Московский физико-технический институт
(государственный университет)

Курс семинаров по предмету "Защита информации"
Эссе

Алгоритм Rijndael

Глаз Роман Сергеевич
Группа Б01-008а

Долгопрудный
2023

Содержание

1. Введение

Rijndael на настоящее время используется в стандарте шифрования привительства США по результатам проведённого конкурса *Advanced Encryption Standard*, организованного Национальным институтом стандартов и технологий США.

Потребности принятия нового стандарта возникли из-за того, что предыдущий стандарт – *Data Encryption Standard* – имел ключ длиной всего в 56 бит, что позволяло взломать шифр простым перебором ключей.

Алгоритм Rijndael стал настолько популярным, что даже производители процессоров Intel и AMD (имеющие архитектуру AMD-64) ввели аппаратную поддержку инструкций, ускоряющих работу Rijndael.

2. Принцип работы Rijndael

2.1. Краткое описание алгоритма шифрования

Пусть имеется набор входных данных I и ключ K, а B – количество 32-битных слов, из которых состоят ключ и входные данные, то есть $I = (i_1, \ldots, i_B, \ldots, i_{4B})$ и $K = (k_1, \ldots, k_V, \ldots, k_{4V})$. Возможные значения B: 4, 5, 6, 7 и 8. Возможные значения V: 4, 5, 6, 7 и 8.

Rijndael сводится к следующей формальной процедуре: получить согласно некоторым правилам шифро-текст $C = (c_1, \ldots, c_B, \ldots, c_{4B})$.

Введём понятие S (state) – текущее состояние алгоритма, которое в начале соответствует входным данным I, в процессе применения алгоритма соответствует некоторому промежуточному представлению, а после применения алгоритма – шифро-тексту C. S является матрицей размером $4 \cdot B$.

Алгоритм состоит из следующих процедур:

1. Исходные данные помещаются в текущее состояние S по следующему правилу:

$$S = \begin{Vmatrix} s_{11} & s_{12} & \ldots & s_{1B} \\ \ldots & & & \\ s_{41} & s_{42} & \ldots & s_{4B} \end{Vmatrix} = \begin{Vmatrix} i_1 & i_2 & \ldots & i_B \\ \ldots & & & \\ i_{3B+1} & i_{3B+2} & \ldots & i_{4B} \end{Vmatrix} \tag{1}$$

2. К состоянию S применяется процедура трансформации – раунд (round) – $N_R - 1$ раз, где N_R может принимать значения от 10 до 14 включительно в зависимости от длины ключа K. При минимальной длине ключа – 128 бит – используется количество раундов 10, далее при повышении длины ключа на 32 бит добалвяется дополнительно один раунд: при длине ключа 256 бит используются все 14 раундов. Полное описание раунда изложено во главе 2.2.

3. К состоянию S применяется последний раунд N_R – он немного отличается от предыдущих (подробнее об этом позже, см. главу 2.2.2).

4. Состояние S благополучно копируется в шифро-текст C:

$$C : \begin{Vmatrix} c_1 & c_2 & \ldots & c_B \\ \ldots & & & \\ c_{3B+1} & c_{3B+2} & \ldots & c_{4B} \end{Vmatrix} = \begin{Vmatrix} s_{11} & s_{12} & \ldots & s_{1B} \\ \ldots & & & \\ s_{41} & s_{42} & \ldots & s_{4B} \end{Vmatrix} \tag{2}$$

2.2. Описание процедуры трансформации round

2.2.1. Про раундовые ключи

Для каждого раунда генерируется собственный раундовый ключ W_r, представляющий собой матрицу такого же размера, что и матрица текущего состояния S, причём r – номер раунда. Имеется массив раундовых ключей размером $(N_R + 1)$: $W = (W_0, \ldots, W_{N_R})$ (будем считать нумерацию раундов с нуля).

Процедура генерации раундовых ключей W_r называется "Расширение ключа" (*KeyExpansion, подробнее в главе 2.3.5*), в процедуре используется поданный на вход шифро-ключ K.

2.2.2. Описание процедуры

Процедура round при $0 \leq r < N_R$ над текущим состоянием S состоит из следующих этапов:

1. Применение процедуры "Сложить S с ключом раунда W_r" (*AddRoundKey, подробнее в главе 2.3.1*).
2. Применение процедуры "Использовать нелинейную таблицу замен для S" (*SubBytes, подробнее в главе 2.3.2*).
3. Применение процедуры "Сдвинуть строки в S" (*ShiftRows, подробнее в главе 2.3.3*).
4. Применение процедуры "Перемножить колонки S с полиномом" (*MixColumns, подробнее в главе 2.3.4*).

