

### 1.1 Проект «NESSIE»

Проект «NESSIE» (англ. New European Schemes for Signatures, Integrity and Encryptions – новые европейские алгоритмы для электронной подписи, целостности и шифрования) – это научно-исследовательский проект для определения новых шифровальных алгоритмов, на базе которых должны быть созданы новые криптографические стандарты Европы. Конкурс проходил с января 2000 года по февраль 2003 года. Участниками проекта могли стать любые организации и частные лица, приславшие свой криптографический алгоритм для анализа [8].

Проект насчитывал 5 категорий шифров:

1. блочные алгоритмы шифрования;
2. ассиметричные алгоритмы шифрования;
3. алгоритмы хэширования;
4. алгоритмы электронной подписи;
5. алгоритмы потокового шифрования.

Отбор оптимальных алгоритмов, которые претендовали на победу, проходил в 3 этапа:

1. первичное изучение алгоритмов;
2. криптоаналитические исследования алгоритмов;
3. сравнение алгоритмов по нескольким критериям: быстродействие, минимальные требования к вычислительным ресурсам, производительность и другие.

Всего на конкурс было заявлено 42 криптографических шифра, из которых 18 выбыло после 1-го этапа. В результате работы комиссии на 2-м и 3-м этапах победителями были объявлены 12 алгоритмов.

Победителями в соответствующих категориях стали:

1. блочные шифры: MISTY 1 (Mitsubishi Electric Corp., Япония), Camellia (Nippon Telegraph and Telephone Corp. и Mitsubishi Electric Corp.,

Япония), SHACAL-2 (Gemplus, Франция), AES (В. Рэймен и Й. Даймен, Бельгия);

2. ассиметричные шифры: ACE Encrypt (научно-исследовательская лаборатория IBM, Швейцария), PSEC-KEM (Nippon Telegraph and Telephone Corp., Япония);

3. хэш-функции: Two-Track-MAC (Левенский католический университет, Бельгия и Debis AG, Германия), UMAC (Intel Corp., университет Невады, научно-исследовательская лаборатория IBM, США, Technion, Израиль и университет Калифорнии в Дэвисе, США), Whirlpool (Scopus Tecnologia S.A., Бразилия и Левенский католический университет, Бельгия);

4. алгоритмы для электронной подписи: ECD SA (Certicom Corp., США и Certicom Corp., Канада), RSA-PSS (лаборатория RSA, США), SFLASH (Schlumberger, Франция) [9].

В категории потоковое шифрование было представлено 6 алгоритмов, которые не прошли все испытания и были признаны непригодными для использования. Данный факт послужил началом для старта проекта по выявлению новых потоковых криптографических алгоритмов eStream.

## **1.2 Проект eStream**

Проект eStream (англ. European Stream – Европейский поток) – научно-исследовательский проект по выявлению новых потоковых криптографических алгоритмов. Был начат в феврале 2004 года, после взлома всех 6-ти шифров проекта «NESSIE». Официально проект был завершен в мае 2008 года. Последняя публикация документации шифров-победителей датируется январем 2012 года. После этого изменения в криптографические алгоритмы не вносились.

В ноябре 2004 года началась приемная компания претендентов на участие в конкурсе. К участникам предъявлялись следующие основные требования: длина ключа максимум 128 бит; алгоритмы должны были



работать быстрее, чем американский стандарт шифрования AES-128 (Advanced Encryption Standard) в режиме счетчика.

Все конкурсанты разделялись на 2 категории: «Программно-ориентированные алгоритмы» и «аппаратно-ориентированные алгоритмы».

В результате предварительного отбора, для участия в конкурсе были отобраны следующие участники: Frogbit, Fubuki, MAG, Mir-1, SSS, TRBDK3 YAEA, Phelix, Py, ABC, Achterbahn, DICING, Hermes8, NLS, Polar Bear, Pomaranch, SFINKS, TSC-3, VEST, WG, Yamb, ZK-Crypt, CryptMP, DECIM, Dragon, Edon80, LEX, MOSQUITO, Grain, HC128, MICKEY, Rabbit, Salsa20, SOSEMANUK, Trivium, F-FC-SR [3].

На 1-м этапе проекта (29 апреля 2005 – 27 марта 2006) все участники подверглись общему анализу. Шифры были исследованы на надежность, производительность, корректность работы, простоту и гибкость реализации. Была изучена подробная документация по каждому алгоритму.

2-й этап конкурса (2 августа 2006 – апрель 2007) включал в себя анализ алгоритмов производительности на различных платформах, операционных системах и архитектурах процессоров.

3-й этап проекта (апреля 2007 – май 2008) был ознаменован выпуском книги «New stream cipher designs» (Новые конструкции поточного шифра), в которую вошли история разработки и описание всех криптографических алгоритмов, вышедших в финальную часть.

Для выбора алгоритмов-победителей была сформирована комиссия, в которую вошли разработчики и исследователи крупнейших компаний и университетов Европы: С.Бэббидж (Vodafone, Великобритания), К. Де Каннье, Б. Пренел и У Хунцзюнь (Левенский католический университет, Бельгия), А. Канто (INRIA, Франция), К. Сид (Лондонский университет, Великобритания), А. Жильбер и М. Робшоу (France Telecom R&D, Франция), Т. Юханссон (Лундский университет, Швеция), К. Пар (Рурский университет, Германия), М. Паркер (Бергенский университет, Норвегия), В. Реймен (Технический университет Граца, Австрия) [3].

Комиссия изучила всю имеющуюся документацию по алгоритмам, выслушала презентации авторов, и 15 апреля 2008 года, на основании тщательного анализа, были объявлены алгоритмы-победители проекта eStream в обеих категориях.

В категории «Программно-ориентированные алгоритмы» победителями стали: HC128, Rabbit, Salsa, Sosemanuk.

В категории «Аппаратно-ориентированные алгоритмы» победителями стали: Grain, MICKEY, Trivium, F-FCSR.

В сентябре 2008 года криптографический шифр F-FCSR был исключен из финальной публикации в виду криптографической слабости, обнаруженной после завершения проекта.

Следующие разделы будут посвящены алгоритмам шифрования, которые стали победителями проекта eStream.

### **1.3 Алгоритм шифрования Salsa**

Salsa – алгоритм поточного шифрования информации, разработанный Даниэлем Бернштейном. Алгоритм был впервые представлен на проекте eStream, где стал победителем в категории «Программно-ориентированные алгоритмы» [10].

Шифр использует хэш-функцию с 20-ю циклами, в которых поочередно применяются операции побитового сложения и циклического сдвига 32-битных машинных слов, которые обеспечивают криптостойкость алгоритма.

Основные характеристики алгоритма:

1. длина ключа не должна превышать 256 бит. В зависимости от длины ключа (до 128 бит или 256 бит) алгоритм будет видоизменяться;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. в функции расширения ключа (в зависимости от длины ключа) используется константа: «expand 16-byte key» или «expand 32-byte key»;

5. на вход алгоритма подается уникальный 64-битный вектор инициализации.

Согласно официальной документации, в алгоритме Salsa определены несколько стандартных функций, определяющих ключевые преобразования: quarterround, rowround, columnround, doubleround, хэш-функция salsa20 и расширение ключа [11].

Базовые операции представляют собой следующие преобразования.

Функция quarterround осуществляет преобразование над 4-мя 32-битными словами, при этом для каждого слова складываются 2 предыдущих. Затем полученная сумма сдвигается на определенное количество бит и полученный результат побитового суммируется с исходным словом. Математически функция quarterround(y) выглядит следующим образом:

$$z_1 = y_1 \oplus ((y_0 + y_3) \lll 7);$$

$$z_2 = y_2 \oplus ((z_1 + y_0) \lll 9);$$

$$z_3 = y_3 \oplus ((z_2 + z_1) \lll 13);$$

$$z_0 = y_0 \oplus ((z_3 + z_2) \lll 18),$$

где  $z_i$  и  $y_i$  – это 32-битные слова.

В функцию rowround передается 16 слов, которые представляются в виде матрицы 4x4. Затем каждый ряд этой матрицы передается в функцию quarterround(y). Слова из строки берутся по порядку, начиная с  $i$ -го для  $i$ -ой строки, где  $i = \{0, 1, 2, 3\}$ . Математически функция rowround(y) может быть представлена:

$$(z_0, z_1, z_2, z_3) = \text{quarterround}(y_0, y_1, y_2, y_3);$$

$$(z_5, z_6, z_7, z_4) = \text{quarterround}(y_5, y_6, y_7, y_4);$$

$$(z_{10}, z_{11}, z_8, z_9) = \text{quarterround}(y_{10}, y_{11}, y_8, y_9);$$

$$(z_{15}, z_{12}, z_{13}, z_{14}) = \text{quarterround}(y_{15}, y_{12}, y_{13}, y_{14}),$$

где  $y = (y_0, \dots, y_{15})$  – 16-битные слова,  $z = (z_0, \dots, z_{15})$  – 16-битные слова, получаемые на выходе функции rowround(y).

На вход функции columnround(y) подаются 16 32-битных слов, которые представляются в виде матрицы 4x4, аналогично функции rowround. Затем

каждый столбец этой матрицы подается в функцию  $\text{quarterround}(y)$ . Слова из столбца берутся по порядку, начиная с  $j$ -го для  $j$ -го столбца, где  $j = \{0, 1, 2, 3\}$ . Функцию  $\text{columnround}(y)$  можно представить следующим образом:

$$\begin{aligned}(y_0, y_4, y_8, y_{12}) &= \text{quarterround}(x_0, x_4, x_8, x_{12}); \\(y_5, y_9, y_{13}, y_1) &= \text{quarterround}(x_5, x_9, x_{13}, x_1); \\(y_{10}, y_{14}, y_2, y_6) &= \text{quarterround}(x_{10}, x_{14}, x_2, x_6); \\(y_{15}, y_3, y_7, y_{11}) &= \text{quarterround}(x_{15}, x_3, x_7, x_{11}),\end{aligned}$$

где  $x = (x_1, \dots, x_{15})$  – 16-битные слова,  $y = (y_0, \dots, y_{15})$  – 16-битные слова, получаемые на выходе функции  $\text{columnround}(y)$ .

Функция  $\text{doubleround}(y)$  является последовательным выполнением функций  $\text{columnround}(y)$  и  $\text{rowround}(y)$ . В общем виде данное преобразование можно записать так:

$$\text{doubleround}(x) = \text{rowround}(\text{columnround}(y)),$$

где  $x$  и  $y$  – это последовательность из 16-ти 32-битных слов.

Хэш-функция предназначена для генерирования уникальной 64-битной ключевой последовательности.

На вход функции подается 64-байтная последовательность. Затем каждые 4 байта подвергаются операции приведения к обратному порядку байт ( $\text{littleendian}$ ). Математически эта операция выглядит так:

$$\text{littleendian}(b) = b_0 + 2^8 \cdot b_1 + 2^{16} \cdot b_2 + 2^{24} \cdot b_3,$$

где  $b = (b_0, b_1, b_2, b_3)$  – 32-битовое слово, то есть упорядоченная последовательность из 4 байт  $(b_0, b_1, b_2, b_3)$ .

На выходе преобразования  $\text{littleendian}$  получается последовательность, состоящая из 16-ти 32-битных слов, которая 20 раз подается на вход функции  $\text{doubleround}$ .

На выходе хэш-функции алгоритма Salsa получается 512-ти битовая ключевая последовательность, которая используется при шифровании/расшифровывании потоков данных.

На вход функции расширения ключа подается секретный ключ (до 256 бит) и 128-битный вектор инициализации.

В функции используются 2 константы: «expand 16-byte key» (если длина ключа до 128 бит) или «expand 32-byte key» (если длина ключа до 256 бит).

Расширение ключа происходит за счет последовательной записи секретного ключа, вектора инициализации и константы в 512-ти битовую последовательность, которая передается в хэш-функцию для генерации ключевой последовательности. Данная 512-ти битовая последовательность схематично выглядит следующим образом:

для ключа до 128 бит:  $\text{key\_exp} = (\sigma_0, k, \sigma_1, n, \sigma_2, k, \sigma_3)$ ;

для ключа до 256 бит:  $\text{key\_exp} = (\sigma_0, k_0, \sigma_1, n, \sigma_2, k_1, \sigma_3)$ ,

где  $k$  – секретный ключ, либо его  $i$ -часть,  $\sigma_i$  – константа (определяется в зависимости от длины ключа),  $n$  – вектор инициализации.

#### 1.4 Алгоритм шифрования Rabbit

Rabbit – алгоритм поточного шифрования информации, впервые был представлен широкой публике в феврале 2003 года. В мае 2005 года Rabbit был заявлен для участия в проекте eStream, где стал победителем в категории «Программно-ориентированные алгоритмы» [10]. Разработчиками алгоритма являются М. Боесгард, М. Вестэрагер, Т. Кристиансен и Э. Зеннер.

Основные характеристики алгоритма:

1. длина секретного ключа до 128 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход подается уникальный 64-битный вектор инициализации.

Согласно официальной документации в алгоритме Rabbit определены несколько стандартных функций, определяющих ключевые преобразования:  $\text{key\_setup}$ ,  $\text{iv\_setup}$ ,  $\text{next\_state}$  [12].

Рассмотрим базовые преобразования. Функция  $\text{key\_setup}$  предназначена для установления ключа в память компьютера по следующему алгоритму. Секретный ключ разбивается на 8 подключей  $k_0, k_1, \dots, k_7$ , на основании

которых генерируются 8 переменных состояний  $x_{j,0}$  и 8 счетчиков состояний  $c_{j,0}$ , по следующей схеме:

$$x_{j,0} = k_{(j+1 \bmod 8)} \parallel k_j, \text{ где } j \text{ четно};$$

$$x_{j,0} = k_{(j+5 \bmod 8)} \parallel k_{(j+4 \bmod 8)}, \text{ где } j \text{ нечетно};$$

$$c_{j,0} = k_{(j+4 \bmod 8)} \parallel k_{(j+5 \bmod 8)}, \text{ где } j \text{ четно};$$

$$c_{j,0} = k_j \parallel k_{(j+1 \bmod 8)}, \text{ где } j \text{ нечетно},$$

где операция  $\parallel$  означает конкатенацию 2-х подключей,  $j = \{0, 1, \dots, 7\}$ .

Вся схема проходит подобные преобразования 4 раза с помощью функции следующего шага (next\_state).

После чего все счетчики состояний переинициализируются согласно следующей формуле:

$$c_{j,4} = c_{j,4} \oplus x_{(j+4 \bmod 8),4}$$

После функции key\_setup следует функция iv\_setup, которая предназначена для установления вектора инициализации в память компьютера по следующему алгоритму:

$$c_{0,4} = c_{0,4} \oplus IV^{[31..0]};$$

$$c_{2,4} = c_{2,4} \oplus IV^{[63..32]};$$

$$c_{4,4} = c_{4,4} \oplus IV^{[31..0]};$$

$$c_{6,4} = c_{6,4} \oplus IV^{[63..32]};$$

$$c_{1,4} = c_{1,4} \oplus (IV^{[63..48]} \times IV^{[31..16]});$$

$$c_{3,4} = c_{3,4} \oplus (IV^{[47..32]} \times IV^{[15..0]});$$

$$c_{5,4} = c_{5,4} \oplus (IV^{[63..48]} \times IV^{[31..16]});$$

$$c_{7,4} = c_{7,4} \oplus (IV^{[47..32]} \times IV^{[15..0]}),$$

где  $IV^{[31..0]}$  – это последовательность бит (с 0 по 31) вектора инициализации.

