**Направление:** Защита информации

**Kейс:** Исследование актуальных криптоалгоритмов

**Организация-работодатель, представившая кейс:** Лаборатория Касперского

**Отчет по проекту**

**Название проекта:** Studio Lab / Security Studio / Security Lab

**Авторы проекта**

Учащиеся ГБОУ лицея № 1581

Состав команды: Калашников Дмитрий Павлович, 9 «Г»

Пронин Арсений Сергеевич, 9 «Г»

Заварзина Яна Игоревна, 9 «Г»

Куратор команды: Камалин Алексей Викторович

Москва, 2016

Аннотация

Проблема защиты информации, а, следовательно, и использования различных механизмов криптозащиты в последнее время требует всё большего внимания. Однако перед человеком, нуждающемся в особой защите своей информации, встает вопрос, каким методом воспользоваться для осуществления своей цели. В данном проекте было проведено исследование новейших криптоалгоритмов и функций хэширования, позволяющих хранить данные защищенными, с оценкой их криптостойкости и быстродействия, а также осуществлена реализация одного из криптоалгоритмов. На основе проведенного исследования получена сравнительная характеристика новейших алгоритмов шифрования и хэш-функций . Результаты исследования помогают выбрать между криптостойкостью и скоростью работы, оценить наиболее оптимальные способы защиты данных, а представленная реализация может служить в качестве инструмента для шифрования важной информации.

Введение.

Зачем же нужно шифрование данных?

На данный момент различные компании имеют дело с большим объёмом данных, касающихся, например, финансовой информации, информации о конкурентах и др. Эта информация, зачастую находясь на различных носителях, может быть подвержена определённым рискам:

1. Риск потери информации. Порча носителя или утеря его пользователем.
2. Риск вирусной кражи информации по вине пользователя. Избежать данной проблемы помогают антивирусные системы. При этом, благодаря шифрованию, при краже данных, злоумышленник не сможет ими воспользоваться.
3. Риск кражи информации с носителей, промышленный шпионаж. Аналогично, благодаря шифрованию, тот, кому в руки попадут данные, не сможет ими воспользоваться.

Исходя из этого, в наше время шифрование информации крайне актуально и пользуется широким применением.

В ходе исследования производился анализ наиболее популярных и современных алгоритмов шифрования и функций хэширования, использованы надежные и свежие источники, все анализы и теоретические расчеты производились с точки зрения современного оборудования и ЭВМ.

Проблема защиты данных крайне важна в наши дни. Участники нашей группы осознают это, что и вызвало интерес к исследованию решений рассматриваемой проблемы.

Люди нуждаются в быстрой передачи данных, порой очень важных и даже секретных. Но в связи с развитием технологий злоумышленнику иногда не составляет большого труда перехватить передаваемые данные или просто получить к ним открытый доступ. В связи с данной проблемой возникает необходимость использовать различные способы защиты информации. Итак,

Цель проекта:

*Провести исследование актуальных криптоалгоритмов.*

В результате ожидалось получить определенные статистические данные и сравнительную характеристику, определяющую актуальные на данный момент криптоалгоритмы. Также, в результате мы должны были получить самостоятельную реализацию одного из алгоритмов на языке программирования C++.

В процессе исследования от нас требовалось проанализировать принцип действия современных симметричных блочных и поточных алгоритмов шифрования и функций хэширования, оценить их криптографическую стойкость и производительность, сравнить данные характеристики и в итоге реализовать один из рассмотренных алгоритмов на известном нам языке программирования .

Этап 1.

Рассмотрение нескольких новейших алгоритмов симметричного блочного и поточного шифрования. Оценка их криптостойкости и быстродействия.

Итак, перед началом первого этапа нам было необходимо ознакомиться с базовыми понятиями, такими, как, собственно, криптография, криптоалгоритмы и их классификации и др.

Затем мы приступили с исследованию новейших криптоалгоритмов. Нами были выбраны такие алгоритмы, как RC4, ГОСТ 28147-89 и AES. Анализ алгоритмов мог быть выполнен с помощью трех источников:

1. Литература.

«+» :

достоверность;

большой объём нужной информации

«-»:

отсутствие быстрого доступа в случае, когда нет электронной версии

1. Интернет-ресурсы.

«+»:

быстрый доступ;

различная степень сложности подачи информации;

большой объём информации

«-»:

относительная достоверность;

1. Специалисты в данной сфере

«+»:

достоверность;

возможность консультации;

оптимальная подача информации;

«-»:

труднодоступность.

В следствии ощутимого минуса п.3 – труднодоступности специалистов, было решено в основном использовать научную литературу и интернет-ресурсы, указанные ниже в источниках.

Анализ каждого из алгоритмов производился в 3 этапа:

1. Сбор теоретических сведений.

Используя различные источники была собрана информация об алгоритмах.

Также теоретические сведения включают принцип работы каждого из алгоритмов.

1. Оценка теоретической производительности

производилась на основе анализа алгоритмов и их реализаций, а также расчетов, связанных с оценкой вычислительной сложности алгоритмов.

1. Оценка криптостойкости

производилась на основе сбора материалов о наиболее известных атаках на криптоалгоритмы, оценки вычислительной сложности полного перебора ключей, а также методом выявления слабостей каждого из криптоалгоритомов.

В итоге нами была получена сравнительная характеристика, статистические данные и несколько диаграмм, демонстрирующих основные показатели алгоритмов, таких, как, например, производительность алгоритмов и функций хэширования.

В процессе работы перед нами возник вопрос - использовать готовые, проверенные данные(например, о производительности криптоалгоритмов), или произвести собственные расчеты на основе имеющихся ресурсов и знаний. В итоге было решено, что самостоятельные, *обоснованные* расчеты скорее всего окажутся полезнее и достовернее тех, что взяты из различных источников.

Итак, начнем с теоретической части.

Криптология – наука, составляемая криптографией в совокупности с криптоанализом.

Существует несколько классификаций криптоалгоритмов, основанных на определённых признаках.

Криптография с ключом характеризуется действиями, производимыми над передаваемыми данными, которые зависят от определённого ключа. Криптография с ключом подразделяется на:

1. Симметричные криптоалгоритмы.  
   характеризуются тем, что для зашифровки и расшифровки передаваемого сообщения используется один и тот же ключ.
2. Асимметричные криптоалгоритмы  
   характеризуются тем, что для зашифровки передаваемого сообщения отправитель имеет открытый ключ, доступный другим лицам, а для расшифровки получатель использует закрытый ключ, известный только ему.

В зависимости от памяти, которую занимает информация, криптоалгоритмы подразделяются на:

1. Поточные шифры.  
   Единицей кодирования является один бит. Результат кодирования не зависит от прошедшего ранее входного потока.
2. Блочные шифры  
   Единицей кодирования является блок из нескольких байтов. Результат кодирования зависит от всех исходных байтов этого блока.

Рассмотрим несколько наиболее известных современных криптоалгоритмов.

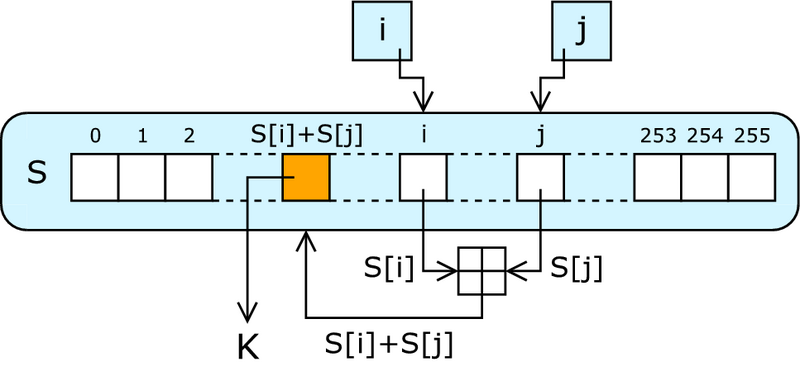
**RC4**

* + Принцип работы

Криптоалгоритм RC4 широко известен своим высоким быстродействием и достаточно низкой криптостойкостью. Проанализируем одну из реализаций данного алгоритма.

