Минобрнауки России
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Сыктывкарский государственный университет имени Питирима Сорокина»
(ФГБОУ ВО «СГУ им. Питирима Сорокина»)
Институт точных наук и информационных технологий
Кафедра прикладной математики и информационных технологий в образовании

Допустить к защите
Зав. кафедрой доцент, доцент
_____Ермоленко А.В.
(звание, должность, Ф.И.О.)
_____
“__” июня 2016 г.

«Эффективные реализации современных криптографических алгоритмов»

01.03.02, Прикладная математика и информатика

Исполнитель: Габова Елена Юрьевна

Научный руководитель: Ильчуков Александр Сергеевич,
старший преподаватель

Сыктывкар 2016

Тема: «Эффективные реализации современных криптографических алгоритмов»

Объём дипломной работы 77 страниц, на которых размещены 12 рисунков и 3 таблицы. При написании диплома использовалось 14 источников.

Ключевые слова: шифрование, хеширование, AES, WHIRLPOOL.

В качестве обьекта исследования были выбраны блочный шифр AES и хэш-функция WHIRLPOOL.

Предмет исследования: Реализация криптографически стойких алгоритмов на языке C и на языке ассемблера ARMv7.

В дипломную работу входит введение, 5 глав теоретической части, 1 глава практической части, итоговое заключение.

Во введении раскрывается актуальность исследования по выбранному направлению, ставится проблема, цель и задачи исследования, определяются объект исследования, кратко обозреваются алгоритмы AES и WHIRLPOOL.

В первой главе данной работы можно познакомиться с основными понятиями современной криптографии, возможностями ее практического применения. Во второй главе кратко описана теория конечных полей, необходимая для построения некоторых криптографических систем. Третья глава данной работы рассказывает о построении алгоритма блочного шифрования AES. В четвертой главе будет рассказано о построении хэш-функции WHIRLPOOL. В пятой главе дипломной работы описаны некоторые ассемблерные иснтрукции.

В заключении сравнивается скорость работы программ, и делаются общие выводы.

Содержание

Словарь терминов и сокращений

В алгоритме AES используются следующие определения:

Block — последовательность бит, из которых состоит ввод (Input), вывод (Output), состояние (State) и раундовый ключ (Round Key). Также под Block можно понимать последовательность байт.

Cipher Key — секретный криптографический ключ, который используется процедурой Key Expansion, чтобы произвести набор ключей для раундов (Round Keys); может быть представлен как прямоугольный массив байтов, имеющий четыре строки и некоторое количество (Nk) колонок.

Input — входные данные алгоритма шифрования.

Output — выходные данные алгоритма шифрования.

Key Expansion — процедура генерации раундовых ключей (Round Keys) из исходного ключа (Cipher Key).

Round Keys — раундовые ключи, которые получаются из исходного ключа (Cipher Key) после использования процедуры Key Expansion. Эти ключи используются при шифровании и дешифровании.

State — промежуточный результат шифрования, который может быть представлен как прямоугольный массив байт имеющий 4 строки и некоторое количество (Nb) колонок.

AES_sbox — таблица замен, использующаяся в нескольких преобразованиях и в процедуре Key Expansion для замены значения байта. Предварительно рассчитанный AES_sbox можно увидеть в приложении 1.

— число столбцов (32-х битных слов), составляющих состояние State. В алгоритме AES Nb = 4.

— число 32-х битных слов, составляющих ключ шифрования. В алгоритме AES Nk = 4, 6, или 8.

r — число раундов в функции сжатия. В алгоритме AES Nr = 10, 12, 14.

Rcon[] — массив, который состоит из битов 32-х разрядного слова и является постоянным для данного раунда. Предварительно рассчитанный Rcon[] можно увидеть в приложении 1.

В алгоритме WHIRLPOOL используются следующие определения:

B oc — последовательность бит, из которых состоит ввод (Input), вывод (Output), состояние (State). Также под Block можно понимать последовательность байт.

'e — криптографический ключ, который используется для хеширования данных.

np ut — входные данные алгоритма хеширования.

tp ut — выходные данные алгоритма хеширования.

t ti — промежуточный результат хеширования, который может быть представлен как прямоугольный массив байтов имеющий размер 8*8.

! s ox — таблица замен, использующаяся в нескольких трансформациях замены байт. Предварительно рассчитанный WHIRLPOOL_sbox можно увидеть в приложении 1.

B C BY E — число байт в таблице состояния. В алгоритме WHIRLPOOL BLOCK_NBYTES = 64.

! B — число строк и столбцов в матрице состояния. В алгоритме WHIRLPOOL WHIRLPOOL_NB = 8.

Введение

Проблема защиты секретной информации путем ее преобразования, исключающего ее прочтение нежелательным лицом, волновала человеческий ум с давних времен. Первоначально письменность сама по себе была криптографической системой, в Древнем Египте и Индии ею владели только избранные.

Бурное развитие криптографические системы получили в годы первой и второй мировых войн. Появление вычислительных средств ускорило разработку и совершенствование криптографических методов [11, стр. 16-18].

В 1976 году Диффи и Хеллман опубликовали статью [9]. Эта статья представила революционную концепцию криптографии с открытым ключом, а также в ней был предложен новый метод для обмена ключами. Хотя авторы не имели практической реализации схемы шифрования с открытым ключом, идея была ясна, и она породила огромный интерес в криптографическом сообществе.

Одними из наиболее востребованных в криптографии семействами алгоритмов являются блочные шифры и хеш-функции.

Блочный шифр — разновидность симметричного шифра, оперирующего группами бит фиксированной длины. Если исходный текст меньше размера блока, перед шифрованием его дополняют. Блочный шифр шифрует только один блок исходного сообщения, поэтому необходим какой-то метод, позволяющий шифровать несколько блоков.

Режим шифрования — метод применения блочного шифра, преобразующий последовательность блоков входного текста в последовательность блоков зашифрованных данных. При этом для шифрования одного блока могут быть использованы данные другого блока. В рамках дипломной работы рассматриваются два режима шифрования: ECB (Electronic Codebook) и CBC (Cipher Block Chaining).

Термин хеш-функция (или функция хеширования) возник в теории сложности вычислений, где он обозначал функцию, сжимающую строку чисел произвольного размера в строку чисел фиксированного размера. Данное понятие использовалось в алгоритмах поиска данных по значениям хеш-функции от них. В настоящее время хеш-функции применяются для

построения ассоциативных массивов, для хранения паролей в системах защиты, при создании цифровой подписи.

В качестве объекта исследования были выбраны блочный шифр AES и хеш-функция WHIRLPOOL в связи с тем, что они используют схожий математический аппарат и обладают подобной алгоритмической структурой. В рамках дипломной работы написаны несколько версий программ для шифрования и хеширования данных и проведен их сравнительный анализ.

AES (RIJNDAEL) — симметричный алгоритм блочного шифрования, принятый в качестве стандарта шифрования правительством США. Разработан бельгийцами Винсентом Рэйменом и Йоаном Дайменом в 1997 году.

В 2000 году Винсент Рэймен и Паоло Баретто разработали хеш-функцию WHIRLPOOL. WHIRLPOOL осуществляет хеширование входного сообщения с длиной до 2^{256} бит. Основой для хеш-функции WHIRLPOOL является блочный шифр с симметричными ключами AES.

К сожалению, AES не является устойчивым к атакам на кэш [5]. Атака по сторонним каналам на кэш основана на временном различии при доступе к кэшированным и некэшированным данным.

В апреле 2005 года Дэниел Бернштейн опубликовал работу с описанием атаки, использующей для взлома информацию о времени выполнения каждой операции шифрования [6]. Данная атака потребовала более 200 миллионов выбранных шифротекстов для нахождения ключа. В октябре 2005 года Даг Арне Освик, Ади Шамир и Эран Трумер представили работу [12] с описанием нескольких атак, использующих время выполнения операций для нахождения ключа.

Для защиты от атак по сторонним каналам были написаны реализации, исключающие атаки на кэш, они не используют таблицы подстановки. В данных версиях происходит большое количество вычислений, и выполнение занимает гораздо больше времени. С целью ускорения времени выполнения ключевые части алгоритмов были переписаны на ассемблере.

Платформой для ассемблерной реализации было выбрано семейство мобильных процессоров ARMv7. Мобильные платформы характеризуются повышенными требованиями к энергоэффективности и слабой производительностью. В то же время мобильные устройства представляют широкий

интерес, по итогам второго квартала 2015 года в мире было продано 446 млн единиц мобильной техники. Около 98 % мобильных устройств оснащено процессорами ARM. Потребительские и серверные платформы также представляют интерес, но в рамках дипломной работы решено было заострить внимание именно на мобильных платформах.

Предмет исследования: Реализация криптографически стойких алгоритмов на языке C и на языке ассемблера ARMv7.

Цель дипломной работы: Разработка программной библиотеки для шифрования и хеширования данных.

Задачи:

1. Изучение современных криптографических алгоритмов.
2. Изучение семейства процессоров ARMv7.
3. Создание эффективной и безопасной (с точки зрения атак на кэш) реализации AES и WHIRLPOOL.

В рамках дипломной работы:

- Изучаются команды для взаимодействия с процессорами ARMv7.
- Изучается процедура генерации ключа и шифрования для алгоритма AES с длиной ключа 128, 192, и 256 бит.
- Реализуются тестовые программы для работы в режимах ECB и CBC.
- Изучается процедура хеширования для алгоритма WHIRLPOOL с длиной ключа 512 бит.
- Реализуются тестовые программы для хеширования файлов.
- Программы для шифрования и хеширования данных проверяются на различных типах файлов.

В первой главе данной работы можно познакомиться с основными понятиями современной криптографии, возможностями ее практического применения. Во второй главе кратко описана теория конечных полей, необходимая для построения некоторых криптографических систем. Третья глава данной работы рассказывает о построении алгоритма блочного шифрования AES. В четвертой главе будет рассказано о построении хэш-функции WHIRLPOOL. В пятой главе дипломной работы описаны некоторые ассемблерные инструкции. В практической части описана программная реализация алгоритмов AES и WHIRLPOOL.

. Основные понятия современной крипто ра ии

Проблемой защиты информации путем ее преобразования занимается криптология (kryptos — тайный, logos — наука). Криптология включает в себя два направления — криптографию и криптоанализ. Криптография занимается поиском и исследованием математических методов преобразования информации. Сфера интересов криптоанализа — исследование возможности расшифровывания информации без знания ключей.

Современная криптография включает в себя четыре крупных раздела:

1) криптографические системы с открытым ключом;
2) симметричные криптографические системы;
3) системы электронной подписи;
4) управление ключами.

Ши рование — преобразование информации с целью сокрытия от нежелательных лиц. В данный момент шифрование существует на пересечении математики, информатики и электроники. Шифрование применяется для передачи данных о банковских картах, компьютерных паролях, и так далее.

Деши рование — обратный шифрованию процесс. С помощью ключа шифрованный текст преобразуется в исходный.

К юч — информация, необходимая для беспрепятственного шифрования и дешифрования информации.

Важной особенностью любого алгоритма шифрования является использование ключа, с помощью которого можно выбрать конкретное преобразование из всех, представленных алгоритмом. Создатель зашифрованного сообщения делится техникой декодирования, необходимой для восстановления сообщения, только с получателем, и нежелательные лица не могут прочитать или изменить сообщение.

Различают симметричные [11, стр. 30-37] и асимметричные [11, стр. 40-48] методы шифрования. Симметричное шифрование использует один и тот же ключ для шифрования и дешифрования, асимметричное шифрование использует два разных ключа (открытый и закрытый).

В симметричных системах шифрования алгоритм и ключ выбирается заранее и известен обеим сторонам. Сохранение ключа в секретности является важной задачей для установления и поддержки защищённого канала связи. В связи с этим, возникает проблема синхронизации ключей. В качестве примера симметричного метода шифрования можно привести шифр Цезаря.

В системах с открытым ключом используются два ключа (открытый и закрытый), связанные определенным математическим алгоритмом. Открытый ключ передаётся по доступному для наблюдения каналу и используется для шифрования сообщения. Для дешифрования сообщения используется секретный ключ. Данная схема решает проблему симметричных схем, связанную с синхронизацией ключей. Если в симметричных схемах злоумышленник перехватит ключ, то он сможет вносить правки в передаваемую информацию. В асимметричных системах другой стороне передается открытый ключ, который позволяет шифровать, но не расшифровывать информацию.

Криптостойкостью называется характеристика шифра, определяющая его стойкость к криптоанализу. Имеется несколько показателей криптостойкости, среди которых можно выделить количество всех возможных ключей и среднее время, необходимое для взлома шифра.

Хеширование — преобразование исходной информации произвольной длины в битовую строку фиксированной длины.

Криптографическая хеш-функция — хеш-функция, являющаяся криптографически стойкой, удовлетворяющая ряду требований, специфичных для криптографических приложений.

Требования к криптографически стойким хеш-функциям:

- описание функции $H(X)$ должно быть открыто;
- аргумент X функции $H(X)$ может быть строкой чисел произвольной длины, а значение функции должно быть строкой чисел фиксированной длины;
- для любого X вычисление $H(X)$ должно происходить за полиномиальное время;

- для заданного значения хеш-функции X должно быть вычислительно трудно подобрать блок данных M, для которого $H(M) = X$;
- для заданного сообщения M должно быть вычислительно трудно подо

. Конечные поля

Конечным полем, или полем Галуа, называется конечный набор элементов, на котором определены операции сложения, умножения, вычитания и деления (кроме деления на 0), удовлетворяющие аксиомам поля [1]. Понятие конечного поля используется, в теории чисел, теории групп, алгебраической геометрии, криптографии, в разработке криптографически стойких шифров, таких как AES [2].

Элементы поля Галуа можно складывать и умножать, но эти операции отличаются от тех, которые используются для чисел. В данном параграфе будут описаны основные математические операции в полях Галуа.

. . Группы

Группой [3] называется множество G, на котором определена ассоциативная бинарная операция $\cdot$, содержащее элемент e такой, что для любого элемента $a \in G$ выполняется

$$e \cdot a = a \cdot e = a,$$

и существует элемент a^{-1} такой, что

$$a \cdot a^{-1} = a^{-1} \cdot a = e$$

Указанный элемент e называется **единицей** группы. Элемент a^{-1} называется **обратным** к элементу a. Если операция $\cdot$ коммутативна, то группа называется **коммутативной**, или **абелевой**.

В теории групп используется две равноправных и эквивалентных друг другу терминологических системы: аддитивная и мультипликативная. В аддитивной системе групповую операцию называют операцией сложения, а группы в аддитивной записи для краткости иногда будем назвать **аддитивными**. Группы, операцию в которых называют умножением, иногда именуются **мультипликативными** группами.

Единичный элемент аддитивной группы обозначается 0 и называется **нулем**. Элемент аддитивной группы, обратный к элементу a обозначается $-a$, и называется **противоположным** к этому элементу. Единичный элемент мультипликативной группы обозначается 1 и называется **единицей**. Элемент мультипликативной группы, обратный к элементу a обозначается

a^{-1}.

В аддитивной группе определяется операция **вычитания** $a - b$ по формуле:

$$a - b = a + (-b)$$

В мультипликативной группе определяется операция **деления** a/b по формуле:

$$a/b = a \times (b^{-1)}$$

Результат $a - b$ называется **разностью** элементов a и b, результат a/b называется **частным** от деления элемента a на элемент b.

Используя мультипликативные и аддитивные обозначения в общем случае имеем следующие правила:

Мультипликативные обозначения:	Аддитивные обозначения:
$a^{-n} = (a^{-1)^n}$	$(-n)a = n(-a)$
$a^m a^n = (a^{m+n})$	$ma + na = (m + n)(a)$
$(a^{m)^n} = (a^{m \times n})$	$m(na) = (m \times n)a$

Мультипликативная группа G называется **циклической**, если в ней имеется такой элемент а , что каждый элемент $b \in G$ является степенью элемента а т. е. существует целое число К такое, что $b = a^K$. Этот элемент а называется **образующим** элементом группы G. Для циклической группы G применяют обозначение $G =<$ а $>$.

Отношением эквивалентности называется бинарное отношение, которое обладает свойствами транзитивности, рефлексивности и симметричности. Множество, на котором задано это отношение, разбивается на классы эквивалентности $[a]$, где a - представитель класса. Совокупность классов эквивалентности есть фактор-множество множества, на котором определено это бинарное отношение эквивалентности.

Важным классом эквивалентности на множестве целых чисел является отношение сравнимости чисел а и b $(a, b \in z)$ по модулю n. $a = b(mod\ \ n)$. Классы эквивалентности, на которые отношение сравнимости по модулю n разбивает множество целых чисел Z, называют **классами вычетов по модулю** n.

Группа называется **конечной (бесконечной)**, если она состоит из конечного (бесконечного) числа элементов. Число элементов конечной группы G называется ее порядком и обозначается $|G|$. Подмножества H

группы G называется **под руппой** этой группы, если H само образует группу относительно операций группы.

. . Ко ца

Ко цом R называется множество R с двумя бинарными операциями, обозначаемыми $+$ и $\times$ такими, что:

1. Операция $+$ коммутативна, для любых $a, b \in R$, справедливо $a + b = b + a$

2. Операция $\times$ ассоциативна, для любых $a, b, c \in R$, справедливо $(a \times b) \times c = a \times (b \times c)$

3. Выполняются законы дистрибутивности: для любых $a, b, c \in R$ справедливо $a \times (b + c) = ab + ac; (b + c) \times a = ba + ca.$

Кольцо называется **кольцом с единицей**, если существует такой элемент $e \in R$, что $a \times e = e \times a = a$ для всех $a \in R$

Кольцо называется **коммутативным**, если операция $\times$ коммутативна: $a \times b = b \times a$

Кольцо называется **целостным кольцом**, если оно является коммутативным кольцом с единицей $e \neq 0$, в котором равенство $a \times b = 0$ влечет за собой $a = 0$ или $b = 0$.

Кольцо R называется **телом**, если это кольцо с единицей, в котором для каждого элемента $a \neq 0$ существует обратный по умножению элемент a^{-1}: $a \times (a^{-1)} = e$.

Аддитивной руппой кольца R называется группа, заданная на множестве R с помощью операции $+$, имеющейся в кольце. Такая группа обозначается R^+. Обозначим через R^* множество тех элементов $a \in R$, для которых существует обратный элемент $b \in R$ со свойством $ab = ba = e$. Такая группа называется **мультипликативной руппой кольца** R.

Пусть x и m — натуральные числа. Обозначим через $r_m(x)$ наименьший отрицательный остаток от деления x на m. Таким образом, $0 \le r_m(x) \le m - 1$ и разность $x - r_m(x)$ делится на m. Наименьшие неотрицательные остатки при делении натуральных чисел на m образуют множество $Z_m = 0,1,..m-1$.

Зададим на множестве Z_m сложение и умножение по следующим правилам:

– сумма элементов i и j равна $r_m(i + j)$;

– произведение элементов i и j равно $r_m(i * j)$.

Тогда множество Z_m становится кольцом. Оно называется **кольцом вычетов по модулю** m.

Подмножество S кольца R называется **подкольцом это о кольца**, если оно замкнуто относительно операций сложения и умножения и образует кольцо относительно этих операций (то есть результат операций $a + b$ и $a \times b$ для a, b ∈ S также ∈ S).

Пусть $R_1 .. R_s$ — некоторые кольца. Тогда $R_1 \oplus .. \oplus R_s$ является кольцом. Это кольцо называется **прямой суммой колец** $K_1 .. K_s$;

.3. Поле Галуа

Полем называется коммутативное и ассоциативное кольцо F с единицей, в котором для любого элемента $a \in F$ существует обратный элемент $a^{-1} \in F$, $a \times a^{-1} = e$. Конечное поле обычно обозначается F_q или $GF(q)$ (сокращение от Galois field) и называется полем Галуа порядка q, где q — число элементов поля.

Важным является пример конечного поля $GF(p)$ второго порядка, составленного из элементов [0,1]. Поле Галуа второго порядка можно представить в виде кольца Z_2, в котором сложение и умножение элементов сводится к покомпонентному сложению по модулю 2.

Для каждого простого числа p и каждого натурального числа n существует конечное поле из p^n элементов.

Это утверждение позволяет построить конечное поле Галуа порядка q, содержащее q элементов, где q — это степень простого числа p, которое является характеристикой этого поля. Поля этого типа обозначают $F(q)$, $GF(q)$ или $GF(p^n)$. Любые два поля $GF(p^n)$ **изоморфны между собой**.

Большую роль в криптографических приложениях играют многочлены, и наиболее интересные результаты получены в теории конечного поля.

Мно оч еном или полиномом называется выражение вида

$$f(x) = a_0 + a_1 x + .. a_n x^n$$

где n — целое неотрицательное число (степень многочлена),

a_i — элементы кольца (коэффициенты многочлена),

x — неизвестное.

Кольцо S, образованное многочленами над кольцом R, называется **кольцом многочленов над** R и обозначается $R[x]$.

Два многочлена $f(x)$ и $g(x) \in F[x]$ над полем F считаются **равными**, если:

$$f(x) = \sum_{i=1}^{n} a_i x^i \ , \ g(x) = \sum_{i=1}^{n} b_i x^i \ , \text{и } a_i = b_i \text{ для любого } i \in [0, n]$$

Многочлен $f(x) \in F[x]$ называется **неприводимым над полем** F или в кольце $F[x]$, если он имеет положительную степень и равенство $f(x) = g(x) \times h(x), g(x), h(x) \in F[x]$ может выполнятся лишь в том случае, когда либо $g(x)$, либо $h(x)$ имеют степень $n \le 0$ и являются константами.

Для любого простого числа p и натурального числа q существует неприводимый многочлен степени q над полем Z_p. Любой неприводимый многочлен из $Z_p[x]$ является делителем $x^{p^n} - x$.

Многочлен $f(x) \in Z_p[x]$ для поля $GF(p^n)$ представляется в следующем виде:

$$f(x) = a_0 + a_1 x + .. a_{n-1} x^{n-1} + x^n \qquad (2.1)$$

Для построения конечного поля Z_p можно использовать неприводимый полином. В результате получаем поле $GF(p^n)$ — единственное с точностью до изоморфизма поле с p^n элементами. Умножение в этом поле происходит по модулю этого неприводимого полинома.

4. Умножение полиномов в GF( )

Умножение $b(x)$ на 2 по модулю $m(x)$ в GF(2^8) можно реализовать с помощью левого сдвига $b(x)$ и побитового сложения с многочленом $m(x)$. Назовем эту операцию *xtime* и рассмотрим ее работу для $m(x) = x^8 + x^4 + x^3 + x + 1$.

$$[57] \times [13] =$$
$$[57] \times [02] = xtime([57]) = [ae]$$
$$[57] \times [04] = xtime([ae]) = [47]$$
$$[57] \times [08] = xtime([47]) = [8e]$$
$$[57] \times [10] = xtime([8e]) = [07]$$

Тогда

$$[57] \times [13] = [57] \times ([01] \oplus [02] \oplus [10]) = [57] \oplus [ae] \oplus [07] = [fe]$$
$$[57] \times [13] = [fe]$$

Для умножения 2 полиномов $a(x)$ и $b(x)$ в GF(2^8) по модулю $m(x)$ можно воспользоваться следующим алгоритмом:

```
uint8_t gmul(uint8_t a, uint8_t b)
begin
   uint8_t c = 0
   b = 0

   while (b)
     if (b & P 0x1)
        c = c xor a;
        high_bit_on = a & 0x80;
        a <<= 1;
        if (high_bit_on)
           a = a xor m;
           b >>= 1;
   end while

return c
end
```

.5. Поле Галуа в алгоритме шифрования AE

В алгоритме AES используется умножение матрицы состояния на многочлен $c(x)$ по модулю $d(x)$ в $GF(2^8)$, где многочлены $c(x)$ и $d(x)$ имеют вид

$$c(x) = [03]x^3 + [01]x^2 + [01]x + [02] \quad (2.2)$$
$$d(x) = x^8 + x^4 + x^3 + x + 1. \quad (2.3)$$

Рассмотрим перемножение полиномов $a(x)$ и $c(x)$ в поле GF(2^8):

$$b(x) = a(x) \times c(x) \quad (2.4)$$

Выражение (2.4) может быть записано в матричном виде:

$$\begin{bmatrix} b_{0,c} \\ b_{1,c} \\ b_{2,c} \\ b_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0,c} \\ a_{1,c} \\ a_{2,c} \\ a_{3,c} \end{bmatrix} \quad \text{для } 0 \le c \le Nb.$$

(2.5)

В результате можно найти $b_{0,c}..\ b_{3,c}$:

$$b_{0,c} = ([02]\times a_{0,c}) \oplus ([03]\times a_{1,c}) \oplus a_{2,c} \oplus a_{3,c}$$
$$b_{1,c} = a_{0,c} \oplus ([02]\times a_{1,c}) \oplus ([03]\times a_{2,c}) \oplus a_{3,c}$$
$$b_{2,c} = a_{0,c} \oplus a_{1,c} \oplus ([02]\times a_{2,c}) \oplus ([03]\times a_{3,c})$$
$$b_{3,c} = ([03]\times a_{0,c}) \oplus a_{1,c} \oplus a_{2,c} \oplus ([02]\times a_{3,c})$$

.6. По е Га уа в а оритме

В алгоритме WHIRLPOOL используется умножение матрицы состояния на многочлен $c(x)$ по модулю $d(x)$ в $GF(2^8)$, при этом многочлены $c(x)$ и $d(x)$ имеют вид:

$$c(x) = [01]x^7 + [01]x^6 + [04]x^5 + [01]x^4 + [08]x^3 + [05]x^2 + [02]x + [09] \quad (2.6)$$
$$d(x) = x^8 + x^4 + x^3 + ^2 + 1 \quad (2.7)$$

Рассмотрим перемножение полиномов $a(x)$ и $c(x)$ в поле GF(2^8):

$$b(x) = a(x)\times c(x) \quad (2.8)$$

Выражение (2.8) может быть записано в матричном виде ($0 \le i \le Nb$):

$$\begin{bmatrix} b_{i,0} \\ b_{i,1} \\ b_{i,2} \\ b_{i,3} \\ b_{i,4} \\ b_{i,5} \\ b_{i,6} \\ b_{i,7} \end{bmatrix} = \begin{bmatrix} 01 & 01 & 04 & 01 & 08 & 05 & 02 & 09 \\ 09 & 01 & 01 & 04 & 01 & 08 & 05 & 02 \\ 02 & 09 & 01 & 01 & 04 & 01 & 08 & 05 \\ 05 & 02 & 09 & 01 & 01 & 04 & 01 & 08 \\ 08 & 05 & 02 & 09 & 01 & 01 & 04 & 01 \\ 01 & 08 & 05 & 02 & 09 & 01 & 01 & 04 \\ 04 & 01 & 08 & 05 & 02 & 09 & 01 & 01 \\ 01 & 04 & 01 & 08 & 05 & 02 & 09 & 01 \end{bmatrix} \begin{bmatrix} a_{i,0} \\ a_{i,1} \\ a_{i,2} \\ a_{i,3} \\ a_{i,4} \\ a_{i,5} \\ a_{i,6} \\ a_{i,7} \end{bmatrix} \quad (2.9)$$

В результате можно найти $b_{i,0}..\ b_{i,7}$:

$$b_{i,0} = a_{i,0} \oplus a_{i,1} \oplus ([04]\times a_{i,2}) \oplus a_{i,3} \oplus ([08]\times a_{i,4}) \oplus ([05]\times a_{i,5}) \oplus ([02]\times a_{i,6}) \oplus ([09]\times a_{i,7})$$
$$b_{i,1} = ([09]\times a_{i,0}) \oplus a_{i,1} \oplus a_{i,2} \oplus ([04]\times a_{i,3}) \oplus a_{i,4} \oplus ([08]\times a_{i,5}) \oplus ([05]\times a_{i,6}) \oplus ([02]\times a_{i,7})$$
$$b_{i,2} = ([02]\times a_{i,0}) \oplus ([09]\times a_{i,1}) \oplus a_{i,2} \oplus a_{i,3} \oplus ([04]\times a_{i,4}) \oplus a_{i,5} \oplus ([08]\times a_{i,6}) \oplus ([05]\times a_{i,7})$$
$$b_{i,3} = ([05]\times a_{i,0}) \oplus ([02]\times a_{i,1}) \oplus ([09]\times a_{i,2}) \oplus a_{i,3} \oplus a_{i,4} \oplus ([04]\times a_{i,5}) \oplus a_{i,6} \oplus ([08]\times a_{i,7})$$
$$b_{i,4} = ([08]\times a_{i,0}) \oplus ([05]\times a_{i,1}) \oplus ([02]\times a_{i,2}) \oplus ([09]\times a_{i,3}) \oplus a_{i,4} \oplus a_{i,5} \oplus ([04]\times a_{i,6}) \oplus a_{i,7}$$
$$b_{i,5} = a_{i,0} \oplus ([08]\times a_{i,1}) \oplus ([05]\times a_{i,2}) \oplus ([02]\times a_{i,3}) \oplus ([09]\times a_{i,4}) \oplus a_{i,5} \oplus a_{i,6} \oplus ([04]\times a_{i,7})$$
$$b_{i,6} = ([04]\times a_{i,0}) \oplus a_{i,1} \oplus ([08]\times a_{i,2}) \oplus ([05]\times a_{i,3}) \oplus ([02]\times a_{i,4}) \oplus ([09]\times a_{i,5}) \oplus a_{i,6} \oplus a_{i,7}$$
$$b_{i,7} = a_{i,0} \oplus ([04]\times a_{i,1}) \oplus a_{i,2} \oplus ([08]\times a_{i,3}) \oplus ([05]\times a_{i,4}) \oplus ([02]\times a_{i,5}) \oplus ([09]\times a_{i,6}) \oplus a_{i,7}$$

2. Алгоритм блочного шифрования AES

2.1. История AES

В 1997 году NIST объявил конкурс на создание алгоритма, удовлетворяющего выдвинутым институтом требованиям. Чтобы быть утвержденным в качестве стандарта, алгоритм должен был

1) реализовать шифрование частным ключом;

2) представлять собой блочный шифр;

3) работать со 128-разрядными блоками данных и ключами трех размеров (128, 192 и 256 разрядов).

Перед первым туром конкурса в NIST поступило 21 предложение, 15 из которых соответствовали выдвинутым критериям. 2 октября 2000 года NIST сообщил о своем выборе, победителем конкурса стал алгоритм RIJNDAEL [7] бельгийских криптографов Винсента Раймана и Йоана Дамана, который в настоящее время зарегистрирован в качестве официального федерального стандарта как FIPS 197 (Federal Information Processing Standard).

AES доступен в различных пакетах шифрования и одобрен Агенством Национальной Безопасности (АНБ) для шифрования секретной информации.

2.2. Криптостойкость

В июне 2003 года Агентство национальной безопасности США постановило, что шифр AES является достаточно надёжным, чтобы использовать его для защиты сведений, составляющих государственную тайну (англ. classified information). Вплоть до уровня SECRET было разрешено использовать ключи длиной 128 бит, для уровня TOP SECRET требовались ключи длиной 192 и 256 бит.

3.3. Основные процедуры алгоритма AE

Входным и выходным потоком для алгоритма AES являются последовательности из 128 бит. Ключ шифрования для алгоритма AES — последовательность из 128, 192, или 256 бит. Другие длины входного и выходного потока и ключа шифрования не допускаются стандартом. Алгоритм состоит из 10, 12, или 14 раундов. На каждом раунде используется функция раунда.

128 бит входного блока группируются в 16 байт $a_0 .. a_{16}$. Затем эти байты записываются в матрицу состояния размером 4 * 4:

$$\begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

Процедуры, входящие в функцию раунда:

1) сдвиг строк матрицы состояния на различную величину (ShiftRows);

2) перемешивание данных каждого столбца матрицы состояния (SubBytes);

3) умножение каждого столбца матрицы состояния на фиксированный многочлен в $GF(2^8)$ (MixColums);

4) сложение ключа раунда с матрицей состояния (AddRoundKey).

Эти преобразования (и обратные к ним) описаны в пунктах 3.3.1 — 3.3.4.

В дальнейшем будет рассматриваться конечное поле $GF(2^8)$. Модель конечного поля $GF(2^8)$ зависит от выбора неприводимого многочлена степени 8. Для AES этим многочленом является:

$$d(x) = x^8 + x^4 + x^3 + x + 1 \qquad (3.1)$$

3.3.1. SubBytes

Процедура SubBytes (рис. 1) обеспечивает замену байт матрицы состояния с помощью таблицы (AES_sbox). Байт представляют в виде двух шестнадцатиричных цифр. Левая цифра определяет строку, а правая — столбец таблицы подстановки. Например, если входной байт 100001011, мы находим элемент на пересечении 8 строки и 11 колонки. Это элемент 61, в

двоичном виде 111101. Это выходной байт процедуры SubBytes. Таблица AES_sbox приведена в приложении 1.

Результат процедуры SubBytes можно получить в результате замены каждого байта матрицы состояния по формуле $Ax^{(-1)} + b$ в поле поле Галуа $GF(2^8)$. Обратный к x элемент можно вычислить, возведя x в 254 степень: $x^{-1} = x^{254}$

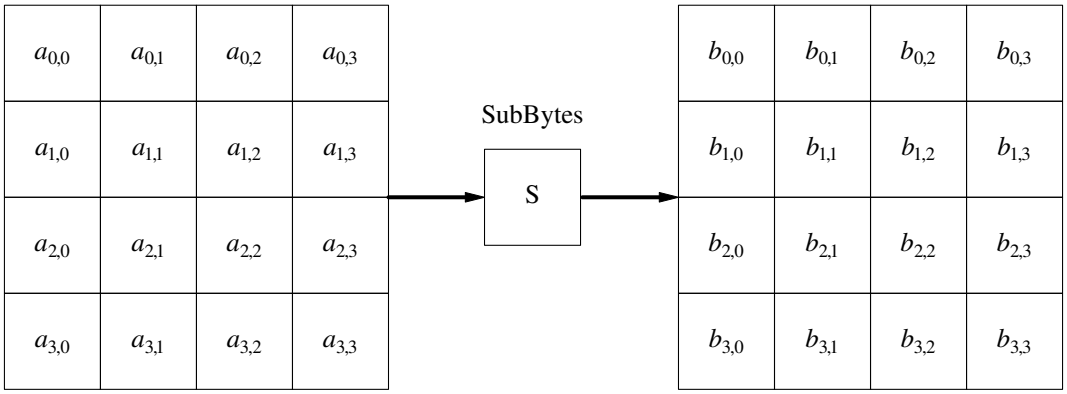

Рис. 1. Процедура SubBytes

3.3. . i t o s

Процедура ShiftRows (рис. 2) обеспечивает сдвиг строк матрицы состояния на различную величину. Строка j сдвигается на j байт (0 строка не изменяется, 3 строка сдвигается на 3 байта).