Вся схема проходит подобные преобразования 4 раза.

Функция next\_state содержит следующие уравнения:

$$x_{0,i+1} = g_{0,i} + (g_{7,i} \lll 16) + (g_{6,i} \lll 16);$$

$$x_{1,i+1} = g_{1,i} + (g_{7,i} \lll 16) + g_{7,i};$$

$$x_{2,i+1} = g_{2,i} + (g_{1,i} \lll 16) + (g_{0,i} \lll 16);$$

$$x_{3,i+1} = g_{3,i} + (g_{7,i} \lll 16) + g_{1,i};$$

$$x_{4,i+1} = g_{4,i} + (g_{3,i} \lll 16) + (g_{2,i} \lll 16);$$

$$x_{5,i+1} = g_{5,i} + (g_{7,i} \lll 16) + g_{3,i};$$

$$x_{6,i+1} = g_{6,i} + (g_{5,i} \lll 16) + (g_{4,i} \lll 16);$$

$$x_{7,i+1} = g_{7,i} + (g_{7,i} \lll 16) + g_{5,i},$$

где  $x_{i,j}$  – переменные состояния, функция  $g_{j,i} = ((x_{i,j} + c_{j,i+1})^2 \oplus ((x_{i,j} + c_{j,i+1})^2 \gg 32)) \bmod 2^{32}$ .

Система счетчиков так же подвержена изменению согласно функции следующего шага:

$$c_{0,i+1} = c_{0,i+1} + a_0 + \varphi_{7,i} \bmod 2^{32};$$

$$c_{1,i+1} = c_{1,i+1} + a_1 + \varphi_{0,i+1} \bmod 2^{32};$$

$$c_{2,i+1} = c_{2,i+1} + a_2 + \varphi_{1,i+1} \bmod 2^{32};$$

$$c_{3,i+1} = c_{3,i+1} + a_3 + \varphi_{2,i+1} \bmod 2^{32};$$

$$c_{4,i+1} = c_{4,i+1} + a_4 + \varphi_{3,i+1} \bmod 2^{32};$$

$$c_{5,i+1} = c_{5,i+1} + a_5 + \varphi_{4,i+1} \bmod 2^{32};$$

$$c_{6,i+1} = c_{6,i+1} + a_6 + \varphi_{5,i+1} \bmod 2^{32};$$

$$c_{7,i+1} = c_{7,i+1} + a_7 + \varphi_{6,i+1} \bmod 2^{32},$$

где функция  $\varphi_{j,i+1}$  определяется следующим образом:

$$\varphi_{j,i+1} = 1, \text{ если } c_{0,i} + a_0 + \varphi_{7,i} \geq 2^{32};$$

$$\varphi_{j,i+1} = 1, \text{ если } c_{j,i} + a_j + \varphi_{j-1,i+1} \geq 2^{32};$$

$$\varphi_{j,i+1} = 0, \text{ если другое.}$$

Константы  $a_i$  определены согласно официальной документации:

$$a_0 = 0x4D34D34D; \quad a_1 = 0xD34D34D3;$$

$$a_2 = 0x34D34D34; \quad a_3 = 0x4D34D34D;$$

$$a_4 = 0xD34D34D3; \quad a_5 = 0x34D34D34;$$

$$a_6 = 0x4D34D34D; \quad a_7 = 0xD34D34D3.$$

В результате работы функций `next_state` генерируется уникальная ключевая последовательность по следующей схеме:

$$s_i^{[15..0]} = x_{0,i}^{[15..0]} \oplus x_{5,i}^{[31..16]}, \quad s_i^{[31..16]} = x_{0,i}^{[31..16]} \oplus x_{3,i}^{[15..0]},$$

$$s_i^{[47..32]} = x_{2,i}^{[15..0]} \oplus x_{7,i}^{[31..16]}, \quad s_i^{[63..48]} = x_{2,i}^{[31..16]} \oplus x_{5,i}^{[15..0]},$$

$$s_i^{[15..0]} = x_{4,i}^{[15..0]} \oplus x_{1,i}^{[31..16]}, \quad s_i^{[95..80]} = x_{4,i}^{[31..16]} \oplus x_{7,i}^{[15..0]},$$

$$s_i^{[15..0]} = x_{6,i}^{[15..0]} \oplus x_{3,i}^{[31..16]}, \quad s_i^{[127..112]} = x_{6,i}^{[31..16]} \oplus x_{1,i}^{[15..0]},$$

где  $s_i$  – 16 бит ключевой последовательности.

Ключевая последовательность используется для операции побитового сложения по модулю 2 с каждым байтом входного потока. Вследствие чего на выходе получается зашифрованный поток данных.

Функция расшифровывания аналогична функции шифрования.

## 1.5 Алгоритм шифрования HC128

HC – алгоритм поточного шифрования информации. 128-битный вариант реализации алгоритма был заявлен для участия в проекте eStream, где стал победителем в категории «Программно-ориентированные алгоритмы» [10].

Основные характеристики алгоритма:

1. длина ключа не должна превышать 128 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход подается уникальный 128-битный вектор инициализации.

Согласно официальной документации в алгоритме HC128 определены несколько стандартных функций, определяющих базовые преобразования: функция инициализации и функция генерирования ключевой последовательности [13].

В алгоритме HC128 используется 6 стандартных преобразований:  $f_1$ ,  $f_2$ ,  $g_1$ ,  $g_2$ ,  $h_1$ ,  $h_2$ . На математическом языке данные преобразования можно представить так:

$$f_1(x) = (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3);$$

$$f_2(x) = (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10);$$

$$g_1(x, y, z) = ((x \ggg 10) \oplus (z \ggg 23)) + (y \ggg 8);$$

$$g_2(x, y, z) = ((x \lll 10) \oplus (z \lll 23)) + (y \lll 8);$$

$$h_1(x) = Q[x_0] + Q[256+x_2];$$

$$h_2(x) = P[x_0] + P[256+x_2],$$



где  $x, y, z$  – 32-битные машинные слова,  $P$  и  $Q$  – массивы 32-битных слов.

Функция инициализации предназначена для записи и расширения секретного ключа и вектора инициализации в массивы  $P$  и  $Q$  соответственно. Схематично данный процесс можно представить так:

$$W_i = K_i, \text{ при } 0 \leq i \leq 7;$$

$$W_i = IV_i, \text{ при } 8 \leq i \leq 15;$$

$$W_i = f_2(W_{i-2}) + W_{i-7} + f_1(W_{i-15}) + W_{i-16} + i, \text{ при } 16 \leq i \leq 1279;$$

$$P[i] = W_{i+256}, Q[i] = W_{i+768}, \text{ при } 0 \leq i \leq 511;$$

Система начинает процесс шифрования 1024 раза без генерации ключевой последовательности:

$$P[i] = (P[i] + g_1(P[i-3], P[i-10], P[i-511]) \oplus h_1(P[i-12])), \text{ при } 0 \leq i \leq 511;$$

$$Q[i] = (Q[i] + g_1(Q[i-3], Q[i-10], Q[i-511]) \oplus h_1(Q[i-12])) \text{ при } 0 \leq i \leq 511,$$

где операция «-» – операция вычитания по модулю 512.

Функцию генерирования ключевой последовательности можно представить так:

$$\text{при } (i \bmod 1024) < 512$$

$$P[i] = (P[i] + g_1(P[i-3], P[i-10], P[i-511]));$$

$$s = h_1(P[i-12]) \oplus P[i].$$

$$\text{при } (i \bmod 1024) \geq 512$$

$$Q[i] = (Q[i] + g_1(Q[i-3], Q[i-10], Q[i-511]));$$

$$s = h_2(Q[i-12]) \oplus Q[i],$$

где  $s_i$  – 32-битный элемент ключевой последовательности,  $i$  – длина входного потока данных.

## 1.6 Алгоритм шифрования Sosemanuk

Sosemanuk – алгоритм поточного шифрования информации. Был заявлен для участия в проекте eStream, где стал победителем в категории «Программно-ориентированные алгоритмы» [10].

Разработчиками алгоритма являются К. Бербайн, О. Биллет, А. Кантейт, Н. Кортойс, Х. Гилберт, Л. Губин, А. Гойджет, Л. Гранболан, К. Люрадойкс, М. Миниер, Т. Порнин и Х. Сиберт.

Основные характеристики алгоритма:

1. длина ключа не должна превышать 256 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход подается уникальный 128-битный вектор инициализации.

Согласно официальной документации в алгоритме Sosemanuk определены несколько стандартных функций, определяющих ключевые преобразования: функции расширения ключа и вектора инициализации, функция генерирования ключевой последовательности [14].

Рассмотрим основные базовые преобразования. Функция расширения ключа основана на разбиении секретного ключа на 8 подключей по 32 бита каждый. Затем все подключи побитового складываются по модулю 2 и полученный результат циклически сдвигается на 11 бит. Затем массив подключей подается на вход 1 раунду блочного алгоритма Serpent. Алгоритм расширения ключа насчитывает 25 подобных преобразований, на выходе которых генерируется 100 32-битных слов. Схема расширения ключа:

$$w_0 = K_0, \dots, K_{31};$$

$$w_1 = K_{32}, \dots, K_{63};$$

.

.

$$w_7 = K_{224}, \dots, K_{255};$$

$$tt = w_i \oplus w_{i+3} \oplus w_{i+5} \oplus w_{i+7} \text{ при } i \in \{0, \dots, 7\};$$

$$w_i = tt \lll 11;$$

$$\text{serpent}(w),$$

где  $w_i$  –  $i$ -й 32-х битный подключ,  $K_i$  –  $i$ -й бит ключа, serpent – расширение ключа блочного алгоритма Serpent,  $tt$  – переменная для промежуточных вычислений.

Функция расширения вектора инициализации основана на использовании блочного шифра Serpent. Вектор инициализации разбивается на 4 32-х битных вектора, которые подаются на вход 1-му раунду шифра Serpent. Вся схема проходит 24 итерации, что позволяет достичь криптостойкости алгоритма. Расширение вектора инициализации можно представить так:

$$\begin{aligned} r_0 &= IV_{0, \dots, IV_{31}}; \\ r_1 &= IV_{32, \dots, IV_{63}}; \\ r_2 &= IV_{64, \dots, IV_{95}}; \\ r_3 &= IV_{96, \dots, IV_{127}}; \\ \text{serpent24}(r), \end{aligned}$$

где  $r_i$  –  $i$ -й 32-х битный вектор,  $IV_i$  –  $i$ -й бит вектора инициализации, serpent24 – 24 раунда блочного алгоритма Serpent.

Функция генерации ключевой последовательности осуществляет преобразования над массивами байт полученных при работе функций расширения ключа и вектора инициализации. Функцию генерации ключевой последовательности на псевдокоде можно записать следующим образом:

```
for  $i = 0$  to 4
     $tt = \text{XMUX}(r1, s_j, s_t)$ ;
     $or1 = r1$ ;
     $r1 = r2 + tt$ ;
     $tt = 0x54655307 \cdot r1$ ;
     $r2 = tt \lll 7$ ;
     $v_i = s_t$ ;
     $u_i = (s_j + r1) \oplus r2$ ;
end for,

 $z_0 = (u_2 \oplus v_0); z_1 = (u_3 \oplus v_1);$ 
 $z_2 = (u_1 \oplus v_2); z_3 = (u_4 \oplus v_3),$ 
```

где  $s$  – массив битов, полученный путем расширения секретного ключа и вектора инициализации,  $z$  – поток ключевой последовательности,  $u_i, v_i, r1, r2, tt, or1$  – переменные для промежуточных вычислений,  $i$  и  $j$  – индексы,

генерируемые в зависимости от раунда алгоритма, функция  $\text{XMUX}(rI, s_j, s_t)$  возвращает  $s_j$ , если  $rI = 0$ , или  $s_t \oplus s_j$ , если  $rI = 1$ .

### 1.7 Алгоритм шифрования Grain

Grain – алгоритм поточного шифрования информации, разработанный Т. Юханссоном, А. Максимовым и М. Хеллом. Версия со 128-битным ключом стала победителем в категории «Аппаратно-ориентированные алгоритмы» [10].

Основные характеристики алгоритма:

1. длина ключа не должна превышать 128 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход подается уникальный 96-битный вектор инициализации.

Согласно официальной документации в алгоритме Grain определены несколько стандартных функций, определяющих базовые преобразования: регистр сдвига с линейной обратной связью, регистр сдвига с нелинейной обратной связью и выходная функция [15].

Базовые операции представляют собой следующие преобразования.

Секретный ключ разбивается на биты и загружается в регистр сдвига с нелинейной обратной связью (РССНОС), а вектор инициализации – в регистр сдвига с линейной обратной связью (РССЛОС).

РССЛОС, объемом 96 бит, предназначен для генерации нового бита на основе битов вектора инициализации. Формула для генерирования нового бита:

$$s_{i+96} = s_0 \oplus s_7 \oplus s_{38} \oplus s_{70} \oplus s_{81} \oplus s_{96},$$

где  $s_i$  –  $i$ -ый бит РССЛОС.

РССНОС, объемом 128 бит, предназначен для генерирования нового бита на основе битов секретного ключа и 1-го бита регистра с линейной обратной связью. Формула для генерации нового бита:

$$b_{i+128} = s_i \oplus b_i + b_{26} \oplus b_{56} \oplus b_{91} \oplus b_{96} \oplus (b_3 \& b_{67}) \oplus (b_{11} \& b_{13}) \oplus (b_{17} \& b_{18}) \\ \oplus (b_{27} \& b_{59}) \oplus (b_{40} \& b_{48}) \oplus (b_{61} \& b_{65}) \oplus (b_{68} \& b_{84}),$$

где  $b_i$  –  $i$ -ый бит регистра РСЧНОС,  $s_i$  –  $i$ -ый бит РССЛОС.

При старте работы алгоритма, генерация новых битов с помощью РСЧНОС и РССЛОС, повторяется 256 раз без генерации битов ключевого потока. Весь регистр сдвигается на 1-ну позицию влево, а новые сгенерированные биты записываются в последние позиции соответствующих регистров.

Выходом выходной функции является бит ключевой последовательности, который вычисляется по следующей формуле:

$$z_i = \sum_{j \in A} b_{j+i} \oplus h(x) \oplus s_{i+93} \text{ при } A = \{2, 15, 36, 45, 64, 73, 89\},$$

где  $b_i$  –  $i$ -ый бит регистра РСЧНОС,  $s_i$  –  $i$ -ый бит РССЛОС,  $h(x) = (b_{12} \& s_8) \oplus (s_{13} \& s_{20}) \oplus (b_{95} \& s_{42}) \oplus (s_{60} \& s_{79}) \oplus (b_{12} \& b_{95} \& s_{95})$ ,

Таким образом, за одну итерацию алгоритма генерируется 1 бит ключевой последовательности.

## 1.8 Алгоритм шифрования Trivium

Trivium – алгоритм поточного шифрования информации, разработанный К. де Канньером и Б. Пренелом. Шифр стал победителем в категории «Аппаратно-ориентированные алгоритмы» [10].

Основные характеристики алгоритма:

1. длина ключа не должна превышать 80 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход подается уникальный 80-битный вектор инициализации.

Согласно официальной документации в Trivium определены несколько стандартных функций, определяющих базовые преобразования: функция инициализации и функция генерации ключевой последовательности [16].

