

AKADEMIA TECHNICZNO - ROLNICZA W BYDGOSZCZY

Wydział Telekomunikacji i Elektrotechniki

PRACA MAGISTERSKA

Temat: Przegląd systemów szyfrowania informacji cyfrowych

Prowadzący: prof. dr inż. Antoni Zabłudowski

Autor: Rafał Siuda

Bydgoszcz 2002

Spis treści

Wstęp	2
1. Szyfry strumieniowe	9
1.1 Wiadomości wstępne	10
1.2 Budowa szyfrów strumieniowych	12
1.3 A5 jako przykład szyfru strumieniowego	13
2. Podstawowe wiadomości o szyfrach blokowych	17
2.1 Wiadomości wstępne	18
2.2 Tryby pracy szyfrów blokowych	18
2.3 Dopełnianie oraz Ciphertext stealing	24
3. Data Encryption Standard	26
3.1 Opis algorytmu	27
3.2 Tryby pracy DES	35
3.3 Triple-DES	35
3.4 Komentarze i obserwacje DES	36
3.5 Szybkość DES	39
3.6 Siła kryptograficzna i bezpieczeństwo	39
4. Inne szyfry blokowe	40
4.1 IDEA	41
4.2 RC5	48
4.3 RC6	54
5. Rijndael - Advanced Encryption Standard	63
5.1 Wstęp	63
5.2 Opis algorytmu	64
5.3 Przebieg szyfrowania	65
5.4 Opis poszczególnych transformacji	65
5.5 Deszyfrowanie	72
5.6 Aspekty i wskazówki dot. implementacji	72
5.7 Przesłanki projektowe	76
5.8 Wide Trail Strategy	79
5.9 Odporność i bezpieczeństwo szyfru	85
6. Systemy szyfrowania z kluczem publicznym	88
6.1 Wstęp	89
6.2 System RSA	91
7. Podstawowe ataki na szyfry	98
7.1 Wstęp	99
7.2 Techniki kryptoanalizy	100
7.3 Kryptoanaliza różnicowa	101
7.4 Kryptoanaliza liniowa	112
7.5 Analiza różnicowa poboru mocy	118
7.6 Atak bazujący na pomiarze czasu wyk. operacji	122
8. Implementacja programowa AES	126
8.1 Wstęp	127
8.2 Instrukcja obsługi programu	128
Literatura	129

Wstęp

Analizując procesy komunikacji międzyludzkich, jednoznacznie można stwierdzić, że ludzie, podczas komunikowania się ze sobą, wymieniają informację. Informacja ta może mieć wartość bezcenną, zwłaszcza w XXI wieku, w erze społeczeństwa informacyjnego. Oznacza to, że informacji tej, należy zapewnić maksymalną ochronę, bezpieczeństwo oraz dostępność dla ograniczonej liczby osób. Rozważając na temat bezpieczeństwa informacji można zadać pytanie: Czym jest tak naprawdę bezpieczeństwo informacji ?

Czy, jeżeli dowolny dokument umieścimy w bezpiecznym sejfie, ukrytym w dowolnym miejscu, o którym będzie wiedział tylko właściwy odbiorca tego dokumentu, czy będzie on bezpieczny ? Nie!, bezpieczny on będzie tylko z pozoru. Ale, jeśli ten sam dokument, umieścimy w sejfie, o którym zamieścimy wszelkie informacje projektowe, specyfikację zamków, a także udostępniemy setki innych identycznych sejfów z odpowiadającymi im kombinacjami zamków, w taki sposób, że najlepsi specjaliści będą mogli przeanalizować mechanizm zamka i nadal nie otworzyć tego jednego sejfu, wówczas możemy mówić o prawdziwym bezpieczeństwie.

Nauką zajmującą się zapewnianiem bezpieczeństwa informacji, z wykorzystaniem matematyki, jest kryptografia.

Kryptografia ma długą i zarazem fascynującą historię. Stosowana była już 4000 lat temu w starożytnym Egipcie, przez Juliusza Cezara - jednego z największych cesarzy Imperium Rzymskiego, poprzez okres średniowiecza, niebagatelną rolę odegrała w czasie II Wojny Światowej, znaczną rolę odgrywa czasach obecnych, w czasach w których można zaobserwować jej rozkwit. Do niedawna głównymi odbiorcami rozwiązań kryptograficznych było wojsko oraz instytucje rządowe. Rządy niektórych państw wydawały setki milionów dolarów na zapewnienie poufności i ochrony swych tajemnic. W ciągu ostatnich 20 lat, wraz z postępem technologicznym w dziedzinie elektroniki, telekomunikacji oraz informatyki, można zaobserwować gwałtowny rozwój kryptografii. Zaowocowało to tym, że ówczesne techniki kryptograficzne pozwalają współczesnemu człowiekowi chronić informację przez wiele lat, nawet przed najsilniejszymi rządami, instytucjami dysponującymi wręcz nieograniczoną mocą obliczeniową.

Czy przeciętny człowiek potrzebuje aż takiej ochrony? Odpowiedź brzmi: Tak. Może on planować kampanię polityczną, projektować nowy produkt, strategię marketingową czy też strategię przejęcie konkurencyjnej firmy. Może również żyć w kraju, w którym nie są respektowane prawa prywatności obywatela. Dla tych i wielu innych powodów, informacji



wymienianej w trakcie komunikacji międzyludzkich należy zapewnić prywatność, poufność oraz ochronę.