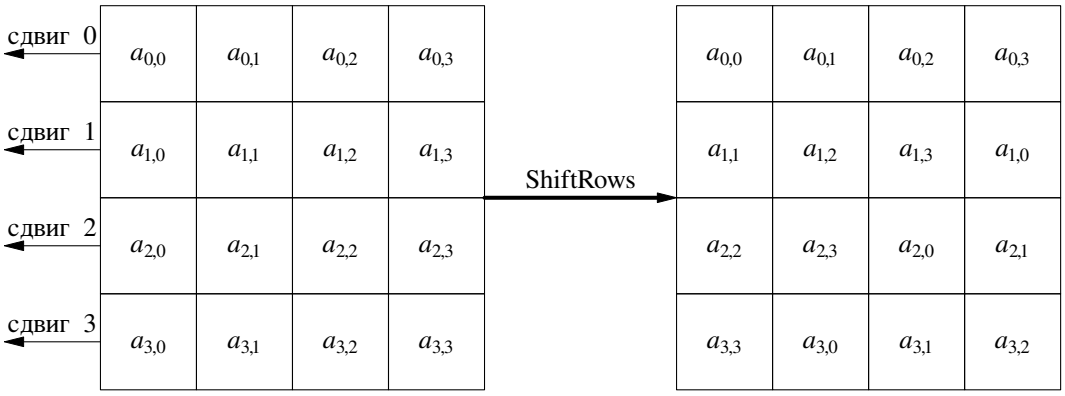

Рис. 2. Процедура ShiftRows

3.3.3. MixColumns

В процедуре MixColumns (рис. 3) каждый столбец матрицы состояния перемножается с фиксированным многочленом c(x) по модулю $d(x)$ (3.1) в поле Галуа GF(2^8). Умножение на матрицы состояния на многочлен $c(x)$ описано выше (стр. 17). В алгоритме AES многочлен c(x) имеет вид:

$$c(x) = [03]x^3 + [01]x^2 + [01]x + [02] \tag{3.2}$$

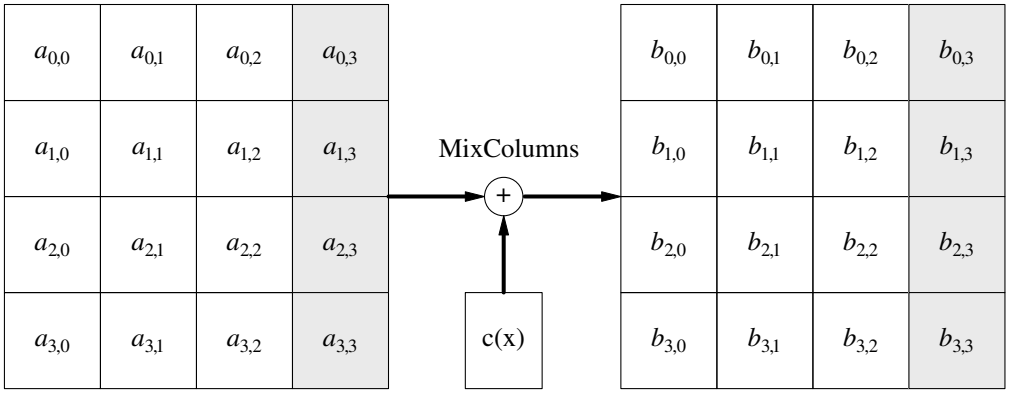

Рис. 3. Процедура MixColumns

3.3.4. AddRoundKey

В процедуре AddRoundKey (рис. 4) байт матрицы состояния складывается в поле GF(2^8) с соответствующим байтом ключей раунда (RoundKey). Ключ ранда получается из секретного ключа (CipherKey) с помощью процедуры расширения ключа. Результат — новый байт в матрице состояния.

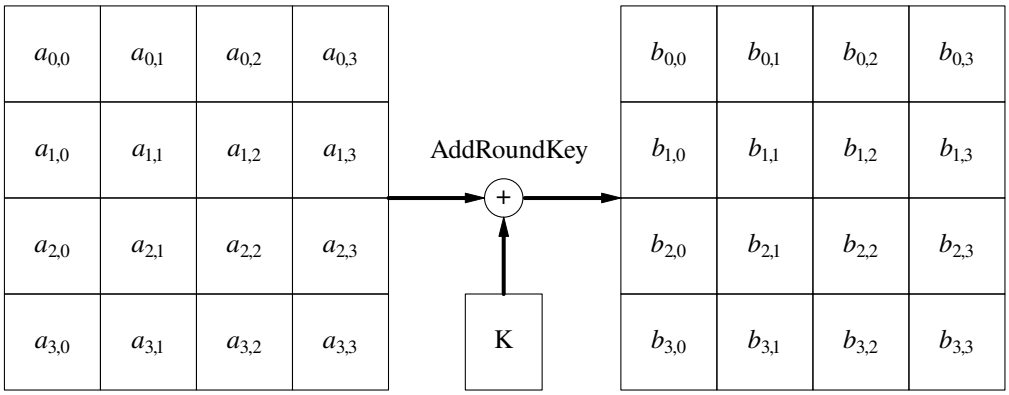

Рис. 4. Процедура AddRoundKey

3.4. Процедура расширения ключа

Процедура расширения ключа используется в алгоритме AES для создания массива подключей из ключа шифрования K. Расширение ключа выполняется в два этапа. На первом этапе выполняется инициализация слов расширенного ключа: первые Nk слов расширенного ключа заполняются ключом шифрования. Остальные слова W_i формируются по следующему алгоритму:

1) в переменную *temp* записывается предыдущее слово W_{i-1};

2) если i кратно Nk, в *temp* записывается значение SubWord (RotWord(temp) ⊕ Rcon [i/Nk]);

3) если $Nk \equiv 8$ и $(i \ mod \ Nk) == 4$, то в *temp* записывается значение $SubWord(temp)$;

4) создается слово W_i расширенного ключа: $W_i = W_{i-Nk} \oplus temp$.

Функция SubWord выполняет над каждым байтом входного значения табличную замену с помощью таблицы AES_sbox. Функция RotWord выполняет циклический сдвиг слова на 1 байт влево. Константы Rcon представляют собой слова, в которых все байты, кроме первого, являются нулевыми, а первый байт имеет значение $2^{n-1} \ mod \ 256$. Далее представлен псевдокод для процедуры расширения ключа:

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
  word temp

  i = 0;

  while (i < Nk)
  w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
  i = i+1
  end while

  i = Nk

  while (i < Nb * (Nr+1))
```

```
  temp = w [i-1]

  if (i mod Nk = 0)
  temp = SubWord (RotWord(temp) XOR Rcon [i/Nk]
  else if (Nk > 6 and i mod Nk = 4)
  temp = SubWord(temp)
  end if

  w[i] = w [i-Nk] XOR temp
  i = i + 1
  end while
end
```

Псевдокод для процедуры расширения ключа

3.5. Процедура шифрования

В начале процедуры шифрования входной поток копируется в матрицу состояния. После этого происходит первое сложение матрицы состояния и ключа раунда с помощью процедуры AddRoundKey. Затем матрица состояния преобразуется с помощью функции раунда *Nr* раз в зависимости от длины ключа. На последнем раунде процедура MixColumns не выполняется. Далее представлен псевдокод для процедуры шифрования:

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb* (Nr+1)])
begin
byte state[4, Nb]

state = in

AddRoundKey(state, w[0, Nb-1])

for round 1 step 1 to Nr-1
SubBytes(state)
ShiftRows(state)
MixColumns(state)
```

```
AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
end for

SubBytes(state)
ShiftRows(state)
AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

out = state
end
```

Псевдокод для процедуры шифрования

3.6. Процедура дешифрования

Описанные в подразделе 3.3.1 — 3.3.4 преобразования, составляющие процедуру шифрования, могут быть инвертированы и применены в обратном порядке для получения процедуры дешифрования алгоритма

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb* (Nr+1)])
begin
byte state[4, Nb]

state = in

AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

for round Nr-1 step -1 downto -1
InvSubBytes(state)
InvShiftRows(state)
InvMixColumns(state)
AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
end for

InvSubBytes(state)
InvShiftRows(state)
AddRoundKey(state, w[0, Nb-1])

out = state
end
```

Псевдокод для процедуры дешифрования

4. Хеш-функция WHIRLPOOL

4.1. История WHIRLPOOL

WHIRLPOOL [13] – криптографическая хеш-функция, разработанная Винсентом Риджменом и Пауло Баретто. Впервые опубликована в ноябре 2000 года. Осуществляет хеширование входного сообщения с длиной до 2^{256} бит. Ообрена европейской организацией NESSIE (New European Schemes for Signature, Integrity and Encryption – Новые европейские схемы подписей, целостности и шифрования)

С момента создания в 2000 году WHIRLPOOL дважды подвергалась модификации. Первая версия, WHIRLPOOL-0, была представлена в качестве кандидата в проекте NESSIE в 2000 году.

Модификация WHIRLPOOL-0, названная WHIRLPOOL-T, в 2003 году была добавлена в перечень рекомендованных к использованию криптографических функций NESSIE. Изменения касались блока подстановки WHIRLPOOL_sbox: в первой версии структура WHIRLPOOL_sbox не была описана, и он генерировался совершенно произвольно. В новой версии WHIRLPOOL_sbox приобрёл чёткую структуру.

Конечная версия WHIRLPOOL была принята в стандарте ISO/IEC 10118-3:2004 в 2004 году.

4.2. Криптостойкость

В 2004 году Тайзо Сираи и Кёдзи Сибутани обнаружили дефект в диффузных матрицах WHIRLPOOL [14], в последствии он был исправлен.

В 2009 году был опубликован новый способ атаки на хеш-функции — The Rebound Attack [10]. Удалось сгенерировать коллизию для 4,5 раундов WHIRLPOOL со сложностью 2^{120}. Однако, данная атака не сильно повлияла на криптостойкость WHIRLPOOL, сложность вычислений оказалась достаточно высока. На сегодняшний день хеш-функция WHIRLPOOL устойчива ко всем видам криптоанализа.

4.3. Схема Миагучи—Пренеля

WHIRLPOOL использует структуру Меркля—Дамгарда [8] и одностороннюю функцию сжатия Миагучи—Пренеля (рис. 5) для формирования 512-битного блока зашифрованого текста W. Внутренний блок шифрования W работает с 512-битными входными сообщениями.

Входное сообщение дополняется до нужной длины и разбивается на 512-битовые блоки M_1, M_2, M_k, которые используются для генерации хеш-значений H_1, H_2, H_k (H_0 — строка из 512 «0» бит). Для вычисления H_i, W хеширует M_i, используя в качестве ключа H_{i-1}, и выполняет XOR между H_{i-1} и M_i. Значением хеш-функции является H_k.

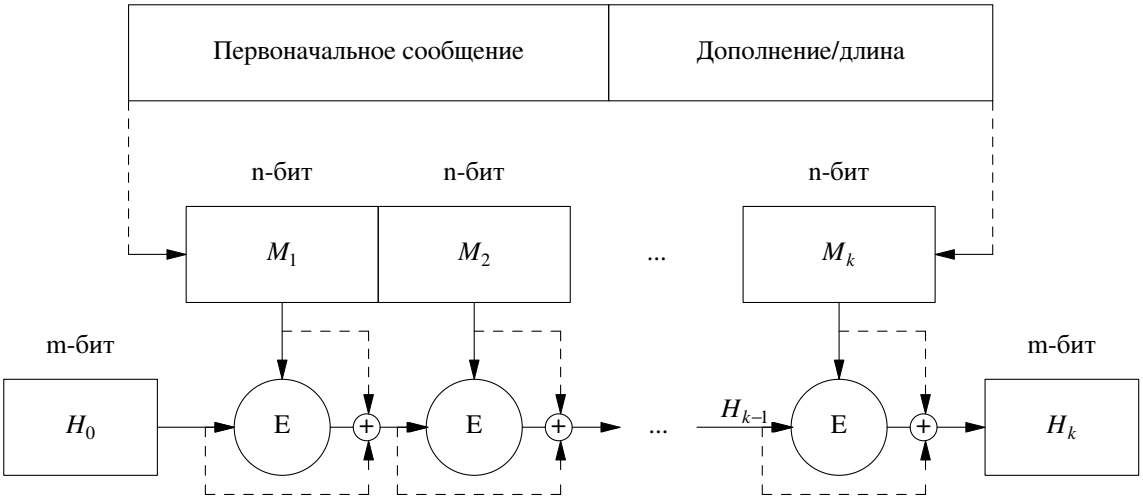

Рис. 5. Функция сжатия Миагучи—Пренеля

4.4. Дополнение сообщения

Входное сообщение разбивается на 512-битовые блоки M_1, M_2, M_t, если длина блока $M_t < 512$ бит, M_t дополянется до 512 бит. Результатом дополнения сообщения M является сообщение M', длина которого кратна 512 бит. Пусть L — длина сообщения M.

Результатом дополнения сообщения M является сообщение M', длина которого кратна 512 бит. Пусть L — длина сообщения M. Для того, чтобы получить M', необходимо сделать 3 операции:

1) к концу сообщения M приписать бит «1»;

2) приписать x битов «0» так, чтобы длина полученной строки $L + 1 + x$ была кратна 256 нечетное число раз;

3) приписать 256-битное представление числа L.

4.5. Функция раунда

Входным потоком для алгоритма WHIRLPOOL является последовательность битов произвольной длины, выходным потоком является последовательность 512 бит. Ключ хеширования для алгоритма WHIRLPOOL — последовательность из 512 бит. Алгоритм состоит из 10 раундов. На каждом раунде используется функция раунда.

512 бит входного блока группируются в 64 байта $a_0 .. a_{63}$. Затем эти байты записываются в матрицу состояния размером 8 * 8 (рис. 6):

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$
$a_{4,0}$	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	$a_{4,7}$
$a_{5,0}$	$a_{5,1}$	$a_{5,2}$	$a_{5,3}$	$a_{5,4}$	$a_{5,5}$	$a_{5,6}$	$a_{5,7}$
$a_{6,0}$	$a_{6,1}$	$a_{6,2}$	$a_{6,3}$	$a_{6,4}$	$a_{6,5}$	$a_{6,6}$	$a_{6,7}$
$a_{7,0}$	$a_{7,1}$	$a_{7,2}$	$a_{7,3}$	$a_{7,4}$	$a_{7,5}$	$a_{7,6}$	$a_{7,7}$

Рис. 6. Матрица состояния

Процедуры, входящие в функцию раунда:

1) сдвиг столбцов матрицы состояния на различную величину (ShiftColumns);

2) перемешивание данных каждого столбца матрицы состояния (SubBytes);

3) умножение каждой строки матрицы состояния на фиксированный многочлен в $GF(2^8)$ (MixRows);

4) сложение ключа раунда с матрицей состояния (AddRoundKey).

Эти преобразования описаны в пунктах 4.4.1—4.4.4. Вычисления при этом происходят в конечном поле $GF(2^8)$, полученного с помощью непри-

водимого полинома

$$d(x) = x^8 + x^4 + x^3 + x^2 + 1 \tag{4.1}$$

4.5.1. ShiftColumns

Процедура ShiftColumns (рис. 7) обеспечивает сдвиг столбцов матрицы состояния на различную величину. Столбец j сдвигается на j байт (столбец 0 не изменяется, столбец 7 сдвигается на 7 байт). В результате в матрице состояния записаны новые 8 байт.

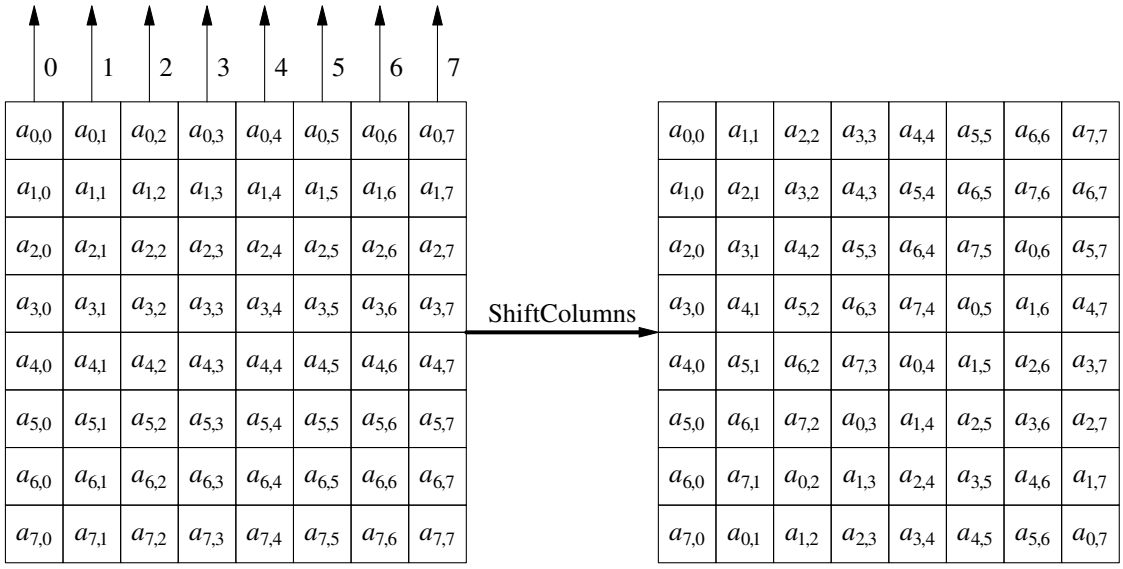

Рис. 7. Процедура ShiftColumns

4.5. . i B ts

Процедура SubBytes (рис. 8) обеспечивает замену байт матрицы состояния с помощью таблицы (WHIRLPOOL_sbox). Байт представляют в виде двух шестнадцатиричных цифр. Левая цифра определяет строку, а правая — столбец таблицы подстановки. Две шестнадцатеричные цифры на пересечении строки и столбца определяют новый байт. Например, два байта, $5A_{16}$ и $5B_{16}$, которые отличаются только одним битом, преобразованы к $5B_{16}$ и 88_{16}, которые отличаются пятью битами. Таблица WHIRLPOOL_sbox приведена в приложении 1.

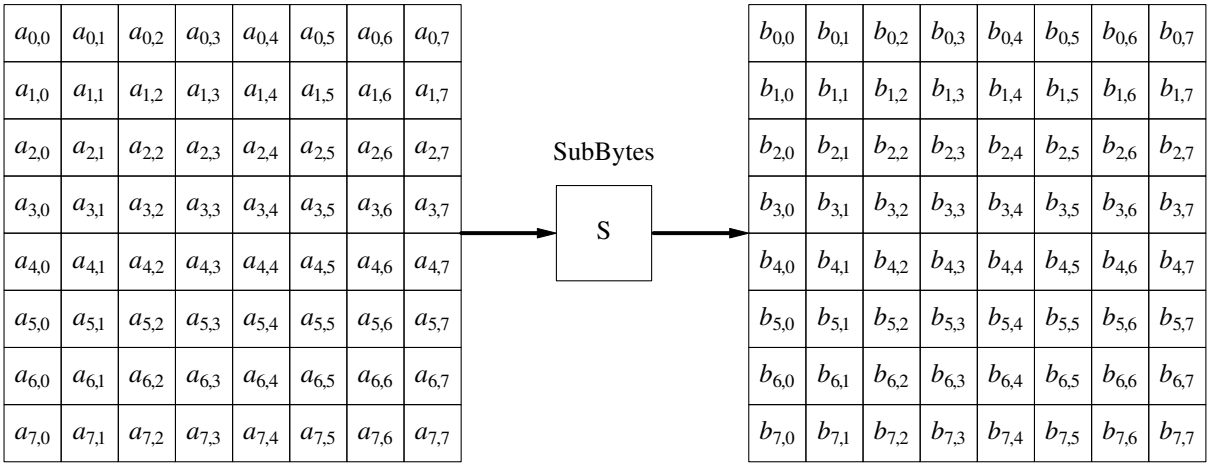

Рис. 8. Процедура SubBytes

4.5.3. MixRows

В процедуре MixRows (рис. 9) каждая строка матрицы состояния перемножается с фиксированным многочленом c(x) по модулю $d(x)$ (4.1) в поле Галуа GF(2^8). В алгоритме WHIRLPOOL многочлен $c(x)$ имеет вид:

$$c(x) = [01]x^7 + [01]x^6 + [04]x^5 + [01]x^4 + [08]x^3 + [05]x^2 + [02]x + [09] \quad (4.2)$$

Умножение на матрицы состояния на многочлен $c(x)$ подробно описано на стр. 18.

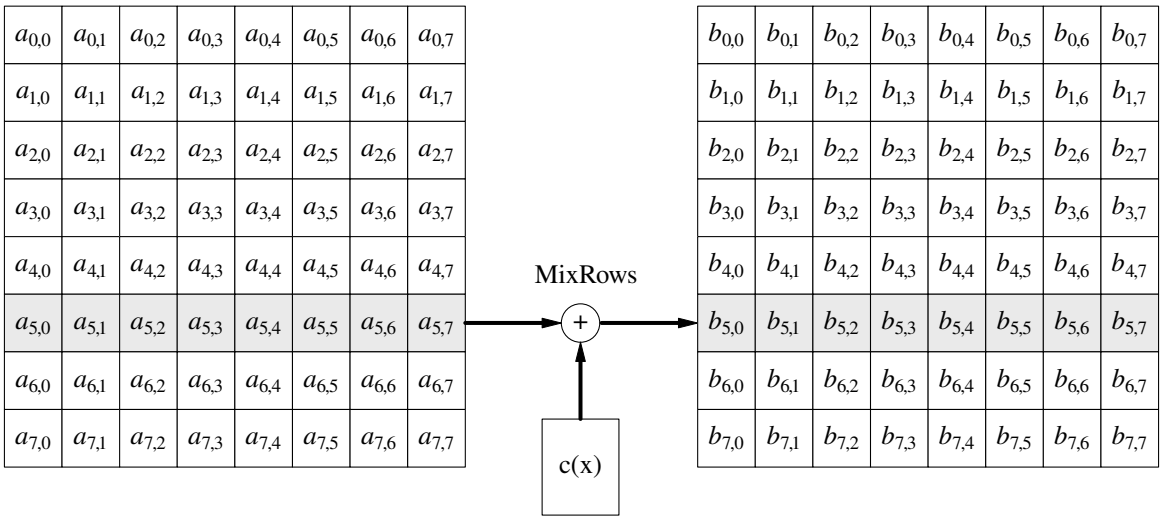

Рис. 9. Процедура MixRows

4.5.4. AddRoundKey

В процедуре AddRoundKey (рис. 10) байт матрицы состояния складывается в поле GF(2^8) с соответствующим байтом раундового ключа. Результат — новый байт матрицы состояния.

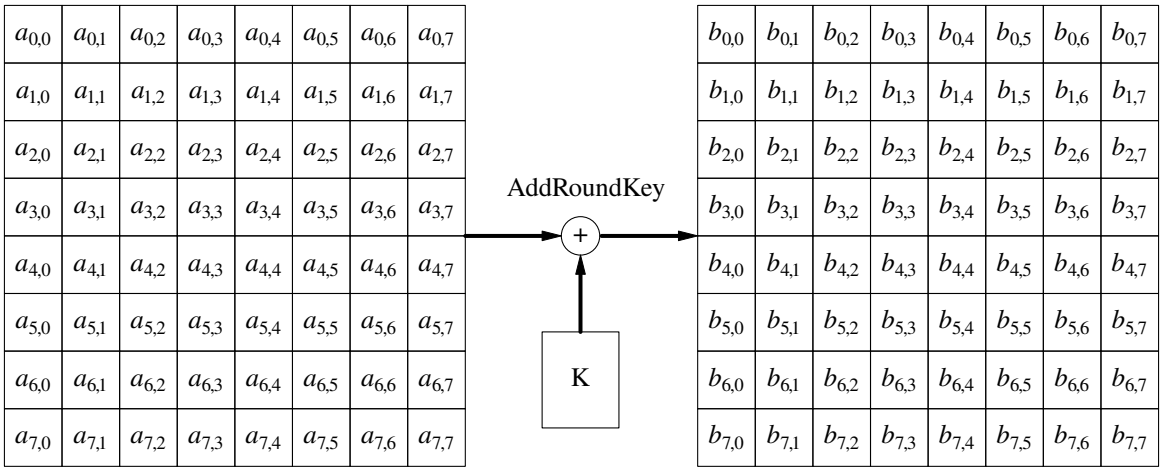

Рис. 10. Процедура AddRoundKey

4.5.5. Константы раунда

Каждая константа раунда RC^r является матрицей 8 * 8, где только первая строка имеет ненулевые значения. Остальные строки являются нулевыми. Значения для первой строки в каждой матрице констант могут быть вычислены с помощью таблицы WHIRLPOOL_sbox (RC^1 использует первую строку в таблице WHIRLPOOL_sbox, RC^2 использует вторую строку и так далее).

4.5.6. Расширение ключа

Для каждого раунда r, $0 \le r \le 10$ необходим 512-битный ключ хеширования, который получается с помощью процедуры расширения ключа. В WHIRLPOOL расширение ключа реализуется следующим образом:

$$K^0 = K,\ \ K^r = p[RC^r] * K^{r-1},\ \ r \ge 0$$

где K^r — ключ раунда, RC^r — константа раунда, а p — функция раунда. Таким образом, из известного ключа K производится необходимая последовательность $K^0 .. K^{10}$ ключей для каждого раунда блочного шифра W.

4.6. Процедура хеширования

В начале процедуры хеширования входной поток копируется в матрицу состояния, ключ хеширования копируется в массив key [8]. После этого происходит первое сложение матрицы состояния и ключа хеширования с помощью процедуры AddRoundKey. Затем матрица состояния и ключ хеширования преобразуются с помощью функции раунда 10 раз. На каждом раунде ключ хеширования складывается с константой раунда. После выполнения функции раунда ключ раунда складывается с матрицей состояния и с соответствующим байтом входного потока. Далее представлен псевдокод для процедуры хеширования.

```
word cr[10]

HASH(char in[64], word w[8])
begin
word state[8]
word key[8]

state = in
key = w

AddRoundKey(state, key)

for round 0 step 1 to 10
SubBytes(key)
SubBytes(state)

ShiftRows(key)
ShiftRows(state)

MixColumns(key)
MixColumns(state)

key[0] ^= cr[round]

AddRoundKey(state,key)
end for

for round 0 step 1 to 8
w[round] ^= state[round] ^ ((word) in)[i]

end
```

Псевдокод для процедуры хеширования

5. Архитектура ARM

Архитектура ARM (Advanced RISC Machine, Acorn RISC Machine, усовершенствованная RISC-машина) [4] — семейство лицензируемых 32-битных и 64-битных микропроцессорных ядер разработки компании ARM Limited. Компания занимается разработкой ядер и инструментов для них (компиляторы, средства отладки и так далее).

В 2007 году около 98% мобильных телефонов были оснащены процессором ARM. По состоянию на 2009 на процессоры ARM приходилось до 90 % всех встроенных 32-разрядных процессоров. Процессоры ARM широко используются в потребительской электронике, в том числе на мобильных телефонах, цифровых носителях и плеерах, портативных игровых консолях, калькуляторах и компьютерных периферийных устройствах (жесткие диски, маршрутизаторы). Эти процессоры имеют низкое энергопотребление, поэтому находят широкое применение во встраиваемых системах и преобладают на рынке мобильных устройств, для которых данный фактор немаловажен.

Архитектура развивалась с течением времени, и, начиная с ARMv7, были определены три профиля [4]:

«A» (application) — приложения;

«R» (real time) — в реальном времени;

«M» (microcontroller) — микроконтроллер.

Каждый ARM процессор создан из следующих блоков:

1. 37 регистров (из которых видимых при разработке только 17);

2. арифметико-логическое устройство (АЛУ), выполняющее арифметические и логические задачи;

3. barrel shifter — устройство, созданное для перемещения блоков данных на определенное количество бит;

4. CP15 — система, контролирующая ARM сопроцессоры;

5. декодер инструкций, занимающийся преобразованием инструкции в последовательность микроопераций.

5.1. Основы синтаксиса ARM ассемблера

В ARM разрешены следующие имена регистров:

- r0-r15
- a1-a4
- v1-v8 (переменные регистры, с r4 по r11)
- sb and SB (статический регистр, r9)
- sl and SL (r10)
- fp and FP (r11)
- ip and IP (r12)
- sp and SP (r13)
- lr and LR (r14)
- pc and PC (программный счетчик, r15).

В ARM ассемблере, как и любом другом языке программирования могут использоваться переменные и константы. Они разделяются на следующие типы:

- Числовые
- Логические
- Строковые

Числовые переменные инициализируются так:

```
a SETA 100;
```

Создается числовая переменная «a» с значением «100».

Строковые переменные:

```
improb SETS "literal";
```

Создается переменная «improb» с значение «literal». Значение переменной не может превышать 5120 символов.

В логических переменных соответственно используются значения «TRUE» и «FALSE».

ARM NEON — расширенный набор инструкций для процессоров с ARM архитектурой. Технология NEON ускоряет обработку видео, 2D/3D графики, игр, изображений, телефонии и синтеза звука. Использование этой

технологии в разы увеличивает производительность процессоров ARM, её поддерживает подавляющее большинство потребительских устройств.

Регистры NEON:

32 64-битных регистра d_0-d_{31}

16 128-битных регистров q_0-q_{15}

q_n определяет регистры d_{2n} и d_{2n+1}.

Большинство команд из набора NEON подразумевает параллелизм обработки на уровне данных. Так, соответствующие команды сложения могут обрабатывать до 16 байт одновременно.

Практическая часть

В референсной версии нелинейное (SubBytes) и диффузное (MixColumns, MixRows) преобразования выполняются с помощью таблиц подстановок.

В реализации с защитой от атак на кэш таблицы подстановки заменены вычислениями (именно поэтому она является медленной). В данной реализации AES процедура SubBytes не использует таблиц подстановки, новое значение байта x вычисляется с помощью обратного элемента x^{-1} в GF(2^8). В WHIRLPOOL новое значение байта вычисляется с помощью 3 миниблоков: E, E^{-1} и R. Диффузное преобразование в алгоритме AES и WHIRLPOOL выполняется без таблиц подстановки, элементы матрицы состояния перемножаются с многочленом c(x) по модулю d(x) в GF(2^8).

Основной идеей для ассемблерной реализации WHIRLPOOL является распараллеливание вычислений на уровне данных в регистрах NEON. Нелинейное и диффузное преобразования выполняются не для одной, а для двух строк матрицы состояния, по тому же алгоритму что и в реализации с защитой от атак на кэш.

1. Требования к программам

Целью дипломного проекта является разработка пакета программ для шифрования и хеширования данных. Разрабатываемые программы должны выполнять следующие функции:

1. Предоставлять возможность ввода информации для шифрования и хеширования.

2. Предоставлять информацию о времени работы программы, ключе шифрования или хеширования,зашифрованном сообщении или хеше сообщения, режиме шифрования.

Входными данными для программ является:

— текстовое сообщение или имя файла для шифрования или хеширования.

Выходными данными программ являются:

— текстовое сообщение с информацией о зашифрованом или хешированом сообщении, или имя файла зашифрованного сообщения;

— время работы программы.

Все программы должны быть написаны на языке C и на языке ассембера ARMv7. Входные данные для программ должны передаваться с помощью аргумента командной строки, или с помощью файла, имя файла передается с помощью командной строки.

Программы для шифрования должны выводить в стандартный поток сообщение о завершении процесса, или сообщение об ошибке, и время работы программы.

Программы для хеширования должны выводить в стандартный поток результат вычислений в следующих двух форматах.

А) Формат по умолчанию. Предполагает вывод четырёх строк, каждая из которых содержит сокращённое название вызванной реализации, вычисленное значение хеш-функции в шестнадцатеричном виде (от младших байт дайджеста к старшим) и время в долях секунды, потраченное на вычисления. Каждая строка должна заканчиваться символом перевода строки.

Б) Сокращённый формат, включаемый ключом командной строки. Предполагает вывод четырёх строк, содержащих значение хеш-функции как в пункте А и символ перевода строк

2. Референсная реализация AES

2.1. SubBytes

Для реализации процедуры SubBytes с таблицей подстановок была написана процедура sub_bytes. 16 байт матрицы состояния заменяются с помощью таблицы AES_sbox.

```
void sub_bytes(uint32_t *state) {
uint8_t *temp =  (uint8_t *)(state);

for (int i = 0; i < BLOCK_NBYTES; i++)
temp[i] = AES_sbox[temp[i]];
}
```

Для реализации процедуры InvSubBytes с таблицей подстановок была написана процедура inv_sub_bytes. 16 байт матрицы состояния заменяются с помощью таблицы INV_AES_sbox.

