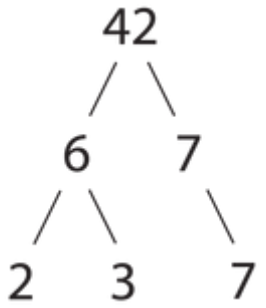
Простые числа. Поля Галуа. Арифметика в конечных полях. Алгоритм AES

Лекция №5

Простые числа

- **Простое число** – это целое положительное число, имеющее ровно два различных натуральных делителя – единицу и самого себя
- Другими словами число **x** является простым, если оно больше единицы и делится без остатка только на 1 и на **x**
- Например, число 5 – простое число, а 6 – составное число, так как помимо 1 и 6 делится еще на 2 и на 3
- Натуральные числа, которые больше единицы и не являются простыми, называются **составными**
- **Основная теорема арифметики** устанавливает центральную роль простых чисел в теории чисел: любое целое число, большее 1, либо является простым, либо может быть выражено как произведение простых чисел, причём это выражение единственно с точностью до порядка сомножителей



Алгоритмы поиска простых чисел

- **Алгоритмы перебора**
  - Простой перебор
  - Решето Эратосфена
  - Проверка делимости на первые N простых чисел
- **Вероятностные алгоритмы**
  - Тест Рабина-Миллера
  - Алгоритм Лемана

О других алгоритмах определения простоты числа: https://ru.wikipedia.org/wiki/Простое_число (раздел «Тест простоты»)

Тест Рабина-Миллера

- Алгоритм является **вероятностно-полиномиальным**, т.е. быстро вычислимым (за приемлемое время) и дающим ответ с высокой точностью. При этом, жертвуя временем, можно добиться любой требуемой точности
- Таким образом, отвечая на вопрос “является ли число простым?”, тест Миллера – Рабина дает один из двух вариантов ответа: “**число составное**” или “**вероятно простое**“
- Порядок работы:
  - Генерируют случайное число P длиной n-бит
  - Старшие и младшие биты числа P устанавливают в 1
  - Проверяют чтобы P не делилось на маленькие простые числа (до 2000)
  - Для увеличения вероятности простоты числа данный тест прогоняют несколько раз (k раундов) – обычно 5 тестов
  - Число P должно пройти все тесты
- Пример кода на C#: https://vscode.ru/prog-lessons/test-millera-rabina-na-prostotu-chisla.html

Тест Лемана

1. Выбрать случайно число ***a***, меньшее ***p***
2. Вычислить $a^{(p-1)/2} \bmod p$
3. Если $a^{(p-1)/2} <> 1 \pmod p$ или $a^{(p-1)/2} <> -1 \pmod p$, то p не является простым
4. Если $a^{(p-1)/2} = 1 \pmod p$ или $a^{(p-1)/2} = -1 \pmod p$, то вероятность того, что число p не является простым, не больше 50%

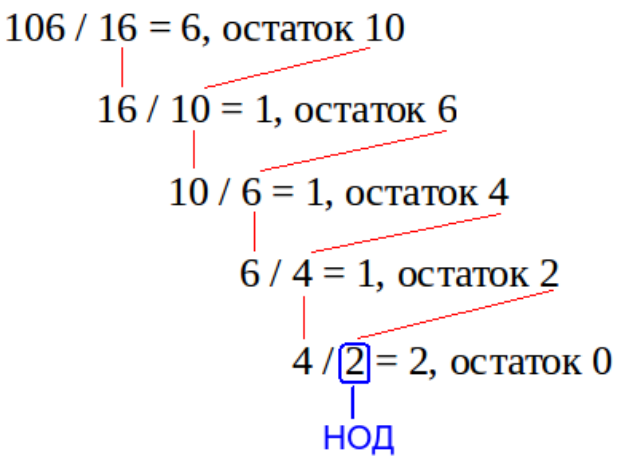
- Порядок работы:
  - Генерируют случайное число P длиной n-бит
  - Старшие и младшие биты числа P устанавливают в 1
  - Проверяют чтобы P не делилось на маленькие простые числа (до 2000)
  - Повторить эту проверку t раз.
  - Если результат вычислений равен 1 или -1, но не всегда равен 1, то p является простым числом с вероятностью ошибки $(1/2)^t$
- Пример кода на C#: http://www.cyberforum.ru/cpp-beginners/thread1121987.html

Взаимно простые числа

- Натуральные числа **a** и **b** называют **взаимно простыми**, если их наибольший общий делитель равен 1

НОД(a, b) = 1

- Другими словами, если числа a и b не имеют никаких общих делителей, кроме 1, то они взаимно просты
- Пример взаимно простых чисел: 13 и 16 (НОД = 1)
- Пример НЕ взаимно простых чисел: 20 и 6 (НОД <> 1)
- Для нахождения НОД используется **алгоритм Евклида**:




```
static long GCD(long a, long b)
{
    if (a == 0)
        return Math.Abs(b);
    if (b == 0)
        return Math.Abs(a);
    for ( ; ; )
        if ((a %= b) == 0)
            return Math.Abs(b);
        else if ((b %= a) == 0)
            return Math.Abs(a);
}
```

