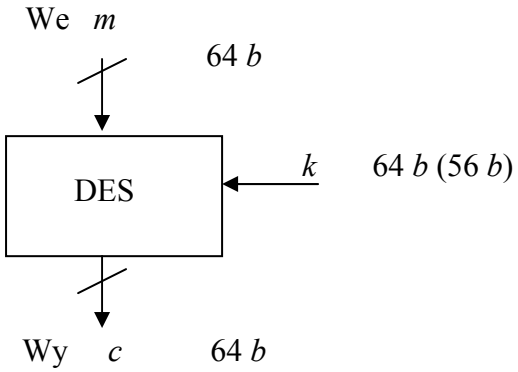
4. Szyfry z kluczem prywatnym

4.1. DES (Data Encryption Standard)

Krótki opis.

1.Szyfr DES (Data Encryption Standard) jest typowym symetrycznym szyfrem blokowym szyfruje 64 bitowe bloki (jednostki tekstu). Kryptogram jest również 64 bitowy. Klucze szyfru DES są 64 bitowe.

DES to jeden z najbardziej rozpowszechnionych algorytmów kryptograficznych, jeden ze standardów. DES ma ponad dwadzieścia lat i do niedawna skutecznie opierał się różnym atakom. Ponad 20 lat badań nad DES'em potwierdza, że konstrukcja algorytmu DES jest bardzo wyrafinowana.



Rys.4.1 Szyfr DES

2.Klucz w DES ma 64 bity ale 8 bitów z tych 64 to bity parzystości tak więc w istocie klucz w DES jest 56 bitowy. Ten sam klucz jest używany do szyfrowania i deszyfrowania (co nie znaczy jednak że szyfrowanie i deszyfrowanie to identyczny algorytm). DES ma 16 rund .

Runda Przez rundę rozumiemy powtarzający się standardowy fragment obliczeń realizowanych przez szyfr blokowy. Obliczenia te parametryzowane są tzw. podkluczami (kluczami rund) tworzonymi z klucza szyfru.

3.Szczegółowy opis algorytmu został celowo opublikowany. Chodziło o przekonanie potencjalnych użytkowników, że bezpieczeństwo metody nie tkwi w tajności jej budowy, ale w konstrukcji odpornej na kryptoanalizę. Jest to istotne, każda metoda bowiem, której szczegóły nie zostały ujawnione może zawierać w sobie tzw. ukryte drzwi czyli miejsce w algorytmie, które może być wykorzystane przez przeciwnika znającego szczegóły algorytmu, do zdobycia informacji niedostępnych w zwykłym trybie (na przykład dotyczące używanych kluczy).

4.DES jest znacznie szybszy, gdy jest zaimplementowany jako hardware, a nie jako software. Algorytm został celowo tak zaprojektowany, aby zniechęcić do implementacji software'owych, uważanych za bardziej podatne na atak (łatwiej jest się włamać do systemu i niepostrzeżenie wymienić software, niż dokonać fizycznego włamania, by wymienić hardware). Układy realizujące DES są bardzo szybkie (na przykład 1 GB/sek.); dla porównania dobry software na PC może mieć prędkość tysiące razy niższą.

5.DES szyfruje bloki złożone z 64 bitów, odpowiada to 8 słowom kodowym ASCII.. Klucze składają się również z 64 bitów przy czym 8 bitów jest bitami parzystości. Tak więc w istocie w trakcie wyboru klucza można określić jedynie 56 bitów, reszta jest generowana automatycznie. Postęp w dziedzinie technologii i związane z tym obniżenie kosztów kryptoanalizy wyzwoliły dyskusje, czy długość kluczy DES'a nie jest za mała Przy dostatecznej liczby par składających się z tekstu jawnego oraz kryptogramu utworzonego przy pomocy tego samego klucza K.

6.DES został w USA uznany za standard dla celów niemilitarnych. Został wstępnie zaprojektowany w ośrodku badawczym f-my IBM w Yorktown Heights, a następnie zmodyfikowany przez NSA (National Security Agency), rządowy organ w USA zajmujący się problemami bezpieczeństwa narodowego. National Security Agency przeprojektowała tzw. S-Boxy, które są sercem DES'a.

Wywołało to wiele podejrzeń, że NSA zna ukryte drzwi umożliwiające łamanie szyfrów generowanych za pomocą DES-a. Ponieważ DES stał się w międzyczasie standardem w zastosowaniach komercyjnych na całym świecie, dawałoby to USA olbrzymie korzyści w zakresie militarnym i gospodarczym.

Permutacja początkowa IP i końcowa IP^{-1}

Na początku bity tekstu jawnego są permutowane. Zauważmy jednak, że permutacja ta może być z łatwością zaimplementowana hardware'owo: poszczególne bity doprowadzone są za pomocą "drutów" (dokładniej połączeń w układzie VLSI) na odpowiednie miejsca. Czas obliczeń permutacji odpowiada tu jedynie czasowi, w jakim informacje dotrą po "drutach" na miejsca przeznaczenia. Z drugiej strony, implementacja software'owa wymaga długich obliczeń i każdy bit oddzielnie musi być przekopiowany na miejsce przeznaczenia.

Pod koniec szyfrowania uzyskane bity są permutowane permutacją odwrotną do początkowej. Permutacja ta jest nazywana permutacją końcową.

S-Box'y

Nazwa S-Box pochodzi od słowa angielskiego „substitution" czyli podstawienie. DES ma 8 różnych S-Box'ów oznaczanych symbolami S1, S2, S3,…,S8. Każdy S-Box realizuje pewną funkcję $S:\{0,1\}^6 \to \{0,1\}^4$ taką, że dla każdego ustalonego $x_0, x_5 \in \{0,1\}$ funkcja $S(x_0,\cdot,x_5):\{0,1\}^4 \to \{0,1\}^4$ jest różnowartościowa i na (jest permutacją). S-Box'y DES'a są zadane tabelkami por [1].

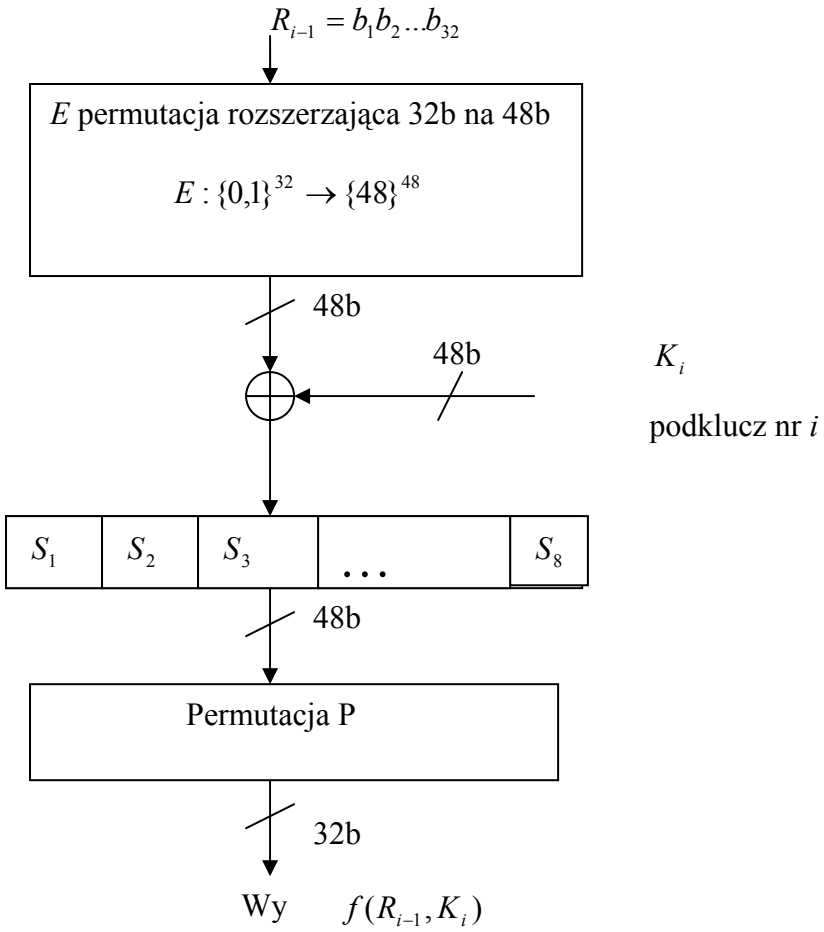
Wybór S-Box'ów czyli funkcji f opisujących S-box'y ma zasadnicze znaczenia dla jakości szyfrowania w DES'ie.

Funkcja f wewnętrzna funkcja DES'a

Sposób obliczania funkcji f pokazany jest na rys. 2. Permutacja rozszerzająca E jest stablicowana por. [1]. Poprzez permutację rozszerzającą E otrzymuje się ciąg złożony z 48 bitów. Bity te, to 32 bity R_{i-1} kopiowane na 48 pozycji, niektóre z nich dwukrotnie. Na otrzymanych 48 bitach jest dokonywana operacja XOR z odpowiadającymi im bitami podklucza K_i.

Otrzymane 48 bitów dzielone jest na 8 grup po 6 bitów. Każda grupa poddawana jest działaniu S-boksu. S-boks używany przez i-tą grupę nazywany jest Si.

Otrzymane 32 bity są na koniec permutowane w ustalony sposób permutacją P.



Rys. 4.2 Funkcja f DES'a

Runda DES'a

Dane wejściowe rundy $i + 1$ składają się z dwu ciągów 32-bitowych: L_i (pierwszych 32 bitów będących wynikiem rundy i) oraz R_i (pozostałe 32 bity uzyskane w rundzie i). Zachodzą następujące związki:

$$L_{i+1} = R_i \qquad R_{i+1} = L_i \oplus f(R_i, K_{i+1})$$

gdzie f jest opisaną funkcją, która posiada zasadnicze znaczenie dla szyfrowania. Obliczenie wartości funkcji f jest przedstawione na rys.2.

Szyfrowanie DES-em

Szyfr DES jest klasyczną siecią permutacyno-podstawieniową. Używa się w nim zarówno przestawiania. jak i podstawienia (realizowanego przez 8 tzw. S-box'ów).

Szyfrowanie i deszyfrowanie za pomocą DES'a składa się z 16 rund. W trakcie każdej rundy dokonywane są te same obliczenia, ale na wynikach obliczeń z poprzedniej rundy i specjalnym podkluczu generowanym z 64 bitowego klucza. Dodatkowo, przed pierwszą rundą i po ostatniej rundzie bity są permutowane w ustalony sposób.

Schemat blokowy algorytmu DES pokazany jest na rys.3.Obliczanie szyfrogramu z 64 bitowej wiadomości jawnej m odbywa się następująco. Najpierw poddajemy m przekształceniu IP tzw. permutacji początkowej.

Następnie przekształcamy wynikowe słowo 64 bitowe w 16 tzw. rundach „operując w opisany niżej sposób połówkami: lewą L_i i prawą R_i 64 bitowego słowa przetwarzanego. Runda o numerze i dla i=1,2,...,16 układu jest opisywana funkcją $\{0,1\}^{64} \ni L_{i-1}R_{i-1} \to L_iR_i \in \{0,1\}^{64}$ zadaną wzorem

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

gdzie i=1,2,...,16 oraz L_i oznacza lewą 32 bitową połówkę słowa 64 bitowego na wyjściu i-tej rundy a R_i prawą. Symbol K_i oznacza tzw. półklucz i-tej rundy, 48 bitowe słowo utworzone z klucza K. Po ostatniej rundzie dokonujemy przestawienia lewej i prawej strony słowa $L16$ $R16$. Dokładniej, po rundzie 16 następuje natomiast „zamiana stronami" strony prawej 32 bitowej R_{16} i lewej 32 bitowej L_{16} (co można interpretować jako pewną nieregularność wprowadzoną w rundzie 16-tej). Zatem na wejściu bloku permutacji końcowej IP^{-1} mamy $R16$ $L16$.

Deszyfrowanie szyfrem DES

Jeśli podklucze $K_1,K_2,...,K_{16}$ występują od góry w odwrotnej kolejności to znaczy $K_{16},K_{15},...,K_1$ i na wejście podamy szyfrogram, to na wyjściu układu szyfrującego otrzymamy tekst jawny m . Z układowego punktu widzenia odwrócenie kolejności podkluczy jest bardzo proste (można wykorzystać w tym celu bramki trójstanowe lub multipleksery).

