**BC/NW 2018 № 2 (33):11.2**

**ОПТИМИЗАЦИЯ АЛГОРИТМОВ СИММЕТРИЧНОГО БЛОЧНОГО ШИФРОВАНИЯ С ПРИМЕНЕНИЕМ ГРАФИЧЕСКОГО УСКОРИТЕЛЯ**

Прокофьев М.А.

**Введение**

В современном мире информация играет важную роль в жизни человека. Ежедневно в мире обрабатываются терабайты данных, хранение которых является довольно непростой задачей. Большая часть таких данных не предназначена для публичного доступа и требует применения различных алгоритмов шифрования, для исключения утечки конфиденциальной информации.

Среди всех применяемых в настоящее время разновидностей алгоритмов шифрования особую роль играют симметричные блочные шифры. Сходство процедур шифрования и дешифрования, высокая степень обеспечения защиты и производительности, например при работе с большими объемами данных, обеспечили блочному шифрованию большую популярность и широкое распространение на рынке криптографического прикладного обеспечения.

Как и любая другая информационная технология, блочное шифрование имеет несколько стандартов, разрабатываемых международными институтами. В качестве их примеров, в данной работе приводятся американский стандарт *AES* и российский ГОСТ, каждый из которых имеет свои достоинства и недостатки.

Большинство ПО, использующего данные алгоритмы, практически не использует потенциал современного оборудования, такого, как например различные графические ускорители, за счет чего процесс шифрования может занимать продолжительное время. В связи с этим актуальной задачей становится разработка специальных библиотек, с использованием технологий параллельного программирования, в частности *OpenCL*, позволяющих решить поставленную выше проблему.

Цельюисследования является анализ описанных выше алгоритмов, их реализация в классическом и многопоточном варианте и тестирование полученного в ходе разработки программного продукта

**1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

**Основные понятия**

***Симметричные алгоритмы*** шифрования (или криптография с секретными ключами) основаны на том, что отправитель и получатель информации используют один и тот же ключ. Этот ключ должен храниться в тайне и передаваться способом, исключающим его перехват. С одной стороны, это является преимуществом, т. к. становится возможным защищенная двусторонняя передача информации, но с другой стороны возникает проблема: получатель на основании наличия зашифрованного и расшифрованного сообщения не может доказать, что он получил это сообщение от конкретного отправителя, поскольку такое же сообщение он мог сгенерировать самостоятельно.

Несмотря на это, криптография с симметричными ключами считается **стойкой**, что делает практически невозможным процесс дешифрования без знания ключа. При прочих равных условиях стойкость определяется длиной ключа. Но тем не менее, при использовании симметричных алгоритмов шифрования сами механизмы для распределения ключей должны обладать высокой надежностью. [1]

Процесс обмена информацией осуществляется в 3 этапа:

− отправитель передает получателю ключ (в случае сети с несколькими абонентами у каждой пары абонентов должен быть свой ключ, отличный от ключей других пар);

− отправитель, используя ключ, зашифровывает сообщение, которое пересылается получателю;

− получатель получает сообщение и расшифровывает его.

ёСтоит отметить, что в таких алгоритмах достигается высокая скорость шифрования больших объемов данных за счет использования ключей не очень большой длины. Это положительно сказывается на общей производительности высоконагруженных систем, например в банковской сфере.

Таким образом, при соблюдении всех условий тайной передачи ключа шифрования и корректной реализации соответствующих алгоритмов, симметричное шифрование остаётся самым актуальным и криптографически гарантированным методом защиты информации.

**Обзор алгоритмов симметричного шифрования**

***Advanced Encryption Standard***

В настоящее время в США используется принятый и стандартизированный правительством алгоритм *Advanced Encryption Standard (AES)*. На данный момент он является одним из самых распространенных алгоритмов симметричного шифрования. Он оперирует блоком в 128 бит и ключом размера [128…256] бит.

Шифрование состоит из следующих функций преобразования:

1. *ExpandKey* — функция для вычисления всех **раундовых ключей** – ключей, используемых на каждой итерации преобразования над специальной матрицей байтов, расположенных в определенном порядке (**формой**);

2. *SubBytes* — функция для подстановки байтов, использующую **таблицу подстановок**, задающую отображение одного байта в другой;

3. *ShiftRows* — функция, обеспечивающая циклический сдвиг в форме на различные величины;

4. *MixColumns* — функция, которая смешивает данные внутри каждого столбца формы;

5. *AddRoundKey* — функция сложения ключа раунда с формой посредством поразрядного *XOR*.