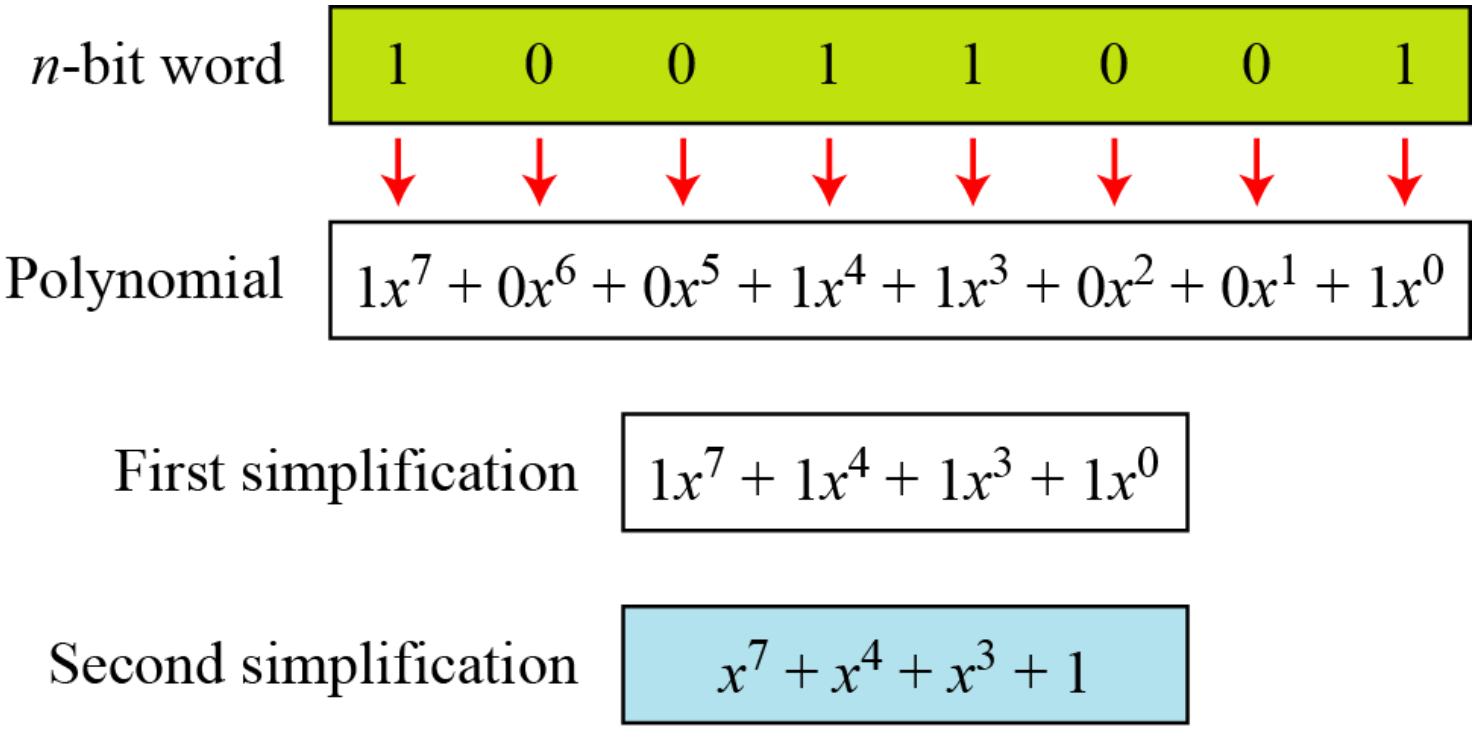
Факторизация чисел

- **Факторизацией** натурального числа называется его разложение в произведение простых сомножителей. Существование и единственность такого разложения следует из основной теоремы арифметики

  23244 = 2 * 2 * 3 * 13 * 149 = 2^2 * 3 * 13 * 149

- Разложение **n = p_1 * p_2 *...* p_t** называется разложением на простые множители числа **n**
- Любое разложение на простые числа будет **тождественным**, за исключением порядка делителей
- **Лемма Евклида**: Если простое число p делит без остатка произведение двух целых чисел **x*y**, то **p** делит без остатка или **x** или **y**
- Например, 52*2 = 104. Простое число 13 делит 104 без остатка (104/13=8), следовательно оно делит без остатка и один из сомножителей (52/13 = 4)

Полиномиальная форма числа



Конечные поля - поля Галуа

- **Конечное поле**, или **поле Галуа́** в общей алгебре — поле, состоящее из конечного числа элементов. Число элементов в поле называется его **порядком**
- Конечное поле обычно обозначается ***GF(q)*** и называется **полем Галуа порядка *q***, где *q* — число элементов поля
- $\mathbf{q} = p^n$, ***p*** – простое число, При этом *p* будет являться **характеристикой** этого поля
- Пример поля GF(3^1) с определением операций сложения и умножения

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

×	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Э. Галуа.

Пример поля $GF(2^3)$

- Данное поле состоит из чисел от 0 до 7
- Определим **операцию сложения**, которая является простым побитовым сложением по модулю 2
- Пример: 5 + 3 = 110 = 6

$$\begin{array}{r} 1\,0\,1 \\ \oplus\ 0\,1\,1 \\ \hline 1\,1\,0 \end{array}$$

- Такой же результат мы получим, если будем складывать данные числа в полиномиальной форме:

$$(X^2 + 1) + (X + 1) = X^2 + X + 1 + 1 = X^2 + X = 110$$

Пример поля $GF(2^3)$

- **Операция умножения** несколько сложнее и ее можно осуществить только в полиномиальной форме
- Пример умножения:

$$5*7 = (X^2 + 1)*(X^2 + X + 1) = X^4 + X^3 + X^2 + X^2 + X + 1 =$$
$$= X^4 + X^3 + X + 1 = 11011 = \mathbf{27}$$

