**Министерство науки и высшего образования Российской Федерации**

**ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»**

Кафедра «школы бакалавриата (школа)»

Оценка работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель от УрФУ Попов В.Ю.

Тема задания на практику

Ускоренное шифрование на основе разделения секрета

ОТЧЕТ

Вид практики Производственная практика

Тип практики Практика по получению профессиональных умений и опыта профессиональной деятельности

Студент Кулаков В.С.

Специальность (направление подготовки) 10.05.01 Компьютерная безопасность

Группа МЕН-541015

Екатеринбург 2019

# **Введение**

Актуальность проблемы сокрытия информации существует с момента появления самой информации. Ещё в Древнем Риме Цезарь использовал шифр сдвига, который был частным случаем шифра простой замены для передачи секретной информации (например, в зону военных действий). В настоящее время потребность сокрытия информации наиболее актуальна для электронных устройств и сетей. С увеличением важности и количества информации на современные шифры накладываются различные дополнительные сложности. Во-первых, алгоритм должен быть стойким, что достигается путём увеличения ключа шифрования и сложностью вычисления данных. Во-вторых, алгоритм должен быть быстрым и сложность вычисления алгоритма должна быть всё же значительно меньше сложности его взлома, ведь нам же не хочется ждать шифрования/дешифрования столько же времени, сколько потратит взломщик. Наиболее современным алгоритмом шифрования информации является AES. Для этого алгоритма существуют все возможные ухищрения в ускорении вплоть до реализации команд процессора. Самым стойким вариантом AES является AES256. Однако чем длиннее ключ, тем тяжелее вычисления. Предложено для ускорения процесса шифрования реализовать схему разделения секрета которая позволит разгрузить AES и ускорить его работу за счёт увеличения потребляемой памяти. Также разделение секрета позволяет разделить информацию между группой участников, которые не имеют возможности взломать весь секрет.

**Объект исследования** – алгоритм шифрования с разделением секрета.

**Целью работы** является реализация алгоритма шифрования на основе AES256 и внедрённым алгоритмом разделения секрета с возможностью восстановления секрета при деструктивной активности одного из участников.

В работе решаются следующие задачи:

* Рассмотрение вариантов внедрения разделения секрета
* Подбор параметров разделения секрета
* Анализ скорости работы алгоритма и сравнение его с AES256

# **Глава 1 Алгоритм AES**

## **Описание**

Для того чтобы составить алгоритм работающий лучше другого алгоритма необходимо изначально осознать и реализовать исходный. Итак, алгоритм AES – симметричный итеративный блоковый алгоритм.

Характеристики AES:

* Работает с 128-bit блоками данных (16 байт)
* Три типа ключа 128/192/256
* Три типа раундов 10/12/14 в зависимости от ключа
* Все раунды идентичны, кроме последнего

Операции шифра происходят в поле Галуа GF(28) по модулю многочлена x8+x4+x3+x+1.

**Элемент поля:** Битовый многочлен

**Операция сложения:**

Где

**Операция умножения:**

Перемножение многочлена работает по обычным алгебраическим правилам и взятие многочлена по модулю x8+x4+x3+x+1.

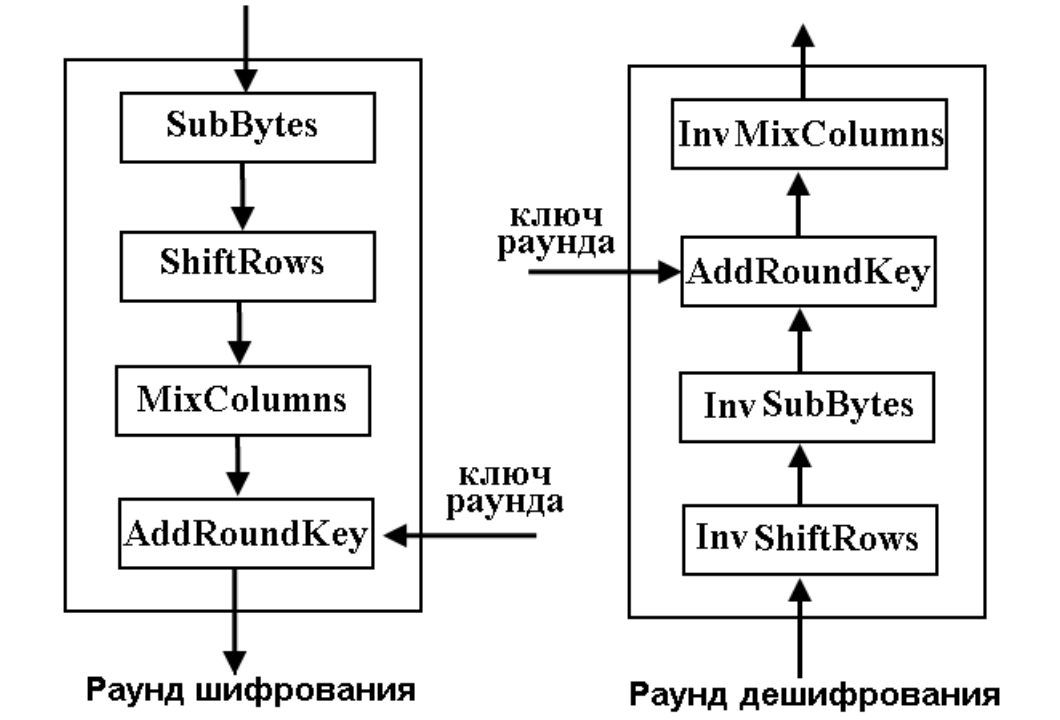
Но для более быстро умножения возьмём таблицу ([Приложение 1](#_№1_Таблицы_степени)) степени и логарифма в поле Галуа GF(28) над многочленом x+1. Теперь умножение:

