Практическая работа 7. Победитель AES – шифр Rijndael

1 История возникновения и основные понятия AES

В 1998 году NIST (National Institute of Standards and Technology) объявил конкурс на создание алгоритма симметричного шифрования, алгоритм получил название AES (Advanced Encryption Standard). Алгоритм планировалось принять как стандарт Соединенных Штатов Америки взамен устаревшего к этому времени стандарту DES (Digital Encryption Standard), являвшегося американским стандартом с 1977 года. Необходимость в принятии нового стандарта была вызвана небольшой длиной ключа DES (56 бит), что позволяло успешно применять метод прямого перебора ключей для взлома DES. Кроме того, архитектура DES была ориентирована на аппаратную реализацию, и программная реализация алгоритма на платформах с ограниченными ресурсами не давала достаточного быстродействия. Модификация TripleDES обладала достаточной длиной ключа, но при этом была еще медленнее.

2 января 1997 года NIST объявляет о намерении выбрать преемника для DES. Конкурс был объявлен 12 сентября 1997г. Свой алгоритм могла предложить любая организация или группа исследователей. NIST опубликовал все данные о тестировании кандидатов на роль AES и потребовал от авторов алгоритмов сообщить о базовых принципах построения алгоритмов, используемых в них константах, таблицах для замен (S-box) и т.п. В отличие от ситуации с DES, NIST при выборе AES не стал опираться на секретные и, как следствие, запрещенные к публикации данные об исследовании алгоритмов-кандидатов.

Требования к кандидатам на новый стандарт были следующими:

- блочный шифр;
- длина блока, равная 128 битам;
- ключи длиной 128, 192 и 256 бит.

Дополнительно кандидатам рекомендовалось:

- использовать операции, легко реализуемые как аппаратно (в микрочипах), так и программно (на персональных компьютерах и серверах);
- ориентироваться на 32-разрядные процессоры
- не усложнять без необходимости структуру шифра для того, чтобы все заинтересованные стороны были в состоянии самостоятельно провести независимый криптоанализ алгоритма и убедиться, что в нём не заложено каких-либо недокументированных возможностей.

Кроме того, алгоритм, претендующий на то, чтобы стать стандартом, должен распространяться по всему миру на не эксклюзивных условиях и без платы за пользование патентом.

Алгоритм AES прежде всего должен предлагать высокую степень защиты, обладать простой структурой и высокой производительностью. Уже на уровне внутренней архитектуры он должен обладать надежностью, достаточной для того, чтобы противостоять будущим попыткам его взлома.

Был проведен конкурс среди алгоритмов шифрования на роль AES, 2 октября 2000 года было объявлено, что победителем конкурса стал алгоритм **Rijndael** и началась процедура стандартизации. 28 февраля 2001 года был опубликован проект, а 26 ноября 2001 года AES был принят как стандарт FIPS 197.

В литературе Rijndael часто называют AES, но AES не тождественен Rijndael, т.к. оригинальный алгоритм Rijndael поддерживает более широкий диапазон длин ключей и блоков. В AES размер ключа фиксирован и равен 128 бит, а в Rijndael поддерживаются различные длины ключей - от 128 до 256 бит, с шагом 32 бита. AES оперирует с блоком данных 16 байт, а Rijndael позволяет выбирать размер блока.

Rijndael – быстрый и компактный алгоритм с простой математической структурой. Он продемонстрировал хорошую устойчивость к атакам на реализацию, при которых пытаются декодировать зашифрованное сообщение, анализируя внешние проявления алгоритма, в том числе уровень энергопотребления и время выполнения. Алгоритму присущ внутренний параллелизм, что позволяет без труда обеспечить эффективное использование процессорных ресурсов. Далее под AES можно понимать Rijndael с ключом в 128 бит и блоком данных 16 байт.

Таблица 1. Сравнительные характеристики алгоритмов – финалистов AES.

Преимущество	Алгоритмы				
	RIJNDAEL	**SERPENT**	**TWOFISH**	**RC6**	**MARS**
Быстродействие: аппаратная реализация	+ 	+ 	 	 	
программная реализация: на слабых ВС на мощных ВС	 +	 +	 	 + +	
Этап расширения ключа	+		+		
Распараллеливаемость	+				
Этап дешифрирования			+	+	+

Запас криптостойкости	Оптимум	Завышен	Завышен	Оптимум	Завышен
Количество голосов	86	59	31	23	13

2 октября 2000 года алгоритм RIJNDAEL был официально оглашен как победителем конкурса и получил второе название – *AES – Advanced Encryption Standard.*

2 Операции алгоритма Rijndael

Алгоритм **Rijndael** (допускаются различные варианты произношения, например, *англ.* "Ра"йндэл", *интерн.* "Рийнд'эл") разработан двумя специалистами (**Dae**men J., **Rij**men V.) по криптографии из Бельгии. Он является нетрадиционным блочным шифром, так как не использует сеть Фейштеля для криптопреобразований. Данный блочный шифр развивает нетрадиционную для последнего десятилетия структуру: прямоугольные KASLT–сети. Эта структура получила свое название из-за того, что шифруемый блок представляется в виде прямоугольника, например, 4×4 или 4× 6 байт, а затем над ним построчно, по столбцам и побайтно производятся криптопреобразования. KASLT представляет собой аббревиатуру английских терминов *Key Addition* (добавление ключа), *Substitution* (табличная подстановка) и *Linear Transposition* (линейное перемешивание).

Алгоритм **Rijndael** представляет каждый *блок* кодируемых данных в виде *двумерного массива байт* размером 4x4, 4x6 или 4x8 в зависимости от установленной длины блока.

Далее на соответствующих этапах преобразования производятся либо над независимыми столбцами, либо над независимыми строками, либо вообще над отдельными байтами в таблице.