Входными данными является текст и ключ. Для алгоритма RC4 ключ может быть от 8 до 2048 бит, но обычно используется диапазон 40 — 256 бит. Каждый байт подаваемой на вход информации суммируется по модулю 2 с преобразованным S-блоком.

На картинке изображен принцип работы RC4.



* + Быстродействие и константный расход памяти.

Среди особенностей алгоритма и его реализации можно выделить:

* заведомо придется выделять память для 2 массивов под S-блок размером 256 типа *byte.* Т.е. 512 байт. Также нам нужно будет где-то хранить поступающий со входа текст размером **M**(байт). В подсчете памяти, расходуемой алгоритмом, *не будем* учитывать память, выделяющуюся конкретно при реализации алгоритма, т.к. единой реализации не существует.
* Вычислительная сложность алгоритма будет складываться из вычислительной сложности функций, используемых при реализации данного алгоритма. Примерный подсчет операций приведен в таблице:

|  |  |  |  |
| --- | --- | --- | --- |
| ***функция*** | **условия** | **операции(внутри ф-ии)** | **кол-во вызовов** |
| *обмена* | - | 1 | 512+m/8 |
| *инициализации s-блока* | - | 512 | 1 |
| *генерирование ПС символа* | - | 1 | m/8 |
| *(де)шифрование* | - | 1 | 1 |
| *итого операций* | **512+m/8** | | |
| *используемая память*  *(байт)* | 512+M | | |

* *Назовем высокоуровневыми операциями возведение числа в степень, логарифм и другие операции, основанные на зацикленном повторении простых арифметических действий. В данном анализе и далее игнорируются невысокоуровневые операции. Функции, выполняющие операции вне цикла, будем считать за O(1)*
* *m - размер текста в битах*

Алгоритм требует 512 МБ теоретической памяти. Как можно увидеть далее, это приемлемо по сравнению с другими известными криптоалгоритмами.

Современные ЭВМ способны совершать ~10^8 сложных арифметических операций в секунду. Исходя из этого здесь и далее и будем рассчитывать производительность криптоалгоритма *для одного компьютера*.

Благодаря таблице можем заключить, что вычислительная сложность RC4 составляет **O(m/8),** где **m** - размер текста в битах. Так что теоретическая производительность алгоритма будет рассчитываться следующим образом:

m=10^8\*8 => P=8\*10^8 (бит/с) ~ ***95,367 (МБ/с)***.

К примеру, все альбомы группы the Beatles размером 5 ГБ будут зашифрованы/расшифрованы примерно за *54* секунды.

При расшифровке используются аналогичные операции, поэтому скорость дешифрования можно считать равной скорости шифрования.

* + Криптостойкость

В таблице приведены наиболее известные атаки на алгоритм RC4:

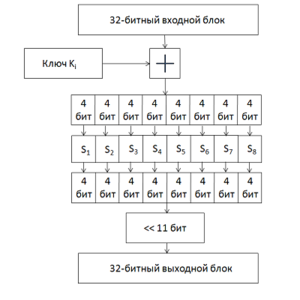
|  |  |  |
| --- | --- | --- |
| **Название атаки** | **Принцип** | **Количество операций при ключе 2048 бит, размер текста 4 гб** |
| Исследования Руза и восстановление ключа из перестановки | 1-й байт ключевого потока взаимосвязан с первыми 3 байтами ключа , а первые байты перестановки после алгоритма расписания ключей взаимосвязаны с комбинацией байт ключа. | 2^1024 |
| Атака Флурера, Мантина и Шамира | первые байты всех возможных ключей выбраны неслучайным образом. Зная первое слово K и m байтов ключа, можно получить m + 1 байт ключа. | 2^8 |
| Атака Кляйна | взаимосвязь ключа и ключевого потока алгоритма. Для применения нужно угадать первый ключевой байт. | O(256\*ln 256\*((k-1) +1))  2^6 |
| Комбинаторная проблема | Если из возможных 256 элементов известно x, следовательно, если предположить, что все остальные элементы нулевые, то максимальное количество элементов определенного алгоритма также равно x. Из шифротекста должна быть известна расшифровка хотя бы 1 символа. | 2^11 |

Как видно из таблицы, алгоритм крайне не устойчив к криптоанализу. В связи с этим RC4 обзавелся множеством модификаций, разработанных для повышения криптостойкости алгоритма.

**ГОСТ 28147-89**

ГОСТ 28147-89 (Магма) — современный российский стандарт [симметричного шифрования](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), отличающийся высокими показателями скорости и криптографической стойкости.

* + Принцип работы.

Проанализируем данный криптоалгоритм в режиме простой замены. Последовательность действий:

1. Ввод исходных данных для основного шага криптопреобразования. N – вводимый 64-битовый блок данных, который в ходе выполнения разбивается на две части N1 и N2, каждая из которых составляет 32 бита. Следовательно, N=(N1,N2). X – элемент ключа, составляющий 32 бита.
2. Сложения первой части (N1) с элементом ключа по модулю 2^32. Результат передаётся на шаг 2.
3. 32-битовое значение, полученное на шаге 1, интерпретируется как массив из восьми 4-битовых блоков кода. Следовательно, S = (S0,S1,S2,S3,S4,S5,S6,S7). Значение каждого блока заменяется новым, которое выбирается по таблице замен. В качестве замены для значения блока выбирается элемент из таблицы замен с номером строки, равным номеру заменяемого блока, и номером столбца, равным значению заменяемого блока как 4-битового целого неотрицательного числа.
4. Результат шага 2 сдвигается на 11 бит влево, в сторону старших разрядов, и передается на шаг 4. R11 - функция циклического сдвига своего аргумента на 11 бит в сторону старших разрядов.
5. Значение, полученное на шаге 3, побитово складывается по модулю 2 со второй частью (N2) исходного блока.
6. Первая часть преобразуемого блока занимает место второй, а на ее место помещается результат выполнения шага 4.
7. Т.к. N=(N1, N2), конечное значение исходного блока является результатом выполнения алгоритма основного шага криптопреобразования.

* Быстродействие и расход памяти.

Основную работу алгоритма будет выполнять главная функция, содержащая цикл считывания и преобразования блоков. Поэтому рассмотрим расход ресурсов и времени данной функции в одной из реализаций.

Алгоритм ГОСТ 28147-89 требует наличия 3 массивов:

1. таблица замен(двумерный) размером 8\*16 типа *byte,* т.е. 128 байт,
2. ключ размером 8 типа *long*, т.е. 8\*4=32 байт,
3. 32-разрядный накопитель, т.е. 1\*4=4 байт.

Итого, не считая переменных, создаваемых при реализации алгоритма, нам потребуется *164* байта заранее зарезервированной памяти. Стоит отметить, что данная цифра меньше, чем для расхода памяти алгоритма RC4.

|  |  |  |
| --- | --- | --- |
| Условия | Вычислительная сложность | Память(байты) |
| 20 | O(10m/64) | 164+M |

\*m - в битах

Основную вычислительную сложность алгоритма составляет цикл считывания и преобразования блоков, зависящий от длины текста: от 0 до m/64, где m - размер текста в битах. Внутри данной программы выполняется в общей сложности ~10 значащих итераций циклов. Поэтому вычислительная сложность алгоритма составит O((10m)/64):

((10^8)\*64) / 10 ~ **76,29 (Мб/с)**.

Т.е. скорость алгоритма ГОСТ 28147-89 составляет 76,29 МБ/с.

Как мы видим, это значительно ниже скорости работы алгоритма RC4.

Тем не менее, все альбомы the Beatles размером 5 ГБ будут зашифрованы/расшифрованы данным алгоритмом одним компьютером ~ за *1 минуту 7 секунд*, что на 13 секунд дольше времени шифрования алгоритмом RC4.