Przeanalizujmy teraz co się dzieje w układzie z odwróconą kolejnością podkluczy tzn. gdy $K_{16},K_{15},...,K_1$ jeśli na wejście podany kryptogram $c = IP^{-1}(R_{16}L_{16})$

$$L_0^d R_0^d = IP \circ IP^{-1}(R_{16}L_{16}) = R_{16}L_{16}$$

Indeks d – będzie teraz oznaczał deszyfrację. Zatem

$$L_0^d = R_{16}$$
$$R_0^d = L_{16} = R_{15}$$

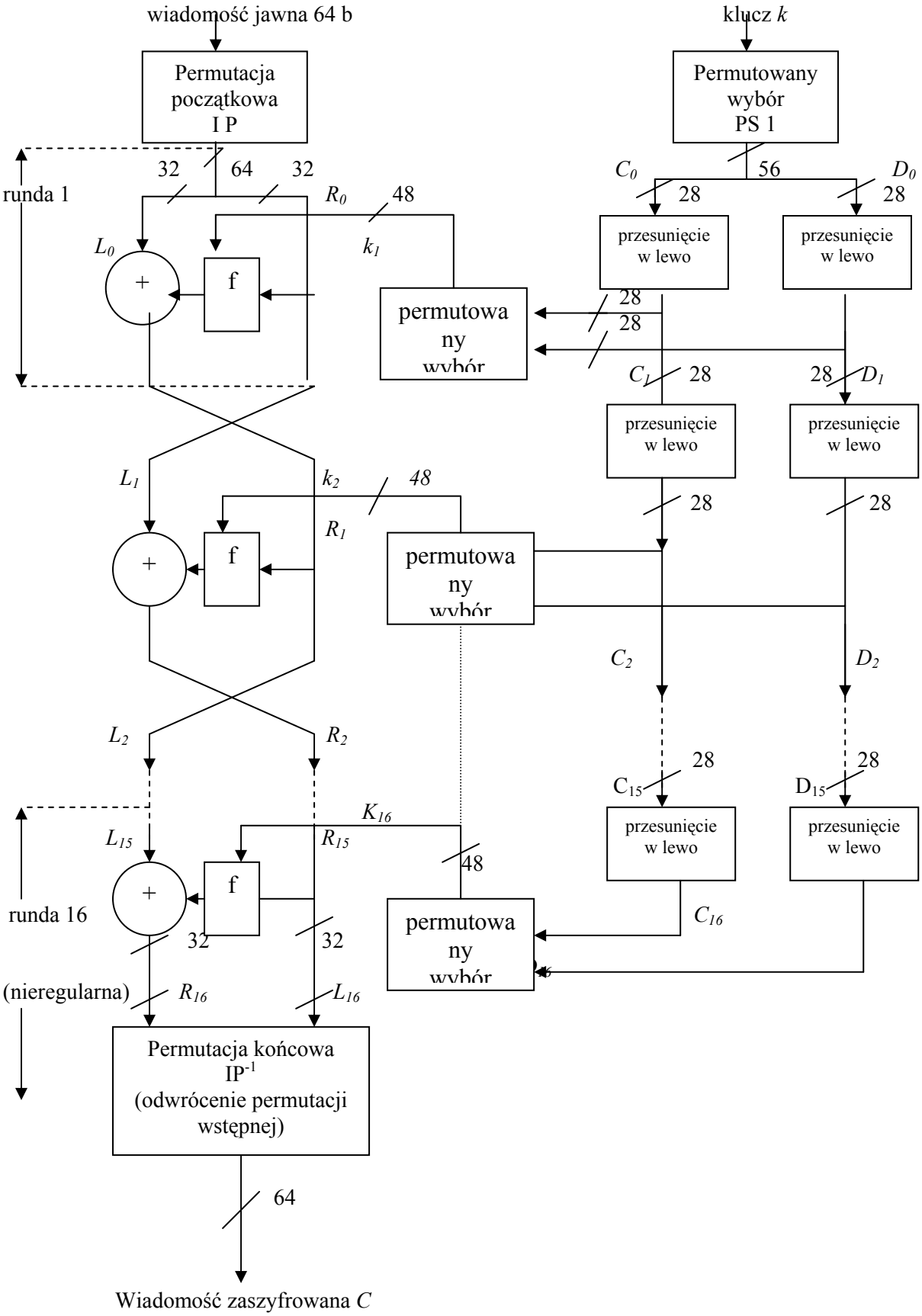
Wynik działania rundy 1 przy deszyfracji jest więc następujący:

$$L_1^d = R_0^d = L_{16} = R_{15}$$

$$R_1^d = L_0^d \oplus f(R_0^d, K_{16}) = R_{16} \oplus f(L_{16}, K_{16}) =$$
$$= L_{15} \oplus f(R_{15}, K_{16}) \oplus f(R_{15}, K_{16}) = L_{15}$$

Ostatecznie więc na wyjściu układu rundy 1 mamy $L_1^d R_1^d = R_{15}L_{15}$

Do wejścia drugiej rundy mamy doprowadzone 64 bitowe słowo $R_{15}L_{15}$ i 48 bitowy klucz K_{15}. Rozumując jak poprzednio dostajemy, że na wyjściu rundy drugiej $R_{14}L_{14}$ itd., na wyjściu rundy 16-tej mamy R_0L_0, co po przestawieniu daje L_0R_0. Wiadomość jawną otrzymamy jako $IP^{-1}(R_0L_0)$.



Rys. 4.3 Schemat blokowy algorytmu DES

Generowanie podkluczy

Dla uzyskania podkluczy K_i używanych podczas poszczególnych rund usuwamy najpierw 8 bitów parzystości zawartych w kluczu. Następnie z pozostałych 56 bitów tworzonych jest 16 podkluczy, każdy składający się z 48 bitów. Tak utworzony *i*-ty podklucz będziemy oznaczać przez K_i; będzie on używany w trakcie *i*-tej rundy. Każdy podklucz składa się ze z góry określonych bitów oryginalnego klucza - dla każdego podklucza są to inne bity i ustawione w innej kolejności. Sposób tworzenia podkluczy jest jawny i został opublikowany wraz z opisem DES-a. Mając na uwadze koszty hardware'u, wybrano taki sposób wybierania bitów podkluczy, jaki łatwo daje się zrealizować hardware'owo. Interesujące jest, iż znane metody kryptoanalizy DES'a w najbardziej istotnym momencie nie wykorzystują zależności między wartościami bitów podkluczy.

Bezpieczeństwo DES-a

Obecnie 56-bitowy klucz nie jest wystarczający dla wielu zastosowań (np. bankowych) ale użyteczność szyfru w dużym stopniu zależy od wagi informacji szyfrowanej i czasu jej życia.

W połowie 2004 roku Electronic Frontier Foundation (www.eff.com/descracker.html) ogłosiła skonstruowanie specjalizowanej maszyny łamiącej DES-a w ciągu kilku godzin. Koszt wyprodukowania takiego sprzętu jest rzędu 250 tys. dolarów.

Modyfikacje DES'a omawiane w następnym rozdziale jak np. DESX czy 3DES pozostają nadal bezpieczne.

Ataki na DES. Atak na DES zależy oczywiście od informacji wstępnych jakimi dysponujemy. Do łamania szyfru DES można zastosować atak brutalny (tzn. przeglądanie zbioru kluczy do momentu aż znajdzie się właściwy), tzw. metodę Hellmana i kryptoanalizę różnicową.

4.2 Rozszerzenia i modyfikacje DES'a

Długość klucza DES-a jest w obecnej chwili niewystarczająca. Algorytm DES modyfikowany jest na kilka sposobów tak, aby zwiększyć ilość możliwych kluczy.

Nawet jeśli sposoby te nie przynoszą znaczącej poprawy bezpieczeństwa wobec znanych metod kryptoanalizy, można mieć nadzieję, iż chronimy się w ten sposób przed atakiem poprzez ukryte drzwi, jakie, być może, istnieją dla DES-a.

DESX (inaczej DES z wybielaniem lub z „whiteningiem")

Ta wersja algorytmu DES ma 2 dodatkowe 64-bitowe klucze $k_1, k_3 \in \{0,1\}^{64}$. W sumie mamy więc 3 klucze: k_1, k_2, k_3: 56 bitowy klucz DES'a $k_2 \in \{0,1\}^{56}$ i dwa klucze 64 bitowe $k_1, k_3 \in \{0,1\}^{64}$ co daje razem 184 bitowy klucz. Dzięki temu DESX jest bardziej odporny na atak brutalny oraz kryptoanalizę różnicową i liniową .

Dla zaszyfrowania bloku *m* złożonego z 64 bitów używamy trzech kluczy k_1, k_2, k_3.

Szyfrowanie. Wiadomość jawną m (blok 64 bitowy) najpierw modyfikujemy za pomocą 64 bitowego klucza k_1 obliczając $m \oplus k_1$ (suma modulo 2 po współrzędnych). Kryptogram *c* wiadomości jawnej *m* tworzymy następująco:

$$c = k_3 \oplus DES_{k_2}(m \oplus k_1)$$

Deszyfrowanie przebiega następująco

$$m = k_1 \oplus DES_{k_2}^{-1}(c \oplus k_3)$$

Jest to zastosowanie metody zwanej wybielaniem (ang. whitening).

Zauważmy, iż w tym przypadku nawet gdy znamy tekst jawny i kryptogram, nie wiemy, jaki ciąg jest szyfrowany DES-em ani jaki jest wynik tego szyfrowania. Wydaje się to skutecznie utrudniać atak metodą systematycznego przeszukiwania, bo rozważać musimy aż 3 klucze jednocześnie. Stosując jednak ideę kryptoanalizy różnicowej, można ograniczyć ilość prób do około 2^{120}. Algorytm DESX opracowała firma RSA.

Potrójny DES (Triple DES)

Metodą często stosowaną w praktyce jest trzykrotny DES. Stosujemy w nim dwa klucze $k_1, k_2 \in \{0,1\}^{56}$ z których każdy jest zwykłym 56 bitowym kluczem DES-a.

Szyfrowanie. Szyfrowanie tekstu jawnego *m* ma następujący przebieg:

1. tekst jawny *m* szyfrowany jest DES'em z kluczem k_1

2. wynik kroku 1 jest deszyfrowany DES'em z kluczem k_2,
3. wynik kroku 2 jest powtórnie szyfrowany DES'em z kluczem k_1.

$$c = DESk_1(DESk_2^{-1}(DES_{k1}(m)))$$

Deszyfrowanie. Aby z kryptogramu *c* otrzymać z powrotem tekst jawny *m*, wystarczy oczywiście wykonać następujące kroki:

1. kryptogram deszyfrowany jest DES'em z kluczem k_1,
2. wynik kroku 1 jest szyfrowany DES'em z kluczem k_2,
3. wynik kroku 2 jest powtórnie deszyfrowany DES'em kluczem k_1.

$$m = DESk_1^{-1}(DESk_2(DES_{k1}^{-1}(c)))$$

Atak „meet in the middle".
Można się zastanawiać, dlaczego powyższa metoda aż trzykrotnie stosuje szyfrowanie (deszyfrowanie) za pomocą DES-a. Rozważmy w tym celu "dwukrotny DES", dla którego kryptogram obliczany jest wzorem

$$DES_{K_2}(DES_{K_1}(m)).$$

Dla znanej pary tekst jawny *m* i kryptogram *c* atak może przebiegać następująco (jest to tzw. atak „meet in the middle"):

1. Dla wszystkich możliwych kluczy *K* obliczamy $DES_K(m)$, wyniki zapisujemy w tabeli.

2. Dla wszystkich możliwych kluczy *K*' obliczamy $DES_{K'}^{-1}(P)$, wyniki zapisujemy w tabeli.

3.Porównujemy zawartośći obu tabel. Ten sam wynik w obu tabelach wskazuje na parę kluczy K, K', dla których

$$c = DES_{K'}(DES_K(m)).$$

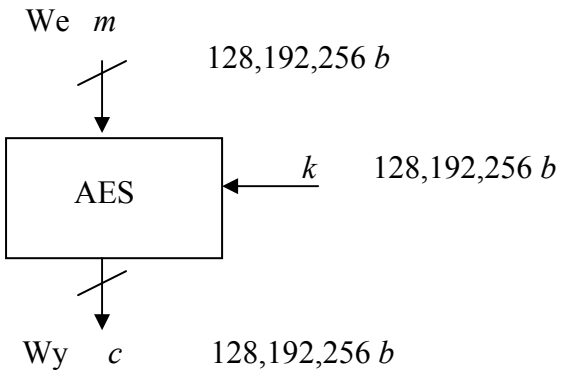
Czy para taka jest właściwa, można sprawdzić testując ją na innej parze tekst jawny-kryptogram.