Все преобразования в шифре имеют строгое математическое обоснование. Сама структура и последовательность операций позволяют выполнять данный алгоритм эффективно как на 8-битных так и на 32-битных процессорах. В структуре алгоритма заложена возможность параллельного исполнения некоторых операций, что на многопроцессорных рабочих станциях может еще поднять скорость шифрования в 4 раза.

Алгоритм **Rijndael** состоит из некоторого количества раундов (от 10 до 14 – это зависит от размера блока и длины ключа), в которых последовательно выполняются следующие *операции*:

ByteSub – табличная подстановка 8x8 бит (рис.1),

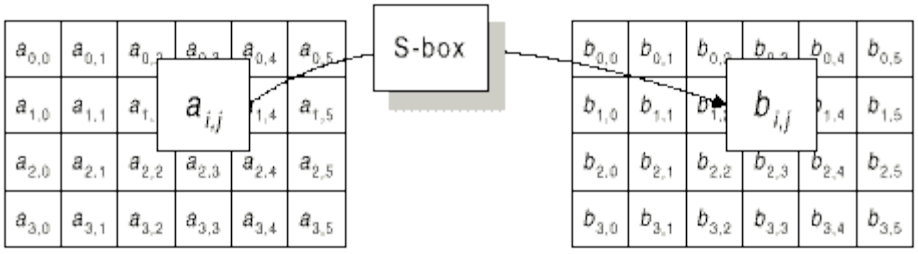


Рис.1.

ShiftRow – сдвиг строк в двумерном массиве на различные смещения (рис.2),

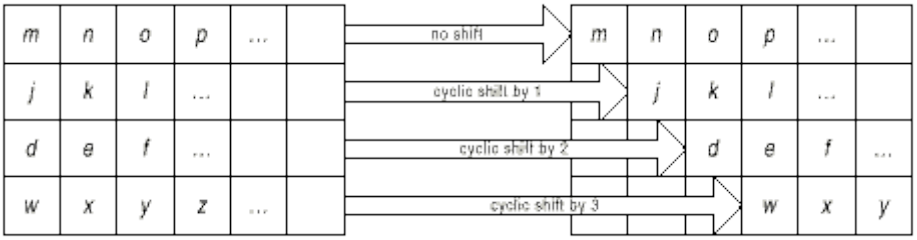


Рис.2.

MixColumn – математическое преобразование, перемешивающее данные внутри столбца (рис.3),

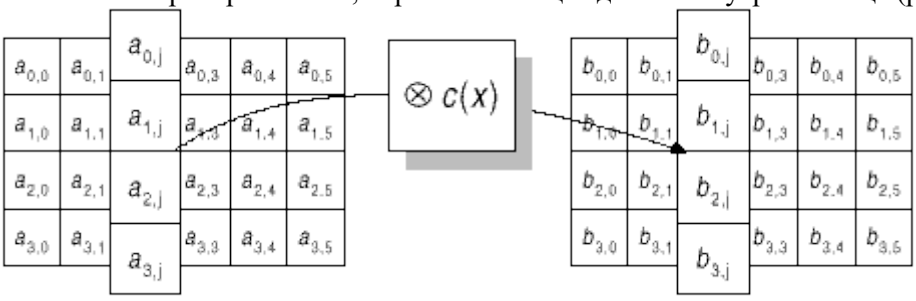


Рис.3.

AddRoundKey – добавление материала ключа операцией XOR (рис.4).

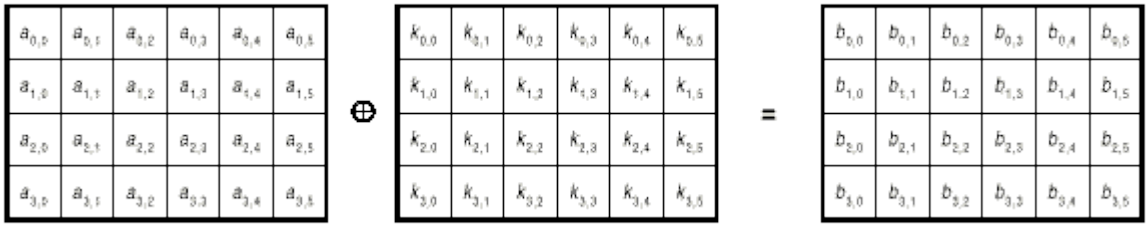


Рис.4.

В последнем раунде операция перемешивания столбцов отсутствует, что делает всю последовательность операций симметричной.

3 Предварительные математические понятия

Практически все операции Rijndael определяются на уровне байта. Байты можно рассматривать как элементы конечного поля GF (2^8). Некоторые операции определены в терминах четырехбайтных слов.
Введем основные математические понятия, необходимые для обсуждения алгоритма.

Поле GF(2^8)

Элементы конечного поля могут быть представлены несколькими различными способами. Для любой степени простого числа существует единственное конечное поле, поэтому все представления GF (2^8) являются изоморфными. Несмотря на подобную эквивалентность, представление влияет на сложность реализации. Выберем классическое полиномиальное представление.
Байт b, состоящий из битов b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0, представляется в виде полинома с коэффициентами из {0, 1}: $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$
Пример: байт с шестнадцатеричным значением '57' (двоичное 01010111) соответствует полиному $x^6 + x^4 + x^2 + x + 1$.

Сложение

В полиномиальном представлении сумма двух элементов является полиномом с коэффициентами, которые равны сумме по модулю 2 (т.е. 1 + 1 = 0) коэффициентов слагаемых.
Пример: '57' + '83' = 'DA' или в полиномиальной нотации: $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) =$
$x^7 + x^6 + x^4 + x^2$
В бинарной нотации мы имеем: 01010111 + 10000011 = 11011010. Очевидно, что сложение соответствует простому XOR (обозначается как $\oplus$) на уровне байта.
Выполнены все необходимые условия Абелевой группы: операция сложения (каждой паре элементов сопоставляется третий элемент группы, называемый их суммой), ассоциативность, нулевой элемент ('00'), обратный элемент (относительно операции сложения) и коммутативность.