Cele te można osiągnąć tylko poprzez zaszyfrowanie jej, za pomocą wybranego szyfru i klucza. Niniejsza praca stanowi przegląd rozwiązań systemów szyfrowania informacji cyfrowych. . Przedstawione zostały systemy szyfrowania wykorzystujące:

- klucze symetryczne - przy stosowaniu szyfrów z takim kluczem, w celu zaszyfrowania i odszyfrowania wiadomości obie strony używają jednakowego, tajnego klucza. Do szyfrów takich, przedstawionych w niniejszym opracowaniu należą: A5, DES, 3DES, IDEA, RC5, RC6 oraz Rijndael. W przypadku DES, powstałego w latach 70. , uznanego za standard szyfrowania danych do roku 2000, długość klucza wynosi 64 bity (co oznacza zbiór 2^{64} możliwych kluczy). Możliwości obliczeniowe ówczesnych komputerów oraz zaawansowane techniki kryptoanalizy pokazują jednak, że długość ta okazuje się zbyt mała dla zapewnienia bezpieczeństwa informacji w dzisiejszych czasach, dlatego też, szyfr Rijndael uznany za standard szyfrowania danych od roku 2001, operuje na długościach kluczy wynoszących 128, 192 a nawet 256 bitów. Im dłuższy klucz, tym większy zbiór możliwych kluczy a tym samym większa złożoność obliczeniowa ataku polegającego na przeszukaniu wszystkich możliwych kluczy. Dla przykładu, przy wykorzystaniu miliona układów scalonych, z których każdy mógłby w ciągu 1 sekundy sprawdzić 1 milion kluczy, czas potrzebny na znalezienie właściwego klucza o długości 128 bitów wynosi ok 10^{19} lat.. Wadą takich systemów jest to, że klucz musi zostać przekazany/uzgodniony w kanale całkowicie bezpiecznym.
- klucze asymetryczne - stosując takie systemy, każdy z użytkowników generuje parę kluczy: klucz publiczny i klucz prywatny. Klucz publiczny udostępniany jest wszystkim użytkownikom i za jego pośrednictwem odbywa się szyfrowanie wiadomości, klucz prywatny, służący do odszyfrowywania wiadomości, pozostaje w tajemnicy. Fundamentem działania systemów z kluczem asymetrycznym są nierozwiązywalne problemy matematyczne, takie jak: faktoryzacja liczb całkowitych, problem obliczania pierwiastka kwadratowego w zbiorze liczb *modulo* n , czy też problem dyskretnych logarytmów. Do szyfrów należących do tej grupy, opisanych w niniejszej pracy należy szyfr RSA. Długość klucza w tych systemach wynosi 512, 1024 i, 2048 i więcej bitów. Im jest ona większa, tym większa jest liczba, której faktoryzacji należy dokonać w celu znalezienia klucza prywatnego. Współczesne algorytmy faktoryzacji (rozkładu na czynniki pierwsze) oraz moc obliczeniowa pozwalają na efektowny rozkład liczb 512

bitowych w stosunkowo krótkim czasie¹, dlatego też standardem na dzień dzisiejszy, oferującym odpowiednim poziom bezpieczeństwa jest długość klucza wynosząca 1024 bity. Systemy z kluczem asymetrycznym stanowią jeden z fundamentalnych mechanizmów działania podpisu elektronicznego.

Dodatkowo przedstawione zostały techniki ataków na szyfry przy użyciu różnych technik kryptoanalizy. Omówiona została kryptoanaliza różnicowa oraz kryptoanaliza liniowa na przykładzie szyfru DES - jedne z najbardziej zaawansowanych technik w ciągu ostatnich lat, pozwalające znacznie zredukować czas poszukiwań właściwego klucza. Działanie tych technik oparte jest na aproksymacji szyfru równaniem różnicowym lub liniowym. Siła kryptoanalizy liniowej i różnicowej stała się czynnikiem sprawczym i inspiracją do konstruowania nowych, odpornych na tego rodzaju atak systemów, takich jak Rijndael czy też RC6.

Pokazano możliwości ataku nie tylko na model matematyczny szyfru, lecz także na jego niewłaściwe implementacje, na przykładzie ataku bazującego na pomiarze poboru mocy układu realizującego szyfrowanie/desyfrowanie, czy też atak bazujący na pomiarze czasu wykonywanych operacji. Okazuje się, że informacje takie, znacznie mogą ułatwić, a nawet pozwolić w ciągu kilku godzin, za pośrednictwem stosunkowo niewielkich nakładów finansowych znaleźć właściwy klucz.

Siła ówczesnych algorytmów szyfrujących, ich odporność na techniki kryptoanalizy, powoduje, że w wielu krajach stosowanie tak zaawansowanych metod kryptograficznych jest zabronione, gdyż gwarantuje ochronę danych nawet przed organami ścigania, co uniemożliwia dokonywanie skutecznej rewizji w sieciach komputerowych, co może zostać wykorzystywane w celach przestępczych. Ograniczeniu możliwości stosowania metod kryptograficznych sprzeciwiają się gwałtownie środowiska biznesowe. W chwili obecnej, jednym z niezbędnych warunków rozwoju gospodarczego państwa, jest rozbudowa globalnej sieci komputerowej, pełniącej funkcję infrastruktury informacyjnej gospodarki. Jedyną drogą do gwarancji bezpieczeństwa wiedzie poprzez stosowanie technik kryptograficznych.

¹ Liczba 512 bit, - RSA 155 o długości 155 cyfr, została rozłożona na czynniki pierwsze w ciągu 5 miesięcy za pomocą algorytmu GNFS.
Zob. <http://www.rsasecurity.com/rsalabs/challenges/factoring/rsa155.html>

Rozdział 1

Szyfry strumieniowe

Jeśli transformacją łączącą h strumień klucza wraz ze strumieniem tekstu jawnego jest funkcja XOR, wówczas błędy bitowe w pewnych pozycjach szyfrogramu, będą miały wpływ tylko na odpowiednie pozycje tekstu jawnego. Nie ma zatem w tym przypadku propagacji błędów. Wadą tego rozwiązania jest konieczność zapewnienia pełnej synchronizacji pomiędzy nadawcą a odbiorcą. W przypadku jej braku odtworzenie tekstu jawnego stanie się dla odbiorcy niemożliwe. Aby zminimalizować wpływ utraty synchronizacji, stosuje się różne techniki takie jak sekwencje inicjujące w przypadku redundancji tekstu, numerację ramek. itp.