Zauważmy, iż powyższa metoda wykonuje jedynie $2 \cdot 2^{56}$ szyfrowań i deszyfrowań w krokach 1 i 2. Nie jest to znacząco więcej niż 2^{56} szyfrowań w przypadku systematycznego przeszukiwania w odniesieniu do zwykłego DES'a. Oczywiście, realizacja kroku 3 wymaga operacji przeszukiwania olbrzymich tabel, co może być praktycznie niewykonalne. Niemniej jednak lepiej jest stosować trzykrotny DES, dla którego podobny atak wymaga znacznie więcej operacji.

4.3. Szyfr AES

Szyfr AES (od ang. Advanced Encryption Standard) to inaczej szyfr Rijndael.

1. Krótki opis szyfru. Jest to szyfr blokowy. Nie jest i nie będzie (jak twierdzą twórcy tego szyfru) opatentowany. Wiadomość jawna m to słowo binarne o długości 128,192 lub 256 bitów.



Rys. 4.4 Szyfr AES

Wiadomość zaszyfrowana c to słowo binarne o długości 128,192 lub 256 bitów. Wiadomość jawna i zaszyfrowana są zawsze tej samej długości.

Klucz k to słowo binarne o jednej z 3 długości: 128,192 lub 256 bitów. Klucz nie musi być tej samej długości co wiadomość jawna. Długość klucza k i długość wiadomości jawnej m są niezależne.

2. Podstawowa struktura algebraiczna AES'a , ciało $GF(2^8)$.

Podstawowym pojęciem, które posłuży nam do zdefiniowania AES'a jest ciało skończone $GF(2^8)$. Punktem wyjścia jest więc dobrze znane Czytelnikowi dwuelementowe ciało $(Z_2,\oplus_2,\otimes_2)$. Dodawanie w Z_2 to suma modulo 2, czyli EXOR. Mnożenie w Z_2 to iloczyn modulo 2, który okazuje się być zwykłym iloczynem boolowskim.

Przypomnimy jak konstruuje się ciało $GF(2^8)$. Rozważamy najpierw wszystkie wielomiany o współczynnikach w ciele $Z_2 = \{0,1\}$ (ciało Z_2 oznaczane jest też symbolem $GF(2)$ lub F_2). Zbiór wszystkich takich wielomianów tworzy pierścień (przemienny z 1) oznaczany symbolem $Z_2[x]$.

Z pierścienia $Z_2[x]$ tworzymy w standardowy sposób ciało oznaczone symbolem $GF(2^8)$, wybierając dowolny wielomian nierozkładalny 8-go stopnia i tworząc pierścień ilorazowy. W przypadku AES'a wybranym wielomianem nierozkładalnym jest wielomian:

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

Każdy współczynnik wielomianu z pierścienia $Z_2[x]$ jest bitem wiążemy więc z każdym wielomianem z $Z_2[x]$ słowo binarne współczynników (wygodnie jest je zapisywać w zapisie heksadecymalnym). Na przykład wielomian $m(x) = x^8 + x^4 + x^3 + x + 1$ to w zapisie heksadecymalnym '11*B*'.

Elementami $GF(2^8)$ są reszty z dzielenia wielomianów z pierścienia $Z_2[x]$ przez wielomian $m(x) = x^8 + x^4 + x^3 + x + 1$. Wszystkie elementy $GF(2^8)$ są więc wielomianami co najwyżej 7 stopnia. Z kolei wielomian 7-go stopnia $b_7x^{7+}b_6x^6 + b_5x^5 + ... + b_1x + b_0$, gdzie $b_i \in Z_2 = \{0,1\}$ można utożsamiać z bajtem $b_7b_6b_5b_4...b_1b_0$. Łatwo widać, że tyle jest wielomianów z $GF(2^8)$ ile bajtów.

Przykład. Przykład zwartego opisu wielomianów o współczynnikach w Z_2:
'57' – to zapis heksadecymalny bajtu
'01010111' – to zapis binarny bajtu
Ten bajt odpowiada wielomianowi: $x^6 + x^4 + x^2 + x + 1$
■

Dodawanie. Dodawanie w $GF(2^8)$: to suma modulo 2 po współrzędnych.

Przykład. '57'+'83'='*D*4' lub inaczej:

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

lub binarnie: '01010111'+'10000011'='11010100'

Mnożenie.
Mnożymy „normalnie" wielomiany jak wielomiany, a potem dzielimy przez $m(x) = x^8 + x^4 + x^3 + x + 1$, reszta z tego dzielenia, to wyniki mnożenia w $GF(2^8)$. Tak zdefiniowane mnożenie jest łączne i przemienne. Element $'01' \in GF(2^8)$ jest elementem neutralnym tego mnożenia.

Uwaga. Jeśli mamy wielomian $b(x) \in GF(2^8)$, to z rozszerzonego algorytmu Euklidesa wynika, że znajdziemy takie wielomiany $a(x)$ i $d(x)$, gdzie $a(x), d(x) \in GF(2^8)$ że:

$$a(x)b(x) + m(x)d(x) = 1$$

co można zapisać jako:

$$a(x) \bullet b(x) \equiv 1 (\mathrm{mod}\, m(x))$$

Warto podkreślić, że rozszerzony algorytm Euklidesa działa w każdym tzw. pierścieniu euklidesowym, w szczególności w pierścieniu liczb całkowitych Z i w pierścieniu wielomianów $K[x]$ o współczynnikach z dowolnego ciała K.

Przykład '57'•'83'='$C1$'**.** Dokładniej:

$$(x^6+x^4+x^2+x+1)(x^7+x+1)=x^{13}+x^{11}+x^9+x^8+x^7+x^7+x^5+x^3+x+x^6+x^4+x^2+x+1=$$

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

$$(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1)(\mathrm{mod}(x^8+x^4+x^3+x+1)=x^7+x^6+1$$

Mnożenie przez *x*.

Mnożenie przez x jest proste w $GF(2^8)$ i stanowi bardzo użyteczną operację. Niech $b(x)=b_7x^7+b_6x^6+...+b_0$. Chcemy obliczyć (*)

$$b(x)\cdot x=b_7x^8+b_6x^7+...+b_0x \qquad (*)$$

Jeśli $b_7=0$, to wynik uzyskujemy przez przesuwanie w lewo.

Jeśli $b_7=1$, to odejmujemy (czyli dodajemy EXOR'em) wielomian $m(x)$ od wielomianu (*). co jest równoważne dzieleniu wielomianu $b(x)\cdot x=b_7x^8+b_6x^7+...+b_0x$ przez $m(x)=x^8+x^4+x^3+x+1$

Mnożenie przez x^n. Wielokrotnie *n*-razy stosujemy ten sam wyżej opisany algorytm.

Mnożenie przez wielomian.

Mnożymy przez odpowiednie x^i a wyniki dodajemy.

Oznaczamy algorytm mnożenia $a=a_7a_6...a_0$ przez x jako $b=xtime(a)$

Przykład

'57'•'13'='FE'

Obliczamy kolejne potęgi:

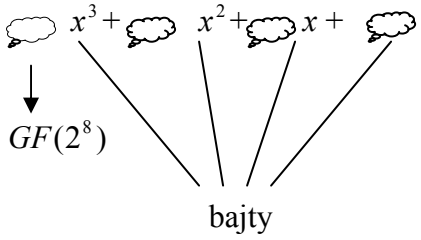
'57'•'02'= $xtime(57)$ ='AE'
'57'•'04'= $xtime(AE)$ ='47'
'57'•'08'= $xtime(47)$ ='8E'
'57'•'10'= $xtime(8E0$ ='07'

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

3. Wielomiany o współczynnikach w $GF(2^8)$. Drugim podstawowym pojęciem , które posłuży nam do zdefiniowania AES'a są wielomiany o współczynnikach w ciele $GF(2^8)$ (wszystkich różnych współczynników jest więc 256). Pierścień wielomianów o współczynnikach w $GF(2^8)$ oznaczamy symbolem $GF(2^8)[x]$.

Rozważamy 4 bajtowy ciąg, wektor 32 bitowy, słowo 32 bitowe.

Wprowadzamy więc NOWE OBIEKTY odpowiadające wielomianom stopnia co najwyżej 3



Dokładniej będą to elementy pierścienia ilorazowego $GF(2^8)[x]/(M)$, gdzie M jest pewnym wielomianem stopnia 4. Konkretnie jako wielomian M wybieramy wielomian x^4+1 i tworzymy pierścień wielorazowy $GF(2^8)[x]/(x^4+1)$.

Elementami $GF(2^8)[x]/(x^4+1)$ będą wielomiany stopnia co najwyżej 3, które możemy reprezentować jako słowa 4 bajtowe (4 współczynniki z $GF(2^8)$) co daje słowo 32 bitowe na wielomian.

Dodawanie – normalne jak wielomianów, ale w naszym przypadku to EXOR'owanie słów 32 bitowych.

$$(a_3x^3+a_2x^2+a_1x+a_0)+(b_3x^3+b_2x^2+b_1x+b_0)=c_3x^3+c_2x^2+c_1x+c_0$$

32 bitowe słowo wynikowe reprezentujące wielomian $c_3x^3+c_2x^2+c_1x+c_0$ otrzymujemy po exorowaniu po współrzędnych słów 32 bitowych reprezentujących wielomian $a_3x^3+a_2x^2+a_1x+a_0$ i wielomian $b_3x^3+b_2x^2+b_1x+b_0$

Mnożenie – Wprowadzamy wielomian $M(x)=x^4+1$, ale uwaga wielomian ten nie jest nierozkładalny nad ciałem $GF(2^8)$. Wielomiany mnożymy normalnie jak wielomiany i na koniec dzielimy przez $M(x)$, reszta z tego dzielenia to wynik mnożenia.

Chwyt ułatwiający wykonywanie działań w $GF(2^8)[x]/(M)$

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

Jeśli to uwzględnimy tę równość to dostajemy

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

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

$$d(x) = a(x) \otimes b(x)$$

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

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

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

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

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

Co odpowiada operacji macierzowej:

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

Mamy więc ogólny wzór na mnożenie elementów w pierścieniu ilorazowym $GF(2^8)[x]/(M)$. Łatwo zauważyć, że macierz tworzymy przesuwając cyklicznie 3 razy pierwszą kolumnę o 1 pozycję.

Mnożenie przez *x*: Mamy $b(x) = b_3x^4 + b_2x^3 + b_1x^2 + b_0x$ i chcemy obliczyć $x \otimes b(x)$. Mnożymy wielomiany normalnie, w końcu dzielimy przez $x^4 + 1$, co daje:

$$b_2x^3 + b_1x^2 + b_0x + b_3 \quad \longleftarrow \quad \text{rotacja}$$

co jest równoważne mnożeniu macierzy:

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

Wykażemy teraz fakt, z którego skorzystaliśmy ale sformułujemy go nieco ogólniej.

Fakt. Rozważmy pierścień wielomianów $GF(2^k)[x]$ (gdzie k jest dowolną ustaloną liczbą naturalną) o współczynnikach w ciele $GF(2^k)$. Jeśli x^n jest dla $n \in N$ wielomianem z $GF(2^k)[x]$, to mamy:

$$x^n (\mathrm{mod}(x^4+1)) = x^{n(\mathrm{mod}\,4)} \qquad (*)$$

Uwaga. Powyższą równość wykorzystujemy do przyspieszenia obliczeń w pierścieniu $GF(2^k)[x]/(x^4+1)$. Warto przypomnieć, że $GF(2^8)$, $GF(2^8)[x]$ i $GF(2^8)[x]/(x^4+1)$ są zasadniczymi strukturami algebraicznymi wykorzystywanymi w szyfrze symetrycznym AES (Rijndael). Elementy $GF(2^8)$ reprezentowane są w komputerze jako bajty a elementy $GF(2^8)[x]/(x^4+1)$ jako słowa 32 bitowe.

