Rijndael – szyfr blokowy

Andrzej Chmielowiec

24 lipca 2002

1 Podstawy matematyczne

Kilka operacji w standardzie Rijndael jest zdefiniowanych na poziomie bajta, przy czym bajty reprezentują elementy ciała $GF(2^8)$. Pozostałe operacje są zdefiniowane na cztero-bajtowych słowach. W tej części wprowadzone zostaną podstawowe koncepcje matematyczne, niezbędne w dalszej części artykułu.

1.1 Ciało $GF(2^8)$

Elementy ciała skończonego mogą być reprezentowane na kilka różnych sposobów. Dla każdej potęgi liczby pierwszej istnieje dokładnie jedno ciało skończone, dlatego wszystkie reprezentacje ciała $GF(2^8)$są izomorficzne. Pomimo tej równoważności, wybór reprezentacji ma wpływ na złożoność. Dlatego wybraliśmy klasyczną reprezentację wielomianową.

Bajt b, złożony z bitów $b_7b_6b_5b_4b_3b_2b_1b_0$, jest utożsamiany z wielomianem o współczynnikach ze zbioru $\{0, 1\}$:

$$b_7x^7 + b_6x^6 + \cdots + b_1x + b_0.$$

Przykład 1 Bajt mający wartość '`57`' w zapisie szestnastkowym (co w zapisie dwójkowym ma postać `01010111`) odpowiada wielomianowi

$$x^6 + x^4 + x^2 + x + 1.$$

□

1.1.1 Dodawanie

W reprezentacji wielomianowej, suma dwóch elementów jest wielomianem o współczynnikach będących sumą modulo 2 współczynników składników.

Przykład 2 `'57'` + `'83'` = `D4'`, co w notacji wielomianowej wygląda następująco

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2.$$

W zapisie dwójkowym wygląda to następująco: `'01010111'` + `'10000011'` = `'11010100'`. Oczywiście takie dodawaniu odpowiada po prostu wykonaniu operacji EXOR (oznaczonej $\oplus$).

$\square$

Jak widać spełnione są wszystkie warunki konieczne do tego, aby wprowadzona struktura była grupą abelową. Dodawanie (EXOR) jest łączne, przemienne, dla każdego elementu istnieje element przeciwny (elementem przeciwnym do danego jest on sam) oraz istnieje element neutralny `'00'`. Warto również zauważyć, że skoro elementem przeciwnym jest on sam, to nie ma różnicy pomiędzy dodawaniem, a odejmowaniem.

1.1.2 Mnożenie

Używając reprezentacji wielomianowej, mnożenie w $GF(2^8)$musimy rozpatrywać, jako zwykłe mnożenie modulo pewien nierozkładalny wielomian stopnia 8. Dla szyfru Rijndael wybrano w tym celu wielomian

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

co można zapisać szesnastkowo jako `'11B'`.

Przykład 3 `'57'`•`'83'` = `'C1'`, lub:

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) =$$

$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1,$$

co daje nam

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \mod (x^8 + x^4 + x^3 + x + 1) =$$

$$= x^7 + x^6 + 1.$$

$\square$

Tak zdefiniowane mnożenie jest łączne i przemienne, z elementem neutralnym ('01'). Ponieważ wielomian $m(x)$ jest nierozkładalny, to dla każdego niezerowego wielomianu $b(x)$ stopnia mniejszego niż 8, możemy znaleźć (używając rozszerzonego algorytmu Euklidesa) wielomiany $a(x)$ i $c(x)$ takie, że

$$b(x)a(x) + m(x)c(x) = 1.$$

Dlatego $a(x) \bullet b(x) \mod m(x) = 1$, co oznacza, że $a(x)$ jest odwrotnością $b(x)$.

1.1.3 Mnożenie przez x

Jeśli mnożymy $b(x)$ przez wielomian x, to mamy

$$b_7x^8 + b_6x^7 + b_5x^6 + \cdots + b_1x^2 + b_0x.$$

Mnożenie $x \bullet b(x)$ polega na zredukowaniu powyższego wyniku modulo $m(x)$. Jeśli $b_7 = 0$, to redukcja nie jest potrzebna, jeśli natomiast $b_7 = 1$, to $m(x)$ musi zostać odjęte od otrzymanego rezultatu. To oznacza, że mnożenie przez x (szesnastkowo '02') może być zaimplementowane jako przesunięcie w lewo i ewentualne sksorowanie wyniku z '1B'. Oznaczmy tę operację przez `xtime(x)`. Zauważmy, że mnożenie przez wyższe potęgi można zrealizować, przez kilkukrotne zastosowanie operacji `xtime(x)`, co w połączeniu z dodawaniem daje nam mnożenie przez dowolny wielomian.