Умножение

В полиномиальном представлении умножение в GF (2^8) соответствует умножению полиномов по модулю неприводимого двоичного полинома степени 8. Полином является неприводимым, если он не имеет делителей, кроме 1 и самого себя. Для Rijndael такой полином называется m(x) и определяется следующим образом:
$m(x) = x^8 + x^4 + x^3 + x + 1$ или '11B' в шестнадцатеричном представлении.
Пример: '57' • '83' = 'C1' или
$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1 =$
$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$
$(x^8 + x^4 + x^3 + x + 1)(x^5 + x^3) + x^7 + x^6 + 1 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$
Следовательно,
$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^8 + 1 \bmod (x^8 + x^4 + x^3 + x + 1) = x^7 + x^6 + 1$
Ясно, что результат является двоичным полиномом не выше 8 степени. В отличие от сложения, простой операции умножения на уровне байтов не существует.
Умножение, определенное выше, является ассоциативным, и существует единичный элемент ('01'). Для любого двоичного полинома b(x) не выше 8-й степени можно использовать расширенный алгоритм Евклида для вычисления полиномов a(x) и c(x) таких, что $b(x)\,a(x) + m(x)\,c(x) = 1$.
Следовательно, $a(x) \bullet b(x) \bmod m(x) = 1$ или $b^{-1}(x) = a(x) \bmod m(x)$
Более того, можно показать, что $a(x) \bullet (b(x) + c(x)) = a(x) \bullet b(x) + a(x) \bullet c(x)$
Из всего этого следует, что множество из 256 возможных значений байта образует конечное поле GF (2^8) с XOR в качестве сложения и умножением, определенным выше.

Умножение на x

Если умножить b(x) на полином x, мы будем иметь:
$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$

x • b(x) получается понижением предыдущего результата по модулю m(x). Если $b_7 = 0$, то данное понижение является тождественной операцией. Если $b_7 = 1$, m(x) следует вычесть (т.е. XORed). Из этого следует, что умножение на x может быть реализовано на уровне байта как левый сдвиг и последующий побитовый XOR с '1B'. Данная операция обозначается как b = xtime (a).

Полиномы с коэффициентами из GF

Полиномы могут быть определены с коэффициентами из $GF(2^8)$. В этом случае четырехбайтный вектор соответствует полиному степени 4.

Полиномы могут быть сложены простым сложением соответствующих коэффициентов. Как сложение в $GF(2^8)$ является побитовым XOR, так и сложение двух векторов является простым побитовым XOR.

Умножение представляет собой более сложное действие. Предположим, что мы имеем два полинома в $GF(2^8)$.

$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$

$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$

c(x) = a(x) b(x) определяется следующим образом:

$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$

$c_0 = a_0 • b_0$

$c_1 = a_1 • b_0 \oplus a_0 • b_1$

$c_2 = a_2 • b_0 \oplus a_1 • b_1 \oplus a_0 • b_2$

$c_3 = a_3 • b_0 \oplus a_2 • b_1 \oplus a_1 • b_2 \oplus a_0 • b_3$

$c_4 = a_3 • b_1 \oplus a_2 • b_2 \oplus a_1 • b_3$

$c_5 = a_3 • b_2 \oplus a_2 • b_3$

$c_6 = a_3 • b_3$

Ясно, что в таком виде c(x) не может быть представлен четырехбайтным вектором. Понижая c(x) по модулю полинома 4-й степени, результат может быть полиномом степени ниже 4. В Rijndael это сделано с помощью полинома $M(x) = x^4 + 1$, так как

$x^j \bmod (x^4 + 1) = x^{j \bmod 4}$

Модуль, получаемый из a(x) и b(x), обозначаемый $d(x) = a(x) \otimes b(x)$, получается следующим образом:

$d_0 = a_0 • b_0 \oplus a_3 • b_1 \oplus a_2 • b_2 \oplus a_1 • b_3$

$d_1 = a_1 • b_0 \oplus a_0 • b_1 \oplus a_3 • b_2 \oplus a_2 • b_3$

$d_2 = a_2 • b_0 \oplus a_1 • b_1 \oplus a_0 • b_2 \oplus a_3 • b_3$

$d_3 = a_3 • b_0 \oplus a_2 • b_1 \oplus a_1 • b_2 \oplus a_0 • b_3$

Операция, состоящая из умножения фиксированного полинома a(x), может быть записана как умножение матрицы, где матрица является циклической. Мы имеем

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Замечание: $x^4 + 1$ не является несократимым полиномом в $GF(2^8)$, следовательно, умножение на фиксированный полином необязательно обратимо. В алгоритме Rijndael выбран фиксированный полином, который имеет обратный.

Умножение на x

При умножении b(x) на полином x будем иметь: $b_3x^4 + b_2x^3 + b_1x^2 + b_0x$

$x \otimes b(x)$ получается понижением предыдущего результата по модулю $1 + x^4$.

Это дает $b_2x^3 + b_1x^2 + b_0x + b_3$

Умножение на x эквивалентно умножению на матрицу, как описано выше со всеми a_i = '00' за исключением a_1 = '01'. Имеем:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Следовательно, умножение на x соответствует циклическому сдвигу байтов внутри вектора.

4 Обоснование разработки

При разработке алгоритма учитывались следующие три критерия:

- противодействие всем известным атакам;
- скорость и компактность кода для широкого круга платформ;
- простота разработки.

В большинстве алгоритмов шифрования преобразование каждого раунда имеет структуру сети Фейштеля. В этом случае обычно часть битов в каждом промежуточном состоянии просто перемещается без изменения в другую половину. Преобразование раунда алгоритма Rijndael не имеет структуру сети Фейштеля. Вместо этого преобразование каждого раунда состоит из четырех различных преобразований, называемых *слоями*.