Dowód 1.W ciele $Z_2 = \{0,1\}$ dodawanie jest zwykłą sumą modulo 2 (oznaczamy je symbolem "$\oplus$" i układowo realizujemy za pomocą bramki EXOR). I co zaskakujące również odejmowanie w Z_2 jest sumą modulo 2, bo mamy $1 \oplus 1 = 0$ i $0 \oplus 0 = 0$ więc $-a = a$ dla $a \in Z_2$ oraz $a - b = a \oplus b$ dla $a, b \in Z_2$, gdzie - jest odejmowaniem modulo 2 w Z_2.

2.W ciele $GF(2^k)$, (którego elementami są słowa binarne o długości k) definiujemy działanie dodawania standardowo jako sumę wielomianów ale w naszej sytuacji jest to jednocześnie suma modulo 2 po współrzędnych tzn. jeśli $a = (a_1, a_2, ..., a_k) \in GF(2^k)$, gdzie $a_i \in \{0,1\}$ oraz $b = (b_1, b_2, ..., b_k) \in GF(2^k)$, gdzie $b_i \in \{0,1\}$ to:

$$a + b = (a_1 \oplus b_1, a_2 \oplus b_2, ..., a_n \oplus b_k)$$

oraz

$$a - b = (a_1 - b_1, a_2 - b_2, ..., a_k - b_k) = (a_1 \oplus b_1, a_2 \oplus b_2, ..., a_k \oplus b_k)$$

3.Oczywiście, jeśli $n < 4$, to dowodzony wzór (*) jest prawdziwy. Dla $n \geq 4$ istnieje takie $q \in N$, że $n = q \cdot 4 + r$ i $0 \leq r < 4$, oczywiście $r = n(\mathrm{mod}\,4)$.

Zauważmy jak przebiega dzielenie wielomianu x^n dla $n \geq 4$. Uwzględniając uwagi z p.2 (że „odejmowanie to dodawanie") mamy:

$x^{n-4} + x^{n-8} + \ldots + x^{n-q\cdot 4}$

x^n $\quad : x^4 + 1$

$x^n + x^{n-4}$

x^{n-4}

$x^{n-4} + x^{n-8}$

x^{n-8}

.

.

.

x^r

a więc istotnie: $x^n (\mathrm{mod}(x^4 + 1)) = x^{n(\mathrm{mod}\,4)}$.

a więc istotnie: $x^n (\mathrm{mod}(x^4 + 1)) = x^{n(\mathrm{mod}\,4)}$.

■

Fakt. Rozważmy pierścień wielomianów $GF(2^k)[x]$ (gdzie k jest dowolną ustaloną liczbą naturalną) o współczynnikach w ciele $GF(2^k)$. Wykazać, że jeśli x^n jest dla $n \in N$ wielomianem z $GF(2^k)[x]$ to mamy dla każdego naturalnego $s \geq 2$:

$$x^n (\mathrm{mod}(x^s + 1)) = x^{n(\mathrm{mod}\,s)} \qquad (*)$$

Dowód. Dowód jest analogiczny do rozwiązania zadania 1. Punkty1. i 2. nie ulegają zmianie. W ostatnim 3-cim punkcie mamy:

3. Jeśli $n < s$, to dowodzony wzór (*) jest oczywiście prawdziwy. Dla $n \geq s$ istnieje takie $q \in N$, że $n = q \cdot s + r$ i $0 \leq r < 4$, oczywiście $r = n(\mathrm{mod}\,s)$.

Zauważmy jak przebiega dzielenie wielomianu x^n dla $n \geq s$. Uwzględniając uwagi z p.2 (że „odejmowanie to dodawanie") mamy:

$$
\begin{array}{l|l}
x^{n-s}+x^{n-s}+\ldots+x^{n-q\cdot s} & \\
\hline
x^{n} & :x^{s}+1 \\
x^{n}+x^{n-s} & \\
\hline
\quad x^{n-s} & \\
\quad x^{n-s}+x^{n-2\cdot s} & \\
\quad \_\_\_\_\_ & \\
\qquad x^{n-2\cdot s} & \\
\qquad\quad . & \\
\qquad\quad . & \\
\qquad\quad . & \\
\qquad\quad \_\_\_\_\_ & \\
\qquad\qquad x^{r} &
\end{array}
$$

a więc istotnie: $x^{n}(\mathrm{mod}(x^{s}+1))=x^{n(\mathrm{mod}\,s)}$.

■

Fakt. Mnożenie dwóch wielomianów $a(x)=a_{s-1}x^{s-1}+a_{s-2}x^{s-2}+\ldots+a_1x+a_0$ i $b(x)=b_{s-1}x^{s-1}+b_{s-2}x^{s-2}+\ldots+b_1x+b_0$ (gdzie s jest liczbą naturalną $s\geq 2$) w pierścieniu ilorazowym $GF(2^k)[x]/(x^s+1)$ (gdzie k jest dowolną ustaloną liczbą naturalną) sprowadza się do następującego mnożenia macierzy przez wektor

$$
\begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{s-1} \end{bmatrix} = \begin{bmatrix} a_0 & a_{s-1} & \ldots & a_1 \\ a_1 & a_0 & \cdots & a_2 \\ \vdots & \vdots & \ldots & \vdots \\ a_{s-1} & a_{s-2} & \cdots & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{s-1} \end{bmatrix}
$$

gdzie $d(x)=d_{s-1}x^{s-1}+d_{s-2}x^{s-2}+\ldots+d_1x+d_0$ jest wynikiem mnożenia.

Uwaga. Łatwo zauważyć, że macierz tworzymy przesuwając cyklicznie pierwszą kolumnę.

Stan algorytmu

Stan algorytmu AES to obiekt na którym pracuje algorytm. Jest to macierz o wymiarach 4xN_b (zawsze 4 wiersze na N_b kolumn, gdzie N_b jest zależne od długości słowa wejściowego *m*), której elementami są elementy ciała $GF(2^8)$ a więc bajty.

Z kolei kolumny tej macierzy traktowane są jako elementy pierścienia $GF(2^8)[x]/(x^4+1)$. Początkowy stan tworzy słowo wejściowe – wiadomość jawna m. Wiadomość jawna składa się z podbloków 32 bitowych (stąd oznaczenie N_b „number of blocks") i ma zawsze długość

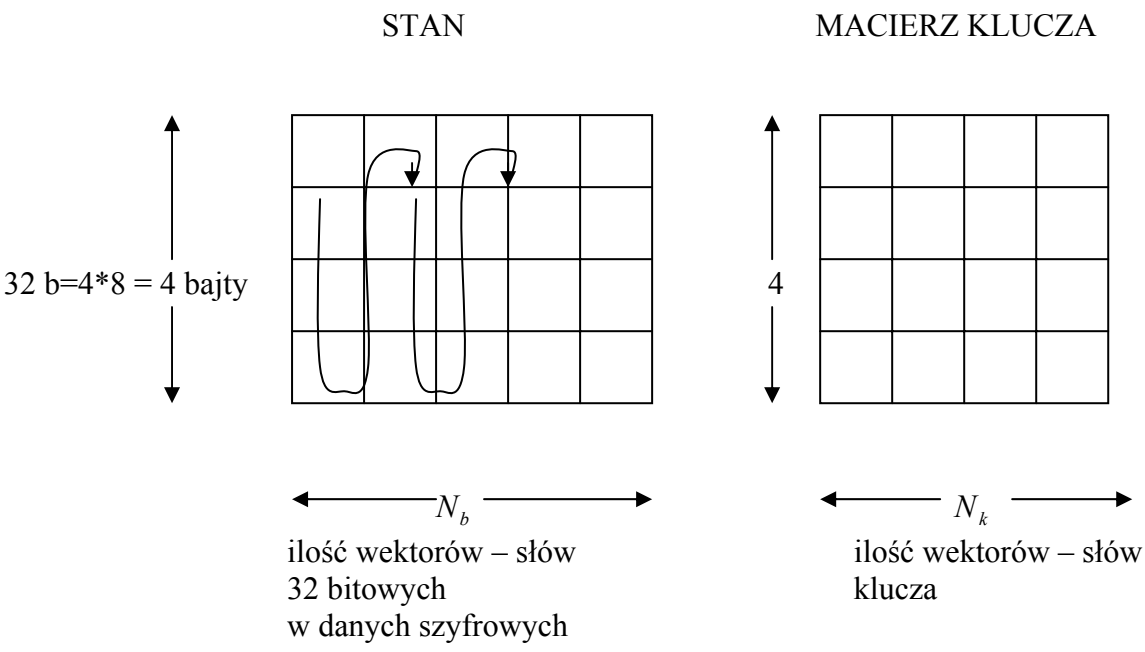
$N_b * 32b$. Tworząc stan początkowy macierzy stanu po prostu umieszczamy kolejne słowa 32 bitowe tworzące wiadomość jawną *m* w kolumnach macierzy stanu.

Na przykład dla słowa wejściowego 192 bitowego mamy $N_b = 6$ i tworzymy taką macierz : 4 wiersze (zawsze) i 6 kolumn.

$$\underbrace{\begin{matrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \end{matrix}}_{N_b}$$

Klucz, z którego tworzymy podklucze – też będzie w algorytmie macierzą złożoną z kolejnych 32 bitowych słów klucza umieszczonych w kolumnach:

$$\underbrace{\begin{matrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{matrix}}_{Nk}$$



Rys. 4.5 Stan algorytmu AES i macierz klucza

N_r	$N_b = 4$	$N_b = 6$	$N_b = 8$
$N_k = 4$	10	12	14
$N_k = 6$	12	12	14
$N_k = 8$	14	14	14

Rys. 4.6 Tabelka podająca liczbę N_r rund algorytmu AES w funkcji długości bloku wiadomości jawnej i długości klucza ($N_k \times 32$ =długość klucza, $N_b \times 32 =$ długość wiadomości jawnej)

Runda.

Runda składa się z czterech różnych przekształceń **stanu**, 4 procedur co można zapisać w pseudo *C* jako funkcję stanu „State" i klucza rundy „RoundKey" następująco:

```
Round (State, Round Key)
{
Byte Sub (State);
Shift Row (State);
Mix Column (State);
Add Round Key (State, Round Key)
}
```

Runda końcowa algorytmu AES jest nieco inna. Pozbawiona jest procedury MixColumn

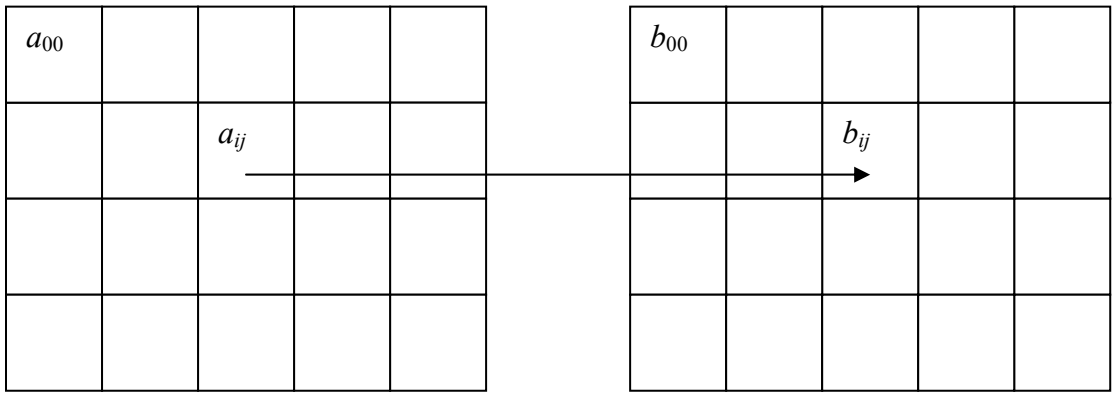
```
Final Round (State, Round Key)          // bez Mix Column (State)
{
Byte Sub (State);
Shift Row (State);
Add Round Key (State, Round Key);
}
```

Omówimy teraz po kolei 4 procedury występujące w rundzie.