Przykład 4 '57' •'13' = 'FE'

'57' •'02' = xtime(57) = 'AE'

'57' •'04' = xtime(AE) = '47'

'57' •'08' = xtime(47) = '8E'

'57' •'10' = xtime(8E) = '07'

'57' •'13' = '57' • ('01' ⊕'02' ⊕'10') = '57' ⊕'AE' ⊕'07' = 'FE'

□

1.2 Wielomiany o współczynnikach z $GF(2^8)$

Można zdefiniować wielomiany o współczynnikach z ciała $GF(2^8)$. Postępując w ten sposób, cztero-bajtowe wektory odpowiadają wielomianom o stopniu mniejszym niż 4.

Wielomiany takie są dodawane przez dodawanie odpowiadających współczynników. Ponieważ w ciele $GF(2^8)$dodawanie polega po prostu na wykonywaniu operacji EXOR, to dodawanie wielomianów jest ksorowaniem cztero-bajtowych wektorów.

Mnożenie jest bardziej skomplikowane. Załóżmy, że mamy dwa wielomiany o współczynnikach z ciała $GF(2^8)$

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0, \quad b(x) = b_3x^3 + b_2x^2 + b_1x + b_0.$$

Wtedy produkt $c(x) = a(x)b(x)$ można wyrazić jako

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

gdzie

$$c_k = \prod_{\substack{0 \leq i,j \\ i+j=k}} a_i \bullet b_j.$$

Oczywiście, $c(x)$ nie może być dłużej reprezentowany za pomocą cztero-bajtowego wektora. Chcąc uniknąć tej niedogodności zredukujemy $c(x)$ modulo pewien wielomian stopnia 4. W szywsze Rijndael wykorzystuje się w tym celu wielomian $M(x) = x^4 + 1$. Ponieważ

$$x^j \mod x^4 + 1 = x^{j \mod 4},$$

to zredukowany produkt wielomianów $a(x)$ i $b(x)$ można wyrazić jako

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0,$$

gdzie

$$d_k = \prod_{\substack{0 \leq i,j \leq 3 \\ (i+j) \mod 4=k}} a_i \bullet b_j.$$

Operacja polegająca na mnożeniu przez pewien ustalony wielomian $a(x)$, może być zapisana jako mnożenia przez ustaloną, związaną z $a(x)$ macierz. Dla $M(x)$ określonego jak wyżej mamy

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

Mnożenie zdefiniowane w tem sposób będziemy oznaczać przez $\otimes$, zatem $d(x) = a(x) \otimes b(x)$.

Jeśli przyjmiemy, że naszym ustalonym wielomianem jest $a(x) = x$, to mamy

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

To prowadzi natomiast do wniosku, że mnożenie przez x jest niczym innym, jak tylko cyklicznym przesunięciem bajtów wektora o jeden.

2 Specyfikacja

Rijndael jest szyfrem blokowym o zmiennej długości bloku danych i klucza szyfrującego. Długości bloków i kluczy mogą być niezeleżnie od siebie ustawione na 128, 192 i 256 bitów.

2.1 Stan, klucz i liczba rund

Stan można przedstawić, jako prostokątną tablicę bajtów. Ta tablica ma cztery wiersze, natomiast liczba kolumn jest oznaczana przez **Nb** i jest równa długości bloku (w bitach) podzielonemu przez 32 (rozmiar pojedynczej kolumny).

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

Przykład stanu dla **Nb** = 6.

Klucz szyfrujący możemy przedstawić analogicznie.

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

Przykład klucza dla **Nk** = 4.

Chcąc szyfrować wiadomości możemy użyć 9 możliwych kombinacji długości klucza i stanu. Dla każdej z tych kombinacji określono liczbę rund algorytmu. Poniższa tabela opisuje tą zależność

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

Liczba rund **Nr** jako funkcja długości klucza i bloku.

2.2 Runda algorytmu

Runda algorytmu składa sie z czterech różnych przekształceń.

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

Ostatnia runda różni się od pozostałych i wygląda następująco

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

Funkcje `Round`, `ByteSub`, `ShiftRow`, `MixColumn`, `AddRoundKey` i `FinalRound` operują na wskaźnikach do tablic `State` i `RoundKey`, które zawierają odpowiednio stan i klucz.

Funkcje składowe pojedynczej rundy zostaną przedstawione w poniższych rozdziałach.