Каждый слой разрабатывался с учетом противодействия линейному и дифференциальному криптоанализу. В основу каждого слоя положена своя собственная функция:

1. Нелинейный слой состоит из параллельного применения S-boxes для оптимизации нелинейных свойств в наихудшем случае.
2. Слой линейного перемешивания строк гарантирует высокую степень диффузии для нескольких раундов.
3. Слой линейного перемешивания столбцов также гарантирует высокую степень диффузии для нескольких раундов.
4. Дополнительный слой ключа состоит из простого XOR промежуточного состояния с ключом раунда.

Перед первым раундом применяется дополнительное забеливание с использованием ключа. Причина этого состоит в следующем. Любой слой после последнего или до первого добавления ключа может быть просто снят без знания ключа и тем самым не добавляет безопасности в алгоритм (например, начальная и конечная перестановки в DES). Начальное или конечное добавление ключа применяется также в некоторых других алгоритмах, например IDEA, SAFER и Blowfish.

Для того чтобы сделать структуру алгоритма более простой, слой линейного перемешивания последнего раунда отличается от слоя перемешивания других раундов. Можно показать, что это в любом случае не повышает и не понижает безопасность. Это аналогично отсутствию операции swap в последнем раунде DES.

5 Спецификация алгоритма Rijndael

Rijndael является блочным алгоритмом шифрования с переменной длиной блока и переменной длиной ключа. Длина блока и длина ключа могут быть независимо установлены в 128, 192 или 256 бит.

Состояние, ключ шифрования и число раундов

Различные преобразования выполняются над промежуточным результатом, называемым состоянием.

Состояние можно рассматривать как двумерный массив байтов. Этот массив имеет четыре строки и различное число столбцов, обозначаемое как Nb, равное длине блока, деленной на 32.

Ключ также можно рассматривать как двумерный массив с четырьмя строками. Число столбцов ключа шифрования, обозначаемое как Nk, равно длине ключа, деленной на 32.

В некоторых случаях эти блоки также рассматриваются как одномерные массивы четырехбайтных векторов, где каждый вектор состоит из соответствующего столбца. Такие массивы имеют длину 4, 6 или 8 соответственно, и индексы в диапазонах 0 … 3, 0 … 5 или 0 … 7. Четырехбайтные вектора иногда мы будем называть *словами*.

Если необходимо указать четыре отдельных байта в четырехбайтном векторе, будет использоваться нотация (a, b, c, d), где a, b, c и d являются байтами в позициях 0, 1, 2 и 3, соответственно, в рассматриваемом столбце, векторе или слове.

$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$	$A_{0,4}$	$A_{0,5}$
$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$	$A_{1,4}$	$A_{1,5}$
$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$	$A_{2,4}$	$A_{2,5}$
$A_{3,0}$	$A_{3,1}$	$A_{3,1}$	$A_{3,3}$	$A_{3,4}$	$A_{3,5}$

$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$

Рис. 5. Пример состояния (с Nb = 6) и ключа шифрования (с Nk = 4)

Входы и выходы Rijndael считаются одномерными массивами из 8 байтов, пронумерованными от 0 до 4* Nb - 1. Следовательно, эти блоки имеют длину 16, 24 или 32 байта, и массив индексируется в диапазонах 0 … 15, 0 … 23 или 0 … 31.

Ключ считается одномерным массивом 8-битных байтов, пронумерованных от 0 до 4* Nk - 1. Следовательно, эти блоки имеют длину 16, 24 или 32 байта, и массив индексируется в диапазонах 0 … 15, 0 … 23 или 0 … 31.

Входные байты алгоритма отображаются в байты состояния в следующем порядке: $A_{0,0}$, $A_{1,0}$, $A_{2,0}$, $A_{3,0}$, $A_{0,1}$, $A_{1,1}$, $A_{2,1}$, $A_{3,1}$, …

Байты ключа шифрования отображаются в массив в следующем порядке: $K_{0,0}$, $K_{1,0}$, $K_{2,0}$, $K_{3,0}$, $K_{0,1}$, $K_{1,1}$, $K_{2,1}$, $K_{3,1}$, …

После выполнения операции шифрования выход алгоритма получается из байтов состояния аналогичным образом. Следовательно, если одноразмерный индекс байта в блоке есть n, и двухмерный индекс есть (i,j), то мы имеем:

I = n mod 4

J = ⌊ n / 4 ⌋

N = i + 4*j

Более того, индекс i является также номером байта в четырехбайтном векторе или слове, j является индексом вектора или слова во вложенном блоке.

Число раундов обозначается Nr и зависит от значений Nb и Nk, что показано в следующей таблице.

Таблица 1. Число раундов как функция от длины блока и длины ключа

Nr	Nb = 4	Nb = 6	Nb = 8
Nk = 4	10	12	14
Nk = 6	12	12	14
Nk = 8	14	14	14

Преобразование раунда

Преобразование раунда состоит из четырех различных преобразований. В нотации на псевдо С это можно записать следующим образом:

```
Round (State, RoundKey)
{
  ByteSub (State);
  ShiftRow (State);
  MixColumn (State);
  AddRoundKey (State, RoundKey);
}
```

Заключительный раунд алгоритма немного отличается и выглядит следующим образом:

```
FinalRound (State, RoundKey)
{
  ByteSub (State);
  ShiftRow (State);
  AddRoundKey (State, RoundKey);
}
```

Как мы видим, заключительный раунд эквивалентен остальным, за исключением того, что отсутствует слой MixColumn.