Базовые операции представляют собой следующие преобразования.

Функция инициализации осуществляет преобразование 80-битного секретного ключа и вектора инициализации в последовательность бит и затем их трансформацию для дальнейшей генерации ключевого потока.

Схематично функцию инициализации можно представить так:

$$\begin{aligned}(s_1, s_2, \dots, s_{93}) &= (K_1, \dots, K_{80}, 0, \dots, 0); \\(s_{94}, s_{95}, \dots, s_{177}) &= (IV_1, \dots, IV_{80}, 0, \dots, 0); \\(s_{178}, s_{179}, \dots, s_{288}) &= (0, \dots, 0, 1, 1, 1); \\ \text{for } i &= 1 \text{ to } 4 * 288 \text{ do} \\ & t_1 = s_{66} \oplus s_{91} \& s_{92} \oplus s_{93} \oplus s_{171}; \\ & t_2 = s_{161} \oplus s_{175} \& s_{176} \oplus s_{177} \oplus s_{264}; \\ & t_3 = s_{243} \oplus s_{286} \& s_{287} \oplus s_{288} \oplus s_{69}; \\ & (s_1, s_2, \dots, s_{93}) = (t_3, s_1, s_2, \dots, s_{92}); \\ & (s_{94}, s_{95}, \dots, s_{177}) = (t_1, s_{94}, s_{95}, \dots, s_{176}); \\ & (s_{178}, s_{179}, \dots, s_{288}) = (t_2, s_{178}, s_{179}, \dots, s_{287}); \\ \text{end for,}\end{aligned}$$

где  $s$  – 288-ми битовый регистр,  $K_i$  –  $i$ -й бит ключа,  $IV_i$  –  $i$ -й бит вектора инициализации,  $t_1, t_2, t_3$  – биты для промежуточных вычислений.

После описанного преобразования регистр  $s$  подается на вход функции генерации ключевой последовательности (алгоритм приведен ниже):

$$\begin{aligned}\text{for } i &= 1 \text{ to } N \text{ do} \\ & t_1 = s_{66} \oplus s_{93}; \\ & t_2 = s_{162} \oplus s_{177}; \\ & t_3 = s_{243} \oplus s_{288}; \\ & z_i = t_1 \oplus t_2 \oplus t_3; \\ & t_1 = t_1 \oplus s_{91} \& s_{92} \oplus s_{171}; \\ & t_2 = t_2 \oplus s_{175} \& s_{176} \oplus s_{264}; \\ & t_3 = t_3 \oplus s_{286} \& s_{287} \oplus s_{69}; \\ & (s_1, s_2, \dots, s_{93}) = (t_3, s_1, s_2, \dots, s_{92}); \\ & (s_{94}, s_{95}, \dots, s_{177}) = (t_1, s_{94}, s_{95}, \dots, s_{176});\end{aligned}$$

$$(s_{178}, s_{179}, \dots, s_{288}) = (t_2, s_{178}, s_{179}, \dots, s_{287});$$

end for,

где  $N$  – длина входного потока в битах,  $z_i$  – бит ключевой последовательности.

## 1.9 Алгоритм шифрования Mickey

Mickey – алгоритм поточного шифрования информации, разработанный С. Бэббиджем и М. Доддом. Участвовал в проекте eStream, где стал победителем в категории «Аппаратно-ориентированные алгоритмы» [10].

Основные характеристики алгоритма:

1. длина ключа не должна превышать 80 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход подается уникальный 80-битный вектор инициализации.

Согласно официальной документации в алгоритме Mickey определены несколько стандартных функций, определяющих базовые преобразования: сдвиг регистра  $R$  (CLOCK\_R), сдвиг регистра  $S$  (CLOCK\_S), функция управления алгоритмом (CLOCK\_KG), функция загрузки ключа и вектора инициализации и функция генерирования ключевой последовательности [17].

Рассмотрим базовые преобразования. Функция сдвига регистра  $R$  принимает несколько параметров: CLOCK\_R ( $R$ ,  $input\_bit\_r$ ,  $control\_bit\_r$ ). Здесь  $R$  – это указатель на сам регистр, а  $input\_bit\_r$  и  $control\_bit\_r$  – специальные биты. Данное преобразование выглядит следующим образом:

$$\text{CLOCK\_R}(R, input\_bit\_r, control\_bit\_r)$$

$$feedback\_bit = r_{99} \oplus input\_bit\_r;$$

$$\text{для } 1 \leq i \leq 99, r'_i = r_{i-1}; r'_0 = 0;$$

$$\text{для } 0 \leq i \leq 99, \text{ если } i \in RTAPS, r'_i = r'_i \oplus feedback\_bit;$$

$$\text{если } control\_bit\_r = 1, \text{ то}$$

$$\text{для } 0 \leq i \leq 99, r'_i = r'_i \oplus r_i,$$

где  $r_0, r_1, \dots, r_{99}$  – биты регистра  $R$  до сдвига,  $r'_1, r'_2, \dots, r'_{99}$  – биты регистра  $R$  после сдвига,  $RTAPS$  – набор входов обратной связи.

Массив *RTAPS* определен следующим образом:  $RTAPS = \{ 0, 1, 3, 4, 5, 6, 9, 12, 13, 16, 19, 20, 21, 22, 25, 28, 37, 38, 41, 42, 45, 46, 50, 52, 54, 56, 58, 60, 61, 63, 64, 65, 66, 67, 71, 72, 79, 80, 81, 82, 87, 88, 89, 90, 91, 92, 94, 95, 96, 9 \}$ .

Функция сдвига регистра *S* так же принимает несколько параметров:  $CLOCK\_S(S, input\_bit\_s, control\_bit\_s)$ . Здесь *S* – это указатель на регистр, а *input\_bit\_s*, *control\_bit\_s* – специальные биты. Псевдокод сдвига регистра *S*:

```

CLOCK_S(S, input_bit_s, control_bit_s)
feedback_bit =  $s_{99} \oplus input\_bit\_s$ ;
для  $1 \leq i \leq 98$ ,
 $\hat{s}_i = s_{i-1} \oplus ((s_i \oplus COMP0_i) \& (s_{i+1} \oplus COMP1_i))$ ;  $\hat{s}_0 = 0$ ;  $\hat{s}_{99} = s_{98}$ ;
если control_bit_s = 0, то
для  $0 \leq i \leq 99$ ,  $s'_i = \hat{s}_i \oplus (FB0_i \& feedback\_bit)$ ;
если control_bit_s = 1, то
для  $0 \leq i \leq 99$ ,  $s'_i = \hat{s}_i \oplus (FB1_i \& feedback\_bit)$ ,

```

где  $s_0, s_1, \dots, s_{99}$  – биты регистра *S* до сдвига,  $\hat{s}_0, \hat{s}_1, \dots, \hat{s}_{99}$  – промежуточные значения битов регистра,  $s'_0, s'_1, \dots, s'_{99}$  – биты регистра *S* после сдвига.

Массивы *COMP0*, *COMP1*, *FB0* и *FB1*, в зависимости от раунда, определяются согласно таблицам смещения [17].

Порядок работы алгоритма регулируется функцией управления. Функция принимает 4 параметра:  $CLOCK\_KG(R, S, mixing, input\_bit)$ . Здесь *R* и *S* указатели на регистры *R* и *S* соответственно, а *mixing* и *input\_bit* являются служебными битами. На псевдокоде данную функцию можно записать так:

```

CLOCK_KG(R, S, mixing, input_bit)
control_bit_r =  $s_{34} \oplus r_{67}$ ;
control_bit_s =  $s_{67} \oplus r_{33}$ ;
если mixing = TRUE, тогда input_bit_r =  $input\_bit \oplus s_{50}$ ,
а если mixing = FALSE, тогда input_bit_r = input_bit;
input_bit_s = input_bit;
CLOCK_R(R, input_bit_r, control_bit_r);

```



$\text{CLOCK\_S}(S, \text{input\_bit\_s}, \text{control\_bit\_s}).$

При начале работы алгоритма происходит загрузка секретного ключа и вектора инициализации в регистры  $R$  и  $S$  соответственно.

Псевдокод данной функцию приведен ниже.

$(r_0, r_1, \dots, r_{99}) = (0, \dots, 0);$

$(s_0, s_1, \dots, s_{99}) = (0, \dots, 0);$

для  $0 \leq i \leq \text{длина IV} - 1,$

$\text{CLOCK\_KG}(R, S, \text{mixing} = \text{TRUE}, \text{input\_bit} = \text{IV}_i);$

для  $0 \leq i \leq 79,$

$\text{CLOCK\_KG}(R, S, \text{mixing} = \text{TRUE}, \text{input\_bit} = K_i);$

для  $0 \leq i \leq 99,$

$\text{CLOCK\_KG}(R, S, \text{mixing} = \text{TRUE}, \text{input\_bit} = 0),$

где  $K_i$  –  $i$ -й бит секретного ключа,  $\text{IV}_i$  –  $i$ -й бит вектора инициализации.

Функция генерирования ключевой последовательности на псевдокоде выглядит следующим образом:

для  $0 \leq i \leq L - 1,$

$z_i = r_0 \oplus s_0;$

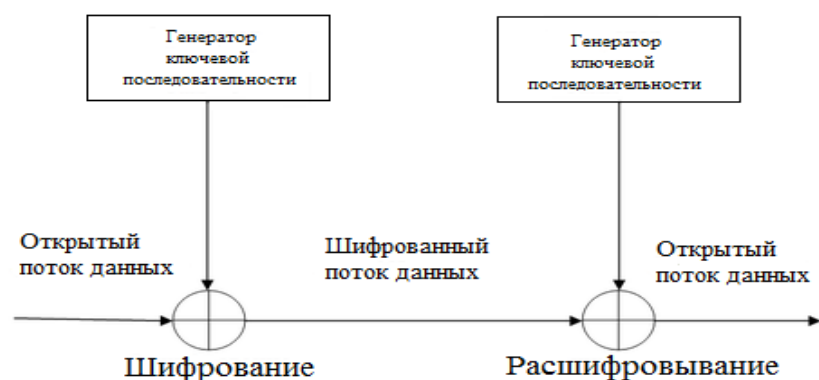
$\text{CLOCK\_KG}(R, S, \text{mixing} = \text{FALSE}, \text{input\_bit} = 0),$

где  $L$  – длина входного потока,  $z_i$  –  $i$ -й бит ключевой последовательности.

### 1.10 Общая схема функции шифрования/расшифровывания

Алгоритмы проекта eStream за одну итерацию генерируют последовательность байт, называемую ключевой последовательностью. Для каждого шифра длина ключевой последовательности определена согласно официальной документации. Эта последовательность байт используется для побитового сложения по модулю 2 с каждым байтом входного потока данных. Вследствие чего на выходе получается зашифрованный поток данных.

Функция расшифровывания аналогична функции шифрования. Блок-схема процесса шифрования/расшифровывания представлена на Рисунке 1.



*Рисунок 1 – Блок схема процесса шифрования/расшифровывания*

### **1.11 Алгоритмы проекта eStream в действующих проектах**

В виду большой скорости шифрования входного потока и повышенной криптостойкости, которая была заложена в алгоритмы, область применения шифров проекта eStream достаточно разнообразна:

1. шифрование потоков данных в режиме реального времени (системы мгновенного обмена сообщениями, видеоконференции, защищенный канал передачи данных и другое);
2. шифрование информации в режиме блочных шифров (шифрование файлов, разделов жестких дисков и другое);
3. аппаратные комплексы шифрования (организация защищенных каналов связи);
4. использование алгоритмов как составной части системы шифрования (как промежуточная стадия шифрования информации или как одновременное использование нескольких алгоритмов шифрования).

Результаты работы проекта eStream используются в криптографической библиотеке Crypto++, распространяющейся под лицензией «Boost Software License 1.0». Проект Crypto++ является кроссплатформенным, реализован преимущественно на языке программирования C++ [18]. Объединяет несколько типов криптографических алгоритмов: потоковые, блочные, хэш-функции, ассиметричные. Раздел потоковые шифры содержит алгоритмы

проекта eStream из категории «программно-ориентированные алгоритмы»: Salsa и Sosemanuk.

Шифр HC128 используется в кроссплатформенной криптографической библиотеке Bounce Castle, которая разрабатывается австралийскими исследователями. Библиотека разработана на языках программирования java и C# и распространяется под лицензией «Massachusetts Institute of Technology License». Последняя публикация проекта датируется мартом 2015 года [19].

Алгоритм Rabbit входит в состав веб-приложения для шифрования информации Crypto Tools, которое распространяется под лицензией «GNU GPL». Приложение встраивается в браузер в виде дополнительного расширения и предназначено для шифрования небольших объемов информации [20].

Шифры Grain и Trivium используются в библиотеке AVR Crypto lib, которая ориентирована преимущественно для работы на микроконтроллерах. Разработкой библиотеки занимается немецкая лаборатория Das Labor. Библиотека реализована на языке программирования C и распространяется под лицензией «GPLv3» [21].

Исходные коды алгоритмов проекта eStream, которые включены в состав вышеописанных проектов, используются в том виде, в котором их предложили авторы шифров, без добавления каких-либо модификаций.

В результате изучения проекта возникла идея разработки библиотеки, которая объединяла бы в себе алгоритмы-победители проекта eStream, предоставив разработчикам программного обеспечения удобный интерфейс взаимодействия с шифрами.

### **1.12 Безопасность алгоритмов проекта eStream**

В проекте eStream, наряду со скоростью шифрования информации, изучался аспект стойкости алгоритмов к современным методам атак. Для этих целей была разработана платформа для анализа шифров, так называемый Testing framework [22].

За время работы проекта криптографические шифры подвергались различным испытаниям и тестам: анализ свойств генератора ключевой последовательности; анализ процедур инициализации (процедуры установки секретного ключа и вектора инициализации); атака по времени; атака по сторонним каналам; дифференциальная атака и многие другие [23].

Все 7 алгоритмов-победителей успешно прошли испытания. Подробные результаты тестов, а так же отчеты и протоколы конференций, содержатся в книге «New Stream Cipher Designs» и на официальном сайте проекта eStream в разделе «Результаты тестирования» [24].

## ГЛАВА 2 БИБЛИОТЕКА `estream.h`

Организаторы представили результаты конкурса как разрозненные проекты, независимые друг от друга. Единственным исключением является то, что функции, используемые в каждом шифре, называются одинаково. Но не было представлено единого механизма, который бы позволил использовать конкурсные разработки в рамках единого проекта. Сторонним разработчикам предлагается самим выбрать нужный алгоритм и интегрировать его со своими разработками.

В ходе выполнения практической части выпускной квалификационной работы разработана криптографическая библиотека потокового шифрования информации, которая объединила в себе 7 алгоритмов проекта `eStream`. В библиотеке использованы модифицированные реализации шифров и предоставлен удобный интерфейс для интеграции алгоритмов шифрования в более крупные проекты.