Рассмотрим процесс шифрования подробнее [3]. В его начале, входные данные разбиваются на блоки размером 16 байт или 128 бит. Если полный размер данных не кратен 16 байтам ⎯ данные дополняются до размера кратного 16 байтам

Блок данных в алгоритме *AES* называется *state* и обычно представляется в виде матрицы 44. Операция шифрования каждого блока данных проводится независимо от содержимого других блоков. По окончанию шифрования блока ⎯ матрица заполняется следующей порцией данных и процесс повторяется. В силу независимости шифрования одного блока от другого процесс шифрования хорошо поддается распараллеливанию. Схема данного процесса представлена на рис. 1.1.



Рис.1.1 Схема шифрования стандарта *AES*

Функция расширения ключей ***KeyExpansion*** предназначена для создания набора раундовых ключей. Расширенный ключ *W* содержит 4\*(10+1) слов ⎯ начальный ключ в 4 слова и по 4 слова расширенного ключа на каждый из 10 раундов. Расширенный ключ *W* состоит из слов (четыре байта на слово), обозначаемых ниже, как *wi*, где *i* находится в диапазоне [0..44]. Полная длина расширенного ключа 1408 бит, по 128 бит на каждый раунд.

В процессе расширения ключа используется массив констант *Rcon*. Значения элементов данного массива определены так:

*Rcon1* = 1;

*Rcon*k = 2 \* *Rconk*-1 = 2k-1, для k = 2, 3,255;

*Rcon*k = 0, для *k* = 256, 257, 258;

Сам процесс расширения ключа представляет собой следующую последовательность операций:

1) Четыре слова ключа шифрования К копируются в первые четыре слова расширенного ключа *W*: *wi = ki* для *i* = 0, 1, 2, 3

2) Остальные слова расширенного ключа W для *i* = 4, 5, 44 генерируются так:

− если i кратно 4, то wi = SubBytes(RotByte(wi-1))  Rcon(i/4);

− если *i* не кратно 4, то wi = wi-4  wi-1.

Преобразование ***SubBytes*** – это нелинейная замена байт, которая проводится над каждым байтом *state*, используя фиксированную таблицу замены *S-Box*. Цель применения таблицы замен ⎯ затруднить линейный и дифференциальный криптоанализ.

В преобразовании ***MixColumns*** столбцы состояния (*state*) рассматриваются как полиномы над полем *GF(28)* и умножаются по модулю x4 + 1 на постоянный полином: 𝑎(𝑥)=3𝑥3+𝑥2+𝑥+2

Процесс умножения полиномов эквивалентен матричному умножению:

где *с* – номер столбца массива *state* и 0  *с*  3. Такое преобразование применяется к каждому из четырех столбцов *state*. [4]

Расшифрование данных происходит аналогичным образом, только в обратном порядке.

**Преимущества** алгоритма *AES* заключаются в том, что он не подвержен многим видам криптоаналитических атак, таких как: дифференциальный и линейный криптоанализ, метод интерполяции и другие, а также в высоком быстродействии на различных платформах.

В качестве его **недостатков** стоит выделить то, что каждый блок шифруется одним и тем же способом, а также то, что в основе алгоритма лежит простое математическое описание, вызывающее определенную долю скептицизма с точки зрения применения его в области секретных документов.

***ГОСТ Р 34.12-2015 (Кузнечик)***

В России в 2015 году был принят новый стандарт шифрования данных, пришедший на замену ГОСТ 28147-89. В отличие от своего предшественника, утратившего в данный момент свою актуальность по причине малого размера ключей и низкой защищенности, в основе нового стандарта лежит т.н. ***SP*-сеть**.

Шифр на основе *SP*-сети получает на вход блок и ключ и совершает несколько последовательных раундов, состоящих из чередующихся стадий подстановки и перестановки. Для достижения безопасности достаточно одного S-блока, но такой блок будет требовать большого объёма памяти. Поэтому используются маленькие *S*-блоки, смешанные с *P*-блоками.