1.1.2 Asynchroniczne szyfry strumieniowe

Asynchroniczne szyfry strumieniowe, nazywane także samo-synchronizującymi szyframi strumieniowymi, należą do grupy szyfrów, w których generowany strumień klucza jest zależny od wartości klucza oraz od wartości szyfrogramów poprzednich. Ilość szyfrogramów wpływających na wartość strumienia klucza, musi być stała w obszarze działania całego szyfru. Proces szyfrowania za pomocą synchronicznych szyfrów strumieniowych można opisać za pomocą równań:

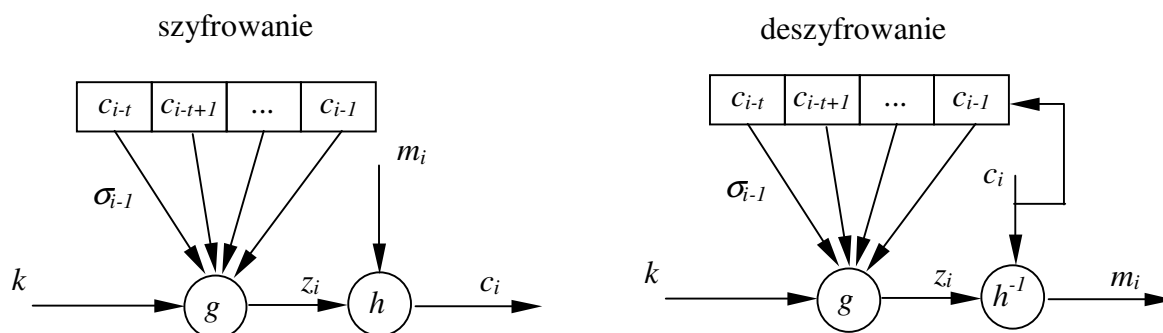
$$\sigma_i = f(c_{i-t}, c_{i-t+1}, c_{i-t+2}, \dots, c_{i-1}),$$

$$z_i = g(\sigma_i, k),$$

$$c_i = h(z_i, m_i)$$

gdzie: σ_i jest stanem szyfru w i -tej chwili, f jest funkcją opisującą wpływ szyfrogramów na stan szyfru, g jest funkcją generującą strumień klucza, zaś h jest funkcją łączącą strumień klucza z_i wraz z tekstem jawnym m_i w celu wytworzenia szyfrogramu c_i .

Zasadę szyfrowania i deszyfrowania za pomocą synchronicznych szyfrów strumieniowych można też przedstawić za pomocą poniższego rysunku.



Rys. 1.2. Model asynchronicznego szyfru strumieniowego.

Analizując zasadę działania asynchronicznych szyfrów strumieniowych widać, że są to szyfry z pamięcią, stanowiącą bufor dla szyfrogramów stanów poprzednich. Implementując tego rodzaju szyfry pojawia się również problem polegający na wygenerowaniu strumienia klucza w pierwszych *t-chwilach*. Jedną z technik rozwiązania tego problemu może być użycie dla pierwszych *t-chwil* wektora inicjalizującego *IV*.

Jedną z podstawowych własności tych szyfrów jest samosynchronizacja. Szyfry te po utracie synchronizacji zdolne są do jej automatycznego przywrócenie, ponieważ proces deszyfrowania zależy tylko od *t* stanów poprzednich, a nie jak w przypadku szyfrów synchronicznych od wszystkich stanów. Zależność od *t* stanów poprzednich korzystnie wpływa również na propagację błędów, która jest w tym przypadku ograniczona do *t* szyfrogramów. Kolejną zaletą są dobre własności statystyczne. Ponieważ każdy bit tekstu jawnego wpływa na wartość szyfrogramu, statystyczne własności tekstu jawnego ulegają rozrzuceniu w obrębie wszystkich szyfrogramów, dlatego też szyfry te są odporniejsze na ataki bazujące na redundancji tekstu jawnego.

1.2 Budowa szyfrów strumieniowych

Podstawowym elementem używanym do generowania strumienia klucza są rejestry liniowe ze sprzężeniem zwrotnym LFSR. Na fakt ten składa się kilka przyczyn:

- są one łatwe w implementacji,
- mogą generować sekwencje pseudolosową o stosunkowo długim okresie,
- generowane przez nie sekwencje mają bardzo dobre własności statystyczne,
- z uwagi na ich strukturę, możliwa jest ich analiza za pomocą technik algebraicznych.

Za pomocą algorytmu Berlekampa-Massey można dokonać analizy pracy rejestru LFSR, wyznaczyć wartość wielomianu definiującego sprzężenia oraz okresu. W praktyce oznacza to możliwość łatwej predykcji wartości bitów strumienia klucza. Możliwe jest jednak połączenie wyjść kilku rejestrów za pomocą określonej funkcji boolowskiej i a jej pomocą generowania strumienia klucza, użycia wyjścia jednego z rejestrów w celu taktowania innego oraz wiele innych kombinacji. Dzięki takim zabiegom drastycznie zwiększa się odporność na ataki oparte na korelacjach, oraz na opisie matematycznym pracy szyfru.

Układ zbudowany jest z trzech liniowych rejestrów cyklicznych, ze sprzężeniem zwrotnym. Pierwszy rejestr - R1 ma długość 19 bitów, drugi - R2 - długość 22 bity, zaś trzeci - R3 - długość 23 bitów. Najmłodszy bit każdego z rejestrów jest oznaczony jako zero.

Każdy z trzech rejestrów tworzy generator sekwencji pseudolosowej. Oprócz tego, że rejestry są różnej wielkości, mają również różne punkty wyprowadzeń sygnałów sprzężenia zwrotnego. Rejestr R1 ma je w pozycjach 13; 16;17;18, rejestr R2 w pozycjach 20 i 21, rejestr R3 w pozycjach 7,20,21,22. Punkty te zostały tak, aby okresy sekwencji pseudolosowych generowanych przez poszczególne rejestry były jak najdłuższe. Wynoszą one odpowiednio $2^{19}-1$, $2^{22}-1$ oraz $2^{23}-1$.