При шифровании и дешифровании производятся аналогичные действия, поэтому скорость расшифрования можно считать равной скорости шифрования.

* Криптостойкость

ГОСТ крайне устойчив к криптоанализу, при грамотной реализации единственный способ его взлома - полный перебор. Более оптимальный способ взлома возможен только при некоторых условиях, например, при генерации слабых ключей(где значительно преобладают 0 или 1), при определенных видах таблиц и при определенной реализации самого алгоритма. Также существуют теоретические решения, не осуществимые на практике из-за огромной вычислительной сложности. Вообще криптостойкость ГОСТа нельзя определить, не зная таблиц замен, но если сравнивать рассмотренные нами алгоритмы при идеальных условиях, то ГОСТ имеет преимущество перед ними в криптостойкости.

В отличие от криптоалгоритма AES, рассмотренного ниже, размер ключа которого может составлять 128,192 и 256 бит, в стандарте ГОСТ 28147-89 ключ имеет фиксированную длину 256 бит, что позволяет алгоритму чаще иметь одинаковую криптостойкость в любой реализации, а также влияет на скорость, которую нельзя повысить, выбрав меньшую длину ключа.

В таблице приведены наиболее известные атаки на алгоритм ГОСТ:

|  |  |  |  |
| --- | --- | --- | --- |
| Атака | Трудоемкость | Память | Требуемый материал |
| Исобе | **2224** | **264** | **232** |
| Динур-Данкельман-Шамир, FP, 2DMitM | **2192** | **236** | **264** |
| Динур-Данкельман-Шамир, FP, low-memory | **2204** | **219** | **264** |
| Динур-Данкельман-Шамир, Reflection, 2DMitM | **2224** | **236** | **232** |
| Динур-Данкельман-Шамир, Reflection, 2DMitM | **2236** | **219** | **232** |
| Полный перебор | **2256** | **1** | **4** |

\*трудоемкость - в операциях шифрования

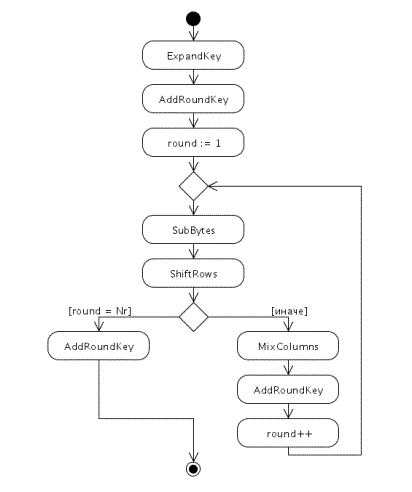
\*\*память, материал - в блоках алгоритма (по 64 бита)

**AES**

Advanced Encryption Standard (AES), также известный как Rijndael — симметричный алгоритм блочного шифрования, принятый в качестве стандарта шифрования правительством США. Этот алгоритм хорошо проанализирован и сейчас широко используется. По состоянию на 2009 год AES является одним из самых распространённых алгоритмов симметричного шифрования.

* Принцип работы

данного алгоритма представлена на блок-схеме:



Где:

*ExpandKey* — функция для вычисления всех раундовых ключей;

*SubBytes* — функция для подстановки байтов, использующая таблицу подстановок;

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

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

*AddRoundKey* — сложение ключа раунда с формой.

* Быстродействие и расход памяти.

Алгоритм AES требует наличия 4 массивов для шифрования(+для дешифрования, соответственно, также 4):

1. Прямая замена(одномерный) размером 256.
2. Расширенная прямая замена(двумерный) - 4\*256.
3. Расширенные ключи(2) - по 64

— все типа unsigned long int(по 4 байта).

Итого, не считая переменных, создаваемых при реализации алгоритма, нам потребуется 5632 байта заранее зарезервированной памяти, что намного больше требуемой памяти алгоритмом ГОСТ 28147-89 и RC4.

Алгоритм AES разбивает текст на блоки по 128 бит. Далее генерируются ключи и таблицы замен, потом в блоках происходят замены и логические операции с текстом - считывание блока и добавление раундового ключа + „раунды дешифрования” - в общей сложности для каждого блока потребуется ~2^5 операций. Таким образом,

2^5\*(m/2^7)=10^8

P=10^8\*2^7/(2^5\*2^23) ~ **47,683 Мб/с**,

что значительно ниже производительности двух рассмотренных выше алгоритмов. На скорость работы алгоритма в данной реализации также повлияет то, что тут таблицы замен генерировались, тогда как в ГОСТ 28147-89, например, использовались готовые таблицы замен.

Вычислительная сложность сложность дешифрования приблизительно равна вычислительной сложности шифрования, поэтому скорость дешифрования можно считать равной скорости шифрования.

Так что если потребуется зашифровать/расшифровать все творчество the Beatles размером 5 ГБ данным алгоритмом с помощью одного компьютера, на это уйдет *1 минута 47 секунд*, что на 53 секунды дольше времени шифрования алгоритмом RC4 и на 40 секунд дольше шифрования алгоритмом ГОСТ 28147-89.

* Криптостокость

Озвученные на конференции CRYPTO 2011 результаты дополнительного криптоанализа алгоритма блочного шифрования AES указывают на новый способ атаки, позволяющий в четыре раза сократить трудоёмкость выполнения операций по подбору секретного ключа. Иными словами на деле криптостойкость AES-128 сводится к AES-126, а AES-192 к AES-189, что само по себе остается достаточно внушительным показателем (для взлома AES-128

требуется выполнить 2 в 126 степени операций, что на 2.552 \* 10^38 операций меньше). Предложенный метод атаки работает со всеми вариантами AES. Возможность совершения атаки указанным методом признали создатели AES, Винсент Рэймен и Йоан Даймен.

Интерес представляет то, что это первый независящий от конкретной реализации подтвержденный способ атаки, указывающий на существование недоработок в алгоритме AES. Все ранее представленные методы сокращения количества операций для взлома носили теоретический характер и были опровергнуты при проверке. Новый способ атаки пока представляет собой только теоретический интерес, так как с практической точки зрения он не позволяет в обозримом будущем сформировать инфраструктуру для взлома реальных ключей:

чтобы перебрать 2^126 вариантов ключей на 100 компьютерах одинаковой мощности, потребуется:  
2^126 / 10^10 = 2.363 \* 10^24 часов,

так что — у AES по прежнему остается внушительный запас прочности.

Этап 2.

Исследование актуальных функций хэширования. Оценка производительности и криптостойкости.

Перед началом второго этапа нам надо было необходимо понять что такое хэш-функции, какие они бывают, чем отличаются, какие имеют признаки. После этого мы начали изучать наиболее известные из них:  
SHA-2, BMW и ГОСТ Р 34.11-2012.

Исследование было проведено по следующей схеме:

1. Изучение истории.  
   Кто, когда и как придумал функцию.
2. Анализ алгоритма.  
   То, как работает данная хэш-функция пошагово, как её реализовать на разных языках, альтернативы стандартной реализации.
3. Криптостойкость.  
   Самая сложная часть анализа. Производилось множество исследований по изучению криптостойкости разных хэш-функций, но определить криптографическую силу алгоритма непросто.

[Криптоанализ](https://ru.wikipedia.org/wiki/%25D0%259A%25D1%2580%25D0%25B8%25D0%25BF%25D1%2582%25D0%25BE%25D0%25B0%25D0%25BD%25D0%25B0%25D0%25BB%25D0%25B8%25D0%25B7) хэш-функции подразумевает исследование устойчивости алгоритма по отношению, по меньшей мере, к следующим видам атак:

* нахождение [коллизий](https://ru.wikipedia.org/wiki/%25D0%259A%25D0%25BE%25D0%25BB%25D0%25BB%25D0%25B8%25D0%25B7%25D0%25B8%25D1%258F_%25D1%2585%25D0%25B5%25D1%2588-%25D1%2584%25D1%2583%25D0%25BD%25D0%25BA%25D1%2586%25D0%25B8%25D0%25B8), то есть разных сообщений с одинаковым хэшем,
* нахождение [прообраза](https://ru.wikipedia.org/wiki/%25D0%259F%25D1%2580%25D0%25BE%25D0%25BE%25D0%25B1%25D1%2580%25D0%25B0%25D0%25B7), то есть неизвестного сообщения по его хэшу.