Такая структура иногда также называется AES-*like*, однако, в отличие от последнего у «Кузнечика» есть ряд своих особенностей:

− линейное преобразование может быть реализовано с помощью регистра сдвига;

− ключевая развертка реализуется с помощью сети Фейстеля, в которой в качестве функции используется раундовое преобразование исходного алгоритма. [2]

Сам процесс шифрования основан на последовательном применении нескольких однотипных раундов, каждый из которых содержит три преобразования: сложение с раундовым ключом ***X***, преобразование блоком подстановок ***S*** и линейное преобразование ***L***.

Схема алгоритма шифрования одного 128-битного входного блока представлена на рис 1.2.



Рис.1.2. Схема шифрования одного блока стандарта ГОСТ 34.12.12-2015

**Преимущества** алгоритма «Кузнечик» заключаются в схожести с алгоритмом AES с точки зрения безопасности, меньшем количестве итераций по сравнению с предыдущий версией стандарта 89-го года, а также использовании последних наработок в области криптографии.

С другой стороны, в качестве его **недостатков** стоит отметить, что скорость работы данного алгоритма ниже, чем у его американского аналога, по причине наличия в современных центральных процессорах специальных команд, направленных на эффективное выполнение команд шифрования, связанных с алгоритмом *AES*.

**Обзор стандарта *Open***

***OpenCL*** (*Open Computing Language)* - это открытый, свободно распространяемый стандарт для кросс-платформенного программирования различных типов процессоров, применяемых в персональных компьютерах

серверах, мобильных устройствах и встраиваемых системах. *OpenCL* значительно увеличивает скорость выполнения вычислений для широкого спектра приложений различных категорий: от игровых и развлекательных продуктов, до научных и профессиональных решений, применяемых в производстве.

Рассмотрим модель платформы OpenCL [6]. Центральным элементом данной модели выступает понятие хоста (*host*) - первичного устройства, которое управляет *OpenCL*-вычислениями и осуществляет все взаимодействия с пользователем. Хост всегда представлен в единственном экземпляре, в то время как *OpenCL*-устройства (*devices*), на которых выполняются *OpenCL*-инструкции могут быть представлены во множественном числе. Каждое такое устройство в свою очередь состоит из вычислительных блоков (*Compute Unit*), которые далее разделяются на один или более элементы-обработчики (*Processing Elements*). На рис. 1.3 схематически изображена платформа, состоящая из 3-х устройств.



Рис.1.3 Схематическое представление OpenCL-платформы

Приложение *OpenCL* состоит из двух отдельных частей: программа хоста и одного или нескольких программных ядер. Эти ядра выполняются на *OpenCL* устройствах и исполняют основную работу приложения. 13

Хост выдает специальную команду для того, чтобы представить ядро для выполнения на устройстве OpenCL. Когда эта команда выдается хостом, исполняющая система OpenCL создает пространство индексов. Экземпляр ядра выполняется для каждой точки в этом индексном пространстве.

Каждый экземпляр ядра, которое называют рабочим элементом, запускается в конкретных определенных для него координатах в пространстве индексов. Эти координаты являются глобальными идентификаторами рабочих элементов. Рабочие элементы в свою очередь организовывают рабочие группы (*workgroup*), в рамках которой они выполняются параллельно на обрабатывающих элементах одного вычислительного модуля.

Разработка приложений с применением стандарта *OpenCL* обычно ведется на языках *C/C++ или Java*, что обеспечивает его кроссплатформенность и широкое распространение среди разработчиков высокопроизводительных систем.

**Экспериментальная часть**

Тестирование работоспособности ПО

Тестирование проводилось в соответствии с ГОСТ 19.301-79. Объектом испытаний является разработанное в рамках данного курсового проекта программная библиотека «*BlockCipher*». Цели испытаний**:**

− демонстрация работоспособности библиотеки;

− проверка на соответствие требованиям, установленным в задании;

− выявление ситуаций, в которых поведение библиотеки является неправильным.

