Wykład 5

Temat: Inne symetryczne algorytmy kryptograficzne: Blowfish, CAST, 3DES, GOST, IDEA, RC2, RC4, Rijndael (AES).

5.1. Blowfish

Algorytm Blowfish (pol. *rozdymka*) został zaprojektowany by spełnić następujące wymagania:

1. Duża szybkość działania – algorytm szyfruje dane z szybkością 26 taktów zegarowych na bajt.
2. Małe wymagania co do pamięci – algorytm wymaga nie więcej niż 5KB pamięci.
3. Prostota algorytmu – algorytm stosuje tylko proste operacje tj. dodawanie, sumowanie modulo 2 i tablicę przeszukiwania jednego z 32 argumentów.
4. Zmienna długość klucza – algorytm akceptuje klucze o długości mniejszej niż 448 bitów.

Algorytm ten nadaje się przede wszystkim w zastosowaniach nie wymagających częstych zmian kluczy, a więc przykładowo w łączach telekomunikacyjnych albo automatycznym szyfrowaniu plików.

Algorytm Blowfish jest szyfrem blokowym z 64-bitowymi blokami i kluczem o zmiennej długości. Algorytm ten składa się z dwóch części: rozszerzającej klucz i szyfrującej. Część rozszerzająca klucz zmienia klucz o długości do 448 bitów na kilka tablic podkluczy o łącznej wielkości 4168 bajtów. Część szyfrująca składa się z prostej funkcji wykonywanej 16 razy. Każdy cykl składa się z permutacji zależnej od klucza oraz podstawienia zależnego od klucza i danych. Operacje wykonywane przez algorytm są proste dla realizacji sprzętowych.

Algorytm używa jako podkluczy dużych liczb. Podklucze muszą zostać wyznaczone przed procesem szyfrowania i deszyfrowania. Macierz P składa się z osiemnastu 32-bitowych podkluczy: P_1, P_2,..., P_{18}.

Każdy z czterech 32-bitowych S-bloków ma 256 wyjść:

$$S_{1,0}, S_{1,1},..., S_{1,255}$$

$$S_{2,0}, S_{2,1},..., S_{2,255}$$

$$S_{3,0}, S_{3,1},..., S_{3,255}$$

$$S_{4,0}, S_{4,1},..., S_{4,255}$$

Blowfish jest 16 rundową siecią Feistela. Oznaczamy 64-bitowy ciąg danych wejściowych jako x. Aby zaszyfrować blok x należy:

1. Podzielić blok wejściowy x na dwie 32-bitowe części x_L, x_R
2. Dla i od 1 do 16:

   $x_L = x_L \oplus P_i$

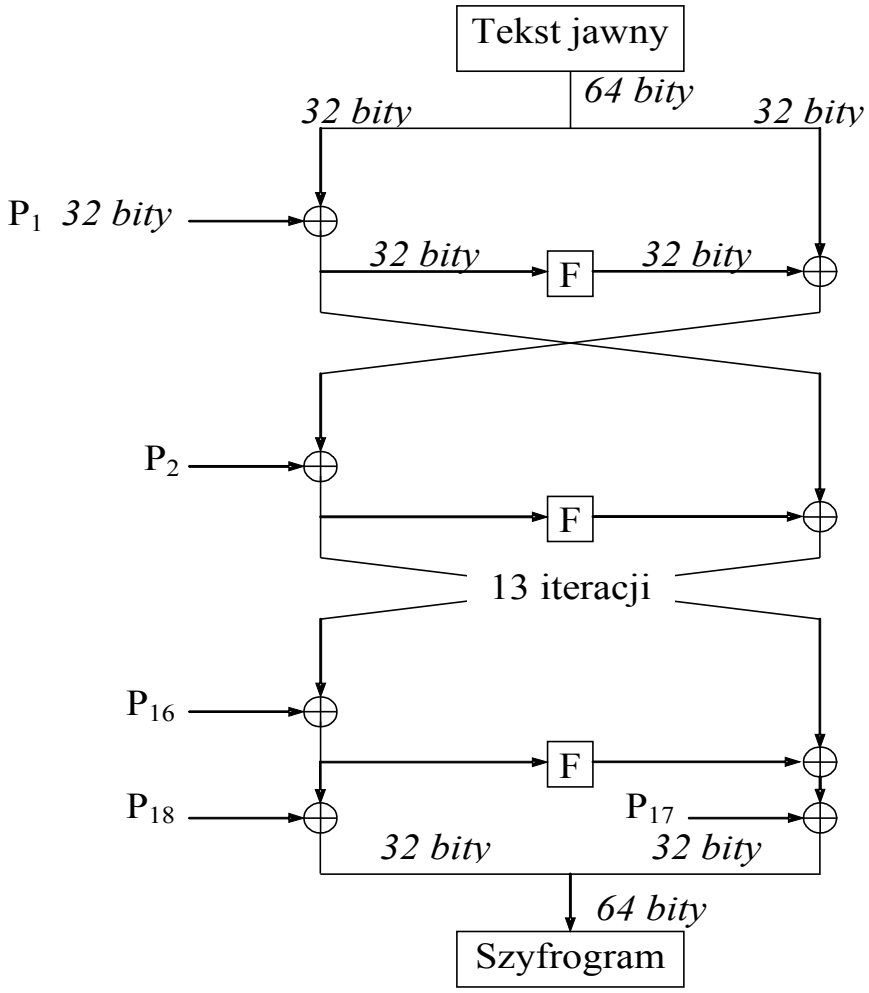
   $x_R = F(x_L) \oplus x_R$

   Zamienić miejscami x_L i x_R
3. Po 16 cyklach następuje wyznaczenie ostatecznych wartości x_L i x_R:

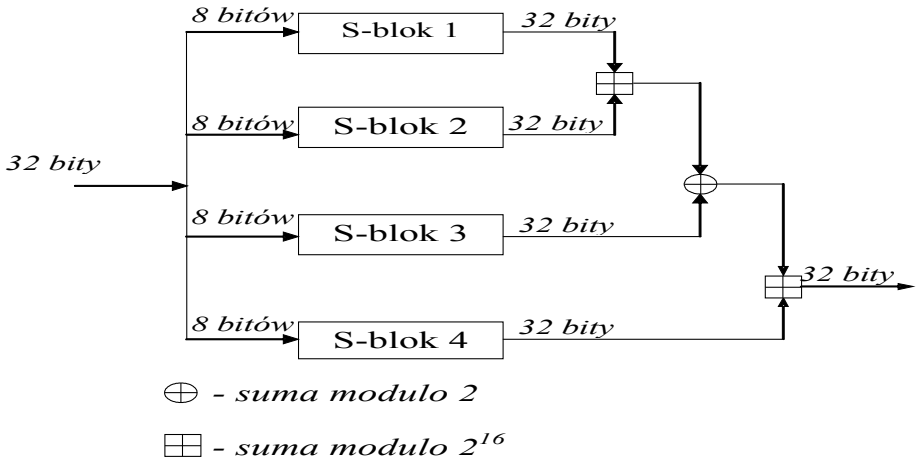
   $x_R = x_R \oplus P_{17}$

   $x_L = x_L \oplus P_{18}$

   Ponownie połączyć x_L i x_R.