- Результат умножения 27 не входит в используемое поле $GF(2^3)$ (оно же состоит из чисел от 0 до 7, как было сказано выше)
- Чтобы бороться с этой проблемой, необходимо использовать **порождающий полином**. Порождающий полином является неприводимым, то есть простым (по аналогии с простыми числами делится без остатка на 1 и на самого себя). В арифметике полей Галуа неприводимым полиномом является аналог простых чисел
- Используем для примера порождающий полином $f(x) = x^3+x+1$, при этом полагаем, что $x^3+x+1 = 0$

Пример поля $GF(2^3)$

- Вернемся к примеру с умножением

$$5 \cdot 7 = x^4 + x^3 + x + 1 = \left| \begin{array}{c} \text{Добавим некоторые слагаемые} \\ \text{но так, чтобы ничего не изменилось} \\ \text{(еще раз напомню, что под сложением} \\ \text{понимаю сложение по модулю 2)} \end{array} \right| =$$

$$(x^4 + x^2 + x) + (x^3 + x + 1) + x^2 + x = x \cdot (x^3 + x + 1) + (x^3 + x + 1) + x^2 + x =$$

$$= \left| \begin{array}{c} \text{Так как } x^3 + x + 1 = 0, \text{то} \\ \text{полученное выражение} \\ \text{можно упростить} \end{array} \right| = x^2 + x = 110 = 6$$

Пример поля GF(2^3)

- Такой же результат можно получить как остаток от деления полинома, полученного при умножении на порождающий полином

$$
\begin{array}{rl|l}
 & x^4+x^3+x+1 & x^3+x+1 \\
+ & x^4+x^2+x & \hline x+1 \\
\hline
 & x^3+x^2+1 & \\
+ & x^3+x+1 & \\
\hline
 & x^2+x & \text{- Остаток от деления} = 110 = 6
\end{array}
$$

Пример поля GF(2^3)

- Составим **таблицу умножения**

Полиномиальное представление		**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	**1**	1	2	3	4	5	6	7
x	**2**	2	4	6	3	1	7	5
x+1	**3**	3	6	5	7	4	1	2
x^2	**4**	4	3	7	6	2	5	1
x^2+1	**5**	5	1	4	2	7	3	6
x^2+x	**6**	6	7	1	5	3	2	4
x^2+x+1	7	7	5	2	1	6	4	3

- **Операцию деления** в полиномиальной форме понять, возможно, но достаточно тяжело. Поэтому гораздо лучше осуществлять его по таблице умножения
- Пример: 6÷5=7

Пример поля $GF(2^3)$

- Большое значение имеет **таблица степеней** элементов поля Галуа. Возведение в степень также осуществляется в полиномиальной форме, аналогично умножению

- $5^2 = [\![(x^2+1)]\!]^2 = x^4+x^2+x^2+1 = x^4+x^2+x+x^2+x+1 = x\cdot(x^3+x+1)+ x^2+x+1 = x^2+x+1 = 111 = 7$
- Составим таблицу степеней

			Степени							
			0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Полиномиальное представление	1	**1**	1	1	1	1	1	1	1	1
	x	**2**	1	2	4	3	6	7	5	1
	x+1	**3**	1	3	5	4	7	2	6	1
	x^2	**4**	1	4	6	5	2	3	7	1
	x^2+1	**5**	1	5	7	6	3	4	2	1
	x^2+x	**6**	1	6	2	7	4	5	3	1
	x^2+x+1	7	1	7	3	2	5	6	4	1

Пример поля GF(2^3)

- Таблица степеней обладает **цикличностью**: седьмая степень соответствует нулевой, значит восьмая соответствует первой и т.д.
- В полях Галуа существует понятие **примитивного члена** – элемент поля, чьи степени содержат **все** ненулевые элементы поля. Просмотрев таблицу степеней видно, что этому условию соответствуют все элементы (ну кроме 1 естественно)

			Степени							
			0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Полиномиальное представление	1	**1**	1	1	1	1	1	1	1	1
	x	**2**	1	2	4	3	6	7	5	1
	x+1	**3**	1	3	5	4	7	2	6	1
	x^2	**4**	1	4	6	5	2	3	7	1
	x^2+1	**5**	1	5	7	6	3	4	2	1
	x^2+x	**6**	1	6	2	7	4	5	3	1
	x^2+x+1	**7**	1	7	3	2	5	6	4	1

Пример поля GF(2^3)

- Однако это выполняется не всегда, для примера рассмотрим таблицу степеней для GF(16)

	Степени															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	3	6	12	11	5	10	7	14	15	13	9	1
3	1	3	5	15	2	6	10	13	4	12	7	9	8	11	14	1
4	1	4	3	12	5	7	15	9	2	8	6	11	10	14	13	1
5	1	5	2	10	4	7	8	14	3	15	6	13	12	9	11	1
6	1	6	7	1	6	7	1	6	7	1	6	7	1	6	7	1
7	1	7	6	1	7	6	1	7	6	1	7	6	1	7	6	1
8	1	8	12	10	15	1	8	12	10	15	1	8	12	10	15	1
9	1	9	13	15	14	7	10	5	11	12	6	3	8	4	2	1
10	1	10	8	15	12	1	10	8	15	12	1	10	8	15	12	1
11	1	11	9	12	13	6	15	3	14	8	7	4	10	2	5	1
12	1	12	15	8	10	1	12	15	8	10	1	12	15	8	10	1
13	1	13	14	10	11	6	8	2	9	15	7	5	12	3	4	1
14	1	14	11	8	9	7	12	4	13	10	6	2	15	5	3	1
15	1	15	10	12	8	1	15	10	12	8	1	15	10	12	8	1