Так же вычисляем обратный элемент обходя тяжёлый алгоритм Эвклида:

Данные в алгоритме AES раскладываются в виде квадрата 16 байт(4x4):

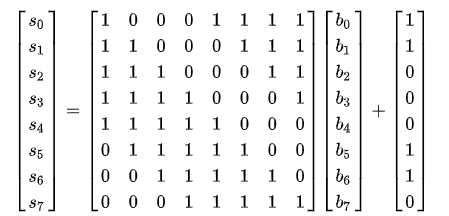
Сам алгоритм шифрования и дешифрования состоит из 4 процедур, циклически применяемых к блоку в течение 14 раундов.

1 раунд шифрования/дешифрования:

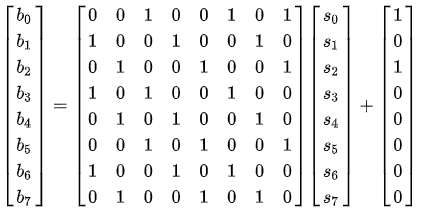


## **Процедуры преобразования состояний**

1. Процедура SubBytes производит над каждым байтом преобразование вида:



Но для ускорения в памяти просто хранится матрица Sbox ([Приложение 2](#_№2_Матрицы_Sbox)) по которой заменяются байты. Обратное преобразование процедура InvSubBytes выглядит так (Таблица InvSbox [Приложение 2](#_№2_Матрицы_Sbox)):



2. Процедура ShiftRows просто сдвигает строчки блока

Понятно, что процедура InvShiftRows просто сдвигает в другую сторону

3. Процедура MixColumns перемешивает байты в столбцах матрицы, каждый столбец представляется как многочлен над полем GF(28):

Он умножается на многочлен: по модулю многочлена x4+1.

Обратная функция InvMixColumns просто умножается на обратный многочлен:

4. Операция AddRoundKey является обратной самой себе и представляет собой суммированием в поле GF(28) байтов состояния с байтами расширения ключа.

## **Процедура расширения ключа**

При инициализации шифра ключом происходит расширение этого ключа на все раунды работы алгоритма, так как рассматривается алгоритм c 256-bit(32 byte) ключом, то нужно расширить его на 14 раундов или 15 16-bit слов (так как размер блока с которым работает AES 128 bit для любого ключа).

Для AES256 процедура заполнения выглядит так:

: Копируется последний столбец в первый столбец, он циклически сдвигается на 1 вниз, к байтам применяется операция SubWord аналог SubBytes, но для 4 байт и всё это суммируется с таблицей Rcon ([Приложение 2](#_№2_Матрицы_Sbox)) i-го раунда. Потом столбец суммируется с первым столбцом

Остальные столбцы просто суммируют столбец слева с соответствующим им столбцом

: Всё то же самое, что с , но без сдвига и суммирования с Rcon.

**Вывод по первой главе**

Алгоритм AES достаточно простой для построения и состоит из обратимых операций. Сложность алгоритма определяется скоростью обработки операций в поле Галуа GF(28), которые ускорены за счёт использования памяти и таблиц, тем самым по сложности они примерно равны сложности сложения двух обычных чисел. Так же на сложность алгоритма влияют 14 раундов по 4 операции над квадратами из 16 байт. Это достаточно много что даёт нам возможность внедрить какой-нибудь алгоритм разделения секрета, который будет работать быстрее, чем 14 раундов множества операций в поле. Сам алгоритм реализован в однопоточном режиме для шифрования файла.