Преобразование ByteSub

Преобразование ByteSub является нелинейной байтовой подстановкой, выполняющейся для каждого байта состояния независимо. Таблица подстановки является обратимой и сконструирована в виде композиции двух преобразований:

1. Во-первых, берется мультипликативная инверсия в GF (2^8) с определенным выше представлением. '00' отображается сам в себя.
2. Затем применяется афинное (в GF (2)) преобразование, определяемое следующим образом:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Применение описанного S-box ко всем байтам состояния обозначается как ByteSub (State)
На рисунке 6 показан результат применения преобразования ByteSub к State.

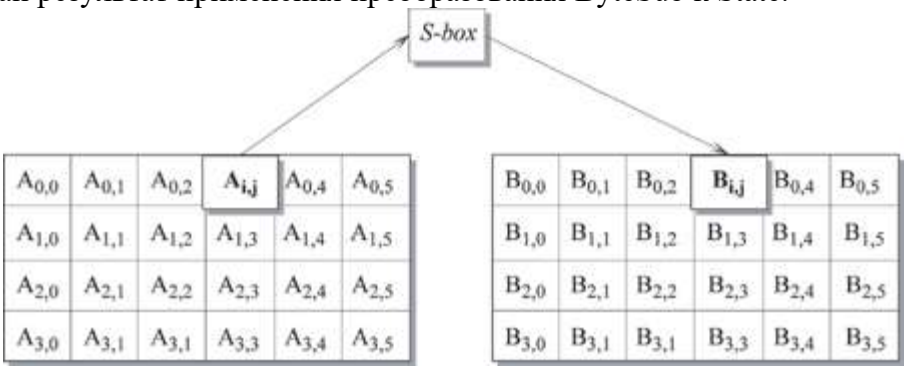


Рис. 6. Применение ByteSub для каждого байта в State

Инверсия ByteSub есть применение байтовой подстановки в соответствии с инверсной таблицей. Это получается инверсией афинного отображения и мультипликативной инверсией в GF (2^8).

Преобразование ShiftRow

В ShiftRow строки состояния циклически сдвигаются на различные значения. Нулевая строка не сдвигается. Строка 1 сдвигается на C_1 байтов, строка 2 на C_2 байтов, строка 3 на C_3 байтов. Величины C_1, C_2 и C_3 зависят от Nb. Значения приведены в следующей таблице.

Таблица 2. Величина сдвига в зависимости от длины блока

Nb	**C_1**	**C_2**	**C_3**
4	1	2	3
6	1	2	3
8	1	3	4

Операция сдвига строк на указанные значения обозначается как ShiftRow (State)
Инверсией для ShiftRow является циклический сдвиг трех нижних строк соответственно на Nb - C_1, Nb - C_2 и Nb - C_3 байт, чтобы байт в позиции j в строке i перемещался в позицию (j + Nb - C_i) mod Nb.

Преобразование MixColumn

В MixColumn столбцы состояния рассматриваются как полиномы в GF (2^8) и умножаются по модулю x^4 + 1 на фиксированный полином: c(x) = '03' x^3 + '01' x^2 + '01' x + '02'
Данный полином является взаимнопростым с x^4 + 1 и, следовательно, инвертируем. Как было описано выше, это может быть записано в виде умножения матрицы. Пусть b(x) = c(x) ⊗a(x)

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Применение данной операции ко всем столбцам состояния обозначается как MixColumn (State)
Инверсия MixColumn является аналогичным MixColumn. Каждый столбец преобразуется умножением его на полином d(x), определяемый следующим образом:
('03' x^3 + '01' x^2 + '01' x + '02') ⊗ d(x) = '01'
В результате получаем d(x) = '0B' x^3 + '0D' x^2 + '09' x + '0E'

Сложение с ключом раунда
Выполняется операция побитового XOR ключа раунда с текущим состоянием. Длина ключа раунда равна длине блока Nb. Данное преобразование обозначается как AddRoundKey (State, RoundKey)
AddRoundKey является инверсией самого себя.

Создание ключей раунда
Ключи раунда получаются из ключа шифрования с помощью преобразования, состоящего из двух компонентов: расширение ключа и выбор ключа раунда. Основной принцип состоит в следующем:

- Общее число битов ключа раунда равно длине блока, умноженной на количество раундов плюс 1. Например, для длины блока 128 бит и 10 раундов необходимо 1408 битов ключа раунда.
- Ключ шифрования расширяется в ExpandedKey.
- Ключи раунда получаются из этого ExpandedKey следующим способом: первый ключ раунда состоит из первых Nb слов, второй состоит из следующих Nb слов и т.д.

Расширение ключа
Expanded Key является линейным массивом четырехбайтных слов и обозначается как W [Nb * (Nr + 1)]. Первые Nk слов состоят из ключа шифрования. Остальные слова определяются рекурсивно. Функция расширения ключа зависит от значения Nk: существует версия функции для Nk, равным или меньшим 6, и версия для Nk больше 6.

Для Nk ≤6 мы имеем:

```
KeyExpansion (byte Key [4*Nk]
          word W[Nb * (Nr + 1)])
{
 for (i = 0; i < Nk; i++)
   W[i] =(Key [4*i], Key [4*i+1],  Key [4*i+2], Key [4*i+3]);
 for (i = Nk; i < Nb * (Nr + 1); i++) {
   temp = W [i - 1];
   if (i % Nk == 0)
     temp = SubByte (RotByte (temp)) ^ Rcon [i / Nk];
   W [i] = W [i- Nk] ^ temp;
 }
}
```

В данном случае SubByte (W) является функцией, которая возвращает четырехбайтное слово, в котором каждый байт является результатом применения S-box Rijndael к байту в соответствующей позиции во входном слове. Функция RotByte (W) возвращает слово, в котором байты циклически переставлены таким образом, что для входного слова (a, b, c, d) создается выходное слово (b, c, d, a).
Можно заметить, что первые Nk слов заполняются ключом шифрования. Каждое следующее слово W[i] равно XOR предыдущего слова W[i-1] и позиций слова Nk до W[i - Nk]. Для слов в позициях, которые кратны Nk, сначала применяется преобразование XOR к W[i-1] и константой раунда. Данное

преобразование состоит из циклического сдвига байтов в слове RotByte, за которым следуют применение табличной подстановки для всех четырех байтов в слове (SubByte).