Пример поля $GF(2^3)$

- Для полей, которые мы рассматриваем, то есть с характеристикой 2, **в качестве примитивного члена всегда выбирают 2**. Учитывая его свойство, любой элемент поля можно выразить через степень примитивного члена.
- Пример: $5=2^6$, $7=2^5$

			Степени							
			0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Полиномиальное представление	1	**1**	1	1	1	1	1	1	1	1
	x	**2**	1	2	4	3	6	7	5	1
	x+1	**3**	1	3	5	4	7	2	6	1
	x^2	**4**	1	4	6	5	2	3	7	1
	x^2+1	**5**	1	5	7	6	3	4	2	1
	x^2+x	**6**	1	6	2	7	4	5	3	1
	x^2+x+1	7	1	7	3	2	5	6	4	1

Пример поля $GF(2^3)$

- Воспользовавшись этим свойством, и учитывая цикличность таблицы степеней, попробуем снова перемножить числа:

$$5 \cdot 7 = 2^6 \cdot 2^5 = 2^{(6+5)} = 2^{11} = 2^{(11 \bmod 7)} = 2^4 = 6$$

Результат совпал с тем, что мы вычислили раньше

- А теперь выполним деление:

$$6 \div 5 = 2^4 \div 2^6 = 2^{(4-6)} = 2^{(-2)} = 2^{((-2) \bmod 7)} = 2^5 = 7$$

Полученный результат тоже соответствует действительности

- Ну и для полноты картины посмотрим на возведение в степень:

$$5^2 = (\ [\![2^6)]\!]\ ^2 = 2^{(6 \cdot 2)} = 2^{12} = 2^{(12 \bmod 7)} = 2^5 = 7$$

Получился такой же результат

Расширенное конечное поле

- Подобно этому **расширенное поле GF(p^m)** порядка $q=p^m$ при $m>1$ можно ассоциировать с множеством остатков от деления полиномов над GF(p) на некоторый неприводимый полином f(x) степени m с операциями сложения и умножения по модулю f(x)
- Другими словами, поле GF(p^m) можно представить всеми полиномами над простым полем GF(p) степени не выше $m-1$ с обычным полиномиальным сложением
- Умножение же в нем выполняется в два шага – сперва как обычное умножение полиномов, но с удержанием в качестве конечного итога лишь остатка от деления полученного произведения на неприводимый полином f(x)

Поля GF(2^n)

- Для построения поля GF(2^n) используются многочлены – модули над полем GF(2), которые должны быть неприводимыми

Degree	*Irreducible Polynomials*
1	$(x + 1), (x)$
2	$(x^2 + x + 1)$
3	$(x^3 + x^2 + 1), (x^3 + x + 1)$
4	$(x^4 + x^3 + x^2 + x + 1), (x^4 + x^3 + 1), (x^4 + x + 1)$
5	$(x^5 + x^2 + 1), (x^5 + x^3 + x^2 + x + 1), (x^5 + x^4 + x^3 + x + 1),$ $(x^5 + x^4 + x^3 + x^2 + 1), (x^5 + x^4 + x^2 + x + 1)$

Пример построения полей $GF(2^n)$

- Возьмем неприводимый полином $f(x)=x^3+x+1$ и используем его для построения расширенного поля $GF(2^3) = GF(8)$
- Элементами этого поля являются элементы множества (в полиномиальной форме):

$$\{0,1,x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$$

- **Пример сложения** $a(x)=x^2+x+1$ и $b(x)=x+1$ в поле $GF(8)$: $a(x)+b(x)= x^2+x+1+x+1= x^2$
- **Пример умножения** $a(x)=x^2+x+1$ и $b(x)=x+1$:

$$(x^2+x+1)(x+1) = x^3+x^2+x^2+x+x+1 = x^3+1$$

- Разделим полученный результат на $f(x)$ с последующим удержанием остатка
- $x^3+1=q(x)f(x)+r(x)=1\cdot(x^3+x+1)+x$, следовательно

$$a(x)b(x)=(x^2+x+1)(x+1)=x$$

Пример построения полей GF(2^n)

- Выполнив умножение над всеми полиномами получим таблицу умножения заданную в общем виде:

$\times$	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
0	0	0	0	0	0	0	0	0
1	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
x	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
$x+1$	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
x^2	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	$x+1$

Арифметика вычетов

Основные тождества

$(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$

$(a - b) \bmod n = ((a \bmod n) - (b \bmod n)) \bmod n$

$(a * b) \bmod n = ((a \bmod n) * (b \bmod n)) \bmod n$

$(a * (b+c)) \bmod n = (((a*b) \bmod n) + ((a*c) \bmod n)) \bmod n$

Возведение в степень обычным умножением

$a^8 \bmod n = (a * a * a * a * a * a * a * a) \bmod n$

Возведение в степень с промежуточными привидениями

$a^8 \bmod n = ((a^2 \bmod n)^2 \bmod n)^2 \bmod n$

Для нечетных степеней, например для 25

$25 = 110011_2 \Rightarrow \quad 25 = 2^4 + 2^3 + 2^0$

$a^{25} \bmod n = (a*a^{24}) \bmod n = (a* a^8 *a^{16}) \bmod n =$

$= (a*((a^2)^2)^2 *(((a^2)^2)^2)^2) \bmod n =$

$= (a*(((a*a^2)^2)^2)^2) \bmod n$

Прием «Цепочка сложений»

$(((((((a^2 \bmod n)* a)^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 *a) \bmod n$