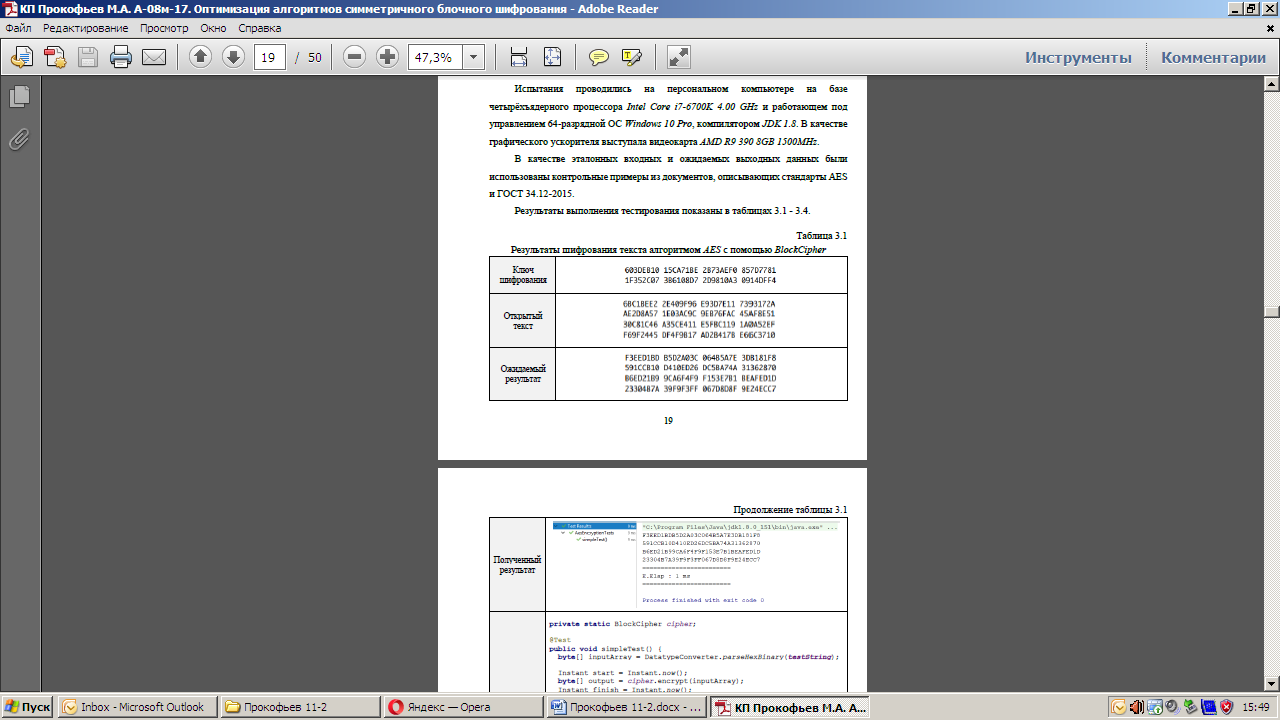
Испытания проводились на персональном компьютере на базе четырёхъядерного процессора *Intel Core i7-6700K 4.00 GHz* и работающем под управлением 64-разрядной ОС *Windows 10 Pro*, компилятором *JDK 1.8*. В качестве графического ускорителя выступала видеокарта *AMD R9 390 8GB 1500MHz*.

В качестве эталонных входных и ожидаемых выходных данных были использованы контрольные примеры из документов, описывающих стандарты AES и ГОСТ 34.12-2015.

Результаты выполнения тестирования показаны в таблицах 3.1 - 3.4.