1.Przekształcenie "podstawianie bajtów" (ang. Byte Sub transformation)

Działa ono na macierzy stanu, „po współrzędnych" por. rys. 3. Każdy bajt zmieniamy niezależnie za pomocą tzw. S-box'a – przekształcenia działającego na pojedynczy bajt.

stan wejściowy stan wyjściowy



$$a_{ij} \to b_{ij}$$

Rys. 4.7. Działanie procedury Byte Sub

Przekształcenie realizowane przez *S*-box $f : GF(2^8) \to GF(2^8)$ zdefiniowane jest następująco:

1.Bierzemy odwrotność $x \in GF(2^8)$ w ciele $GF(2^8)$ przy czym jeśli $x = '00'$, to $f('00') = '00'$

2.Do wyniku stosujemy następujące przekształcenie afiniczne $g : Z_2^8 \to Z_2^8$ (nad ciałem $GF(2)$)

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

Tak zdefiniowane przekształcenie $f : GF(2^8) \to GF(2^8)$ jest właśnie S-box'em. Jest to przekształcenie równowartościowe i „na".

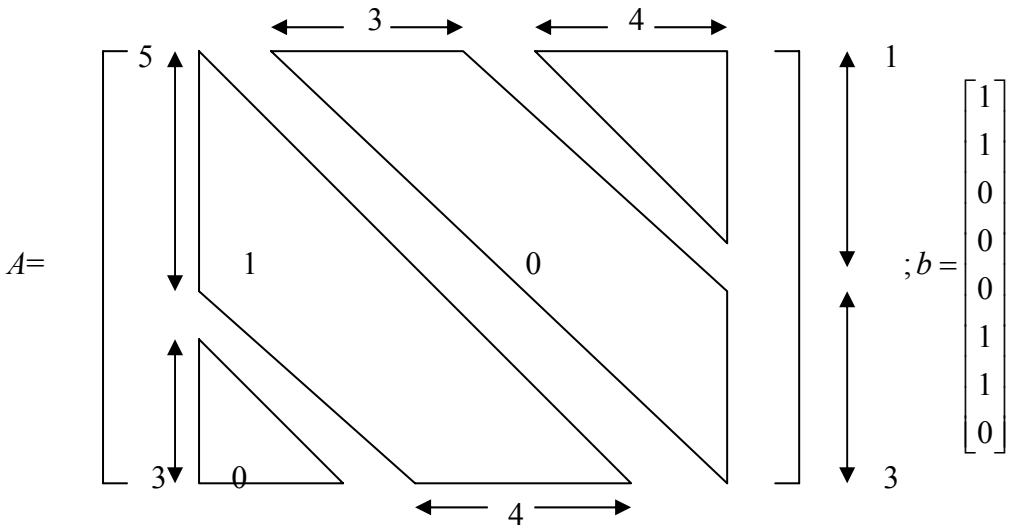
Jeśli więc stosujemy S-box do elementu a_{ij} to obliczamy najpierw a_{ij}^{-1} a następnie mnożymy tak otrzymany wektor (bajt) przez powyżej zdefiniowaną macierz zerojedynkową, do wyniku dodajemy wektor stały i w ten sposób otrzymujemy wektor wynikowy.

Jeśli niezależnie do każdej współrzędnej macierzy zastosujemy S-box to tak zdefiniowane przekształcenie macierzy stanu oznaczamy jako ByteSub (State). Jest to przekształcenie równowartościowe i „na".

Łatwo znaleźć przekształcenie odwrotne do ByteSub (State). W tym celu wystarczy znaleźć przekształcenie odwrotne do S-box'a. Można to zrobić następująco:

1) Bierzemy odwrotność macierzy A czyli A^{-1} i obliczamy $A^{-1} \cdot (y - b)$

2) A następnie obliczamy odwrotność elementu (bajtu) $A^{-1} \cdot (y - b)$ w ciele $GF(2^8)$.



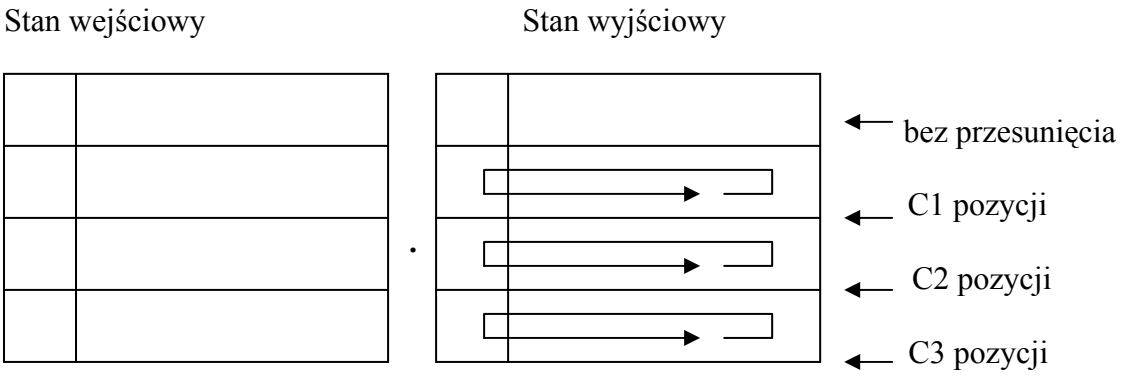
Rys. 4.8 Struktura macierzy A występującej w definicji przekształcenia ByteSub (State)

2.Przekształcenie „przesunięcia wierszy"

Przesuwamy (ściślej jest to tzw. rotacja) bajty z wierszy macierzy stanu według poniższej tabelki, gdzie N_b jest szerokością słowa wejściowego w 32 bitowych słowach.

N_b	C 1	C 2	C 3
4	1	2	3
6	1	2	3
8	1	3	4

C Ø jest zawsze równe 0.



Tak zdefiniowane przekształcenie ShiftRow(State) jest równowartościowe i „na". Przekształcenie odwrotne do przesunięcia wierszy to jest analogicznie zdefiniowane przekształcenie, ale z rotacją w lewo o odpowiednią liczbę pozycji.

3.Przekształcenie „mieszanie kolumn".

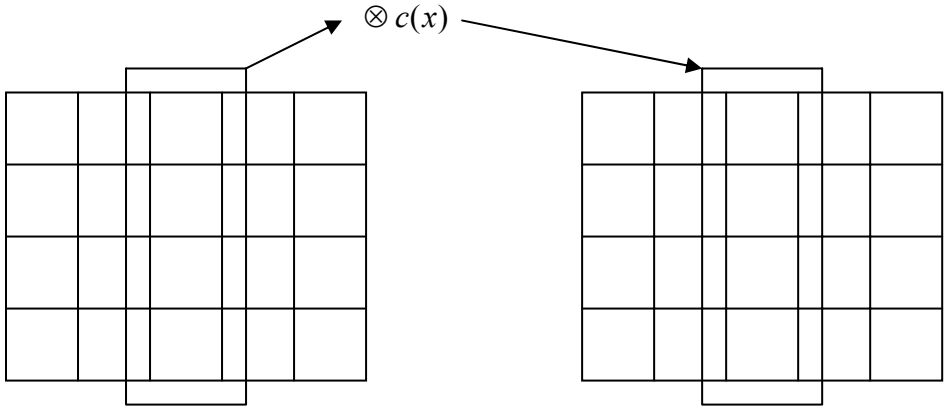
Przekształcenie to będziemy oznaczać symbolem MixColumn (State). Działa ono tak. Kolumny macierzy stanu traktujemy jak wielomiany i mnożymy modulo x^4+1 przez ustalony wielomian $c(x)$ (wielomian 3 stopnia o współczynnikach w $GF(2^8)$).

$$c(x)='03'x^3+'01'x^2+'01'x+'02'$$

Wielomian $c(x)$ jest względnie pierwszy z x^4+1, jest więc odwracalny w pierścieniu $GF(2^8)[x]/(x^{4+1})$.

MixColumn działający na pojedynczej kolumnie daje się łatwo sprawdzić, opisać macierzowo tak:

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



Rys. 4.9 Przekształcenie MixColumn działa na kolumnach macierzy stanu.

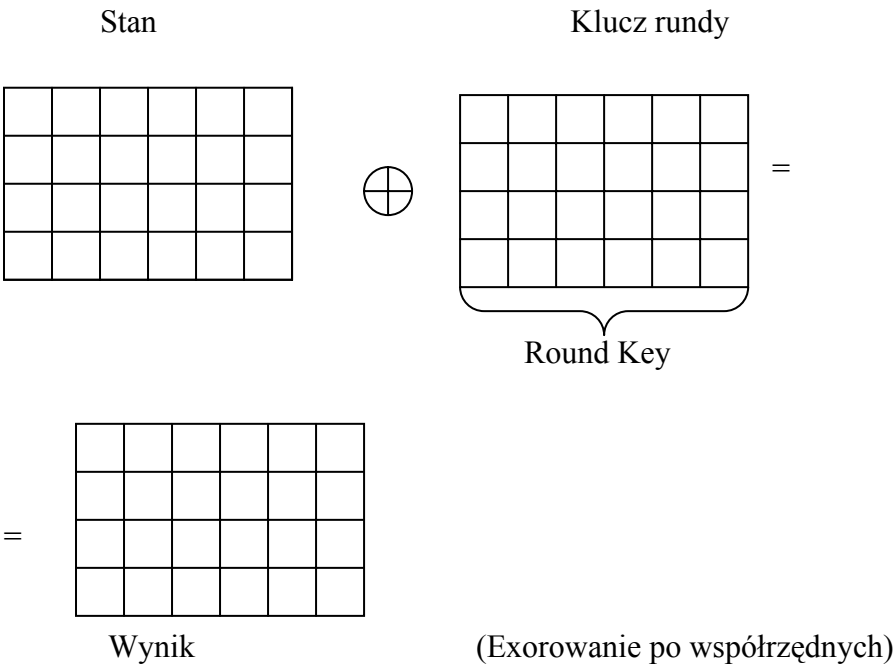
Przekształcenie MixColumn jest różnowartościowe i „na”.

Łatwo uzyskujemy przekształcenie odwrotne do MixColumn. Najpierw znajdujemy wielomian $d(x)$ taki, że:

$$('03'x^3+'01'x^2+'01'x+'02')\otimes d(x)='01'$$

czyli wielomian odwrotny względem $c(x)='03'x^3+'01'x^2+'01'x+'02'$. Łatwo sprawdzić, że: $d(x)='0B'x^3+'0D'x^2+'09'x+'0E'$. Mnożymy teraz kolumny macierzy stanu MixColumn(State) przez $d(x)$ i uzyskujemy wartość funkcji odwrotnej czyli State.

4.Dodawanie klucza rundy (ang. round key) Add Round Key (State, Round Key)



Rys. 4.10 Schemat realizacji procedury AddRoundKey(State, Round Key) czyli procedury dodawania klucza rundy

Oczywiście, dodawanie klucza rundy jest równowartościowe i „na". Odwracanie tego przekształcenia to ponowne dodanie takiego samego klucza rundy do wyniku.

Tworzenie kluczy rundy (ang. Round Key) dla danego klucza szyfru (ang.Cipher Key)

Omówiliśmy już strukturę rundy. Runda w gruncie rzeczy to pewne przekształcenie stanu algorytmu parametryzowane kluczem rundy składające się z elementarnych procedur. By zrealizować algorytm szyfrowania wykonujemy kilka lub kilkanaście rund przy różnych ogólnie rzecz biorąc kluczach rundy. Klucze rundy muszą więc być wstępnie utworzone.