Порядок элементов поля

- В любом поле *GF*(*q*), будь оно простым или расширенным, можно перемножать любые операнды, в том числе *l*-кратно умножать элемент α на себя. Естественно называть такое произведение *l*-й **степенью** элемента α, обозначив его как:

$$\underbrace{\alpha\alpha...\alpha}_{l \text{ раз}} = \alpha^l.$$

$$\alpha^l\alpha^s = \underbrace{\underbrace{\alpha\alpha...\alpha}_{l \text{ раз}}\underbrace{\alpha\alpha...\alpha}_{s \text{ раз}}}_{l+s \text{ раз}} = \alpha^{l+s}$$

$$\alpha^l / \alpha^s = \underbrace{\alpha\alpha...\alpha}_{l \text{ раз}} / \underbrace{\alpha\alpha...\alpha}_{s \text{ раз}} = \underbrace{\alpha\alpha...\alpha}_{l \text{ раз}}\underbrace{\alpha^{-1}\alpha^{-1}...\alpha^{-1}}_{s \text{ раз}} = \alpha^{l-s}.$$

Порядок элементов поля

- Возьмем некоторый ненулевой элемент $\alpha \in GF(q)$ и рассмотрим его степени α^1, α^2, …, α^l, … . Поскольку все они принадлежат конечному полю $GF(q)$, в рассматриваемой последовательности рано или поздно появятся повторения, так что для некоторых l и s $(l>s)$ $\alpha^l = \alpha^s$, а значит, $\alpha^{l-s}=1$.
- Назовем минимальное натуральное число t, для которого

$$a^t = 1$$

мультипликативным порядком элемента

Порядок элементов поля

- Например, элемент 2 поля *GF*(7) имеет мультипликативный порядок $t=3$ поскольку для него $2^1=2$, $2^2=4$, $2^3=1$. Подобно этому, как легко видеть, для элемента 3 мультипликативный порядок $t=6$, для элемента 4 t =3, *для 5* $t=6$, для 6 $t=2$
- Все найденные мультипликативные порядки делят число $p-1=6$ ненулевых элементов поля, то есть являются его делителями
- В поле GF(8) число ненулевых элементов поля – простое: 8–1=7, а, значит, его делители – только числа 1 и 7. Так как единственный элементом мультипликативного порядка 1 – единица поля, все остальные ненулевые элементы имеют максимальный мультипликативный порядок, равный 7

Малая теорема Ферма

- Если P – простое число, то

$$\mathbf{X^{p-1} = 1 \pmod p}$$

для любого x простого относительно p

Следовательно

- $X^p * X^{-1} = 1 \pmod p$
- $X^p / X = 1 \pmod p \mid *X$
- $X^p = X \pmod p$

Функция Эйлера

- **Функцией Эйлера (n)** называется число положительных целых меньших n и простых относительно n, на которые n не делится без остатка

N	2	3	4	5	6	7	8	9	10	11	12
(n)	1	2	2	4	2	6	4	6	4	10	4
	1	1	1	1	1	1	1	1	1	1	1
		2	3	2	5	2	3	2	3	2	5
С				3		3	5	4	7	3	7
П				4		4	7	5	9	4	11
И						5		7		5	
С						6		8		6	
О										7	
К										8	
										9	
										10	

Теоремы для простых чисел

- **Теорема 1:** Если n = p*q, где p и q простые и не равны друг другу, то

$$\varphi(n) = (p-1)*(q-1)$$

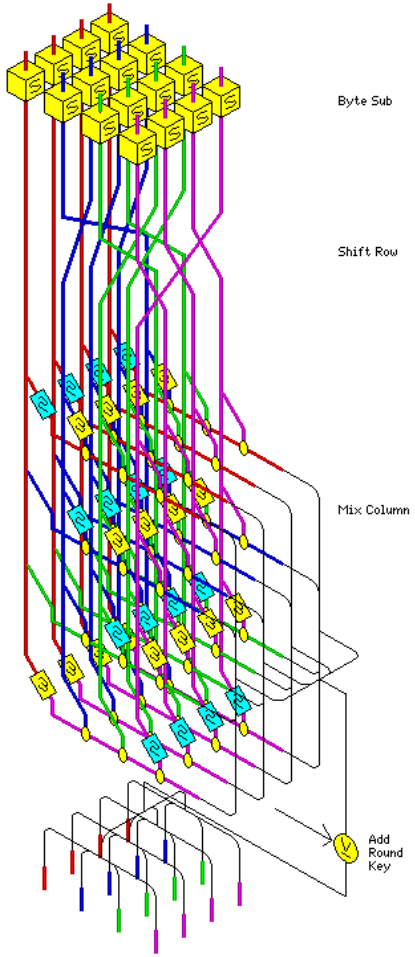
- Например, 3*5 = 15, (15) = 8
- **Теорема 2:** Если n = p*q, где p и q – простые числа и не равны друг другу, а x – простое относительно p и q, то

$$x*\varphi(n) = 1 \pmod{p}$$

- Из этого следует, что если e – простое относительно n, то достаточно легко можно подобрать целое число d, такое, что x*d = 1 (mod(n))
- Для вычисления числа d используется **расширенный алгоритм Евклида**

Advanced Encryption Standard(AES)

- Advanced Encryption Standard— симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса AES.
- Национальный институт стандартов и технологий США (NIST) опубликовал спецификацию AES 26 ноября 2001 года после пятилетнего периода, в ходе которого были созданы и оценены 15 кандидатур.

