3.7. Алгоритм AES

Инициатива в разработке *AES* принадлежит NIST. В августе 1999 года были представлены пять финалистов. Ими стали *MARS, RC6™, Rijndael, Serpent* и *Twofish*. В итоге в качестве нового стандарта был принят *Rijndael.*

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.

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}:

$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$

В полиномиальном представлении сумма двух элементов является полиномом с коэффициентами, которые равны сумме по модулю 2 коэффициентов слагаемых.

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:

$m(x) = x^8 + x^4 + x^3 + x + 1$

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.

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 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 на x

При умножении b(x) на полином x будем иметь:

$b_3x^4 + b_2x^3 + b_1x^2 + b_0x$

x $\oplus$ 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.

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.

Спецификация алгоритма

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.

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

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

Рисунок 3.14 – 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:

$I = n \bmod 4$

$J = \lfloor n / 4 \rfloor$

$N = i + 4*j$

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 3.1. 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 C 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.

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 $Nk \leq 6$ 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 следует 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)

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

Алгоритм шифрования

Алгоритм шифрования Rijndael состоит из

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

В C-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.