Процедура round при $r = N_R$, как уже было сказано, слегка отличается от предыдущих:

1. Применение процедуры "Сложить S с ключом раунда W_{N_R}" (*AddRoundKey, подробнее в главе 2.3.1*).
2. Применение процедуры "Использовать нелинейную таблицу замен для S" (*SubBytes, подробнее в главе 2.3.2*).
3. Применение процедуры "Сдвинуть строки в S" (*ShiftRows, подробнее в главе 2.3.3*).
4. Применение процедуры "Сложить S с ключом раунда" (*AddRoundKey, подробнее в главе 2.3.1*).

2.3. Описание вспомогательных процедур

2.3.1. Процедура *AddRoundKey*

Процедура может быть описана следующим образом: имеется текущее состояние S, описываемое в виде матрицы, и раундовый ключ W_r, представляющий собой матрицу такого же размера, что и S: новое состояние получается операцией

$$S := \begin{Vmatrix} s_{11} \oplus w_{11} & s_{12} \oplus w_{12} & \ldots & s_{1B} \oplus w_{1B} \\ \ldots & & & \\ s_{41} \oplus w_{41} & s_{42} \oplus w_{42} & \ldots & s_{4B} \oplus w_{4B}. \end{Vmatrix} \tag{3}$$

2.3.2. Процедура *SubBytes*

Для процедуры *SubBytes* требуется таблица замен *Sbox*: именно благодаря этой операции обеспечивается нелинейность алгоритма шифрования. Рассмотрим подробнее алгоритм генерации таблицы замен *Sbox*.

Для начала определим $x^8+x^4+x^3+x+1$ – неприводимый многочлен в поле Галуа $GF(2^8)$. В дальнейшем в рассматриваемой процедуре будем работать только в этом поле.

Sbox – таблица размерами $16 \cdot 16$ байт, в которой изначально ij-ый байт имеет значение $16 \cdot (i-1)+j-1$. Каждый байт b матрицы *Sbox* заменяется обратным ему элементом b^{-1}. Заметим, что элементом со значением 0 не имеет обратного – присвоим ему значение 63_{16}.

Далее, выберем произвольный байт $Sbox_{ij} = c$, который преобразуем следующим образом:

$$\begin{Vmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{Vmatrix} := \begin{Vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{Vmatrix} \cdot \begin{Vmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{Vmatrix} + \begin{Vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{Vmatrix}, \tag{4}$$

где c_i - i-ый бит байта c.

Более коротким способом эту формулу можно записать как

$$c_i := c_i \oplus c_{(i+4)\ mod\ 8} \oplus c_{(i+5)\ mod\ 8} \oplus c_{(i+6)\ mod\ 8} \oplus c_{(i+7)\ mod\ 8} \oplus d_i, \tag{5}$$

где d_i – i-ый бит числа $63_{16} = 01100011_2$.

Проделывая такие же шаги для всех остальный байтов $Sbox_{ij}$ матрицы S-box, получим готовую таблицу замен.

Остаётся лишь ей воспользоваться: в матрицей текущего состояния S берём байт $s_{ij} = (\overline{x1\ x2\ x3\ x4}\ \overline{y1\ y2\ y3\ y4})$, где x_i, y_i – последовательные биты байта s_{ij}. Тогда пусть $\overline{x1\ x2\ x3\ x4}_2+1$ – номер строки в таблице замен, $\overline{y1\ y2\ y3\ y4}+1$ – номер столбца в таблице замен, сопоставляем по этим номерам новое значение s_{ij}, полученное с помощью S-box.

2.3.3. Процедура *ShiftRows*

Процедура может быть описана следующим образом: имеется текущее состояние S, а новое состояние получается путем циклического сдвига влево строк матрицы S: i-ая строка сдвигается на $(i-1)$ байт.

В матричмном виде это может быть записано как

$$S := \begin{Vmatrix} s_{11} & s_{12} & \dots & s_{1(B-1)} & s_{1B} \\ s_{22} & s_{23} & \dots & s_{2B} & s_{21} \\ s_{33} & s_{34} & \dots & s_{31} & s_{32} \\ s_{44} & s_{45} & \dots & s_{42} & s_{43} \end{Vmatrix} \tag{6}$$

Однако у алгоритма Rijndael есть одна особенность: если $B = 8$, тогда смещения нужно делать не на $0, 1, 2$ и 3 байта соответственно, а на $0, 1, 3$ и 4 байта.