От устойчивости хэш-функции к нахождению коллизий зависит безопасность электронной цифровой подписи с использованием данного хэш-алгоритма. От устойчивости к нахождению прообраза зависит безопасность хранения хэшей паролей для целей аутентификации.

1. Производительность.

Или же быстродействие. По кодовой реализации, мы посчитали их теоретическую скорость, пользуясь знаниями информатики и математики.

Информацию, в основном, мы брали из интернета или консультируясь с учителями, а также пользовались книгами по криптографии из Лицейской библиотеки.

Начнем с теоретической части.

**Хэш**-**функция** - алгоритм, конвертирующий строку произвольной длины (сообщение) в битовую строку фиксированной длины, называемой **хэш**-кодом, проверочной суммой или цифровым отпечатком.

Криптоанализ – наука о способах расшифровки информации без предназначенного для этого ключа. Способность криптоалгоритма или хэш-функции противостоять криптоанализу называется криптостойкостью. Хеш-функция – это результат хеширования, т.е. преобразования входного массива данных в выходную битовую строку определённой длины. Криптографической хеш-функцией, соответственно, называют криптостойкую хеш-функцию.

Требования к хеш-функциям:

1. Хеш-функции должны быть непредсказуемыми, т.е. представлять собой некий случайный генератор битовых наборов.
2. Стойкость к восстановлению прообраза, где прообразом является x для значения функции f(x).
3. По заданному тексту x должно быть трудно найти z со свойством f(x) = f(z).

Рассмотрим несколько наиболее известных современных хэш-функций.

**SHA-2**

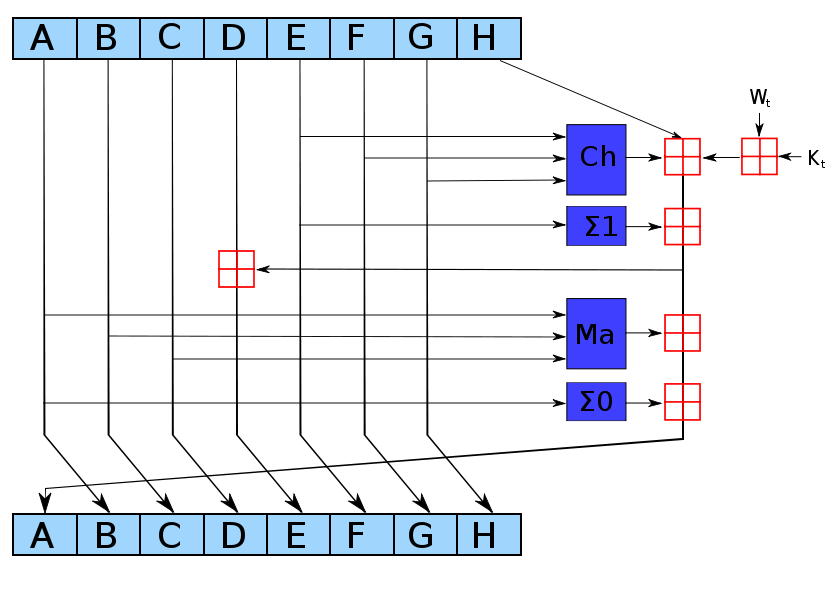


Схема одной итерации алгоритмов SHA-2

Хеш-функции семейства *SHA-2* построены на основе [структуры Меркла — Дамгарда](https://ru.wikipedia.org/wiki/%2525D0%2525A1%2525D1%252582%2525D1%252580%2525D1%252583%2525D0%2525BA%2525D1%252582%2525D1%252583%2525D1%252580%2525D0%2525B0_%2525D0%25259C%2525D0%2525B5%2525D1%252580%2525D0%2525BA%2525D0%2525BB%2525D0%2525B0%2525E2%252580%252594%2525D0%252594%2525D0%2525B0%2525D0%2525BC%2525D0%2525B3%2525D0%2525B0%2525D1%252580%2525D0%2525B4%2525D0%2525B0).

1. Исходное сообщение после дополнения разбивается на блоки, каждый блок — на 16 слов.
2. Алгоритм пропускает каждый блок сообщения через цикл с 64 или 80 итерациями (раундами). На каждой итерации 2 слова преобразуются, функцию преобразования задают остальные слова.
3. Результаты обработки каждого блока складываются, [сумма](https://ru.wikipedia.org/wiki/%2525D0%2525A1%2525D1%252583%2525D0%2525BC%2525D0%2525BC%2525D0%2525B0_%252528%2525D0%2525BC%2525D0%2525B0%2525D1%252582%2525D0%2525B5%2525D0%2525BC%2525D0%2525B0%2525D1%252582%2525D0%2525B8%2525D0%2525BA%2525D0%2525B0%252529) является значением хеш-функции.

Тем не менее, инициализация внутреннего состояния производится результатом обработки предыдущего блока. Поэтому независимо обрабатывать блоки и складывать результаты нельзя.

**Алгоритм использует следующие битовые операции:**

ǁ — конкатенация,

+ — сложение,

*and* — побитовое «И»,

*or* — побитовое «ИЛИ»,

*xor* — исключающее «ИЛИ»,

*shr* (shift right) — логический сдвиг вправо,

*rotr* (rotate right) — циклический сдвиг вправо.

В следующей таблице показаны некоторые технические характеристики различных вариантов SHA-2. «Внутреннее состояние» обозначает промежуточную хеш-сумму после обработки очередного блока данных:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Хеш-функция** | **Длина дайджеста сообщения (бит)** | **Длина внутреннего состояния (бит)** | **Длина блока (бит)** | **Максимальная** **длина сообщения (бит)** | **Длина слова (бит)** | **Количество итераций в цикле** |
| *SHA256*, *SHA224* | 256/224 | 256 | 512 | 264 − 1 | 32 | 64 |
| *SHA512*, *SHA384*, *SHA512/256*, *SHA512/224* | 512/384/256/224 | 512 | 1024 | 2128 − 1 | 64 | 80 |

* Криптостойкость

В [2003 году](https://ru.wikipedia.org/wiki/2003) Гилберт и Хандшух провели исследование *SHA-2*, но не нашли каких-либо уязвимостей. Однако в марте 2008 года индийские исследователи Сомитра Кумар Санадия и Палаш Саркар опубликовали найденные ими коллизии для 22 итераций *SHA-256* и *SHA-512*. В сентябре того же года они представили метод конструирования коллизий для усечённых вариантов *SHA-2* (21 итерация).

[Криптоанализ](https://ru.wikipedia.org/wiki/%2525D0%25259A%2525D1%252580%2525D0%2525B8%2525D0%2525BF%2525D1%252582%2525D0%2525BE%2525D0%2525B0%2525D0%2525BD%2525D0%2525B0%2525D0%2525BB%2525D0%2525B8%2525D0%2525B7) хеш-функции подразумевает исследование устойчивости алгоритма по отношению, по меньшей мере, к следующим видам атак:

* нахождение [коллизий](https://ru.wikipedia.org/wiki/%2525D0%25259A%2525D0%2525BE%2525D0%2525BB%2525D0%2525BB%2525D0%2525B8%2525D0%2525B7%2525D0%2525B8%2525D1%25258F_%2525D1%252585%2525D0%2525B5%2525D1%252588-%2525D1%252584%2525D1%252583%2525D0%2525BD%2525D0%2525BA%2525D1%252586%2525D0%2525B8%2525D0%2525B8), то есть разных сообщений с одинаковым хешем,
* нахождение [прообраза](https://ru.wikipedia.org/wiki/%2525D0%25259F%2525D1%252580%2525D0%2525BE%2525D0%2525BE%2525D0%2525B1%2525D1%252580%2525D0%2525B0%2525D0%2525B7), то есть неизвестного сообщения по его хешу.