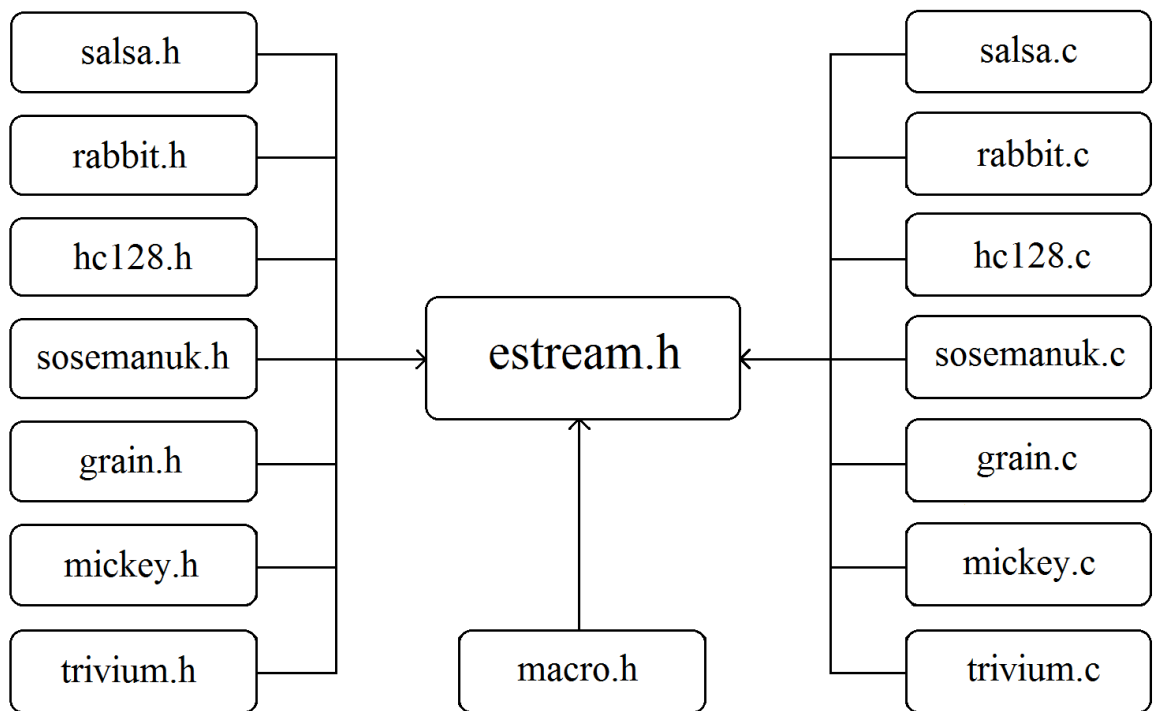
### 2.1 Структура библиотеки `estream.h`

Библиотека получила название `estream.h` в честь проекта, который подарил миру новые криптографические алгоритмы. Она была разработана на языке программирования C.

Библиотека насчитывает 16 файлов. 9 заголовочных файлов (header-файлы): `estream.h`, `macro.h`, `salsa.h`, `rabbit.h`, `hc128.h`, `sosemanuk.h`, `grain.h`, `mickey.h`, `trivium.h`; и 7 исходных файлов: `salsa.c`, `rabbit.c`, `hc128.c`, `sosemanuk.c`, `grain.c`, `mickey.c`, `trivium.c`.

Структура библиотеки `estream.h` представлена на Рисунке 2.

Основным файлом всего проекта является заголовочный файл `estream.h`, который разработчикам необходимо подключать к своей работе. Файл `estream.h` содержит инструкции, которые подгружают остальные header-файлы.



*Рисунок 2 – Структура библиотеки estream.h*

Подгружаемые в estream.h файлы содержат базовый набор функций доступных разработчикам и специально сформированные структуры данных. Все подгружаемые заголовочные файлы называются в зависимости от алгоритма шифрования, который они описывают. Например, файл salsa.h содержит исходный код, описывающий криптографический шифр Salsa.

Каждый заголовочный файл содержит описание структуры данных, которая представляет собой объединение нескольких переменных. В листинге 1 приведен исходный код структуры данных для алгоритма Salsa (header-файл salsa.h).

*Листинг 1 – Исходный код структуры данных salsa\_context*

```

struct salsa_context {
    int keylen;
    int ivlen;
    uint8_t key[32];
    uint8_t iv[16]
    uint32_t x[16]; }

```

Для каждого алгоритма в соответствующем header-файле заведена своя структура данных. Причем количество полей в структуре данных в каждом алгоритме разное, а общими являются следующие поля:

1. `keylen` – длина секретного ключа в байтах;
2. `ivlen` – длина вектора инициализации в байтах;
3. `key` – массив, содержащий биты секретного ключа;
4. `iv` – массив, содержащий биты вектора инициализации.

Использование структур данных позволяет облегчить взаимодействие разработчика с библиотекой, поскольку после однократного заполнения структуры нужными значениями, разработчику нет необходимости заботиться о том, что происходит с этими значениями в процессе работы алгоритма.

Помимо структур данных, каждый header-файл содержит ссылки на 3 функции: `*_set_key_and_iv`, `*_crypt`, `*_test_vectors` (\* – обозначает один из 7-ми алгоритмов шифрования).

Функция `*_set_key_and_iv(struct *ctx, const uint8_t *key, const int keylen, const uint8_t *iv, const int ivlen)` предназначена для заполнения структуры данных битами секретного ключа и вектора инициализации. Перед заполнением структуры происходит ее инициализация (заполнение всех полей нулями). Это необходимо для устранения попадания «мусора» в поля структуры.

Функция принимает 5 аргументов: указатель на структуру данных, указатель на секретный ключ, длину секретного ключа, указатель на вектор инициализации и длину вектора инициализации. При успешном выполнении функция возвращает 0, в случае ошибки возвращает значение –1.

Функция `*_crypt(struct *ctx, const uint8_t *buf, uint32_t buflen, uint8_t *out)` шифрует/расшифровывает входной поток данных. Функция принимает на вход 4 аргумента: указатель на структуру данных, указатель на входной поток данных, длину входного потока в байтах и указатель на выходной массив данных, в который будет помещен результат шифрования/расшифровывания. Код возврата функции отсутствует.

Функция `*_test_vectors(struct *ctx)` предназначена для вывода ключевого потока, который генерируется за одну итерацию алгоритма. Длина ключевого потока зависит от алгоритма. Аргументом функции является указатель на заполненную ключом и вектором инициализации структуру данных. Код возврата функции отсутствует.

Заголовочный файл `macro.h` содержит описания некоторых математических преобразований, которые являются общими для нескольких криптографических алгоритмов. Подобный подход позволил уменьшить дублирование исходного кода. В `header-файле macro.h` содержатся макросы для обеспечения кроссплатформенности библиотеки и макросы циклических сдвигов машинных слов.

Библиотека `estream.h` поддерживает два основных порядка байтов: `big-endian` и `little-endian`.

`Big-endian` (порядок от старшего к младшему) – запись машинного слова начинается со старшего разряда и заканчивается младшим. Подобный порядок байтов поддерживается процессорами `IBM 360/370/390`, `MIPS`, `SPARC` и другими моделями процессоров.

`Little-endian` (порядок от младшего к старшему) – запись машинного слова начинается с младшего разряда и заканчивается старшим. Подобный порядок байтов впервые стала использовать компания `Intel`, разработавшая процессоры с `x86` архитектурой.

Если же в процессоре используется иной порядок байтов, что маловероятно, то библиотека выдаст ошибку `«unsupported byte order»` (порядок байт не поддерживается).

Наряду с заголовочными файлами библиотека содержит 7 файлов описания, которые содержат исходный программный код криптографических алгоритмов на языке программирования `C`. Файл описания носит такое же имя, как и заголовочный файл только с расширением `.c`. Например, пара: `header-файл salsa.h` и его файл описание `salsa.c`.



Разработанная библиотека `estream.h` предоставляет удобный интерфейс взаимодействия между разработчиком и криптографическими алгоритмами.

## **2.2 Механизмы модернизации**

Главным достоинством потоковых криптографических алгоритмов является высокая скорость шифрования информации. Поэтому недостаточно просто реализовать шифр. Необходимо предложить такую реализацию, которая бы обеспечила высокую производительность алгоритма и смогла бы работать на различных архитектурах и платформах. Подобный подход был использован при разработке библиотеки.

Раздел посвящен механизмам и способам модернизации алгоритмов для увеличения скоростных показателей, достижения кроссплатформенности при стабильной и корректной работе шифров.

### **2.2.1 Использование макросов**

Макрос (в терминологии языка программирования C) – символьное имя, заменяемое при обработке препроцессором на последовательность программных инструкций [25].

Макросы используются для уменьшения дублирования программного кода, а также для облегчения понимания исходного кода, когда макросом заменяется какая-либо часть сложной составной инструкции.

Если препроцессор, при обработке файла с исходным кодом, встречает символьное имя, которое определено с помощью директивы `#define`, то он подставляет значение макроса вместо символьного имени.

Подобное поведение препроцессора можно использовать для увеличения скорости работы алгоритма. Вместо многократного использования функции, при котором время затрачивается на ее вызов, можно использовать макрос. В ходе обработки файла препроцессор сам подставит необходимый программный код. Такой подход позволяет получить выигрыш по времени, если набирается большое количество вызовов функции.

Например, в алгоритме HC128 (hc128.c) используется макрос GENERATE\_P, его исходный код приведен в листинге 2. Данный макрос производит генерацию ключевого потока.

Листинг 2 – Исходный код макроса GENERATE\_P

```
#define GENERATE_P(ctx, a, b, c, d, e, f, res) {  
    uint32_T res1, res2;  
    G1(ctx->x[e], ctx->x[d], ctx->w[b], res1);  
    H1(ctx, ctx->x[f], res2);  
    ctx->w[a] += res1;  
    ctx->x[c] = ctx->w[a];  
    res = U32TO32((res2 ^ ctx->w[a]));  
}
```

При шифровании 512 байт информации данный макрос вызывается 32 раза. Использование макроса в данном случае дает значительный выигрыш в скорости, по сравнению с вызовом аналогичной функции 32 раза. С ростом объемов данных выигрыш в скорости становится более ощутимым, что показывают проведенные исследования, представленные далее.

### 2.2.2 Использование битовых операций

Язык программирования C представляет удобный инструмент для работы с битами. В рамках разработки библиотеки использовались следующие битовые операции: левый битовый сдвиг, правый битовый сдвиг, побитовое отрицание, побитовое сложение и побитовое умножение.

Использование битовых операций позволяет увеличить производительность работы алгоритмов, так как все операции производятся непосредственно с битами машинного слова. Пример комплексного использования битовых операций приведен в исходном коде макроса U32TO32 (файл macro.h), который в зависимости от порядка байтов, выполняет заданное преобразование. Исходный код макроса приведен в листинге 3.

```

#if __BYTE_ORDER == __BIG_ENDIAN
#define U32TO32(x)
    ((x<<24) | ((x<<8) & (0xFF0000)) | ((x>>8) & 0xFF00) | (x>>24))
#elif __BYTE_ORDER == __LITTLE_ENDIAN
#define U32TO32(x)    (x)
#else
#error unsupported byte order
#endif

```

В приведенном макросе используются битовые сдвиги, побитовые умножения и сложения, которые производят поворот машинного слова. Подобная конструкция практически не затрачивает время ввиду того, что побитовые операции являются операциями низкого уровня.

### 2.3 Модернизация алгоритмов

В ходе изучения официальной документации и исходных кодов реализаций криптографических алгоритмов, предложенные их авторами и находящиеся в открытом доступе, были найдены участки кода, изменение которых позволило увеличить производительность алгоритмов.

В алгоритмах Salsa и Sosemanuk была модифицирована функция шифрования/расшифровывания. Было уменьшено количество обращений к входному потоку данных за счет использования 32-битных массивов машинных слов, генерируемых за одну итерацию алгоритма. После модернизации, функция шифрования в алгоритме Salsa способна за один вызов обработать до 512 бит данных, вместо 64-х бит, а в алгоритме Sosemanuk – до 640 бит за один проход, вместо 80 проходов по 8 бит. Общая схема модернизации для алгоритма Salsa приведена в листинге 4.

Листинг 4 – Исходный код функции шифрования алгоритма Salsa

```
uint32_t keystream[16];
```

```

for(; buflen >= 64, buflen -= 64, buf += 64, out += 64) {
    salsa20(ctx, keystream);
    *(uint32_t*)(out+0) = *(uint32_t*)(buf+0) ^ keystream[0];
    *(uint32_t*)(out+4) = *(uint32_t*)(buf+0) ^ keystream[1];
    .
    .
    *(uint32_t*)(out+60) = *(uint32_t*)(buf+0) ^ keystream[16];
}

```

Исходный код функции шифрования алгоритма Sosemanuk аналогичен вышеприведенному, лишь с изменением размера массива `keystream`.

В алгоритме Trivium разработчиками предложено использовать один макрос для инициализации начального состояния алгоритма и генерации ключевой последовательности. При инициализации алгоритма происходит генерация 32-х битов ключевой последовательности, которая на данном этапе шифра не используется, то есть ресурсы процессора, затраченные на процесс инициализации, расходуются нерационально. Так же авторами шифра предложено использовать два макроса, с использованием кроссплатформенных преобразований, для загрузки и выгрузки битов секретного ключа и вектора инициализации при каждой итерации цикла. В реализации алгоритма Trivium, используемой в библиотеке `estream.h`, исключена генерация ключевой последовательности при инициализации алгоритма, а так же произведена замена макросов выгрузки битов на стандартную функцию языка программирования C – `memcpy`.

Достижение кроссплатформенности достигается при загрузке ключа и вектора инициализации в структуру данных. В листинге 5 представлены модернизированные макросы для инициализации начального состояния и генерации ключевой последовательности. Макрос `WORK_1` предназначен для инициализации начального состояния, а `WORK_2` – для генерации ключевой последовательности.

## Листинг 5 – Макросы WORK\_1 и WORK\_2

```
#define WORK_1 {
    uint32_t t1, t2, t3;
    T(w);
    UPDATE(w); }

#define WORK_2{
    uint32_t t1, t2, t3;
    T(w);
    z = t1 ^ t2 ^ t3;
    UPDATE(w); }
```

Наибольший выигрыш в скорости удалось добиться в алгоритме Grain путем применения для вычисления битов ключевой последовательности макросов, работающих с массивами регистров не через взаимодействие со структурой данных, а напрямую с битами регистров. Исходный код макросов приведен в листинге 6. Макрос LFSR предназначен для генерации нового бита линейного регистра, макрос NFSR – для генерации нового бита нелинейного регистра, макрос OUTBIT – для генерации нового бита ключевой последовательности

## Листинг 6 – Макросы LFSR, NFSR, OUTBIT

```
#define LFSR(S) (s[0] ^ s[7] ^ s[38] ^ s[70] ^ s[81] ^ s[96])

#define NFSR(b, s) \
    (s[0] ^ b[26] ^ b[56] ^ b[91] ^ b[96] ^ \
    (b[3] & b[57]) ^ (b[11] & (b[13] ^ (b[17] & b[18])) ^ \
    (b[27] & (b[59] ^ (b[40] & b[48]) ^ (b[61] & b[65])) ^ \
    (b[68] & b[84]))

#define OUTBIT(b, s) \
    (b[2] ^ b[15] ^ b[36] ^ b[45] ^ b[64] ^ b[73] ^ b[89] ^ H(b, s) ^ s[93])
```

Подобный подход позволил увеличить производительность алгоритма Grain более чем в 2 раза. Результаты тестов будут представлены в следующих разделах.

Так же во всех алгоритмах проекта eStream не была заложена проверка на длину вводимых секретного ключа и вектора инициализации. Исходный код проверки для алгоритма Salsa приведен в листинге 7.