Rys. 5.1. Algorytm Blowfish



Rys. 5.2. Funkcja F algorytmu Blowfish

Na rysunku powyżej przedstawiono funkcje F algorytmu Blowfish. Wejściowe 32-bitowe słowo x_L (lewa strona 64-bitowego tekstu jawnego) jest dzielone na cztery 8-bitowe części: a, b, c, d. Proces ten można zapisać następująco:

$$F(x_L) = ((S_{1,a} + S_{2,b} \bmod 2^{16}) \oplus S_{3,c}) + S_{4,d} \bmod 2^{16}$$

Deszyfrowanie jest dokonywane w taki sam sposób, z tym że P_1, P_2,..., P_{18} są używane w odwrotnej kolejności.

Proces wyznaczania podkluczy przebiega nastepująco:

1. Zainicjowanie macierzy P i czterech S-bloków stałymi ciągami liczbowymi. Elementy ciągów są kolejnymi liczbami szesnastkowymi.
2. Zsumowanie modulo 2 P_1 z pierwszymi 32 bitami klucza, następnie zsumowanie modulo 2 P_2 z kolejnymi 32 bitami klucza. Proces ten jest przeprowadzany dla wszystkich bitów klucza (do P_{18}) aż do czasu zsumowania klucza ze wszystkimi bitami macierzy P.
3. Zaszyfrowanie wszystkich ciągów zawierających same zera algorytmem Blowfish, stosując podklucze opisane w krokach 1 i 2.
4. Zastąpienie P_1 i P_2 wynikami kroku 3.
5. Zaszyfrowanie wyniku kroku 3 przy wykorzystaniu algorytmu Blowfish ze zmodyfikowanymi podkluczami.
6. Zastąpienie P_3 i P_4 wynikami kroku 5.
7. Kontynuowanie procesu generowania podkluczy, zastępując kolejno wszystkie elementy macierzy P i dalej wszystkich (czterech) S-bloków wynikami ciągle zmieniającego się algorytmu Blowfish.

Taki sposób generowania podkluczy wymaga 521 iteracji by proces tworzenia podkluczy został zakończony. Problemem algorytmu Blowfish mogą okazać się klucze słabe. W przypadku wystąpienia niektórych słabych kluczy mogą pojawić się w efekcie słabe S-bloki (ponieważ są zależne od klucza). W takim przypadku do rozpoznania macierzy P są wymagane tylko 2^{4r+1} teksty jawne, gdzie r jest liczbą cykli algorytmu. Klucze słabe algorytmu Blowfish to takie klucze, dla których dwa wejścia do S-bloków są identyczne. Te niebezpieczeństwa dotyczą tylko przypadku zmniejszenia liczby cykli, zatem Blowfish z 16 cyklami okazuje się bezpieczny. Algorytm Blowfish można często spotkać w komercyjnych systemach. Firma Kent Marsh Ltd. zastowała algorytm Blowfish w produktach FolderBolt dla Microsoft Windows i Macintosh. Algorytm ten jest również częścią produktów Nautilius i PGPfone. Blowfish z kluczem 448 bitowym zastosowano w produkcie firmy Jetico BestCrypt, natomiast Blowfish z kluczem 192 bitowym zastosowała firma PC Guardian w produkcie Encryption Plus Folders.

5.2. CAST

Twórcami algorytmu CAST są Carlisle Adams i Stafford Tavares. Algorytm CAST jest algorytmem uniwersalnym – umożliwia realizację z wykorzystaniem różnych długości bloków i kluczy. Przykładowy algorytm CAST stosuje 64-bitowy blok i 64-bitowy klucz.

Siła algorytmu jest ukryta w S-blokach. S-bloki nie mają z góry określonych wartości, są one tworzone dla każdego zastosowania. Kolumnami S-bloku są funkcje wygięte, które są tak dobierane by były spełnione wymagane właściwości S-bloków. Gdy S-bloki są już zaprojektowane (dla danej implementacji), nie ulegają dalszym modyfikacjom. Zależą od implementacji, a nie od klucza. Algorytm stosuje sześć S-bloków o 8-bitowych wejściach i 32-bitowych wyjściach.

Podczas procesu szyfrowania blok tekstu jawnego jest dzielony na dwie połowy. W każdym cyklu prawa połowa jest składana, przy wykorzystaniu funkcji f, z fragmentami klucza i następnie sumowana modulo 2 z lewą połową, tworząc nową prawą połowę. Pierwotna prawa połowa w kolejnym cyklu staje się lewą połową. Takich iteracji jest osiem, po czym obie połowy są łączone tworząc szyfrogram. Po ósmym cyklu nie następuje zamiana stronami prawej i lewej części tekstu jawnego.

Funkcja f przebiega następująco:

1. Wejściowy 32-bitowy blok jest dzielony na cztery 8-bitowe części: a, b, c, d.
2. Podklucz o długości 16 bitów jest dzielony na dwie 8-bitowe części: e,f.
3. Następuje proces przetwarzania 8-bitowych części (a, b, c, d, e, f) przez S-bloki tj. pierwszy S-blok przetwarza część a, drugi S-blok część b itd.
4. Wartości wyjściowe S-bloków są sumowane modulo 2, tworząc 32-bitową wyjściową wartość funkcji f.

Innym możliwym rozwiązaniem jest:

1. Wyznaczenie sumy modulo 2 wejściowej 32-bitowej sekwencji z 32-bitami klucza.
2. Wartość wynikowa sumowania jest dzielona na cztery 8-bitowe części.
3. Następuje proces przetwarzania tak uzyskanych części przez 6 S-bloków.
4. Wartości wyjściowe S-bloków są sumowane modulo 2, tworząc 32-bitową wyjściową wartość funkcji f.

16-bitowe podklucze dla kolejnych cykli są wyznaczane w prosty sposób z 64-bitowego klucza. Jeśli k_1, k_2, ..., k_8 są kolejnymi bajtami klucza, to podkluczami dla poszczególnych cykli są:

Cykl 1: k_1 k_2
Cykl 2: k_3 k_4
Cykl 3: k_5 k_6
Cykl 4: k_7 k_8
Cykl 5: k_4 k_3
Cykl 6: k_2 k_1
Cykl 7: k_8 k_7
Cykl 8: k_6 k_5

Algorytm jest odporny na wszystkie znane ataki, jedynym sposobem na jego złamanie jest atak metodą brutalną.