```
void inv_sub_bytes(uint32_t *state) {
uint8_t *temp = (uint8_t *)(state);

for (int i = 0; i < BLOCK_NBYTES; i++)
temp[i] = INV_AES_sbox[temp[i]];
}
```

2.2. MixColumns

Для реализации процедуры MixColumns с таблицами подстановок M0, M1, M2 и M3 была написана процедура mix_columns:

```
static void mix_columns(uint32_t *state) {

/*запоминаем 1 элемент каждого столбца, заменяем его с помощью
 таблиц подстановки, складываем результат замены, и переходим
 к следующему элементу в столбце:
*/

uint8_t *s0, *s1, *s2, *s3;
uint32_t res;

s0 = (uint8_t *) state;
s1 = (uint8_t *) state + 4;
s2 = (uint8_t *) state + 8;
s3 = (uint8_t *) state + 12;

for (i = 0, i < 4; i++) {

res = M0[*s0] ^ M1[*s1] ^ M2[*s2] ^ M3[*s3];
*s0++ = res & 0xff;
*s1++ = (res & 0xff00) >> 8;
*s2++ = (res & 0xff0000) >> 16;
*s3++ = (res & 0xff000000) >> 24;
}
```

Для реализации процедуры InvMixColumns с таблицами подстановок I0, I1, I2 и I3 была написана процедура inv_mix_columns:

```
static inv_void mix_columns(uint32_t *state) {

/*запоминаем 1 элемент каждого столбца, заменяем его с помощью
 таблиц подстановки, складываем результат замены, и переходим
 к следующему элементу в столбце:
*/
uint8_t *s0, *s1, *s2, *s3;
uint32_t res;

s0 = (uint8_t *) state;
s1 = (uint8_t *) state + 4;
s2 = (uint8_t *) state + 8;
s3 = (uint8_t *) state + 12;

for (i = 0, i < 4; i++) {

res = I0[*s0] ^ I1[*s1] ^ I2[*s2] ^ I3[*s3];
*s0++ = res & 0xff;
*s1++ = (res & 0xff00) >> 8;
*s2++ = (res & 0xff0000) >> 16;
*s3++ = (res & 0xff000000) >> 24;
}
}
```

2.3. ShiftRows

Процедура ShiftRows выполняет циклический сдвиг столбцов матрицы состояния влево:

```
void shift_rows(uint32_t* state) {

int i;
uint32_t *s = state;

s[1] = s[1] << 8 | s[1] >> 24;
s[2] = s[2] << 16 | s[2] >> 16;
s[3] = s[3] >> 8 | s[3] << 24;
```

Процедура InvShiftRows выполняет циклический сдвиг столбцов матрицы состояния вправо:

```
void inv_shift_rows(uint32_t* state) {

int i;
uint32_t *s = state;

s[1] = s[1] >> 8 | s[1] << 24;
s[2] = s[2] >> 16 | s[2] << 16;
s[3] = s[3] << 8 | s[3] >> 24;
```

2.4. AddRoundKey

Процедура AddRoundKey производит побитовый *XOR* каждого байта State с каждым байтом открытого ключа раунда w:

```
void add_round_key(struct AES_context *ctx, int round) {
uint8_t *wp;
uint8_t *temp;

wp = (uint8_t *)(ctx->w + AES_NB * round);
temp = (uint8_t *)(ctx->state);

for (int i = 0; i < AES_NB; i++) {
for (int j = 0; j < AES_NB; j++)
temp[4 * i + j] ^= wp[4 * j + i];
}
}
```

3. Реализация AES с защитой от атак на кэш

3.1. S-box

Для генерации таблицы AES_sbox необходимы процедуры умножения полиномов в поле GF(2^8) и нахождения обратного элемента (см. гл. 2, стр. 15). Для умножения полиномов в поле GF(2^8) была написана функция gmul:

```
#define GENERATING_POLY 0x03
#define RIJNDAEL_POLY 0x1b


uint8_t
gmul(uint8_t a, uint8_t b)
{
      uint8_t c;
      int i;

      for (i = 0, c = 0; i < 8; i++) {
            if ((b & 1) != 0)
                  c ^= a;
            b >>= 1;
            if ((a & 0x80) != 0)
                  a = (a << 1) ^ RIJNDAEL_POLY;
            else
                  a <<= 1;
      }

      return c;
}
```

Далее нам необходимо найти обратный элемент в GF(2^8). Нам дан полином $a(x)$, и нужно найти такой полином $a^{-1}(x)$, чтобы $a(x) \times a^{-1}(x) = 1$. Для нахождения $a^{-1}(x)$ необходимо $a(x)$ возвести в 254 степень. Возведение в 254 степень можно представить следующим образом:

$$a^{254} = a^2 * a^4 * a^8 * a^{16} * a^{32} * a^{64} * a^{128} \qquad (3.1)$$

Для возведения в квадрат в GF(2^8) была написана процедура gsquare:

```
#define RIJNDAEL_POLY    0x1B
#define RIJNDAEL_POLY2   0x6C
#define RIJNDAEL_POLY4   0xAB
#define RIJNDAEL_POLY8   0x09A

static uint8_t
gsquare(uint8_t a)
{
     uint8_t p;

     p = (a & 0x01) ^ ((a & 0x02) << 1) ^
               ((a & 0x04) << 2) ^ ((a & 0x08) << 3);
     if ((a & 0x10) != 0)
          p ^= RIJNDAEL_POLY;
     if ((a & 0x20) != 0)
          p ^= RIJNDAEL_POLY2;
     if ((a & 0x40) != 0)
          p ^= RIJNDAEL_POLY4;
     if ((a & 0x80) != 0)
          p ^= RIJNDAEL_POLY8;

     return p;
}
```

Используя процедуру gsquare можно быстро найти обратный элемент в GF(2^8) по формуле (3.1)

```
static uint8_t
gmul_inv(uint8_t a)
{
     uint8_t b;
     uint8_t c;
     int i;

     b = gsquare(a);
     c = b;
     for (i = 0; i < 6; i++) {
          b = gsquare(b);
          c = gmul(b, c);
     }

     return c;
}
```

Для получения нового значения байта без использования таблицы AES_sbox была написана процедура transform_bits. На вход передается байт u. Для того, чтобы получить результат процедуры transform_bits *w*, нам необходимо найти обратный элемент x и сделать побитовое сложение байтов x. В конце необходимо сделать *XOR* x и $0x63$:

$$w = x_i \oplus x_{(i+4)\ mod\ 8} \oplus x_{(i+5)\ mod\ 8} \oplus x_{(i+6)\ mod\ 8} \oplus x_{(i+7)\ mod\ 8} \oplus 0x63 \quad (3.2)$$

```
/*Макрос для циклического сдвига*/
#define BYTE_ROL(x, s)   (((x) << (s)) | ((x) >> (8 - s)))

uint8_t
transform_bits(uint8_t a)
{
     uint8_t b;

     b = gmul_inv(a);

     return b ^ BYTE_ROL(b, 1) ^ BYTE_ROL(b, 2) ^
         BYTE_ROL(b, 3) ^ BYTE_ROL(b, 4) ^ POLY_VECTOR;
}
```

Таблица подстановок INV_AES_sbox является обратной для AES_sbox. Для получения нового значения байта без изпользования таблицы INV_AES_sbox была написана процедура inv_transform_bits. На вход передается байт u. Для того, чтобы получить результат процедуры inv_transform_bits *w*, можно воспользоваться следующей формулой

$$w = (u_i \oplus u_{(i+3) \bmod 8} \oplus u_{(i+6) \bmod 8} \oplus 0x05)^{-1} \qquad (3.3)$$

```
#define POLY_VECTOR_INV 0x05

uint8_t
inv_transform_bits(uint8_t u)
{
     return gmul_inv(BYTE_ROL(u, 1) ^ BYTE_ROL(u, 3)
                     ^ BYTE_ROL(u, 6)^ POLY_VECTOR_INV);
}
```

3.2. MixColumns

Результат преобразования MixColumns можно получить по следующим формулам:

$$\begin{aligned} b_{0,c} &= ([02]\times a_{0,c}) \oplus ([03]\times a_{1,c}) \oplus a_{2,c} \oplus a_{3,c} \\ b_{1,c} &= a_{0,c} \oplus ([02]\times a_{1,c}) \oplus ([03]\times a_{2,c}) \oplus a_{3,c} \\ b_{2,c} &= a_{0,c} \oplus a_{1,c} \oplus ([02]\times a_{2,c}) \oplus ([03]\times a_{3,c}) \\ b_{3,c} &= ([03]\times a_{0,c}) \oplus a_{1,c} \oplus a_{2,c} \oplus ([02]\times a_{3,c}) \end{aligned} \qquad (3.4)$$

где $0 \le c \le Nb$ означает номер столбца, a — байт матрицы состояния после преобразования ShiftRows.

Формулы (3.4) можно представить в следующем виде:

$$b_{0,c} = ([02]\times a_{0,c}) \oplus ([02]\times a_{1,c}) \oplus a_{1,c} \oplus a_{2,c} \oplus a_{3,c} \quad (3.5)$$
$$b_{1,c} = a_{0,c} \oplus ([02]\times a_{1,c}) \oplus ([02]\times a_{2,c}) \oplus a_{2,c} \oplus a_{3,c}$$
$$b_{2,c} = a_{0,c} \oplus a_{1,c} \oplus ([02]\times a_{2,c}) \oplus ([02]\times a_{3,c}) \oplus a_{3,c}$$
$$b_{3,c} = ([02]\times a_{0,c}) \oplus a_{0,c} \oplus a_{1,c} \oplus a_{2,c} \oplus ([02]\times a_{3,c})$$

где $0 \le c \le Nb$ означает номер столбца, a — байт матрицы состояния после преобразования ShiftRows.

Мы можем посчитать результат $a(x)\times c(x)$ без таблиц подстановок. Для этого была написана процедура mix_columns_slow.

```
/* Умножает каждый байт массива на 2 в поле Rijndael */
void
g2times(uint8_t *p, size_t n)
{
    int i;

    for (i = 0; i < n; i++)
        if ((p[i] & 0x80) != 0)
            p[i] = (p[i] << 1) ^ RIJNDAEL_POLY;
        else
            p[i] <<= 1;
}

static void
vec4_uint32_xor(uint32_t dest[4], uint32_t src[4])
{
    dest[0] ^= src[0];
    dest[1] ^= src[1];
    dest[2] ^= src[2];
    dest[3] ^= src[3];
}
```

```
static void
mix_columns_slow(uint32_t *state)
{
    uint8_t buf[28];

    memcpy(buf, state, 16);
    memcpy(buf + 16, state, 12);

    /* state = [s4,.., s15, s0,.., s3] */
    memcpy(state, buf + 4, 16);

    /* state ^= [s8,.., s15, s0,.., s7] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 8));

    /* state ^= [s12,.., s15, s0,.., s11] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 12));

    /* Умножение на 2 в поле Rijndael */
    g2times(buf, 20);

    /* state ^= [2 * s0,.., 2 * s15] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 0));

    /* state ^= [2 * s4,.., 2 * s15, 2 * s0,.., 2 * s3] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 4));
}
```

Преобразование InvMixColums можно получить по следующим формулам:

$$b_{0,c} = ([0x14]\times a_{0,c}) \oplus ([0x11]\times a_{1,c}) \oplus ([0x13]\times a_{2,c}) \oplus ([0x09]\times a_{3,c}) \quad (3.6)$$
$$b_{1,c} = ([0x09]\times a_{0,c}) \oplus ([0x14]\times a_{1,c}) \oplus ([0x11]\times a_{2,c}) \oplus ([0x13]\times a_{3,c})$$
$$b_{0,c} = ([0x13]\times a_{0,c}) \oplus ([0x09]\times a_{1,c}) \oplus ([0x14]\times a_{2,c}) \oplus ([0x11]\times a_{3,c})$$
$$b_{0,c} = ([0x11]\times a_{0,c}) \oplus ([0x13]\times a_{1,c}) \oplus ([0x09]\times a_{2,c}) \oplus ([0x14]\times a_{3,c})$$

Мы можем сделать замену в формуле (3.6) по формулам (3.7):

$$([0x09]\times a_{i,c}) = ([0x08]\times a_{i,c}) \oplus (a_{i,c}) \quad (3.7)$$
$$([0x11]\times a_{i,c}) = ([0x09]\times a_{i,c}) \oplus ([0x02]\times a_{i,c})$$
$$([0x13]\times a_{i,c}) = ([0x09]\times a_{i,c}) \oplus ([0x04]\times a_{i,c})$$
$$([0x14]\times a_{i,c}) = ([0x08]\times a_{i,c}) \oplus ([0x04]\times a_{i,c}) \oplus ([0x02]\times a_{i,c})$$

где $0 \le c \le Nb$ означает номер столбца, a — байт матрицы состояния после преобразования InvShiftRows.

Процедура inv_mix_columns_slow аналогична mix_columns_slow, меняется только матрица $c(x)$. Ниже приведен пример процедуры inv_mix_columns_slow:

```
static void
inv_mix_columns_slow(uint32_t *state)
{
    uint8_t buf[28];

    memcpy(buf, state, 16);
    memcpy(buf + 16, state, 12);

    /* state = [s4,.., s15, s0,.., s3] */
    memcpy(state, buf + 4, 16);

    /* state ^= [s8,.., s15, s0,.., s7] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 8));

    /* state ^= [s12,.., s15, s0,.., s11] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 12));
```

```
    /* Умножение на 2 в поле Rijndael */
    g2times(buf, 28);

    /* state ^= [2 * s0,.., 2 * s15] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 0));

    /* state ^= [2 * s4,.., 2 * s15, 2 * s0,.., 2 * s3] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 4));

    /* Умножение на 2 в поле Rijndael */
    g2times(buf, 28);

    /* state ^= [4 * s0,.., 4 * s15] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 0));

    /* state ^= [4 * s8,.., 4 * s15, 4 * s0,.., 4 * s7] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 8));

    /* Умножение на 2 в поле Rijndael */
    g2times(buf, 28);

    /* state ^= [8 * s0,.., 8 * s15] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 0));

    /* state ^= [8 * s4,.., 8 * s15, 8 * s0,.., 8 * s3] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 4));

    /* state ^= [8 * s8,.., 8 * s15, 8 * s0,.., 8 * s7] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 8));

    /* state ^= [8 * s12,.., 8 * s15, 8 * s0,.., 8 * s11] */
    vec4_uint32_xor(state, (uint32_t *) (buf + 12));
}
```

3.3. Генерация таблиц подстановок для MixColumns

Для умножения 2 полиномов $a(x)\times c(x)$ в поле Галуа GF(2^8) была написана процедура gmul. Для алгоритма AES матрица $c(x)$ для MixColumns выглядит следующим образом:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Для алгоритма AES матрица c(x) для InvMixColumns выглядит следующим образом:

$$\begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

Для генерации таблиц подстановки для матрицы state была написана процедура create_mix_tables. Дальнейшие действия описаны для одного столбца матрицы состояния, для остальных столбцов действия аналогичны. Объявляем макросы P0, и Q0 для умножения столбца на матрицу $c(x)$. Макросы используются в процедуре create_mix_tables:

```
#define P0(x)
(gmul((x), 0x2) | ((x) << 8) |
(gmul((x), 0x3) << 24)) | ((x) << 16)


#define Q0(x)
(gmul((x), 0xe) | (gmul((x), 0x9) << 8) |
(gmul((x), 0xd) << 16)  |(gmul((x), 0xb) << 24))


static uint32_t M0[256];
static uint32_t I0[256];
```

```
void create_mix_tables()
{
   int x;

   for (x = 0; x < 256; x++) {
       M0[x] = P0(x);
       I0[x] = Q0(x);

   }
}
```

После выполнения процедуры create_mix_tables получаем таблицы подстановок M0 и I0 для одного столбца матрицы состояния. M0 используется для процедуры MixColumns, I0 — для процедуры InvMixColumns.

4. Ассемблерная реализация AES

4.1. Sub Bytes

Процедура реализует следующий фрагмент кода с помощью ассемблерной команды vtbx:

```
for (i = 0; i < 16; i++)
      state[i] = sbox[state[i]];
```

Таблица sbox задана, state — 16-байтовый массив состояния, который целиком загружается в регистр q0. Таблица sbox содержит 256 байт, для индексации сначала загрузим первую половину таблицы, потом вторую. Загружаем половину таблицы в регистры d16-d31. для получения индекса используем инструкцию vtbx. Инструкция vtbx может индексировать таблицу только из 32 записей. Разбиваем таблицу sbox на части по 32 записи. Индекс от 0 до 255 может быть найден как $32 * i + j$ где $0 \le i \le 7$ и $0 \le j \le 31$. Для каждого $0 \le i \le 7$ мы используем i блок из sbox. На i итерации мы вычитаем $32 * i$ из индексов.

```
static void
sub_bytes_asmv(uint32_t *state) {
asm(
  "mov r0, %[sbox]"          // Сохраняем указатель на sbox
  "vldm %[state], {q0}"       // загружаем state в q0
  "vmov. u8 q7, #32"          // в q7 записываем 32
  "vldm r0!, {d16-d31}"       // загружаем 1 половину sbox

  "vtbx. 8 d4, {d16-d19}, d0"   // индексы меняются от 0 до 31
  "vtbx. 8 d5, {d16-d19}, d1"   // сохраняем результат в q2

  "vsub. i8 q0, q7"           // вычитаем 32 из индексов
  "vtbx. 8 d4, {d20-d23}, d0"   // индексы от 32 до 63
  "vtbx. 8 d5, {d20-d23}, d1"

  "vsub. i8 q0, q7"
  "vtbx. 8 d4, {d24-d27}, d0"   // индексы от 64 до 95
  "vtbx. 8 d5, {d24-d27}, d1"
```

```
  "vsub. i8 q0, q7"
  "vtbx. 8 d4, {d28-d31}, d0"   // индексы от 96 до 127
  "vtbx. 8 d5, {d28-d31}, d1"

  "vldm r0, {d16-d31}"        // загружаем 2 половину sbox

  "vsub. i8 q0, q7"
  "vtbx. 8 d4, {d16-d19}, d0"   // индексы от 128 до 159
  "vtbx. 8 d5, {d16-d19}, d1"

  "vsub. i8 q0, q7"
  "vtbx. 8 d4, {d20-d23}, d0"   // индексы от 160 до 191
  "vtbx. 8 d5, {d20-d23}, d1"

  "vsub. i8 q0, q7"
  "vtbx. 8 d4, {d24-d27}, d0"   // индексы от 192 до 223
  "vtbx. 8 d5, {d24-d27}, d1"

  "vsub. i8 q0, q7"
  "vtbx. 8 d4, {d28-d31}, d0"   // индексы от 224 до 255
  "vtbx. 8 d5, {d28-d31}, d1"

  "vstm %[state], {q2}"       // в q2 содержится результат
          :
          : [state] "r" (state), [sbox] "r" (sbox)
          : "r0", "q0", "q2", "q7", "q8", "q9", "q10",
            "q11", "q12", "q13", "q14", "q15");

}
```

4.2. MixColumns

Алгоритм умножения матрицы состояния на $c(x)$ в GF(2^8) описан в разделе 3.2 практической части:

```
static void
mix_columns_asmv(uint32_t *state) {
asm(
   "vldm. 64 %[state], {q0}"  //  s0,.., s15 загружаем q0

   "vmov. u8 q1, q0"          // в q1 будет  2 * s0,.., 2 * s15

   "vmov. i8 q2, #0x80"       // в q2 записать 0x80
   "vmov. i8 q3, #0x1B"       // в q3 записать 0x1B

   "vtst. 8 q2, q1"
   "vand. i8 q2, q3"          // каждый байт в q2 = 0x1B,если
                                старший байт = 1 или 0x00,
                                если байт = 0
   "vshl. u8 q1, #1"          // Циклический сдвиг на 1 бит влево

   "veor. i8 q1, q2"          // XOR q1 и q2
                                // q1 = 2 * s0,.., 2 * s15

   "vext. 8 q2, q1, q1, #4"   // q2 = 2 * s4,.., 2 * s15, 2 * s0,
                                ..., 2 * s3

   "veor. 8 q1, q2"           // первый XOR

   "vext. 8 q2, q0, q0, #4"   // q2 = s4,.., s15, s0,.., s3
   "veor. 8 q1, q2""          // второй XOR

   "vext. 8 q2, q0, q0, #8"   // q2 = s8,.., s15, s0,.., s7
   "veor. 8 q1, q2"           // третий XOR

   "vext. 8 q2, q0, q0, #12" // q2 is s12,.., s15, s0,.., s11
   "veor. 8 q1, q2"           // четвертый XOR

   "vstm. 64 %[state], {q1}"
      :
      : [state] "r" (state)
      : "memory", "q0", "q1", "q2", "q3"
        );
}
```

4.3. InvMixColumns

```
static void
inv_mix_columns_asmv(uint32_t *state) {
asm(
  "vldm. 64 %[state], {q0}"        // s0,.., s15 записываем в q0


  "vmov. u8 q1, q0"                // в q1 будет храниться 2 * s0,
  "vmov. i8 q4, #0x80"             //..., 2 * s15
                                    // в q4  записываем 0x80
  "vmov. i8 q5, #0x1B"             // в q5 записываем 0x1B
```

```
"vtst. 8 q6, q1, q4"

"vand. i8 q6, q5"                    // каждый байт в q6 = 0x1B,
                                     старший байт = 1 или 0x00,
                                          если байт = 0

"vshl. u8 q1, #1"                    // сдвиг на 1 байт влево

"veor. i8 q1, q6"                    // XOR q1 и  q6
                                    // q1 = 2 * s0,.., 2 * s15

"vmov. u8 q2, q1"                    // q2 = 4 * s0,.., 4 * s15
"vtst. 8 q6, q2, q4"
"vand. i8 q6, q5"
"vshl. u8 q2, #1"
"veor. i8 q2, q6"

"vmov. u8 q3, q2"                   // в q3 записываем  8 * s0,.., 8 * s15
"vtst. 8 q6, q3, q4"
"vand. i8 q6, q5"
"vshl. u8 q3, #1"
"veor. i8 q3, q6"

                                    //  q4 = 9 * s12, 9 * s13,
                                    // 9 * s14, 9 * s15, 9 * s0,..,
                                    // 9 * s11

"vext. 8 q4, q3, q3, #12"  // q4 = 8 * s12,.., 8 * s15,
                                   // 8 * s0,.., 8 * s11

"vext. 8 q5, q0, q0, #12"  // q5 = s12,.., s15, s0,.., s11
"veor. 8 q4, q5"

"vmov. 8 q7, q4"
                          // Нужно найти 11 * s4, ..., 11 * s15,
                          // 11 * s0,.., 11 * s3
```

```
  "vext. 8 q4, q4, #8"      // q4 = 9 * s4,.., 9 * s15,
                            // 9 * s0,.., 9 * s3
  "veor. 8 q7, q4"

  "vext. 8 q5, q1, q1, #4" // q5 = 2 * s4,.., 2 * s15, 2 * s0,
                           //..., 2 * s3
  "veor. 8 q7, q5"

                           // Нужно найти 13 * s8,.., 13 * s15,
                           // 13 * s0,.., 13 * s7

  "vext. 8 q4, q4, #4"      // q4 = 9 * s8,.., 9 * s15,
                           // 9 * s0,.., 9 * s7
  "veor. 8 q7, q4"

  "vext. 8 q5, q2, q2, #8" // q5 = 4 * s8,.., 4 * s15, 4 * s0,
                           //..., 4 * s8
  "veor. 8 q7, q5"

                           //Находим 14 * s0,.., 14 * s15
  "veor. 8 q7, q3"
  "veor. 8 q7, q2"
  "veor. 8 q7, q1"

  "vstm. 64 %[state], {q7}"
    :
    : [state] "r" (state)
    : "memory", "q0", "q1", "q2", "q3", "q4", "q5",
                "q6", "q7"
        );
}
```

4.4. ShiftRows

Делаем замену байт в state с помощью команды vtbl. Индексы для замены заданы в массивах sr_indices и inv_sr_indices.

```
static const uint8_t sr_indices[] = {
      0, 1, 2, 3,
      5, 6, 7, 4,
      10, 11, 8, 9,
      15, 12, 13, 14,
};

static const uint8_t inv_sr_indices[] = {
      0, 1, 2, 3,
      7, 4, 5, 6,
      10, 11, 8, 9,
      13, 14, 15, 12,
};

static void
shift_rows_asmv(uint32_t* state, const uint8_t *indices)
{
      asm(
            "vldm. 64 %[state], {q0}"
            "vldm. 64 %[indices], {q1}"
            "vtbl. 8 d4, {d0-d1}, d2"
            "vtbl. 8 d5, {d0-d1}, d3"
            "vstm. 64 %[state], {q2}"
            :
            : [state] "r" (state), [indices] "r" (indices)
            : "memory", "q0", "q1", "q2"
         );

}
```

4.5. AddRoundKey

```
add_round_key_asmv(struct AES_context *ctx, int round)
{
      asm(
            "vldm. 64 %[state], {q0}"
            "vldm. 64 %[rkey], {q1}"
            "veor. 8 q0, q1"
            "vstm. 64 %[state], {q0}"
            :
            : [state] "r" (ctx->state),
               [rkey] "r" (ctx->w + AES_NB * round)
            : "memory", "q0", "q1"
         );
}
```

5. Референсная реализация WHIRLPOOL

5.1. SubBytes

Для реализации процедуры SubBytes с таблицей подстановок была написана процедура sub_bytes:

```
void sub_bytes(uint64_t state[8]) {

      /*Все элементы state заменяются с помощью таблицы
      WHIRLPOOL_sbox*/

      uint8_t *temp = (uint8_t *) state;
      int i;
      .
      for (i = 0; i < BLOCK_NBYTES; i++)
            temp[i] = WHIRLPOOL_sbox[temp[i]];
}
```

5.2. MixRows

Для реализации процедуры MixRows с таблицами подстановок W0, W1, W2 и W3 была написана процедура mix_rows. Так как $i+4$ строку матрицы $c(x)$ можно получить, сдвинув i строку на 4 байта вправо, где $0 \le i \le 4$, то необходимо только 4 таблицы подстановки и макрос для сдвига значения на 4 байта вправо. Для этого используется макрос R64:

```
#define R64(x)   ((x >> 32) | (x << 32))
```

Для получения нового значения из таблиц подстановки был написан макрос MIX64:

```
#define MIX64(x)
(
      W0[(x) & 0xff] ^
      W1[((x) >> 8) & 0xff] ^
      W2[((x) >> 16) & 0xff] ^
      W3[((x) >> 24) & 0xff] ^
      R64(W0[((x) >> 32) & 0xff]) ^
      R64(W1[((x) >> 40) & 0xff]) ^
      R64(W2[((x) >> 48) & 0xff]) ^
      R64(W3[((x) >> 56) & 0xff])
)
```

Окончательно процедура mix_rows приняла следующий вид:

```
void mix_rows(uint64_t state[8]){

      int i;

      for(i = 0; i < WHIRLPOOL_NB; i++)
            state[i] = MIX64(state[i]);
}tr
```

5.3. ShiftColumns

Процедура ShiftColumns выполняет циклический сдвиг столбцов матрицы состояния вправо:

```
void shift_columns(uint64_t state[8]) {

      uint8_t *temp = (uint8_t *) state;
      uint8_t b[8];
      int i, j, shift;

      for(i = 1; i < WHIRLPOOL_NB; i++){

            for(j = 0; j < WHIRLPOOL_NB; j++) {
                  shift = i + 8 * ((8 + j - i) % 8);
                  b[j] = temp[shift];
                  temp[j * 8 + i] = b[j];

            }
      }
```

5.4. AddRoundKey

Процедура AddRoundKey производит побитовый *XOR* каждого байта state с каждым байтом ключа раунда rk:

```
void add_round_key(uint64_t state[8], uint64_t rk[8]) {

      int i;

      for(i = 0; i < WHIRLPOOL_NB; i++)
            state[i] ^= rk[i];
}
```

6. Реализация WHIRLPOOL с защитой от атак на кэш

6.1. S B ts

Результат процедуры SubBytes может быть вычислен алгебраически, используя поле $GF(2^4)$ и неприводимый полином $(x^4 + x + 1)$. Для этого используются 3 миниблока: E, E^{-1} и R, их значения приведены в таблице 1.

Таблица 1. Значения миниблоков для генерации WHIRLPOOL_sbox:

u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$E(u)$	1	B	9	C	D	6	F	3	E	8	7	4	A	2	5	0
$E^{-1}(u)$	F	0	D	7	B	E	5	A	9	2	C	1	3	4	8	6
$R(u)$	7	C	B	D	E	4	9	F	6	3	8	A	2	5	1	0

Для получения нового значения байта без использования таблицы WHIRLPOOL_sbox была написана процедура transform_bits. На вход передается байт *u*, 4 сталших бита *u* заменяются с помощью миниблока *E*, 4 младших бита *u* заменяются с помощью блока E^{-1}. Результаты блоков *E* и E^{-1} заменяются с помощью блока *R*. Чтобы получить старшие 4 бита результата, выполняется *XOR* между выходами миниблоков *E* и *R*. Чтобы получить младшие 4 бита результата, выполняется *XOR* между выходами миниблоков E^{-1} и *R*.

```
char transform_bits(uint8_t u) {

      unsigned char x, y, r;

      x = ebox[(u & 0xf0) >> 4];
      y = iebox[u & 0x0f];
      r = rbox[x ^ y];

      x = x ^ r;
      y = y ^ r;

      return (ebox[x] << 4) | iebox[y];
}
```

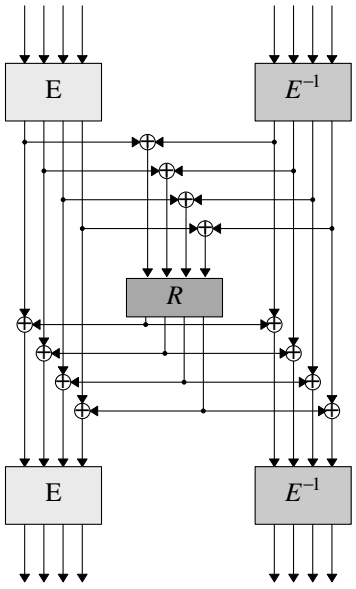

Рис. 11. Алгоритм получения результата процедуры SubBytes

Для реализации процедуры SubBytes без таблицы подстановок была написана процедура sub_bytes_slow:

```
void sub_bytes_slow(uint64_t state[8]) {
```

*/*Для всех элементов массива state выполняется процедура transform_bits()*/*

```
      uint8_t *temp = (uint8_t *) state;
      int i;

      for (i = 0; i < BLOCK_NBYTES; i++)
            temp[i] = transform_bits(temp[i]);
}
```

6.2. MixRows

Результат преобразования MixRows можно получить по следующим формулам:

$$b_{i,0} = a_{i,0} \oplus a_{i,1} \oplus ([04]\times a_{i,2}) \oplus a_{i,3} \oplus ([08]\times a_{i,4}) \oplus ([05]\times a_{i,5}) \oplus ([02]\times a_{i,6}) \oplus ([09]\times a_{i,7})$$
$$b_{i,1} = ([09]\times a_{i,0}) \oplus a_{i,1} \oplus a_{i,2} \oplus ([04]\times a_{i,3}) \oplus a_{i,4} \oplus ([08]\times a_{i,5}) \oplus ([05]\times a_{i,6}) \oplus ([02]\times a_{i,7})$$
$$b_{i,2} = ([02]\times a_{i,0}) \oplus ([09]\times a_{i,1}) \oplus a_{i,2} \oplus a_{i,3} \oplus ([04]\times a_{i,4}) \oplus a_{i,5} \oplus ([08]\times a_{i,6}) \oplus ([05]\times a_{i,7})$$
$$b_{i,3} = ([05]\times a_{i,0}) \oplus ([02]\times a_{i,1}) \oplus ([09]\times a_{i,2}) \oplus a_{i,3} \oplus a_{i,4} \oplus ([04]\times a_{i,5}) \oplus a_{i,6} \oplus ([08]\times a_{i,7})$$
$$b_{i,4} = ([08]\times a_{i,0}) \oplus ([05]\times a_{i,1}) \oplus ([02]\times a_{i,2}) \oplus ([09]\times a_{i,3}) \oplus a_{i,4} \oplus a_{i,5} \oplus ([04]\times a_{i,6}) \oplus a_{i,7}$$
$$b_{i,5} = a_{i,0} \oplus ([08]\times a_{i,1}) \oplus ([05]\times a_{i,2}) \oplus ([02]\times a_{i,3}) \oplus ([09]\times a_{i,4}) \oplus a_{i,5} \oplus a_{i,6} \oplus ([04]\times a_{i,7})$$
$$b_{i,6} = ([04]\times a_{i,0}) \oplus a_{i,1} \oplus ([08]\times a_{i,2}) \oplus ([05]\times a_{i,3}) \oplus ([02]\times a_{i,4}) \oplus ([09]\times a_{i,5}) \oplus a_{i,6} \oplus a_{i,7}$$
$$b_{i,7} = a_{i,0} \oplus ([04]\times a_{i,1}) \oplus a_{i,2} \oplus ([08]\times a_{i,3}) \oplus ([05]\times a_{i,4}) \oplus ([02]\times a_{i,5}) \oplus ([09]\times a_{i,6}) \oplus a_{i,7}$$

где $0 \le i \le 8$ и означает номер строки, a — байт матрицы состояния после преобразования ShiftRows.

Используя процедуру gmul, можно посчитать результат $a(x)\times c(x)$ без таблиц подстановок.

```
uint8_t gmul(uint8_t a, uint8_t c) {

        int high_bit_on;
        uint8_t b = 0;

        while (c) {

                if (b & 0x1)
                        b ^= a;
                high_bit_on = a & 0x80;
                a <<= 1;
```

```
            if (high_bit_on)
                    a ^= WHIRLPOOL_POLY;
            c >>= 1;
        }

        return b;
}
```

Ниже приведен пример процедуры mix_rows_slow для одной строки матрицы состояния, для остальных строк процедура аналогична.

```
void mix_rows_slow(uint32_t state) {
```

*/*мы запоминаем значение строки матрицы state*/*

```
        uint8_t *s0;
        uint8_t n0;
        int i;

        s0 = (uint8_t *) state;
```

*/*производим умножение по формулам, указаным выше, и двигаем указатель к следующей строке*/*

```
        for (i = 0; i < 8; i++) {

              n0 = *s0 ^ gmul(*s1, 0x09) ^
                gmul(*s2, 0x02) ^
                gmul(*s3, 0x05) ^
                gmul(*s4, 0x08) ^
                gmul(*s6, 0x04) ^
                ^ *s7 ^ *s5

              *s0++ = n0;
        }
}
```

6.3. Генерация таблиц подстановок для MixRows

Рассмотрим умножение $a(x)\times c(x)$ в алгоритме WHIRLPOOL. Матрица c(x) выглядит следующим образом:

$$\begin{bmatrix} 01 & 01 & 04 & 01 & 08 & 05 & 02 & 09 \\ 09 & 01 & 01 & 04 & 01 & 08 & 05 & 02 \\ 02 & 09 & 01 & 01 & 04 & 01 & 08 & 05 \\ 05 & 02 & 09 & 01 & 01 & 04 & 01 & 08 \\ 08 & 05 & 02 & 09 & 01 & 01 & 04 & 01 \\ 01 & 08 & 05 & 02 & 09 & 01 & 01 & 04 \\ 04 & 01 & 08 & 05 & 02 & 09 & 01 & 01 \\ 01 & 04 & 01 & 08 & 05 & 02 & 09 & 01 \end{bmatrix}$$

Заметим, что $i+4$ строку матрицы $c(x)$ можно получить, сдвинув строку i на 4 байта вправо, где $0 \le i \le 4$. Для генерации таблиц подстановки для первых четырех столбцов матрицы state была написана процедура create_mix_tables. Дальнейшие действия описаны для 1 и 5 строк матрицы состояния, для остальных строк действия аналогичны. Для умножения столбца матрицы состояния на соответствующую строку матрицы $c(x)$ был написан макрос T64.