Листинг 7 – Процедура проверки длины секретного ключа и вектора инициализации для алгоритма Salsa

```
if((keylen <= 32) && (keylen > 0))
    ctx->keylen = keylen;
else
    return -1;
if((ivlen > 0) && (ivlen <= 8))
    ctx->ivlen = ivlen;
else
    return -1;
```

Для остальных алгоритмов библиотеки estream.h процедура проверки аналогична, за тем исключением, что сравнение производится с максимально возможной длиной секретного ключа и вектора инициализации предъявляемой к каждому алгоритму.

## 2.4 Сравнение библиотеки estream.h и алгоритмов разработчиков

Использование механизмов модернизации в библиотеке estream.h позволили увеличить скоростные показатели некоторых шифров, по сравнению с реализациями, предложенными авторами алгоритмов. Исходные коды разработчиков были взяты с официального сайта проекта [10].

Время, затраченное алгоритмами на процесс шифрования информации, будет вычисляться с помощью UNIX-утилиты time. Утилита time производит 3 измерения одновременно: real (реально затраченное пользовательское время), user (процессорное время, затраченное процессом, без учета времени вызова системных функций), sys (процессорное время, затраченное процессом для вызова системных функций). Более точное вычисление реального

процессорного времени, которое будет затрачено для шифрования потока данных, вычисляет режим измерения user. В данном режиме не учитывается время, которое затрачивает процесс на ожидание своей очереди исполнения в процессоре и время, затраченное на вызов системных функций [26].

Измерения скорости шифрования производились на операционной системе Debian Wheezy с архитектурой x86. Для измерений использовались объемы данных 0,5 МБ (524 288 байт), 10 МБ (10 485 760 байт) и 1 ГБ (1 073 741 824 байт). Компиляция исходных кодов производилась с помощью компилятора GCC версии 4.9.2. (GNU Compiler Collection), в котором были использованы следующие опции:

1. -Wall – усиленная проверка синтаксиса исходного кода;
2. -O3 – 3-й уровень оптимизации ассемблерного кода;
3. -I – путь к каталогам, в которых необходимо искать заголовочные файлы.

Измерения производились на следующих типах процессоров:

1. Intel Atom N2600 Dual Core с тактовой частотой 1,66 ГГц;
2. Intel Quad Core i7 4710HQ с тактовой частотой 2,5 ГГц;
3. AMD Phenom Triple Core 8450 с тактовой частотой 2,1 ГГц.

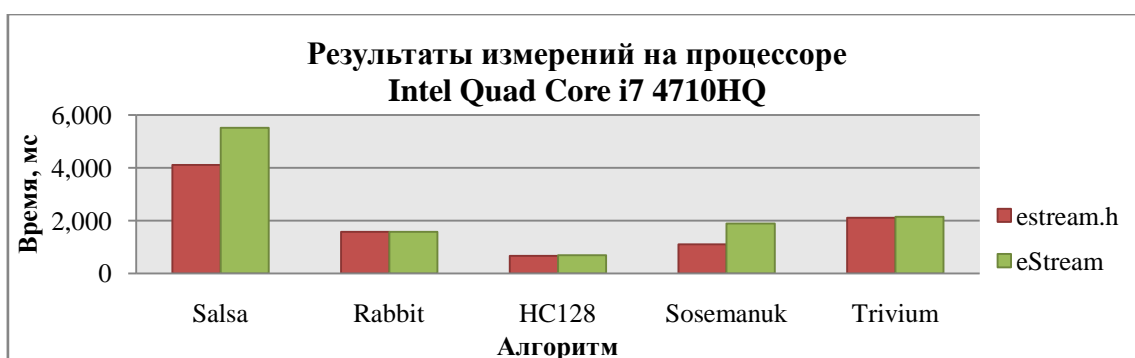
Для сравнения результатов времени работы алгоритмов было использовано среднее арифметическое значение всех полученных измерений времени по каждому шифру. Среднее значение с учетом погрешности наиболее точно характеризует обработку данных различного объема. Для вычисления погрешности среднего арифметического значения была использована формула (1).

$$S_x = \frac{S}{\sqrt{n}} = \sqrt{\sum_{i=1}^n \frac{(x_i - \bar{x})^2}{n(n-1)}}, \quad (1)$$

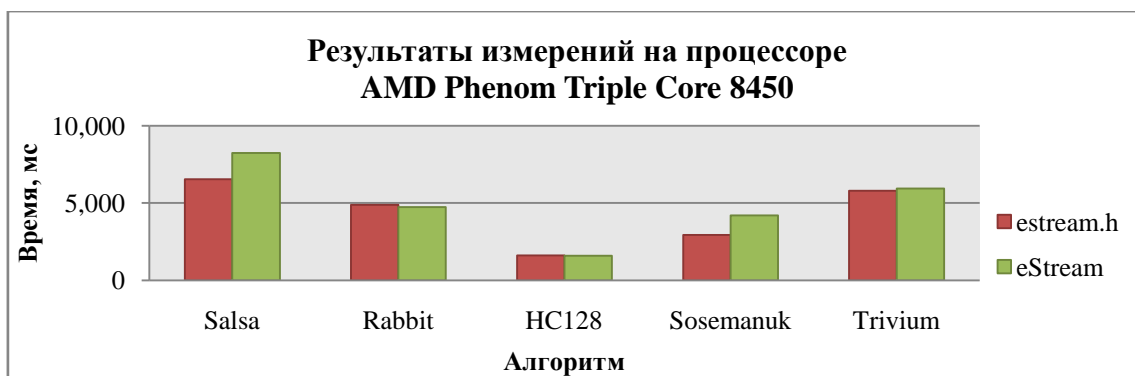
где  $n$  – число измерений,  $\bar{x}$  – среднее значение,  $x_i$  – значение  $i$ -го измерения,  $S_x$  – значение погрешности [27].

На Рисунках 3, 4 и 5 представлены диаграммы с результатами измерений времени на объеме данных 1 Гб для алгоритмов Salsa, Rabbit, HC128, Sosemanuk и Trivium. На рисунках для удобства введены следующие обозначения:

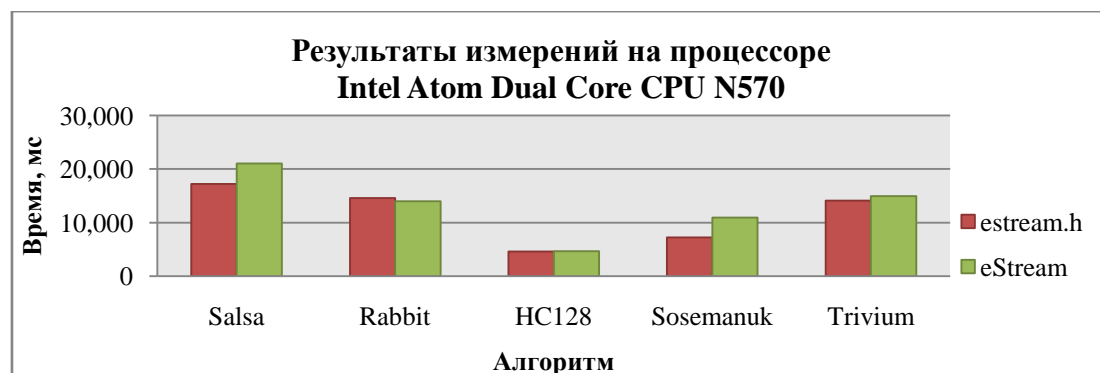
1. estream.h – библиотека разработанная в ходе выполнения выпускной квалификационной работы, содержащая модернизированные реализации алгоритмов;
2. eStream – программные реализации алгоритмов, предложенные авторами шифров, которые стали победителями в проекте eStream.



*Рисунок 3 – Результаты измерений на процессоре Intel Quad Core i7*



*Рисунок 4 – Результаты измерений на процессоре AMD Phenom Triple Core*



*Рисунок 5 – Результаты измерений на процессоре Intel Atom Dual Core*



Аппаратно ориентированные шифры Mickey и Grain тестировались на объемах данных 0,5 МБ и 10 МБ, так как их программные реализации значительно уступают остальным алгоритмам. Результаты тестирования представлены в виде диаграмм на Рисунках 6, 7 и 8.

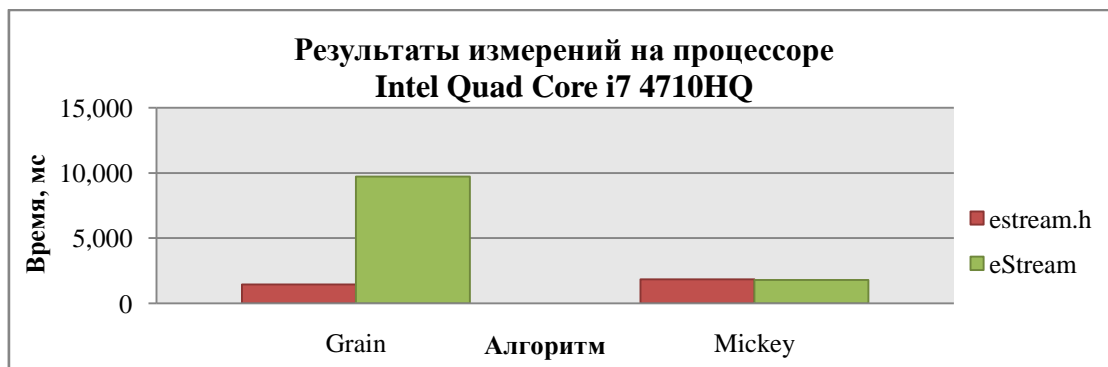


Рисунок 6 – Результаты измерений на процессоре Intel Quad Core i7

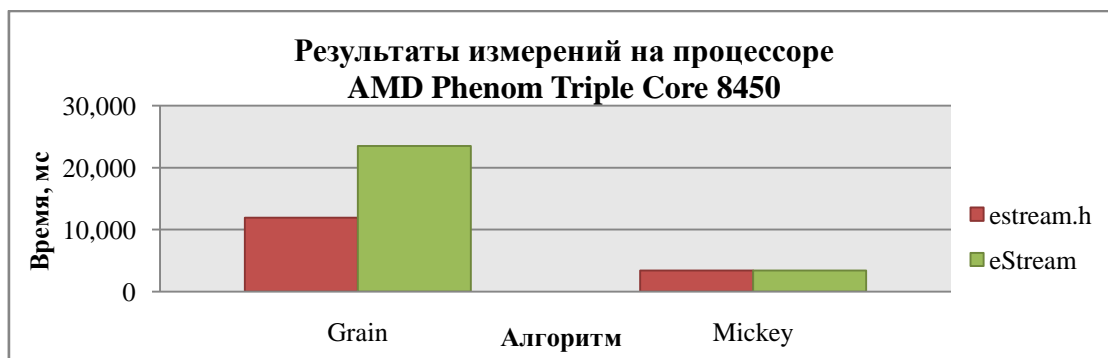


Рисунок 7 – Результаты измерений на процессоре AMD Phenom Triple Core

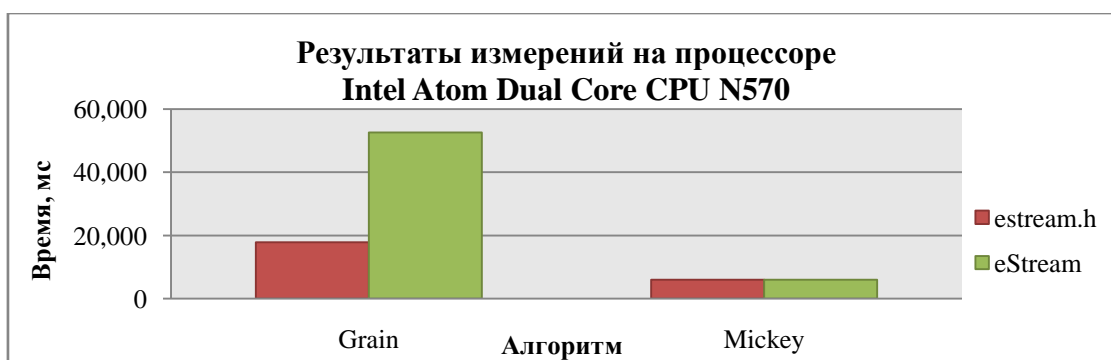


Рисунок 8 – Результаты измерений на процессоре Intel Atom Dual Core

Более подробно результаты измерений с вычислением погрешности представлены в приложениях 1, 2, 3.

По результатам измерений можно заключить следующее:

1. модифицированный алгоритм Salsa из библиотеки estream.h работает на 18-28% эффективнее реализации, предложенной авторами;

2. модифицированный алгоритм Sosemanuk из библиотеки estream.h работает на 25-35% эффективнее реализации, предложенной авторами шифра;
3. модифицированный алгоритм Trivium из библиотеки estream.h работает на 3-5% эффективнее реализации, предложенной авторами шифра;
4. реализация алгоритма Grain из библиотеки estream.h примерно в 2,9 раза быстрее обрабатывает информацию, чем реализация из проекта eStream;
5. реализации алгоритмов HC128 и Mickey, входящие в состав библиотеки estream.h, демонстрируют результаты, которые в пределах погрешности совпадают с результатами реализаций авторов алгоритмов;
6. реализация алгоритма Rabbit из библиотеки estream.h незначительно уступает реализации из проекта eStream, из-за использования авторами алгоритмов специализированных ассемблерных вставок;
7. алгоритм Trivium из категории «аппаратно-ориентированные алгоритмы» имеет эффективную программную реализацию;
8. программные реализации алгоритмов Grain и Mickey значительно уступают программным реализациям других шифров;
9. использование макросов и битовых операций на больших объемах данных приводит к выигрышу по времени;
10. уменьшение количества итераций циклов при шифровании/расшифровывании информации ведет к увеличению производительности алгоритмов.

Исходные коды реализаций алгоритмов проекта eStream, предложенные авторами шифров, а так же разработанной библиотеки estream.h приведены в приложениях 5 и 6 на электронном носителе.



После того как заголовочный файл `estream.h` будет подключен к проекту, разработчику станут доступны 7 алгоритмов шифрования: Salsa, Rabbit, HC128, Sosemanuk, Grain, Mickey, Trivium.

Выбор алгоритма шифрования осуществляется путем использования названия перед соответствующей функцией. Например, в листинге 8 приведен фрагмент исходного кода на языке программирования C, реализующий взаимодействие с библиотекой `estream.h`, путем использования алгоритма шифрования Sosemanuk.

Листинг 8 – Взаимодействие с библиотекой `estream.h`

```
#include «estream.h» \\ подключение библиотеки

int main()
{
    struct sosemanuk_context ctx; \\ использование структуры данных
    if(sosemanuk_set_key_and_iv(ctx, key, 32, iv, 16))
        return -1;
    sosemanuk_crypt(ctx, buf, buflen, buf); \\ функция шифрования
    return 0;
}
```

Использование других алгоритмов шифрования происходит аналогично приведенному выше примеру с изменением первого слова в названиях функций. Количество и тип аргументов в функциях остается неизменным.

Максимальный размер потока данных, который способна обработать библиотека за один вызов функции шифрования равен  $2^{32}$  байтов (максимальное число которое вмещает в себя переменная типа `uint32_t`).

Рекомендуется проверять код возврата функции `*_set_key_and_iv` (где \* – один из 7-ми алгоритмов шифрования) во избежание возникновения ошибок.