Taktowanie rejestrów odbywa się według następującej reguły start/stop:

Każdy z rejestrów posiada punkt wyprowadzenia sygnału sterującego taktowaniem. Dla rejestru R1 jest to punkt C1, znajdujący się na pozycji 8, dla R2 - punkt C2 na pozycji 10, oraz dla R3 - punkt C3 na pozycji 10. W każdym cyklu analizowana jest zawartość bitów C1, C2, C3. Jeżeli dwa z trzech bitów C mają wartość '1', taktowane są tylko te rejestry, w których bity C miały wartości '1'. Jeżeli dwa z trzech bitów C miały wartość '0', taktowane są tylko te rejestry, których bity C miały wartość '0'. Funkcją realizującą taktowanie jest więc funkcja większościowa (*majority function*). W każdym kroku taktowane są zatem przynajmniej dwa rejestry.

Proces generowania sekwencji pseudolosowej na podstawie klucza sesji - K , oraz wartości licznika ramki F_n realizowany jest w czterech krokach.

1. Najpierw wszystkie rejestry są zerowane, a następnie taktowane są 64 cyklami zegarowymi z pominięciem reguły start/stop. W czasie tym, każdy bit klucza K (od LSB do MSB) jest wprowadzany równoległe , za pomocą xor do trzech rejestrów, w miejsce LSB.
2. W kolejnych 22 cyklach zegara (również z pominięciem reguły start/stop) wprowadzane są 22 bity licznika ramki F_n , w sposób identyczny jak powyżej. Na końcu 22 cyklu zawartość trzech rejestrów tworzy wektor inicjujący ramki,
3. Rejestry są taktowane 100 cyklami zegara z uwzględnieniem reguły start/stop. W czasie tym dane wyjściowe rejestrów są ignorowane.
4. Rejestry są taktowane przez kolejne 228 cykli z uwzględnieniem reguły start/stop w celu wygenerowania 228 bitów wyjściowych. W każdym cyklu zegara generowany jest jeden bit, będący sumą xor trzech najstarszych bitów (MSB) rejestrów.

1.3.1 Ataki i obserwacje A5

Implementacje A5 stanowią tajemnice operatorów GSM, jednak na początku 1998 Smartcard Developer Association opublikowała ich kody źródłowe zdobyte w wyniku *reverse-engineeringu*. Kryptoanaliza A5/2 została dokonana w 1998 roku przez Slobodana Pertovica², za pomocą której on pokazał atak na A5/2 opierający się na algebrze, o złożoności 2^{17} .

Na wstępie zakłada się, że atakujący kryptoanalityk zna wartości kilku pseudolosowych bitów generowanych przez A5/1 w różnych ramkach. Powyższe założenie stanowi fundamentalne założenie wstępne kryptoanalizy szyfrów strumieniowych. Zakłada się również, że atakujący ma do dyspozycji fragment sekwencji wyjściowej A5 - szyfrogram (będący zresztą fragmentem rozmowy) i jego celem jest znalezienie klucza sesji umożliwiającego odszyfrowanie pozostałej części rozmowy. Ponieważ w GSM każda ramka ma czas trwania równy 4,6ms zatem w jednej sekundzie przesyłanych jest około 220 ramek.

Bezpieczeństwo A5 zostało przeanalizowane w różnych publikacjach. Wyniki tej analizy można podsumować następująco:

1. R.Anderson i M.Roe³ zaproponowali atak bazujący na zgadywaniu 41 bitów z krótszych rejestrów - R1 oraz R2 oraz wyliczeniu 23 bitów R3 poprzez analizę wartości wyjściowej. Należy jednak wziąć pod uwagę wartości bitów funkcji większościowej, co powoduje, że łączna złożoność ataku wynosi 2^{54} . Zakładając, że standardowy komputer PC potrafi przetestować 10 mln bitów w sekundzie, atak ten wymaga ponad miesiąca czasu w celu odnalezienia klucza, co czyni jego beżytecznym.
2. M.Briceno⁴ analizując A5, odkrył, że w każdej z jego implementacji 10 najmłodszych bitów 64-bitowego klucza ma wartość zerową. Dzięki temu złożoność ataku z wyczerpaniem wszystkich możliwości klucza redukuje się z 2^{64} do 2^{54} .
3. Podczas Eurocrypt 97. J.Golic przedstawił atak⁵ typu „dziel i zwyciężaj”, mający na celu uzyskanie nieznannej wartości wektora inicjującego rejestrów poprzez wykorzystanie specyficznej reguły taktowania rejestrów oraz rozwiązanie systemu równań. Do przeprowadzenia ataku Golic potrzebował jedną parę tekstu jawnego i szyfrogramu. Atak charakteryzuje się złożonością ok. 2^{40} .

² S.Petrovic, A.Fuster-Sabater, *Cryptoanalysis of the A5/2 Algorithm*, Crypto 98, Springer-Verlag 1998

³ Zob. R. Anderson, M. Roe, A5, <http://jya.com/crack-a5.htm>, 1994.

⁴ Zob. M. Briceno, I. Goldberg, D. Wagner, A pedagogical implementation of A5/1, <http://www.scard.org/>, May 1999.

⁵ J.D. Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, Eurocrypt 97, Springer Verlag 1998

4. Kolejnym atakiem przedstawionym przez J.Golica był atak zwany Time Memory Trade Off Attack. bazujący na probabilistycznym problemie urodzin. Golic pokazuje, że możliwe jest znalezienie klucza gdy $TM > 2^{63}$, gdzie T jest czasem obliczeń, zaś M wielkością pamięci mierzoną w 128 bitowych słowach. Biorąc jednak pod uwagę możliwości współczesnych komputerów atak ten również jest bezużyteczny.