```
#define T64(x, w)
((uint64_t)gmul(x, (w)[0]) |
((uint64_t)gmul(x, (w)[1])) << 8 |
((uint64_t)gmul(x, (w)[2])) << 16 |
((uint64_t)gmul(x, (w)[3])) << 24 |
((uint64_t)gmul(x, (w)[4])) << 32 |
((uint64_t)gmul(x, (w)[5])) << 40 |
((uint64_t)gmul(x, (w)[6])) << 48 |
((uint64_t)gmul(x, (w)[7])) << 56)

void create_mix_tables()  {

      uint8_t w0[] = { 0x01, 0x01, 0x04, 0x01,
                       0x08, 0x05, 0x02, 0x09};
      int x = 0;

      for (x = 0; x < 256; x++) {
            W0[x] = T64(x, w0);
      }
}
```

7. Ассемблерная реализация WHIRLPOOL

7.1. S B ts

```
const uint8_t boxes[48] = {
      /* rbox */
      0x7, 0xC, 0xB, 0xD, 0xE, 0x4, 0x9, 0xF,
      0x6, 0x3, 0x8, 0xA, 0x2, 0x5, 0x1, 0x0,
      /* ebox */
      0x1, 0xB, 0x9, 0xC, 0xD, 0x6, 0xF, 0x3,
      0xE, 0x8, 0x7, 0x4, 0xA, 0x2, 0x5, 0x0,
      /* iebox */
      0xF, 0x0, 0xD, 0x7, 0xB, 0xE, 0x5, 0xA,
      0x9, 0x2, 0xC, 0x1, 0x3, 0x4, 0x8, 0x6,
};
void sub_bytes(uint64_t state[8]) {
```

*/*Значение блоков R, E, E^(-1) загружаем в q0-q2, в q3 загружаем 0xF, значение state загружаем в q4-q7:*/*

```
      asm(
     "vldm. 64 %[a], {d0-d5}"
     "vmov. i8 q3, #0xF"
     "vldm. 64 %[s], {d8-d15}"
     "vshr. u8 q12, q4, #4"          /*в q12 загружаем i >> 4*/
     "vand q13, q4, q3"             /*в q13 загружаем i & 0xF*/

     "vtbl. 8 d24, {d2, d3}, d24"  /*в q12 =  x = E[i >> 4]*/
     "vtbl. 8 d25, {d2, d3}, d25"
     "vtbl. 8 d26, {d4, d5}, d26"  /*в q13 = E^(-1) [i & 0xF]*/
     "vtbl. 8 d27, {d4, d5}, d27"
     "veor. 8 q14, q12, q13"       /*в q14 = x^y*/
     "vtbl. 8 d28, {d0, d1}, d28"  /*в q14 = r = R[x^y]*/
     "vtbl. 8 d29, {d0, d1}, d29"
      veor. 8 q12, q14"             /*в q12 = x=^r*/
     "veor. 8 q13, q14"            /*в q13 = y=^r*/
     "vtbl. 8 d24, {d2, d3}, d24" /*в q12 = E[x] << 4*/
     "vtbl. 8 d25, {d2, d3}, d25"
     "vshl. u8 q12, #4"
     "vtbl. 8 d26, {d4, d5}, d26" /* в q13 = E^(-1)[x]*/
     "vtbl. 8 d27, {d4, d5}, d27"
     "vorr. 8 q4, q12, q13"
     "vorr. 8 q4, q12, q13"        /* в q4 - E[x] << 4 | E^(-1)[x]*/

     /*повторяем процедуру для матрицы состояния в регистрах q5, q6 и q7
}
```

.2. MixRows

В процедуре MixRows происходит умножение на 2 в GF(2^8). Алгоритм умножения матрицы состояния и c(x) показан на рис. 12:

```
Рассмотрим 1 ряд новой матрицы состояния в транспонированной форме:
[t00]   [s00 * 0x1]   [s01 * 0x9]       [s07 * 0x1]
[t01]   [s00 * 0x1]   [s01 * 0x1]       [s07 * 0x4]
[t02]   [s00 * 0x4]   [s01 * 0x1]       [s07 * 0x1]
[t03] = [s00 * 0x1] ^ [s01 * 0x4] ^...^ [s07 * 0x8].
[t04]   [s00 * 0x8]   [s01 * 0x1]       [s07 * 0x5]
[t05]   [s00 * 0x5]   [s01 * 0x8]       [s07 * 0x2]
[t06]   [s00 * 0x2]   [s01 * 0x5]       [s07 * 0x9]
[t07]   [s00 * 0x9]   [s01 * 0x2]       [s07 * 0x1]

Мы можем сделать следующую замену:
[t00]   [s00]   [s05]   [s07]    [s02]    [s06]    [s03]    [s04]
[t01]   [s01]   [s06]   [s00]    [s03]    [s07]    [s04]    [s05]
[t02]   [s02]   [s07]   [s01]    [s04]    [s00]    [s05]    [s06]
[t03] = [s03] ^ [s00] ^ [s02] ^ 2[s05] ^ 4[s01] ^ 5[s06] ^ 8[s07] ^
[t04]   [s04]   [s01]   [s03]    [s06]    [s02]    [s07]    [s00]
[t05]   [s05]   [s02]   [s04]    [s07]    [s03]    [s00]    [s01]
[t06]   [s06]   [s03]   [s05]    [s00]    [s04]    [s01]    [s02]
[t07]   [s07]   [s04]   [s06]    [s01]    [s05]    [s02]    [s03]


          [s01]   [s00]   [s05]   [s07]   [s03]   [s01]
          [s02]   [s01]   [s06]   [s00]   [s04]   [s02]
          [s03]   [s02]   [s07]   [s01]   [s05]   [s03]
       ^ 9[s04] = [s03] ^ [s00] ^ [s02] ^ [s06] ^ [s04] ^
          [s05]   [s04]   [s01]   [s03]   [s07]   [s05]
          [s06]   [s05]   [s02]   [s04]   [s00]   [s06]
          [s07]   [s06]   [s03]   [s05]   [s01]   [s07]
          [s00]   [s07]   [s04]   [s06]   [s02]   [s00]

          [s02]    [s06]    [s03]    [s04]    [s01]
          [s03]    [s07]    [s04]    [s05]    [s02]
          [s04]    [s00]    [s05]    [s06]    [s03]
       ^ 2[s05] ^ 4[s01] ^ 4[s06] ^ 8[s07] ^ 8[s04].
          [s06]    [s02]    [s07]    [s00]    [s05]
          [s07]    [s03]    [s00]    [s01]    [s06]
          [s00]    [s04]    [s01]    [s02]    [s07]
          [s01]    [s05]    [s02]    [s03]    [s00]
```

Рис. 12. Коментарии к процедуре MixRows

```
asm(
  "mov r0, #8"            // в r0 записываем количество строк
  "mov r1, %[s]"

  "vmov. i8 q14, #0x80"   // Регистры необходимы для умножения
  "vmov. i8 q15, #0x1D"   // на 2, 4, 8. Мы используем q13 для vtst
```

```
"mix_rows_loop_start:"

"subs r0, #2"           // Мы обрабатываем 2 ряда сразу
"bmi mix_rows_loop_end"

"vldm. 64 r1, {q0}"       // в q0 хранятся текущие строки,
                          // и будет записан результат

"vext. 8 d2, d0, d0, #1"  // q1 = [s01,.., s00, s11,.., s10]
"vext. 8 d3, d1, d1, #1"
"veor. 8 q0, q1"

"vext. 8 d2, d2, #2"      // q1 = [s03,.., s02, s13,.., s12]
"vext. 8 d3, d3, #2"
"veor. 8 q0, q1"

"vext. 8 d2, d2, #2"      // q1 = [s05,.., s04, s15,.., s14]
"vext. 8 d3, d3, #2"
"veor. 8 q0, q1"

"vext. 8 d2, d2, #2"      // q1 = [s07,.., s06, s17,.., s16]
"vext. 8 d3, d3, #2"
"veor. 8 q0, q1"

                          // Умножение на  2
"vtst. 8 q13, q1, q14"

"vand. i8 q13, q15"       // каждый байт в q13 = 0x1D,если старший
                          //  байт = 1 или 0x00, если байт = 0


"vshl. u8 q1, #1"         // Циклический сдвиг на 1 бит влево

"veor. i8 q1, q13"        // q1 = 2 * [s07,.., s06, s17,.., s16]

"vext. 8 d2, d2, #3"      // q1 = 2 * [s02,.., s01, s12,.., s11]
```

```
"vext. 8 d3, d3, #3"
"veor. 8 q0, q1"
```

// У множение на 4

```
"vtst. 8 q13, q1, q14"
"vand. i8 q13, q15"
"vshl. u8 q1, #1"
"veor. i8 q1, q13"     // q1 = 4 * [s02,.., s01, s12,.., s11]

"vext. 8 d2, d2, #1"   // q1 = 4 * [s03,.., s02, s13,.., s12]
"vext. 8 d3, d3, #1"
"veor. 8 q0, q1"

"vext. 8 d2, d2, #3"     // q1 = 4 * [s06,.., s05, s16,.., s15]
"vext. 8 d3, d3, #3"
"veor. 8 q0, q1"
```

// У множение на 8

```
"vtst. 8 q13, q1, q14"
"vand. i8 q13, q15"
"vshl. u8 q1, #1"
"veor. i8 q1, q13"      // q1 = 8 * [s06,.., s05, s16,.., s15]

"vext. 8 d2, d2, #3"    // q1 = 8 * [s01,.., s00, s11,.., s00]
"vext. 8 d3, d3, #3"
"veor. 8 q0, q1"

"vext. 8 d2, d2, #3"   // q1 =  8 * [s04,.., s03, s14,.., s03]
"vext. 8 d3, d3, #3"
"veor. 8 q0, q1"

"vstm. 64 r1!, {q0}"

"b mix_rows_loop_start"

"mix_rows_loop_end:"
```

```
    :
    : [s] "r" (state)
    : "memory", "r0", "r1", "q0", "q1", "q13", "q14", "q15"
        );
}
```

7.3. ShiftColumns

Процедура ShiftColumns выполняет циклический сдвиг столбцов матрицы состояния вправо. На вход процедуры передается матрица состояния state. Значение массива state загружается в регистры $d_0 - d_7$.

```
void shift_columns(uint64_t state[8])
{
asm (
     "mov r0, %[a]"
     "vld4. 8 {d0-d3}, [r0]!"
     "vld4. 8 {d4-d7}, [r0]"
```

*/*для реализации циклического сдвига мы транспонируем матрицу состояния, осуществляем циклический сдвиг строк, после чего повторно транспонируем матрицу состояния*/*

```
     "vuzp. 8 d0, d4"
     "vuzp. 8 d1, d5"
     "vuzp. 8 d2, d6"
     "vuzp. 8 d3, d7"
     "vext. 8 d1, d1, #7"
     "vext. 8 d2, d2, #6"
     "vext. 8 d3, d3, #5"
     "vext. 8 d4, d4, #4"
     "vext. 8 d5, d5, #3"
     "vext. 8 d6, d6, #2"
     "vext. 8 d7, d7, #1"
     "vzip. 8 d0, d4"
     "vzip. 8 d1, d5"
     "vzip. 8 d2, d6"
     "vzip. 8 d3, d7"
     "mov r0, %[a]"
     "vst4. 8 {d0-d3}, [r0]!"
     "vst4. 8 {d4-d7}, [r0]"
     : [a] "r" (state)
     : "memory", "r0", "d0", "d1", "d2",
   "d3", "d4", "d5", "d6", "d7"
     );
```

7.4. AddRoundKey

На вход процедуре передается матрица состояния state и ключ раунда rk. Значение массива state загружается в регистры $d_0 - d_7$, значение rk — в регистры $d_8 - d_{15}$. Далее вычисляется результат операции *XOR* соответ-

ствующих байт указанных регистров, который записывается обратно в state.

```
void add_round_key(uint64_t state[8], uint64_t rk[8]) {
      asm(
    "vldm. 64 %[s], {d0-d7}"
    "vldm. 64 %[r], {d8-d15}"

    "veor. 8 q0, q4"
    "veor. 8 q1, q5"
    "veor. 8 q2, q6"
    "veor. 8 q3, q7"
    "vstm. 64 %[s], {d0-d7}"
      :
      : [s] "r" (state), [r] "r" (rk)
      : "memory", "q0", "q1", "q2", "q3",
            "q4", "q5", "q6", "q7"
        );
}
```

Тестирование реализаций

Платформой для ассемблерной реализации было выбрано семейство процессоров ARMv7. В рамках дипломной работы была разработана программная библиотека для шифрования и хеширования данных и проведен сравнительный анализ с криптографическим пакетом OpenSSL, являющимся индустриальным стандартом. Библиотека libcrypto, входящая в состав этого пакета (использовалась версия 1.02h), была скомпилирована под ARMv7 с настройками по умолчанию с помощью компилятора gcc из состава пакета Linaro 5.3.0. Используемый API в реализации OpenSSL выбирает наиболее подходящий код для платформы выполнения.

Библиотека, разработанная в рамках дипомной работы, была скомпилирована под ARMv7 с параметрами оптимизации O3 с помощью компилятора gcc из состава пакета Linaro 5.3.0.

Все реализации были протестированы на планшете ASUS MeMO Pad FHD 10 2014, с процессором Qualcomm S4 Pro APQ8064 1,5 GHz с четырьмя ядрами Krait и оперативной памятью LPDDR2-1066 объёмом 2 гигабайта.

В рамках дипломной работы сравниваются 4 реализации алгоритма шифрования AES:

1. OSSL — криптографический пакет с открытым исходным кодом для работы с SSL/TLS;
2. REF — основная реализация, созданная в рамках дипломной работы;
3. SLOW — реализация с защитой от атак на кэш, созданная в рамках дипломной работы;
4. ASMV — ассемблерная реализация, созданная в рамках дипломной работы;

Результаты сравнения приведены в таблице 2. В качестве критериев сравнения были выбраны совпадение исходного и дешифрованного текста, совпадение зашифрованного текста с зашифрованным текстом OSSL и скорость работы.

Таблица 2. Сравнение реализаций AES:

	OSSL	REF	SLOW	ASMV
Совпадение исходного и дешифрованного текста	да	да	да	да
Совпадение зашифрованного текста с OSSL	да	да	да	да
Скорость работы AES-128 (Мб / с)	6.201	6.704	0.154	7.756
Скорость работы AES-192 (Мб / с)	5.615	6.391	0.128	6.238
Скорость работы AES-256 (Мб / с)	4.802	5.281	0.110	5.446

Основная реализация AES работает на 8% быстрее, чем реализация OpenSSL. В реализации AES с защитой от атак на кэш происходит большое количество вычислений, скорость работы падает в 40 раз. Сильный разрыв между скоростью работы реализаций связан с тем, что во втором случае обратный элемент в $GF(2^8)$ находится с помощью возведения в 254 степень. Ассемблерная реализация оказывается лидером по скорости за счёт распараллеливания вычислений на уровне данных, оптимизации выполнения команд и использования таблиц подстановок с постоянным временем доступа в нелинейном преобразовании.

```
shell@android:/data/local/tmp/crypto  ./test_AES -f 5MB.bin
Using filename 5MB.bin of 5242880 bytes size
```

```
Testing AES-CBC-128
Using random key: BC6BB0C110B524E78E2F9B26B21FE322
Using random IV:  A019D44E64A73054B9A8B73A8450A1A5
OSSL: plain and decoded messages are the same, 0.844987
REF:  plain and decoded messages are the same, encoded message is the same as of OSSL's one, 0.781596
SLOW: plain and decoded messages are the same, encoded message is the same as of OSSL's one, 33.859149
ASMV: plain and decoded messages are the same, encoded message is the same as of OSSL's one, 0.675568
```

```
Testing AES-CBC-192
Using random key: 78E15800D0CFDA28B231EE38ABABB3ACDACB10D18CEB0385
Using random IV:  8A8B3069EA37F6D5FA9715981ECFF2E9
OSSL: plain and decoded messages are the same, 0.933130
REF:  plain and decoded messages are the same, encoded message is the same as of OSSL's one, 0.819899
SLOW: plain and decoded messages are the same, encoded message is the same as of OSSL's one, 40.720494
ASMV: plain and decoded messages are the same, encoded message is the same as of OSSL's one, 0.840500
```

```
Testing AES-CBC-256
Using random key: 91A3E0992DCA2076A4FE5404DC139BC08A9EC852EF1F00368989C0642A08B414
Using random IV:  1129223C9E4DE6FE522C518F3C89F3CC
OSSL: plain and decoded messages are the same, 1.091653
REF:  plain and decoded messages are the same, encoded message is the same as of OSSL's one, 0.992736
SLOW: plain and decoded messages are the same, encoded message is the same as of OSSL's one, 47.569693
ASMV: plain and decoded messages are the same, encoded message is the same as of OSSL's one, 0.962582
```

В рамках дипломной работы сравниваются 5 реализаций алгоритма хеширования WHIRLPOOL:

1. REFO — реализация, разработанная авторами алгоритма;
2. OSSL — криптографический пакет с открытым исходным кодом для работы с SSL/TLS;
3. REF — основная реализация, созданная в рамках дипломной работы;
4. SLOW — реализация с защитой от атак на кэш, созданная в рамках дипломной работы;
5. ASMV — ассемблерная реализация, созданная в рамках дипломной работы;

Результаты сравнения приведены в таблице 3. В качестве критериев сравнения были выбраны совпадение хешированного текста реализации с хешем REFO и скорость работы.

Таблица 3. Сравнение реализаций WHIRLPOOL:

	REFO	OSSL	REF	SLOW	ASMV
Совпадение хешированного текста с REFO	да	да	да	да	да
Скорость работы WHIRLPOOL (Мб / с)	7.693	5.236	4.739	0.963	9.216

Основная реализация WHIRLPOOL работает в 2 раз медленнее реализации Винсента Реймена и Паоло Баретто и оказывается незначительно медленнее реализации OpenSSL. В реализации WHIRLPOOL с защитой от атак на кэш происходит большое количество вычислений, скорость работы падает в 5 раз по сравнению с основной реализацией. Ассемблерная реализация оказывается лидером по скорости за счёт распараллеливания вычислений на уровне данных, оптимизации выполнения команд и использования таблиц подстановок с постоянным временем доступа в нелинейном преобразовании.

```
shell@android:/data/local/tmp/crypto ./test_whirlpool -f 5MB.bin
Using filename 5MB.bin of 91327 bytes size
```

```
REFO 2DEC8128F06E89BD70A16182E413BE0C34AFEBFC74A7CD45DFD9B7B551D0B4279
        BA91632B54136065A55490FCD9B61D728EB1CC4CF614F1226768C4260BDAE8B
0.681429 seconds
```

```
OSSL 2DEC8128F06E89BD70A16182E413BE0C34AFEBFC74A7CD45DFD9B7B551D0B4279
        BA91632B54136065A55490FCD9B61D728EB1CC4CF614F1226768C4260BDAE8B

1.001191 seconds
```

```
REF 2DEC8128F06E89BD70A16182E413BE0C34AFEBFC74A7CD45DFD9B7B551D0B4279
        BA91632B54136065A55490FCD9B61D728EB1CC4CF614F1226768C4260BDAE8B

1.106242 seconds
```

```
SLOW 2DEC8128F06E89BD70A16182E413BE0C34AFEBFC74A7CD45DFD9B7B551D0B4279
        BA91632B54136065A55490FCD9B61D728EB1CC4CF614F1226768C4260BDAE8B

5.440134 seconds
```

```
ASMV 2DEC8128F06E89BD70A16182E413BE0C34AFEBFC74A7CD45DFD9B7B551D0B4279
        BA91632B54136065A55490FCD9B61D728EB1CC4CF614F1226768C4260BDAE8B

0.568869 seconds
```

Заключение

В рамках дипломной работы выполнены все поставленные задачи, библиотека для шифрования и хеширования данных протестирована и успешно работает.

В процессе изучения криптографических алгоритмов были описаны математический аппарат и алгоритмическая структура блочного шифра AES и хеш-функции WHIRLPOOL. С их помощью были написаны основные реализации алгоритмов AES и WHIRLPOOL. Данные реализации используют таблицы подстановки и не являются устойчивыми к атакам на кэш.

На базе основных реализаций были написаны реализации, исключающие атаки на кэш. В реализациях, исключающих атаки на кэш, происходит большое количество вычислений, и выполнение занимает гораздо больше времени, нежели в реализациях, основанных на таблицах подстановок. Для решения возникшей проблемы были написаны ассемблерные реализации. Ассемблерные реализации работают быстрее, чем основные реализации и реализации от OpenSSL. Высокая скорость работы ассемблерных реализаций связана с распараллеливанием вычислений на уровне данных и оптимизацией выполняемых команд.

Таким образом, в работе показано, что за счёт эффективного использования возможностей современных мобильных вычислительных устройств можно написать реализации AES и WHIRLPOOL, не являющихся уязвимыми к атакам на кэш и не уступающим по скорости промышленным реализациям.

Автор выражает благодарность своему научному руководителю Ильчукову Александру Сергеевичу и руководителю Лаборатории Прикладной Математики и Программирования Ситкареву Григорию Александровичу за неоценимую помощь в написании данной работы.

Список литературы

1. Галуев Г. А. Математические основы криптологии: М.: ТРТУ, 2003. С. 38–274.

2. Зензин О. С., Иванов М. А. Стандарт криптографической защиты — AES. Конечные поля. М.: КУДИЦ-Образ, 2002. С. 41–78.

3. Журавлев Ю. И., Флеров Ю. А., Вялый М. Н. Дискретный анализ. Основы высшей алгебры. 2-е изд. М.: МЗ Пресс, 2007. С. 151–224.

4. ARMv7-M Architecture Reference Manual, ARM Limited, 2010.

5. Bangerter E., Gullasch D., Krenn S. Bringing Access-Based Cache Attacks on AES to Practice. https://eprint.iacr.org/2010/594.pdf

6. Bernstein D. J. Cache-timing attacks on AES, The University of Illinois at Chicago, Chicago, 2005, pp. 1–37.

7. Daemen J., Rijmen V. The design of Rijndael, Berlin: Springer-Verlag, 2002, pp. 10–13.

8. Damgard I. A design principle for hash functions // Advances in Cryptology. Crypto. New York, 1989. Vol. 435. pp. 416–427.

9. Diffiee W., Hellman M. New directions in cryptography // IEEE Transactions on Information Theory. New Jersey, 1976. Vol 22. No 8. pp. 644–654.

10. Mendel F., Rechberger C., Schlaffer M., Thomsen S. The Rebound Attack: Cryptanalysis of Reduced WHIRLPOOL and Grostl. https://www.iacr.org/archive/fse2009/56650270/56650270.pdf

11. Menezes A., Oorschot P., Vanstone S. Handbook of Applied Cryptography, New York: CRC Press, 1997, 794 p.

12. Osvik D., Shamir A., Tromer E. Cache Attacks and Countermeasures: the Case of AES, 20 p. https://eprint.iacr.org/2005/271.pdf

13. Rijmen V., Barreto P. The WHIRLPOOL Hashing Function, San Paulo: Scopus Tecnologia, 2000, 20 p.

14. Shirai T., Shibutani K. On the diffusion matrix employed in the WHIRLPOOL hashing function, NESSIE public report, 2003, 7 p.

Приложение 1. Таблицы подстановки

```
AES_sbox = array{
   0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
   0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
   0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
   0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
   0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
   0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
   0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
   0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
   0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
   0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
   0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
   0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
   0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
   0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
   0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
   0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
   0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
   0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
   0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
   0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
   0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
   0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
   0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
   0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
   0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
   0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
   0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
   0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
   0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
   0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
   0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
   0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};
```

```
INV_AES_sbox = array{
   0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38,
   0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
   0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87,
   0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
   0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d,
   0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
   0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2,
   0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
   0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16,
   0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
   0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda,
   0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
   0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a,
   0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
   0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02,
   0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
   0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea,
   0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
   0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85,
   0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
   0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89,
   0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
   0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20,
   0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
   0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31,
   0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
   0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d,
   0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
   0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0,
   0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
   0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26,
   0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
   };
```

```
static const uint32_t M0[] = {
   0x00000000, 0x03010102, 0x06020204, 0x05030306,
   0x0c040408, 0x0f05050a, 0x0a06060c, 0x0907070e,
   0x18080810, 0x1b090912, 0x1e0a0a14, 0x1d0b0b16,
   0x140c0c18, 0x170d0d1a, 0x120e0e1c, 0x110f0f1e,
   0x30101020, 0x33111122, 0x36121224, 0x35131326,
   0x3c141428, 0x3f15152a, 0x3a16162c, 0x3917172e,
   0x28181830, 0x2b191932, 0x2e1a1a34, 0x2d1b1b36,
   0x241c1c38, 0x271d1d3a, 0x221e1e3c, 0x211f1f3e,
   0x60202040, 0x63212142, 0x66222244, 0x65232346,
   0x6c242448, 0x6f25254a, 0x6a26264c, 0x6927274e,
   0x78282850, 0x7b292952, 0x7e2a2a54, 0x7d2b2b56,
   0x742c2c58, 0x772d2d5a, 0x722e2e5c, 0x712f2f5e,
   0x50303060, 0x53313162, 0x56323264, 0x55333366,
   0x5c343468, 0x5f35356a, 0x5a36366c, 0x5937376e,
   0x48383870, 0x4b393972, 0x4e3a3a74, 0x4d3b3b76,
   0x443c3c78, 0x473d3d7a, 0x423e3e7c, 0x413f3f7e,
   0xc0404080, 0xc3414182, 0xc6424284, 0xc5434386,
   0xcc444488, 0xcf45458a, 0xca46468c, 0xc947478e,
   0xd8484890, 0xdb494992, 0xde4a4a94, 0xdd4b4b96,
   0xd44c4c98, 0xd74d4d9a, 0xd24e4e9c, 0xd14f4f9e,
   0xf05050a0, 0xf35151a2, 0xf65252a4, 0xf55353a6,
   0xfc5454a8, 0xff5555aa, 0xfa5656ac, 0xf95757ae,
   0xe85858b0, 0xeb5959b2, 0xee5a5ab4, 0xed5b5bb6,
   0xe45c5cb8, 0xe75d5dba, 0xe25e5ebc, 0xe15f5fbe,
   0xa06060c0, 0xa36161c2, 0xa66262c4, 0xa56363c6,
   0xac6464c8, 0xaf6565ca, 0xaa6666cc, 0xa96767ce,
   0xb86868d0, 0xbb6969d2, 0xbe6a6ad4, 0xbd6b6bd6,
   0xb46c6cd8, 0xb76d6dda, 0xb26e6edc, 0xb16f6fde,
   0x907070e0, 0x937171e2, 0x967272e4, 0x957373e6,
   0x9c7474e8, 0x9f7575ea, 0x9a7676ec, 0x997777ee,
   0x887878f0, 0x8b7979f2, 0x8e7a7af4, 0x8d7b7bf6,
   0x847c7cf8, 0x877d7dfa, 0x827e7efc, 0x817f7ffe,
   0x9b80801b, 0x98818119, 0x9d82821f, 0x9e83831d,
   0x97848413, 0x94858511, 0x91868617, 0x92878715,
   0x8388880b, 0x80898909, 0x858a8a0f, 0x868b8b0d,
   0x8f8c8c03, 0x8c8d8d01, 0x898e8e07, 0x8a8f8f05,
   0xab90903b, 0xa8919139, 0xad92923f, 0xae93933d,
   0xa7949433, 0xa4959531, 0xa1969637, 0xa2979735,
   0xb398982b, 0xb0999929, 0xb59a9a2f, 0xb69b9b2d,
   0xbf9c9c23, 0xbc9d9d21, 0xb99e9e27, 0xba9f9f25,
   0xfba0a05b, 0xf8a1a159, 0xfda2a25f, 0xfea3a35d,
   0xf7a4a453, 0xf4a5a551, 0xf1a6a657, 0xf2a7a755,
   0xe3a8a84b, 0xe0a9a949, 0xe5aaaa4f, 0xe6abab4d,
   0xefacac43, 0xecadad41, 0xe9aeae47, 0xeaafaf45,
   0xcbb0b07b, 0xc8b1b179, 0xcdb2b27f, 0xceb3b37d,
   0xc7b4b473, 0xc4b5b571, 0xc1b6b677, 0xc2b7b775,
   0xd3b8b86b, 0xd0b9b969, 0xd5baba6f, 0xd6bbbb6d,
   0xdfbcbc63, 0xdcbdbd61, 0xd9bebe67, 0xdabfbf65,
   0x5bc0c09b, 0x58c1c199, 0x5dc2c29f, 0x5ec3c39d,
   0x57c4c493, 0x54c5c591, 0x51c6c697, 0x52c7c795,
   0x43c8c88b, 0x40c9c989, 0x45caca8f, 0x46cbcb8d,
   0x4fcccc83, 0x4ccdcd81, 0x49cece87, 0x4acfcf85,
   0x6bd0d0bb, 0x68d1d1b9, 0x6dd2d2bf, 0x6ed3d3bd,
   0x67d4d4b3, 0x64d5d5b1, 0x61d6d6b7, 0x62d7d7b5,
   0x73d8d8ab, 0x70d9d9a9, 0x75dadaaf, 0x76dbdbad,
   0x7fdcdca3, 0x7cdddda1, 0x79dedea7, 0x7adfdfa5,
```

```
    0x3be0e0db, 0x38e1e1d9, 0x3de2e2df, 0x3ee3e3dd,
    0x37e4e4d3, 0x34e5e5d1, 0x31e6e6d7, 0x32e7e7d5,
    0x23e8e8cb, 0x20e9e9c9, 0x25eaeacf, 0x26ebebcd,
    0x2fececc3, 0x2cededc1, 0x29eeeec7, 0x2aefefc5,
    0x0bf0f0fb, 0x08f1f1f9, 0x0df2f2ff, 0x0ef3f3fd,
    0x07f4f4f3, 0x04f5f5f1, 0x01f6f6f7, 0x02f7f7f5,
    0x13f8f8eb, 0x10f9f9e9, 0x15fafaef, 0x16fbfbed,
    0x1ffcfce3, 0x1cfdfde1, 0x19fefee7, 0x1affffe5
};
```

```
static const uint32_t M1[] = {
   0x00000000, 0x01010203, 0x02020406, 0x03030605,
   0x0404080c, 0x05050a0f, 0x06060c0a, 0x07070e09,
   0x08081018, 0x0909121b, 0x0a0a141e, 0x0b0b161d,
   0x0c0c1814, 0x0d0d1a17, 0x0e0e1c12, 0x0f0f1e11,
   0x10102030, 0x11112233, 0x12122436, 0x13132635,
   0x1414283c, 0x15152a3f, 0x16162c3a, 0x17172e39,
   0x18183028, 0x1919322b, 0x1a1a342e, 0x1b1b362d,
   0x1c1c3824, 0x1d1d3a27, 0x1e1e3c22, 0x1f1f3e21,
   0x20204060, 0x21214263, 0x22224466, 0x23234665,
   0x2424486c, 0x25254a6f, 0x26264c6a, 0x27274e69,
   0x28285078, 0x2929527b, 0x2a2a547e, 0x2b2b567d,
   0x2c2c5874, 0x2d2d5a77, 0x2e2e5c72, 0x2f2f5e71,
   0x30306050, 0x31316253, 0x32326456, 0x33336655,
   0x3434685c, 0x35356a5f, 0x36366c5a, 0x37376e59,
   0x38387048, 0x3939724b, 0x3a3a744e, 0x3b3b764d,
   0x3c3c7844, 0x3d3d7a47, 0x3e3e7c42, 0x3f3f7e41,
   0x404080c0, 0x414182c3, 0x424284c6, 0x434386c5,
   0x444488cc, 0x45458acf, 0x46468cca, 0x47478ec9,
   0x484890d8, 0x494992db, 0x4a4a94de, 0x4b4b96dd,
   0x4c4c98d4, 0x4d4d9ad7, 0x4e4e9cd2, 0x4f4f9ed1,
   0x5050a0f0, 0x5151a2f3, 0x5252a4f6, 0x5353a6f5,
   0x5454a8fc, 0x5555aaff, 0x5656acfa, 0x5757aef9,
   0x5858b0e8, 0x5959b2eb, 0x5a5ab4ee, 0x5b5bb6ed,
   0x5c5cb8e4, 0x5d5dbae7, 0x5e5ebce2, 0x5f5fbee1,
   0x6060c0a0, 0x6161c2a3, 0x6262c4a6, 0x6363c6a5,
   0x6464c8ac, 0x6565caaf, 0x6666ccaa, 0x6767cea9,
   0x6868d0b8, 0x6969d2bb, 0x6a6ad4be, 0x6b6bd6bd,
   0x6c6cd8b4, 0x6d6ddab7, 0x6e6edcb2, 0x6f6fdeb1,
   0x7070e090, 0x7171e293, 0x7272e496, 0x7373e695,
   0x7474e89c, 0x7575ea9f, 0x7676ec9a, 0x7777ee99,
   0x7878f088, 0x7979f28b, 0x7a7af48e, 0x7b7bf68d,
   0x7c7cf884, 0x7d7dfa87, 0x7e7efc82, 0x7f7ffe81,
   0x80801b9b, 0x81811998, 0x82821f9d, 0x83831d9e,
   0x84841397, 0x85851194, 0x86861791, 0x87871592,
   0x88880b83, 0x89890980, 0x8a8a0f85, 0x8b8b0d86,
   0x8c8c038f, 0x8d8d018c, 0x8e8e0789, 0x8f8f058a,
   0x90903bab, 0x919139a8, 0x92923fad, 0x93933dae,
   0x949433a7, 0x959531a4, 0x969637a1, 0x979735a2,
   0x98982bb3, 0x999929b0, 0x9a9a2fb5, 0x9b9b2db6,
   0x9c9c23bf, 0x9d9d21bc, 0x9e9e27b9, 0x9f9f25ba,
   0xa0a05bfb, 0xa1a159f8, 0xa2a25ffd, 0xa3a35dfe,
   0xa4a453f7, 0xa5a551f4, 0xa6a657f1, 0xa7a755f2,
   0xa8a84be3, 0xa9a949e0, 0xaaaa4fe5, 0xabab4de6,
   0xacac43ef, 0xadad41ec, 0xaeae47e9, 0xafaf45ea,
   0xb0b07bcb, 0xb1b179c8, 0xb2b27fcd, 0xb3b37dce,
   0xb4b473c7, 0xb5b571c4, 0xb6b677c1, 0xb7b775c2,
   0xb8b86bd3, 0xb9b969d0, 0xbaba6fd5, 0xbbbb6dd6,
   0xbcbc63df, 0xbdbd61dc, 0xbebe67d9, 0xbfbf65da,
   0xc0c09b5b, 0xc1c19958, 0xc2c29f5d, 0xc3c39d5e,
   0xc4c49357, 0xc5c59154, 0xc6c69751, 0xc7c79552,
   0xc8c88b43, 0xc9c98940, 0xcaca8f45, 0xcbcb8d46,
   0xcccc834f, 0xcdcd814c, 0xcece8749, 0xcfcf854a,
   0xd0d0bb6b, 0xd1d1b968, 0xd2d2bf6d, 0xd3d3bd6e,
   0xd4d4b367, 0xd5d5b164, 0xd6d6b761, 0xd7d7b562,
   0xd8d8ab73, 0xd9d9a970, 0xdadaaf75, 0xdbdbad76,
   0xdcdca37f, 0xddddа17c, 0xdedea779, 0xdfdfa57a,
```

```
    0xe0e0db3b, 0xe1e1d938, 0xe2e2df3d, 0xe3e3dd3e,
    0xe4e4d337, 0xe5e5d134, 0xe6e6d731, 0xe7e7d532,
    0xe8e8cb23, 0xe9e9c920, 0xeaeacf25, 0xebebcd26,
    0xececc32f, 0xededc12c, 0xeeeec729, 0xefefc52a,
    0xf0f0fb0b, 0xf1f1f908, 0xf2f2ff0d, 0xf3f3fd0e,
    0xf4f4f307, 0xf5f5f104, 0xf6f6f701, 0xf7f7f502,
    0xf8f8eb13, 0xf9f9e910, 0xfafaef15, 0xfbfbed16,
    0xfcfce31f, 0xfdfde11c, 0xfefee719, 0xffffe51a
};
```

```
static const uint32_t M2[] = {
   0x00000000, 0x01020301, 0x02040602, 0x03060503,
   0x04080c04, 0x050a0f05, 0x060c0a06, 0x070e0907,
   0x08101808, 0x09121b09, 0x0a141e0a, 0x0b161d0b,
   0x0c18140c, 0x0d1a170d, 0x0e1c120e, 0x0f1e110f,
   0x10203010, 0x11223311, 0x12243612, 0x13263513,
   0x14283c14, 0x152a3f15, 0x162c3a16, 0x172e3917,
   0x18302818, 0x19322b19, 0x1a342e1a, 0x1b362d1b,
   0x1c38241c, 0x1d3a271d, 0x1e3c221e, 0x1f3e211f,
   0x20406020, 0x21426321, 0x22446622, 0x23466523,
   0x24486c24, 0x254a6f25, 0x264c6a26, 0x274e6927,
   0x28507828, 0x29527b29, 0x2a547e2a, 0x2b567d2b,
   0x2c58742c, 0x2d5a772d, 0x2e5c722e, 0x2f5e712f,
   0x30605030, 0x31625331, 0x32645632, 0x33665533,
   0x34685c34, 0x356a5f35, 0x366c5a36, 0x376e5937,
   0x38704838, 0x39724b39, 0x3a744e3a, 0x3b764d3b,
   0x3c78443c, 0x3d7a473d, 0x3e7c423e, 0x3f7e413f,
   0x4080c040, 0x4182c341, 0x4284c642, 0x4386c543,
   0x4488cc44, 0x458acf45, 0x468cca46, 0x478ec947,
   0x4890d848, 0x4992db49, 0x4a94de4a, 0x4b96dd4b,
   0x4c98d44c, 0x4d9ad74d, 0x4e9cd24e, 0x4f9ed14f,
   0x50a0f050, 0x51a2f351, 0x52a4f652, 0x53a6f553,
   0x54a8fc54, 0x55aaff55, 0x56acfa56, 0x57aef957,
   0x58b0e858, 0x59b2eb59, 0x5ab4ee5a, 0x5bb6ed5b,
   0x5cb8e45c, 0x5dbae75d, 0x5ebce25e, 0x5fbee15f,
   0x60c0a060, 0x61c2a361, 0x62c4a662, 0x63c6a563,
   0x64c8ac64, 0x65caaf65, 0x66ccaa66, 0x67cea967,
   0x68d0b868, 0x69d2bb69, 0x6ad4be6a, 0x6bd6bd6b,
   0x6cd8b46c, 0x6ddab76d, 0x6edcb26e, 0x6fdeb16f,
   0x70e09070, 0x71e29371, 0x72e49672, 0x73e69573,
   0x74e89c74, 0x75ea9f75, 0x76ec9a76, 0x77ee9977,
   0x78f08878, 0x79f28b79, 0x7af48e7a, 0x7bf68d7b,
   0x7cf8847c, 0x7dfa877d, 0x7efc827e, 0x7ffe817f,
   0x801b9b80, 0x81199881, 0x821f9d82, 0x831d9e83,
   0x84139784, 0x85119485, 0x86179186, 0x87159287,
   0x880b8388, 0x89098089, 0x8a0f858a, 0x8b0d868b,
   0x8c038f8c, 0x8d018c8d, 0x8e07898e, 0x8f058a8f,
   0x903bab90, 0x9139a891, 0x923fad92, 0x933dae93,
   0x9433a794, 0x9531a495, 0x9637a196, 0x9735a297,
   0x982bb398, 0x9929b099, 0x9a2fb59a, 0x9b2db69b,
   0x9c23bf9c, 0x9d21bc9d, 0x9e27b99e, 0x9f25ba9f,
   0xa05bfba0, 0xa159f8a1, 0xa25ffda2, 0xa35dfea3,
   0xa453f7a4, 0xa551f4a5, 0xa657f1a6, 0xa755f2a7,
   0xa84be3a8, 0xa949e0a9, 0xaa4fe5aa, 0xab4de6ab,
   0xac43efac, 0xad41ecad, 0xae47e9ae, 0xaf45eaaf,
   0xb07bcbb0, 0xb179c8b1, 0xb27fcdb2, 0xb37dceb3,
   0xb473c7b4, 0xb571c4b5, 0xb677c1b6, 0xb775c2b7,
   0xb86bd3b8, 0xb969d0b9, 0xba6fd5ba, 0xbb6dd6bb,
   0xbc63dfbc, 0xbd61dcbd, 0xbe67d9be, 0xbf65dabf,
   0xc09b5bc0, 0xc19958c1, 0xc29f5dc2, 0xc39d5ec3,
   0xc49357c4, 0xc59154c5, 0xc69751c6, 0xc79552c7,
   0xc88b43c8, 0xc98940c9, 0xca8f45ca, 0xcb8d46cb,
   0xcc834fcc, 0xcd814ccd, 0xce8749ce, 0xcf854acf,
   0xd0bb6bd0, 0xd1b968d1, 0xd2bf6dd2, 0xd3bd6ed3,
   0xd4b367d4, 0xd5b164d5, 0xd6b761d6, 0xd7b562d7,
   0xd8ab73d8, 0xd9a970d9, 0xdaaf75da, 0xdbad76db,
   0xdca37fdc, 0xdda17cdd, 0xdea779de, 0xdfa57adf,
```