### 3.2 Проверка корректности библиотеки estream.h

Ключевым моментом при разработке криптографической библиотеки является проверка алгоритмов на соответствие стандартам. Данная проверка необходима для обеспечения корректности работы шифров и поиска ошибок в программном коде, так как они могут привести к уменьшению стойкости алгоритмов, возникновению линейных зависимостей между открытым и шифрованным потоками данных.

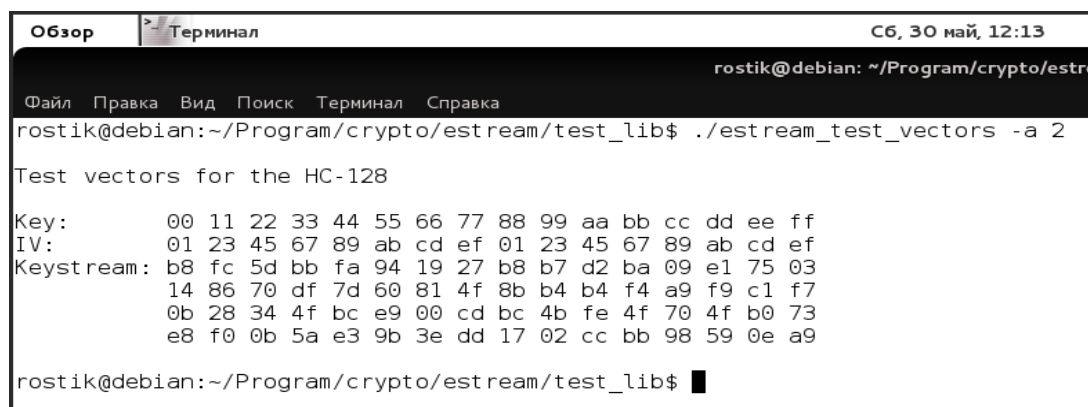
В библиотеке estream.h для проверки алгоритмов используется функция генерирования ключевой последовательности на основе тестовых векторов – \*\_test\_vectors (где \* – один из 7-ми алгоритмов). Под тестовыми векторами понимается пара секретный ключ и вектор инициализации, на основе которых будет вычислена ключевая последовательность, которую необходимо сравнить с последовательностью, представленной в официальной документации. В листинге 9 приведен исходный код на языке программирования C с использованием функции тестовых векторов на примере алгоритма HC128.

Листинг 9 – Использование функции тестовых векторов

```
#define «estream.h» \\ подключение библиотеки
int main()
{
    struct hc128_context ctx; \\ использование структуры данных
    if(hc128_set_key_and_iv(ctx, key, 16, iv, 16))
        return -1;
    hc128_test_vectors(ctx); \\ функция шифрования
    return 0;
}
```

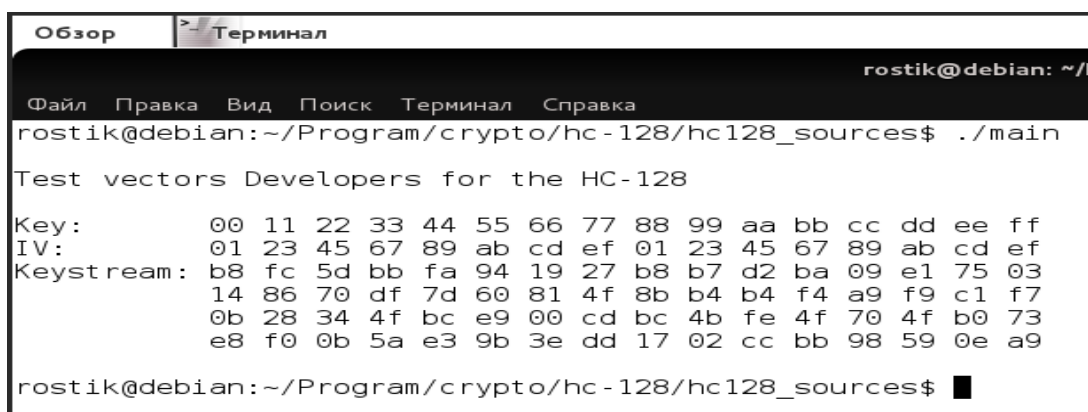
Для тестирования корректности реализации алгоритмов была разработана терминальная программа estream\_test\_vectors, подающая на вход библиотеки estream.h тестовые векторы для каждого реализованного алгоритма. В качестве примера, на Рисунке 10 представлен результат

генерации ключевой последовательности для реализованного в ходе работы алгоритма HC128, а на Рисунке 11 – аналогичный результат, генерируемый алгоритмом, предложенным авторами шифра. Исходный код программы `estream_test_vectors` приведен в приложении 7 на электронном носителе.



```
Обзор Терминал C6, 30 май, 12:13
rostik@debian: ~/Program/crypto/estrea
rostik@debian:~/Program/crypto/estrea/test_lib$ ./estream_test_vectors -a 2
Test vectors for the HC-128
Key:      00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
IV:       01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef
Keystream: b8 fc 5d bb fa 94 19 27 b8 b7 d2 ba 09 e1 75 03
          14 86 70 df 7d 60 81 4f 8b b4 b4 f4 a9 f9 c1 f7
          0b 28 34 4f bc e9 00 cd bc 4b fe 4f 70 4f b0 73
          e8 f0 0b 5a e3 9b 3e dd 17 02 cc bb 98 59 0e a9
rostik@debian:~/Program/crypto/estrea/test_lib$
```

*Рисунок 10 – Генерация ключевой последовательности*



```
Обзор Терминал
rostik@debian: ~/f
rostik@debian:~/Program/crypto/hc-128/hc128_sources$ ./main
Test vectors Developers for the HC-128
Key:      00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
IV:       01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef
Keystream: b8 fc 5d bb fa 94 19 27 b8 b7 d2 ba 09 e1 75 03
          14 86 70 df 7d 60 81 4f 8b b4 b4 f4 a9 f9 c1 f7
          0b 28 34 4f bc e9 00 cd bc 4b fe 4f 70 4f b0 73
          e8 f0 0b 5a e3 9b 3e dd 17 02 cc bb 98 59 0e a9
rostik@debian:~/Program/crypto/hc-128/hc128_sources$
```

*Рисунок 11 – Генерация ключевой последовательности*

По результатам, представленным на Рисунках 10 и 11, можно заключить, что алгоритмы генерируют одинаковую ключевую последовательность, что подтверждает соответствие реализованного алгоритма стандарту.

Для проверки алгоритма на соответствие стандарту можно также использовать терминальную версию программы `escript`, разработанную в данной работе. Более подробно она будет описана в следующем разделе. Суть проверки заключается в шифровании файла алгоритмом из библиотеки `estream.h` и расшифровывании с помощью реализации, предложенной авторами шифра. На Рисунке 12 представлены результаты шифрования и расшифровывания файла `test.png` с использованием алгоритма Rabbit.



```
Обзор Терминал Сб, 6 июн, 11:34
rostik@debian: ~/Program/crypto/estream/test_lib
Файл Правка Вид Поиск Терминал Справка
rostik@debian:~/Program/crypto/estream/test_lib$ ./ecrypt -a 1 -b 10000 -i test.png -o crypt
Enter secret key - 0123456789ABCDEF
Enter vector initialization - 12345678
rostik@debian:~/Program/crypto/estream/test_lib$ ./rabbit_developer -b 10000 -i crypt -o decrypt
Enter secret key - 0123456789ABCDEF
Enter vector initialization - 12345678
rostik@debian:~/Program/crypto/estream/test_lib$ md5sum test.png decrypt
65d11bba71036b538cfd7f9e7a03b35c test.png
65d11bba71036b538cfd7f9e7a03b35c decrypt
rostik@debian:~/Program/crypto/estream/test_lib$
```

*Рисунок 12 – Шифрование и расшифровывание файла test.png*

Файл test.png был зашифрован с помощью программы `ecrypt`. Были использованы следующие опции программы: `-a 1` – выбор алгоритма Rabbit; `-b 10000` – размер считываемого блока в байтах; `-i test.png` – файл для шифрования; `-o crypt` – файл, куда записан результат шифрования.

Затем зашифрованный файл `crypt` был расшифрован с помощью программы `rabbit_developer`, скомпилированной из исходных кодов авторов шифра. Результат расшифровывания сохранен в файл `decrypt`.

С помощью утилиты `md5sum` была вычислена хэш-сумма исходного файла `test.png` и расшифрованного файла `decrypt`. Как видно из Рисунка 12, хэш-суммы совпали. Это подтверждает соответствие реализации алгоритма Rabbit из библиотеки `estream.h` стандарту.

В ходе выполнения работы, все реализации алгоритмов шифрования, представленные в библиотеке `estream.h`, были проверены на соответствие стандартам вышеприведенными способами.

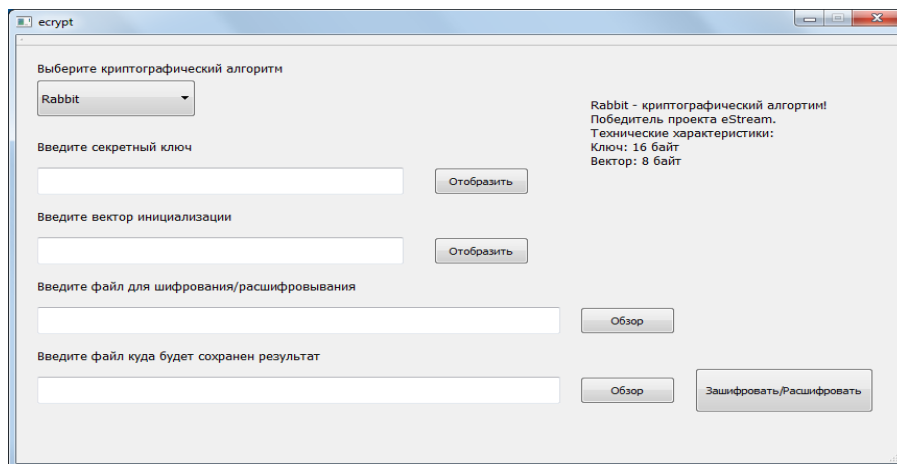
### 3.3 Программа для шифрования файлов

Для демонстрации возможностей библиотеки разработана программа для шифрования/расшифровывания файлов – `ecrypt` в 2-х версиях:

1. с графическим интерфейсом для операционных систем семейства Windows;
2. терминальная версия для операционных систем семейства \*nix.

Графический интерфейс для операционной системы Windows был разработан на платформе Qt Creator (версия Community) с использованием языка программирования C++.

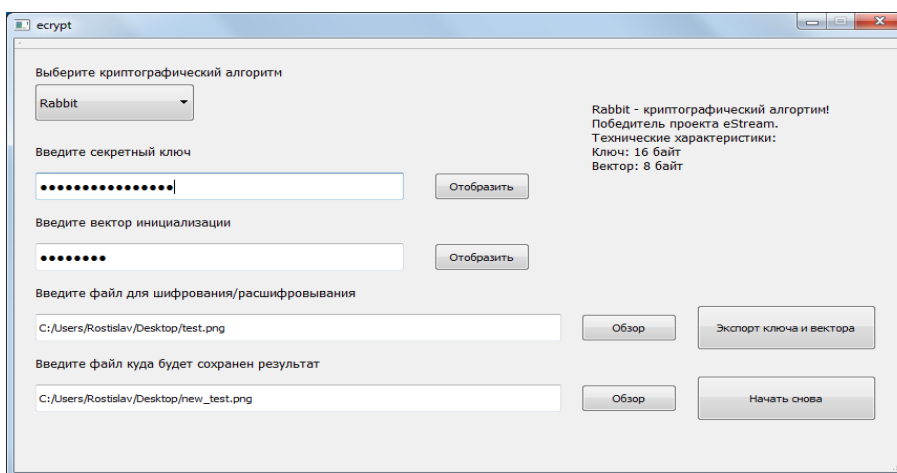
Графический интерфейс программы изображен на Рисунке 13.



*Рисунок 13 – Графический интерфейс программы есcrypt*

Как видно из Рисунка 13, пользователю предлагается ввести секретный ключ, вектор инициализации, выбрать файл для шифрования/расшифрования и файл для записи результата.

При успешном шифровании/расшифровании программа предложит пользователю начать новое шифрование. В случае возникновения каких-либо ошибок в ходе выполнения программы, есcrypt оповестит пользователя, выводом ошибки с помощью дополнительного диалогового окна. На Рисунке 14 изображено состояние программы после успешного завершения шифрования.



*Рисунок 14 – Успешное завершение шифрования/расшифрования*



Исходные коды программы escrypt для операционных систем Windows и семейства \*nix находятся в приложении 8 на электронном носителе.

### 3.4 Клиент-серверное приложение

Основное предназначение потоковой криптографической библиотеки estream.h – это шифрование потоков данных в режиме реального времени, без предварительной записи на носители информации. Для демонстрации возможностей библиотеки разработано клиент-серверное приложение, осуществляющее обмен мгновенными сообщениями между клиентской и серверной частями программы.

Реализованы 2 версии приложения:

1. версия с графическим интерфейсом клиента и сервера для операционных систем семейства Windows;
2. терминальная версия сервера и клиента для операционных систем семейства \*nix.

Приложение для операционной системы Windows было разработано на платформе Qt Creator с использованием языка программирования C++. Клиентская и серверная части программы отличаются набором имеющихся функций и полей для ввода. На Рисунке 15 изображен графический интерфейс серверной части программы.

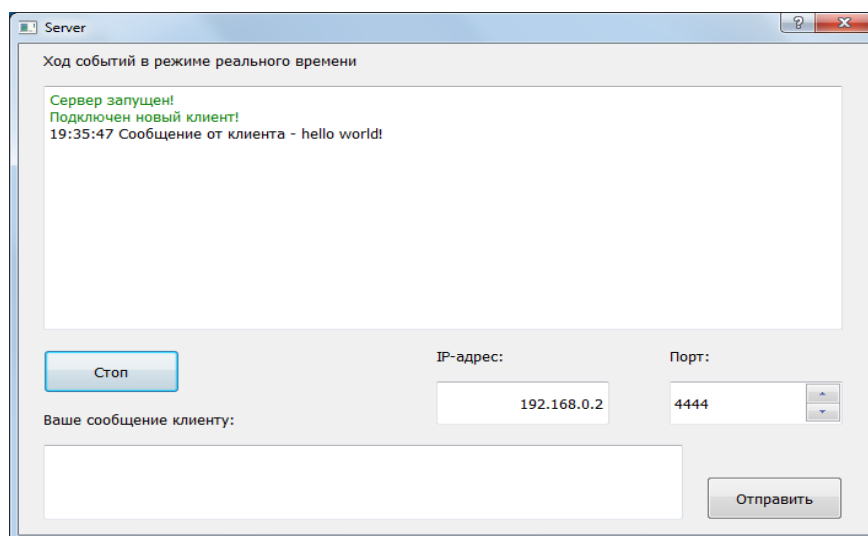
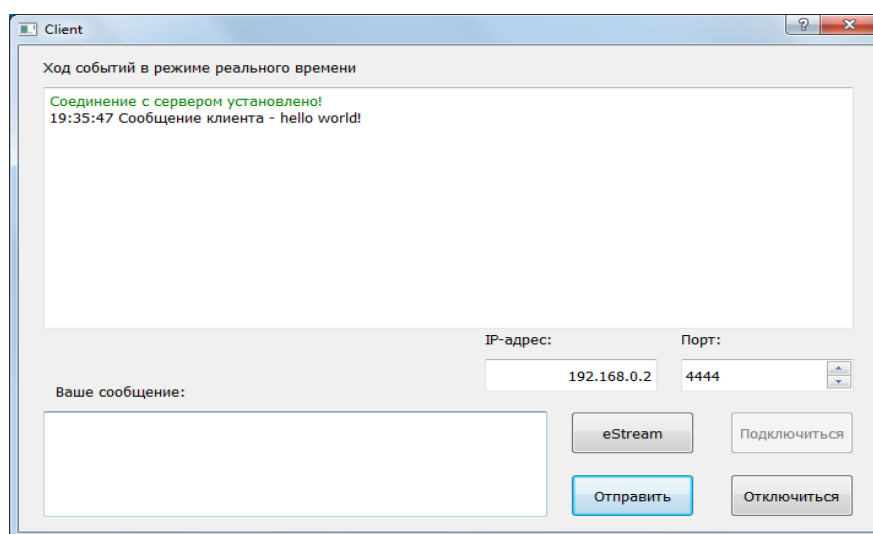


Рисунок 15 – Графический интерфейс серверной части программы

Пользователю необходимо ввести IP-адрес, на котором будет запущен сервер, и номер TCP порта в диапазоне от 1001 до 65535, который будет переведен в режим «Listen» (прослушивание). Поле «Ход событий в режиме реального времени» предназначено для выдачи различной информации пользователю о состоянии сервера: печать ошибок и успешных действий, печать сообщений отправленных клиенту или принятых от клиента.