Для Nk > 6 мы имеем:

```
KeyExpansion (byte Key [4*Nk]
          word W [Nb* (Nr+1)])
{
 for (i=0; i < Nk; i++)
   W[i]= (key [4*i], key [4*i+1],   key [4*i+2], key [4*i+3]);
 for (i = Nk; i < Nb * (Nr + 1); i++) {
   temp = W [i-1];
   if (i % Nk == 0)
     temp = SubByte (RotByte (temp)) ^  Rcon [i / Nk];
   else if (i % Nk == 4)
     temp = SubByte (temp);
   W[i] = W[i - Nk] ^ temp;
 }
}
```

Отличие в схеме для Nk ≤6 состоит в том, что для i-4 кратных Nk, SubByte применяется для W[i-1] перед XOR.

Константы раунда не зависят от Nk и определяются следующим образом:

Rcon [i] = (RC [i], '00', '00', '00')

RC [i] являются элементами в GF (2^8) со значением $x^{(i-1)}$ таким, что:

RC [1] = 1 (т.е. '01')

RC [i] = x (т.е. '02') • (RC [i-1]) = x(i-1)

Выбор ключа раунда

Ключ раунда i получается из слов буфера ключа раунда W [Nb * i] до W [Nb * (i+1)].

6 Алгоритм шифрования

Алгоритм шифрования Rijndael состоит из

- начального сложения с ключом;
- Nr - 1 раундов;
- заключительного раунда.

В С-подобном представлении это выглядит так:

```
Rijndael (State, CipherKey)
{
  KeyExpansion (CipherKey, ExpandedKey);
  AddRoundKey (State, ExpandedKey);
  for (i=1; i < Nr; i++)
    Round (State, ExpandedKey + Nb*i);
  FinalRound (State, ExpandedKey + Nb*Nr)
}
```

Расширение ключа может быть выполнено заранее, и Rijndael может быть специфицирован в терминах расширенного ключа.

```
Rijndael (State, ExpandedKey)
{
  AddRoundKey (State, ExpandedKey);
  for (i=1; i < Nr; i++)
    Round (State, ExpandedKey + Nb*i);
  FinalRound (State, ExpandedKey + Nb*Nr)
}
```

Замечание: расширенный ключ всегда получается из ключа шифрования и никогда не специфицируется непосредственно. Тем не менее, на выбор самого ключа шифрования ограничений не существует.

7 Преимущества алгоритма

Преимущества, относящиеся к аспектам реализации:

- Rijndael может выполняться быстрее, чем обычный блочный алгоритм шифрования. Выполнена оптимизация между размером таблицы и скоростью выполнения.
- Rijndael можно реализовать в смарт-карте в виде кода, используя небольшой RAM и имея небольшое число циклов. Выполнена оптимизация размера ROM и скорости выполнения.
- Преобразование раунда допускает параллельное выполнение, что является важным преимуществом для будущих процессоров и специализированной аппаратуры.
- Алгоритм шифрования не использует арифметические операции, поэтому тип архитектуры процессора не имеет значения.

Простота разработки:

- Алгоритм шифрования полностью "самоподдерживаемый". Он не использует других криптографических компонентов, S-box'ов, взятых из хорошо известных алгоритмов, битов, полученных из специальных таблиц, чисел типа p и тому подобных уловок.
- Алгоритм не основывает свою безопасность или часть ее на неясностях или плохо понимаемых итерациях арифметических операций.
- Компактная разработка алгоритма не дает возможности спрятать люки.

Переменная длина блока:

- Длины блоков от 192 до 256 бит позволяют создавать хэш-функции без коллизий, использующие Rijndael в качестве функции сжатия. Длина блока 128 бит сегодня считается для этой цели недостаточной.

Расширения:

- Разработка позволяет специфицировать варианты длины блока и длины ключа в диапазоне от 128 до 256 бит с шагом в 32 бита.
- Хотя число раундов Rijndael зафиксировано в данной спецификации, в случае возникновения проблем с безопасностью он может модифицироваться и иметь число раундов в качестве параметра.

8 Расширения

Различная длина блока и ключа шифрования

Обработка ключа поддерживает длину ключа, которая была бы кратна 4 байтам. Единственным параметром, который необходим для определения другой длины ключа, отличной от 128, 192 или 256 бит, является число раундов алгоритма.

Структура алгоритма допускает произвольную длину блока, кратную 4 байтам, с минимумом в 16 байтов. Добавление ключа и ByteSub и MixColumn преобразования не зависят от длины блока. Единственным преобразованием, которое зависит от длины блока, является ShiftRow. Для каждой длины блока должен быть определен специальный массив C_1, C_2, C_3.

Можно определить расширение Rijndael, которое также поддерживает длины блока и ключа между 128 и 256 битами с приращением в 32 бита. Число раундов определяется так:

$Nr = \max(Nk, Nb) + 6$

Это расширяет правило для количества раундов для альтернативных длин блока и ключа.

Дополнительные значения C_1, C_2 и C_3 определены в следующей таблице.

Таблица 3. Величина сдвига в зависимости от длины блока

Nb	C1	C2	C3
5	1	2	3
7	1	2	4

9 Другие возможности

Обсудим функции, отличные от шифрования, которые могут выполняться алгоритмом Rijndael.

MAC

Rijndael может применяться в качестве алгоритма MAC. Для этого следует использовать блочный алгоритм в режиме CBC-MAC.