Таблица 3.1

Результаты шифрования текста алгоритмом *AES* с помощью *BlockCipher*



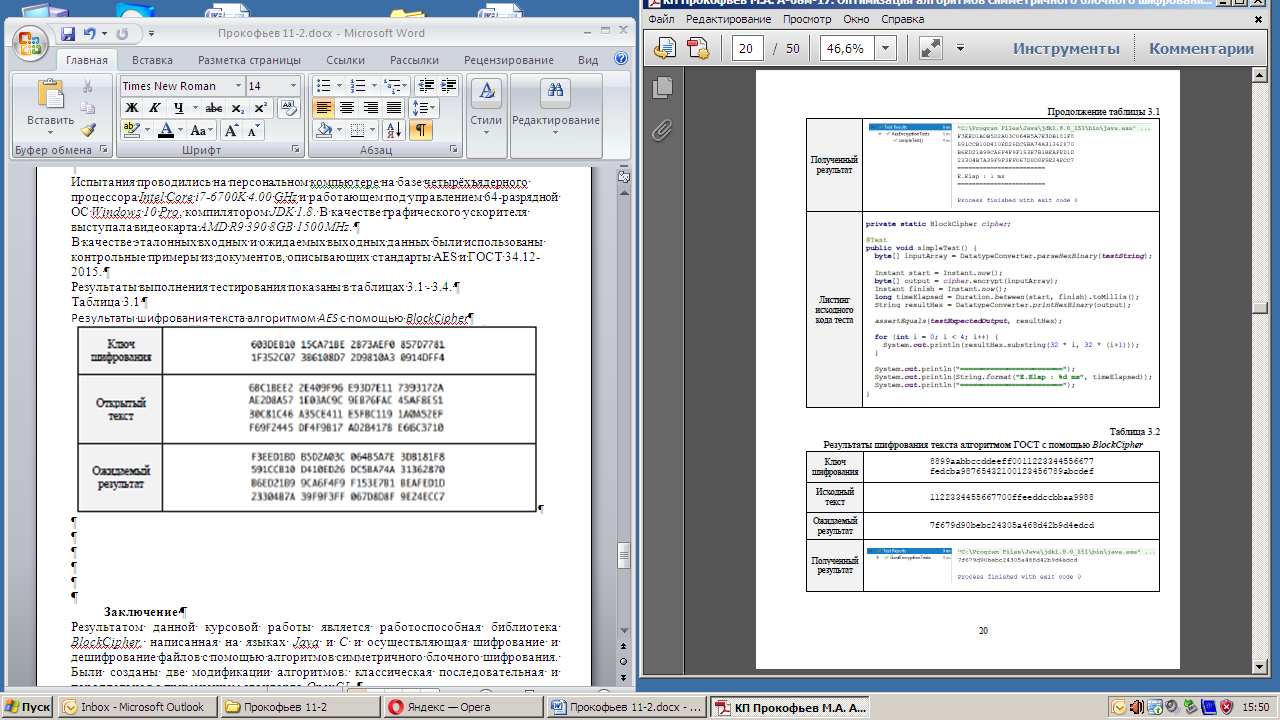
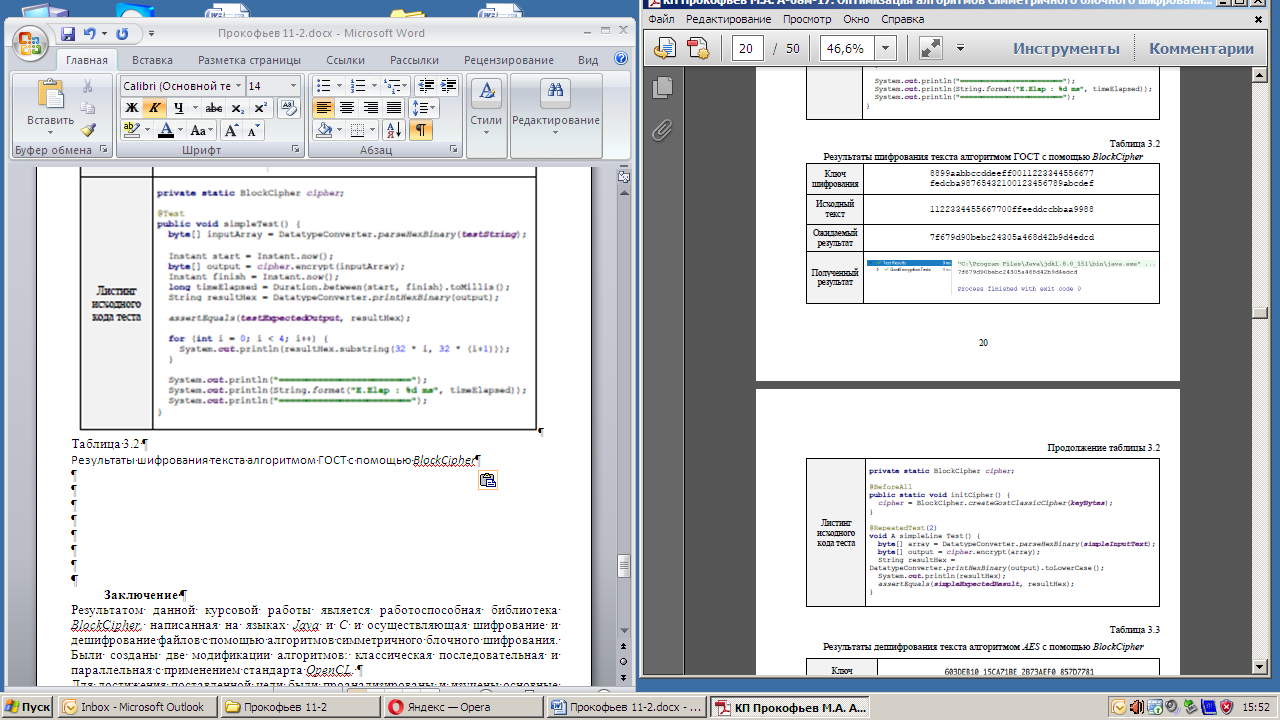