# **Глава 2 Разделение секрета**

## **Выбор схемы**

Схемы разделения предназначены для хранения информации распределённой между участниками, при этом ни один участник в идеале не должен иметь возможность узнать из своей информации хоть что-нибудь об общем секрете. Простейшим примером такого разделения секрета является побитовое сложение или xor. Однако в нашем случае одним из условий является возможность восстановления секрета при деструктивной активности одного участника, то есть одно из разделений может быть неправильно. Для решения этой задачи обратимся к пороговым схемам разделения секрета.

**Пороговая схема разделения секрета (K, N)**  – это схема разделения в которой секрет разделяется на N долей и K долей нужно для его восстановления. (K ≤ N очевидно).

Теперь мы можем восстанавливать секрет из K долей и иметь дополнительные, если одна из долей не верна.

Наиболее универсальной пороговой схемой разделения является схема Шамира. Её и будем использовать для нашего алгоритма.

## **Схема разделения секрета Шамира**

Схема Шамира основана на интерполяционных полиномах Лагранжа.

**Основное свойство:** Для восстановления многочлена степени k-1 нужно k точек.

**Алгоритм:**

1. Пусть нужно разделить M – секрет, между N участниками так, чтобы K участников могли восстановить секрет.
2. Выберем некоторое конечное поле для схемы (например, GF или Zp)
3. Генерируем Ai – случайные коэффициенты многочлена:
4. Генерируем Xi – разные входные данные для многочлена, можно по порядку.
5. Вычисляем доли секрета Vi :
6. Раздаём доли секрета участникам

**Восстановление секрета:** Для восстановления секрета можно решить систему уравнений с K неизвестными, нужно K из N уравнений. Или воспользоваться формулой Лагранжа:

## **Варианты синтеза алгоритма из AES и схемы разделения секрета**

Вариант №1:

* Возьмём секрет
* Разделим его как
* Генерируем так, чтобы он был
* Шифруем его AES
* Так как по зашифрован, то можно сделать вывод, что схема зашифрована AES

Плюсы:

* + Шифрование ускорено
  + Возможность распределённого хранения
  + Длина шифрованного сообщения меньше, следовательно, сложнее взлом

Минусы:

* Слишком маленький (к примеру, 1 бит) уменьшает сложность перебора. А достаточный для AES256 требует очень большое поле.
* Проблема генерации случайных долей в схеме Шамира из заданного диапазона.
* Общее увеличение размера информации

Вариант №2:

Возьмём и будем использовать одинаковые доли секрета, которые будем шифровать, тем самым эти доли будут защищены AES, а другие доли будут меняться в зависимости от секрета. Для восстановления секрета всё ещё нужны K долей, а размер шифрованных долей будет как минимум длинны ключа AES, тем самым перебор этих долей будет как перебор ключа AES.

Плюсы:

* Сложность перебора регулируется длинной шифрованных долей
* Шифрование ускорено
* Длина шифрованного сообщения меньше, следовательно, сложнее взлом

Минусы:

* Проблема генерации случайных долей в схеме Шамира из заданного диапазона.
* Общее увеличение размера информации
* Цикличность

## **Решение проблемы генерации заданных долей в схеме Шамира**

Предположим, что у нас есть заранее сгенерированные доли и нам надо получить оставшиеся. Многочлен Лагранжа выглядит как:

При S ≥ K мы восстанавливаем весь многочлен через систему уравнений или формулу Лагранжа, но при S < K мы не можем восстановить многочлен, не хватает точек. Однако если нам известна часть многочлена, то оставшиеся коэффициенты мы легко восстанавливаем, например системой уравнений с S неизвестными, значит, мы генерируем так, чтобы осталось S не сгенерированных . И многочлен теперь выглядит вот так:

Генерируем под систему уравнений вида:

Справа высчитываемые константы, слева S уравнений с S неизвестными, и мы получаем в пороговой схеме Шамира возможность генерировать нужные нам доли секрета. Очевидным образом с каждой новой узнанной мы можем уменьшать систему на одно уравнение и генерировать следующие с меньшим числом уравнений.

**Пояснение:** Схема не теряет своих свойств, так как такой процесс можно проводить для любых и получать любые . Так как множество значений по мощности равно множеству значений , а значит, из случайной выборки получаем случайные и обратно.

# **Глава 3 Алгоритм и сравнение с AES256**

## **Описание Алгоритма**

Итак, сформулируем наш алгоритм, основанный на синтезе пороговой схемы разделения секрета Шамира и AES.