5. W 1999 roku A.Biryukov, A.Shamir oraz R.Wagner⁶ na podstawie powyższych obserwacji, oraz poprzez zauważenie, że A5/1 może zostać łatwo zaimplementowany na domowym PC, oraz poprzez zastosowanie algorytmu redukcji odczytów z dysku w kryptoanalizie J.Golica zaproponowali dwa nowe ataki, które mogą zostać przeprowadzone z użyciem komputera klasy PC. Celem tych ataków jest wydobycie klucza sesji. Pierwszy z nich, zwany *Biased Birthday Attack*, mając do dyspozycji 2 min rozmowy oraz komputer z macierzą dyskową o pojemności 150Gb potrafi wyciągnąć klucz w czasie 1 sekundy. Drugi atak - *Random Subgraph Attack*, dokonuje ekstrakcji klucza w czasie jednej minuty, mając do dyspozycji PC z pojemnością dyskową 300Gb oraz 2 sekundy rozmowy. Obydwa ataki wymagają jednak obliczeń wstępnych, wykonywanych jednorazowo, w liczbie 2^{48} kroków.

⁶ A.Biryukov, A.Shamir, R. Wagner, Real Time Cryptanalysis of the Alleged A5/1 on a PC, Proceedings of fast Software Encryption Workshop, NewYork 2000, Lecture Notes on Computer Science, Berlin, in press

Rozdział 2

Podstawowe wiadomości o szyfrach blokowych

2.1 Wiadomości wstępne

Szyfr blokowy jest szyfrem przekształcającym n -bitowe bloki tekstu jawnego w n -bitowe bloki zwane szyfrogramami. Parametrem mającym wpływ na to przekształcenie jest k -bitowy klucz. Szyfry blokowe mogą być szyframi pracującymi z kluczem symetrycznym oraz asymetrycznym. Rozdział ten zostanie poświęcony szyfrowaniu z kluczem symetrycznym. Ponieważ w szyfrach tych, używa się jednego klucza do szyfrowania i deszyfrowania danych, by możliwy był proces poprawnego deszyfrowania, funkcja szyfrująca (szyfr) musi być funkcją odwracalną, co oznacza, że dla n -bitowych tekstów jawnych i szyfrogramów oraz k -bitowego klucza, funkcja ta stanowi bijekcję, dokonującą permutacji na n -bitowych wektorach.

O ile szyfry strumieniowe przekształcają pojedyncze bity tekstu jawnego, szyfry blokowe operują na blokach wielkości $n=64, 128, 192$ oraz 256 bitów. Wielkości bloków mogą ulec dalszemu zwiększeniu - ograniczenie stanowią tu jedynie możliwości optymalnej implementacji szyfru.

2.2 Tryby pracy szyfrów blokowych

Każdy z szyfrów blokowych może pracować w różnych trybach pracy. Niektóre z tych trybów pozwalają naśladować szyfry strumieniowe. Zastosowanie różnych trybów pracy pozwala osiągnąć pożądane czasem własności dla systemu w którym szyfry mają być implementowane (np. zależność od poprzednich bloków szyfrogramów). Czynniki determinującymi wybór określonego trybu pracy są następujące kryteria:

- żądany poziom bezpieczeństwa - niektóre z trybów pracy szyfrów oferują podwyższone bezpieczeństwo w stosunku do innych⁷.
- wielkość klucza - w rozważaniu ataku polegającym na wyczerpującym poszukiwaniu klucza, entropia przestrzeni kluczowej, stanowi górny margines bezpieczeństwa szyfru.
- przepustowość (*throughput*) - szybkość pracy. Czynnikiem ten jest związany z złożonością operacji kryptograficznych oraz jakością implementacji szyfru na określonej platformie.

⁷ Bezpieczeństwo szyfru stanowi zagadnienie bardzo złożone - można jednak uznać szyfr za bezpieczny, kiedy jego bezpieczeństwo zostało potwierdzone poprzez dokonanie kilku rodzajów kryptoanaliz. Dla niektórych trybów pracy kryptoanaliza może stanowić bardziej złożony problem niż dla innych, co sugeruje, że tryb taki można uznać za bezpieczny

- wielkość bloku - czynnik ten wpływa zarówno na bezpieczeństwo, jak i na złożoność oraz implementację. Może mieć również wpływ na przepustowość.
- złożoność operacji kryptograficznych - mająca swoje odzwierciedlenie w przepustowości, łatwości implementacji algorytmu oraz bezpieczeństwie.
- propagacja błędów - deszyfrowanie błędnego szyfrogramu może wpłynąć zawartość otrzymanego tekstu jawnego nie tylko odpowiadającego bloku, ale także innych. Niektóre z trybów pracy zapewniając większe bezpieczeństwo, charakteryzują się większą propagacją błędów

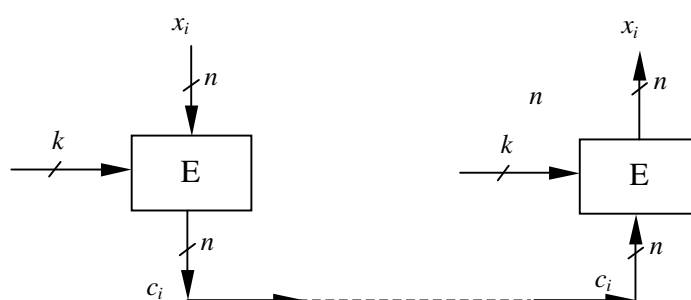
2.2.1 Tryb ECB - elektroniczna książka kodowa

W trybie ECB (*electronic code book*) każdy blok tekstu jawnego szyfrowany jest oddzielnie i niezależnie od innych bloków. Proces szyfrowania i deszyfrowania można zatem opisać następującymi równaniami:

$$c_i = E_K(x_i)$$

$$x_i = E_K^{-1}(c_i)$$

Zasadę tę można przedstawić za pomocą poniższego rysunku:



Rys. 2.1 Zasada szyfrowania w trybie ECB.