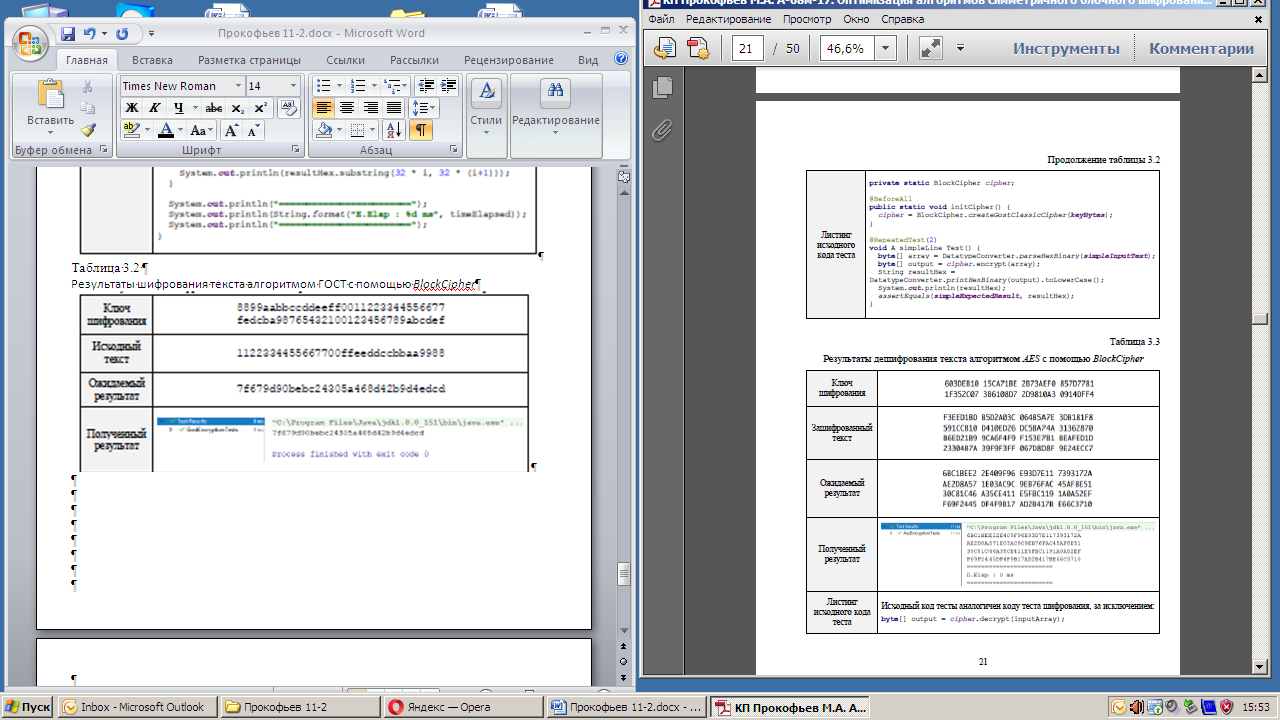
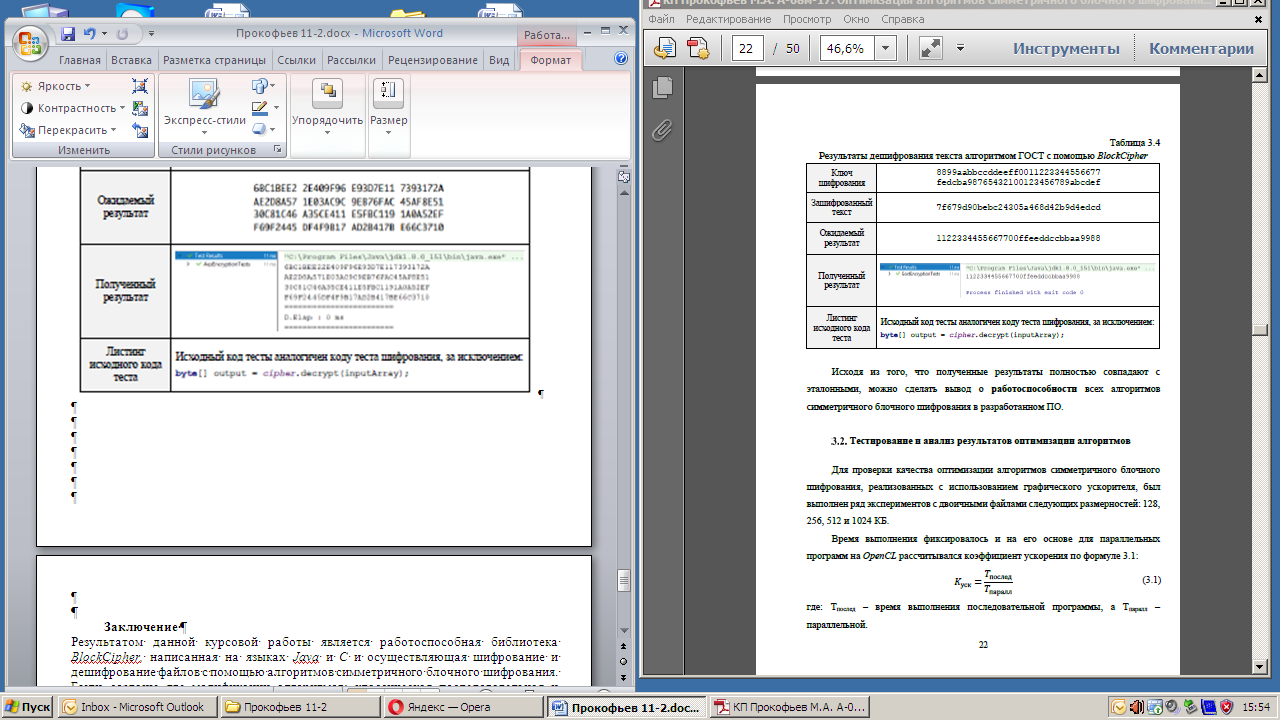