Клиентская часть приложения отличается своим функционалом от серверной части. Графический интерфейс клиентской части программы изображен на Рисунке 16.



*Рисунок 16 – Графический интерфейс клиентской части программы*

Для подключения клиента, пользователю необходимо ввести IP-адрес сервера и номер порта, который прослушивается сервером. Предназначение поля «Ход событий в режиме реального времени» аналогично серверной части приложения.

Главное отличие клиентской части заключается в наличии настроек шифрования, которые открываются в отдельном диалоговом окне при нажатии на кнопку «eStream». Пользователю доступны 2 режима передачи данных между клиентом и сервером: открытый канал и зашифрованный канал. При открытом канале информация передается в незашифрованном виде, а при зашифрованном канале – посредством шифрования с помощью библиотеки estream.h. Так же пользователь должен выбрать алгоритм шифрования и

ввести секретный ключ. После нажатия на кнопку «ОК» клиент и сервер передут в режим шифрования трафика с помощью выбранного алгоритма. Диалоговое окно с настройками шифрования изображено на Рисунке 17.

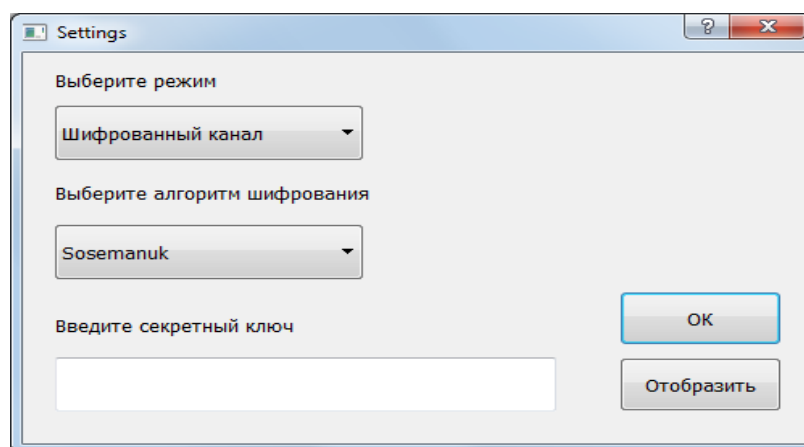


Рисунок 17 – Диалоговое окно с настройками шифрования

При нажатии на кнопку «Отключиться» на главном окне клиентской части программы произойдет разрыв соединения с сервером и уничтожение секретного ключа из памяти приложения.

Для демонстрации возможностей клиент-серверного приложения было отправлено два тестовых сообщения «hello world!» (без кавычек), которые перехвачены с помощью анализатора сетевого трафика Wireshark. На Рисунке 18 изображен перехваченный сетевой пакет с тестовым сообщением, которое было отправлено в режиме «Открытый канал».

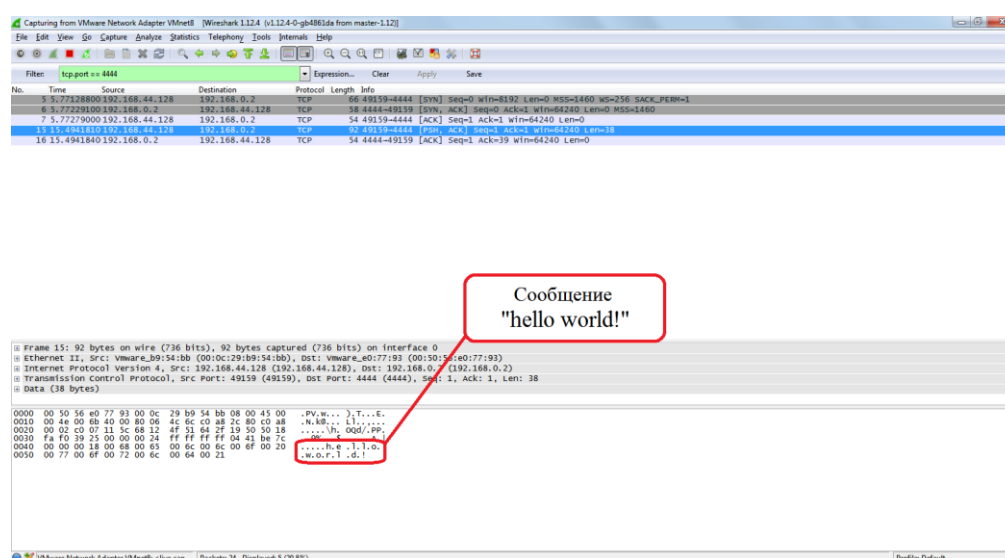


Рисунок 18 – Перехваченное тестовое сообщение, отправленное в режиме «Открытый канал»

На Рисунке 19 изображен перехваченный сетевой пакет с тестовым сообщением, переданным в режиме «Шифрованный канал».

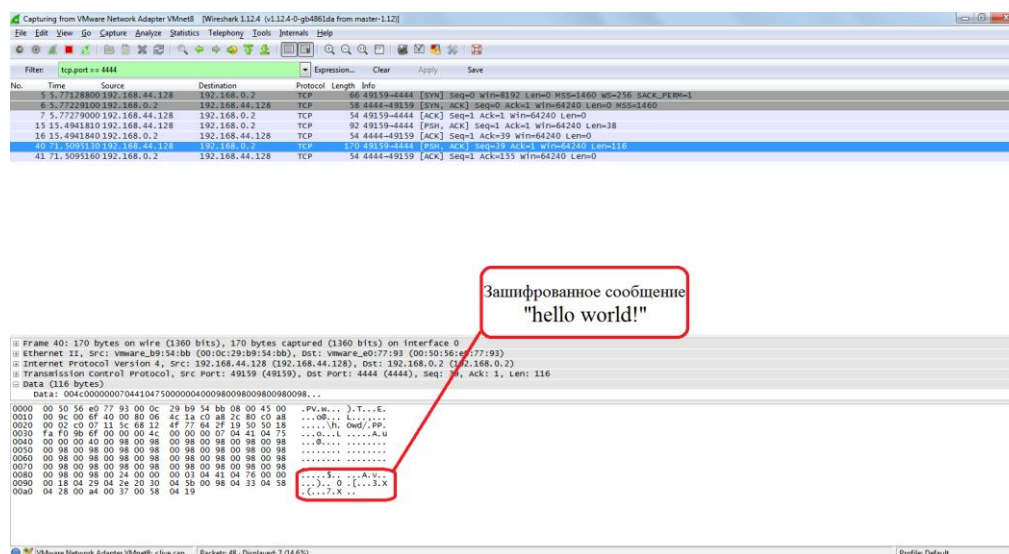


Рисунок 19 – Перехваченное тестовое сообщение, переданное в режиме «Шифрованный канал»

Как видно из Рисунков 18 и 19 тестовое сообщение было зашифровано с помощью алгоритма Sosemanuk из библиотеки estream.h и передано серверной части приложения. Соответственно, процесс чтения и понимания сообщений для третьих лиц становится затруднительным.

Исходные коды клиент-серверного приложения для операционных систем Windows и семейства \*nix представлены в приложении 9 на электронном носителе.

## ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы была разработана криптографическая библиотека потокового шифрования `estream.h` на основе проекта `eStream`. Библиотека включает в себя модифицированные реализации 7-ми алгоритмов: Salsa, Rabbit, HC128, Sosemanuk, Grain, Mickey, Trivium.

В первой главе описывается проект `eStream`. Рассказывается об идее возникновения проекта, ходе проведения конкурсного отбора криптографических алгоритмов, а также раскрывается все более возрастающий интерес разработчиков программного обеспечения в алгоритмах, ставших победителями проекта `eStream`. Даны описания алгоритмов, которые стали победителями проекта. Раскрывается суть базовых преобразований, лежащих в основе каждого шифра.

Глава вторая посвящена непосредственно разработанной библиотеке `estream.h`. В главе приводится описание структуры библиотеки и ее основные функциональные части, рассказывается о способах и механизмах модернизации криптографических алгоритмов. В конце главы приводятся диаграммы, построенные на основе тестов, производимых в ходе разработки библиотеки. На основе полученных результатов сделаны выводы об успешной модернизации некоторых алгоритмов проекта `eStream`.

Принципы взаимодействия с разработанной библиотекой приводятся в третьей главе. Описаны практические рекомендации для пользователей библиотеки по интеграции ее в большие проекты, как составной части системы шифрования. Затем идет описание процедуры проверки корректности реализованных алгоритмов. Приводятся два основных метода проверки соответствия стандартам: с помощью тестовых векторов и взаимозаменяемостью реализаций библиотеки `estream.h` и авторов шифров. Для демонстрации возможностей библиотеки было разработано два

программных приложения: программа для шифрования файлов и клиент-серверное приложение.

Разработанная в рамках данной работы, криптографическая библиотека `estream.h` готова к использованию в составе более крупных проектов, либо как самостоятельный инструмент для шифрования потоков информации.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ И ИСТОЧНИКОВ

1. Алгоритм шифрования AES. [Электронный ресурс] // Официальный сайт Национального института стандартов и технологий США: URL: <http://csrc.nist.gov/publications/fips/fips197/> (Дата обращения: 20.09.2014)
2. ГОСТ Р 34.11-2012 «Информационная технология. Криптографическая защита информации. Функция хэширования». – М.: Стандартинформ, 2012 г.
3. Robshaw M., Billet O. New Stream Cipher Designs: The eStream Finalists (Lecture Notes in Computer Science). – Paris: Springer Science+Business Media, 2008. – 295 с.
4. Алгоритм шифрования AES и его криптоанализ. [Электронный ресурс] // Компьютерра: URL: <http://www.computerra.ru/cio/old/it-market/e-safety/320670/> (Дата обращения: 20.09.2014)
5. Безопасность GSM. [Электронный ресурс] // GSM-security: URL: <http://www.gsm-security.net/1578> (Дата обращения: 20.09.2014)
6. Панасенко С.П. CRYPTREC – проект по выбору криптостандартов Японии. [Электронный ресурс] // Сайт Сергея Панасенко: URL: <http://www.panasenko.ru/Articles/156/156.html> (Дата обращения: 19.09.2014)
7. Скоростное поточное шифрование. [Электронный ресурс] // SecurityLab: URL: <http://www.securitylab.ru/analytics/436620.php> (Дата обращения: 23.09.2014)
8. Панасенко С.П. NESSIE – конкурс криптоалгоритмов. [Электронный ресурс] // Сайт Сергея Панасенко: URL: <http://www.panasenko.ru/Articles/63/63.html> (Дата обращения: 19.09.2014)
9. Проект NESSIE. [Электронный ресурс] // Официальный сайт проекта NESSIE: URL: <http://nessie-project.org/> (Дата обращения: 18.09.2014)
10. Проект eStream. [Электронный ресурс] // Официальный сайт проекта eStream: URL: <http://www.ecrypt.eu.org/stream/index.html> (Дата обращения: 18.09.2014)

11. Алгоритм шифрования Salsa. [Электронный ресурс] // Официальный сайт проекта eStream: URL: <http://www.ecrypt.eu.org/stream/e2-salsa20.html> (Дата обращения: 21.09.2014)
12. Алгоритм шифрования Rabbit. [Электронный ресурс] // Официальный сайт проекта eStream: URL: <http://www.ecrypt.eu.org/stream/e2-rabbit.html> (Дата обращения: 17.11.2014)
13. Алгоритм шифрования HC128. [Электронный ресурс] // Официальный сайт проекта eStream: URL: <http://www.ecrypt.eu.org/stream/e2-hc128.html> (Дата обращения: 04.01.2015)
14. Алгоритм шифрования Sosemanuk. [Электронный ресурс] // Официальный сайт проекта eStream: URL: <http://www.ecrypt.eu.org/stream/e2-sosemanuk.html> (Дата обращения: 14.02.2015)
15. Алгоритм шифрования Grain. [Электронный ресурс] // Официальный сайт проекта eStream: URL: <http://www.ecrypt.eu.org/stream/e2-grain.html> (Дата обращения: 17.02.2015)
16. Алгоритм шифрования Trivium. [Электронный ресурс] // Официальный сайт проекта eStream: URL: <http://www.ecrypt.eu.org/stream/e2-trivium.html> (Дата обращения: 09.01.2015)
17. Алгоритм шифрования Mickey. [Электронный ресурс] // Официальный сайт проекта eStream: URL: <http://www.ecrypt.eu.org/stream/e2-mickey.html> (Дата обращения: 01.02.2015)
18. Crypto++ Library. [Электронный ресурс] // Официальный сайт проекта Crypto++: URL: <http://www.cryptopp.com/> (Дата обращения: 08.05.2015)
19. The Bounce Castle. [Электронный ресурс] // The Legion of the Bounce Castle: URL: <https://www.bouncycastle.org/> (Дата обращения: 08.05.2015)
20. Crypto Tools. [Электронный ресурс] // Официальный сайт сообщества The IBlogBox: URL: <http://iblogbox.com/devtools/crypto/> (дата обращения 09.05.2015)



21. AVR Crypto Lib. [Электронный ресурс] // Официальный сайт лаборатории Das Labor: URL: <http://www.das-labor.org/wiki/AVR-Crypto-Lib/> (Дата обращения: 09.05.2015)
22. eStream Testing framework. [Электронный ресурс] // Официальный сайт проекта eStream: URL: <http://www.ecrypt.eu.org/stream/perf/> (Дата обращения: 21.04.2015)
23. Грибунин В.Г. eStream – дитя лохнесского чудовища. [Электронный ресурс] // Информационная безопасность: URL: [http://www.itsec.ru/articles2/Oborandteh/estream\\_ditya\\_lohnesskogo\\_chudovisha](http://www.itsec.ru/articles2/Oborandteh/estream_ditya_lohnesskogo_chudovisha) (Дата обращения: 22.04.2015)
24. Результаты тестирования алгоритмов проекта eStream. [Электронный ресурс] // Официальный сайт проекта eStream: URL: <http://www.ecrypt.eu.org/stream/perf/#results> (Дата обращения: 21.04.2015)
25. Керниган Б., Ритчи Д. Язык программирования СИ. \ Пер. с англ., 3-е изд., испр. – СПб.: Невский Диалект, 2001. – 352 с.
26. Справочное руководство по утилите time. [Электронный ресурс] // OpenNet: URL: <http://www.opennet.ru/man.shtml?topic=time&category=1> (Дата обращения: 05.05.2015)
27. Новицкий П.В., Зиграф И.А. Оценка погрешностей результатов измерений. – 2-е изд., перераб. и доп. – Л.: Энергоатомиздат. Ленингр. отд-ние, 1991. – 304 с.: ил.
28. Документация по платформе Qt Creator. [Электронный ресурс] // Официальный сайт проекта Qt: URL: <http://doc.qt.io/> (Дата обращения: 02.03.2015)
29. Документация по платформе Qt Creator. [Электронный ресурс] // Все о кроссплатформенном программировании: URL: <http://doc.crossplatform.ru/qt/> (Дата обращения: 02.03.2015)
30. Керниган Б., Пайк Р. Практика программирования: Пер. с англ. – М.: Издательский Дом «Вильямс», 2004. – 288 с.