Algorytm CAST jest opatentowany i dość popularny, przez Kanadę został oceniony jako nowy standard szyfrowania. Northern Telecom stosuje algorytm CAST w programowym pakiecie bezpieczeństwa Entrust przeznaczonym dla komputerów Macintosh, PC i uniksowych stacji roboczych. Algorytm ten jest również stosowany przez PGP (PGP 8.0.2 Personal).

5.3. 3DES

Liczne ataki na DES, pokazały, że DES jest zbyt słaby aby w dalszym ciągu stanowił standard szyfrowania. Przez wiele lat, nie było jednak następcy, szyfru który gwarantowałby znacznie większe bezpieczeństwo, szyfru którego bezpieczeństwo zostało potwierdzone. Próby modyfikacji DES, w celu zwiększenia jego siły kryptograficznej oraz eliminacji wad, nie przynosiły określonych rezultatów, zaś niektóre z nich powodowały nawet osłabienie szyfru.

Fakt ten spowodował wprowadzenie 3-DES (Triple-DES) jako standardu szyfrowania danych zapewniającego zwiększone w stosunku do DES bezpieczeństwo oraz odporność na atak z wyczerpującym poszukiwaniem klucza. Zapewnienie odporności na ten atak było niezwykle istotne, gdyż w 1998 grupa Electronic Foundation Frontier zbudowała maszynę łamiącą DES przy użyciu wyczerpującego poszukiwania klucza w trzy dni, przy nakładzie kosztów 250 tysięcy dolarów. Według badań tej grupy, przy nakładzie kosztów rzędu miliona dolarów, możliwe jest złamanie DES w ciągu 30 minut. 3DES umożliwia trzykrotne szyfrowanie bloku tekstu jawnego, przy użyciu trzech różnych kluczy, lub szyfrowanie przy użyciu dwóch różnych kluczy. W pierwszym przypadku, długość klucza została zwiększona do 168 bitów. Zwiększone zostało zatem bezpieczeństwo, jednakże znacznemu pogorszeniu uległa szybkość szyfru. W drugim przypadku efektywna długość klucza wynosi 112 bitów. Blok tekstu jest najpierw szyfrowany przy użyciu klucza K_1, następnie deszyfrowany przy użyciu klucza K_2, po czym znowu szyfrowany kluczem K_1. Wewnętrznie 3DES może pracować w trybach pracy określonych dla DES, przy czym możliwe są kombinacje. Przykładem może być TCBC (Triple DES Cipher Block Chaining). Złożoność ataku na ten tryb wynosi wówczas 2^{112}.

Dokładna analiza bezpieczeństwa poszczególnych trybów przeprowadzona przez E.Bihama, wykazała, że niektóre warianty 3DES są bezpieczniejsze od innych.

Algorytm ten można spotkać w standardzie PEM (Privacy Enhanced Mail) przyjętym przez Internet Architecture Board (IAB) jako standard ochrony poczty elektronicznej w internecie.

5.4. GOST

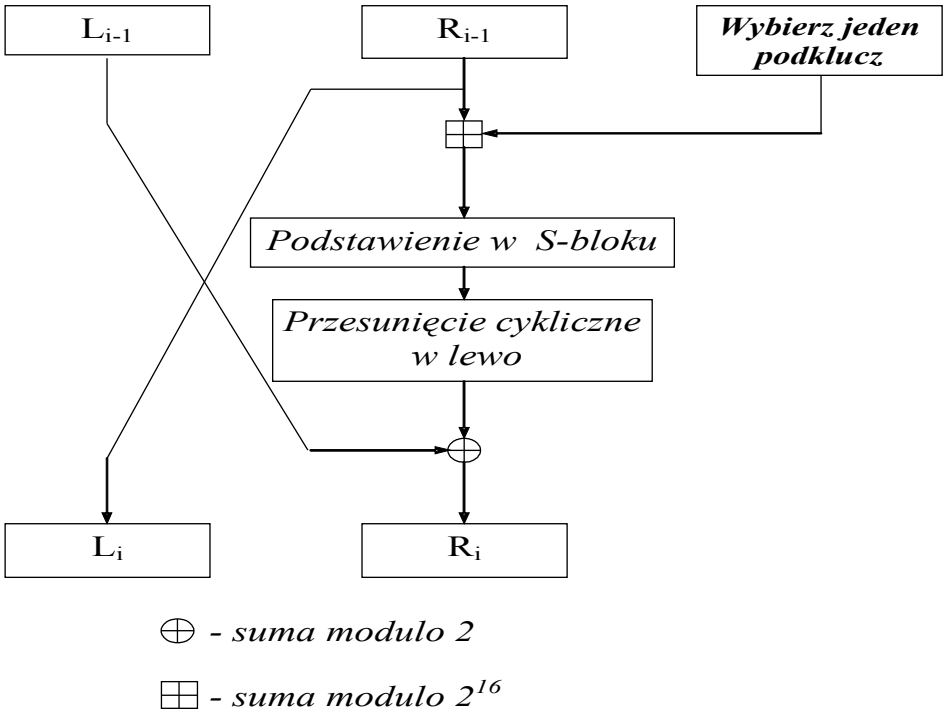
GOST jest algorytmem o blokach 64-bitowych z kluczem o długości 256 bitów. Szyfr powtarza prosty algorytm szyfrowania 32 razy. Przy szyfrowaniu najpierw blok tekstu jawnego jest dzielony na lewą połową L i prawą połowę R. Podkluczem cyklu i jest K_i. Algorytm wykonywany w i-tym cyklu szyfru GOST jest następujący:

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

Rysunek 5.3 przedstawia jeden cykl szyfru. Funkcja f jest prosta. Najpierw prawa połowa i i-ty podklucz są dodawane modulo 2^{16}. Następnie wartość wynikowa tej operacji jest dzielona na osiem części 4-bitowych, które z kolei są wartościami wejściowymi dla ośmiu S-bloków. Pierwsze cztery bity są przeznaczone dla pierwszego S-bloku, drugie cztery bity dla drugiego S-bloku itd. Każdy S-blok realizuje permutację zgodnie z zadanym dla niego wzorcem opisanym liczbami od 0 do 15. Przykładowy wzorzec może wyglądać następująco:

7, 10, 2, 4, 15, 9, 0, 3, 6, 12, 5, 13, 1, 8, 11

Dla tak skonstruowanego wzorca permutacji pojawienie się sekwencji wejściowej o wartości 0 spowoduje wygenerowanie przez S-blok sekwencji wyjściowej o wartości 7. Dla sekwencji wejściowej 1 sekwencja wyjściowa będzie miała wartość 10 itd. Każdy S-blok jest inny, a ich zawartość jest nieznana. Sekwencje wyjściowe poszczególnych S-bloków są łączone w słowo 32-bitowe. Następnie tak skonstruowana wartość jest przesuwana cyklicznie w lewo o 11 pozycji i sumowana modulo 2 z lewą połową bloku tekstu jawnego. Wynik sumowania jest nową prawą połową, a nową lewą połową staje się dotychczasowa prawa połowa. Opisany zestaw operacji jest wykonywany 32 razy.