1. Генерируем или договариваемся о значениях для схемы разделения.
2. Генерируем от 32 схем Шамира ( блоки) из над полем GF(28) (так как он уже есть для AES и длина ключа AES256 32 байта).
3. Собираем файлы из блоков в количестве S и шифруем их AES, и записываем. Это наши первые шифрованные доли.
4. Генерируем оставшиеся доли, подставляя секрет в соответствующую схему разделения, схемы зацикливаем или используем другую последовательность подстановки.
5. Собираем сгенерированные доли и записываем их в файл

Расширенное описание пунктов конкретно моего алгоритма:

1. Значения представляют собой элементы поля Галуа, 1 байт, генерируются случайные и разные.
2. 1 схема Шамира состоит из набора получаемых извне и сгенерированных однобайтовых элементов поля Галуа, которые представляют блоки. Их S штук.
3. Оставшиеся доли генерируются по выше описанному [решению проблемы](#_Решение_проблемы_генерации) также в поле Галуа

Ограничения алгоритма:

1. K < N – условия пороговой схемы, и должна быть возможность восстановить секрет даже если один участник врёт.
2. S < K – количество сгенерированных долей должно быть меньше количества долей требуемых для восстановления секрета, иначе не будет возможности восстанавливать секрет
3. K + S > N для восстановления долей должен обязательно использоваться один из шифрованных секретов, так как иначе секрет не будет защищён
4. S ≥ 2 для моего случая, так как нужно чтобы была возможность восстановить секрет даже если врёт участник с шифрованной долей

Процесс дешифрования такой же, как в схеме разделения секрета, только блоки вначале расшифровываются и подставляются в процесс восстановления в соответствии с последовательностью подстановки.

## **Сравнение скорости работы алгоритма с AES256**

Минимальная схема разделения секрета имеет параметры K=3 S=2 N=4, именно с такой схемой будет сравнительный анализ.

| **Имя файла** | **Размер файла, байт** | **AES.exe** | | | **AESSecretSharing.exe** | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Время шифрования** | **Размер шифрованного файла, байт** | **Время дешифрования** | **Время обработки** | **Размер шифрованных файлов, байт** | **Время дешифрования** |
| test.tif | 46 354 552 | 45s.139 | 46 354 560 | 45s.833 | 34s.908 | 92 709 172 | 26s.482 |
| test.mp3 | 97 114 029 | 1m: 34s.866 | 97 114 032 | 1m: 35s.550 | 1m: 13s.497 | 194 228 126 | 1m: 13s.159 |
| test.iso | 359 661 568 | 5m: 47s.819 | 359 661 584 | 5m: 53s.862 | 4m: 30s.489 | 719 323 204 | 3m: 24s.300 |
| test.avi | 1 555 154 944 | 25m: 6s.405 | 1 555 154 960 | 25m: 38s.985 | 19m: 39s.905 | 3 110 309 956 | 19m: 31s.883 |

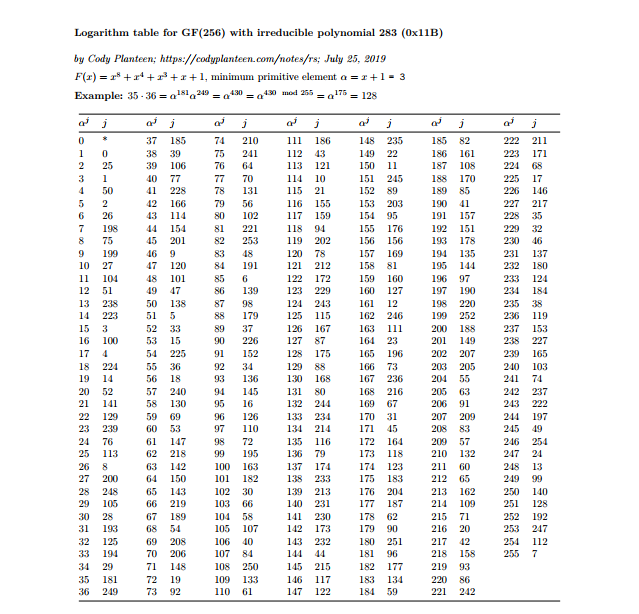
Время работы алгоритма уменьшилось на 22% при шифровании и ещё больше при дешифровании, что опять же доказывает экономическую состоятельность данного способа, если вы не стеснены в ресурсах памяти, потому что её расход увеличился в 2 раза, потому что у нас 2 полноценных доли. Возможно, существуют и другие конфигурации, как данного алгоритма, так и вариаций AES с применением разделения секрета имеющих более сильный прирост производительности. Можно, например, использовать более большое поле Галуа, но с увеличением поля появится и потребность в реализации алгоритмов, которые сейчас сводятся к переходам по таблице и это уменьшит производительность алгоритма, в общем.