От устойчивости хеш-функции к нахождению коллизий зависит безопасность электронной цифровой подписи с использованием данного хеш-алгоритма. От устойчивости к нахождению прообраза зависит безопасность хранения хешей паролей для целей [аутентификации](https://ru.wikipedia.org/wiki/%2525D0%252590%2525D1%252583%2525D1%252582%2525D0%2525B5%2525D0%2525BD%2525D1%252582%2525D0%2525B8%2525D1%252584%2525D0%2525B8%2525D0%2525BA%2525D0%2525B0%2525D1%252586%2525D0%2525B8%2525D1%25258F).

Ввиду алгоритмической схожести *SHA-2* с *SHA-1* и наличия у последней потенциальных уязвимостей принято решение, что SHA-3 будет базироваться на совершенно ином алгоритме. 2 октября 2012 года NIST утвердил в качестве SHA-3 алгоритм [Keccak](https://ru.wikipedia.org/wiki/Keccak).

* Производительность

Рассчитаем вычислительную сложность алгоритма

1. Разделение текста на блоки по 512 бит

Внутри каждого блока проходятся циклы:

2) Разбивание на 16 слов по 32 бита

3) Генерирование дополнительных 48 слов

4) Основной цикл(64 итерации)

Таким образом,

(N/512) \*16 = 10^8

=> производительность алгоритма SHA-2 составляет ~ **381.4 МБ/с.**

# BMW Hash function

Алгоритм BMW работает с сообщениями, разбивая их на блоки. Блок, в свою очередь, делятся на слова. Размеры блоков и слов зависят от конкретной реализации алгоритма. В таблице ниже перечислены основные свойства всех 4х вариаций алгоритма BMW.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Алгоритм** | **Размер сообщения** | **Размер блока** | **Размер слова** | **Цифровая подпись** |
| BMW224 | <264 | 512 | 32 | 224 |
| BMW384 | <264 | 512 | 32 | 384 |
| BMW224 | <264 | 1024 | 64 | 224 |
| BMW512 | <264 | 1024 | 64 | 512 |

### Операции, используемые в алгоритме

1. Побитовая операция [XOR](https://ru.wikipedia.org/wiki/%2525D0%2525A1%2525D0%2525BB%2525D0%2525BE%2525D0%2525B6%2525D0%2525B5%2525D0%2525BD%2525D0%2525B8%2525D0%2525B5_%2525D0%2525BF%2525D0%2525BE_%2525D0%2525BC%2525D0%2525BE%2525D0%2525B4%2525D1%252583%2525D0%2525BB%2525D1%25258E_2)
2. [Операции побитового сложения + или вычитания — по модулю 32 или 64](https://ru.wikipedia.org/wiki/%2525D0%2525A1%2525D1%252580%2525D0%2525B0%2525D0%2525B2%2525D0%2525BD%2525D0%2525B5%2525D0%2525BD%2525D0%2525B8%2525D0%2525B5_%2525D0%2525BF%2525D0%2525BE_%2525D0%2525BC%2525D0%2525BE%2525D0%2525B4%2525D1%252583%2525D0%2525BB%2525D1%25258E_%2525D0%2525BD%2525D0%2525B0%2525D1%252582%2525D1%252583%2525D1%252580%2525D0%2525B0%2525D0%2525BB%2525D1%25258C%2525D0%2525BD%2525D0%2525BE%2525D0%2525B3%2525D0%2525BE_%2525D1%252587%2525D0%2525B8%2525D1%252581%2525D0%2525BB%2525D0%2525B0), в зависимости от модификации алгоритма
3. Операция [сдвига влево (вправо)](https://ru.wikipedia.org/wiki/%2525D0%252591%2525D0%2525B8%2525D1%252582%2525D0%2525BE%2525D0%2525B2%2525D1%25258B%2525D0%2525B9_%2525D1%252581%2525D0%2525B4%2525D0%2525B2%2525D0%2525B8%2525D0%2525B3) на r бит SHLr (соответственно SHRr)
4. Вращение (циклический сдвиг влево) ROTLr

### Общие особенности структуры BMW

BLUE MIDNIGHT WISH следует общим принципам построения [хэш функций](https://ru.wikipedia.org/wiki/%2525D0%2525A5%2525D0%2525B5%2525D1%252588%2525D0%2525B8%2525D1%252580%2525D0%2525BE%2525D0%2525B2%2525D0%2525B0%2525D0%2525BD%2525D0%2525B8%2525D0%2525B5), которые часто употребляются на сегодняшний день. А именно, это значит, что алгоритм разбивается на две части:

#### Preprocessing

1. Ввод сообщения
2. Разбор введённого сообщения на m-битовые блоки
3. Инициализация начальных значений, которые будут использоваться при вычислении хф

#### Hash computition

1. Вычисление регистра сообщения из полученного сообщения
2. Использование этого регистра для вычисления последовательности значений H(i)
3. n наименее значимых бит выбираются как цифровая подпись

#### BMW384 или BMW512

Пусть длина сообщения l. К сообщению приписывается 1, за которой следует последовательность k нулей, где k — наименьшее неотриц. решение уравнения [l+1+k=960 mod1024](https://ru.wikipedia.org/wiki/%2525D0%2525A1%2525D1%252580%2525D0%2525B0%2525D0%2525B2%2525D0%2525BD%2525D0%2525B5%2525D0%2525BD%2525D0%2525B8%2525D0%2525B5_%2525D0%2525BF%2525D0%2525BE_%2525D0%2525BC%2525D0%2525BE%2525D0%2525B4%2525D1%252583%2525D0%2525BB%2525D1%25258E_%2525D0%2525BD%2525D0%2525B0%2525D1%252582%2525D1%252583%2525D1%252580%2525D0%2525B0%2525D0%2525BB%2525D1%25258C%2525D0%2525BD%2525D0%2525BE%2525D0%2525B3%2525D0%2525BE_%2525D1%252587%2525D0%2525B8%2525D1%252581%2525D0%2525BB%2525D0%2525B0). Далее приписывается 64-битный блок двоичного представления числа l. Примером может служить представление пос-ти «abc» (согласно [ASCII](https://ru.wikipedia.org/wiki/ASCII))



### Этап вычисления Хэш-Функции

В процессе вычислений используются три функции

* Первая функция **F0** : **{0,1}2m→{0,1}m**. Она принимает два аргумента M(i) и H(i−1) и производит [биективное отображение](https://ru.wikipedia.org/wiki/%2525D0%252591%2525D0%2525B8%2525D0%2525B5%2525D0%2525BA%2525D1%252586%2525D0%2525B8%2525D1%25258F) M(i) XOR H(i−1). Здесь M(i) — i-й блок сообщения, H(i−1) — текущее значение двоичной трубы. Результат записывается в первую часть учетверенной трубы: Q(i, a)=F0(M(i), H(i−1))=**A2**(**A1**(M(i) XOR H(i−1))).
* Вторая функция **F1** : **{0,1}2m→{0,1}m**. Она принимает в качестве аргументов блок сообщения M(i) (длины m бит) и первую часть учетверенной трубы Q(i, a). Результат записывается во вторую часть учетверенной трубы Q(i, b)=F1(M(i),Q(i, a)).
* Третья функция **F3** : **{0,1}3m→{0,1}m**. Для неё используется термин сворачивающей ([англ.](https://ru.wikipedia.org/wiki/%2525D0%252590%2525D0%2525BD%2525D0%2525B3%2525D0%2525BB%2525D0%2525B8%2525D0%2525B9%2525D1%252581%2525D0%2525BA%2525D0%2525B8%2525D0%2525B9_%2525D1%25258F%2525D0%2525B7%2525D1%25258B%2525D0%2525BA) *{{{1}}}* — *folding* ) с целью подчеркнуть свойства её свертки 3m-мерного пространства в m-мерное. Она принимает в качестве аргументов две величины: блок сообщения M(i) и текущее значение учетверенной трубы Q(i)=Q(i, a)||Q(i, b). Результат записывается как новое значение H(i): H(i) = F2(M(i),Q(i))