Rys. 5.3. Jeden cykl algorytmu GOST

Sposób generacji podkluczy jest prosty. Polega na podziale 256 bitów klucza na osiem części o długości 32 bity: k_1, k_2, ..., k_8. W tabeli poniżej przedstawiono przydział kluczy dla poszczególnych cykli algorytmu. Proces deszyfrowania przebiega identycznie z tą tylko zmianą, że kolejność kluczy k_i jest odwrotna.

Cykl:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Podklucz:	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Cykl:	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Podklucz:	1	2	3	4	5	6	7	8	8	7	6	5	4	3	2	1

Tab. 5.1. Numery podkluczy przypisanych do poszczególnych cykli algorytmu GOST

W tabeli poniżej przedstawiono zalecane zawartości S-bloków (te wartości są również używane przez jednokierunkowe funkcje skrótu zalecane przez GOST).

S-blok 1	4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
S-blok 2	14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
S-blok 3	5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
S-blok 4	7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
S-blok 5	6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
S-blok 6	4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
S-blok 7	13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
S-blok 8	1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

Tab. 5.2. Struktura S-bloków algorytmu GOST

Algorytm GOST możemy spotkać w produkcie firmy Jetico – BestCrypt.

5.5. IDEA

W 1990 r. na konferencji Eurocrypt 90, X.Lai oraz J.L.Masses przedstawili projekt nowego szyfru blokowego, o długości bloku równej 64 bity oraz długości klucza głównego – K=128 bitów. Szyfr ten miał

zastąpić DES, co sugerowała nawet jego nazwa: PES - Proposed Encryption Standard, dlatego też głównymi przesłankami projektowymi szyfru były: prostota, identyczność procesu szyfrowania i deszyfrowania oraz łatwość w implementacji zarówno programowej jak i sprzętowej. Po wprowadzeniu (Eurocrypt 91) pewnych modyfikacji (polegającej na uproszczeniu permutacji na końcu cyklu) szyfr przyjął nazwę IPES (Improved PES), zaś później IDEA (International Data Encryption Standard).

Analizując budowę szyfru, przedstawioną na rysunku 5.4, widać, że ma on strukturę zaproponowaną przez Feistela, z pewnymi innowacjami. W przeciwieństwie do klasycznej struktury Feistela, gdzie blok tekstu jawnego o długości n=2t bitów dzielony jest na dwa bloki o długości t bitów, IDEA blok 64 bitów tekstu jawnego - X, na wstępie szyfrowania, dzieli na cztery mniejsze bloki o długości 16 bitów - X_1, X_2, X_3, X_4. Proces szyfrowania przebiega w 8 cyklach zakończonych transformacją wyjściową. Każdy cykl używa sześciu 16 bitowych kluczy cyklu $K_i^{(r)}$ (i=1...6, r – numer cyklu), uzyskanych w procesie rozszerzenia klucza głównego. Transformacja wyjściowa używa czterech kluczy cyklu Łączna ilość kluczy wygenerowanych w procesie rozszerzenia klucza głównego wynosi zatem 52.

Projekt szyfru bazuje na koncepcji łączenia operacji z trzech różnych algebraicznych grup i za ich pomocą realizowana jest warstwa mieszająca (confusion layer) oraz dyfuzyjna szyfru (diffusion layer). Schemat pojedynczego cyklu pracy przedstawia rys. 5.4.

Przebieg szyfrowania można opisać następująco:

1. Na początku z algorytmu rozszerzenia klucza generowane są 52 klucze - po 6 dla każdego cyklu i 4 dla transformacji końcowej.
2. 64-bitowy blok tekstu jawnego dzielony jest na cztery bloki 16-bitowe: X_1-X_{16} ;
   X_{17}-X_{32}...
3. Następnie przez kolejnych 8 cykli wykonywane są następujące operacje:

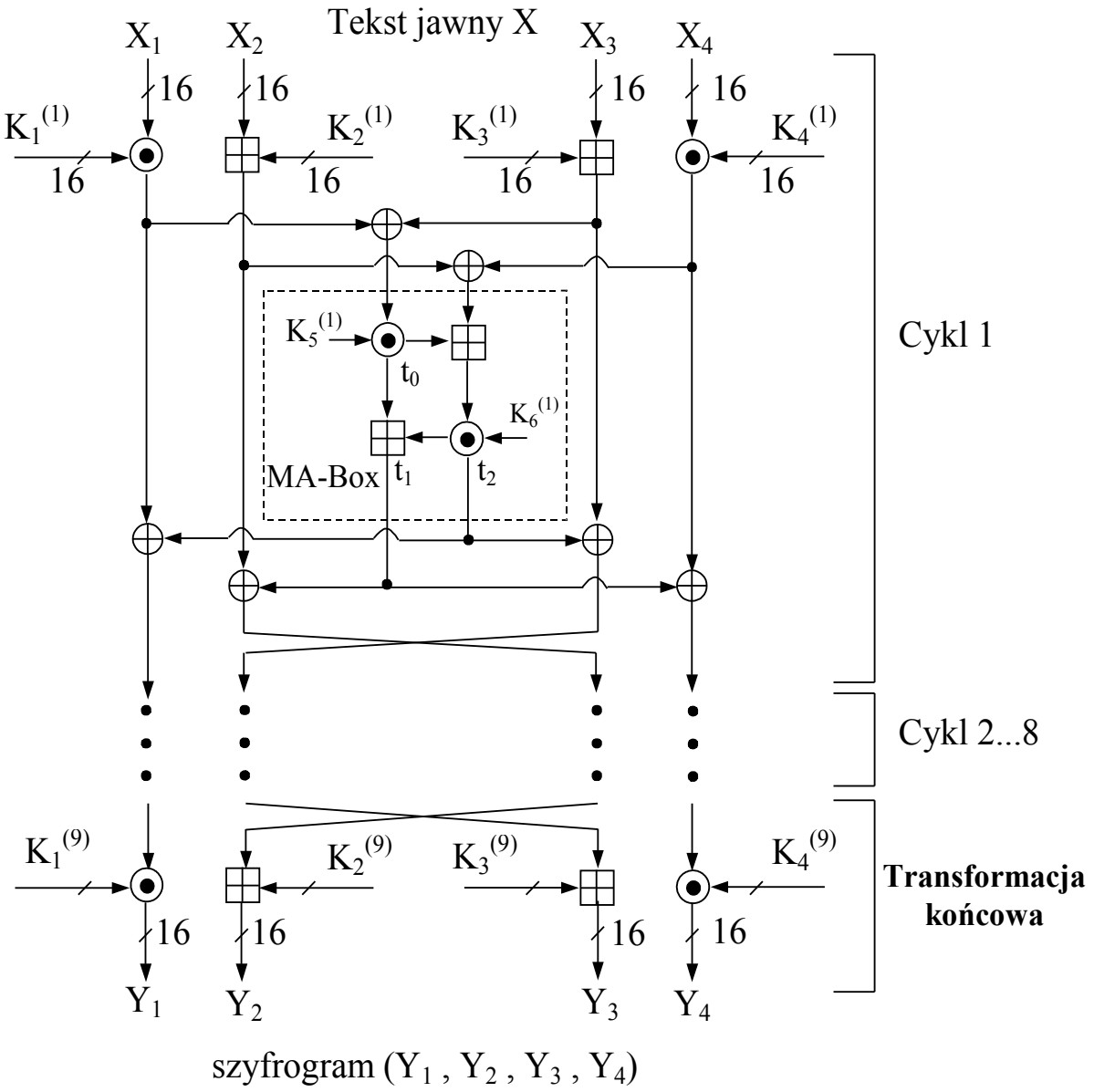
$$X_1 = X_1 * K_1^{(r)} \bmod 2^{16}+1$$
$$X_4 = X_4 * K_4^{(r)} \bmod 2^{16}+1$$
$$X_2 = X_2 + K_2^{(r)} \bmod 2^{16}$$
$$X_3 = X_3 + K_3^{(r)} \bmod 2^{16}$$
$$t_0 = K_5^{(r)} * (X_1 \text{ xor } X_3) \bmod 2^{16}+1$$
$$t_1 = K_6^{(r)} * (t_0 + (X_2 \text{ xor } X_4) \bmod 2^{16}) \bmod 2^{16}+1$$
$$t_2 = t_0 + t_1 \bmod 2^{16}$$
$$X_1 = X_1 \text{ xor } t_2$$
$$X_4 = X_4 \text{ xor } t_1$$
$$a = X_2 \text{ xor } t_1$$
$$X_2 = X_3 \text{ xor } t_2$$
$$X_3 = a$$