```
    0xe0db3be0,  0xe1d938e1,  0xe2df3de2,  0xe3dd3ee3,
    0xe4d337e4,  0xe5d134e5,  0xe6d731e6,  0xe7d532e7,
    0xe8cb23e8,  0xe9c920e9,  0xeacf25ea,  0xebcd26eb,
    0xecc32fec,  0xedc12ced,  0xeec729ee,  0xefc52aef,
    0xf0fb0bf0,  0xf1f908f1,  0xf2ff0df2,  0xf3fd0ef3,
    0xf4f307f4,  0xf5f104f5,  0xf6f701f6,  0xf7f502f7,
    0xf8eb13f8,  0xf9e910f9,  0xfaef15fa,  0xfbed16fb,
    0xfce31ffc,  0xfde11cfd,  0xfee719fe,  0xffe51aff
};
```

```
const uint32_t M3[] = {
   0x00000000, 0x02030101, 0x04060202, 0x06050303,
   0x080c0404, 0x0a0f0505, 0x0c0a0606, 0x0e090707,
   0x10180808, 0x121b0909, 0x141e0a0a, 0x161d0b0b,
   0x18140c0c, 0x1a170d0d, 0x1c120e0e, 0x1e110f0f,
   0x20301010, 0x22331111, 0x24361212, 0x26351313,
   0x283c1414, 0x2a3f1515, 0x2c3a1616, 0x2e391717,
   0x30281818, 0x322b1919, 0x342e1a1a, 0x362d1b1b,
   0x38241c1c, 0x3a271d1d, 0x3c221e1e, 0x3e211f1f,
   0x40602020, 0x42632121, 0x44662222, 0x46652323,
   0x486c2424, 0x4a6f2525, 0x4c6a2626, 0x4e692727,
   0x50782828, 0x527b2929, 0x547e2a2a, 0x567d2b2b,
   0x58742c2c, 0x5a772d2d, 0x5c722e2e, 0x5e712f2f,
   0x60503030, 0x62533131, 0x64563232, 0x66553333,
   0x685c3434, 0x6a5f3535, 0x6c5a3636, 0x6e593737,
   0x70483838, 0x724b3939, 0x744e3a3a, 0x764d3b3b,
   0x78443c3c, 0x7a473d3d, 0x7c423e3e, 0x7e413f3f,
   0x80c04040, 0x82c34141, 0x84c64242, 0x86c54343,
   0x88cc4444, 0x8acf4545, 0x8cca4646, 0x8ec94747,
   0x90d84848, 0x92db4949, 0x94de4a4a, 0x96dd4b4b,
   0x98d44c4c, 0x9ad74d4d, 0x9cd24e4e, 0x9ed14f4f,
   0xa0f05050, 0xa2f35151, 0xa4f65252, 0xa6f55353,
   0xa8fc5454, 0xaaff5555, 0xacfa5656, 0xaef95757,
   0xb0e85858, 0xb2eb5959, 0xb4ee5a5a, 0xb6ed5b5b,
   0xb8e45c5c, 0xbae75d5d, 0xbce25e5e, 0xbee15f5f,
   0xc0a06060, 0xc2a36161, 0xc4a66262, 0xc6a56363,
   0xc8ac6464, 0xcaaf6565, 0xccaa6666, 0xcea96767,
   0xd0b86868, 0xd2bb6969, 0xd4be6a6a, 0xd6bd6b6b,
   0xd8b46c6c, 0xdab76d6d, 0xdcb26e6e, 0xdeb16f6f,
   0xe0907070, 0xe2937171, 0xe4967272, 0xe6957373,
   0xe89c7474, 0xea9f7575, 0xec9a7676, 0xee997777,
   0xf0887878, 0xf28b7979, 0xf48e7a7a, 0xf68d7b7b,
   0xf8847c7c, 0xfa877d7d, 0xfc827e7e, 0xfe817f7f,
   0x1b9b8080, 0x19988181, 0x1f9d8282, 0x1d9e8383,
   0x13978484, 0x11948585, 0x17918686, 0x15928787,
   0x0b838888, 0x09808989, 0x0f858a8a, 0x0d868b8b,
   0x038f8c8c, 0x018c8d8d, 0x07898e8e, 0x058a8f8f,
   0x3bab9090, 0x39a89191, 0x3fad9292, 0x3dae9393,
   0x33a79494, 0x31a49595, 0x37a19696, 0x35a29797,
   0x2bb39898, 0x29b09999, 0x2fb59a9a, 0x2db69b9b,
   0x23bf9c9c, 0x21bc9d9d, 0x27b99e9e, 0x25ba9f9f,
   0x5bfba0a0, 0x59f8a1a1, 0x5ffda2a2, 0x5dfea3a3,
   0x53f7a4a4, 0x51f4a5a5, 0x57f1a6a6, 0x55f2a7a7,
   0x4be3a8a8, 0x49e0a9a9, 0x4fe5aaaa, 0x4de6abab,
   0x43efacac, 0x41ecadad, 0x47e9aeae, 0x45eaafaf,
   0x7bcbb0b0, 0x79c8b1b1, 0x7fcdb2b2, 0x7dceb3b3,
   0x73c7b4b4, 0x71c4b5b5, 0x77c1b6b6, 0x75c2b7b7,
   0x6bd3b8b8, 0x69d0b9b9, 0x6fd5baba, 0x6dd6bbbb,
   0x63dfbcbc, 0x61dcbdbd, 0x67d9bebe, 0x65dabfbf,
   0x9b5bc0c0, 0x9958c1c1, 0x9f5dc2c2, 0x9d5ec3c3,
   0x9357c4c4, 0x9154c5c5, 0x9751c6c6, 0x9552c7c7,
   0x8b43c8c8, 0x8940c9c9, 0x8f45caca, 0x8d46cbcb,
   0x834fcccc, 0x814ccdcd, 0x8749cece, 0x854acfcf,
   0xbb6bd0d0, 0xb968d1d1, 0xbf6dd2d2, 0xbd6ed3d3,
   0xb367d4d4, 0xb164d5d5, 0xb761d6d6, 0xb562d7d7,
   0xab73d8d8, 0xa970d9d9, 0xaf75dada, 0xad76dbdb,
   0xa37fdcdc, 0xa17cdddd, 0xa779dede, 0xa57adfdf,
```

```
    0xdb3be0e0, 0xd938e1e1, 0xdf3de2e2, 0xdd3ee3e3,
    0xd337e4e4, 0xd134e5e5, 0xd731e6e6, 0xd532e7e7,
    0xcb23e8e8, 0xc920e9e9, 0xcf25eaea, 0xcd26ebeb,
    0xc32fecec, 0xc12ceded, 0xc729eeee, 0xc52aefef,
    0xfb0bf0f0, 0xf908f1f1, 0xff0df2f2, 0xfd0ef3f3,
    0xf307f4f4, 0xf104f5f5, 0xf701f6f6, 0xf502f7f7,
    0xeb13f8f8, 0xe910f9f9, 0xef15fafa, 0xed16fbfb,
    0xe31ffcfc, 0xe11cfdfd, 0xe719fefe, 0xe51affff
};
```

```
static const uint32_t I0[] = {
   0x00000000,  0x0b0d090e,  0x161a121c,  0x1d171b12,
   0x2c342438,  0x27392d36,  0x3a2e3624,  0x31233f2a,
   0x58684870,  0x5365417e,  0x4e725a6c,  0x457f5362,
   0x745c6c48,  0x7f516546,  0x62467e54,  0x694b775a,
   0xb0d090e0,  0xbbdd99ee,  0xa6ca82fc,  0xadc78bf2,
   0x9ce4b4d8,  0x97e9bdd6,  0x8afea6c4,  0x81f3afca,
   0xe8b8d890,  0xe3b5d19e,  0xfea2ca8c,  0xf5afc382,
   0xc48cfca8,  0xcf81f5a6,  0xd296eeb4,  0xd99be7ba,
   0x7bbb3bdb,  0x70b632d5,  0x6da129c7,  0x66ac20c9,
   0x578f1fe3,  0x5c8216ed,  0x41950dff,  0x4a9804f1,
   0x23d373ab,  0x28de7aa5,  0x35c961b7,  0x3ec468b9,
   0x0fe75793,  0x04ea5e9d,  0x19fd458f,  0x12f04c81,
   0xcb6bab3b,  0xc066a235,  0xdd71b927,  0xd67cb029,
   0xe75f8f03,  0xec52860d,  0xf1459d1f,  0xfa489411,
   0x9303e34b,  0x980eea45,  0x8519f157,  0x8e14f859,
   0xbf37c773,  0xb43ace7d,  0xa92dd56f,  0xa220dc61,
   0xf66d76ad,  0xfd607fa3,  0xe07764b1,  0xeb7a6dbf,
   0xda595295,  0xd1545b9b,  0xcc434089,  0xc74e4987,
   0xae053edd,  0xa50837d3,  0xb81f2cc1,  0xb31225cf,
   0x82311ae5,  0x893c13eb,  0x942b08f9,  0x9f2601f7,
   0x46bde64d,  0x4db0ef43,  0x50a7f451,  0x5baafd5f,
   0x6a89c275,  0x6184cb7b,  0x7c93d069,  0x779ed967,
   0x1ed5ae3d,  0x15d8a733,  0x08cfbc21,  0x03c2b52f,
   0x32e18a05,  0x39ec830b,  0x24fb9819,  0x2ff69117,
   0x8dd64d76,  0x86db4478,  0x9bcc5f6a,  0x90c15664,
   0xa1e2694e,  0xaaef6040,  0xb7f87b52,  0xbcf5725c,
   0xd5be0506,  0xdeb30c08,  0xc3a4171a,  0xc8a91e14,
   0xf98a213e,  0xf2872830,  0xef903322,  0xe49d3a2c,
   0x3d06dd96,  0x360bd498,  0x2b1ccf8a,  0x2011c684,
   0x1132f9ae,  0x1a3ff0a0,  0x0728ebb2,  0x0c25e2bc,
   0x656e95e6,  0x6e639ce8,  0x737487fa,  0x78798ef4,
   0x495ab1de,  0x4257b8d0,  0x5f40a3c2,  0x544daacc,
   0xf7daec41,  0xfcd7e54f,  0xe1c0fe5d,  0xeacdf753,
   0xdbeec879,  0xd0e3c177,  0xcdf4da65,  0xc6f9d36b,
   0xafb2a431,  0xa4bfad3f,  0xb9a8b62d,  0xb2a5bf23,
   0x83868009,  0x888b8907,  0x959c9215,  0x9e919b1b,
   0x470a7ca1,  0x4c0775af,  0x51106ebd,  0x5a1d67b3,
   0x6b3e5899,  0x60335197,  0x7d244a85,  0x7629438b,
   0x1f6234d1,  0x146f3ddf,  0x097826cd,  0x02752fc3,
   0x335610e9,  0x385b19e7,  0x254c02f5,  0x2e410bfb,
   0x8c61d79a,  0x876cde94,  0x9a7bc586,  0x9176cc88,
   0xa055f3a2,  0xab58faac,  0xb64fe1be,  0xbd42e8b0,
   0xd4099fea,  0xdf0496e4,  0xc2138df6,  0xc91e84f8,
   0xf83dbbd2,  0xf330b2dc,  0xee27a9ce,  0xe52aa0c0,
   0x3cb1477a,  0x37bc4e74,  0x2aab5566,  0x21a65c68,
   0x10856342,  0x1b886a4c,  0x069f715e,  0x0d927850,
   0x64d90f0a,  0x6fd40604,  0x72c31d16,  0x79ce1418,
   0x48ed2b32,  0x43e0223c,  0x5ef7392e,  0x55fa3020,
   0x01b79aec,  0x0aba93e2,  0x17ad88f0,  0x1ca081fe,
   0x2d83bed4,  0x268eb7da,  0x3b99acc8,  0x3094a5c6,
   0x59dfd29c,  0x52d2db92,  0x4fc5c080,  0x44c8c98e,
   0x75ebf6a4,  0x7ee6ffaa,  0x63f1e4b8,  0x68fcedb6,
   0xb1670a0c,  0xba6a0302,  0xa77d1810,  0xac70111e,
   0x9d532e34,  0x965e273a,  0x8b493c28,  0x80443526,
   0xe90f427c,  0xe2024b72,  0xff155060,  0xf418596e,
   0xc53b6644,  0xce366f4a,  0xd3217458,  0xd82c7d56,
```

```
    0x7a0ca137, 0x7101a839, 0x6c16b32b, 0x671bba25,
    0x5638850f, 0x5d358c01, 0x40229713, 0x4b2f9e1d,
    0x2264e947, 0x2969e049, 0x347efb5b, 0x3f73f255,
    0x0e50cd7f, 0x055dc471, 0x184adf63, 0x1347d66d,
    0xcadc31d7, 0xc1d138d9, 0xdcc623cb, 0xd7cb2ac5,
    0xe6e815ef, 0xede51ce1, 0xf0f207f3, 0xfbff0efd,
    0x92b479a7, 0x99b970a9, 0x84ae6bbb, 0x8fa362b5,
    0xbe805d9f, 0xb58d5491, 0xa89a4f83, 0xa397468d
};
```

```
static const uint32_t I1[] = {
   0x00000000, 0x0d090e0b, 0x1a121c16, 0x171b121d,
   0x3424382c, 0x392d3627, 0x2e36243a, 0x233f2a31,
   0x68487058, 0x65417e53, 0x725a6c4e, 0x7f536245,
   0x5c6c4874, 0x5165467f, 0x467e5462, 0x4b775a69,
   0xd090e0b0, 0xdd99eebb, 0xca82fca6, 0xc78bf2ad,
   0xe4b4d89c, 0xe9bdd697, 0xfea6c48a, 0xf3afca81,
   0xb8d890e8, 0xb5d19ee3, 0xa2ca8cfe, 0xafc382f5,
   0x8cfca8c4, 0x81f5a6cf, 0x96eeb4d2, 0x9be7bad9,
   0xbb3bdb7b, 0xb632d570, 0xa129c76d, 0xac20c966,
   0x8f1fe357, 0x8216ed5c, 0x950dff41, 0x9804f14a,
   0xd373ab23, 0xde7aa528, 0xc961b735, 0xc468b93e,
   0xe757930f, 0xea5e9d04, 0xfd458f19, 0xf04c8112,
   0x6bab3bcb, 0x66a235c0, 0x71b927dd, 0x7cb029d6,
   0x5f8f03e7, 0x52860dec, 0x459d1ff1, 0x489411fa,
   0x03e34b93, 0x0eea4598, 0x19f15785, 0x14f8598e,
   0x37c773bf, 0x3ace7db4, 0x2dd56fa9, 0x20dc61a2,
   0x6d76adf6, 0x607fa3fd, 0x7764b1e0, 0x7a6dbfeb,
   0x595295da, 0x545b9bd1, 0x434089cc, 0x4e4987c7,
   0x053eddae, 0x0837d3a5, 0x1f2cc1b8, 0x1225cfb3,
   0x311ae582, 0x3c13eb89, 0x2b08f994, 0x2601f79f,
   0xbde64d46, 0xb0ef434d, 0xa7f45150, 0xaafd5f5b,
   0x89c2756a, 0x84cb7b61, 0x93d0697c, 0x9ed96777,
   0xd5ae3d1e, 0xd8a73315, 0xcfbc2108, 0xc2b52f03,
   0xe18a0532, 0xec830b39, 0xfb981924, 0xf691172f,
   0xd64d768d, 0xdb447886, 0xcc5f6a9b, 0xc1566490,
   0xe2694ea1, 0xef6040aa, 0xf87b52b7, 0xf5725cbc,
   0xbe0506d5, 0xb30c08de, 0xa4171ac3, 0xa91e14c8,
   0x8a213ef9, 0x872830f2, 0x903322ef, 0x9d3a2ce4,
   0x06dd963d, 0x0bd49836, 0x1ccf8a2b, 0x11c68420,
   0x32f9ae11, 0x3ff0a01a, 0x28ebb207, 0x25e2bc0c,
   0x6e95e665, 0x639ce86e, 0x7487fa73, 0x798ef478,
   0x5ab1de49, 0x57b8d042, 0x40a3c25f, 0x4daacc54,
   0xdaec41f7, 0xd7e54ffc, 0xc0fe5de1, 0xcdf753ea,
   0xeec879db, 0xe3c177d0, 0xf4da65cd, 0xf9d36bc6,
   0xb2a431af, 0xbfad3fa4, 0xa8b62db9, 0xa5bf23b2,
   0x86800983, 0x8b890788, 0x9c921595, 0x919b1b9e,
   0x0a7ca147, 0x0775af4c, 0x106ebd51, 0x1d67b35a,
   0x3e58996b, 0x33519760, 0x244a857d, 0x29438b76,
   0x6234d11f, 0x6f3ddf14, 0x7826cd09, 0x752fc302,
   0x5610e933, 0x5b19e738, 0x4c02f525, 0x410bfb2e,
   0x61d79a8c, 0x6cde9487, 0x7bc5869a, 0x76cc8891,
   0x55f3a2a0, 0x58faacab, 0x4fe1beb6, 0x42e8b0bd,
   0x099fead4, 0x0496e4df, 0x138df6c2, 0x1e84f8c9,
   0x3dbbd2f8, 0x30b2dcf3, 0x27a9ceee, 0x2aa0c0e5,
   0xb1477a3c, 0xbc4e7437, 0xab55662a, 0xa65c6821,
   0x85634210, 0x886a4c1b, 0x9f715e06, 0x9278500d,
   0xd90f0a64, 0xd406046f, 0xc31d1672, 0xce141879,
   0xed2b3248, 0xe0223c43, 0xf7392e5e, 0xfa302055,
   0xb79aec01, 0xba93e20a, 0xad88f017, 0xa081fe1c,
   0x83bed42d, 0x8eb7da26, 0x99acc83b, 0x94a5c630,
   0xdfd29c59, 0xd2db9252, 0xc5c0804f, 0xc8c98e44,
   0xebf6a475, 0xe6ffaa7e, 0xf1e4b863, 0xfcedb668,
   0x670a0cb1, 0x6a0302ba, 0x7d1810a7, 0x70111eac,
   0x532e349d, 0x5e273a96, 0x493c288b, 0x44352680,
   0x0f427ce9, 0x024b72e2, 0x155060ff, 0x18596ef4,
```

```
    0x3b6644c5, 0x366f4ace, 0x217458d3, 0x2c7d56d8,
    0x0ca1377a, 0x01a83971, 0x16b32b6c, 0x1bba2567,
    0x38850f56, 0x358c015d, 0x22971340, 0x2f9e1d4b,
    0x64e94722, 0x69e04929, 0x7efb5b34, 0x73f2553f,
    0x50cd7f0e, 0x5dc47105, 0x4adf6318, 0x47d66d13,
    0xdc31d7ca, 0xd138d9c1, 0xc623cbdc, 0xcb2ac5d7,
    0xe815efe6, 0xe51ce1ed, 0xf207f3f0, 0xff0efdfb,
    0xb479a792, 0xb970a999, 0xae6bbb84, 0xa362b58f,
    0x805d9fbe, 0x8d5491b5, 0x9a4f83a8, 0x97468da3
};
```

```
static const uint32_t I2[] = {
   0x00000000, 0x090e0b0d, 0x121c161a, 0x1b121d17,
   0x24382c34, 0x2d362739, 0x36243a2e, 0x3f2a3123,
   0x48705868, 0x417e5365, 0x5a6c4e72, 0x5362457f,
   0x6c48745c, 0x65467f51, 0x7e546246, 0x775a694b,
   0x90e0b0d0, 0x99eebbdd, 0x82fca6ca, 0x8bf2adc7,
   0xb4d89ce4, 0xbdd697e9, 0xa6c48afe, 0xafca81f3,
   0xd890e8b8, 0xd19ee3b5, 0xca8cfea2, 0xc382f5af,
   0xfca8c48c, 0xf5a6cf81, 0xeeb4d296, 0xe7bad99b,
   0x3bdb7bbb, 0x32d570b6, 0x29c76da1, 0x20c966ac,
   0x1fe3578f, 0x16ed5c82, 0x0dff4195, 0x04f14a98,
   0x73ab23d3, 0x7aa528de, 0x61b735c9, 0x68b93ec4,
   0x57930fe7, 0x5e9d04ea, 0x458f19fd, 0x4c8112f0,
   0xab3bcb6b, 0xa235c066, 0xb927dd71, 0xb029d67c,
   0x8f03e75f, 0x860dec52, 0x9d1ff145, 0x9411fa48,
   0xe34b9303, 0xea45980e, 0xf1578519, 0xf8598e14,
   0xc773bf37, 0xce7db43a, 0xd56fa92d, 0xdc61a220,
   0x76adf66d, 0x7fa3fd60, 0x64b1e077, 0x6dbfeb7a,
   0x5295da59, 0x5b9bd154, 0x4089cc43, 0x4987c74e,
   0x3eddae05, 0x37d3a508, 0x2cc1b81f, 0x25cfb312,
   0x1ae58231, 0x13eb893c, 0x08f9942b, 0x01f79f26,
   0xe64d46bd, 0xef434db0, 0xf45150a7, 0xfd5f5baa,
   0xc2756a89, 0xcb7b6184, 0xd0697c93, 0xd967779e,
   0xae3d1ed5, 0xa73315d8, 0xbc2108cf, 0xb52f03c2,
   0x8a0532e1, 0x830b39ec, 0x981924fb, 0x91172ff6,
   0x4d768dd6, 0x447886db, 0x5f6a9bcc, 0x566490c1,
   0x694ea1e2, 0x6040aaef, 0x7b52b7f8, 0x725cbcf5,
   0x0506d5be, 0x0c08deb3, 0x171ac3a4, 0x1e14c8a9,
   0x213ef98a, 0x2830f287, 0x3322ef90, 0x3a2ce49d,
   0xdd963d06, 0xd498360b, 0xcf8a2b1c, 0xc6842011,
   0xf9ae1132, 0xf0a01a3f, 0xebb20728, 0xe2bc0c25,
   0x95e6656e, 0x9ce86e63, 0x87fa7374, 0x8ef47879,
   0xb1de495a, 0xb8d04257, 0xa3c25f40, 0xaacc544d,
   0xec41f7da, 0xe54ffcd7, 0xfe5de1c0, 0xf753eacd,
   0xc879dbee, 0xc177d0e3, 0xda65cdf4, 0xd36bc6f9,
   0xa431afb2, 0xad3fa4bf, 0xb62db9a8, 0xbf23b2a5,
   0x80098386, 0x8907888b, 0x9215959c, 0x9b1b9e91,
   0x7ca1470a, 0x75af4c07, 0x6ebd5110, 0x67b35a1d,
   0x58996b3e, 0x51976033, 0x4a857d24, 0x438b7629,
   0x34d11f62, 0x3ddf146f, 0x26cd0978, 0x2fc30275,
   0x10e93356, 0x19e7385b, 0x02f5254c, 0x0bfb2e41,
   0xd79a8c61, 0xde94876c, 0xc5869a7b, 0xcc889176,
   0xf3a2a055, 0xfaacab58, 0xe1beb64f, 0xe8b0bd42,
   0x9fead409, 0x96e4df04, 0x8df6c213, 0x84f8c91e,
   0xbbd2f83d, 0xb2dcf330, 0xa9ceee27, 0xa0c0e52a,
   0x477a3cb1, 0x4e7437bc, 0x55662aab, 0x5c6821a6,
   0x63421085, 0x6a4c1b88, 0x715e069f, 0x78500d92,
   0x0f0a64d9, 0x06046fd4, 0x1d1672c3, 0x141879ce,
   0x2b3248ed, 0x223c43e0, 0x392e5ef7, 0x302055fa,
   0x9aec01b7, 0x93e20aba, 0x88f017ad, 0x81fe1ca0,
   0xbed42d83, 0xb7da268e, 0xacc83b99, 0xa5c63094,
   0xd29c59df, 0xdb9252d2, 0xc0804fc5, 0xc98e44c8,
   0xf6a475eb, 0xffaa7ee6, 0xe4b863f1, 0xedb668fc,
   0x0a0cb167, 0x0302ba6a, 0x1810a77d, 0x111eac70,
   0x2e349d53, 0x273a965e, 0x3c288b49, 0x35268044,
   0x427ce90f, 0x4b72e202, 0x5060ff15, 0x596ef418,
```

```
    0x6644c53b, 0x6f4ace36, 0x7458d321, 0x7d56d82c,
    0xa1377a0c, 0xa8397101, 0xb32b6c16, 0xba25671b,
    0x850f5638, 0x8c015d35, 0x97134022, 0x9e1d4b2f,
    0xe9472264, 0xe0492969, 0xfb5b347e, 0xf2553f73,
    0xcd7f0e50, 0xc471055d, 0xdf63184a, 0xd66d1347,
    0x31d7cadc, 0x38d9c1d1, 0x23cbdcc6, 0x2ac5d7cb,
    0x15efe6e8, 0x1ce1ede5, 0x07f3f0f2, 0x0efdfbff,
    0x79a792b4, 0x70a999b9, 0x6bbb84ae, 0x62b58fa3,
    0x5d9fbe80, 0x5491b58d, 0x4f83a89a, 0x468da397
};
```

```
static const uint32_t I3[] = {
   0x00000000,  0x0e0b0d09,  0x1c161a12,  0x121d171b,
   0x382c3424,  0x3627392d,  0x243a2e36,  0x2a31233f,
   0x70586848,  0x7e536541,  0x6c4e725a,  0x62457f53,
   0x48745c6c,  0x467f5165,  0x5462467e,  0x5a694b77,
   0xe0b0d090,  0xeebbdd99,  0xfca6ca82,  0xf2adc78b,
   0xd89ce4b4,  0xd697e9bd,  0xc48afea6,  0xca81f3af,
   0x90e8b8d8,  0x9ee3b5d1,  0x8cfea2ca,  0x82f5afc3,
   0xa8c48cfc,  0xa6cf81f5,  0xb4d296ee,  0xbad99be7,
   0xdb7bbb3b,  0xd570b632,  0xc76da129,  0xc966ac20,
   0xe3578f1f,  0xed5c8216,  0xff41950d,  0xf14a9804,
   0xab23d373,  0xa528de7a,  0xb735c961,  0xb93ec468,
   0x930fe757,  0x9d04ea5e,  0x8f19fd45,  0x8112f04c,
   0x3bcb6bab,  0x35c066a2,  0x27dd71b9,  0x29d67cb0,
   0x03e75f8f,  0x0dec5286,  0x1ff1459d,  0x11fa4894,
   0x4b9303e3,  0x45980eea,  0x578519f1,  0x598e14f8,
   0x73bf37c7,  0x7db43ace,  0x6fa92dd5,  0x61a220dc,
   0xadf66d76,  0xa3fd607f,  0xb1e07764,  0xbfeb7a6d,
   0x95da5952,  0x9bd1545b,  0x89cc4340,  0x87c74e49,
   0xddae053e,  0xd3a50837,  0xc1b81f2c,  0xcfb31225,
   0xe582311a,  0xeb893c13,  0xf9942b08,  0xf79f2601,
   0x4d46bde6,  0x434db0ef,  0x5150a7f4,  0x5f5baafd,
   0x756a89c2,  0x7b6184cb,  0x697c93d0,  0x67779ed9,
   0x3d1ed5ae,  0x3315d8a7,  0x2108cfbc,  0x2f03c2b5,
   0x0532e18a,  0x0b39ec83,  0x1924fb98,  0x172ff691,
   0x768dd64d,  0x7886db44,  0x6a9bcc5f,  0x6490c156,
   0x4ea1e269,  0x40aaef60,  0x52b7f87b,  0x5cbcf572,
   0x06d5be05,  0x08deb30c,  0x1ac3a417,  0x14c8a91e,
   0x3ef98a21,  0x30f28728,  0x22ef9033,  0x2ce49d3a,
   0x963d06dd,  0x98360bd4,  0x8a2b1ccf,  0x842011c6,
   0xae1132f9,  0xa01a3ff0,  0xb20728eb,  0xbc0c25e2,
   0xe6656e95,  0xe86e639c,  0xfa737487,  0xf478798e,
   0xde495ab1,  0xd04257b8,  0xc25f40a3,  0xcc544daa,
   0x41f7daec,  0x4ffcd7e5,  0x5de1c0fe,  0x53eacdf7,
   0x79dbeec8,  0x77d0e3c1,  0x65cdf4da,  0x6bc6f9d3,
   0x31afb2a4,  0x3fa4bfad,  0x2db9a8b6,  0x23b2a5bf,
   0x09838680,  0x07888b89,  0x15959c92,  0x1b9e919b,
   0xa1470a7c,  0xaf4c0775,  0xbd51106e,  0xb35a1d67,
   0x996b3e58,  0x97603351,  0x857d244a,  0x8b762943,
   0xd11f6234,  0xdf146f3d,  0xcd097826,  0xc302752f,
   0xe9335610,  0xe7385b19,  0xf5254c02,  0xfb2e410b,
   0x9a8c61d7,  0x94876cde,  0x869a7bc5,  0x889176cc,
   0xa2a055f3,  0xacab58fa,  0xbeb64fe1,  0xb0bd42e8,
   0xead4099f,  0xe4df0496,  0xf6c2138d,  0xf8c91e84,
   0xd2f83dbb,  0xdcf330b2,  0xceee27a9,  0xc0e52aa0,
   0x7a3cb147,  0x7437bc4e,  0x662aab55,  0x6821a65c,
   0x42108563,  0x4c1b886a,  0x5e069f71,  0x500d9278,
   0x0a64d90f,  0x046fd406,  0x1672c31d,  0x1879ce14,
   0x3248ed2b,  0x3c43e022,  0x2e5ef739,  0x2055fa30,
   0xec01b79a,  0xe20aba93,  0xf017ad88,  0xfe1ca081,
   0xd42d83be,  0xda268eb7,  0xc83b99ac,  0xc63094a5,
   0x9c59dfd2,  0x9252d2db,  0x804fc5c0,  0x8e44c8c9,
   0xa475ebf6,  0xaa7ee6ff,  0xb863f1e4,  0xb668fced,
   0x0cb1670a,  0x02ba6a03,  0x10a77d18,  0x1eac7011,
   0x349d532e,  0x3a965e27,  0x288b493c,  0x26804435,
   0x7ce90f42,  0x72e2024b,  0x60ff1550,  0x6ef41859,
```

```
    0x44c53b66, 0x4ace366f, 0x58d32174, 0x56d82c7d,
    0x377a0ca1, 0x397101a8, 0x2b6c16b3, 0x25671bba,
    0x0f563885, 0x015d358c, 0x13402297, 0x1d4b2f9e,
    0x472264e9, 0x492969e0, 0x5b347efb, 0x553f73f2,
    0x7f0e50cd, 0x71055dc4, 0x63184adf, 0x6d1347d6,
    0xd7cadc31, 0xd9c1d138, 0xcbdcc623, 0xc5d7cb2a,
    0xefe6e815, 0xe1ede51c, 0xf3f0f207, 0xfdfbff0e,
    0xa792b479, 0xa999b970, 0xbb84ae6b, 0xb58fa362,
    0x9fbe805d, 0x91b58d54, 0x83a89a4f, 0x8da39746
};
```

```
const uint8_t WHIRLPOOL_sbox[256] = {,
   0x18, 0x23, 0xc6, 0xe8, 0x87, 0xb8, 0x01, 0x4f,
   0x36, 0xa6, 0xd2, 0xf5, 0x79, 0x6f, 0x91, 0x52,
   0x60, 0xbc, 0x9b, 0x8e, 0xa3, 0x0c, 0x7b, 0x35,
   0x1d, 0xe0, 0xd7, 0xc2, 0x2e, 0x4b, 0xfe, 0x57,
   0x15, 0x77, 0x37, 0xe5, 0x9f, 0xf0, 0x4a, 0xda,
   0x58, 0xc9, 0x29, 0x0a, 0xb1, 0xa0, 0x6b, 0x85,
   0xbd, 0x5d, 0x10, 0xf4, 0xcb, 0x3e, 0x05, 0x67,
   0xe4, 0x27, 0x41, 0x8b, 0xa7, 0x7d, 0x95, 0xd8,
   0xfb, 0xee, 0x7c, 0x66, 0xdd, 0x17, 0x47, 0x9e,
   0xca, 0x2d, 0xbf, 0x07, 0xad, 0x5a, 0x83, 0x33,
   0x63, 0x02, 0xaa, 0x71, 0xc8, 0x19, 0x49, 0xd9,
   0xf2, 0xe3, 0x5b, 0x88, 0x9a, 0x26, 0x32, 0xb0,
   0xe9, 0x0f, 0xd5, 0x80, 0xbe, 0xcd, 0x34, 0x48,
   0xff, 0x7a, 0x90, 0x5f, 0x20, 0x68, 0x1a, 0xae,
   0xb4, 0x54, 0x93, 0x22, 0x64, 0xf1, 0x73, 0x12,
   0x40, 0x08, 0xc3, 0xec, 0xdb, 0xa1, 0x8d, 0x3d,
   0x97, 0x00, 0xcf, 0x2b, 0x76, 0x82, 0xd6, 0x1b,
   0xb5, 0xaf, 0x6a, 0x50, 0x45, 0xf3, 0x30, 0xef,
   0x3f, 0x55, 0xa2, 0xea, 0x65, 0xba, 0x2f, 0xc0,
   0xde, 0x1c, 0xfd, 0x4d, 0x92, 0x75, 0x06, 0x8a,
   0xb2, 0xe6, 0x0e, 0x1f, 0x62, 0xd4, 0xa8, 0x96,
   0xf9, 0xc5, 0x25, 0x59, 0x84, 0x72, 0x39, 0x4c,
   0x5e, 0x78, 0x38, 0x8c, 0xd1, 0xa5, 0xe2, 0x61,
   0xb3, 0x21, 0x9c, 0x1e, 0x43, 0xc7, 0xfc, 0x04,
   0x51, 0x99, 0x6d, 0x0d, 0xfa, 0xdf, 0x7e, 0x24,
   0x3b, 0xab, 0xce, 0x11, 0x8f, 0x4e, 0xb7, 0xeb,
   0x3c, 0x81, 0x94, 0xf7, 0xb9, 0x13, 0x2c, 0xd3,
   0xe7, 0x6e, 0xc4, 0x03, 0x56, 0x44, 0x7f, 0xa9,
   0x2a, 0xbb, 0xc1, 0x53, 0xdc, 0x0b, 0x9d, 0x6c,
   0x31, 0x74, 0xf6, 0x46, 0xac, 0x89, 0x14, 0xe1,
   0x16, 0x3a, 0x69, 0x09, 0x70, 0xb6, 0xd0, 0xed,
   0xcc, 0x42, 0x98, 0xa4, 0x28, 0x5c, 0xf8, 0x86,
};,
```