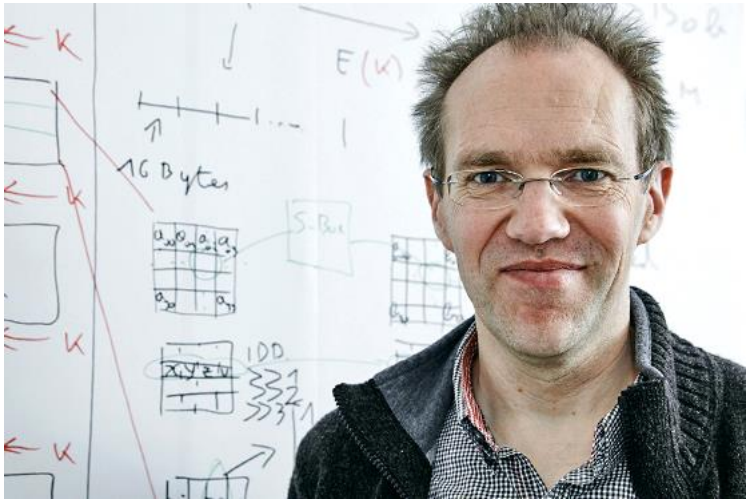


История создания алгоритма AES

"*…главным недостатком этого шифра является сложность американцам понять его …. Разработчики, Винсент Рэймен и Йоан Даймен, знают, что делают…* ",
— Bruce Schneier,Dr. Dobbs Journal (Online Edition) 2000

Винсент Рэймен

Йоан Даймен



AES/Шифрование

- Для AES длина *input* (блока входных данных) и *State* (состояния) постоянна и равна 128 бит, а длина шифроключа ***K*** составляет 128, 192, или 256 бит.
- Для обозначения выбранных длин *input*, *State* и *Cipher Key* в 32-битных словах используется нотация
  - *Nb* = 4 для input и State
  - *Nk* = 4, 6, 8 для *Cipher Key* соответственно для разных длин ключей

AES/Шифрование

- В начале шифрования input копируется в массив State по правилу

$$State[r, c] = input[r + 4c]$$

- После этого к State применяется процедура AddRoundKey() и затем State проходит через процедуру трансформации (раунд) 10, 12, или 14 раз (в зависимости от длины ключа), при этом надо учесть, что последний раунд несколько отличается от предыдущих. В итоге, после завершения последнего раунда трансформации, State копируется в output по правилу

$$output[r + 4c] = State[r, c]$$

AES/Шифрование

```
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
```

AES/SubBytes()

- Процедура SubBytes() обрабатывает каждый байт состояния, независимо производя нелинейную замену байтов используя таблицу замен (S-box). Такая операция обеспечивает нелинейность алгоритма шифрования.
- Построение S-box состоит из двух шагов.
  - Во-первых, производится взятие обратного числа в GF(2^8) (http://www.cyberforum.ru/algebra/thread551303.html)
  - Во-вторых, к каждому байту b, из которых состоит S-box, применяется следующая операция:

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

$$c = 63_{16} = 99_{10} = 01100011_2 .$$

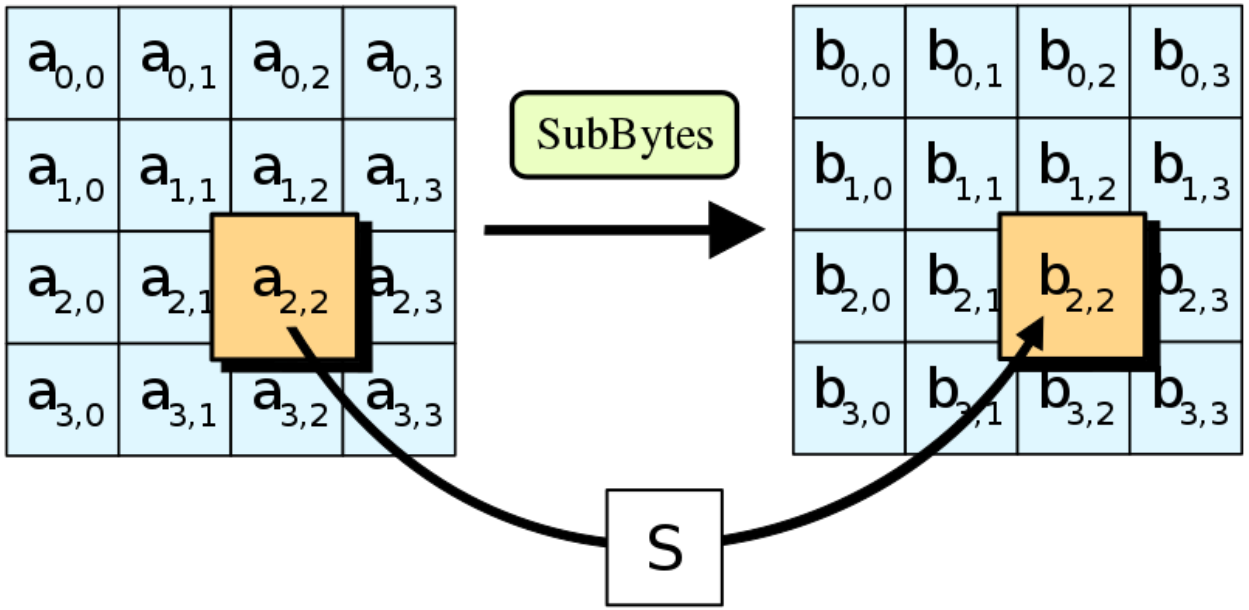
AES/SubBytes()

$$\begin{Vmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{Vmatrix} = \begin{Vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{Vmatrix} * \begin{Vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{Vmatrix} + \begin{Vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{Vmatrix}$$

AES/SubBytes() – Sbox таблица

```
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
    );
```