## ПРИЛОЖЕНИЕ 1

### Результаты измерений на объеме данных 0,5 МБ, мс

№ п/п теста	Процессор	Intel Quad Core i7 4710HQ с тактовой частотой 2,5 GHz							AMD Phenom Triple Core 8450 с тактовой частотой 2,1 GHz							Intel Atom Dual Core CPU N570 с тактовой частотой 1,66 GHz						
	Алгоритм	Salsa	Rabbit	HC128	Sosemanuk	Grain	Mickey	Trivium	Salsa	Rabbit	HC128	Sosemanuk	Grain	Mickey	Trivium	Salsa	Rabbit	HC128	Sosemanuk	Grain	Mickey	Trivium
	Проект																					
1	estream.h	1	1,0	1,0	1,0	71	91	1,0	4,0	5,0	1,0	3,0	590	176	5,0	8	8,0	3,0	4,0	964	310	12,0
	eStream	5,0	1,0	1,0	1,0	457	90	2,0	8,0	4,0	1,0	4,0	1 120	180	5,0	12,0	7	3,0	10,0	2 640	300	12,0
2	estream.h	1	1,0	1,0	1,0	67	89	1,0	4,0	4,0	1,0	3,0	580	172	8,0	10	8,0	2,0	4,0	975	300	8,0
	eStream	4,0	1,0	1,0	1,0	453	89	1,0	12,0	4,0	2,0	4,0	1 120	160	8,0	8,0	8	2,0	6,0	2 679	312	12,0
3	estream.h	1	2,0	2,0	1,0	65	90	1,0	8,0	4,0	1,0	4,0	570	176	5,0	8	8,0	2,0	4,0	916	304	8,0
	eStream	4,0	1,0	2,0	2,0	456	85	2,0	8,0	4,0	1,0	5,0	1 120	164	8,0	12,0	9	2,0	8,0	2 652	304	12,0
4	estream.h	1	1,0	1,0	1,0	71	89	1,0	4,0	4,0	2,0	2,0	580	176	8,0	12	8,0	2,0	4,0	920	304	12,0
	eStream	4,0	2,0	1,0	1,0	448	90	1,0	12,0	3,0	2,0	4,0	1 130	160	8,0	16,0	8	2,0	8,0	2 648	300	12,0
5	estream.h	3	1,0	2,0	1,0	71	89	1,0	8,0	3,0	2,0	3,0	590	184	5,0	8	8,0	2,0	6,0	968	298	8,0
	eStream	4,0	1,0	1,0	1,0	453	85	1,0	8,0	5,0	1,0	4,0	1 110	176	8,0	8,0	8	2,0	12,0	2 644	304	12,0
6	estream.h	3	1,0	1,0	2,0	69	91	1,0	8,0	4,0	2,0	3,0	570	176	8,0	12	8,0	3,0	4,0	925	304	12,0
	eStream	4,0	1,0	1,0	2,0	449	86	1,0	8,0	4,0	1,0	4,0	1 130	172	8,0	16,0	8	2,0	12,0	2 644	315	12,0
7	estream.h	3	2,0	1,0	1,0	72	84	2,0	4,0	3,0	1,0	4,0	580	176	8,0	8	8,0	3,0	8,0	972	304	6,0
	eStream	4,0	1,0	1,0	1,0	459	85	2,0	8,0	4,0	2,0	5,0	1 120	180	5,0	12,0	8	2,0	8,0	2 648	300	8,0
8	estream.h	3	1,0	1,0	2,0	72	90	1,0	4,0	3,0	2,0	4,0	580	172	5,0	8	7,0	2,0	8,0	916	296	8,0
	eStream	5,0	1,0	1,0	1,0	468	90	1,0	8,0	3,0	1,0	4,0	1 120	168	8,0	12,0	6	2,0	12,0	2 660	304	8,0
9	estream.h	3	2,0	1,0	1,0	71	58	1,0	8,0	4,0	2,0	3,0	590	172	5,0	8	8,0	2,0	8,0	921	300	8,0
	eStream	4,0	1,0	2,0	1,0	451	59	1,0	8,0	3,0	2,0	5,0	1 120	172	8,0	8,0	8	3,0	9,0	2 644	300	8,0
10	estream.h	1	1,0	1,0	2,0	73	90	2,0	4,0	4,0	1,0	3,0	570	168	8,0	8	10,0	2,0	4,0	916	304	12,0
	eStream	4,0	1,0	1,0	2,0	459	89	1,0	12,0	3,0	2,0	4,0	1 140	164	5,0	12,0	10	3,0	8,0	2 656	304	12,0
Среднее значение	estream.h	2	1,3	1,2	1,3	70	86	1,2	5,6	3,8	1,5	3,2	580	175	6,5	9	8,1	2,3	5,4	940	302	9,4
	eStream	4,2	1,1	1,2	1,3	455	85	1,3	9,2	3,7	1,5	4,3	1 120	170	7,1	11,6	8	2,3	9,3	2 650	304	10,8
Погреш- ность	estream.h	1	0,7	0,8	0,7	5	5	0,8	2,4	1,2	0,5	1,2	10	9	1,5	3	1,9	0,7	2,6	40	8	3,4
	eStream	0,8	0,9	0,8	0,7	7	5	0,7	2,8	1,3	0,5	0,7	13	10	2,1	4,4	2	0,7	3,3	30	11	2,8

## ПРИЛОЖЕНИЕ 2

### Результаты измерений на объеме данных 10 МБ, мс

№ п/п теста	Процессор	Intel Quad Core i7 4710HQ с тактовой частотой 2,5 GHz							AMD Phenom Triple Core 8450 с тактовой частотой 2,1 GHz							Intel Atom Dual Core CPU N570 с тактовой частотой 1,66 GHz						
	Алгоритм	Salsa	Rabbit	HC128	Sosemanuk	Grain	Mickey	Trivium	Salsa	Rabbit	HC128	Sosemanuk	Grain	Mickey	Trivium	Salsa	Rabbit	HC128	Sosemanuk	Grain	Mickey	Trivium
	Проект																					
1	estream.h	37	18	9	13	1 460	1 830	23	76	48	28	44	11 920	3 500	72	160	140	48	68	17 860	5 960	140
	eStream	56	19	8	18	9 740	1 820	25	96	68	24	52	23 500	3 430	76	200	136	48	108	52 700	5 980	144
2	estream.h	41	18	8	11	1 480	1 800	23	84	64	20	36	11 980	3 500	72	160	136	44	72	17 860	5 970	132
	eStream	60	18	9	18	9 750	1 810	24	104	52	16	56	23 560	3 420	80	200	136	44	100	52 600	5 960	136
3	estream.h	41	17	8	13	1 460	1 850	23	76	64	16	48	11 960	3 520	68	164	132	48	72	17 810	5 960	140
	eStream	59	18	9	22	9 700	1 810	24	100	48	24	48	23 460	3 450	76	204	128	48	100	52 500	5 970	144
4	estream.h	45	17	9	13	1 450	1 800	23	76	48	16	40	11 990	3 490	68	160	136	48	64	17 810	5 950	144
	eStream	56	19	8	22	9 740	1 780	24	88	48	28	52	23 440	3 440	72	196	128	52	104	52 600	5 960	140
5	estream.h	44	18	8	13	1 440	1 830	19	76	64	28	48	11 910	3 500	64	164	140	52	72	17 810	5 950	140
	eStream	59	18	8	22	9 750	1 800	20	112	52	28	52	23 430	3 430	88	196	132	44	100	52 600	5 970	144
6	estream.h	44	17	8	13	1 450	1 830	24	80	52	20	44	11 900	3 490	72	168	140	48	72	17 800	5 940	136
	eStream	60	19	9	18	9 730	1 810	25	100	64	20	56	23 500	3 410	72	200	136	48	100	52 600	5 970	148
7	estream.h	45	18	8	13	1 450	1 820	23	72	52	28	36	11 940	3 480	62	164	136	48	76	17 790	5 960	132
	eStream	59	18	8	22	9 700	1 810	24	96	52	24	52	23 470	3 420	76	192	136	44	100	52 500	5 970	144
8	estream.h	45	17	8	13	1 450	1 820	19	84	52	28	36	11 920	3 490	60	162	142	48	72	17 800	5 950	132
	eStream	56	14	8	22	9 750	1 810	20	100	64	28	48	23 470	3 420	72	200	136	48	112	52 600	5 970	140
9	estream.h	39	17	8	13	1 490	1 840	23	76	60	20	36	11 910	3 490	64	160	140	48	72	17 790	5 950	136
	eStream	59	19	8	18	9 750	1 800	20	96	68	28	52	23 460	3 430	88	200	132	44	104	52 500	5 970	144
10	estream.h	41	14	9	14	1 460	1 840	24	72	60	20	40	11 910	3 490	64	164	140	44	72	17 790	5 950	136
	eStream	63	18	10	22	9 710	1 780	21	100	48	20	60	23 490	3 420	72	200	132	44	112	52 500	5 950	140
Среднее значение	estream.h	42	17	8	13	1 460	1 830	22	77	56	22	41	11 930	3 400	67	163	138	48	71	17 810	5 950	137
	eStream	59	18	9	20	9 730	1 800	23	99	56	24	53	23 480	3 430	77	199	133	46	104	52 600	5 970	142
Погреш- ность	estream.h	5	3	1	2	30	30	3	7	8	6	7	60	20	7	5	6	4	7	40	20	7
	eStream	4	4	1	2	30	30	3	13	12	8	7	80	30	11	7	5	6	8	100	20	6

### ПРИЛОЖЕНИЕ 3

#### Результаты измерений на объеме данных 1 ГБ, мс

№ п/п теста	Процессор	Intel Quad Core i7 4710HQ с тактовой частотой 2,5 GHz					AMD Phenom Triple Core 8450 с тактовой частотой 2,1 GHz					Intel Atom Dual Core CPU N570 с тактовой частотой 1,66 GHz				
	Алгоритм	Salsa	Rabbit	HC128	Sosemanuk	Trivium	Salsa	Rabbit	HC128	Sosemanuk	Trivium	Salsa	Rabbit	HC128	Sosemanuk	Trivium
	Проект															
1	estream.h	4 090	1 540	660	1 100	2 120	6 580	4 880	1 620	2 950	5 790	17 180	14 620	4 600	7 250	14 110
	eStream	5 560	1 570	700	1 890	2 110	8 220	4 730	1 600	4 230	5 950	21 060	13 950	4 630	10 940	14 920
2	estream.h	4 100	1 580	670	1 100	2 110	6 540	4 880	1 620	2 930	5 790	17 200	14 610	4 600	7 270	14 110
	eStream	5 550	1 580	710	1 890	2 150	8 300	4 770	1 590	4 220	5 950	21 070	13 960	4 650	10 910	14 910
3	estream.h	4 120	1 560	670	1 110	2 100	6 530	4 870	1 630	2 950	5 770	17 200	14 620	4 590	7 260	14 110
	eStream	5 480	1 580	700	1 880	2 140	8 200	4 770	1 600	4 220	5 940	21 040	13 950	4 640	10 910	14 910
4	estream.h	4 080	1 580	660	1 100	2 090	6 530	4 880	1 620	2 940	5 800	17 210	14 610	4 600	7 240	14 110
	eStream	5 550	1 570	690	1 900	2 160	8 300	4 760	1 580	4 190	5 960	21 050	13 950	4 630	10 900	14 930
5	estream.h	4 120	1 580	680	1 120	2 110	6 530	4 860	1 610	2 950	5 790	17 190	14 610	4 600	7 240	14 100
	eStream	5 540	1 570	690	1 890	2 150	8 200	4 750	1 590	4 200	5 990	21 060	13 970	4 640	10 900	14 940
6	estream.h	4 150	1 560	680	1 100	2 080	6 500	4 870	1 620	2 940	5 790	17 190	14 590	4 600	7 250	14 110
	eStream	5 480	1 600	690	1 880	2 130	8 300	4 760	1 580	4 220	5 940	21 010	13 950	4 640	10 890	14 910
7	estream.h	4 080	1 560	670	1 090	2 120	6 560	4 870	1 620	2 970	5 790	17 220	14 570	4 600	7 240	14 100
	eStream	5 530	1 580	690	1 890	2 160	8 300	4 740	1 580	4 220	5 940	21 010	13 970	4 640	10 910	14 930
8	estream.h	4 070	1 570	660	1 100	2 120	6 550	4 900	1 620	2 930	5 800	17 210	14 580	4 600	7 250	14 100
	eStream	5 440	1 550	680	1 900	2 150	8 200	4 720	1 590	4 190	5 950	21 040	13 960	4 640	10 930	14 910
9	estream.h	4 150	1 570	670	1 090	2 110	6 540	4 890	1 630	2 930	5 780	17 210	14 600	4 580	7 250	14 110
	eStream	5 500	1 550	690	1 880	2 170	8 220	4 760	1 580	4 210	5 980	21 020	13 970	4 630	10 930	14 930
10	estream.h	4 130	1 580	680	1 100	2 110	6 560	4 860	1 630	2 920	5 800	17 190	14 590	4 620	7 250	14 120
	eStream	5 540	1 600	690	1 850	2 130	8 290	4 750	1 600	4 220	5 950	21 050	13 960	4 640	10 950	14 910
Среднее значение	estream.h	4 110	1 570	670	1 100	2 110	6 540	4 880	1 620	2 940	5 790	17 200	14 600	4 600	7 250	14 110
	eStream	5 520	1 580	690	1 890	2 150	8 250	4 750	1 590	4 210	5 950	21 040	13 960	4 640	10 920	14 920
Погреш- ность	estream.h	40	20	10	20	30	40	20	10	30	20	20	30	30	20	20
	eStream	40	30	20	20	30	50	30	20	20	40	30	20	20	30	20

## **ПРИЛОЖЕНИЕ 4**

### **Список приложений на электронном носителе**

1. Приложение 5. Исходные коды реализаций алгоритмов проекта eStream, предложенные авторами шифров;
2. Приложение 6. Исходные коды библиотеки estream.h;
3. Приложение 7. Исходный код программы для тестирования алгоритмов с помощью тестовых векторов estream\_test\_vectors;
4. Приложение 8. Исходные коды программы для шифрования файлов escrypt;
5. Приложение 9. Исходные коды клиент-серверного приложения;
6. Приложение 10. Видеозапись с демонстрацией возможностей библиотеки estream.h на примере работы клиент-серверного приложения.