```
LE64(0xd7e0ada770dd7070ULL), LE64(0xdee2a8af71d97171ULL),
LE64(0xc5e4a7b772d57272ULL), LE64(0xcce6a2bf73d17373ULL),
LE64(0xf3e8b98774cd7474ULL), LE64(0xfaeabc8f75c97575ULL),
LE64(0xe1ecb39776c57676ULL), LE64(0xe8eeb69f77c17777ULL),
LE64(0x9ff085e778fd7878ULL), LE64(0x96f280ef79f97979ULL),
LE64(0x8df48ff77af57a7aULL), LE64(0x84f68aff7bf17b7bULL),
LE64(0xbbf891c77ced7c7cULL), LE64(0xb2fa94cf7de97d7dULL),
LE64(0xa9fc9bd77ee57e7eULL), LE64(0xa0fe9edf7fe17f7fULL),
LE64(0xf41dba74803a8080ULL), LE64(0xfd1fbf7c813e8181ULL),
LE64(0xe619b06482328282ULL), LE64(0xef1bb56c8368383ULL),
LE64(0xd015ae54842a8484ULL), LE64(0xd917ab5c852e8585ULL),
LE64(0xc211a44486228686ULL), LE64(0xcb13a14c87268787ULL),
LE64(0xbc0d9234881a8888ULL), LE64(0xb50f973c891e8989ULL),
LE64(0xae0998248a128a8aULL), LE64(0xa70b9d2c8b168b8bULL),
LE64(0x980586148c0a8c8cULL), LE64(0x9107831c8d0e8d8dULL),
LE64(0x8a018c048e028e8eULL), LE64(0x8303890c8f068f8fULL),
LE64(0x643deaf4907a9090ULL), LE64(0x6d3feffc917e9191ULL),
LE64(0x7639e0e492729292ULL), LE64(0x7f3be5ec93769393ULL),
LE64(0x4035fed4946a9494ULL), LE64(0x4937fbdc956e9595ULL),
LE64(0x5231f4c496629696ULL), LE64(0x5b33f1cc97669797ULL),
LE64(0x2c2dc2b4985a9898ULL), LE64(0x252fc7bc995e9999ULL),
LE64(0x3e29c8a49a529a9aULL), LE64(0x372bcdac9b569b9bULL),
LE64(0x0825d6949c4a9c9cULL), LE64(0x0127d39c9d4e9d9dULL),
LE64(0x1a21dc849e429e9eULL), LE64(0x1323d98c9f469f9fULL),
LE64(0xc95d1a69a0baa0a0ULL), LE64(0xc05f1f61a1bea1a1ULL),
LE64(0xdb591079a2b2a2a2ULL), LE64(0xd25b1571a3b6a3a3ULL),
LE64(0xed550e49a4aaa4a4ULL), LE64(0xe4570b41a5aea5a5ULL),
LE64(0xff510459a6a2a6a6ULL), LE64(0xf6530151a7a6a7a7ULL),
LE64(0x814d3229a89aa8a8ULL), LE64(0x884f3721a99ea9a9ULL),
LE64(0x93493839aa92aaaaULL), LE64(0x9a4b3d31ab96ababULL),
LE64(0xa5452609ac8aacacULL), LE64(0xac472301ad8eadadULL),
LE64(0xb7412c19ae82aeaeULL), LE64(0xbe432911af86afafULL),
LE64(0x597d4ae9b0fab0b0ULL), LE64(0x507f4fe1b1feb1b1ULL),
LE64(0x4b7940f9b2f2b2b2ULL), LE64(0x427b45f1b3f6b3b3ULL),
LE64(0x7d755ec9b4eab4b4ULL), LE64(0x74775bc1b5eeb5b5ULL),
LE64(0x6f7154d9b6e2b6b6ULL), LE64(0x667351d1b7e6b7b7ULL),
LE64(0x116d62a9b8dab8b8ULL), LE64(0x186f67a1b9deb9b9ULL),
LE64(0x036968b9bad2babaULL), LE64(0x0a6b6db1bbd6bbbbULL),
LE64(0x35657689bccabcbcULL), LE64(0x3c677381bdcebdbdULL),
LE64(0x27617c99bec2bebeULL), LE64(0x2e637991bfc6bfbfULL),
LE64(0x8e9de74ec027c0c0ULL), LE64(0x879fe246c123c1c1ULL),
LE64(0x9c99ed5ec22fc2c2ULL), LE64(0x959be856c32bc3c3ULL),
LE64(0xaa95f36ec437c4c4ULL), LE64(0xa397f666c533c5c5ULL),
LE64(0xb891f97ec63fc6c6ULL), LE64(0xb193fc76c73bc7c7ULL),
LE64(0xc68dcf0ec807c8c8ULL), LE64(0xcf8fca06c903c9c9ULL),
LE64(0xd489c51eca0fcacaULL), LE64(0xdd8bc016cb0bcbcbULL),
LE64(0xe285db2ecc17ccccULL), LE64(0xeb87de26cd13cdcdULL),
LE64(0xf081d13ece1fceceULL), LE64(0xf983d436cf1bcfcfULL),
LE64(0x1ebdb7ced067d0d0ULL), LE64(0x17bfb2c6d163d1d1ULL),
LE64(0x0cb9bdded26fd2d2ULL), LE64(0x05bbb8d6d36bd3d3ULL),
LE64(0x3ab5a3eed477d4d4ULL), LE64(0x33b7a6e6d573d5d5ULL),
LE64(0x28b1a9fed67fd6d6ULL), LE64(0x21b3acf6d77bd7d7ULL),
LE64(0x56ad9f8ed847d8d8ULL), LE64(0x5faf9a86d943d9d9ULL),
LE64(0x44a9959eda4fdadaULL), LE64(0x4dab9096db4bdbdbULL),
LE64(0x72a58baedc57dcdcULL), LE64(0x7ba78ea6dd53ddddULL),
LE64(0x60a181bede5fdedeULL), LE64(0x69a384b6df5bdfdfULL),
LE64(0xb3dd4753e0a7e0e0ULL), LE64(0xbadf425be1a3e1e1ULL),
```

```
    LE64(0xa1d94d43e2afe2e2ULL), LE64(0xa8db484be3abe3e3ULL),
    LE64(0x97d55373e4b7e4e4ULL), LE64(0x9ed7567be5b3e5e5ULL),
    LE64(0x85d15963e6bfe6e6ULL), LE64(0x8cd35c6be7bbe7e7ULL),
    LE64(0xfbcd6f13e887e8e8ULL), LE64(0xf2cf6a1be983e9e9ULL),
    LE64(0xe9c96503ea8feaeaULL), LE64(0xe0cb600beb8bebebULL),
    LE64(0xdfc57b33ec97ececULL), LE64(0xd6c77e3bed93ededULL),
    LE64(0xcdc17123ee9feeeeULL), LE64(0xc4c3742bef9befefULL),
    LE64(0x23fd17d3f0e7f0f0ULL), LE64(0x2aff12dbf1e3f1f1ULL),
    LE64(0x31f91dc3f2eff2f2ULL), LE64(0x38fb18cbf3ebf3f3ULL),
    LE64(0x07f503f3f4f7f4f4ULL), LE64(0x0ef706fbf5f3f5f5ULL),
    LE64(0x15f109e3f6fff6f6ULL), LE64(0x1cf30cebf7fbf7f7ULL),
    LE64(0x6bed3f93f8c7f8f8ULL), LE64(0x62ef3a9bf9c3f9f9ULL),
    LE64(0x79e93583facffafaULL), LE64(0x70eb308bfbcbfbfbULL),
    LE64(0x4fe52bb3fcd7fcfcULL), LE64(0x46e72ebbfdd3fdfdULL),
    LE64(0x5de121a3fedffefeULL), LE64(0x54e324abffdbffffULL),
};,
```

```
const uint64_t W1[] = {,
   LE64(0x0000000000000000ULL), LE64(0x0205080104010109ULL),
   LE64(0x040a100208020212ULL), LE64(0x060f18030c03031bULL),
   LE64(0x0814200410040424ULL), LE64(0x0a1128051405052dULL),
   LE64(0x0c1e300618060636ULL), LE64(0x0e1b38071c07073fULL),
   LE64(0x1028400820080848ULL), LE64(0x122d480924090941ULL),
   LE64(0x1422500a280a0a5aULL), LE64(0x1627580b2c0b0b53ULL),
   LE64(0x183c600c300c0c6cULL), LE64(0x1a39680d340d0d65ULL),
   LE64(0x1c36700e380e0e7eULL), LE64(0x1e33780f3c0f0f77ULL),
   LE64(0x2050801040101090ULL), LE64(0x22558811441111199ULL),
   LE64(0x245a901248121282ULL), LE64(0x265f98134c13138bULL),
   LE64(0x2844a014501414b4ULL), LE64(0x2a41a815541515bdULL),
   LE64(0x2c4eb016581616a6ULL), LE64(0x2e4bb8175c1717afULL),
   LE64(0x3078c018601818d8ULL), LE64(0x327dc819641919d1ULL),
   LE64(0x3472d01a681a1acaULL), LE64(0x3677d81b6c1b1bc3ULL),
   LE64(0x386ce01c701c1cfcULL), LE64(0x3a69e81d741d1df5ULL),
   LE64(0x3c66f01e781e1eeeULL), LE64(0x3e63f81f7c1f1fe7ULL),
   LE64(0x40a01d208020203dULL), LE64(0x42a5152184212134ULL),
   LE64(0x44aa0d228822222fULL), LE64(0x46af05238c232326ULL),
   LE64(0x48b43d2490242419ULL), LE64(0x4ab13525942525l0ULL),
   LE64(0x4cbe2d269826260bULL), LE64(0x4ebb25279c272702ULL),
   LE64(0x50885d28a0282875ULL), LE64(0x528d5529a429297cULL),
   LE64(0x54824d2aa82a2a67ULL), LE64(0x5687452bac2b2b6eULL),
   LE64(0x589c7d2cb02c2c51ULL), LE64(0x5a99752db42d2d58ULL),
   LE64(0x5c966d2eb82e2e43ULL), LE64(0x5e93652fbc2f2f4aULL),
   LE64(0x60f09d30c03030adULL), LE64(0x62f59531c43131a4ULL),
   LE64(0x64fa8d32c83232bfULL), LE64(0x66ff8533cc3333b6ULL),
   LE64(0x68e4bd34d0343489ULL), LE64(0x6ae1b535d4353580ULL),
   LE64(0x6ceead36d836369bULL), LE64(0x6eeba537dc373792ULL),
   LE64(0x70d8dd38e03838e5ULL), LE64(0x72ddd539e43939ecULL),
   LE64(0x74d2cd3ae83a3af7ULL), LE64(0x76d7c53bec3b3bfeULL),
   LE64(0x78ccfd3cf03c3cc1ULL), LE64(0x7ac9f53df43d3dc8ULL),
   LE64(0x7cc6ed3ef83e3ed3ULL), LE64(0x7ec3e53ffc3f3fdaULL),
   LE64(0x805d3a401d40407aULL), LE64(0x8258324119414173ULL),
   LE64(0x84572a4215424268ULL), LE64(0x8652224311434361ULL),
   LE64(0x88491a440d44445eULL), LE64(0x8a4c124509454557ULL),
   LE64(0x8c430a460546464cULL), LE64(0x8e46024701474745ULL),
   LE64(0x90757a483d484832ULL), LE64(0x9270724939494 3bULL),
   LE64(0x947f6a4a354a4a20ULL), LE64(0x967a624b314b4b29ULL),
   LE64(0x98615a4c2d4c4c16ULL), LE64(0x9a64524d294d4d1fULL),
   LE64(0x9c6b4a4e254e4e04ULL), LE64(0x9e6e424f214f4f0dULL),
   LE64(0xa00dba505d5050eaULL), LE64(0xa208b251595151e3ULL),
   LE64(0xa407aa52555252f8ULL), LE64(0xa602a253515353f1ULL),
   LE64(0xa8199a544d5454ceULL), LE64(0xaa1c9255495555c7ULL),
   LE64(0xac138a56455656dcULL), LE64(0xae168257415757d5ULL),
   LE64(0xb025fa587d5858a2ULL), LE64(0xb220f259795959abULL),
   LE64(0xb42fea5a755a5ab0ULL), LE64(0xb62ae25b715b5bb9ULL),
   LE64(0xb831da5c6d5c5c86ULL), LE64(0xba34d25d695d5d8fULL),
   LE64(0xbc3bca5e655e5e94ULL), LE64(0xbe3ec25f615f5f9dULL),
   LE64(0xc0fd27609d606047ULL), LE64(0xc2f82f619961614eULL),
   LE64(0xc4f7376295626255ULL), LE64(0xc6f23f639163635cULL),
   LE64(0xc8e907648d646463ULL), LE64(0xcaec0f658965656aULL),
   LE64(0xcce3176685666671ULL), LE64(0xcee61f6781676778ULL),
   LE64(0xd0d56768bd68680fULL), LE64(0xd2d06f69b9696906ULL),
   LE64(0xd4df776ab56a6a1dULL), LE64(0xd6da7f6bb16b6b14ULL),
   LE64(0xd8c1476cad6c6c2bULL), LE64(0xdac44f6da96d6d22ULL),
   LE64(0xdccb576ea56e6e39ULL), LE64(0xdece5f6fa16f6f30ULL),
```

```
LE64(0xe0ada770dd7070d7ULL), LE64(0xe2a8af71d97171deULL),
LE64(0xe4a7b772d57272c5ULL), LE64(0xe6a2bf73d17373ccULL),
LE64(0xe8b98774cd7474f3ULL), LE64(0xeabc8f75c97575faULL),
LE64(0xecb39776c57676e1ULL), LE64(0xeeb69f77c17777e8ULL),
LE64(0xf085e778fd78789fULL), LE64(0xf280ef79f9797996ULL),
LE64(0xf48ff77af57a7a8dULL), LE64(0xf68aff7bf17b7b84ULL),
LE64(0xf891c77ced7c7cbbULL), LE64(0xfa94cf7de97d7db2ULL),
LE64(0xfc9bd77ee57e7ea9ULL), LE64(0xfe9edf7fe17f7fa0ULL),
LE64(0x1dba74803a8080f4ULL), LE64(0x1fbf7c813e8181fdULL),
LE64(0x19b06482328282e6ULL), LE64(0x1bb56c83368383efULL),
LE64(0x15ae54842a8484d0ULL), LE64(0x17ab5c852e8585d9ULL),
LE64(0x11a44486228686c2ULL), LE64(0x13a14c87268787cbULL),
LE64(0x0d9234881a8888bcULL), LE64(0x0f973c891e8989b5ULL),
LE64(0x0998248a128a8aaeULL), LE64(0x0b9d2c8b168b8ba7ULL),
LE64(0x0586148c0a8c8c98ULL), LE64(0x07831c8d0e8d8d91ULL),
LE64(0x018c048e028e8e8aULL), LE64(0x03890c8f068f8f83ULL),
LE64(0x3deaf4907a909064ULL), LE64(0x3feffc917e91916dULL),
LE64(0x39e0e49272929276ULL), LE64(0x3be5ec937693937fULL),
LE64(0x35fed4946a949440ULL), LE64(0x37fbdc956e959549ULL),
LE64(0x31f4c49662969652ULL), LE64(0x33f1cc976697975bULL),
LE64(0x2dc2b4985a98982cULL), LE64(0x2fc7bc995e999925ULL),
LE64(0x29c8a49a529a9a3eULL), LE64(0x2bcdac9b569b9b37ULL),
LE64(0x25d6949c4a9c9c08ULL), LE64(0x27d39c9d4e9d9d01ULL),
LE64(0x21dc849e429e9e1aULL), LE64(0x23d98c9f469f9f13ULL),
LE64(0x5d1a69a0baa0a0c9ULL), LE64(0x5f1f61a1bea1a1c0ULL),
LE64(0x591079a2b2a2a2dbULL), LE64(0x5b1571a3b6a3a3d2ULL),
LE64(0x550e49a4aaa4a4edULL), LE64(0x570b41a5aea5a5e4ULL),
LE64(0x510459a6a2a6a6ffULL), LE64(0x530151a7a6a7a7f6ULL),
LE64(0x4d3229a89aa8a881ULL), LE64(0x4f3721a99ea9a988ULL),
LE64(0x493839aa92aaaa93ULL), LE64(0x4b3d31ab96abab9aULL),
LE64(0x452609ac8aacaca5ULL), LE64(0x472301ad8eadadacULL),
LE64(0x412c19ae82aeaeb7ULL), LE64(0x432911af86afafbeULL),
LE64(0x7d4ae9b0fab0b059ULL), LE64(0x7f4fe1b1feb1b150ULL),
LE64(0x7940f9b2f2b2b24bULL), LE64(0x7b45f1b3f6b3b342ULL),
LE64(0x755ec9b4eab4b47dULL), LE64(0x775bc1b5eeb5b574ULL),
LE64(0x7154d9b6e2b6b66fULL), LE64(0x7351d1b7e6b7b766ULL),
LE64(0x6d62a9b8dab8b811ULL), LE64(0x6f67a1b9deb9b918ULL),
LE64(0x6968b9bad2baba03ULL), LE64(0x6b6db1bbd6bbbb0aULL),
LE64(0x657689bccabcbc35ULL), LE64(0x677381bdcebdbd3cULL),
LE64(0x617c99bec2bebe27ULL), LE64(0x637991bfc6bfbf2eULL),
LE64(0x9de74ec027c0c08eULL), LE64(0x9fe246c123c1c187ULL),
LE64(0x99ed5ec22fc2c29cULL), LE64(0x9be856c32bc3c395ULL),
LE64(0x95f36ec437c4c4aaULL), LE64(0x97f666c533c5c5a3ULL),
LE64(0x91f97ec63fc6c6b8ULL), LE64(0x93fc76c73bc7c7b1ULL),
LE64(0x8dcf0ec807c8c8c6ULL), LE64(0x8fca06c903c9c9cfULL),
LE64(0x89c51eca0fcacad4ULL), LE64(0x8bc016cb0bcbcbddULL),
LE64(0x85db2ecc17cccce2ULL), LE64(0x87de26cd13cdcdebULL),
LE64(0x81d13ece1fcecef0ULL), LE64(0x83d436cf1bcfcff9ULL),
LE64(0xbdb7ced067d0d01eULL), LE64(0xbfb2c6d163d1d117ULL),
LE64(0xb9bded26fd2d20cULL), LE64(0xbbb8d6d36bd3d305ULL),
LE64(0xb5a3eed477d4d43aULL), LE64(0xb7a6e6d573d5d533ULL),
LE64(0xb1a9fed67fd6d628ULL), LE64(0xb3acf6d77bd7d721ULL),
LE64(0xad9f8ed847d8d856ULL), LE64(0xaf9a86d943d9d95fULL),
LE64(0xa9959eda4fdada44ULL), LE64(0xab9096db4bdbdb4dULL),
LE64(0xa58baedc57dcdc72ULL), LE64(0xa78ea6dd53dddd7bULL),
LE64(0xa181bede5fdede60ULL), LE64(0xa384b6df5bdfdf69ULL),
LE64(0xdd4753e0a7e0e0b3ULL), LE64(0xdf425be1a3e1e1baULL),
```

```
    LE64(0xd94d43e2afe2e2a1ULL), LE64(0xdb484be3abe3e3a8ULL),
    LE64(0xd55373e4b7e4e497ULL), LE64(0xd7567be5b3e5e59eULL),
    LE64(0xd15963e6bfe6e685ULL), LE64(0xd35c6be7bbe7e78cULL),
    LE64(0xcd6f13e887e8e8fbULL), LE64(0xcf6a1be983e9e9f2ULL),
    LE64(0xc96503ea8feaeae9ULL), LE64(0xcb600beb8bebebe0ULL),
    LE64(0xc57b33ec97ececdfULL), LE64(0xc77e3bed93ededd6ULL),
    LE64(0xc17123ee9feeeecdULL), LE64(0xc3742bef9befefc4ULL),
    LE64(0xfd17d3f0e7f0f023ULL), LE64(0xff12dbf1e3f1f12aULL),
    LE64(0xf91dc3f2eff2f231ULL), LE64(0xfb18cbf3ebf3f338ULL),
    LE64(0xf503f3f4f7f4f407ULL), LE64(0xf706fbf5f3f5f50eULL),
    LE64(0xf109e3f6fff6f615ULL), LE64(0xf30cebf7fbf7f71cULL),
    LE64(0xed3f93f8c7f8f86bULL), LE64(0xef3a9bf9c3f9f962ULL),
    LE64(0xe93583facffafa79ULL), LE64(0xeb308bfbcbfbfb70ULL),
    LE64(0xe52bb3fcd7fcfc4fULL), LE64(0xe72ebbfdd3fdfd46ULL),
    LE64(0xe121a3fedffefe5dULL), LE64(0xe324abffdbffff54ULL),
};
```

```
const uint64_t W2[] = {,
   LE64(0x0000000000000000ULL), LE64(0x0508010401010902ULL),
   LE64(0x0a1002080202120 4ULL), LE64(0x0f18030c03031b06ULL),
   LE64(0x1420041004042408ULL), LE64(0x1128051405052d0aULL),
   LE64(0x1e3006180606360cULL), LE64(0x1b38071c07073f0eULL),
   LE64(0x2840082008084810ULL), LE64(0x2d480924090941 12ULL),
   LE64(0x22500a280a0a5a14ULL), LE64(0x27580b2c0b0b5316ULL),
   LE64(0x3c600c300c0c6c18ULL), LE64(0x39680d340d0d651aULL),
   LE64(0x36700e380e0e7e1cULL), LE64(0x33780f3c0f0f771eULL),
   LE64(0x5080104010109020ULL), LE64(0x5588114411119922ULL),
   LE64(0x5a9012481212822 4ULL), LE64(0x5f98134c13138b26ULL),
   LE64(0x44a014501414b428ULL), LE64(0x41a815541515bd2aULL),
   LE64(0x4eb016581616a62cULL), LE64(0x4bb8175c1717af2eULL),
   LE64(0x78c018601818d830ULL), LE64(0x7dc819641919d132ULL),
   LE64(0x72d01a681a1aca34ULL), LE64(0x77d81b6c1b1bc336ULL),
   LE64(0x6ce01c701c1cfc38ULL), LE64(0x69e81d741d1df53aULL),
   LE64(0x66f01e781e1eee3cULL), LE64(0x63f81f7c1f1fe73eULL),
   LE64(0xa01d208020203d40ULL), LE64(0xa5152184212134 42ULL),
   LE64(0xaa0d228822222f44ULL), LE64(0xaf05238c23232646ULL),
   LE64(0xb43d249024241948ULL), LE64(0xb13525942525104aULL),
   LE64(0xbe2d269826260b4cULL), LE64(0xbb25279c2727024eULL),
   LE64(0x885d28a028287550ULL), LE64(0x8d5529a429297c52ULL),
   LE64(0x824d2aa82a2a6754ULL), LE64(0x87452bac2b2b6e56ULL),
   LE64(0x9c7d2cb02c2c5158ULL), LE64(0x99752db42d2d585aULL),
   LE64(0x966d2eb82e2e435cULL), LE64(0x93652fbc2f2f4a5eULL),
   LE64(0xf09d30c03030ad60ULL), LE64(0xf59531c43131a462ULL),
   LE64(0xfa8d32c83232bf64ULL), LE64(0xff8533cc3333b666ULL),
   LE64(0xe4bd34d034348968ULL), LE64(0xe1b535d43535806aULL),
   LE64(0xeead36d836369b6cULL), LE64(0xeba537dc3737926eULL),
   LE64(0xd8dd38e03838e570ULL), LE64(0xddd539e43939ec72ULL),
   LE64(0xd2cd3ae83a3af774ULL), LE64(0xd7c53bec3b3bfe76ULL),
   LE64(0xccfd3cf03c3cc178ULL), LE64(0xc9f53df43d3dc87aULL),
   LE64(0xc6ed3ef83e3ed37cULL), LE64(0xc3e53ffc3f3fda7eULL),
   LE64(0x5d3a401d40407a80ULL), LE64(0x5832411941417382ULL),
   LE64(0x572a421542426884ULL), LE64(0x5222431143436186ULL),
   LE64(0x491a440d44445e88ULL), LE64(0x4c1245094545578aULL),
   LE64(0x430a460546464c8cULL), LE64(0x46024701474745 8eULL),
   LE64(0x757a483d48483290ULL), LE64(0x7072493949493b92ULL),
   LE64(0x7f6a4a354a4a2094ULL), LE64(0x7a624b314b4b2996ULL),
   LE64(0x615a4c2d4c4c1698ULL), LE64(0x64524d294d4d1f9aULL),
   LE64(0x6b4a4e254e4e049cULL), LE64(0x6e424f214f4f0d9eULL),
   LE64(0x0dba505d5050eaa0ULL), LE64(0x08b251595151e3a2ULL),
   LE64(0x07aa52555252f8a4ULL), LE64(0x02a253515353f1a6ULL),
   LE64(0x199a544d5454cea8ULL), LE64(0x1c9255495555c7aaULL),
   LE64(0x138a56455656dcacULL), LE64(0x168257415757d5aeULL),
   LE64(0x25fa587d5858a2b0ULL), LE64(0x20f259795959abb2ULL),
   LE64(0x2fea5a755a5ab0b4ULL), LE64(0x2ae25b715b5bb9b6ULL),
   LE64(0x31da5c6d5c5c86b8ULL), LE64(0x34d25d695d5d8fbaULL),
   LE64(0x3bca5e655e5e94bcULL), LE64(0x3ec25f615f5f9dbeULL),
   LE64(0xfd27609d606047c0ULL), LE64(0xf82f619961614ec2ULL),
   LE64(0xf7376295626255c4ULL), LE64(0xf23f639163635cc6ULL),
   LE64(0xe907648d646463c8ULL), LE64(0xec0f658965656acaULL),
   LE64(0xe3176685666671ccULL), LE64(0xe61f6781676778ceULL),
   LE64(0xd56768bd68680fd0ULL), LE64(0xd06f69b9696906d2ULL),
   LE64(0xdf776ab56a6a1dd4ULL), LE64(0xda7f6bb16b6b14d6ULL),
   LE64(0xc1476cad6c6c2bd8ULL), LE64(0xc44f6da96d6d22daULL),
   LE64(0xcb576ea56e6e39dcULL), LE64(0xce5f6fa16f6f30deULL),
```

```
LE64(0xada770dd7070d7e0ULL), LE64(0xa8af71d97171dee2ULL),
LE64(0xa7b772d57272c5e4ULL), LE64(0xa2bf73d17373cce6ULL),
LE64(0xb98774cd7474f3e8ULL), LE64(0xbc8f75c97575faeaULL),
LE64(0xb39776c57676e1ecULL), LE64(0xb69f77c17777e8eeULL),
LE64(0x85e778fd78789ff0ULL), LE64(0x80ef79f9797996f2ULL),
LE64(0x8ff77af57a7a8df4ULL), LE64(0x8aff7bf17b7b84f6ULL),
LE64(0x91c77ced7c7cbbf8ULL), LE64(0x94cf7de97d7db2faULL),
LE64(0x9bd77ee57e7ea9fcULL), LE64(0x9edf7fe17f7fa0feULL),
LE64(0xba74803a8080f41dULL), LE64(0xbf7c813e8181fd1fULL),
LE64(0xb06482328282e619ULL), LE64(0xb56c83368383ef1bULL),
LE64(0xae54842a8484d015ULL), LE64(0xab5c852e8585d917ULL),
LE64(0xa44486228686c211ULL), LE64(0xa14c87268787cb13ULL),
LE64(0x9234881a8888bc0dULL), LE64(0x973c891e8989b50fULL),
LE64(0x98248a128a8aae09ULL), LE64(0x9d2c8b168b8ba70bULL),
LE64(0x86148c0a8c8c9805ULL), LE64(0x831c8d0e8d8d9107ULL),
LE64(0x8c048e028e8e8a01ULL), LE64(0x890c8f068f8f8303ULL),
LE64(0xeaf4907a9090643dULL), LE64(0xeffc917e91916d3fULL),
LE64(0xe0e4927292927639ULL), LE64(0xe5ec937693937f3bULL),
LE64(0xfed4946a94944035ULL), LE64(0xfbdc956e95954937ULL),
LE64(0xf4c4966296965231ULL), LE64(0xf1cc976697975b33ULL),
LE64(0xc2b4985a98982c2dULL), LE64(0xc7bc995e9999252fULL),
LE64(0xc8a49a529a9a3e29ULL), LE64(0xcdac9b569b9b372bULL),
LE64(0xd6949c4a9c9c0825ULL), LE64(0xd39c9d4e9d9d0127ULL),
LE64(0xdc849e429e9e1a21ULL), LE64(0xd98c9f469f9f1323ULL),
LE64(0x1a69a0baa0a0c95dULL), LE64(0x1f61a1bea1a1c05fULL),
LE64(0x1079a2b2a2a2db59ULL), LE64(0x1571a3b6a3a3d25bULL),
LE64(0x0e49a4aaa4a4ed55ULL), LE64(0x0b41a5aea5a5e457ULL),
LE64(0x0459a6a2a6a6ff51ULL), LE64(0x0151a7a6a7a7f653ULL),
LE64(0x3229a89aa8a8814dULL), LE64(0x3721a99ea9a9884fULL),
LE64(0x3839aa92aaaa9349ULL), LE64(0x3d31ab96abab9a4bULL),
LE64(0x2609ac8aacaca545ULL), LE64(0x2301ad8eadadac47ULL),
LE64(0x2c19ae82aeaeb741ULL), LE64(0x2911af86afafbe43ULL),
LE64(0x4ae9b0fab0b0597dULL), LE64(0x4fe1b1feb1b1507fULL),
LE64(0x40f9b2f2b2b24b79ULL), LE64(0x45f1b3f6b3b3427bULL),
LE64(0x5ec9b4eab4b47d75ULL), LE64(0x5bc1b5eeb5b57477ULL),
LE64(0x54d9b6e2b6b66f71ULL), LE64(0x51d1b7e6b7b76673ULL),
LE64(0x62a9b8dab8b8116dULL), LE64(0x67a1b9deb9b9186fULL),
LE64(0x68b9bad2baba0369ULL), LE64(0x6db1bbd6bbbb0a6bULL),
LE64(0x7689bccabcbc3565ULL), LE64(0x7381bdcebdbd3c67ULL),
LE64(0x7c99bec2bebe2761ULL), LE64(0x7991bfc6bfbf2e63ULL),
LE64(0xe74ec027c0c08e9dULL), LE64(0xe246c123c1c1879fULL),
LE64(0xed5ec22fc2c29c99ULL), LE64(0xe856c32bc3c3959bULL),
LE64(0xf36ec437c4c4aa95ULL), LE64(0xf666c533c5c5a397ULL),
LE64(0xf97ec63fc6c6b891ULL), LE64(0xfc76c73bc7c7b193ULL),
LE64(0xcf0ec807c8c8c68dULL), LE64(0xca06c903c9c9cf8fULL),
LE64(0xc51eca0fcacad489ULL), LE64(0xc016cb0bcbcbdd8bULL),
LE64(0xdb2ecc17cccce285ULL), LE64(0xde26cd13cdcdeb87ULL),
LE64(0xd13ece1fcecef081ULL), LE64(0xd436cf1bcfcff983ULL),
LE64(0xb7ced067d0d01ebdULL), LE64(0xb2c6d163d1d117bfULL),
LE64(0xbdded26fd2d20cb9ULL), LE64(0xb8d6d36bd3d305bbULL),
LE64(0xa3eed477d4d43ab5ULL), LE64(0xa6e6d573d5d533b7ULL),
LE64(0xa9fed67fd6d628b1ULL), LE64(0xacf6d77bd7d721b3ULL),
LE64(0x9f8ed847d8d856adULL), LE64(0x9a86d943d9d95fafULL),
LE64(0x959eda4fdada44a9ULL), LE64(0x9096db4bdbdb4dabULL),
LE64(0x8baedc57dcdc72a5ULL), LE64(0x8ea6dd53dddd7ba7ULL),
LE64(0x81bede5fdede60a1ULL), LE64(0x84b6df5bdfdf69a3ULL),
LE64(0x4753e0a7e0e0b3ddULL), LE64(0x425be1a3e1e1badfULL),
```

```
    LE64(0x4d43e2afe2e2a1d9ULL), LE64(0x484be3abe3e3a8dbULL),
    LE64(0x5373e4b7e4e497d5ULL), LE64(0x567be5b3e5e59ed7ULL),
    LE64(0x5963e6bfe6e685d1ULL), LE64(0x5c6be7bbe7e78cd3ULL),
    LE64(0x6f13e887e8e8fbcdULL), LE64(0x6a1be983e9e9f2cfULL),
    LE64(0x6503ea8feaeae9c9ULL), LE64(0x600beb8bebebe0cbULL),
    LE64(0x7b33ec97ececdfc5ULL), LE64(0x7e3bed93ededd6c7ULL),
    LE64(0x7123ee9feeeecdc1ULL), LE64(0x742bef9befefc4c3ULL),
    LE64(0x17d3f0e7f0f023fdULL), LE64(0x12dbf1e3f1f12affULL),
    LE64(0x1dc3f2eff2f231f9ULL), LE64(0x18cbf3ebf3f338fbULL),
    LE64(0x03f3f4f7f4f407f5ULL), LE64(0x06fbf5f3f5f50ef7ULL),
    LE64(0x09e3f6fff6f615f1ULL), LE64(0x0cebf7fbf7f71cf3ULL),
    LE64(0x3f93f8c7f8f86bedULL), LE64(0x3a9bf9c3f9f962efULL),
    LE64(0x3583facffafa79e9ULL), LE64(0x308bfbcbfbfb70ebULL),
    LE64(0x2bb3fcd7fcfc4fe5ULL), LE64(0x2ebbfdd3fdfd46e7ULL),
    LE64(0x21a3fedffefe5de1ULL), LE64(0x24abffdbffff54e3ULL),
};
```