AES/SubBytes()

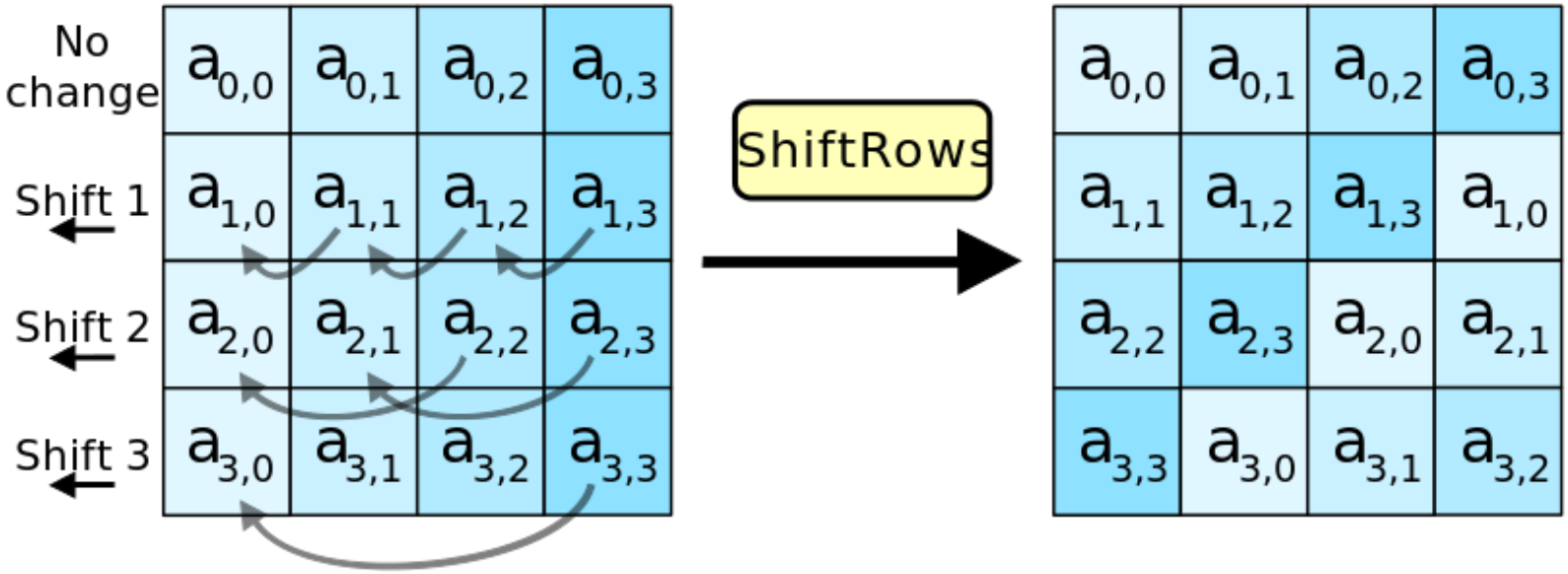


В процедуре SubBytes, каждый байт в state заменяется соответствующим элементом в фиксированной 8-битной таблице поиска, S; $b_{ij} = S(a_{ij})$.

AES/ShiftRows()

- ShiftRows работает со строками State. При этой трансформации строки состояния циклически сдвигаются на r байт по горизонтали, в зависимости от номера строки. Для нулевой строки r = 0, для первой строки r = 1 и т. д. Таким образом, каждая колонка выходного состояния после применения процедуры ShiftRows состоит из байтов из каждой колонки начального состояния.

AES/ShiftRows()



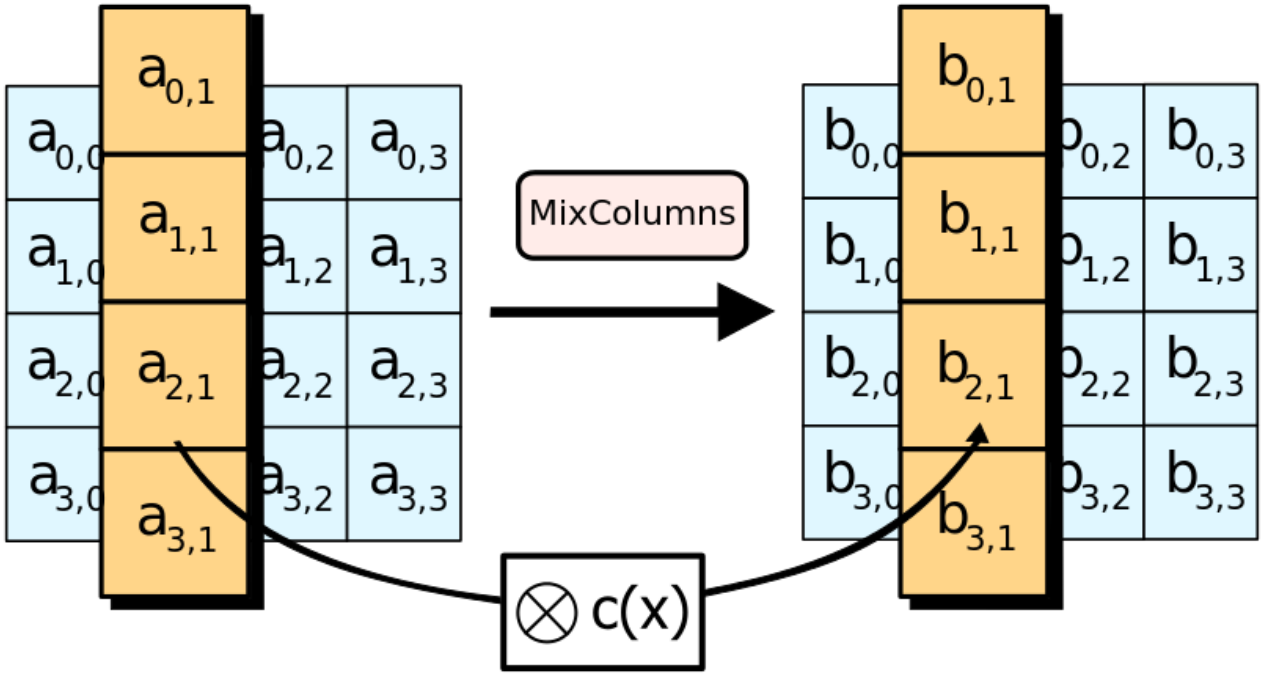
В процедуре ShiftRows, байты в каждой строке state циклически сдвигаются влево. Размер смещения байтов каждой строки зависит от её номера