# **Заключение**

Проведена работа по анализу алгоритма AES. Разобран и реализован сам алгоритм в однопоточном режиме и с наиболее эффективными процедурами для более быстрой работы алгоритма. Разобран процесс внедрения пороговой схемы разделения секрета с возможностью восстановления информации даже при деструктивной активности одного из блоков. На основе схемы разделения секрета Шамира построен алгоритм шифрования с использование AES и возможностью распределения долей секрета между несколькими участниками и учтена возможность ложной информации от одного из них. Этот алгоритм реализован также в однопоточном режиме с таким же буфером обработки, как и у алгоритма AES чтобы алгоритмы находились приблизительно в одинаковых условиях. Внутри кода программы внедрена система измерения времени, как полной работы алгоритма для проверки его возможностей при условии нагрузки на чтение/запись большого числа файлов, так и измерение времени работы чисто над процедурами шифрования/дешифрования, чтобы не было влияния тяжёлых операция чтения/записи больших объёмов файлов. На основе этих данных составлена сравнительная таблица, по которой доказана экономическая целесообразность данного алгоритма. Алгоритм может быть использован как для шифрования файлов, так и для сетевых транзакций с переработкой процедур в потоковый режим. Положительной чертой для сетевых транзакций и других распределённых передач шифрованных данных будет абсолютная невозможность восстановления данных без получения нужного числа долей, даже если злоумышленнику будет известен ключ. Однако тот же ключ является единственной защитой данных при возможности получения нужного числа долей. Есть возможность расширение действия алгоритма за счёт увеличения числа долей на большее количество деструктивных участников.

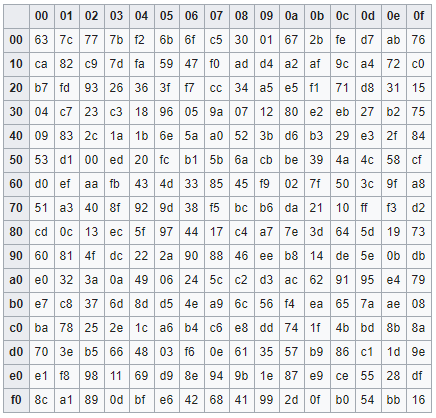
**Приложения**

## **№1 Таблицы степени и логарифма GF(28)**

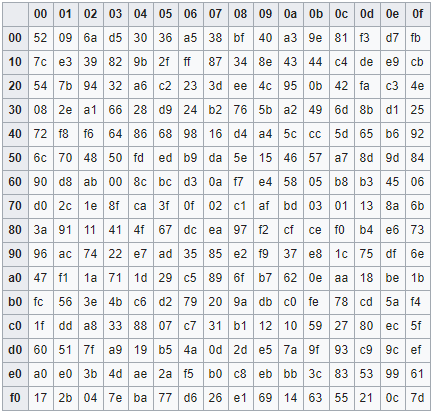


## **№2 Матрицы Sbox, InvSbox, Rcon**

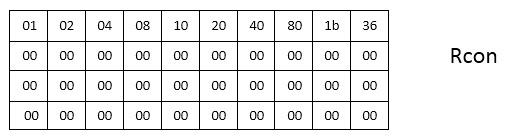
Sbox



InvSbox



Rcon



Оглавление

[**Титульный лист** 1](#_Toc23397444)

[**Введение** 2](#_Toc23397449)

[**Глава 1 Алгоритм AES** 3](#_Toc23397450)

[**Описание** 3](#_Toc23397451)

[**Процедуры преобразования состояний** 5](#_Toc23397452)

[**Процедура расширения ключа** 6](#_Toc23397453)

[**Глава 2 Разделение секрета** 7](#_Toc23397454)

[**Выбор схемы** 7](#_Toc23397455)

[**Схема разделения секрета Шамира** 7](#_Toc23397456)

[**Варианты синтеза алгоритма из AES и схемы разделения секрета** 8](#_Toc23397457)

[**Решение проблемы генерации заданных долей в схеме Шамира** 9](#_Toc23397458)

[**Глава 3 Алгоритм и сравнение с AES256** 11](#_Toc23397459)

[**Описание Алгоритма** 11](#_Toc23397460)

[**Сравнение скорости работы алгоритма с AES256** 12](#_Toc23397461)

[**Заключение** 13](#_Toc23397462)

[**Приложения** 14](#_Toc23397463)

[**№1 Таблицы степени и логарифма GF(28)** 14](#_Toc23397464)

[**№2 Матрицы Sbox, InvSbox, Rcon** 16](#_Toc23397465)