```
const uint64_t W3[] = {,
   LE64(0x0000000000000000ULL), LE64(0x0801040101090205ULL),
   LE64(0x1002080202120402aULL), LE64(0x18030c03031b060fULL),
   LE64(0x2004100404240814ULL), LE64(0x28051405052d0a11ULL),
   LE64(0x3006180606360c1eULL), LE64(0x38071c07073f0e1bULL),
   LE64(0x4008200808481028ULL), LE64(0x4809240909411220dULL),
   LE64(0x500a280a0a5a1422ULL), LE64(0x580b2c0b0b531627ULL),
   LE64(0x600c300c0c6c183cULL), LE64(0x680d340d0d651a39ULL),
   LE64(0x700e380e0e7e1c36ULL), LE64(0x780f3c0f0f771e33ULL),
   LE64(0x8010401010902050ULL), LE64(0x8811441111992255ULL),
   LE64(0x9012481212822450aULL), LE64(0x98134c13138b265fULL),
   LE64(0xa014501414b42844ULL), LE64(0xa815541515bd2a41ULL),
   LE64(0xb016581616a62c4eULL), LE64(0xb8175c1717af2e4bULL),
   LE64(0xc018601818d83078ULL), LE64(0xc819641919d1327dULL),
   LE64(0xd01a681a1aca3472ULL), LE64(0xd81b6c1b1bc33677ULL),
   LE64(0xe01c701c1cfc386cULL), LE64(0xe81d741d1df53a69ULL),
   LE64(0xf01e781e1eee3c66ULL), LE64(0xf81f7c1f1fe73e63ULL),
   LE64(0x1d208020203d40a0ULL), LE64(0x152184212134420a5ULL),
   LE64(0x0d228822222f44aaULL), LE64(0x05238c23232646afULL),
   LE64(0x3d2490242419448b4ULL), LE64(0x3525942525104ab1ULL),
   LE64(0x2d269826260b4cbeULL), LE64(0x25279c2727024ebbULL),
   LE64(0x5d28a02828755088ULL), LE64(0x5529a429297c528dULL),
   LE64(0x4d2aa82a2a675482ULL), LE64(0x452bac2b2b6e5687ULL),
   LE64(0x7d2cb02c2c51589cULL), LE64(0x752db42d2d585a99ULL),
   LE64(0x6d2eb82e2e435c96ULL), LE64(0x652fbc2f2f4a5e93ULL),
   LE64(0x9d30c03030ad60f0ULL), LE64(0x9531c43131a462f5ULL),
   LE64(0x8d32c83232bf64faULL), LE64(0x8533cc3333b666ffULL),
   LE64(0xbd34d034348968e4ULL), LE64(0xb535d43535806ae1ULL),
   LE64(0xad36d836369b6ceeULL), LE64(0xa537dc3737926eebULL),
   LE64(0xdd38e03838e570d8ULL), LE64(0xd539e43939ec72ddULL),
   LE64(0xcd3ae83a3af774d2ULL), LE64(0xc53bec3b3bfe76d7ULL),
   LE64(0xfd3cf03c3cc178ccULL), LE64(0xf53df43d3dc87ac9ULL),
   LE64(0xed3ef83e3ed37cc6ULL), LE64(0xe53ffc3f3fda7ec3ULL),
   LE64(0x3a401d40407a805dULL), LE64(0x32411941417382580ULL),
   LE64(0x2a42154242688457ULL), LE64(0x22431143436186520ULL),
   LE64(0x1a440d44445e8849ULL), LE64(0x1245094545578a4cULL),
   LE64(0x0a460546464c8c43ULL), LE64(0x0247014747458e46ULL),
   LE64(0x7a483d4848329075ULL), LE64(0x72493949493b9270ULL),
   LE64(0x6a4a354a4a20947fULL), LE64(0x624b314b4b29967aULL),
   LE64(0x5a4c2d4c4c169861ULL), LE64(0x524d294d4d1f9a64ULL),
   LE64(0x4a4e254e4e049c6bULL), LE64(0x424f214f4f0d9e6eULL),
   LE64(0xba505d5050eaa00dULL), LE64(0xb251595151e3a208ULL),
   LE64(0xaa52555252f8a407ULL), LE64(0xa253515353f1a602ULL),
   LE64(0x9a544d5454cea819ULL), LE64(0x9255495555c7aa1cULL),
   LE64(0x8a56455656dcac13ULL), LE64(0x8257415757d5ae16ULL),
   LE64(0xfa587d5858a2b025ULL), LE64(0xf259795959abb220ULL),
   LE64(0xea5a755a5ab0b42fULL), LE64(0xe25b715b5bb9b62aULL),
   LE64(0xda5c6d5c5c86b831ULL), LE64(0xd25d695d5d8fba34ULL),
   LE64(0xca5e655e5e94bc3bULL), LE64(0xc25f615f5f9dbe3eULL),
   LE64(0x27609d606047c0fdULL), LE64(0x2f619961614ec2f8ULL),
   LE64(0x376295626255c4f7ULL), LE64(0x3f639163635cc6f2ULL),
   LE64(0x07648d646463c8e9ULL), LE64(0x0f658965656acaecULL),
   LE64(0x176685666671cce3ULL), LE64(0x1f6781676778cee6ULL),
   LE64(0x6768bd68680fd0d5ULL), LE64(0x6f69b9696906d2d0ULL),
   LE64(0x776ab56a6a1dd4dfULL), LE64(0x7f6bb16b6b14d6daULL),
   LE64(0x476cad6c6c2bd8c1ULL), LE64(0x4f6da96d6d22dac4ULL),
   LE64(0x576ea56e6e39dccbULL), LE64(0x5f6fa16f6f30deceULL),
```

```
LE64(0xa770dd7070d7e0adULL), LE64(0xaf71d97171dee2a8ULL),
LE64(0xb772d57272c5e4a7ULL), LE64(0xbf73d17373cce6a2ULL),
LE64(0x8774cd7474f3e8b9ULL), LE64(0x8f75c97575faeabcULL),
LE64(0x9776c57676e1ecb3ULL), LE64(0x9f77c17777e8eeb6ULL),
LE64(0xe778fd78789ff085ULL), LE64(0xef79f9797996f280ULL),
LE64(0xf77af57a7a8df48fULL), LE64(0xff7bf17b7b84f68aULL),
LE64(0xc77ced7c7cbbf891ULL), LE64(0xcf7de97d7db2fa94ULL),
LE64(0xd77ee57e7ea9fc9bULL), LE64(0xdf7fe17f7fa0fe9eULL),
LE64(0x74803a8080f41dbaULL), LE64(0x7c813e8181fd1fbfULL),
LE64(0x6482328282e619b0ULL), LE64(0x6c83368383ef1bb5ULL),
LE64(0x54842a8484d015aeULL), LE64(0x5c852e8585d917abULL),
LE64(0x4486228686c211a4ULL), LE64(0x4c87268787cb13a1ULL),
LE64(0x34881a8888bc0d92ULL), LE64(0x3c891e8989b50f97ULL),
LE64(0x248a128a8aae0998ULL), LE64(0x2c8b168b8ba70b9dULL),
LE64(0x148c0a8c8c980586ULL), LE64(0x1c8d0e8d8d910783ULL),
LE64(0x048e028e8e8a018cULL), LE64(0x0c8f068f8f830389ULL),
LE64(0xf4907a9090643deaULL), LE64(0xfc917e91916d3fefULL),
LE64(0xe4927292927639e0ULL), LE64(0xec937693937f3be5ULL),
LE64(0xd4946a94944035feULL), LE64(0xdc956e95954937fbULL),
LE64(0xc4966296965231f4ULL), LE64(0xcc976697975b33f1ULL),
LE64(0xb4985a98982c2dc2ULL), LE64(0xbc995e9999252fc7ULL),
LE64(0xa49a529a9a3e29c8ULL), LE64(0xac9b569b9b372bcdULL),
LE64(0x949c4a9c9c0825d6ULL), LE64(0x9c9d4e9d9d0127d3ULL),
LE64(0x849e429e9e1a21dcULL), LE64(0x8c9f469f9f1323d9ULL),
LE64(0x69a0baa0a0c95d1aULL), LE64(0x61a1bea1a1c05f1fULL),
LE64(0x79a2b2a2a2db5910ULL), LE64(0x71a3b6a3a3d25b15ULL),
LE64(0x49a4aaa4a4ed550eULL), LE64(0x41a5aea5a5e4570bULL),
LE64(0x59a6a2a6a6ff5104ULL), LE64(0x51a7a6a7a7f65301ULL),
LE64(0x29a89aa8a8814d32ULL), LE64(0x21a99ea9a9884f37ULL),
LE64(0x39aa92aaaa934938ULL), LE64(0x31ab96abab9a4b3dULL),
LE64(0x09ac8aacaca54526ULL), LE64(0x01ad8eadadac4723ULL),
LE64(0x19ae82aeaeb7412cULL), LE64(0x11af86afafbe4329ULL),
LE64(0xe9b0fab0b0597d4aULL), LE64(0xe1b1feb1b1507f4fULL),
LE64(0xf9b2f2b2b24b7940ULL), LE64(0xf1b3f6b3b3427b45ULL),
LE64(0xc9b4eab4b47d755eULL), LE64(0xc1b5eeb5b574775bULL),
LE64(0xd9b6e2b6b66f7154ULL), LE64(0xd1b7e6b7b7667351ULL),
LE64(0xa9b8dab8b8116d62ULL), LE64(0xa1b9deb9b9186f67ULL),
LE64(0xb9bad2baba036968ULL), LE64(0xb1bbd6bbbb0a6b6dULL),
LE64(0x89bccabcbc356576ULL), LE64(0x81bdcebdbd3c6773ULL),
LE64(0x99bec2bebe27617cULL), LE64(0x91bfc6bfbf2e6379ULL),
LE64(0x4ec027c0c08e9de7ULL), LE64(0x46c123c1c1879fe2ULL),
LE64(0x5ec22fc2c29c99edULL), LE64(0x56c32bc3c3959be8ULL),
LE64(0x6ec437c4c4aa95f3ULL), LE64(0x66c533c5c5a397f6ULL),
LE64(0x7ec63fc6c6b891f9ULL), LE64(0x76c73bc7c7b193fcULL),
LE64(0x0ec807c8c8c68dcfULL), LE64(0x06c903c9c9cf8fcaULL),
LE64(0x1eca0fcacad489c5ULL), LE64(0x16cb0bcbcbdd8bc0ULL),
LE64(0x2ecc17cccce285dbULL), LE64(0x26cd13cdcdeb87deULL),
LE64(0x3ece1fcecef081d1ULL), LE64(0x36cf1bcfcff983d4ULL),
LE64(0xced067d0d01ebdb7ULL), LE64(0xc6d163d1d117bfb2ULL),
LE64(0xded26fd2d20cb9bdULL), LE64(0xd6d36bd3d305bbb8ULL),
LE64(0xeed477d4d43ab5a3ULL), LE64(0xe6d573d5d533b7a6ULL),
LE64(0xfed67fd6d628b1a9ULL), LE64(0xf6d77bd7d721b3acULL),
LE64(0x8ed847d8d856ad9fULL), LE64(0x86d943d9d95faf9aULL),
LE64(0x9eda4fdada44a995ULL), LE64(0x96db4bdbdb4dab90ULL),
LE64(0xaedc57dcdc72a58bULL), LE64(0xa6dd53dddd7ba78eULL),
LE64(0xbede5fdede60a181ULL), LE64(0xb6df5bdfdf69a384ULL),
LE64(0x53e0a7e0e0b3dd47ULL), LE64(0x5be1a3e1e1badf42ULL),
```

```
    LE64(0x43e2afe2e2a1d94dULL), LE64(0x4be3abe3e3a8db48ULL),
    LE64(0x73e4b7e4e497d553ULL), LE64(0x7be5b3e5e59ed756ULL),
    LE64(0x63e6bfe6e685d159ULL), LE64(0x6be7bbe7e78cd35cULL),
    LE64(0x13e887e8e8fbcd6fULL), LE64(0x1be983e9e9f2cf6aULL),
    LE64(0x03ea8feaeae9c965ULL), LE64(0x0beb8bebebe0cb60ULL),
    LE64(0x33ec97ececdfc57bULL), LE64(0x3bed93ededd6c77eULL),
    LE64(0x23ee9feeeecdc171ULL), LE64(0x2bef9befefc4c374ULL),
    LE64(0xd3f0e7f0f023fd17ULL), LE64(0xdbf1e3f1f12aff12ULL),
    LE64(0xc3f2eff2f231f91dULL), LE64(0xcbf3ebf3f338fb18ULL),
    LE64(0xf3f4f7f4f407f503ULL), LE64(0xfbf5f3f5f50ef706ULL),
    LE64(0xe3f6fff6f615f109ULL), LE64(0xebf7fbf7f71cf30cULL),
    LE64(0x93f8c7f8f86bed3fULL), LE64(0x9bf9c3f9f962ef3aULL),
    LE64(0x83facffafa79e935ULL), LE64(0x8bfbcbfbfb70eb30ULL),
    LE64(0xb3fcd7fcfc4fe52bULL), LE64(0xbbfdd3fdfd46e72eULL),
    LE64(0xa3fedffefe5de121ULL), LE64(0xabffdbffff54e324ULL),
};
```

Приложение 2. Исходный код AES

AES.h

```
#ifndef AES_H_
#define AES_H_

#define BLOCK_NBYTES   16
#define AES_NB   4

typedef enum {
    //При размере ключа 128 бит длина ключа 16 байт
    BITS_128,

    //При размере ключа 192 бит длина ключа 24 байта
    BITS_192,

    //При размере ключа 256 бит длина ключа 32 байта
    BITS_256
} AES_key_len;

struct AES_context {
    // Секретный ключ
    uint32_t     expanded_key[8];

    // Размер ключа шифорвания
    AES_key_len  keysize;

    // Блок данных, размером 128 бит, 16 байт
    uint32_t     state[AES_NB];

    // Количество 32-битных слов, составляющих ключ шифрования.
    uint32_t     nk;

    // Количество раундов.
    uint32_t     nr;

    // Количество столбцов в матрице состояния.
    uint32_t     nb;

    // Открытый ключ
    uint32_t     w[AES_NB*(14+1)];

};


// Процедура очищает структуру AES
void AES_context_clean(struct AES_context *ctx);

// Инициализация структуры AES
void AES_context_init(struct AES_context *ctx);

// Процедура освобождает все ресурсы, связанные с контекстом
void AES_context_free(struct AES_context *ctx);

// Процедура создания ключа
void AES_set_key(struct AES_context *ctx,
                 uint8_t *key,
```

```
                        AES_key_len keyLength);
```

// инициализация ключа

```
struct AES_context *AES_context_new();
```

// Процедура шифрует 128-битный блок 'in'
//и подставляет его в 128-битный блок 'out'.

```
void AES_encrypt(struct AES_context *ctx,
                        const uint8_t *in,
                        uint8_t *out);
```

// Процедура дешифрует 128-битный блок 'in'
// и подставляет его в 128-битный блок 'out'.

```
void AES_decrypt(struct AES_context *ctx,
                        const uint8_t *in,
                        uint8_t *out);

#endif AES_H_
```

AES.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <malloc.h>
#include <string.h>

#include "../include/AES.h"
#include "../include/macros.h"

#if __BYTE_ORDER == __LITTLE_ENDIAN
#define CreateWord(b0, b1, b2, b3)
    ((b0 & 0xff) | ((b1 << 8) & 0xff00) |
    ((b2 << 16) & 0xff0000) | ((b3 << 24) & 0xff000000))
#elif __BYTE_ORDER == __BIG_ENDIAN
#define CreateWord(b0, b1, b2, b3)
    ((b3 & 0xff) | ((b2 << 8) & 0xff00) |
    ((b1 << 16) & 0xff0000) | ((b0 << 24) & 0xff000000))
#else
#error unsupported byte order
#endif



/*
Процедура используемая в процедуре расширения ключа.
На вход функции поступает 4-байтное слово. Выходное слово формируется
путём замены каждого из этих четырёх байт с помощью s-блока.
*/
static uint32_t sub_word(uint32_t word) {

    uint32_t temp;

    temp = (sbox[word & 0xff]);
    for (int i = 1 ; i < 4; i++)
      temp |= (sbox[(word >> 8*i) & 0xff] << 8*i);
```

```
    return temp;
}

/*------------------------------------------------------------------ */
/*
Процедура расширения ключа
*/
void key_expansion(struct AES_context *ctx) {

    uint32_t temp;
    int keysize = ctx->nb;
    int expanded_keysize =  4 *(ctx->nr + 1);
    int rcon_itteration = 0;
    int nk = ctx->nk;

    for( int i = ctx->nk; i < expanded_keysize; i++) {
        temp = ctx->w[i - 1];
         if (i % nk == 0)
         temp = sub_word(rot_word(temp))
                ^ rcon[rcon_itteration++];

     else if (ctx->nk > 6 && (i % nk) == AES_NB)
            temp = sub_word(temp);
     ctx->w[i] = ctx->w[i- ctx->nk] ^ temp;
    }


    int i, j, k, t;
    uint8_t *p;
    for (k = 0; k < 16; k++) {
     p = (uint8_t *) (ctx->w + AES_NB * k);
     for (i = 0; i < AES_NB; i++)
        for (j = i + 1; j < AES_NB; j++) {
         t = p[i * AES_NB + j];
         p[i * AES_NB + j] = p[j * AES_NB + i];
         p[j * AES_NB + i] = t;
        }
    }
}

/*
/*------------------------------------------------------------------ */
Процедура шифрования
*/
void AES_encrypt(struct AES_context *ctx,
                      const uint8_t *input,
                      uint8_t *output) {

    uint8_t *sp;

    sp = (uint8_t *) ctx->state;

    for (int i = 0; i < 4; i++) {
     for (int j = 0; j < 4; j++)
        sp[i * 4 + j] = input[j * 4 + i];
    }
```

```
    add_round_key(ctx, 0);

    for (int i = 1; i < ctx->nr; i++) {
      sub_bytes(ctx->state);
      shift_rows(ctx->state);
      mix_columns(ctx->state);
      add_round_key(ctx, i);
    }
    sub_bytes(ctx->state);
    shift_rows(ctx->state);
    add_round_key(ctx, ctx->nr);

    for (int i = 0; i < AES_NB; i++) {
        for (int j = 0; j < AES_NB; j++)
          output[i * AES_NB + j] = sp[j * AES_NB + i];
      }
}

/*---------------------------------------------------------------------- */
/*
Процедура расшифрования
*/
void AES_encrypt(struct AES_context *ctx,
                        const uint8_t *input,
                        uint8_t *output) {
    uint8_t *sp;

    sp = (uint8_t *) ctx->state;

    for (int i = 0; i < AES_NB; i++) {
      for (int j = 0; j < AES_NB; j++)
         sp[i * AES_NB + j] = input[j * AES_NB + i];
    }

    add_round_key(ctx,   ctx->nr);
    for (int i = ctx->nr - 1; i > 0; i--) {
      inv_shift_rows(ctx->state);
      inv_sub_bytes(ctx->state);
      add_round_key(ctx, i);
      inv_mix_columns(ctx->state);
    }
    inv_shift_rows(ctx->state);
    inv_sub_bytes(ctx->state);
    add_round_key(ctx, 0);

    for (int i = 0; i < AES_NB; i++) {
        for (int j = 0; j < AES_NB; j++)
          output[i * AES_NB + j] = sp[j * AES_NB + i];
      }
}

/*---------------------------------------------------------------------- */
/*
Процедура создания ключа
*/
void AES_set_key(struct AES_context *ctx,
                        uint8_t *key,
```

```
                        AES_key_len keyLength) {
    ctx->keysize = keyLength;
    switch (ctx->keysize) {
     case BITS_128:
        ctx->nk = 4;
        ctx->nr = 10;
        break;
     case BITS_192:
        ctx->nk = 6;
        ctx->nr = 12;
        break;
     case BITS_256:
        ctx->nk = 8;
        ctx->nr = 14;
        break;
    }
    ctx->nb = AES_NB;
    memcpy(ctx->expanded_key, key, ctx->nk * AES_NB);
    memcpy(ctx->w,ctx->expanded_key, ctx->nk * AES_NB);
    key_expansion(ctx);

}

/*-------------------------------------------------------------------- */
/*
Создание новой структуры шифрования AES
*/
struct AES_context *AES_context_new()
{
     struct AES_context *ctx;

     ctx = malloc(sizeof(*ctx));
     if (ctx == NULL)
         return NULL;
     memset(ctx, 0, sizeof(*ctx));
     ctx->nr = -1;
     ctx->nb = -1;
     ctx->nk = -1;

     return ctx;
}

/*-------------------------------------------------------------------- */
/*
Инициализация структуры шифрования AES
*/
void AES_context_init(struct AES_context *ctx) {
    ctx = malloc(sizeof (*ctx));
}

/*-------------------------------------------------------------------- */
/*Процедура освобождает все ресурсы, связанные с контекстом
*/
void AES_context_free(struct AES_context *ctx) {
    free(ctx);
}
```

```
/*---------------------------------------------------------------------- */
/*
```
Процедура очищает структуру AES
```
*/
void AES_context_clean(struct AES_context *ctx) {
    memset(ctx, 0, sizeof (*ctx));
}

//*---------------------------------------------------------------------- */
```

galois.h

```
#ifndef galois_H_
#define galois_H_

uint8_t transform_bits(uint8_t u);
uint8_t inv_transform_bits(uint8_t u);
```

/ Умножает каждый байт массива на 2 в поле Rijndael */*

```
void g2times(uint8_t *p, size_t n);

#endif /* galois_H_ */
```

galois.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#include "../include/galois.h"

#define RIJNDAEL_POLY    0x1B
#define RIJNDAEL_POLY2   0x6C
#define RIJNDAEL_POLY4   0xAB
#define RIJNDAEL_POLY8   0x9A
#define POLY_VECTOR       0x63
#define POLY_VECTOR_INV 0x05
#define BYTE_ROL(x, s)   (((x) << (s)) | ((x) >> (8 - s)))

/*---------------------------------------------------------------------- */
uint8_t
gmul(uint8_t a, uint8_t b)
{
      uint8_t c;
      int i;

      for (i = 0, c = 0; i < 8; i++) {
            if ((b & 1) != 0)
                  c ^= a;
            b >>= 1;
            if ((a & 0x80) != 0)
                  a = (a << 1) ^ RIJNDAEL_POLY;
            else
                  a <<= 1;
      }
```

```
	return c;
}

/*-------------------------------------------------------------------- */

// Возвращает a^2 в поле Rijndael
// Занимает примерно столько же времени, сколько половина умножения
static uint8_t
gsquare(uint8_t a)
{
	uint8_t p;

	p = (a & 0x01) ^ ((a & 0x02) << 1) ^
		((a & 0x04) << 2) ^ ((a & 0x08) << 3);
	if ((a & 0x10) != 0)
		p ^= RIJNDAEL_POLY;
	if ((a & 0x20) != 0)
		p ^= RIJNDAEL_POLY2;
	if ((a & 0x40) != 0)
		p ^= RIJNDAEL_POLY4;
	if ((a & 0x80) != 0)
		p ^= RIJNDAEL_POLY8;

	return p;
}

/*-------------------------------------------------------------------- */
/* Возвращает a^254 */
static uint8_t
gmul_inv(uint8_t a)
{
	uint8_t b;
	uint8_t c;
	int i;

	// a^254 = a^2 * a^4 * a^8 * a^16 * a^32 * a^64 * a^128 */

	b = gsquare(a);
	c = b;
	for (i = 0; i < 6; i++) {
		b = gsquare(b);
		c = gmul(b, c);
	}

	return c;
}

/*-------------------------------------------------------------------- */
uint8_t
transform_bits(uint8_t a)
{
	uint8_t b;

	b = gmul_inv(a);

	return b ^ BYTE_ROL(b, 1) ^ BYTE_ROL(b, 2) ^
		BYTE_ROL(b, 3) ^ BYTE_ROL(b, 4) ^
```

```
                POLY_VECTOR;
}

/*------------------------------------------------------------------- */
uint8_t
inv_transform_bits(uint8_t  u)
{
     return  gmul_inv(BYTE_ROL(u,  1)  ^  BYTE_ROL(u,  3)
               ^  BYTE_ROL(u,  6)  ^  POLY_VECTOR_INV);
}

/*------------------------------------------------------------------- */
/* Умножает каждый байт массива на 2 в поле Rijndael */
void
g2times(uint8_t  *p,  size_t  n)
{
     int  i;

     for  (i  =  0;  i  <  n;  i++)
          if  ((p[i]  &  0x80)  !=  0)
               p[i]  =  (p[i]  <<  1)  ^  RIJNDAEL_POLY;
          else
               p[i]  <<=  1;
}
```

AES_slow.h

```
#ifndef  AES_slow_H_
#define  AES_slow_H_

#include  "AES.h"

void  AES_encrypt_slow(struct  AES_context  *ctx,
                              const  uint8_t  *in,
                              uint8_t  *out);

void  AES_decrypt_slow(struct  AES_context  *ctx,
                              const  uint8_t  *in,
                              uint8_t  *out);

#endif  /*  AES_slow_H_  */
```

AES_slow.c

```
void
AES_encrypt_slow(struct  AES_context  *ctx_slow,
            const  uint8_t  *input,
            uint8_t  *output)
{
   uint8_t  *sp;

   sp  =  (uint8_t  *)  ctx_slow->state;

   for  (int  i  =  0;  i  <  4;  i++)  {
      for  (int  j  =  0;  j  <  4;  j++)
```

```
            sp[i * 4 + j] = input[j * 4 + i];
      }

       add_round_key_slow(ctx_slow, 0);

      for (int i = 1; i < ctx_slow->nr; i++) {
         sub_bytes_slow(ctx_slow->state);
         shift_rows_slow(ctx_slow->state);
         mix_columns_slow(ctx_slow->state);
         add_round_key_slow(ctx_slow, i);
      }
      sub_bytes_slow(ctx_slow->state);
      shift_rows_slow(ctx_slow->state);
      add_round_key_slow(ctx_slow, ctx_slow->nr);

      for (int i = 0; i < AES_NB; i++) {
            for (int j = 0; j < AES_NB; j++)
               output[i * AES_NB + j] = sp[j * AES_NB + i];
         }
}

/*-------------------------------------------------------------------- */
void
AES_decrypt_slow(struct AES_context *ctx_slow,
               const uint8_t *input,
               uint8_t *output)
{
   uint8_t *sp;

   sp = (uint8_t *) ctx_slow->state;

   for (int i = 0; i < AES_NB; i++)
      for (int j = 0; j < AES_NB; j++)
         sp[i * AES_NB + j] = input[j * AES_NB + i];

   add_round_key_slow(ctx_slow,   ctx_slow->nr);
   for (int i = ctx_slow->nr - 1; i > 0; i--) {
      inv_shift_rows_slow(ctx_slow->state);
      inv_sub_bytes_slow(ctx_slow->state);
      add_round_key_slow(ctx_slow, i);
      inv_mix_columns_slow(ctx_slow->state);
   }
   inv_shift_rows_slow(ctx_slow->state);
   inv_sub_bytes_slow(ctx_slow->state);
   add_round_key_slow(ctx_slow, 0);

   for (int i = 0; i < AES_NB; i++)
      for (int j = 0; j < AES_NB; j++)
         output[i * AES_NB + j] = sp[j * AES_NB + i];
}
```

AES_asmv.h

```
#ifndef AES_asmv_H_
#define AES_asmv_H_
```

```
#include "AES.h"

void AES_encrypt_asmv(struct AES_context *ctx,
                      const uint8_t *in,
                      uint8_t *out);
void AES_decrypt_asmv(struct AES_context *ctx,
                      const uint8_t *in,
                      uint8_t *out);

#endif /* AES_asmv_H_ */
```

AES_asmv.c

```
void
AES_encrypt_asmv(struct AES_context *ctx_asmv,
            const uint8_t *input,
            uint8_t *output)
{
   uint8_t *sp;

   sp = (uint8_t *) ctx_asmv->state;

   for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++)
         sp[i * 4 + j] = input[j * 4 + i];
   }

    add_round_key_asmv(ctx_asmv, 0);

   for (int i = 1; i < ctx_asmv->nr; i++) {
      sub_bytes_asmv(ctx_asmv->state);
      shift_rows_asmv(ctx_asmv->state);
      mix_columns_asmv(ctx_asmv->state);
      add_round_key_asmv(ctx_asmv, i);
   }
   sub_bytes_asmv(ctx_asmv->state);
   shift_rows_asmv(ctx_asmv->state);
   add_round_key_asmv(ctx_asmv, ctx_asmv->nr);

   for (int i = 0; i < AES_NB; i++) {
        for (int j = 0; j < AES_NB; j++)
           output[i * AES_NB + j] = sp[j * AES_NB + i];
      }
}
/*---------------------------------------------------------------------- */
void
AES_decrypt_asmv(struct AES_context *ctx_asmv,
            const uint8_t *input,
            uint8_t *output)
{
   uint8_t *sp;

   sp = (uint8_t *) ctx_asmv->state;

   for (int i = 0; i < AES_NB; i++)
      for (int j = 0; j < AES_NB; j++)
```

```
        sp[i * AES_NB + j] = input[j * AES_NB + i];

   add_round_key_asmv(ctx_asmv,   ctx_asmv->nr);
   for (int i = ctx_asmv->nr - 1; i > 0; i--) {
      inv_shift_rows_asmv(ctx_asmv->state);
      inv_sub_bytes_asmv(ctx_asmv->state);
      add_round_key_asmv(ctx_asmv, i);
      inv_mix_columns_asmv(ctx_asmv->state);
   }
   inv_shift_rows_asmv(ctx_asmv->state);
   inv_sub_bytes_asmv(ctx_asmv->state);
   add_round_key_asmv(ctx_asmv, 0);

   for (int i = 0; i < AES_NB; i++)
      for (int j = 0; j < AES_NB; j++)
         output[i * AES_NB + j] = sp[j * AES_NB + i];
}
```

AES_cbc.h

```
#ifndef AES_CBC_H_
#define AES_CBC_H_

#include "AES.h"


#define CBC_ENCRYPT    1
#define CBC_DECRYPT    0

struct AES_cbc {
    /*Вектор инициализации IV*/
    uint8_t iv[16];

    /*Частичный буффер входного потока*/
    uint8_t buffer[16];

    /*Длина частичного буффера потока*/
    unsigned int len;

    /*Режим шифрования/дешифрования */
    int mode;

    /* Процедура кодирования блочного шифра*/
    void (*encode)(struct AES_context *ctx,
    const uint8_t *in, uint8_t *out);

    /*Структура для блочного шифра*/
    void *ctx;
};
/*---------------------------------------------------------------------- */

/*Процедура инициализирует срутктуру AES_cbc */
void AES_cbc_init(struct AES_cbc *cbc,
            void (*encode)(struct AES_context *ctx,
                           const uint8_t *in,
                           uint8_t *out),
```

```
            void *ctx,
            int encr,
            unsigned char iv[16]);

/*---------------------------------------------------------------------- */
/*Процедура производит шифрование блоков в режиме cbc */
void AES_cbc_update(struct AES_cbc *cbc,
                uint8_t *out,
                          unsigned *out_len,
                          const uint8_t *in,
                          unsigned in_len);

/*---------------------------------------------------------------------- */
/*Последний раунд для  шифрования блоков  в режиме cbc */
void AES_cbc_final(struct AES_cbc *cbc,
                         uint8_t *out,
                         unsigned *out_len);

/*---------------------------------------------------------------------- */
/*Процедура очищяет структуру AES_cbc */
void AES_cbc_clean(struct AES_cbc *cbc);
```

aes_cbc.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#include "../include/aes_cbc.h"


#define CBC(out, iv)                    {

 /*---------------------------------------------------------------------- */
 /*Процедура очищяет структуру aes_cbc*/
void aes_cbc_clean(struct aes_cbc *cbc) {
    memset(cbc->iv, 0, sizeof(cbc->iv));
    memset(cbc->buffer, 0, sizeof(cbc->buffer));
}


 /*---------------------------------------------------------------------- */
 /*Процедура инициализирует сруктуру aes_cbc*/
void aes_cbc_init((struct AES_cbc *cbc,
            void (*encode)(struct AES_context *ctx,
                           const uint8_t *in,
                           uint8_t *out),
            void *ctx,
            int encr,
            unsigned char iv[16])
{
    memset(cbc->buffer , 0, sizeof(cbc->buffer));
    memcpy(cbc->iv , iv, sizeof(cbc->iv));
    cbc->len = 0;
    cbc->encode = encode;
```

```
    cbc->ctx = ctx;
    cbc-> mode = encr ? CBC_ENCRYPT : CBC_DECRYPT;
}


 /*------------------------------------------------------------------- */
 /*Процедура производит шифрование блоков в режиме cbc*/
void AES_cbc_update(struct AES_cbc *cbc,
                    uint8_t *out,
                              unsigned *out_len,
                              const uint8_t *in,
                              unsigned in_len)
{
    int n, space;

    n = cbc->len;
    space = BLOCK_NBYTES - n;
    *out_len = 0;

    if (n > 0 && space > 0) {
        memcpy(cbc->buffer + n, in, space);
        cbc->len += n;
        in += n;
        in_len -= n;
    }
    if (cbc->len >= BLOCK_NBYTES) {
        if (cbc->mode == CBC_ENCRYPT) {
            CBC(cbc->buffer, cbc->iv);
            (*cbc->encode)(cbc->ctx, cbc->buffer, out);
            memcpy(cbc->iv, out, BLOCK_NBYTES);
        }
        else {
            (*cbc->encode)(cbc->ctx, cbc->buffer, out);
            CBC(out, cbc->iv);
            memcpy(cbc->iv, cbc->buffer, BLOCK_NBYTES);
        }
        *out_len += BLOCK_NBYTES;
        out += BLOCK_NBYTES;
        cbc->len = 0;
    }

    while (in_len >= BLOCK_NBYTES) {

        memcpy(out, in, BLOCK_NBYTES);
        if (cbc->mode == CBC_ENCRYPT) {
             /*Сцепляем блоки для режима шифрования*/
            CBC(out, cbc->iv);

             /*Шифруем блок out*/
            (*cbc->encode)(cbc->ctx, out, out);

             /*Сохраняем текущий блок для построения следующего*/
            memcpy(cbc->iv, out, BLOCK_NBYTES);
        }
        else {
             /*Расшифровываем блок out*/
            (*cbc->encode)(cbc->ctx, out, out);
```

```
            /*Сцепляем блоки для режима дешифрования*/
            CBC(out, cbc->iv);

            /*Сохраняем текущий блок для построения следующего*/
            memcpy(cbc->iv, in, BLOCK_NBYTES);
        }
        *out_len += BLOCK_NBYTES;
        out += BLOCK_NBYTES;
        in += BLOCK_NBYTES;
        in_len -= BLOCK_NBYTES;

    }

    if (in_len > 0) {
        n = cbc->len;
        memcpy(cbc->buffer + n, in, in_len);
        cbc->len += in_len;
    }
}


 /*------------------------------------------------------------------------ */
 /*Последний раунд для  шифрования блоков в режиме cbc*/
void aes_cbc_final(struct aes_cbc *cbc,
                uint8_t *out,
                unsigned *out_len)
{
    int n, space;

    n = cbc->len;
    space = BLOCK_NBYTES - n;
    *out_len = 0;

    if (n > 0) {
        memset(cbc->buffer + n, space, space);

     /*Кодируем последний блок*/
        if (cbc->mode == CBC_ENCRYPT) {
            CBC(cbc->buffer, cbc->iv);
            (*cbc->encode)(cbc->ctx, cbc->buffer, out);
            memcpy(cbc->iv, out, BLOCK_NBYTES);
        }
        else {
              (*cbc->encode)(cbc->ctx, cbc->buffer, out);
            CBC(out, cbc->iv);
            memcpy(cbc->iv, cbc->buffer, BLOCK_NBYTES);
        }
        *out_len = n;
    }
}
```