W opisie tym r jest numerem cyklu ($1 \le r \le 8$), zaś t oraz a są zmiennymi pomocniczymi.

4. Po powyższych ośmiu cyklach wykonywana jest końcowa transformacja zgodnie z algorytmem:

$$Y_1 = X_1 * K_1^{(9)} \bmod 2^{16}+1$$
$$Y_4 = X_4 * K_4^{(9)} \bmod 2^{16}+1$$
$$Y_2 = X_3 + K_2^{(9)} \bmod 2^{16}$$
$$Y_3 = X_2 + K_3^{(9)} \bmod 2^{16}$$

Operacje mnożenia w ciele GF(2^{16}+1) przebiegają zgodnie z arytmetyką ciała i założeniem, że elementowi 0 przyporządkowywana jest wartość 2^{16}.



Rys. 5.4. Cykl pracy algorytmu IDEA

Rozszerzenie klucza w IDEA odbywa się w następujący sposób: Klucz o długości 128 bitów $K = [k_1k_2...k_{128}]$, dzielony jest na 8 bloków po 16 bitów każdy. Z pierwszego bloku tworzony jest klucz $K_1^{(1)}$, z drugiego - $K_2^{(1)}$... z siódmego $K_1^{(2)}$, z ósmego bloku - klucz $K_2^{(2)}$. Następnie 128 bitowy klucz główny podlega cyklicznemu przesunięciu w lewo o 25 pozycji. Przesunięty klucz jest ponownie dzielony na osiem 16-bitowych bloków, które tworzą odpowiednio klucze $K_3^{(2)}$, $K_4^{(2)}$, $K_5^{(2)}$, $K_6^{(2)}$, $K_1^{(3)}$, $K_2^{(3)}$, $K_3^{(3)}$, $K_4^{(3)}$. Po wykonaniu tej operacji klucz jest ponownie przesuwany cyklicznie w lewo o 25 oraz dzielony na 8 bloków z których generowane są klucze dla poszczególnych cykli. Operacje przesunięcia są powtarzane do momentu wygenerowania wszystkich 52 kluczy cyklu.

Proces deszyfrowania przebiega w identyczny sposób - za pomocą tego samego algorytmu. Danymi wejściowymi są szyfrogramy, zaś na wyjściu otrzymujemy bloki tekstu jawnego. Istnieje tylko różnica w wartościach używanych kluczy:

- aby możliwe było poprawne deszyfrowanie, muszą one być stosowane w odwrotnej kolejności, oraz muszą stanowić albo multiplikatywną inwersję w ciele $GF(2^{16}+1)$, albo liczbę przeciwną w ciele $GF(2^{16})$, w zależności od operacji algebraicznej której składnik stanowi klucz. Poniżej została pokazana tabela kluczy cyklu dla deszyfrowania

Cykl r	$K'_1{}^{(r)}$	$K'_2{}^{(r)}$	$K'_3{}^{(r)}$	$K'_4{}^{(r)}$	$K'_5{}^{(r)}$	$K'_6{}^{(r)}$
r = 1	$(K_1^{(10-r)})^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$(K_1^{(10-r)})^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
$2 \le r \le 8$	$(K_1^{(10-r)})^{-1}$	$-K_3^{(10-r)}$	$-K_2^{(10-r)}$	$(K_1^{(10-r)})^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
r = 9	$(K_1^{(10-r)})^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$(K_1^{(10-r)})^{-1}$	...	...

Tab. 5.3. Klucze algorytmu IDEA dla poszczególnych cykli w procesie deszyfrowania

Znak – przed wartością oznacza liczbę przeciwną w ciele liczb $GF(2^{16})$, zaś K^{-1} to multiplikatywna odwrotność liczby w ciele $(2^{16}+1)$.

Bezpieczeństwo IDEA, oparte na odpowiedniej długości klucza, oraz odporność na różne rodzaje ataków a także prostota konstrukcji spowodowały, że IDEA stał się chętnie wykorzystywanym algorytmem. Algorytm IDEA można bardzo łatwo i efektywnie realizować na drodze programowej, dlatego też jest jednym z szyfrów używanych w protokole SSH, SSL czy też PGP. Bardzo dobre przyspieszenie szyfrowania IDEA, można uzyskać optymalizując IDEA pod kątem wykorzystania instrukcji MMX. Szybkość działania może zostać dodatkowo zwiększona poprzez implementacje równoległe. Cechy te czynią IDEA bardzo dobrym szyfrem dla aplikacji multimedialnych. Dzięki zastosowaniu operacji na 16 bitowych słowach, łatwo można zrealizować szyfrowania IDEA za pomocą rozwiązań sprzętowych. Istnieją na rynku układy scalone VLSI, które przy częstotliwości zegara wynoszącej 25MHz, potrafią szyfrować dane zgodnie z algorytmem IDEA z prędkością od 55Mbit/sec do blisko 180 Mbit/sec w zależności od rodzaju układu. Możliwe są również rozwiązania wykorzystujące procesory DSP.