AES/MixColumns()

- В процедуре MixColumns четыре байта каждой колонки State смешиваются, используя для этого обратимую линейную трансформацию. MixColumns обрабатывает состояния по колонкам, трактуя каждую из них как полином третьей степени. Над этими полиномами производится умножение GF(2^8) на фиксированный многочлен:

$$c(x) = 3x^3 + x^2 + x + 2.$$

AES/MixColumns()

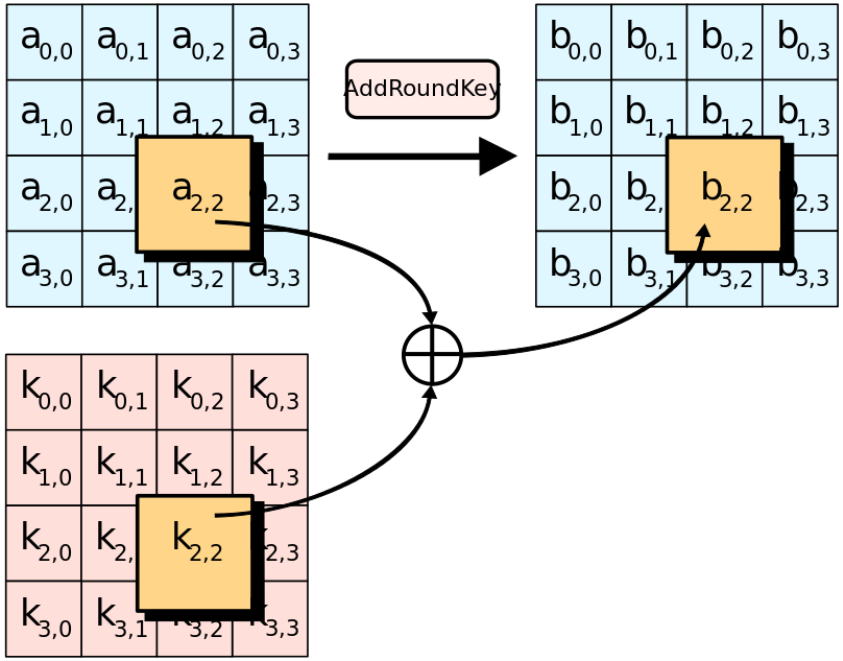


В процедуре MixColumns, каждая колонка состояния перемножается с фиксированным многочленом *c(x)*.

AES/AddRoundKey()

- В процедуре AddRoundKey, RoundKey каждого раунда объединяется со State. Для каждого раунда Roundkey получается из CipherKey с помощью процедуры KeyExpansion; каждый RoundKey такого же размера, что и State. Процедура производит побитовый XOR каждого байта State с каждым байтом RoundKey.

AES/AddRoundKey()



В процедуре AddRoundKey каждый байт состояния объединяется с RoundKey, используя операцию XOR ($\oplus$).

Алгоритм выработки ключей (Key Schedule)

- Цикловые ключи получаются из ключа шифрования посредством алгоритма выработки ключей. Он содержит два компонента: расширение ключа (Key Expansion) и выбор циклового ключа (Round Key Selection). Основополагающие принципы алгоритма выглядят следующим образом:

  - Общее число бит цикловых ключей равно длине блока, умноженной на число циклов плюс 1 (например, для длины блока 128 бит и 10 циклов требуется 1408 бит циклового ключа).
  - Ключ шифрования расширяется в Расширенный Ключ (Expanded Key).
  - Цикловые ключи берутся из Расширенного ключа следующим образом: первый цикловой ключ содержит первые Nb слов, второй - следующие Nb слов и т.д.

Алгоритм выработки ключей (Key Schedule)

```
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
```

Алгоритм выработки ключей (Key Schedule)

```
Rcon = array{
        array{0x00, 0x00, 0x00, 0x00},
        array{0x01, 0x00, 0x00, 0x00},
        array{0x02, 0x00, 0x00, 0x00},
        array{0x04, 0x00, 0x00, 0x00},
        array{0x08, 0x00, 0x00, 0x00},
        array{0x10, 0x00, 0x00, 0x00},
        array{0x20, 0x00, 0x00, 0x00},
        array{0x40, 0x00, 0x00, 0x00},
        array{0x80, 0x00, 0x00, 0x00},
        array{0x1b, 0x00, 0x00, 0x00},
        array{0x36, 0x00, 0x00, 0x00}
    };
```

AES/Дешифрование

```
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, Nb * (Nr+1) - 1])

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

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

    out = state
end
```

AES/Дешифрование

```
InvSbox = 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
        };
```