2.3.4. Процедура $MixColumns$

Данная процедура считает каждый столбец матрицы состояния S как коэффициенты полинома третьей степени в поле Галуа $GF(2^8)$. Каждый столбец умножается на полином $3x^3 + x^2 + x + 2$ по модулю $x^4 + 1$. Данная процедура нужна для того, чтобы внести дополнительную диффузию в текущее состояние S.

Формально процедура может быть описана следующими формулами:

$$(s'_{1j} \cdot x^3 + s'_{2j} \cdot x^2 + s'_{3j} \cdot x^1 + s'_{4j}) = \tag{7}$$

$$= \left((s_{1j} \cdot x^3 + s_{2j} \cdot x^2 + s_{3j} \cdot x^1 + s_{4j}) \cdot (3x^3 + x^2 + x + 2)\right) (mod\ x^4 + 1), \tag{8}$$

$$s_{1j} := s'_{1j}, \ldots, s_{4j} := s'_{4j}. \tag{9}$$

2.3.5. Алгоритм генерации раундовых ключей $KeyExpansion$

Рассмотрим подробнее генерацию раундовых ключей W_r, представляющих собой матрицы того же размера, что и матрица состояния S.

С самого начала имеется шифро-ключ $K = (k_1, \ldots, k_V, \ldots, k_{4V})$, состоящий из V 32-битных слов. Тогда алгоритм заполнения раундовых ключей выглядит следующим образом:

1. Для начала определим массив 32-битных слов $w[B \cdot (N_R + 1)]$. Пусть $i = 1$.
2. Пока $i \leq V$, выполняем: $w[i] = (k_i, k_{i+1}, k_{i+2}, k_{i+3})$, $i = i + 1$.
3. Пока $i \leq B \cdot (N_R + 1)$, выполняем:
   (a) Пусть $a = w[i - 1]$.
      i. Если $i = 0\ (mod\ V)$, тогда $a = SubWord(RotWord(a)) \oplus Rcon[i/V]$.
      ii. Иначе если $V > 6$ и $i = 4\ (mod\ V)$, тогда $a = SubWord(a)$.
   (b) Выполняем: $w[i] = w[i - V]$.
   (c) Выполняем: $i = i + 1$.
4. Копируем массив $w[B \cdot (N_R + 1)]$ в каждый раундовый ключ по следующему правилу: последовательно заполняем столбцы матриц W_r, по очереди перекладывая 32-битные слова $w[i]$, последовательно по каждому раундовому ключу W_r.
5. В итоге имеем заполненные раундовые ключи W_r.

В алгоритме применялись обозначения $SubWord(a)$, $RotWord(a)$ и $Rcon[i]$ – объясним подробнее эти обозначения.

Процедура $SubWord(a)$ является применением процедуры $SubBytes(a_i)$ побайтово для 32-битного слова $a = (a_1, a_2, a_3, a_4)$.

Процедура $RotWord(a)$ делает циклический сдвиг слево байтов в 32-битном слове a, например из (a_1, a_2, a_3, a_4) получаем (a_2, a_3, a_4, a_1).

$Rcon[i]$ – массив слов, постоянный для текущего раунда, причём $Rcon[i] = (2^{i-1}, 0, 0, 0)$, 2^{i-1} считается в поле Галуа $GF(2^8)$. В алгоритме за i/V обозначает целочисленное деление чисел с отбрасыванием отстатка.

2.4. Краткое описание алгоритма дешифрования

Rijndael является симметричным алгоритмом блочного шифрования, что означает обратимость всех операций в алгоритме шифрования, разобранном выше. Действительно, операции $\oplus$, циклические сдвиги слов влево и прочие операции могут быть обращены.

В частности, вводится инвертированная таблица нелинейных замен $InvSbox$, чтобы можно было восстановить данные на входе перед применением процедуры $SubBytes$.

Процедура $MixColumns$ обращается за счёт домножения на другой многочлен в поле Галуа $GF(2^8)$.

Во всём остальном алгоритм дешифрования повторяет алгоритм шифрования, однако по раундам требуется идти с конца в начало для корректного восстановления исходных данных.

Если шифро-ключ неизвестен, то операцию $KeyExpansion$ провести не получится, а значит не будут известны раундовые ключи и расшифровать шифро-текст не получится.