5.6. RC2

Algorytm RC2 jest algorytmem szyfrującym o zmiennej długości klucza, zaprojektowanym przez Rona Rivesta dla RSA Data Security. RC2 jest szyfrem blokowym zaprojektowanym w celu zastąpienia algorytmu DES. Algorytm jednak nie został opublikowany zatem informacje o nim dostępne są szczątkowe. Oczywiście utajnienie algorytmu nie wpływa pozytywnie na jego popularność, wątpliwości budzi próba ukrycia szczegółów algorytmu, a w takiej sytuacji trudno bezkrytycznie ufać zapewnieniom twórcą algorytmu. Zgodnie ze stwierdzeniem firmy implementacje programowe algorytmu RC2 są trzy razy szybsze niż algorytmu DES.

Algorytm dopuszcza klucze zmiennej długości, od 0 bajtów do maksymalnej długości, jaką system komputerowy dopuszcza dla łańcuchów znakowych. Szybkość szyfrowania nie zależy od długości klucza. Klucz jest wstępnie przetwarzany i na jego podstawie powstaje tablica o 128 bajtach. Stąd liczba różnych kluczy jest równa 2^{1024}. Nie występują w nim S-bloki. Są tylko dwie operacje: mieszanie i rozdrabnianie i jedna z tych operacji jest wybierana w każdym cyklu.

5.7. RC4

RC4 został zaprojektowany przez Rona Rivesta dla RSA Data Security. Jest to algorytm strumieniowy o zmiennym rozmiarze klucza. Pracuje w trybie OFB – ciąg klucza jest niezależny od tekstu jawnego. Algorytm używa osiem grup po osiem bloków typu S-blok: S_0, S_1, ..., S_{255}. Wartościami wejściowymi są permutacje liczb od 0 do 255, a permutacje te są funkcją klucza o zmiennej długości. W algorytmie używa się dwóch liczników: i oraz j, początkowo zapełnionych zerami. Proces tworzenia losowego bajta można przedstawić następująco:

1. $i = (i + 1) \bmod 256$
2. $j = (j + S_j) \bmod 256$
3. *zamiana miejscami* S_i *i* S_j
4. $t = (S_i + S_j) \bmod 256$
5. $K = S_t$

Wynikowy bajt K jest sumowany modulo 2 albo z tekstem jawnym w procesie szyfrowania, albo z szyfrogramem w procesie deszyfrowania. Szyfrowanie algorytmem RC4 okazuje się dziesięciokrotnie szybsze niż przy użyciu algorytmu DES.

Proces wypełniania wartościami początkowymi S-bloków przebiega w prosty sposób. Na początku zapełnia się je kolejnymi liczbami całkowitymi począwszy od $S_0 = 0$ aż do $S_{255} = 255$. W następnym kroku zapełnia się inną tablicę 256-bajtową, używając klucza i powtarzając go tyle razy, ile jest to potrzebne do zapełnienia całej tablicy: K_0, K_1, ..., K_{255}. Indeks j ustawia się na 0 i wówczas:

```
For i = 0 to 255
j = (j + Si + Kj) mod 256
zamiana miejscami Si i Sj
```

Algorytm RC4 jest odporny na wszystkie znane ataki i jest stosowany w wielu komercyjnych programach. Algorytm ten stosują firmy Apple i Oracle Secure SQL w swych produktach, jest także częścią specyfikacji protokołu sieci komórkowych (Cellular Digital Packet Data).

5.8. Rijndael (AES)

Algorytm Rijndael jest symetrycznym szyfrem blokowym. Jego parametrami są:

- długość klucza:

  Rijndael może pracować z kluczami długości 128, 192 oraz 256 bitów. Dozwolone są również długości 160 oraz 224 bity, jednakże nie są one oficjalnie uznawane jako standard. Długość klucza można wyrazić jako N_K będącą liczbą 32 bitowych słów tworzących klucz. Dla długości 128 bitów N_K=4, dla 192 bitów N_K=6...

- wielkość bloku:

  Rijndael może pracować na blokach o długości 128, 192 oraz 256 bitów. Dozwolone są również długości 160 oraz 224 bity, jednakże nie są one również oficjalnie uznawane jako standard. Długość bloku można wyrazić jako N_B będącą liczbą 32 bitowych słów tworzących blok. Dla długości 128 bitów N_B=4, dla 192 bitów N_B=6.

Wszelkie transformacje Rijndaela przeprowadzane są na poziomie macierzy stanów, nazywanej State. Macierz ta zbudowana jest z czterech wierszy oraz z N_B kolumn. Wielkość tej macierzy zależy więc od długości bloku. Elementami macierzy są poszczególne bajty formułujące blok, oznaczane jako $S_{[r,c]}$, gdzie r jest numerem wiersza, zaś c numerem kolumny.

Ilość cykli N_R (iteracji) wykonywanych przez szyfr zależna jest zarówno od długości klucza jak i od długości bloku. Zależności te, dla wartości uznanych jako standardowe przedstawiono w poniższej tabeli.

Nr	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

Tab. 5.4. Liczba iteracji algorytmu AES

Liczbę cykli można też przedstawić za pomocą wzoru:

$$N_R = \max (N_K, N_B) + 6,$$

na którego podstawie łatwo można wyznaczyć liczbę iteracji dla niestandardowych wielkości bloku oraz klucza. Każdy cykl ma swój własny klucz, wygenerowany za pomocą algorytmu rozszerzenia klucza.

Przebieg szyfrowania

W pseudokodzie przebieg szyfrowania można opisać następująco:

```
Cipher(State, CipherKey)
    {
    KeyExpansion(CipherKey, RoundKey[NR+1]
    AddRoundKey(State, RoundKey[0])
    for round = 1 step 1 to Nr-1
        {
        SubBytes(State)
        ShiftRows(State)
        MixColumns(State)
        AddRoundKey(State, RoundKey[round])
        }
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, RoundKey[NR])

    }
```

Proces szyfrowania rozpoczyna się wygenerowania listy kluczy cyklu oraz sumowania XOR pierwszego klucza cyklu RoundKey[0] wraz z macierzą State. Następnie N_{R-1} razy wykonywane są kolejno cztery transformacje macierzy State. Ostatnia z nich stanowi sumę XOR tej macierzy i klucza cyklu o numerze określonym numerem iteracji. Po wykonaniu tego kroku wykonywana jest kolejna, ostatnia iteracja, różniąca się od pozostałych tym, że w jej trakcie macierz State nie podlega transformacji MixColumn.