Хэш-функция

Rijndael может использоваться в качестве итерационной хэш-функции. При этом Rijndael применяется в качестве функции раунда. Существует одна возможная реализация. Рекомендуется использовать длину

блока и ключа, равной 256 битам. Блок сообщения подается на вход в качестве ключа шифрования. Другим входом является переменная, выход алгоритма XORed с данной переменной, и полученное значение является новым значением этой переменной.

Генератор псевдослучайных чисел

Существует много способов, с помощью которых Rijndael можно использовать в качестве генератора псевдослучайных чисел. Рассмотрим один из них, в котором применяются длина блока и длина ключа 256 бит.

Существует три операции:

Reset:

- Ключ алгоритма шифрования и состояние устанавливаются в ноль.

Seeding (и reseeding):

- "Seed биты" выбираются таким образом, чтобы обеспечивать минимальную энтропию. Они дополняются нулями до тех пор, пока результирующая строка не будет иметь длину, кратную 256 битам.
- Вычисляется новый ключ шифрования шифрованием с помощью Rijndael блока битов seed, используя текущий ключ шифрования. Это выполняется рекурсивно до тех пор, пока все блоки seed не будут обработаны.
- Состояние изменяется путем применения Rijndael с новым ключом шифрования.

Генератор псевдослучайного числа:

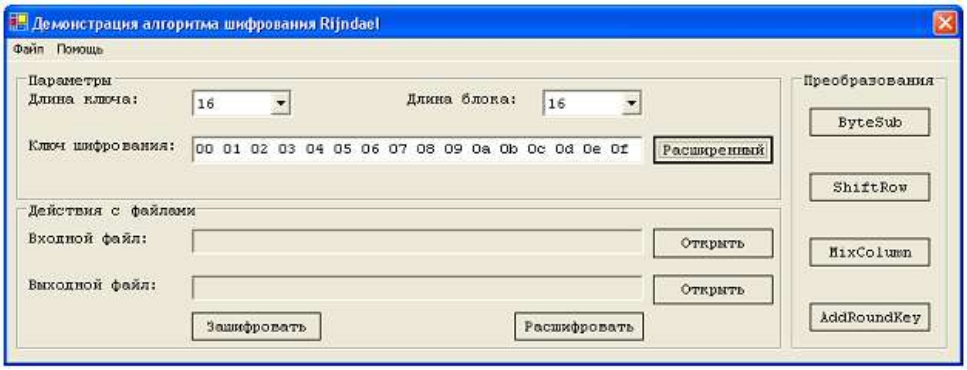
- Состояние изменяется путем применения Rijndael с новым ключом шифрования. Первые 128 бит состояния рассматриваются как псевдослучайное число. Данный шаг может быть повторен много раз.

Альтернативный вариант 1. **Задание для индивидуальной работы (*работа с готовым ПО* RijndaelDemo.exe)**

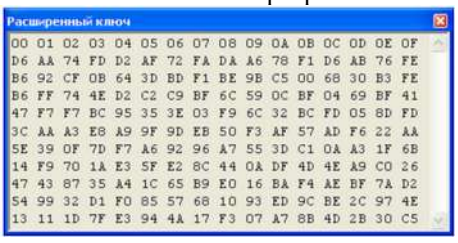
Описание демонстрационной программы

Программа выполнена на языке C# и состоит из двух элементов – файла Rijndael.dll, содержащего реализацию алгоритма шифрования, и демонстрационного приложения RijndaelDemo.exe. Для работы приложения необходима ОС Windows с установленным .NET Framework v1.1.

В основном окне демонстрационной программы задаются длина ключа, длина блока, также можно посмотреть расширенный ключ шифрования, вычисляемый в соответствии с заданным ключом шифрования.