Tryb ten jest najprostszym z trybów pracy. Charakteryzuje się on następującymi własnościami:

- identyczne bloki tekstu jawnego, szyfrowane przy użyciu identycznego klucza, dają w wyniku szyfrowania identyczne szyfrogramy, ponieważ każdy blok jest szyfrowany niezależnie od innych.
- konsekwencją powyższego jest fakt, błąd w *i*-tym bloku tekstu jawnego powoduje pojawienie się błędu tylko w *i*-tym bloku szyfrogramu. Nie ma więc znacznej propagacji błędów bitowych na inne bloki.
- wielkość danych wejściowych musi być równa wielkości bloku, gdyż w przeciwnym razie szyfrowanie oraz deszyfrowanie nie będzie możliwe

Ponieważ każdy blok jest szyfrowany niezależnie od innych, tryb ten idealnie nadaje się do implementacji z wykorzystaniem przetwarzanie równoległego. Analizując działanie ECB można zauważyć, nie jest on bezpiecznym trybem, dlatego też nie zaleca się jego stosowania do szyfrowania dłuższych wiadomości.

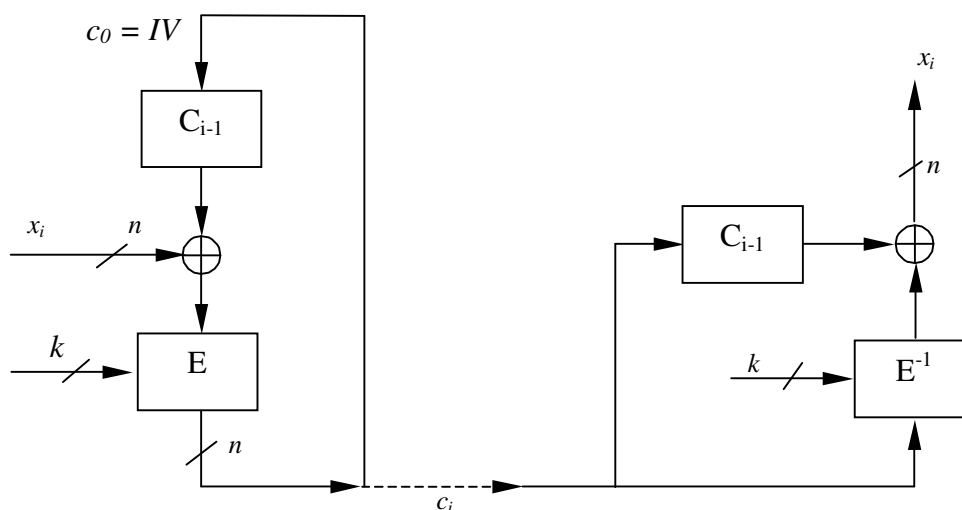
2.2.2 Tryb CBC - szyfrowanie z wiązaniem bloków szyfrogramu

Tryb CBC (*cipher-block chaining*) charakteryzuje się zależnością wartości szyfrogramu c_i nie tylko od wartości tekstu jawnego x_i i klucza, ale także od wartości szyfrogramów poprzednich. W ujęciu matematycznym CBC można opisać za pomocą następujących równań:

$$c_i = E_K(c_{i-1} \oplus x_i)$$

$$x_i = E_K^{-1}(c_i) \oplus c_{i-1}$$

Opis CBC w postaci graficznej można przedstawić w sposób następujący:



Rys.2.2 Zasada szyfrowania w trybie CBC.

Zastosowanie trybu CBC pociąga za sobą, dla stanu zerowego, konieczność użycia wektora inicjującego IV o długości równej długości bloku. Tryb ten jest pozbawiony podstawowej wady ECB - identycznych szyfrogramów dla identycznych tekstów jawnych. Poprzez wprowadzenie łańcuchowej zależności w CBC identyczne teksty jawne dają różne szyfrogramy na wyjściu. Rozwiązanie to zwiększa radykalnie bezpieczeństwo działania szyfru, jednakże posiada też następujące wady:

- proces deszyfrowania musi odbywać się w ściśle wyznaczonej kolejności, gdyż w przeciwnym razie poprawne deszyfrowanie nie jest możliwe,

- CBC ma również gorsze własności propagacji błędów - pojedynczy błąd w transmisji szyfrogramu c_i wpływa nie tylko na poprawność odpowiadającego mu tekstu jawnego x_i , ale także na poprawność tekstu jawnego x_{i+1} otrzymanego w procesie deszyfrowania szyfrogramu kolejnego c_{i+1} . Fakt ten implikuje możliwość predykcji zmian w tekście jawnym x_{i+1} poprzez celową zamianę bitów w szyfrogramie c_i .
- Opisana zależność łańcuchowa wyklucza możliwość efektywnej implementacji szyfrowania CBC w systemach z przetwarzaniem równoległym oraz w systemach z losowym dostępem typu zapis/odczyt do danych. Możliwe jest jednak wykorzystanie implementacji równoległych do deszyfrowania danych.

Podobnie jak w ECB wielkość danych wejściowych musi być równa długości bloku, bądź jego krotnością, gdyż w przeciwnym wypadku poprawny proces szyfrowania nie jest możliwy. Ponieważ CBC do generacji pierwszego szyfrogramu używa wektora inicjującego, należy zapewnić integralność tego wektora, gdyż jakiegokolwiek zmiany wartości wektora mogą prowadzić do obserwacji ich wpływu na zawartość szyfrogramu, co ułatwia ewentualna kryptoanalizę. Nie jest natomiast konieczne utrzymywanie wektora inicjalizującego IV w tajemnicy.