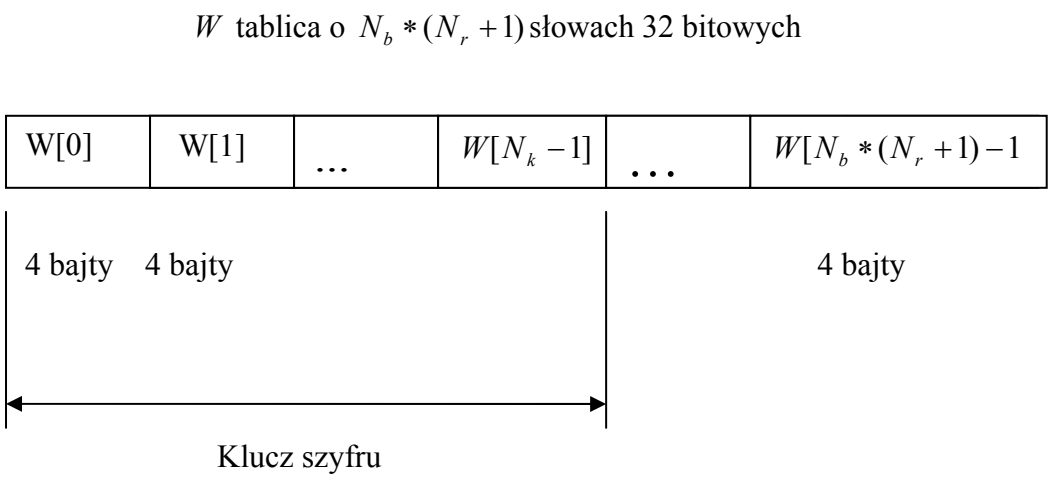
Tworzenie klucza rundy składa się z 2 etapów

1.Etap 1 to tworzenie długiej jednowymiarowej macierzy W o indeksach od 0 do $N_b * (N_r + 1) - 1$. Wartościami tej macierzy są 32 bitowe słowa. Punktem wyjścia jest klucz szyfru (ang Cipher Key) – stanowi on dane wejściowe a procedura tworzenia macierzy W nosi nazwę „key expansion".

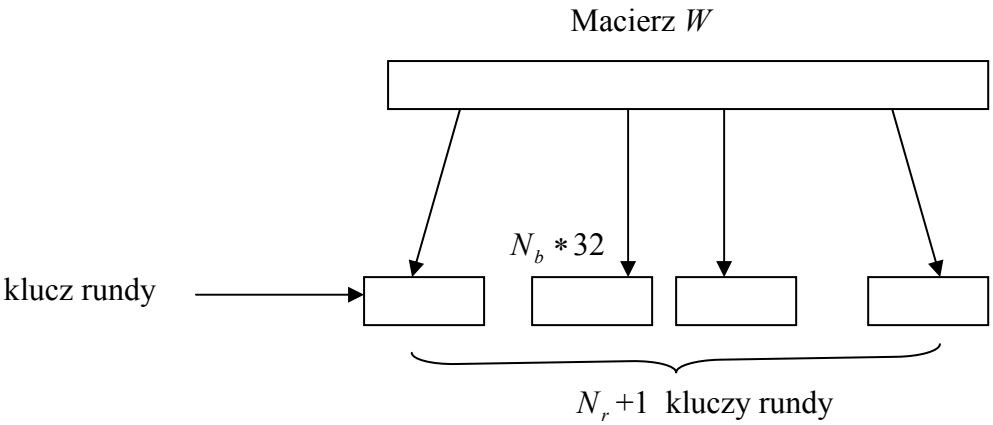
2. Etap 2 to wybór z tak przygotowanej tablicy W kolejno podkluczy czyli kluczy rundy.

Procedura Key Expansion

Procedura Key Expansion wychodząc z danych wejściowych jakie stanowi klucz szyfru rozszerza, rozbudowuje klucz szyfru (o N_k słowach 32 bitowych) tworząc jednowymiarową długą macierz (tablicę) W o $N_b * (N_r + 1)$ słowach 32 bitowych.. Na tej macierzy pracuje procedura Round KeySelect „tnąc" macierz W na fragmenty stanowiące klucze rundy. Czytelnik łaskawie zauważy, że długość bloku wiadomości jawnej $N_b * 32$ jest równa długości klucza rundy.



Rys. 4.11 Struktura macierzy W



długość klucza rundy = długość bloku = $N_b * 32\ b$

Rys 4.12 Procedura Round KeySelect tworzy (czy może raczej wybiera) z macierzy jednowymiarowej W o $N_b * (N_r + 1)$ słowach 32 bitowych kolejne klucze rundy.

Procedurę Key Expansion można dla $N_k \leq 6$ opisać w pseudokodzie C następująco.

```
// Dla Nk ≤ 6 mamy

Key Expansion (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]);

// przepisanie klucza szyfru do macierzy W.

for (i = Nk; i < Nb * (Nr + 1); i++)
        {
        temp = W[i − 1];
        if (i%Nk == 0)  temp = SubByte (RotByte (temp)) ^ Rcon[i / Nk];
// suma modulo 2 czyli EXOR:  SubByte (RotByte (temp))  oraz  Rcon [i / Nk]
        W[i] = W[i − Nk] ^ temp;

        }

}
```

Procedura SubByte jest funkcją pracującą na 4 bajtowym słowie jako danych wejściowych i zwracająca 4 bajtowe słowo. Do każdego z 4 bajtów procedura stosuje opisany powyżej S-box i tak uzyskane dane wyprowadza.

Procedura RotByte jest funkcją pracującą na 4 bajtowym słowie jako danych wejściowych i zwracająca 4 bajtowe słowo. Slowo 32 bitowe (*a*,*b*,*c*,*d*), gdzie *a*,*b*,*c*,*d*, są bajtami zastępowane jest słowem przesuniętym (*b*,*c*,*d*,*a*).

Tablica stałych rundy Rcon $[i]$ zdefiniowana jest następująco jako tablica słów 32 bitowych:

$$\text{Rcon}[i] = (RC[i], '00', '00', '00')$$

gdzie $RC[i]$ jest bajtem reprezentującym w $GF(2^8)$ element x^{i-1} tak więc mamy:

$RC[1] = 1$ tzn. jest to bajt '01'
$RC[2] = x$ tzn. jest to bajt '02'
$RC[i] = x \bullet RC[i-1] = '02' \bullet RC[i-1]$

Dla $N_k > 6$ procedura Key Expansion jest następująca

// Dla $N_k > 6$ mamy
Key Expansion (byte key $[4 + N_k]$, word $W[N_b \cdot (N_r + 1)]$)
{
for $(i = 0; i < N_k; i++)$ $W[i] = (\text{Key}[4 * i], \text{Key}[4 * i + 1], \text{Key}[4 * i + 2], \text{Key}[4 * i + 3])$;

// przepisanie klucza szyfru do macierzy *W*.

for $(i = N_k;\ i < N_b * (N_r + 1);\ i++)$
{
temp = $W[i-1]$;
if $(i\%N_k == 0)$ temp = SubByte (RotByte (temp)) ^ Rcon$[i\ /\ N_k]$;
else if $(i\ \%\ N_k == 4)$ temp=SubByte (temp);
// suma modulo 2 czyli EXOR: SubByte (RotByte (temp)) oraz Rcon $[i / N_k]$
$W[i] = W[i - N_k]$ ^ temp;

}

}

Algorytm szyfru AES.

1.Wstępne dodanie (modulo 2 po współrzędnych) do danych wejściowych Klucza Rundy (ang. Round Key)

2. Wykonanie $N_r - 1$ rund

3. Runda końcowa (nieco inna niż pozostałe)

} W sumie mamy N_r rund

W pseudokodzie *C* algorytm powyższy można zapisać tak:

Rijdael(State, CipherKey);
{
Key Expansion (Cipher Key, Expanded Key);
Add Round Key (State, Expanded Key);
for $(i = 1;\ i < N_r;\ i++)$ *Round*(State, Expanded Key + $N_b * i$);
Final Round(State, Expanded Key $+ N_b * N_r$)
}

4. 4 Szyfr IDEA

Szyfr IDEA (International Data Encryption Algorithm) został zaproponowany przez James'a L.Massey'a oraz Xuejia Lai i opublikowany w 1990 roku pod nazwą PES. Jest to jeden z najbezpieczniejszych dostępnych szyfrów blokowych. IDEA była projektowana tak by była odporna na kryptoanalizę różnicową.

IDEA jest szyfrem z kluczem symetrycznym, operuje na 64-bitowych blokach danych wejściowych (podobnie jak DES), ale długość klucza równa jest 128 bitom. 64-bitowy blok danych wejściowych jest dzielony na cztery 16-bitowe podbloki, które stają się danymi wejściowymi dla pierwszej rundy algorytmu.

Algorytm IDEA operuje na 16-bitowych podblokach i jest typową siecią permutacyjno-podstawieniową. Podobnie jak inne algorytmy blokowe IDEA wykorzystuje do kodowania zarówno mieszanie (odpowiednie permutacje) jak i rozpraszanie (odpowiednie podstawienia). Algorytm szyfrowania IDEA wykorzystuje osiem rund do szyfrowania informacji.

W przypadku rozwiązań hardware'owych szybkością IDEA dorównuje algorytmowi DES. Dla rozwiązań programowych IDEA jest prawie dwukrotnie szybsza od DES-a.

IDEA jest często stosowana w programie do szyfrowania plików i poczty (np. w znanym programie PGP). Szyfr IDEA jest opatentowany.

4.5 Inne szyfry z kluczem prywatnym

Omówione do tej pory szyfry to DES, DESX, 3DES, AES (Rijndael) i IDEA. Inne znane szyfry stosowane w praktyce to LOKI, RC5, CAST-128, GOST-256, BLOWFISH-448, i Twofish.

Literatura

[1] A. Menezes, P. Oorschot, S. Vanstone; Handbook of Applled Cryptography; CRC Press Inc., 1997. (treść jest na stronie: http://cacr.math.uwaterloo.ca/hac)

[2] J.Stokłosa, T.Bilski,T.Pankowski; Bezpieczeństwo danych w systemach informatycznych; PWN, Warszawa 2001.

[3] N.Koblitz; A Course in Number Theory and Cryptography; Springer Verlag, New York 1994. (jest przekład polski p.t. Wykład z teorii liczb i kryptografii; WNT, Warszawa 1995.)

[4] M.Kutyłowski, W.Strothmann; Kryptografia, teoria i praktyka zabezpieczania systemów komputerowych; Oficyna Wydawnicza Read Me, Warszawa 2001.

[5] B.Schneier; Kryptografia dla praktyków; WNT, 2002.

Zadania

Zadanie 1
Wykazać, że w systemie kryptograficznym DES (ang. Data Encryption Standard) dla ustalonego 64 bitowego klucza *K*.

1) każda i ta runda (rund mamy 16) jest pewnym przekształceniem $g_i : \{0,1\}^{64} \to \{0,1\}^{64}$ różnowartościowym i „na".
2) Również przekształcenie szyfrujące DES'a $g : \{0,1\}^{64} \to \{0,1\}^{64}$ jest różnowartościowe i „na".

Rozwiązanie
1. DES jest typowym szyfrem blokowym szyfrującym słowa binarne 64 bitowe (są to jednostki tekstu lub jak czasem mówimy bloki). Dokładniej dla ustalonego klucza słowa binarne 64 bitowe przekształcane są na słowa binarne 64 bitowe. Lewą 32 bitową część słowa 64 bitowego będziemy oznaczać symbolem *L*, a prawą 32 bitową część symbolem *R*. Tak więc przedstawione w algorytmie DES słowo 64 bitowe to konkatenacja *L* i *R*; czyli *LR*.
Obliczenie kryptogramu z wiadomości jawnej

$$m = m_1 m_2 ... m_{64}$$

gdzie $m_i \in \{0,1\}$, przebiega dla ustalonego (64 bitowego) klucza *K* następująco:

1.Obliczamy $IP(m)=L_0R_0$, gdzie $IP : \{0,1\}^{64} \to \{0,1\}^{64}$ jest tzw. permutacją początkową słowa binarnego m oraz $L_0, R_0 \in \{0,1\}^{32}$.