Псевдокод вычисления хэш-функции

for (i=1;i<N;i++)

{

Q(i,a)=F0(M(i),H(i-1));

Q(i,b)=F1(M(i),Q(i,a));

Q(i)=Q(i,a)||Q(i,b);

H(i)=F2(M(i),Q(i));

}

Hash=N\_LeastSignificantBits(H(i));

* Криптостойкость

Согласно исследованиям, проведёнными [группой разработчиков алгоритма BMW](http://www.item.ntnu.no/people/personalpages/fac/danilog/blue_midnight_wish/), можно сформулировать основные положения о криптографической силе, устойчивости к коллизиям, нахождению прообразов, повторных прообразов, устойчивости к удлинению длины и мультиколлизионным атакам:

1. Устойчивость к коллизиям примерно n/2 бит
2. Устойчивость к нахождению прообразов n бит
3. Повторное нахождение прообразов n-k бит для всех сообщений короче 2n-k бит
4. Устойчивость к увеличению длины
5. Устойчивость к мультиколлизионным атакам

* Производительность

Рассчитаем вычислительную сложность алгоритма, выполненного в одной из реализаций:

1. Разбивание текста на блоки по 1024 бит
2. Разбивание каждого блока на 64 слова
3. Вычисление хэш-функции, применение ее к словам.

Таким образом,

(N/1024) \* 2^4 = 10^8

Тогда средняя производительность алгоритма в данной реализации будет приблизительно составлять **763 Мб/с.**

# ГОСТ Р 34.11-2012

* Принцип работы

1. На вход хеш-функции подается сообщение произвольного размера.
2. Сообщение разбивается на блоки по 512 бит, если размер сообщения не кратен 512, то оно дополняется необходимым количеством бит.
3. Потом итерационно используется функции сжатия, в результате действия которой обновляется внутреннее состояние хеш-функции. Также вычисляется контрольная сумма блоков и число обработанных бит.
4. Когда обработаны все блоки исходного сообщения, производятся еще два вычисления:
   * + Обработка функцией сжатия блока с общей длиной сообщения.
     + Обработка функцией сжатия блока с контрольной суммой.

Что и завершает вычисление хеш-функции.

* Криптостойкость

Криптоанализ старого стандарта выявил некоторые его слабые стороны с теоретической точки зрения. Так в одной из работ, посвящённых криптоанализу ГОСТ Р 34.11 — 94, было выявлено, что сложность алгоритма построения прообраза оценивается в 2192 вычислений функций сжатия, коллизии 2105, что меньше «универсальных» оценок, которые для ГОСТ Р 34.11 — 94 равны 2256 и 2128. Хотя по состоянию на конец 2013 года еще нет большого числа работ, посвящённых криптостойкости новой хеш-функции, но, исходя из конструкции новой хеш-функции, можно сделать некоторые выводы о её криптостойкости и предположить, что её криптостойкость будет выше, чем у ГОСТ Р 34.11 — 1994.

* В разделе «Описание» из схемы видно, что все блоки сообщения суммируются по модулю 2512 и уже результат суммирования всех блоков подается на вход завершающего этапа (stage3). Благодаря тому, что здесь суммирование — это не побитовое сложение, получается защита от следующих атак:
  + Построение мультиколлизий.
  + Удлинение прообраза.
  + Дифференциальный криптоанализ.
* В функции сжатия используется конструкция Миагучи-Пренели, это обеспечивает защиту от атаки, основанную на фиксированных точках, так как для конструкции Миагучи-Пренели не найдено способов (легких) для поиска фиксированных точек.
* Также на каждой итерации при вычислении хеш-кода используются различные константы. Это затрудняет атаки на основе связанных и разностных связанных ключей, атаки скольжения и отражения.

Теперь рассмотрим результаты, полученные при криптоанализе ГОСТ Р 34.11 — 2012. В 2013 году на сайте <http://eprint.iacr.org/2013> было опубликовано две статьи, посвящённых криптоанализу новой хеш-функции. Первая статья называется «Rebound attack on Stribog», в ней исследуется устойчивость хеш-функции по отношению к атаке, называемой «The Rebound attack», в основе этой атаки лежит «rotation cryptanalysis» и [дифференциальный криптоанализ](https://ru.wikipedia.org/wiki/%2525D0%252594%2525D0%2525B8%2525D1%252584%2525D1%252584%2525D0%2525B5%2525D1%252580%2525D0%2525B5%2525D0%2525BD%2525D1%252586%2525D0%2525B8%2525D0%2525B0%2525D0%2525BB%2525D1%25258C%2525D0%2525BD%2525D1%25258B%2525D0%2525B9_%2525D0%2525BA%2525D1%252580%2525D0%2525B8%2525D0%2525BF%2525D1%252582%2525D0%2525BE%2525D0%2525B0%2525D0%2525BD%2525D0%2525B0%2525D0%2525BB%2525D0%2525B8%2525D0%2525B7). Криптоаналитики при поиске уязвимостей использовали метод, называемый «free-start». Это означает, что при вычислении хеш-кода фиксируется некоторое состояние хеш-функции и дальше вычисления могут идти как в сторону вычисления хеш-кода, так и в сторону вычисления сообщения. Таким образом, криптоаналитики сумели добиться коллизии за 5 раундов и была получена так называемая «near collision» (это означает, что были найдены два сообщения, хеш-коды которых отличны в малом количестве бит) при использовании 7,75 раундов. Также было установлено, что схема, по которой выбираются ключи для каждого раунда, добавляет устойчивости функции сжатия. Однако, было показано, что коллизия возможна за 7,75 раундов, а «near collision» — за 8,75 и 9,75, соответственно.

Название второй статьи — «Integral Distinguishers for Reduced-round Stribog», в ней рассматривается стойкость хеш-функции (с уменьшенным количеством раундов) по отношению к [интегральному криптоанализу](https://ru.wikipedia.org/wiki/%2525D0%252598%2525D0%2525BD%2525D1%252582%2525D0%2525B5%2525D0%2525B3%2525D1%252580%2525D0%2525B0%2525D0%2525BB%2525D1%25258C%2525D0%2525BD%2525D1%25258B%2525D0%2525B9_%2525D0%2525BA%2525D1%252580%2525D0%2525B8%2525D0%2525BF%2525D1%252582%2525D0%2525BE%2525D0%2525B0%2525D0%2525BD%2525D0%2525B0%2525D0%2525BB%2525D0%2525B8%2525D0%2525B7). В этой работе криптоаналитикам при исследовании функции сжатия удалось найти дифференциал за 4 раунда при вычислении в прямом направлении и за 3,5 раунда при вычислении в обратном направлении. Также было выяснено, что атака нахождения дифференциала на хеш-функцию с числом раундов 6 и 7 требует 264 и 2120 среднераундовых значений, соответственно.