Transformacjami macierzy State, wykonywanymi w procesie szyfrowania są cztery transformacje zdefiniowane i opisane jako:

1. SubByte jest transformacją, podstawiająca każdemu bajtowi macierzy stanów wartość określoną przez przekształcenie algebraiczne, które jest realizowane w dwóch krokach:

   1. wyliczenia multiplikatywnej inwersji bajtu, reprezentowanego w postaci wielomianu, w ciele $GF(2^8)$
   2. dokonania przekształcenia powyższej wartości zgodnie z afiniczną transformacją w ciele GF(2), zdefiniowaną przez następujące równanie

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

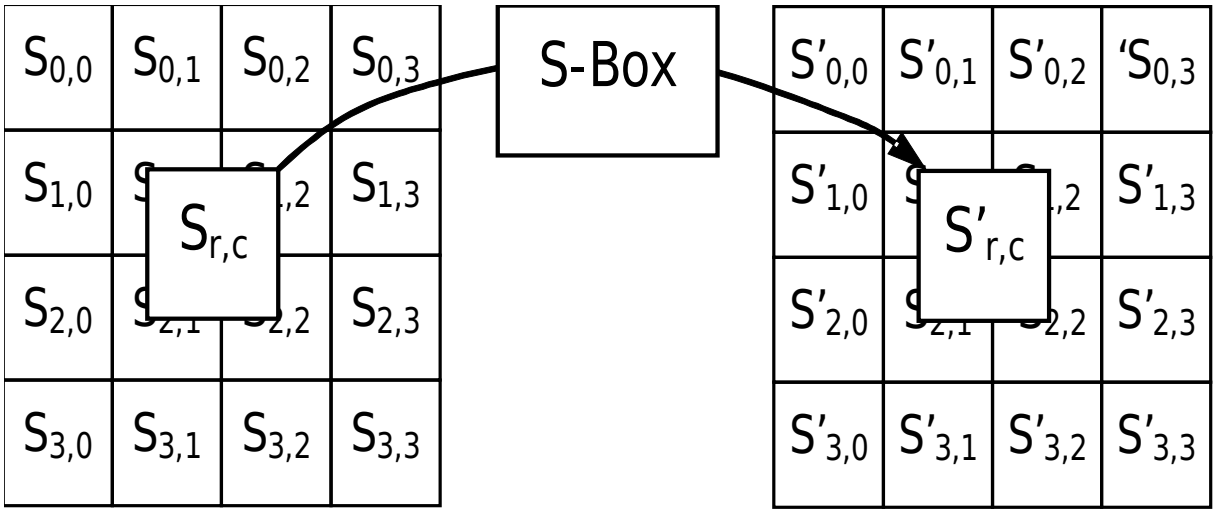
      dla $0 \le i < 8$ gdzie b_i jest i tym bitem bajta b zaś c_i jest i-tym bitem bajta 01100011.

Możliwe jest zatem wyliczenie powyższych przekształceń dla każdej z wartości bajtów i stworzenie odpowiedniej tablicy podstawień, określanej jako S-blok. Podstawienie odbywa się zgodnie z następującym schematem. Każdy bajt można opisać w kodzie heksadecymalnym jako $\{xy\}_H$. Aby odczytać wartość bajtu po dokonaniu transformacji odczytujemy odpowiedni wiersz zdefiniowany przez x, oraz odpowiednia kolumnę zdefiniowaną przez y. Przykładowo B={4F}, po transformacji SubByte przyjmie wartość równą B={84}. Efekt działania funkcji SubByte przedstawia poniższy rysunek:

Hex		**X**															
		0	1	2	3	4	5	6	7	8	9	A	b	c	d	e	f
Y	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76

	1	ca	82	C 9	7d	fa	59	47	f0	ad	d4	A2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	Cc	34	a5	E5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	D 6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	A a	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	C 4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	B8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	C 2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	C	ba	78	25	2e	1c	a6	b4	c6	E8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	F	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tab. 5.5. S-blok używany w transformacji SubByte



Rys. 5.5. Efekt działania transformacji SubByte

Transformacja SubByte jest transformacją odwracalną. Aby jednak odwrócenie było możliwe, operacje algebraiczne definiujące tą transformację muszą być również odwracalne. Transformacją odwrotną jest InvSubByte. Sprowadza się ona do dokonania inwersji na przekształceniu afinicznym, poprzedzonej wyliczeniem multiplikatywnej inwersji w GF(2^8).

Hex		X															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
y	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	A 1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	D 4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	F7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	C1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	E2	f9	37	e8	1c	75	df	6e
	A	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	B	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	C	1f	dd	A 8	33	88	07	c7	31	B1	12	10	59	27	80	ec	5f
	D	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	E	a0	e0	3b	4d	ae	2a	f5	b0	C8	eb	bb	3c	83	53	99	61
	F	17	2b	04	7e	ba	77	d6	26	E1	69	14	63	55	21	0c	7d

Tab.5.6. S-blok realizujący przekształcenie odwrotne w stosunku do SubByte

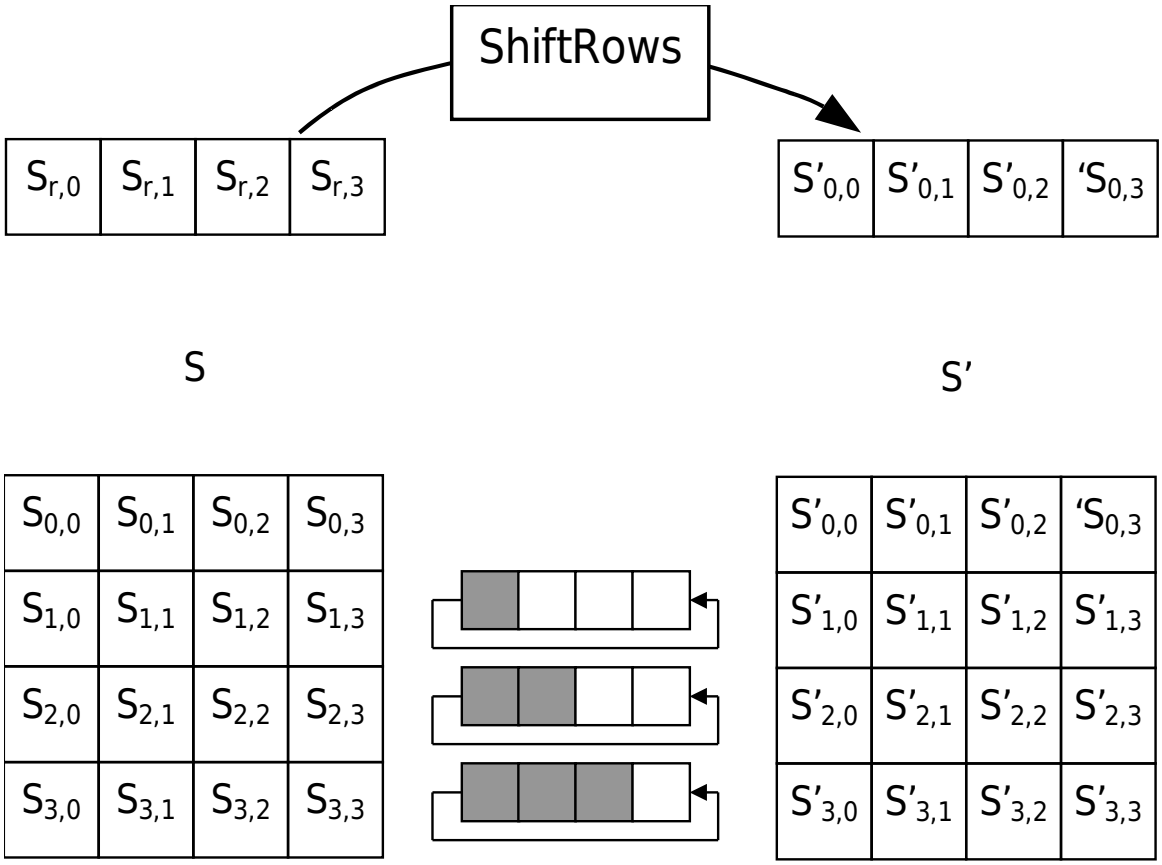
Operację odwrotną transformacji afinicznej można zapisać jako:

$$b_i' = {}_{(i+2) \bmod 8} \oplus\ b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus d_i$$

dla $0 \leq i < 8$ gdzie b_i jest i tym bitem bajta b zaś d_i jest i-tym bitem bajta

$d=05_x = 00000101$

2. W ShiftRow, wiersze macierzy State są cyklicznie przesuwane w prawo o określoną liczbę pozycji.



Rys. 5.6. Efekt działania transformacji ShiftRows

Wiersz 0 nie podlega przesunięciu. Wiersz pierwszy przesuwany jest o jedną pozycję w prawo - C1, wiersz drugi w zależności od wielkości bloku o dwie lub trzy pozycje - C2, zaś wiersz trzeci, przesuwany jest w prawo o 3 lub 4 pozycje - C3, również w zależności od długości bloku. Dokładne wartości przesunięć C pokazuje poniższa tabelka.

N_b	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	2	4

Tab. 5.7. Wartości przesunięć C

Dla długości bloków, które nie są długościami standardowymi, tj. dla $N_B = 5$, wartości przesunięć wynoszą: C1=1, C2=2, C3=3, zaś dla N_B=7: C1=1, C2=2, C3=4

Transformacja odwrotna InvShiftRows, stanowi również cykliczne przesunięcie. Dla InvShiftRows, liczba pozycji o które przesuwane są trzy ostatnie wiersze macierzy State, wynosi N_B-C1, N_B-C2 oraz N_B-C3. Bit z pozycji *j* wiersza numer *i* przesuwa się na pozycję $(j+N_B-C_i) \bmod N_B$.

3. W transformacji MixColumn tej kolumny macierzy State traktowane są jako wielomiany ze współczynnikami w ciele GF(2^8) i mnożone modulo x^4+1 przez wielomian:

$$a(x) = \text{‘03’}x^3 + \text{‘01’}x^2 + \text{’01’}x + \text{’02’}.$$

Niech $s'(x) = a(x) \bullet s(x)$, wówczas, w postaci macierzowej funkcję tą można opisać w sposób następujący:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

W rezultacie mnożenia otrzymujemy:

$$S'_{0,c} = \{02\} \bullet S_{0,c} \oplus \{03\} \bullet S_{1,c} \oplus S_{2},c \oplus S_{3,c}$$

$$S'_{1,c} = S_{1,c} \oplus \{02\} \bullet S_{1,c} \oplus \{03\} \bullet S_{2,c} \oplus S_{3},c$$

$$S'_{2,c} = S_{0},c \oplus S_{1,c} \oplus \{02\} \bullet S_{2,c} \oplus \{03\} \bullet S_{3,c}$$

$$S'_{3,c} = \{03\} \bullet S_{0,c} \oplus S_{1},c \oplus S_{2,c} \oplus \{02\} \bullet S_{3,c}$$

Operację odwrotną InvMixColumn, realizuje się poprzez mnożenie przez wielomian będący multiplikatywną inwersją wielomianu a(x) w skończonym ciele $GF(2^8)$. Wielomian ten, określony jako b(x) jest następującej postaci:

$$b(x) = \text{‘0B’}x^3 + \text{‘0D’}x^2 + \text{’09’}x + \text{’0E’}$$

Transformację InvMixColumn w postaci macierzowej można przedstawić jako:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 09 & 0d & 0b \\ 0b & 0e & 09 & 0d \\ 0d & 0b & 0e & 09 \\ 09 & 0d & 0b & 0e \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

co daje następujący zapis algebraiczny:

$$S'_{0,c} = \{0e\} \bullet S_{0,c} \oplus \{0b\} \bullet S_{1,c} \oplus \{0d\} \bullet S_2,c \oplus \{09\} \bullet S_{3,c}$$

$$S'_{1,c} = \{09\} \bullet S_{0,c} \oplus \{0e\} \bullet S_{1,c} \oplus \{0b\} \bullet S_2,c \oplus \{0d\} \bullet S_{3,c}$$

$$S'_{2,c} = \{0d\} \bullet S_{0,c} \oplus \{09\} \bullet S_{1,c} \oplus \{0e\} \bullet S_2,c \oplus \{0b\} \bullet S_{3,c}$$

$$S'_{3,c} = \{0b\} \bullet S_{0,c} \oplus \{0d\} \bullet S_{1,c} \oplus \{09\} \bullet S_2,c \oplus \{0e\} \bullet S_{3,c}$$

4. AddRoundKey dokonuje sumowania XOR klucza cyklu RoundKey z odpowiadającymi wartościami macierzy State. Klucz cyklu stanowi macierz o wymiarach identycznych z macierzą stanów State.
Matematycznie działanie funkcji AddRoundKey można opisać za pomocą równania.

$$[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = [S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \oplus [W_{l+c}],$$

gdzie wartość l stanowi iloczyn numeru cyklu oraz N_B, zaś bajty klucza - elementy macierzy reprezentującej klucz RoundKey, określone są jako W.

Transformacja odwrotna InvAddRoundKey jest identyczna, co wynika z własności funkcji XOR.

Deszyfrowanie

Zgodnie ze strategią proponowaną przez J.Daemena proces deszyfrowania polega na użyciu operacji odwrotnych do operacji szyfrowania, przy tych samych wartościach kluczy cyklu. Kolejność transformacji musi być również odwrotna. Przebieg procesu deszyfrowania można przedstawić w formie pseudokodu następująco:

```
Decipher(State, CipherKey)
    {
    KeyExpansion(CipherKey, RoundKey[NR+1]
    AddRoundKey(State, RoundKey[NR])
    for round = Nr-1 step 1 to 1
        {
        InvShiftRows(State)
        InvSubBytes(State)
        AddRoundKey(State, RoundKey[round])
        InvMixColumns(State)
        }
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, RoundKey[0])
    }
```

Algorytm Rijndael jest szeroko stosowany w komercyjnych produktach. Przykładami mogą być: BestCrypt 7.09, PGP 8.0.2 Personal, SafeGuard PrivateDisk 1.00, SafeHouse 2.10 czy TopSecret Next Generation.