2.Dla każdej z 15 tzw. rund (czyli etapów obliczeń) obliczamy

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i) \end{aligned} \qquad (*)$$

dla i=1,2,...,15, gdzie $f : \{0,1\}^{32} \times \{0,1\}^{48} \to \{0,1\}^{32}$ jest funkcją zdefiniowaną w standardzie DES'a, a K_i jest i-tym 48 bitowym podkluczem otrzymanym (jak czasem mówimy generowanym) z 64 bitowego klucza *K*. Suma modulo 2 oznaczona symbolem $\oplus$ jest sumą modulo 2 po współrzędnych.

1.16 runda (i=16) opisywana jest nieco innym równaniem niż poprzednie rundy.

$$\begin{aligned} L_{16} &= L_{15} \oplus f(R_{15}, K_{16}) \\ R_{16} &= R_{15} \end{aligned}$$

2.ostatecznie kryptogram jest równy:

$$c = c_1 c_2 ... c_{64} = IP^{-1}(L_{16}, R_{16})$$

gdzie IP^{-1} jest tzw. permutacją końcową. Jest to permutacja odwrotna względem permutacji początkowej *IP*. Permutacja *IP* opisana jest w standardzie DES'a.

2. Zauważmy, że dla sumy modulo 2 w Z_2 mamy dla dowolnego, ustalonego $a \in \{0,1\}$ i każdego $x \in \{0,1\}$:

$$a \oplus a \oplus x = x$$

Zatem odwzorowanie zdefiniowane wzorem:

$$\{0,1\}^n \ni \bar{x} \to \bar{a} \oplus \bar{x} \in \{0,1\}^n$$

(gdzie $\bar{a}, \bar{x} \in \{0,1\}^n$ oraz $n \in N$ jest ustalone a symbol $\oplus$ oznacza sumę modulo 2 po współrzędnych), jest inwolucją określoną na zbiorze skończonym, zatem musi być różnowartościowe i „na". W szczególności więc, każde z odwzorowań (dla *i* =1,2,...,15)

$$g_{r,i} : \{0,1\}^{32} \ni L_{i-1} \to L_{i-1} \oplus f(R_{i-i}, K_i) \in \{0,1\}^{32}$$

(dla ustalonego podklucza K_i i ustalonego R_{i-i}) jest różnowartościowe i „na". Odwzorowanie tożsamościowe:

$$g_{l,i} : \{0,1\}^{32} \ni R_{i-1} \to R_{i-1} \in \{0,1\}^{32}$$

jest oczywiście również różnowartościowe i „na". Zatem odwzorowanie realizowane przez (*), czyli odwzorowanie

$$g_i = (g_{l,i}, g_{r,i}) : \{0,1\}^{64} \ni L_{i-1}R_{i-1} \to L_iR_i \in \{0,1\}^{64}$$

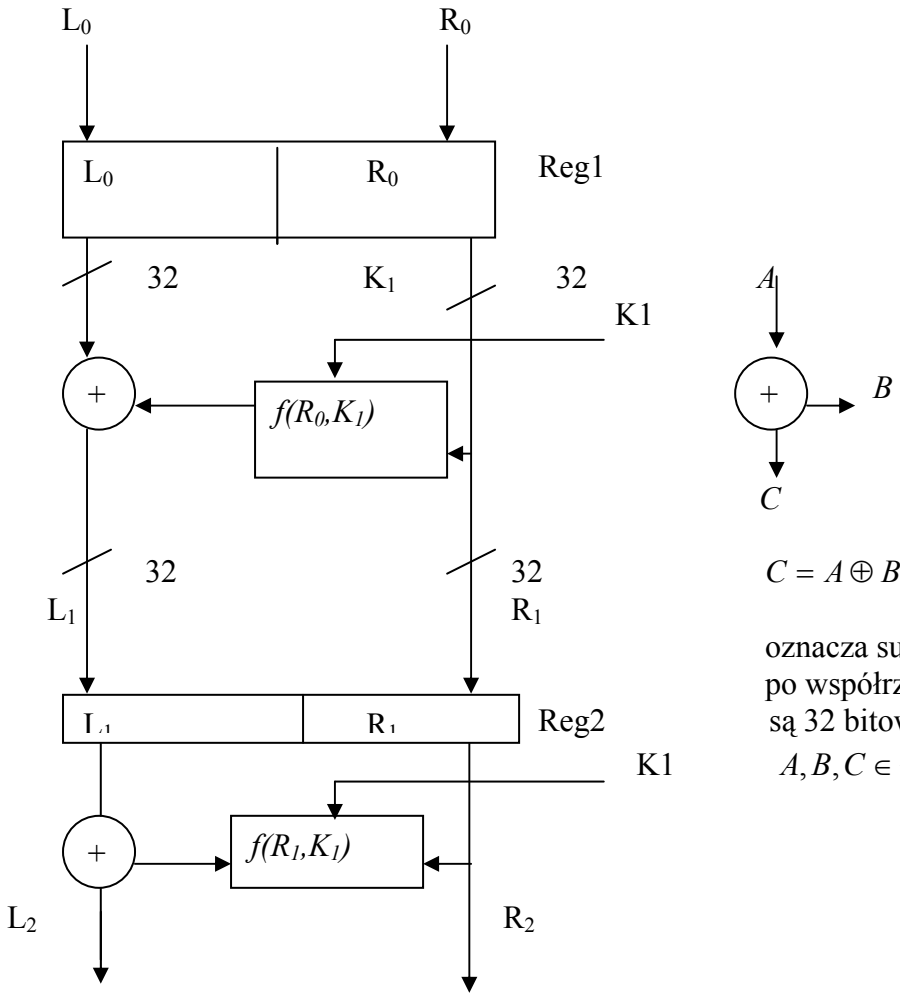
jest różnowartościowe i "na". Istotnie, ponieważ dziedzina i przeciwdziedzina są skończone i równoliczne wystarczy pokazać, że g_i jest "na". Weźmy dowolne ustalone słowo binarne $L_iR_i \in \{0,1\}^{64}$ odpowiadający temu słowu argument znajdziemy jako $L_{i-1}R_{i-1}$ tak $R_{i-i} = R_i$ ($g_{r,i}$ jest różnowartościowe) a $L_{i-1} = L_i \oplus f(R_{i-i}, K_i) \in \{0,1\}^{32}$.
Podobnie wykazujemy, że przekształcenie realizowane przez rundę 16 jest różnowartościowe i „na".

3. Dla ustalonego klucza *K,* przekształcenie szyfrujące DES'a

$$g = IP^{-1} \circ g_{16} \circ g_{15} \circ ... \circ g_1 \circ IP$$

jako superpozycja odwzorowań różnowartościowych i „na" jest różnowartościowe i „na". ■

Zadanie 2
Wykazać, że układ z rys. 2 realizujący 2 rundy DES'a realizuje odwzorowanie tożsamościowe.



Rys. 2 Dwie rundy algorytmu DES realizujące przekształcenie tożsamościowe. Reg1, Reg2 są rejestrami Funkcja $f : \{0,1\}^{32} \times \{0,1\}^{32} \to \{0,1\}^{32}$ jest funkcją z definicji standardu DES ale nawet jeśli $f : \{0,1\}^{32} \times \{0,1\}^{32} \to \{0,1\}^{32}$ jest dowolną funkcją to i tak dowodzony fakt pozostanie prawdziwy.

Rozwiązanie
Istotnie ze schematu przedstawionego na rys. 7.2 mamy

$$L_1 = L_0 \oplus f(R_0, K_1)$$
$$R_1 = R_0$$
$$L_2 = L_1 \oplus f(R_1, K_1)$$
$$R_2 = R_1$$

zatem:

$$L_2 = L_0 \oplus f(R_0, K_1) \oplus f(R_0, K_1)$$
$$R_2 = R_0$$

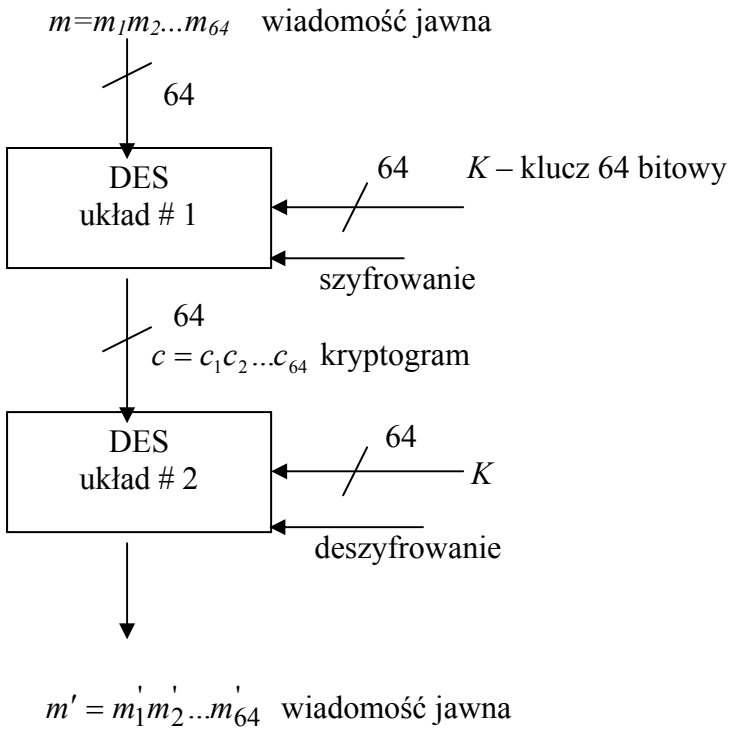
Jeśli $x \in \{0,1\}^n$ to suma modulo 2 po współrzędnych ma tę własność, że $x \oplus x = 0$ oraz $x \oplus 0 = x$ (gdzie 0 oznacza $(0,0,...,0) \in \{0,1\}^n$), zatem $L_2=L_0$ oraz $R_2=R_0$; a więc 2 rundy przedstawione na rysunku realizują odwzorowanie tożsamościowe. ■

Zadanie 3

Pokazać, że w układzie realizującym szyfrowanie DES wystarczy odwrócić kolejność 48 bitowych podkluczy $K_1,K_2,...,K_{16}$, by uzyskać układ deszyfrujący kryptogram.

Rozwiązanie

1. Wystarczy oczywiście pokazać, że jeśli poddajemy wiadomość jawną $m = m_1 m_2 ... m_{64}$, gdzie $m_i \in \{0,1\}$ przekształceniu DES'em dwukrotnie tak jak na rysunku 3, to otrzymamy tą samą wiadomość jawną m. Klucz $K \in \{0,1\}^{64}$ podawany na oba układy jest ten sam, ale sygnał szyfrowanie/deszyfrowanie zmienia kolejność generowanych podkluczy.



Rys. 3 Szyfracja i deszyfracja za pomocą algorytmu DES

Żeby wykazać, że $m = m'$ pokażemy, że układ z rys. 7.3 dla ustalonego klucza K realizuje przekształcenie tożsamościowe $id : \{0,1\}^{64} \to \{0,1\}^{64}$.

W sytuacji jak na rysunku 7.3 z 64 bitowego klucza K, powstaje w układzie #1 16 tzw. podkluczy $K_1,K_2,...,K_{16}$ (są to słowa 48 bitowe), a w układzie #2 16 podlkluczy $K'_1,K'_2,...,K'_{16}$.

Układ jest jednak tak skonstruowany (technicznie jest to bardzo proste, sprowadza się bowiem do wykorzystania bramek trójstanowych lub multiplekserów), że w układzie #2 mamy:

$$K_1^{'} = K_{16},\ \ K_2^{'} = K_{15},\ \ K_3^{'} = K_{14},...,K_{16}^{'} = K_1 .$$

W pierwszym układzie półklucze podawane są więc do kolejnych rund w kolejności K_1, $K_2,...,K_{16}$, a w drugim – w kolejności $K_{16},K_{15},K_{14},...,K_1$.