main.c

```
#include <stdio.h>
#include <stdint.h>
#include <time.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include "../include/openssl/rand.h"
#include "../include/openssl/aes.h"
#include "../include/openssl/evp.h"
#include "../include/aes.h"
#include "../include/aes_slow.h"
#include "../include/aes_asmv.h"
#include "../include/aes_cbc.h"

static void
usage(const char *name)
{
   fprintf(stderr,
         "Usage: %s [OPTION...]"
         "   -f FILENAME   Use file for tests"
         "         Can't be used together with -s option"
         "   -s STR    Use string STR for tests"
         "         Can't be used together with -f option"
         "   -h          Show this usage and exit",
         name);

   exit(EXIT_FAILURE);
}

static void
parse_opts(int argc,
          char **argv,
          char **pfilename,
          char **pstr,
          int *pquiet)
{
   if (argc < 2)
      usage(argv[0]);

   int opt;

   *pfilename = NULL;
   *pstr = NULL;
   *pquiet = 0;

   while ((opt = getopt(argc, argv, "f:s:qh")) != -1)
      switch (opt) {
      case 'f':
         *pfilename = optarg;
         break;
      case 's':
         *pstr = optarg;
         break;
      case 'q':
         *pquiet = 1;
         break;
      case 'h':
         usage(argv[0]);
         break;
      default:
```

```
            usage(argv[0]);
            break;
         }

      if ((*pfilename != NULL && *pstr != NULL) ||
            (*pfilename == NULL && *pstr == NULL))
         usage(argv[0]);
}

static size_t
loadfile(const char *filename, char **out)
{
   int fd;
   fd = open(filename, O_RDONLY);
   if (fd < 0) {
      fprintf(stderr,
            "Error during opening file %s", filename);
      exit(EXIT_FAILURE);
   }

   off_t off;
   off = lseek(fd, (off_t) 0, SEEK_END);
   if (off < 0) {
      fprintf(stderr,
            "Error during processing file %s", filename);
      exit(EXIT_FAILURE);
   }

   size_t filesize;
   filesize = (size_t) off;

   *out = malloc((size_t) filesize);
   if (*out == NULL) {
      fprintf(stderr,
      "Can't allocate %u bytes in memory", filesize);
      exit(EXIT_FAILURE);
   }

   off = lseek(fd, (off_t) 0, SEEK_SET);
   if (off < 0) {
      fprintf(stderr,
            "Error during processing file %s", filename);
      exit(EXIT_FAILURE);
   }

   ssize_t rb;
   rb = read(fd, *out, filesize);
   if (rb < 0) {
      fprintf(stderr,
            "Error during loading file %s", filename);
      exit(EXIT_FAILURE);
   }

   return (size_t) rb;
}

static void
```

```
printhex(uint8_t *a, size_t size)
{
   int i;

   for (i = 0; i < size; i++)
      printf("%02X", a[i]);
}

static double
test_aes_cbc_ossl(const unsigned char *plain, size_t len,
      const unsigned char *key, size_t key_len,
      unsigned char *iv_enc, unsigned char *iv_dec,
      unsigned char *enc, size_t enc_len,
      unsigned char *dec)
{
   clock_t c;
   EVP_CIPHER_CTX ctx;
   const EVP_CIPHER *cipher;
   int l, m;

   switch (key_len) {
      case 128:
         cipher = EVP_aes_128_cbc();
         break;
      case 192:
         cipher = EVP_aes_192_cbc();
         break;
      case 256:
         cipher = EVP_aes_256_cbc();
         break;
      default:
         fprintf(stderr,
               "Invalid key len %d", key_len);
         exit(EXIT_FAILURE);
         break;
   }

   c = clock();

   EVP_CipherInit(&ctx, cipher, key, iv_enc, 1);
   EVP_CipherUpdate(&ctx, enc, &l, plain, len);
   EVP_CipherFinal(&ctx, enc + l, &m);
   EVP_CipherInit(&ctx, cipher, key, iv_dec, 0);
   EVP_CipherUpdate(&ctx, dec, &l, enc, enc_len);
   EVP_CipherFinal(&ctx, dec + l, &m);

   return ((double) clock() - c) / CLOCKS_PER_SEC;
}

static double
test_aes_cbc_ref(const unsigned char *plain, size_t len,
      const unsigned char *key, size_t key_len,
      unsigned char *iv_enc, unsigned char *iv_dec,
      unsigned char *enc, size_t enc_len,
      unsigned char *dec)
{
   clock_t c;
```

```
   struct aes_context ctx;
   struct aes_cbc cbc;
   aes_key_len key_l;
   size_t l, m;

   switch (key_len) {
      case 128:
         key_l = BITS_128;
         break;
      case 192:
         key_l = BITS_192;
         break;
      case 256:
         key_l = BITS_256;
         break;
      default:
         fprintf(stderr,
               "Invalid key len %d", key_len);
         exit(EXIT_FAILURE);
         break;
   }

   c = clock();

   aes_context_init(&ctx);
   aes_set_key(&ctx, key, key_l);
   aes_cbc_init(&cbc, aes_encrypt, &ctx,
                     CBC_ENCRYPT, iv_enc);
   aes_cbc_update(&cbc, enc, &l, plain, len);
   aes_cbc_final(&cbc, enc + l, &m);
   aes_cbc_clean(&cbc);
   aes_context_clean(&ctx);

   aes_context_init(&ctx);
   aes_set_key(&ctx, key, key_l);
   aes_cbc_init(&cbc, aes_decrypt, &ctx,
                     CBC_DECRYPT, iv_dec);
   aes_cbc_update(&cbc, dec, &l, enc, enc_len);
   aes_cbc_final(&cbc, dec + l, &m);
   aes_cbc_clean(&cbc);
   aes_context_clean(&ctx);

   return ((double) clock() - c) / CLOCKS_PER_SEC;
}
static double
test_aes_cbc_slow(const unsigned char *plain, size_t len,
      const unsigned char *key, size_t key_len,
      unsigned char *iv_enc, unsigned char *iv_dec,
      unsigned char *enc, size_t enc_len,
      unsigned char *dec)
{
   clock_t c;
   struct aes_context ctx;
   struct aes_cbc cbc;
   aes_key_len key_l;
   size_t l, m;
```

```
   switch (key_len) {
      case 128:
         key_l = BITS_128;
         break;
      case 192:
         key_l = BITS_192;
         break;
      case 256:
         key_l = BITS_256;
         break;
      default:
         fprintf(stderr,
               "Invalid key len %d", key_len);
         exit(EXIT_FAILURE);
         break;
   }

   c = clock();

   aes_context_init(&ctx);
   aes_set_key(&ctx, key, key_l);
   aes_cbc_init(&cbc, aes_encrypt_slow, &ctx,
                    CBC_ENCRYPT, iv_enc);
   aes_cbc_update(&cbc, enc, &l, plain, len);
   aes_cbc_final(&cbc, enc + l, &m);
   aes_cbc_clean(&cbc);
   aes_context_clean(&ctx);

   aes_context_init(&ctx);
   aes_set_key(&ctx, key, key_l);
   aes_cbc_init(&cbc, aes_decrypt_slow, &ctx,
                    CBC_DECRYPT, iv_dec);
   aes_cbc_update(&cbc, dec, &l, enc, enc_len);
   aes_cbc_final(&cbc, dec + l, &m);
   aes_cbc_clean(&cbc);
   aes_context_clean(&ctx);

   return ((double) clock() - c) / CLOCKS_PER_SEC;
}

static double
test_aes_cbc_asmv(const unsigned char *plain, size_t len,
      const unsigned char *key, size_t key_len,
      unsigned char *iv_enc, unsigned char *iv_dec,
      unsigned char *enc, size_t enc_len,
      unsigned char *dec)
{
   clock_t c;
   struct aes_context ctx;
   struct aes_cbc cbc;
   aes_key_len key_l;
   size_t l, m;

   switch (key_len) {
      case 128:
         key_l = BITS_128;
         break;
```

```
      case 192:
         key_l = BITS_192;
         break;
      case 256:
         key_l = BITS_256;
         break;
      default:
         fprintf(stderr,
               "Invalid key len %d", key_len);
         exit(EXIT_FAILURE);
         break;
   }

   c = clock();

   aes_context_init(&ctx);
   aes_set_key(&ctx, key, key_l);
   aes_cbc_init(&cbc, aes_encrypt_asmv, &ctx,
                     CBC_ENCRYPT, iv_enc);
   aes_cbc_update(&cbc, enc, &l, plain, len);
   aes_cbc_final(&cbc, enc + l, &m);
   aes_cbc_clean(&cbc);
   aes_context_clean(&ctx);

   aes_context_init(&ctx);
   aes_set_key(&ctx, key, key_l);
   aes_cbc_init(&cbc, aes_decrypt_asmv, &ctx,
                     CBC_DECRYPT, iv_dec);
   aes_cbc_update(&cbc, dec, &l, enc, enc_len);
   aes_cbc_final(&cbc, dec + l, &m);
   aes_cbc_clean(&cbc);
   aes_context_clean(&ctx);

   return ((double) clock() - c) / CLOCKS_PER_SEC;
}

static void
test_aes_cbc(char *str, size_t size)
{
   double t;
   size_t key_len, enc_len;
   unsigned char *key, *iv, *iv_enc,
                     *iv_dec, *enc_ossl,
                     *enc, *dec;

   for (key_len = 128; key_len < 320; key_len += 64) {
      printf("0esting AES-CBC-%d", key_len);

      key = malloc(sizeof(*key) * key_len / 8);
      iv = malloc(sizeof(*iv) * AES_BLOCK_SIZE);
      iv_enc = malloc(sizeof(*iv_enc) * AES_BLOCK_SIZE);
      iv_dec = malloc(sizeof(*iv_dec) * AES_BLOCK_SIZE);
      enc_len = ((size + AES_BLOCK_SIZE) / AES_BLOCK_SIZE) *
           AES_BLOCK_SIZE;
      enc_ossl = calloc(sizeof(*enc_ossl) * enc_len, 1);
      enc = calloc(sizeof(*enc) * enc_len, 1);
      dec = calloc(sizeof(*dec) * enc_len, 1);
```

```
if (key == NULL || iv == NULL || iv_enc == NULL ||
      iv_dec == NULL || enc_ossl == NULL ||
   enc == NULL || dec == NULL) {
   fprintf(stderr,
   "Can't allocate enough memory for testing");
   exit(EXIT_FAILURE);
}

if (RAND_bytes(key, key_len / 8) != 1) {
   fprintf(stderr,
        "Can't initialize key with random bytes");
   exit(EXIT_FAILURE);
}

printf("Using random key: ");
printhex(key, key_len / 8);

if (RAND_bytes(iv, AES_BLOCK_SIZE) != 1) {
   fprintf(stderr,
        "Can't initialize IV with random bytes");
   exit(EXIT_FAILURE);
}

printf("0sing random IV:   ");
printhex(iv, AES_BLOCK_SIZE);

printf("0SSL: ");

memcpy(iv_enc, iv, AES_BLOCK_SIZE);
memcpy(iv_dec, iv, AES_BLOCK_SIZE);

t = test_aes_cbc_ossl((const unsigned char *) str,
     size, (const unsigned char *) key, key_len,
     iv_enc, iv_dec, enc_ossl, enc_len, dec);

if (memcmp(str, dec, size) == 0)
   printf("plain and decoded messages are the same, ");
else
   printf("plain and decoded messages are not the same,

printf("%f", t);

printf("REF:   ");

memcpy(iv_enc, iv, AES_BLOCK_SIZE);
memcpy(iv_dec, iv, AES_BLOCK_SIZE);

t = test_aes_cbc_ref((const unsigned char *) str,
     size, (const unsigned char *) key, key_len,
     iv_enc, iv_dec, enc_ossl, enc_len, dec);

if (memcmp(str, dec, size) == 0) {
   printf("plain and decoded messages are the same, ");
   if (memcmp(enc, enc_ossl, enc_len) == 0) {
      printf("encoded message is the same as of "
           "OSSL's one, ");
   } else {
```

```
            printf("but encoded message is not the same "
                 "as of OSSL's one");
            printf("    ENC REF [16] = ");
            printhex(enc, 16);
            printf("ENC OSSL[16] = ");
            printhex(enc_ossl, 16);
            printf("");
         }

      } else {
      printf("plain and decoded messages are not the same, ");
      }
      printf("%f", t);
      printf("0LOW: ");

      memcpy(iv_enc, iv, AES_BLOCK_SIZE);
      memcpy(iv_dec, iv, AES_BLOCK_SIZE);

      t = test_aes_cbc_slow((const unsigned char *) str,
            size, (const unsigned char *) key, key_len,
            iv_enc, iv_dec, enc_ossl, enc_len, dec);

      if (memcmp(str, dec, size) == 0) {
         printf("plain and decoded messages are the same, ");
         if (memcmp(enc, enc_ossl, enc_len) == 0) {
            printf("encoded message is the same as of "
                 "OSSL's one, ");
         } else {
            printf("but encoded message is not the same "
                 "as of OSSL's one");
            printf("    ENC SLOW[16] = ");
            printhex(enc, 16);
            printf("ENC SLOW[16] = ");
            printhex(enc_ossl, 16);
            printf("");
         }

      } else {
    printf("plain and decoded messages are not the same, ");
      }

         printf("%f", t);
      printf("0SMV: ");

      memcpy(iv_enc, iv, AES_BLOCK_SIZE);
      memcpy(iv_dec, iv, AES_BLOCK_SIZE);

      t = test_aes_cbc_asmv((const unsigned char *) str,
            size, (const unsigned char *) key, key_len,
            iv_enc, iv_dec, enc_ossl, enc_len, dec);

      if (memcmp(str, dec, size) == 0) {
         printf("plain and decoded messages are the same, ");
         if (memcmp(enc, enc_ossl, enc_len) == 0) {
            printf("encoded message is the same as of "
                 "OSSL's one, ");
         } else {
```

```
            printf("but encoded message is not the same "
                 "as of OSSL's one");
            printf("    ENC ASMV[16] = ");
            printhex(enc, 16);
            printf("ENC OSSL[16] = ");
            printhex(enc_ossl, 16);
            printf("");
         }

      } else {
     printf("plain and decoded messages are not the same, ");
      }

      printf("%f", t);

      free(key);
      free(iv);
      free(enc_ossl);
      free(dec);
   }
}

int main(int argc, char **argv)
{
   char *filename;
   char *str;
   int quiet;
   size_t size;

   parse_opts(argc, argv, &filename, &str, &quiet);

   if (filename != NULL) {
      size = loadfile(filename, &str);
      if (!quiet)
         printf("Using filename %s of %u bytes size",
               filename, size);
   } else {
      size = strlen(str);
      if (!quiet)
         printf("Using string
               str, size);
   }

   test_aes_cbc(str, size);

   return EXIT_SUCCESS;
}
```

Приложение 3. Исходный код WHIRLPOOL

whirlpool_ref.h

```
#ifndef whirlpool_H_
#define whirlpool_H_

#include <stdint.h>



#define LE32(x)  (x)
#define LE64(x)  (x)
#define R64(x)   ((x >> 32) | (x << 32))
#define BLOCK_NBYTES   64
#define WHIRLPOOL_NB   8


struct context_ref{
      uint32_t length[2];
      uint8_t  buffer[64];
      uint64_t state[8];
};

/*-----------------------------------------------------*/
/*Init structure*/
void whirlpool_init_ref(struct context_ref *ctx);


/*-----------------------------------------------------*/
/*add element to whirlpool structure*/
void whirlpool_update_ref(struct context_ref *ctx,
                                  const void *msg,
                                  uint32_t msglen)   ;


/*-----------------------------------------------------*/
/*final round*/
void whirlpool_final_ref(struct context_ref *pointer,
                                 unsigned char *result);

/*-----------------------------------------------------*/
#endif /* whirlpool_H_ */
```

whirlpool_ref.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <endian.h>
#include <time.h>

#include "../include/whirlpool_ref.h"
```

```
 /*--------------------------------------------------------------------- */
 /* Процедура блочного шифрования W */
static
void whirlpool_hash_ref(uint64_t s[8],
                                  const unsigned char buffer[64

   int i;
   uint64_t state[8];
   uint64_t key[8];

   memcpy(key, s, BLOCK_NBYTES);
   memcpy(state, buffer, BLOCK_NBYTES);
   add_round_key(state, key);

   for (i = 0; i < 10; i++) {

      sub_bytes(key);
      sub_bytes(state);

      shift_columns(key);
      shift_columns(state);

      mix_rows(key);
      mix_rows(state);

      key[0] ^= cr[i];
      add_round_key(state, key);

   }

   for (i = 0; i < WHIRLPOOL_NB; i++)
      s[i] ^= state[i] ^ ((uint64_t *)buffer)[i];

}


 /*--------------------------------------------------------------------- */
 /* Инициализация структуры context */
void whirlpool_init_ref(struct context_ref *ctx) {

   int i;
   ctx->length[0] = 0;
   ctx->length[1] = 0;

   for (i = 0; i < WHIRLPOOL_NB; i++) {
      ((uint64_t *)ctx->buffer)[i] = 0;
      ctx->state[i] = 0;
   }

}


 /*--------------------------------------------------------------------- */
void whirlpool_update_ref(struct context_ref *ctx,
             const void *msg,
             uint32_t msglen) {
   unsigned int n, len;
```

```
    n = ctx->length[0] & 0x3F;
    ctx->length[0] += msglen;
    if (n + msglen < 64) {
        /* just copy the message to the buffer */
        memcpy(ctx->buffer + n, msg, msglen);
    } else {
        /* copy and hash a part of message */
        len = 64 - n;
        memcpy(ctx->buffer + n, msg, len);
        whirlpool_hash_ref(ctx->state, ctx->buffer);
        msglen -= len;
        msg += len;
        /* copy and hash 64-byte blocks */
        while (msglen >= 64) {
            memcpy(ctx->buffer, msg, 64);
            whirlpool_hash_ref(ctx->state, ctx->buffer);
            msglen -= len;
            msg += len;
        }
        /* copy remainder of the message */
        memcpy(ctx->buffer, msg, msglen);
    }
}


 /*---------------------------------------------------------------------- */
static void uint32_to_bytes(unsigned char *out,
                                        const uint32_t *in) {

    int i, j;

    for (i = j = 0; j < 3; j++) {
        out[i++] = (in[j] >> 24) & 0xff;
        out[i++] = (in[j] >> 16) & 0xff;
        out[i++] = (in[j] >> 8) & 0xff;
        out[i++] = in[j] & 0xff;
    }
}


 /*---------------------------------------------------------------------- */
void whirlpool_final_ref(struct context_ref *ctx,
            unsigned char digest[64]) {

    static const unsigned char pad[64] = { 0x80, 0x0 };
    unsigned int n, npad;
    uint32_t nbits[3];
    uint8_t nb[32];

    n = ctx->length[0] & 0x3f;
    npad = ((n < 32) ? 32: 96) - n;

    nbits[0] = nbits[1] = 0;
    nbits[1] += ctx->length[0] >> 29;
    nbits[2] = ctx->length[0] << 3;

    memset(nb, 0, sizeof(nb));
```

```
   uint32_to_bytes(nb+20, nbits);

   whirlpool_update_ref(ctx, pad, npad);
   whirlpool_update_ref(ctx, nb, 32);

   memcpy(digest, ctx->state, 64);
}

/*---------------------------------------------------------------------- */
```

galois.h

```
#ifndef galois_H_
#define galois_H_

void mix_columns_slow(uint32_t *state[16]);
void sub_bytes_slow(uint64_t state[8]);

#endif
```

galois.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#include "../include/whirlpool_slow.h"
#define WHIRLPOOL_POLY 0x1d
#define WHIRLPOOL_GEN_POLY  0x02

static
void galois_init_tables() {

    uint8_t c;
    int i;
    c = 1;

    for (i = 0; i < 256; i++) {
        logtab[c] = i;
        exptab[i] = c;
        c = gmul_slow(c, WHIRLPOOL_GEN_POLY);
      }
}

static
unsigned char transform_bits(uint8_t u){

    unsigned char x, y, r;

    x = ebox[u >> 4];
    y = iebox[u & 0x0f];
    r = rbox[x ^ y];

    return (ebox[x ^ r] << 4) | iebox[y ^ r];
```

```
}

static
void create_sbox() {

    int u;

    for(u = 0; u < 256; u++)   {
        sbox[u] = transform_bits(u);

        printf("%x ", sbox[u]);
        if(u % 16 == 0) printf("");

        }
}
```

whirlpool_slow.h

```
#ifndef whirlpool_H_
#define whirlpool_H_

#include <stdint.h>



#define LE32(x)  (x)
#define LE64(x)  (x)
#define R64(x)   ((x >> 32) | (x << 32))
#define BLOCK_NBYTES   64
#define WHIRLPOOL_NB   8


struct context_slow{
      uint32_t length[2];
      uint8_t   buffer[64];
      uint64_t state[8];
};

/*-----------------------------------------------------*/
/*Init structure*/
void whirlpool_init_slow(struct context_slow *ctx);


/*-----------------------------------------------------*/
/*add element to whirlpool structure*/
void whirlpool_update_slow(struct context_slow *ctx,
                                  const void *msg,
                                  uint32_t msglen)   ;


/*-----------------------------------------------------*/
/*final round*/
void whirlpool_final_slow(struct context_slow *pointer,
                                 unsigned char *result);

/*-----------------------------------------------------*/
```

```
#endif /* whirlpool_H_ */
```

whirlpool_slow.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <endian.h>
#include <time.h>

#include "../include/galois.h"
#include "../include/whirlpool_slow.h"

 /*---------------------------------------------------------------------- */
 /* Процедура блочного шифрования W */
static
void whirlpool_hash_slow(uint64_t s[8],
                                unsigned char buffer[64]) {

   int i;
   uint64_t state[8];
   uint64_t key[8];

   memcpy(key, s, BLOCK_NBYTES);
   memcpy(state, buffer, BLOCK_NBYTES);
   add_round_key(state, key);

   for (i = 0; i < 10; i++) {

      sub_bytes(key);
      sub_bytes(state);

      shift_columns(key);
      shift_columns(state);

      mix_rows(key);
      mix_rows(state);

      key[0] ^= cr[i];
      add_round_key(state, key);

   }

   for (i = 0; i < WHIRLPOOL_NB; i++)
      s[i] ^= state[i] ^ ((uint64_t *)buffer)[i];

}


 /*---------------------------------------------------------------------- */
 /* Инициализация структуры context */
void whirlpool_init_slow(struct context_slow *ctx) {

   int i;
```

```
   ctx->length[0] = 0;
   ctx->length[1] = 0;

   for (i = 0; i < WHIRLPOOL_NB; i++) {
      ((uint64_t *)ctx->buffer)[i] = 0;
      ctx->state[i] = 0;
   }

}


 /*-------------------------------------------------------------------- */
void whirlpool_update_slow(struct context_slow *ctx,
             const void *msg,
             uint32_t msglen) {
   unsigned int n, len;

   n = ctx->length[0] & 0x3F;
   ctx->length[0] += msglen;
   if (n + msglen < 64) {
       /* just copy the message to the buffer */
      memcpy(ctx->buffer + n, msg, msglen);
   } else {
       /* copy and hash a part of message */
      len = 64 - n;
      memcpy(ctx->buffer + n, msg, len);
      whirlpool_hash_slow(ctx->state, ctx->buffer);
      msglen -= len;
      msg += len;
       /* copy and hash 64-byte blocks */
      while (msglen >= 64) {
         memcpy(ctx->buffer, msg, 64);
         whirlpool_hash_slow(ctx->state, ctx->buffer);
         msglen -= len;
         msg += len;
      }
       /* copy remainder of the message */
      memcpy(ctx->buffer, msg, msglen);
   }
}


 /*-------------------------------------------------------------------- */
static void uint32_to_bytes(unsigned char *out,
                                       const uint32_t *in) {

   int i, j;

   for (i = j = 0; j < 3; j++) {
      out[i++] = (in[j] >> 24) & 0xff;
      out[i++] = (in[j] >> 16) & 0xff;
      out[i++] = (in[j] >> 8) & 0xff;
      out[i++] = in[j] & 0xff;
   }
}
```

```
 /*-------------------------------------------------------------- */
void whirlpool_final_slow(struct context_slow *ctx,
            unsigned char digest[64]) {

   static const unsigned char pad[64] = { 0x80, 0x0 };
   unsigned int n, npad;
   uint32_t nbits[3];
   uint8_t nb[32];

   n = ctx->length[0] & 0x3f;
   npad = ((n < 32) ? 32: 96) - n;

   nbits[0] = nbits[1] = 0;
   nbits[1] += ctx->length[0] >> 29;
   nbits[2] = ctx->length[0] << 3;

   memset(nb, 0, sizeof(nb));
   uint32_to_bytes(nb+20, nbits);

   whirlpool_update_slow(ctx, pad, npad);
   whirlpool_update_slow(ctx, nb, 32);

   memcpy(digest, ctx->state, 64);
}


 /*-------------------------------------------------------------- */
```

whirlpool_asmv.h

```
#ifndef whirlpool_H_
#define whirlpool_H_

#include <stdint.h>



#define LE32(x)  (x)
#define LE64(x)  (x)
#define R64(x)   ((x >> 32) | (x << 32))
#define BLOCK_NBYTES   64
#define WHIRLPOOL_NB   8


struct context_asmv{
     uint32_t length[2];
     uint8_t  buffer[64];
     uint64_t state[8];
};

/*-------------------------------------------------*/
/*Init structure*/
void whirlpool_init_asmv(struct context_asmv *ctx);


/*-------------------------------------------------*/
```

```
/*add element to whirlpool structure*/
void whirlpool_update_asmv(struct context_asmv *ctx,
                           const void *msg,
                           uint32_t msglen)   ;


/*-----------------------------------------------------*/
/*final round*/
void whirlpool_final_asmv(struct context_asmv *pointer,
                          unsigned char *result);

/*-----------------------------------------------------*/
#endif /* whirlpool_H_ */
```

whirlpool_asmv.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <endian.h>
#include <time.h>

#include "../include/galois.h"

 /*-------------------------------------------------------------------- */
 /* Процедура блочного шифрования W */
static
void whirlpool_hash_asmv(uint64_t s[8],
                         unsigned char buffer[64]) {

   int i;
   uint64_t state[8];
   uint64_t key[8];

   memcpy(key, s, BLOCK_NBYTES);
   memcpy(state, buffer, BLOCK_NBYTES);
   add_round_key(state, key);

   for (i = 0; i < 10; i++) {

      sub_bytes(key);
      sub_bytes(state);

      shift_columns(key);
      shift_columns(state);

      mix_rows(key);
      mix_rows(state);

      key[0] ^= cr[i];
      add_round_key(state, key);

   }
```

```
   for (i = 0; i < WHIRLPOOL_NB; i++)
      s[i] ^= state[i] ^ ((uint64_t *)buffer)[i];

}


 /*--------------------------------------------------------------------- */
 /* Инициализация структуры context */
void whirlpool_init_asmv(struct context_asmv *ctx) {

   int i;
   ctx->length[0] = 0;
   ctx->length[1] = 0;
   for (i = 0; i < WHIRLPOOL_NB; i++) {
      ((uint64_t *)ctx->buffer)[i] = 0;
      ctx->state[i] = 0;
   }

}


 /*--------------------------------------------------------------------- */
void whirlpool_update_asmv(struct context_asmv *ctx,
             const void *msg,
             uint32_t msglen) {
   unsigned int n, len;
   n = ctx->length[0] & 0x3F;
   ctx->length[0] += msglen;
   if (n + msglen < 64) {
       /* just copy the message to the buffer */
      memcpy(ctx->buffer + n, msg, msglen);
   } else {
       /* copy and hash a part of message */
      len = 64 - n;
      memcpy(ctx->buffer + n, msg, len);
      whirlpool_hash_asmv(ctx->state, ctx->buffer);
      msglen -= len;
      msg += len;
       /* copy and hash 64-byte blocks */
      while (msglen >= 64) {
         memcpy(ctx->buffer, msg, 64);
         whirlpool_hash_asmv(ctx->state, ctx->buffer);
         msglen -= len;
         msg += len;
      }
       /* copy remainder of the message */
      memcpy(ctx->buffer, msg, msglen);
   }
}


 /*--------------------------------------------------------------------- */
static void uint32_to_bytes(unsigned char *out,
                                    const uint32_t *in) {
```

```
   int i, j;

   for (i = j = 0; j < 3; j++) {
      out[i++] = (in[j] >> 24) & 0xff;
      out[i++] = (in[j] >> 16) & 0xff;
      out[i++] = (in[j] >> 8) & 0xff;
      out[i++] = in[j] & 0xff;
   }
}


 /*---------------------------------------------------------------------- */
void whirlpool_final_asmv(struct context_asmv *ctx,
            unsigned char digest[64]) {

   static const unsigned char pad[64] = { 0x80, 0x0 };
   unsigned int n, npad;
   uint32_t nbits[3];
   uint8_t nb[32];

   n = ctx->length[0] & 0x3f;
   npad = ((n < 32) ? 32: 96) - n;

   nbits[0] = nbits[1] = 0;
   nbits[1] += ctx->length[0] >> 29;
   nbits[2] = ctx->length[0] << 3;

   memset(nb, 0, sizeof(nb));
   uint32_to_bytes(nb+20, nbits);

   whirlpool_update_asmv(ctx, pad, npad);
   whirlpool_update_asmv(ctx, nb, 32);

   memcpy(digest, ctx->state, 64);
}


 /*---------------------------------------------------------------------- */
```

main.c

```
#include <stdio.h>
#include <stdint.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>

#include "../include/nessie.h"
#include "../include/openssl/evp.h"
#include "../include/whirlpool_ref.h"
#include "../include/whirlpool_slow.h"
#include "../include/whirlpool_asmv.h"
#include "../include/whirlpool_asmu.h"

static void
```

```
usage(const char *name)
{
   fprintf(stderr,
             "Usage: %s [OPTION...]"
"   -q               Show only hashes for REFO, REF, "
"                     SLOW and ASMV implementations"
"   -f FILENAME   Use file for tests"
"                     Can't be used together with -s option"
"   -s STR          Use string STR for tests"
"                     Can't be used together with -f option"
"   -h                Show this usage and exit",
     name);

   exit(EXIT_FAILURE);
}

static void
parse_opts(int argc, char **argv, char **pfilename,
                char **pstr, int *pquiet)
{
   if (argc < 2)
      usage(argv[0]);

   int opt;

   *pfilename = NULL;
   *pstr = NULL;
   *pquiet = 0;

   while ((opt = getopt(argc, argv, "f:s:qh")) != -1)
      switch (opt) {
      case 'f':
         *pfilename = optarg;
         break;
      case 's':
         *pstr = optarg;
         break;
      case 'q':
         *pquiet = 1;
         break;
      case 'h':
         usage(argv[0]);
         break;
      default:
         usage(argv[0]);
         break;
      }

   if ((*pfilename != NULL && *pstr != NULL) ||
             (*pfilename == NULL && *pstr == NULL))
      usage(argv[0]);
}

static size_t
loadfile(const char *filename, char **out)
{
   int fd;
```

```
   fd = open(filename, O_RDONLY);
   if (fd < 0) {
      fprintf(stderr,
                "Error during opening file %s", filename);
      exit(EXIT_FAILURE);
   }

   off_t off;
   off = lseek(fd, (off_t) 0, SEEK_END);
   if (off < 0) {
      fprintf(stderr,
                "Error during processing file %s", filename);
      exit(EXIT_FAILURE);
   }

   size_t filesize;
   filesize = (size_t) off;

   *out = malloc((size_t) filesize);
   if (*out == NULL) {
      fprintf(stderr,
     "Can't allocate %lu bytes in memory", filesize);
      exit(EXIT_FAILURE);
   }

   off = lseek(fd, (off_t) 0, SEEK_SET);
   if (off < 0) {
      fprintf(stderr,
                "Error during processing file %s", filename);
      exit(EXIT_FAILURE);
   }

   ssize_t rb;
   rb = read(fd, *out, filesize);
   if (rb < 0) {
      fprintf(stderr,
                "Error during loading file %s", filename);
      exit(EXIT_FAILURE);
   }

   return (size_t) rb;
}

static double
test_whirlpool_refo(char *str, size_t size, uint8_t *dgst)
{
   NESSIEstruct ns;
   clock_t c;

   NESSIEinit(&ns);

   c = clock();
   NESSIEadd(str, 8 * size, &ns);
   NESSIEfinalize(&ns, dgst);

   return ((double) clock() - c) / CLOCKS_PER_SEC;
}
```

```
static double
test_whirlpool_ossl(char *str, size_t size, uint8_t *dgst)
{
   EVP_MD_CTX ctx;
   unsigned int m;
   clock_t c;


   c = clock();
   EVP_DigestInit(&ctx, EVP_whirlpool());
   EVP_DigestUpdate(&ctx, str, size);
   EVP_DigestFinal(&ctx, dgst, &m);

   return ((double) clock() - c) / CLOCKS_PER_SEC;
}

static double
test_whirlpool_ref(char *str, size_t size, uint8_t *dgst)
{
   struct context_ref ctx;
   clock_t c;

   whirlpool_init_ref(&ctx);

   c = clock();
   whirlpool_update_ref(&ctx, str, (uint32_t) size);
   whirlpool_final_ref(&ctx, dgst);

   return ((double) clock() - c) / CLOCKS_PER_SEC;
}

static double
test_whirlpool_slow(char *str, size_t size, uint8_t *dgst)
{
   struct context_slow ctx;
   clock_t c;

   whirlpool_init_slow(&ctx);

   c = clock();
   whirlpool_update_slow(&ctx, str, (uint32_t) size);
   whirlpool_final_slow(&ctx, dgst);

   return ((double) clock() - c) / CLOCKS_PER_SEC;
}

static double
test_whirlpool_asmv(char *str, size_t size, uint8_t *dgst)
{
   struct context_asmv ctx;
   clock_t c;

   whirlpool_init_asmv(&ctx);

   c = clock();
   whirlpool_update_asmv(&ctx, str, (uint32_t) size);
   whirlpool_final_asmv(&ctx, dgst);
```

```
   return ((double) clock() - c) / CLOCKS_PER_SEC;
}

static double
test_whirlpool_asmu(char *str, size_t size, uint8_t *dgst)
{
   struct context_asmu ctx;
   clock_t c;

   whirlpool_init_asmu(&ctx);

   c = clock();
   whirlpool_update_asmu(&ctx, str, (uint32_t) size);
   whirlpool_final_asmu(&ctx, dgst);

   return ((double) clock() - c) / CLOCKS_PER_SEC;
}

static void
printhex(uint8_t *a, size_t size)
{
   int i;

   for (i = 0; i < size; i++)
      printf("%02X", a[i]);
}

int main(int argc, char **argv)
{
   char *filename;
   char *str;
   int quiet;
   size_t size;

   parse_opts(argc, argv, &filename, &str, &quiet);

   if (filename != NULL) {
      size = loadfile(filename, &str);
      if (!quiet)
         printf("Using filename %s of %lu bytes size",
                     filename, size);
   } else {
      size = strlen(str);
      if (!quiet)
         printf("Using string
                     str, size);
   }

   double t;

   uint8_t refo_dgst[DIGESTBYTES];

   t = test_whirlpool_refo(str, size, refo_dgst);
   if (quiet) {
      printhex(refo_dgst, DIGESTBYTES);
      printf("");
   } else {
```

```
      printf("REFO ");
      printhex(refo_dgst, DIGESTBYTES);
      printf(" %f seconds", t);
   }

   uint8_t ossl_dgst[BLOCK_NBYTES];

   t = test_whirlpool_ossl(str, size, ossl_dgst);
   if (quiet) {
      printhex(ossl_dgst, BLOCK_NBYTES);
      printf("");
   } else {
      printf("OSSL ");
      printhex(ossl_dgst, BLOCK_NBYTES);
      printf(" %f seconds", t);
   }

   uint8_t ref_dgst[BLOCK_NBYTES];

   t = test_whirlpool_ref(str, size, ref_dgst);
   if (quiet) {
      printhex(ref_dgst, BLOCK_NBYTES);
      printf("");
   } else {
      printf("REF  ");
      printhex(ref_dgst, BLOCK_NBYTES);
      printf(" %f seconds", t);
   }

   uint8_t slow_dgst[BLOCK_NBYTES];

   t = test_whirlpool_slow(str, size, slow_dgst);
   if (quiet) {
      printhex(slow_dgst, BLOCK_NBYTES);
      printf("");
   } else {
      printf("SLOW ");
      printhex(slow_dgst, BLOCK_NBYTES);
      printf(" %f seconds", t);
   }

   uint8_t asmv_dgst[BLOCK_NBYTES];

   t = test_whirlpool_asmv(str, size, asmv_dgst);
   if (quiet) {
      printhex(asmv_dgst, BLOCK_NBYTES);
      printf("");
   } else {
      printf("ASMV ");
      printhex(asmv_dgst, BLOCK_NBYTES);
      printf(" %f seconds", t);
   }

   return EXIT_SUCCESS;
}
```