* Производительность

На сайте, посвящённом VI Международной конференции «Параллельные вычисления и задачи управления» (PACO’2012), в разделе «Параллельные вычисления в прикладных задачах» ([http://paco2012.ipu.ru/content/параллельные-вычисления-в-прикладных-задачах](http://paco2012.ipu.ru/content/%2525D0%2525BF%2525D0%2525B0%2525D1%252580%2525D0%2525B0%2525D0%2525BB%2525D0%2525BB%2525D0%2525B5%2525D0%2525BB%2525D1%25258C%2525D0%2525BD%2525D1%25258B%2525D0%2525B5-%2525D0%2525B2%2525D1%25258B%2525D1%252587%2525D0%2525B8%2525D1%252581%2525D0%2525BB%2525D0%2525B5%2525D0%2525BD%2525D0%2525B8%2525D1%25258F-%2525D0%2525B2-%2525D0%2525BF%2525D1%252580%2525D0%2525B8%2525D0%2525BA%2525D0%2525BB%2525D0%2525B0%2525D0%2525B4%2525D0%2525BD%2525D1%25258B%2525D1%252585-%2525D0%2525B7%2525D0%2525B0%2525D0%2525B4%2525D0%2525B0%2525D1%252587%2525D0%2525B0%2525D1%252585)) представлена статья, написанная Лебедевым П. А., «Сравнение старого и нового стандартов РФ на криптографическую хэш-функцию на ЦП и графический процессорах NVIDIA», в которой проводится сравнение быстродействия семейства криптографических хеш-функций ГОСТ Р 34.11-94 и ГОСТ Р 34.11-2012 на процессорах архитектуры x86\_64 и видеокартах NVIDIA с поддержкой технологии CUDA. Для сравнения быстродействия на процессоре архитектуры x86\_64 были взяты 4 разных реализации хеш-функций:

1. Реализация ГОСТ Р 34.11-1994 из криптографического пакета OpenSSL (версия 1.0.1c) с открытым исходным кодом. В этой реализации нет алгоритмических и программных оптимизаций.
2. Реализация ГОСТ Р 34.11-1994 в программе RHash (версия 1.2.9). В этой реализации есть алгоритмические и программные оптимизации, в том числе ассемблерные оптимизации.
3. Реализация ГОСТ Р 34.11-2012, написанная Александром Казимировым <https://github.com/okazymyrov/stribog>
4. Реализации ГОСТ Р 34.11-1994 и ГОСТ Р 34.11-2012, написанные П. А. Лебедевым.

Производительность:

(N/512)\*10 = 10^8 => **P=608 Мб/с.**

Использовался процессор Intel Core i7-920 CPU @ 2.67 GHz

Результаты производительности представлены в двух таблицах:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **ГОСТ Р 34.11-1994** | | **ГОСТ Р 34.11-2012** | |
| Реализация № | Мб/с | Тактов/байт | Мб/с | Тактов/байт |
| 1 | 18 | 143 | - | - |
| 2 | 49 | 52 | - | - |
| 3 | - | - | 38 | 67 |
| 4 | 64 | 40 | 94 | 27 |

Сравнение быстродействия старого и нового стандартов хеш-функций на ГП проводилось между реализациями П. А. Лебедева Использовалась видеокарта NVIDIA GTX 580 Результаты производительности. (8192 потока данных по 16 Кб)

|  |  |  |  |
| --- | --- | --- | --- |
| **ГОСТ Р 34.11-1994** | | **ГОСТ Р 34.11-2012** | |
| Мб/с | Тактов/байт | Мб/с | Тактов/байт |
| 1697 | - | 608 | - |

На основании этих результатов сделан вывод, что хеш-функция ГОСТ Р 34.11-2012 может быть в два раза быстрее хеш-функции ГОСТ Р 34.11-94 на современных процессорах, но медленнее на графических картах и системах с ограниченными ресурсами. Такие результаты производительности можно объяснить тем, что при вычислении новой хеш-функции используются только сложения по модулю 2 и инструкции пересылки данных. Старая хеш-функции содержит много инструкций перемешивания, которые не лучшим образом отображаются на набор команд ЦП. Но увеличенный размер состояний и таблиц подстановки хеш-функции ГОСТ Р 34.11 — 2012 делает её медленней на высокопараллельных вычислительных средствах, таких как графические процессоры.

Также исследование производительности новой хеш-функции было проведено её разработчиками на 64-битном процессоре Intel Xeon E5335 2ГГц. Использовалось одно ядро. Производительно хеш-функции ГОСТ Р 34.11 — 2012 составила 51 такт процессора на 1 байт хешируемых данных (это примерно 40 Mбайт/с). Полученный результат на 20 % лучше, чем у старой хеш-функции ГОСТ Р 34.11 — 1994.

Этап 3.

1. Реализация алгоритма, вызвавшего наибольший интерес, на любом известном языке программирования

Перед началом реализации требовалось детально разобраться с алгоритмом работы выбранного нами криптоалгоритма RC4, для чего использовалось множество различных источников. При изучении алгоритма можно было выделить основные функции, которые будут использоваться для работы с файлом, а также приблизительно определить ресурсы, которые будут использоваться в программе.

* Выбор криптоалгоритма

Алгоритм RC4 был выбран нами по причине того, что он наглядно показывает общий принцип работы большинства криптоалгоритмов. Как говорилось ранее, RC4 складывает по модулю 2 каждый байт незашифрованного текста с каждым байтом последовательности, сгенерированной с помощью ключа. Также алгоритм отличается своим высоким быстродействием, так что реализация данного алгоритма может оказаться крайне полезной.

* Ресурсы

Реализация производилась на языке **C++**, для разработки использовалась операционная система Mac OS и компилятор Xcode.

* Описание работы

Перед началом работы в директории /Users/Shared должен находиться файл, содержащий ключ, с именем key.txt. В этой же директории должен находится файл, которой нужно зашифровать/расшифровать. Конечный файл также будет расположен в данной директории.

Сначала программа запрашивает у пользователя, в каком режиме она работает - шифрование или дешифрование. В режиме шифрования она запрашивает название файла, который нужно зашифровать, и файла, где должен находиться зашифрованный текст. Далее инициализируется S-блок, и с помощью нескольких функций текст зашифровывается. В режиме дешифрования - аналогично.

* Тестирование

Программа тестировалась на шифровании и дешифровании большинства чаще всего используемых видов файлов. Например, на файлах с расширениями .*txt, .pages, .ppt, .png, .jpg, .mp4, .m4a, .pdf, .zip* и др. Для шифрования папок и приложений стоит сжать их до расширения .*zip*. Программа поддерживает русский язык в файлах и в названиях.

* Время работы

В потоке ввода-вывода также можно увидеть время работы шифрования/дешифрования в секундах. Протестировав несколько файлов в обоих режимах, можно заметить, что время дешифрования обычно на несколько миллисекунд меньше времени шифрования.

Итого

Теоретическая скорость, рассчитанная выше, как правило, значительно отличается от практической. Это связано с тем, что при оценке вычислительной сложности не учитываются отдельные операции, многие циклические операции, не зависящие от кол-ва поступающей информации, и константы при m, условные операторы, инициализации массивов.

В результате анализа можем заключить, что криптоалгоритм RC4 имеет наиболее высокую скорость шифрования и дешифрования по сравнению с другими рассмотренными алгоритмами. Но при этом он имеет наименьшую криптостойкость по сравнению с ними. Наибольший объем памяти займет американский стандарт шифрования AES, наименьший - российский ГОСТ 28147-89. Криптостойкости AES и ГОСТ 28147-89 в общих случаях ***примерно одинаковы***, но для повышения производительности шифрования и дешифрования для AES можно выбрать ключ длинной , например, 128 бит, что случается довольно часто, - в этих случаях криптостойкость российского стандарта, естественно выше(полный перебор ключей: 2^128 операций против 2^256), пусть и практической разницы между этими цифрами нет. Что касается RC4, то его криптостойкость ***гораздо ниже*** криптостойкости других рассмотренных алгоритмов, чем и окупается скорость его работы.

Тем не менее, при грамотной реализации, все рассмотренные алгоритмы устойчивы к криптоанализу, и взлом шифра сводится к прямому перебору ключей, так что скорость взлома(и, собственно, его возможность) зависит от длины выбранного ключа.

В приложении приведен собственный анализ скоростей и расходов памяти рассмотренных алгоритмов, а также некоторые официальные исследования теоретических и практических скоростей различных криптоалгоритмов.