2. Oznaczmy przez $\pi : \{0,1\}^{64} \to \{0,1\}^{64}$ permutację zbioru $\{0,1\}^{64}$, polegającą na przestawieniu $LP \to PL$, gdzie $L, P \in \{0,1\}^{32}$. Jest to przestawienie połówek słowa 64 bitowego. Niech *IP* będzie permutacją początkową, a IP^{-1} permutacją końcową (jest to permutacja odwrotna do *IP*) z definicji standardu DES.
Niech ponadto:

$$g_{L,i}(k,\cdot) : \{0,1\}^{64} \to \{0,1\}^{32} \qquad (*)$$

będzie przekształceniem słowa wejściowego 64 bitowego *i*-tej rundy $L_{i-1}R_{i-1}$ w lewą 32 bitową część $L_i^{'}$ 64 bitowego słowa wyjściowego *i*-tej rundy $L_i^{'}R_i^{'}$ (przed zamianą połówek 64 bitowego słowa, czyli przed wykonaniem permutacji π). Zakładamy przy tym, że do tej rundy doprowadzony jest 48 bitowy podklucz *k*. Przekształcenie (*) zdefiniowane jest wzorem:

$$g_{L,i}(k,\cdot) : \{0,1\}^{64} \ni L_{i-1}R_{i-1} \to L_i^{'} = g_{L,i}(k, L_{i-1}) = L_{i-1} \oplus f(R_{i-1}, k) \in \{0,1\}^{32}$$

gdzie $\oplus$ jest sumą modulo 2 po współrzędnych, $L_i^{'}$ jest lewą połówką słowa wyjściowego, $R_i^{'}$ prawą połówką słowa wyjściowego rundy (przed wykonaniem permutacji π).

Zdefiniujmy jeszcze przekształcenie:

$$g_{R,i}(k,\cdot) : \{0,1\}^{64} \to \{0,1\}^{32}$$

wzorem:

$$L_{i-1}R_{i-1} \to R_i^{'} \overset{ozn.}{=} g_{R,i}(k, R_{i-1}) \overset{df}{=} R_{i-1}$$

a więc słowo wyjściowe przekształcenia $g_{R,i}$, to prawa część słowa wejściowego.

Oznaczmy $g_i(k,\cdot) \overset{df}{=} (g_{L,i}(k,\cdot), g_{R,i}(k,\cdot))$.

Przekształcenie realizowane przez układ #1 możemy teraz opisać jako superpozycję odwzorowań:

$$U_1 \overset{df}{=} IP^{-1} \circ g_{16}(K_{16}) \circ \pi \circ g_{15}(K_{15}) \circ \pi \circ ... \circ g_2(K_2) \circ \pi \circ g_1(K_1) \circ IP$$

Przekształcenie realizowane przez układ #2 możemy z kolei opisać jako superpozycję odwzorowań:

$$U_2 \overset{df}{=} IP^{-1} \circ g_{16}(K_1) \circ \pi \circ g_{15}(K_2) \circ \pi \circ ... \pi \circ g_2(K_{15}) \circ \pi \circ g_1(K_{16}) \circ IP$$

Zatem układ będący połączeniem dwóch układów #1 i #2 realizuje przekształcenie

$$U_2 \circ U_1 \overset{df}{=} IP^{-1} \circ g_{16}(K_1) \circ \pi \circ g_{15}(K_2) \circ \pi \circ ... \pi \circ g_1(K_{16}) \circ IP \circ IP^{-1} \circ$$
$$\circ g_{16}(K_{16}) \circ \pi \circ ... \circ g_2(K_2) \circ \pi \circ g_1(K_1) \circ IP$$

Zaczniemy „składanie" powyższej superpozycji od środka:

$$IP \circ IP^{-1} = id$$

$g_1(K_{16}) \circ g_{16}(K_{16}) = id$ (wykazano to w zadaniu 7.2)

$\pi \circ \pi = id$

$g_2(K_{15}) \circ g_{15}(K_{15}) = id$ (wykazano to w zadaniu 7.2)

itd.

Stwierdzamy więc ostatecznie, że $U_2 \circ U_1 = id$. ■

Zadanie 4
Dodać następujące wielomiany (bajty) w pierścieniu $Z_2[x]/(x^8 + x^4 + x^3 + x + 1) = GF(2^8)$
a) '57'+'02'
b) '03'+'03'
c) 'FF'+'0F'

Uwaga. Ciało skończone $GF(2^8)$ jest podstawową strukturą algebraiczną wykorzystywaną w szyfrze symetrycznym AES. Wielomian $m(x) = x^8 + x^4 + x^3 + x + 1$ jest wielomianem nierozkładalnym w pierścieniu $Z_2[x]$.

Rozwiązanie

a) '57'+'02'='55'

b) '03'+'03'='00'

c) 'FF'+'0F'='F0'

■

Zadanie 5
Pomnożyć następujące wielomiany (bajty) w pierścieniu
$Z_2[x]/(x^8 + x^4 + x^3 + x + 1) = GF(2^8)$

a) ‘57’+’02’
b) ‘57’+’04’
c) ‘57’+’10’

Rozwiązanie

a) ‘57’ ▪ ’02’=’AE’

b) ‘57’ ▪ ’04’=’47’

c) ‘57’ ▪ ’10’=’07’

■

Zadanie 6
Napisać program w asemblerze realizujący procedurę *xtime*(‘XY’) mnożenia wielomianu ‘XY’ (w zapisie heksadecymalnym) przez wielomian x.

Zadanie 7
Rozważmy pierścień wielomianów $GF(2^k)[x]$ (gdzie k jest dowolną ustaloną liczbą naturalną) o współczynnikach w ciele $GF(2^k)$. Wykazać, że jeśli x^n jest dla $n \in N$ wielomianem z $GF(2^k)[x]$, to mamy:

$$x^n (\mathrm{mod}(x^4 + 1)) = x^{n(\mathrm{mod}\,4)} \qquad (*)$$

Uwaga. Powyższą równość wykorzystujemy do przyspieszenia obliczeń w pierścieniu $GF(2^k)[x]/(x^4 + 1)$. Warto przypomnieć, że $GF(2^8)$, $GF(2^8)[x]$ i $GF(2^8)[x]/(x^4 + 1)$ są zasadniczymi strukturami algebraicznymi wykorzystywanymi w szyfrze symetrycznym AES (Rijndael). Elementy $GF(2^8)$ reprezentowane są w komputerze jako bajty a elementy $GF(2^8)[x]/(x^4 + 1)$ jako słowa 32 bitowe.

Rozwiązanie

1.W ciele $Z_2 = \{0,1\}$ dodawanie jest zwykłą sumą modulo 2 (oznaczamy je symbolem "⊕" i układowo realizujemy za pomocą bramki EXOR). I co zaskakujące również odejmowanie w Z_2 jest sumą modulo 2, bo mamy $1 \oplus 1 = 0$ i $0 \oplus 0 = 0$ więc $-a = a$ dla $a \in Z_2$ oraz $a - b = a \oplus b$ dla $a, b \in Z_2$, gdzie - jest odejmowaniem modulo 2 w Z_2.

2.W ciele $GF(2^k)$, (którego elementami są słowa binarne o długości *k*) definiujemy działanie dodawania standardowo jako sumę wielomianów ale w naszej sytuacji jest to jednocześnie suma modulo 2 po współrzędnych tzn. jeśli $a = (a_1, a_2, ..., a_k) \in GF(2^k)$, gdzie $a_i \in \{0,1\}$ oraz $b = (b_1, b_2, ..., b_k) \in GF(2^k)$, gdzie $b_i \in \{0,1\}$ to:

$$a + b = (a_1 \oplus b_1, a_2 \oplus b_2, ..., a_n \oplus b_k)$$

oraz

$$a - b = (a_1 - b_1, a_2 - b_2, ..., a_k - b_k) = (a_1 \oplus b_1, a_2 \oplus b_2, ..., a_k \oplus b_k)$$

3.Oczywiście, jeśli $n < 4$, to dowodzony wzór (*) jest prawdziwy. Dla $n \geq 4$ istnieje takie $q \in N$, że $n = q \cdot 4 + r$ i $0 \leq r < 4$, oczywiście $r = n(\mathrm{mod}\, 4)$.

Zauważmy jak przebiega dzielenie wielomianu x^n dla $n \geq 4$. Uwzględniając uwagi z p.2 (że „odejmowanie to dodawanie") mamy:

$$\begin{array}{l|l}
x^{n-4} + x^{n-8} + ... + x^{n-q\cdot 4} & \\
\hline
x^n & : x^4 + 1 \\
x^n + x^{n-4} & \\
\hline
\quad x^{n-4} & \\
\quad x^{n-4} + x^{n-8} & \\
\quad \underline{\qquad\qquad} & \\
\qquad x^{n-8} & \\
\qquad\quad . & \\
\qquad\quad . & \\
\qquad\quad . & \\
\qquad\quad \underline{\qquad} & \\
\qquad\quad x^r &
\end{array}$$

a więc istotnie: $x^n (\mathrm{mod}(x^4 + 1)) = x^{n(\mathrm{mod}\, 4)}$.

■

Zadanie 8

Rozważmy pierścień wielomianów $GF(2^k)[x]$ (gdzie k jest dowolną ustaloną liczbą naturalną) o współczynnikach w ciele $GF(2^k)$. Wykazać, że jeśli x^n jest dla $n \in N$ wielomianem z $GF(2^k)[x]$, to mamy dla każdego naturalnego $s \geq 2$:

$$x^n \,(\mathrm{mod}(x^s + 1)) = x^{n(\mathrm{mod}\, s)} \qquad (*)$$

Rozwiązanie

Rozwiązanie jest analogiczne do rozwiązania zadania 1. Punkty1. i 2. nie ulegają zmianie. W ostatnim 3-cim punkcie mamy:

3. Jeśli $n < s$, to dowodzony wzór (*) jest oczywiście prawdziwy. Dla $n \geq s$ istnieje takie $q \in N$, że $n = q \cdot s + r$ i $0 \leq r < 4$, oczywiście $r = n(\mathrm{mod}\, s)$.

Zauważmy jak przebiega dzielenie wielomianu x^n dla $n \geq s$. Uwzględniając uwagi z p.2 (że „odejmowanie to dodawanie") mamy:

$$\begin{array}{l|l}
x^{n-s} + x^{n-s} + \ldots + x^{n-q\cdot s} & \\
\hline
x^n & : x^s + 1 \\
x^n + x^{n-s} & \\
\hline
\quad x^{n-s} & \\
\quad x^{n-s} + x^{n-2\cdot s} & \\
\quad \underline{\qquad\qquad} & \\
\quad x^{n-2\cdot s} & \\
\qquad\qquad . & \\
\qquad\qquad . & \\
\qquad\qquad . & \\
\qquad\qquad \underline{\qquad} & \\
\qquad\qquad x^r &
\end{array}$$

a więc istotnie: $x^n \,(\mathrm{mod}(x^s + 1)) = x^{n(\mathrm{mod}\, s)}$.

■

Zadanie 9

Wykazać że mnożenie dwóch wielomianów $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ i $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$ w pierścieniu ilorazowym $GF(2^8)[x]/(x^4 + 1)$ sprowadza się do następującego mnożenia macierzy przez wektor

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

gdzie $d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$ jest wynikiem mnożenia.

Uwaga. Łatwo zauważyć, że macierz tworzymy przesuwając cyklicznie pierwszą kolumnę. Powyższy fakt wykorzystujemy w implementacji szyfru AES.

Zadanie 10

Wykazać że mnożenie dwóch wielomianów $a(x) = a_{s-1}x^{s-1} + a_{s-2}x^{s-2} + ... + a_1x + a_0$ i $b(x) = b_{s-1}x^{s-1} + b_{s-2}x^{s-2} + ... + b_1x + b_0$ (gdzie s jest liczbą naturalną $s \geq 2$) w pierścieniu ilorazowym $GF(2^k)[x]/(x^s + 1)$ (gdzie k jest dowolną ustaloną liczbą naturalną) sprowadza się do następującego mnożenia macierzy przez wektor

$$\begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{s-1} \end{bmatrix} = \begin{bmatrix} a_0 & a_{s-1} & \cdots & a_1 \\ a_1 & a_0 & \cdots & a_2 \\ \vdots & \vdots & \cdots & \vdots \\ a_{s-1} & a_{s-2} & \cdots & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{s-1} \end{bmatrix}$$

gdzie $d(x) = d_{s-1}x^{s-1} + d_{s-2}x^{s-2} + ... + d_1x + d_0$ jest wynikiem mnożenia.

Uwaga. Łatwo zauważyć, że macierz tworzymy przesuwając cyklicznie pierwszą kolumnę.