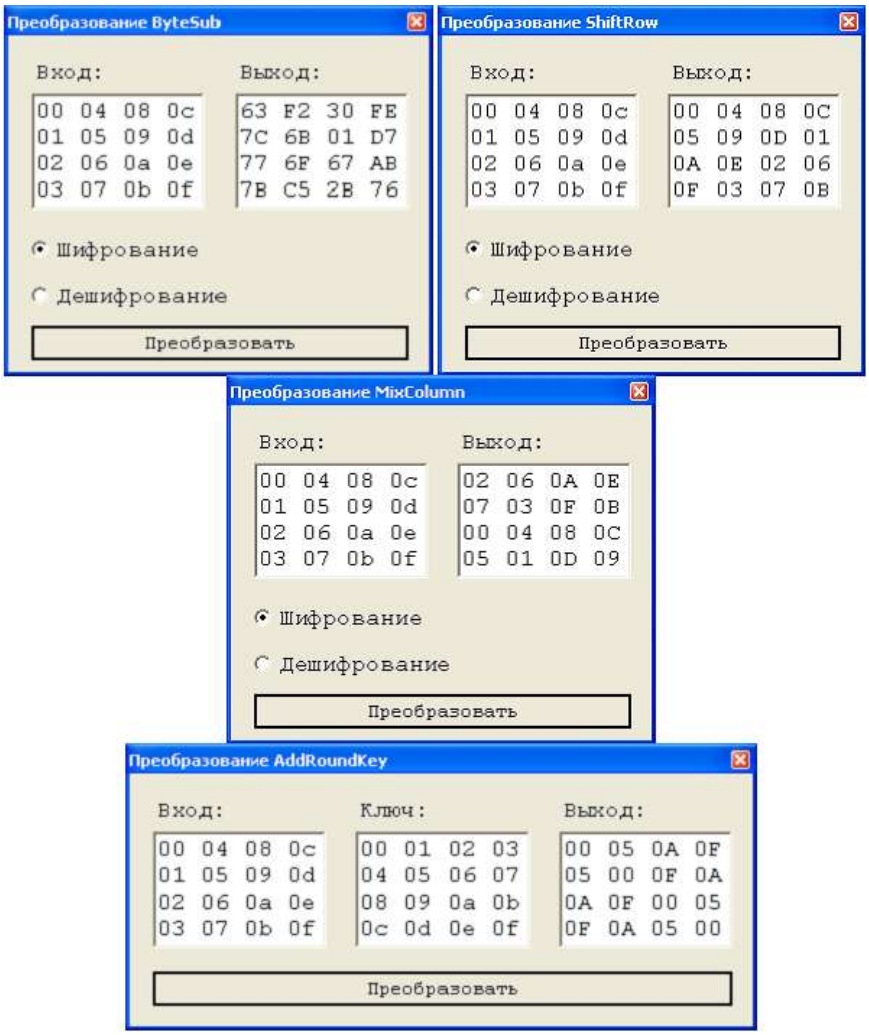


Основное окно программы



Окно демонстрации расширенного ключа

В программе предусмотрена возможность подробно рассмотреть действие всех цикловых преобразований (ByteSub, ShiftRow, MixColumn, AddRoundKey) как при шифровании, так и при расшифровании.



При шифровании предлагается выбрать исходный файл и файл для результата шифрования, при расшифровании соответственно зашифрованный файл и файл, предназначенный для помещения результата расшифрования.

В процессе используются, указанные заранее в главном окне программы, ключ шифрования и длины ключа и блока.

Задание

1. Ознакомиться со сведениями о программе RijndaelDemo. Запустить модуль RijndaelDemo.exe.
2. Изучить на примере обычных *текстовых* файлов способы шифрования и расшифрования с помощью алгоритма Rijndael.

Подробно рассмотреть действие всех цикловых преобразований (ByteSub, ShiftRow, MixColumn, AddRoundKey), как при шифровании, так и расшифровании.

Исходный текст для шифрования должен быть подготовлен заранее и сохранен в файле *.txt. Файл должен также содержать информацию о Вас (фамилия, имя, отчество, студент 2 курса специальности «Информационная безопасность автоматизированных систем»)

3. Сохранить в отчете экранные формы, демонстрирующие процесс шифрования и расшифрования информации, проанализировать полученные результаты.
4. Включить в отчет о лабораторной работе ответы на контрольные вопросы, выбранные в соответствии с номером варианта.

Номер варианта	Контрольные вопросы
1, 10, 19	Сравните основные характеристики алгоритмов Rijndael и ГОСТ 28147-89.
2, 11, 20	Сравните основные характеристики алгоритмов Rijndael и DES.
3, 12, 21	Приведите обобщенные схемы шифрования данных с помощью алгоритма Rijndael и ГОСТ 28147-89. Дайте их сравнительный анализ.
4, 13, 22	Сравните один раунд шифрования данных с помощью алгоритма Rijndael и ГОСТ 28147-89.

5, 14, 23	Сравните эквивалентность прямого и обратного преобразований в алгоритмах Rijndael и ГОСТ 28147-89.
6, 15, 24	Сравните выработку ключевой информации в алгоритмах Rijndael и ГОСТ 28147-89.
7, 16, 25	Сравните алгоритмы Rijndael и ГОСТ 28147-89 по показателям диффузии.
8, 17, 26	Сравните алгоритмы Rijndael и ГОСТ 28147-89 по показателям стойкости
9, 18, 27	Сравните алгоритмы Rijndael и ГОСТ 28147-89 по показателям производительности и удобству реализации.

Альтернативный вариант 2. **Задание для индивидуальной работы (*программирование*)**

В лабораторной работе необходимо реализовать шифрование и дешифрование с помощью алгоритма AES Rijndael для 128-битных ключа и блока.
Программная реализация криптографической системы, основанной на алгоритме шифрования Rijndael, должна быть оформлена как некоторая программная оболочка. В программной реализации должен быть разработан интерфейс, удобный для эксплуатации программы. В интерфейсе следует предусмотреть:
• два режима формирования ключа – ключ задан и ключ формируется по умолчанию;
• ввода начальной информации из сформированного заранее файла, и файла, который создается в оболочке программы;
• режимы шифрования, которые предусмотрены в Rijndael;
• режимы шифрования и дешифрования информации;

В алгоритме Rijndael на вход подается ключ и блок данных, на выходе получается зашифрованный или расшифрованный блок (во все случаях данные в шестнадцатеричном виде, используются 32 шестнадцатеричные цифры подряд без пробелов. Например:
Открытый текст: **00112233445566778899aabbccddeeff**
Ключ: **000102030405060708090a0b0c0d0e0f**
Шифротекст: **69c4e0d86a7b0430d8cdb78070b4c55a**

Подготовить отчет по работе. В отчете описать алгоритм Rijndael, описать структуру представления данных в программе, основные функции программы, назначение функций, входные и выходные параметры функций. В отчет включить описание алгоритма генерации ключа, детали программной реализации, которые представляют интерес с точки зрения разработчика.

Литература
0. Daemen J., Rijmen V. AES Proposal: Rijndael: [http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf], 09.03.1999.
1. Бабаш А.В., Шанкин Г.П. Криптография. / Под редакцией В.П.Шерстюка, Э.А. Применко. – М.: СОЛОН-Р, 2007. – 512 с.
2. Бабаш А.В. Криптографические и теоретико-автоматные аспекты современной защиты информации. Криптографические методы защиты. – М.: Изд.центр ЕАОИ, 2009. – 414 с.
3. Баранова Е.К. Эффективное кодирование и защита информации: Текст лекций для студентов специальности 510200. – М.: МГУЛ, 2002. – 88 с.
4. Мельников В.В. Защита информации в компьютерных системах. М.: Финансы и статистика; Электроинформ, 1997. 368 с.
5. Романец Ю.В., Тимофеев П.А., Шаньгин В.Ф. Защита информации в компьютерных системах и сетях. – М.: Радио и связь, 2001. – 376 с.
6. Смарт Н. Криптография. – М.: Техносфера, 2006. – 528 с.