Скорость анализируемых хеш-функций SHA-2 и BMW приблизительно одинакова, но криптографическая сила преобладает в большей степени у SHA-2, так как исследования этой хеш-функции показали что она не имеет каких-либо уязвимостей. В то время как характеристики хеш-алгоритма BMW доказывает меньшую криптостойкость, по сравнению с SHA-2. А хеш-функция ГОСТ Р 34.11-2012 хоть и не обладает высокой криптографической силой, но восполняет её своим быстродействием. Анализы, и тесты показали на сколько ГОСТ Р 34.11-2012 работает быстрее, остальных хеш-алгоритмов.

В целом, все эти хеш-функции достаточно устойчивы к нахождению коллизий или прообразов, и каждая из них в меру быстродействующая, поэтому с их помощью при хорошей реализации можно спокойно хранить хеши важных данных.

Написанная нами реализация способна наглядно показать основной принцип работы большинства криптоалгоритмов - сложение по модулю 2 текста с некой последовательностью.

**Перспективы использования результатов проекта**

Благодаря проведенному исследованию, любой пожелавший хранить и передавать свои данные защищенными, может определиться с методом, в зависимости от того, что наиболее важно в конкретной ситуации - наиболее высокая скорость работы или наибольшая стойкость к взлому. Благодаря анализу современных функций хэширования любой желающий может определиться с тем, какую из них использовать, также выбирая между производительностью и криптостойкостью.

Как говорилось ранее, алгоритм RC4 был выбран для реализации по причине его высокой производительности. Таким образом, написанная нами программа может оказаться крайне полезной для использования в целях шифрования данных в случаях, когда крайне важна высокая скорость работы, а информация в незашифрованном виде теряет свою актуальность достаточно быстро.

**Источники:**

**ГОСТ:** https://ru.wikipedia.org/wiki/ГОСТ\_28147-89

**ГОСТ+:** https://xakep.ru/2013/10/19/shifrovanie-gost-28147-89/

**ГОСТ+:** http://security-corp.org/infosecurity/crypto/26518-shifrovanie-gost-28147-89-na-h86-i-gpu-processorah.html

**Реализация ГОСТ:** http://habrahabr.ru/post/81032/

**RC4:** <https://ru.wikipedia.org/wiki/RC4>

**RC4+**: http://m.habrahabr.ru/post/111510/

**Реалиазация RC4:** [https://ru.wikibooks.org/wiki/Реализации\_алгоритмов/RC4](https://ru.wikibooks.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2/RC4)

**Криптостойкость RC4:** „A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher” -

Souradyuti Paul and Bart Preneel

**Криптостойкость RC4: „**Attacks on the RC4 stream cipher”

* Andreas Klein

**AES:** <https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard#.D0.9A.D1.80.D0.B8.D0.BF.D1.82.D0.BE.D1.81.D1.82.D0.BE.D0.B9.D0.BA.D0.BE.D1.81.D1.82.D1.8C>

**AES+:** http://teh-box.ru/programming/algoritm-shifrovaniya-aes-dlya-samyx-malenkix.html

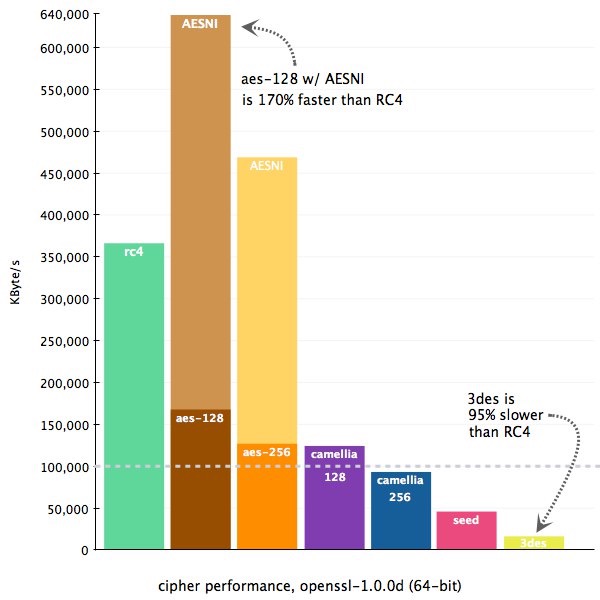
**Реализация AES:** <http://rsdn.ru/forum/src/2477192.hot>

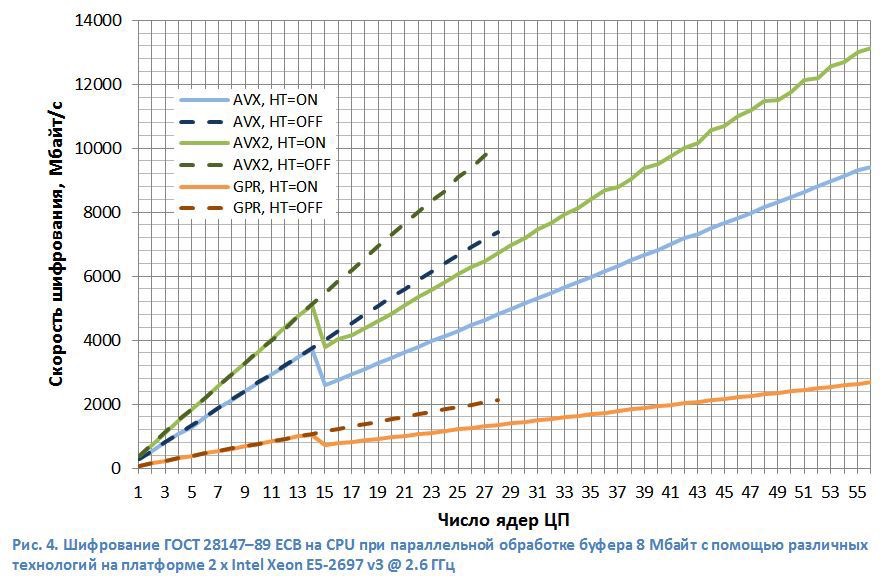
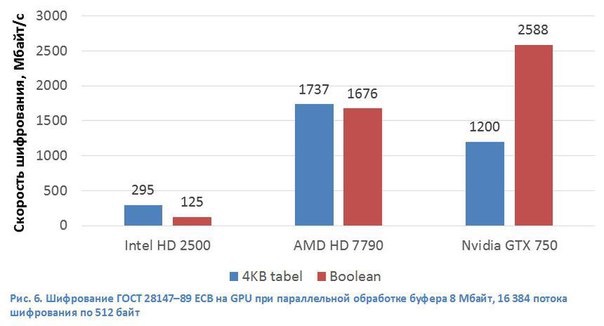
**Теория:** <http://citforum.ru/internet/infsecure/its2000_13.shtml>

**BMW:** <https://ru.wikipedia.org/wiki/BMW_Hash_function#.D0.90.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC>

**SHA-2:** <https://ru.wikipedia.org/wiki/SHA-2#.D0.9F.D1.81.D0.B5.D0.B2.D0.B4.D0.BE.D0.BA.D0.BE.D0.B4_SHA-256>

**ГОСТ Р 34.10-2012:** [https://ru.wikipedia.org/wiki/ГОСТ\_Р\_34.10-2012#.D0.9A.D1.80.D0.B8.D0.BF.D1.82.D0.BE.D1.81.D1.82.D0.BE.D0.B9.D0.BA.D0.BE.D1.81.D1.82.D1.8C](https://ru.wikipedia.org/wiki/%D0%93%D0%9E%D0%A1%D0%A2_%D0%A0_34.10-2012#.D0.9A.D1.80.D0.B8.D0.BF.D1.82.D0.BE.D1.81.D1.82.D0.BE.D0.B9.D0.BA.D0.BE.D1.81.D1.82.D1.8C)



**Прил****ожения**