3. Криптостойкость Rijndael

Алгоритм шифрования имеет ряд характеристик, свойственных надёжным алгоритмам шифрования:

1. Энтропия данных на выходе утрачивает вид энтропии на входе, из-за чего невозможно воспользоваться частотным анализом алфавита для взлома данных, а также любыми другими видами статистических зависимостей.

2. Отсутствует локальность – небольшие изменения данных на входе сильно влияют на шифро-текст на выходе при одном и том же шифро-ключе.

Однако существует ряд недостатоков:

1. Если злоумышленник узнает последний из раундовых ключей в случае использования алгоритма при $V = 4$, тогда он сможет по формулам восстановить все предыдущие раундовые ключи, а значит узнать шифро-ключ, т.е. он сможет взломать шифро-текст. Этот недостаток устраняется с увеличением V: при $V = 8$ количество возможных вариантов шифро-ключей падает с 2^{256} до 2^{128}, что всё представляет собой тяжёлую вычислительную задачу.

2. Вид атак, связанный с анализом времён выполнения математических операций, может взломать Rijndael, так как алгоритм использует вычислительные операции, время исполнения которых варьируется в зависимости от входных данных (например, операции сложения и умножения на число).

3. Вид атак, связанный с измерением энергопотребления устройства, выполняющего шифрование данных, может взломать Rijndael по тому же принципу, что и вид атак по времени: каждая операция алгоритма потребляет различное количество энергии вычислительного устройства в зависимости от входных данных

4. На данный момент считаается, что квантовые атаки способны взломать Rijndael при любом размере ключей, кроме $V = 8$, то есть при использовании 256-битных ключей.

4. Скорость шифрования/дешифрования Rijndael

Алгоритм шифрования и дешифрования использует довольно простые вычислительные действия, поэтому с точки зрения производительности на современных устройствах алгоритм выполняется довольно быстро.

Несомненным преимуществом Rijndael является то, что он используется в AES – стандартом шифрования привительства США, что привело к тому, что многие производители современных процессоров внедрили аппаратную поддержку шифрования и дешифрования – CPU AES. Подчеркнём, что в AES всегда $B = 4$ (меняться может только длина шифро-ключа), то есть AES работает с 128-битными блоками данных, которые можно представить в виде квадратной матрицы $4 \cdot 4$, что проще реализовать "в железе".

На данный скорость шифрования может достигать 1.3 байт/такт (при использовании CPU-AES), несмотря на то, что для алгоритма требуется провести довольно большое число операций. Если принять частоту процессора за 2.2 ГГц (максимальная частота в современных процессорах находится в диапазоне от 2 ГГц до 6 ГГц), то 1 Гб данных может быть зашифрован всего лишь за 0.634 секунды!

5. Режимы использования Rijndael

Как и большинство алгоритмов блочного шифрования, Rijndael может использоваться для шифрования и дешифрования данных в нескольких режимах: сам Rijndael только лишь шифрует 1 блок данных, но вот большое количество блоков может быть зашифровано разными техниками, представленными ниже. Цель режимов шифрования – сделать так, чтобы паттерны блоков в исходных данных не были зишифрованы одинаковым способом, тогда нельзя будет по шифро-тексту определить паттерны в исходных данных, что может помочь взломать их.

Возможные варианты режима шифрования:

1. ”Electronic codebook” – самый простой режим, когда блоки шифруются независимо друг от друга. Данный подход одновременно увеличивает скорость шифрования, так как блоки можно шифровать параллельно на различных единицах вычислительных устройств, но так же позволяет злоумышленнику восстановить паттерны блоков исходных данных – хотя это и не является полноценным взломом, но позволяет свести задачу взлома исходных данных ко взлому отдельных блоков.

2. ”Cipher block chaining” – режим, при котором каждый последующий блок шифруется с использованием шифротекста предыдущего блока, а именно: на вход подаётся сумма ($\oplus$) шифро-текста предыдущего блока и данных текущего блока.

3. ”Propagating cipher block chaining” – режим, при котором каждый последующий блок шифруется с использованием как шифротекста, так и исхожных данных предыдущего блока, а именно: на вход подаётся сумма ($\oplus$) шифро-текста предыдущего блока, данных предыдущего блока и данных текущего блока.

4. ”Cipher feedback” – режим, в котором шифро-текст текущего блока получается с помощью суммы ($\oplus$) ещё раз зашифрованного шифро-текста предыдущего блока

и данных текущего блока.