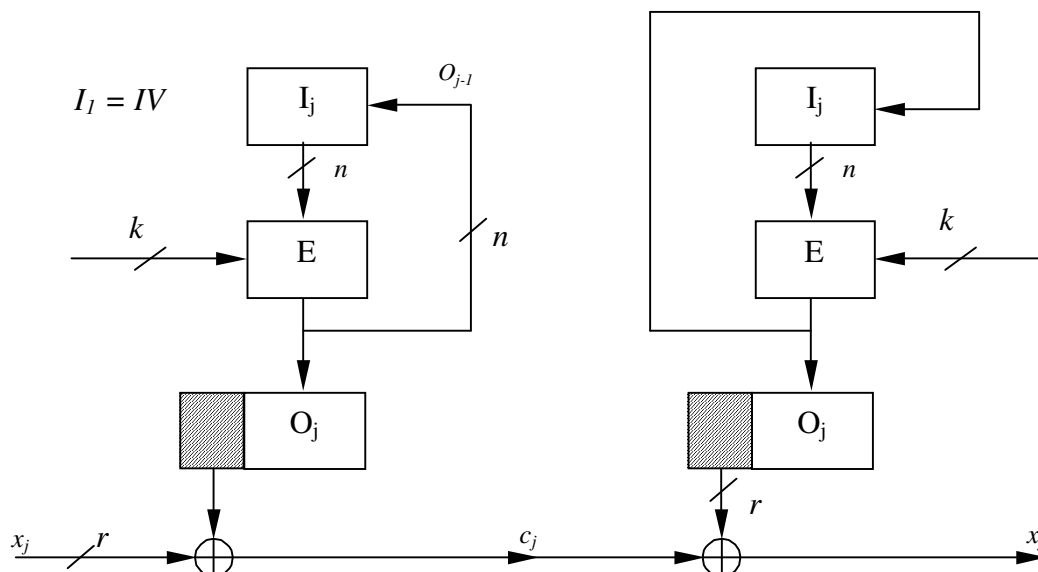
2.2.3 Tryb CFB - szyfrowanie ze sprzężeniem zwrotnym w postaci szyfrogramów

Analizując pracę powyższych trybów, można zauważyć, że dane wyjściowe w postaci szyfrogramu pojawiają się na wyjściu z opóźnieniem zależnym od czasu szyfrowania pojedynczego bloku. Niektóre aplikacje wymagają jednak, aby pewna ilość bitów - r , z n -bitowego bloku tekstu jawnego, była zaszyfrowana natychmiast, bez widocznego opóźnienia. Możliwość taką daje zastosowanie trybu CFB (*cipher feedback*). Proces szyfrowania dla r -bitowego sprzężenia zwrotnego można opisać następująco:

- (1) - w rejestr I_j wpisywana jest zawartość wektora inicjującego IV
- (2) - w rejestr O_j wpisywana jest wartość odpowiadająca zaszyfrowanej wartości rejestru I_j
- (3) - t_j bitów z rejestru O_j (t_j najstarszych bitów O_j) oraz r bitów tekstu jawnego poddawane jest operacji sumy XOR w celu wytworzenia pierwszego szyfrogramu c_1 .
- (4) - zawartość rejestru I_j jest przesuwana o r bitów, w miejsce najmłodszych bitów wczytywane jest r bitów szyfrogramu

ISO10116, oraz niepełnym (r -bitowym ; $r < n$) sprzężeniem zwrotnym, zdefiniowaną przez FIPS81. W obu przypadkach konieczne jest stosowanie wektora inicjującego IV .

Zasadę działania trybu OFB przedstawia rysunek:



Rys.2.4 Zasada szyfrowania w trybie OFB.

W trybie OFB, strumień klucza jest niezależny od zawartości tekstu jawnego. Tak jak w trybie CFB zmniejszona jest szybkość działania szyfru o wartość n/r . Znacznie mniejsza jest natomiast propagacja błędów - błąd w pozycji i -tej szyfrogramu powoduje błąd na tylko odpowiadającej sobie pozycji tekstu jawnego po procesie deszyfrowania, nie mając wpływu na pozostałe części tekstu jawnego. Wadą OFB jest niemożność jego samostnej resynchronizacji, w przypadku utraty pewnej ilości szyfrogramów, ponieważ utracony jest wówczas fragment strumienia klucza.

Tryb OFB pozwala na implementację synchronicznego szyfru strumieniowego na bazie szyfru blokowego. Z obu wersji OFB bezpieczniejsza jest wersja z pełnym sprzężeniem zwrotnym⁸.

2.2.5 Inne tryby pracy

Tryby wymienione w punktach 2.2.1 - 2.2.4 są podstawowymi i najczęściej używanymi trybami. Istnieją jednak również pewne modyfikacje tych trybów, na bazie których zdefiniowane nowe tryby - taką modyfikacją trybu z łańcuchowym wiązaniem - BC, jest tryb CBCC (*Cipher Block Chaining with Checksum*) - tryb w którym sprzężenie stanowi suma kontrolna szyfrogramu. Możliwe jest również wykorzystanie jako sprzężenia zwrotnego

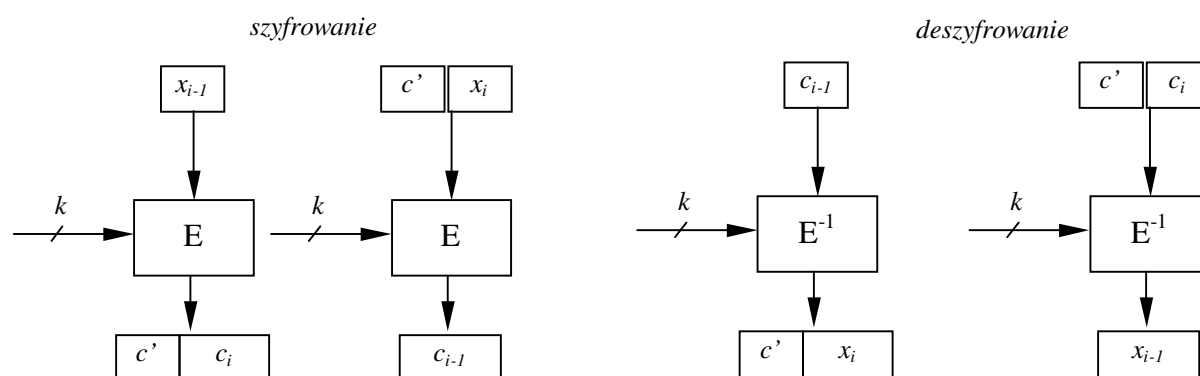
⁸ Dowód matematyczny znajduje się w A. Menezes, Handbook of Applied...

nie szyfrogramów, ale tekstu jawnego - tryb PFB (*Plaintext Feedback Block*). Te oraz inne rzadko używane tryby omówione zostały w literaturze przedmiotu⁹.