2.2.1 Funkcja ByteSub

Funkcja `ByteSub` jest nieliniowym przekształceniem każdego z bajtów stanu. Przekształcenie to jest odwracalne i składa się z dwóch operacji:

1. Obliczamy odwrotność pojedynczego bajtu, jako elementu $GF(2^8)$, przy czym '`00`' przechodzi na siebie.

2. Wykonujemy przekształcenie afiniczne nad $GF(2)$ dane jako:

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

2.2.2 Funkcja ShiftRow

To przekształcenie przesuwa cyklicznie kolejne wiersze o zadaną wartość. Wiersz zerowy nie jest przesuwany, wiersz pierwszy jest przesuwany o `C1` bajtów, drugi o `C2` bajtów i trzeci o `C3` bajtów.

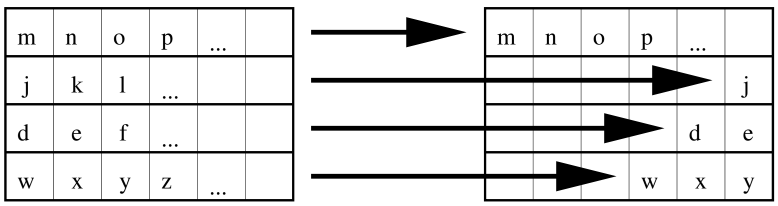
Wartości przesunięć `C1`, `C2` i `C3` zależą od długości bloku **Nb**. Odpowiednie ich wartości przedstawia poniższa tabela.

Nb	`C1`	`C2`	`C3`
4	1	2	3
6	1	2	3
8	1	3	4

Przesunięcia wierszy w zależności od długości bloku.

Poniższy rysunek ilustruje działanie funkcji `ShiftRow` na bloku o rozmiarze 6.

Rysunek 1: Funkcja `ShiftRow`.



2.2.3 Funkcja MixColumn

W tym przekształceniu kolumny tablicy stanu traktowane są jako wielomiany nad $GF(2^8)$i mnożone modulo x^4+1, przez ustalony wielomian $c(x)$ dany jako

$$c(x) = \texttt{'03'}x^3 + \texttt{'01'}x^2 + \texttt{'01'}x + \texttt{'02'}.$$

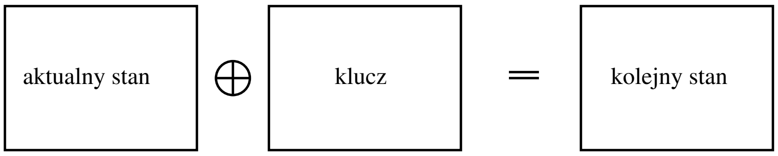
Wielomian $c(x)$ jest względnie pierwszy z x^4+1. Dlatego przekształcenie to jest odwracalne i można je przedstawić jako:

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

2.2.4 Funkcja AddRoundKey

Ta operacja polega po prostu na wykonaniu operacji EXOR, na odpowiadających sobie bajtach stanu i klucza. Klucz dla rundy powstaje na bazie pierwotnego klucza, a jego rozmiar odpowiada rozmiarowi stanu.

Rysunek 2: Funkcja `AddRounKey`.



2.3 Rozszerzanie klucza

Klucz rozszerzony, to liniowa tablica cztero-bajtowych słów oznaczona przez `W[Nb*(Nr+1)]`. Przy czym pierwsze **Nk** słów pochodzi z klucza szyfru. Wszystkie pozostałe słowa są generowane rekurencyjnie, na podstawie pierwszych **Nk**.

Jeżeli **Nk** < 8, to używamy następującego algorytmu

```
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;
  }
}
```

W tym opisie `SubByte(W)` oznacza funkcję, która została wcześniej zdefiniowana dla pojedynczej rundy szyfru. Funkcja `RotByte(W)` przesuwa cyklicznie bajty słów o 1 w lewo ($(a, b, c, d) \rightarrow (b, c, d, a)$).

Można zauważyć, że pierwsze **Nk** słów wypełnia klucz szyfrujący. Każde następne słowo `W[i]` zależy od słowa poprzedniego (`W[i-1]`) i tego, które znajduje się **Nk** pozycji przed nim (`W[i-Nk]`).

Jeżeli natomiast **Nk** > 6, to używamy następującego algorytmu

```
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;
  }
}
```

Przy czym `Rcon[i]` definiujemy jako

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

gdzie

```
RC[1]='01'
RC[i]='02'•(RC[i-1]).
```