Таблица 3.2

Результаты шифрования текста алгоритмом ГОСТ с помощью *BlockCipher*







Исходя из того, что полученные результаты полностью совпадают с эталонными, можно сделать вывод о **работоспособности** всех алгоритмов симметричного блочного шифрования в разработанном ПО.

**Тестирование и анализ результатов оптимизации алгоритмов**

Для проверки качества оптимизации алгоритмов симметричного блочного шифрования, реализованных с использованием графического ускорителя, был выполнен ряд экспериментов с двоичными файлами следующих размерностей: 128, 256, 512 и 1024 КБ.

Время выполнения фиксировалось и на его основе для параллельных программ на *OpenCL* рассчитывался коэффициент ускорения по формуле 3.1:

(3.1

𝐾уск=𝑇послед𝑇паралл

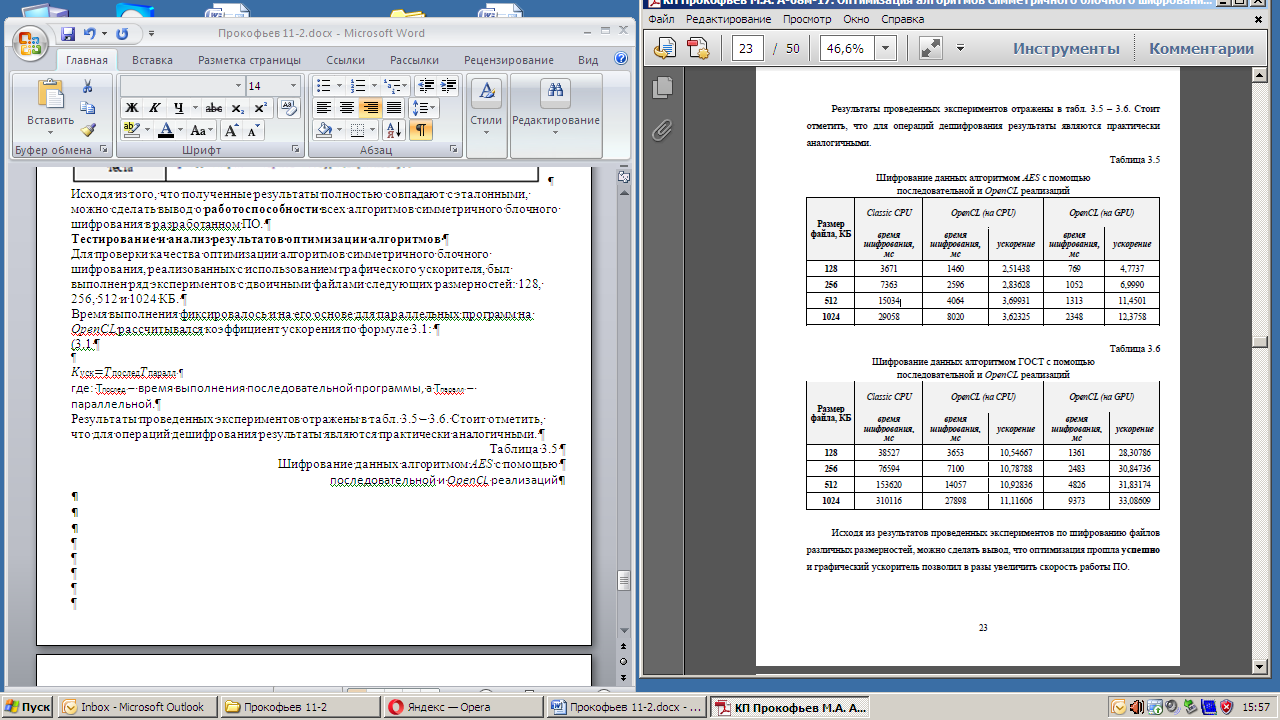
где: Tпослед – время выполнения последовательной программы, а Tпаралл – параллельной.

Результаты проведенных экспериментов отражены в табл. 3.5 – 3.6. Стоит отметить, что для операций дешифрования результаты являются практически аналогичными.

Таблица 3.5

Шифрование данных алгоритмом *AES* с помощью

последовательной и *OpenCL* реализаций



Исходя из результатов проведенных экспериментов по шифрованию файлов различных размерностей, можно сделать вывод, что оптимизация прошла **успешно** и графический ускоритель позволил в разы увеличить скорость работы ПО.

**Заключение**

Результатом данной курсовой работы является работоспособная библиотека *BlockCipher,* написанная на языках *Java* и *C* и осуществляющая шифрование и дешифрование файлов с помощью алгоритмов симметричного блочного шифрования. Были созданы две модификации алгоритмов: классическая последовательная и параллельная с применением стандарта *OpenCL*.

Для достижения поставленной цели были проанализированы и изучены основные технологии и решения, используемые при параллельном программировании на графических ускорителях, а также различные подходы к современному шифрованию файлов.

Было проведено тестирование на ЭВМ под управлением центрального процессора *Intel Core i7 6700K* и графического ускорителя *AMD R9 390*, показавшее готовность библиотеки к применению в различных криптографических системах.

**Литература**

1. Современные алгоритмы блочного шифрования и методы их анализа. Л. К. Бабенко, Е. А. Ищукова. – М.: Гелиос АРВ, 2006. – 376 с.

2. ГОСТ Р 34.12—2015. Информационная технология. Криптографическая защита информации. Блочные шифры. — М.: Стандартинформ, 2015.— 6 с.

3. Advanced Encryption Standard (AES): FIPS Publication 197. – Springfield: National Technical Information Service, 2001. – 47 p.

4. Лекция 2: Алгоритм AES. Режимы выполнения алгоритмов симметричного шифрования. Создание случайных чисел. // Национальный открытый университет «ИНТУИТ» [Электронный ресурс] URL: https://www.intuit.ru/studies/courses/14248/1285/lecture/24208

5. В ГОСТе сидел «Кузнечик» // Хабр [Электронный ресурс]. URL: https://habr.com/post/266359/ (дата обращения: 23.11.2018).

6. OpenCL Overview [Электронный ресурс]. URL: https://www.khronos.org/opencl/ (дата обращения: 09.11.2018).

7. LWJGL - Lightweight Java Game Library [Электронный ресурс] URL: https://www.lwjgl.org/ (дата обращения: 10.12.2018)