2.3 Dopełnianie oraz Ciphertext Stealing

Jeżeli tekst jawny zawiera liczbę bitów nie będącą krotnością wielkości bloku n , wówczas w celu jego poprawnego zaszyfrowania można zastosować procedurę dopełniania (*padding*), polegającą na wypełnieniu brakującej części bloku określoną sekwencją bitową. Rozwiązanie to, jest stosunkowo proste w implementacji, aczkolwiek charakteryzuje się pewną wadą, którą jest zwiększenie długości szyfrogramu w stosunku do tekstu jawnego, co dla większej liczby takich przypadków niekorzystnie wpływa na przepustowość.

Bardziej zaawansowaną techniką, umożliwiającą szyfrowanie danych krótszych niż długość bloku jest *ciphertext stealing*. Przy wykorzystaniu tej techniki, wiadomość jest szyfrowana klasycznym sposobem - blok po bloku, z wyjątkiem bloku ostatniego i przedostatniego. Zasadę *ciphertext stealingu* obrazuje poniższy rysunek:



Rys. 2.5 Zasada działania techniki *ciphertext stealing* w trybie ECB.

Z bloku przedostatniego x_{i-1} tworzony jest ostatni szyfrogram c_i oraz wartość pośrednia c' , która jest użyta w celu formowania przedostatniego szyfrogramu c_{i-1} . Wartość pośrednia c' nie stanowi części kryptogramu. Technika ta jest techniką w której nie zachodzi ekspansja danych. Może ona być stosowana w trybie ECB oraz CBC. W CFB oraz OFB implementacja powyższych technik jest niepotrzebna, ponieważ nie mają one wymogu, aby dane wejściowe stanowiły krotność długości bloku. Omawiając *ciphertext stealing*, i analizując mechanizm jego działania wraz z trybem CBC, warto zwrócić uwagę, że w trybie tym, do ostatniego

⁹ Zob. B. Schneier, *Applied Cryptography*, John Wiley&Sons, 1994, również dokumenty dostępne pod adresem <http://csrc.nist.gov/encryption/modes/>

Rozdział 3

Data Encryption Standard

3. DES - Data Encryption Standard

W 1972 NBS (National Bureau of Standards), współcześnie NIST (National Institute of Standards and Technology) zainicjował program, mający na celu zabezpieczenie danych komputerowych oraz telekomunikacyjnych, którego częścią było opracowanie algorytmu szyfrowania, który można by uznać za standard, oraz przede wszystkim algorytmu, który można by uznać za w pełni bezpieczny. Algorytm ten miał spełniać następujące kryteria:

- zapewniać wysoki poziom bezpieczeństwa,
- posiadać pełną specyfikację, łatwą do zrozumienia,
- być dostępnym dla wszystkich użytkowników oraz łatwym do zaadoptowania w różnego rodzaju aplikacjach,
- być łatwy oraz ekonomiczny w implementacji sprzętowej,
- zapewniać wysoką efektywność

Kandydatem który spełniał powyższe wymogi był algorytm szyfru, bazujący na opracowanym we wczesnych latach 70 przez IBM algorytmie Lucifer. W 1976 roku NBS przeprowadził dwa zebrania dotyczące algorytmu. Tematem pierwszego z nich był opis matematyczny algorytmu oraz możliwości wystąpienia ewentualnych funkcji zapadkowych, tematem drugiego możliwość zwiększenia długości klucza. 23 listopada 1976r. DES został uznany jako oficjalny standard szyfrowania danych, zaś jego pełna specyfikacja została opublikowana jako dokument FIPS PUB46, dnia 15 stycznia 1977 r. W 1980 r. opublikowany został kolejny dokument dotyczący trybów pracy DES - FIPS PUB81, zaś w roku 1982 dokument FIPS PUB74, zawierający wytyczne dotyczące implementacji DES. Od roku 1981 DES używany był do szyfrowania danych przez instytucje finansowe, administracyjne, telekomunikacyjne oraz odbiorców prywatnych. W ciągu lat DES był wielokrotnie recertyfikowany. W 1987 dla NBS stało się jasne, że DES zostanie w niedługim czasie złamany, jednakże w 1993 roku wciąż nie było algorytmu który mógłby zastąpić DES, w związku z czym DES uzyskał kolejną recertyfikację.

3.1 Opis algorytmu

Projekt DES oparty został na dwóch ogólnych koncepcjach budowy szyfrów blokowych: szyfrze kaskadowym oraz szyfrze Feistela¹⁰. DES jest również szyfrem iteracyjnym (*iterated cipher*).

¹⁰ Zob. H. Feistel, Cryptography and Computer Privacy, Scientific American, May 1973, vol.228, nr.5. Jest to artykuł opisujący koncepcję Feistela, koncepcję sieci podstawieniowo-permutacyjnej

DES operuje na 64-bitowej wielkości bloku oraz 64-bitowej długości klucza. Na wyjściu dane mają długość również 64 bitów. Efektywna długość generowanego klucza wynosi jednak 56 bitów. Pozostałe 8 bitów klucza może być używane do detekcji błędów, jako bity parzystości. Są to bity o numerach 8; 16; 24; 32; 40; 48; 56 i 64. Każdy z tych bitów sprawdza parzystość siedmiu bitów go poprzedzających. Efektywna długość klucza wynosząca 56 bitów pozwala uzyskać 2^{56} możliwych kluczy. Ponieważ DES należy do grupy szyfrów symetrycznych wartość klucza używanego do szyfrowania i deszyfrowania musi być taka sama.

Proces szyfrowania 64-bitowego bloku przebiega następująco:

- Przed rozpoczęciem właściwego szyfrowania 64 bity, formułujące blok danych wejściowych, poddawane są procesowi zamiany miejsc według permutacji inicjującej, określonej jako IP (w miejsce m_1 wstawiana jest wartość odpowiadająca m_{58} , w miejsce drugiego bitu - bit m_{50} ... w miejsce ostatniego bitu m_7). Permutacja według której przebiega proces zamiany przedstawiona została poniżej

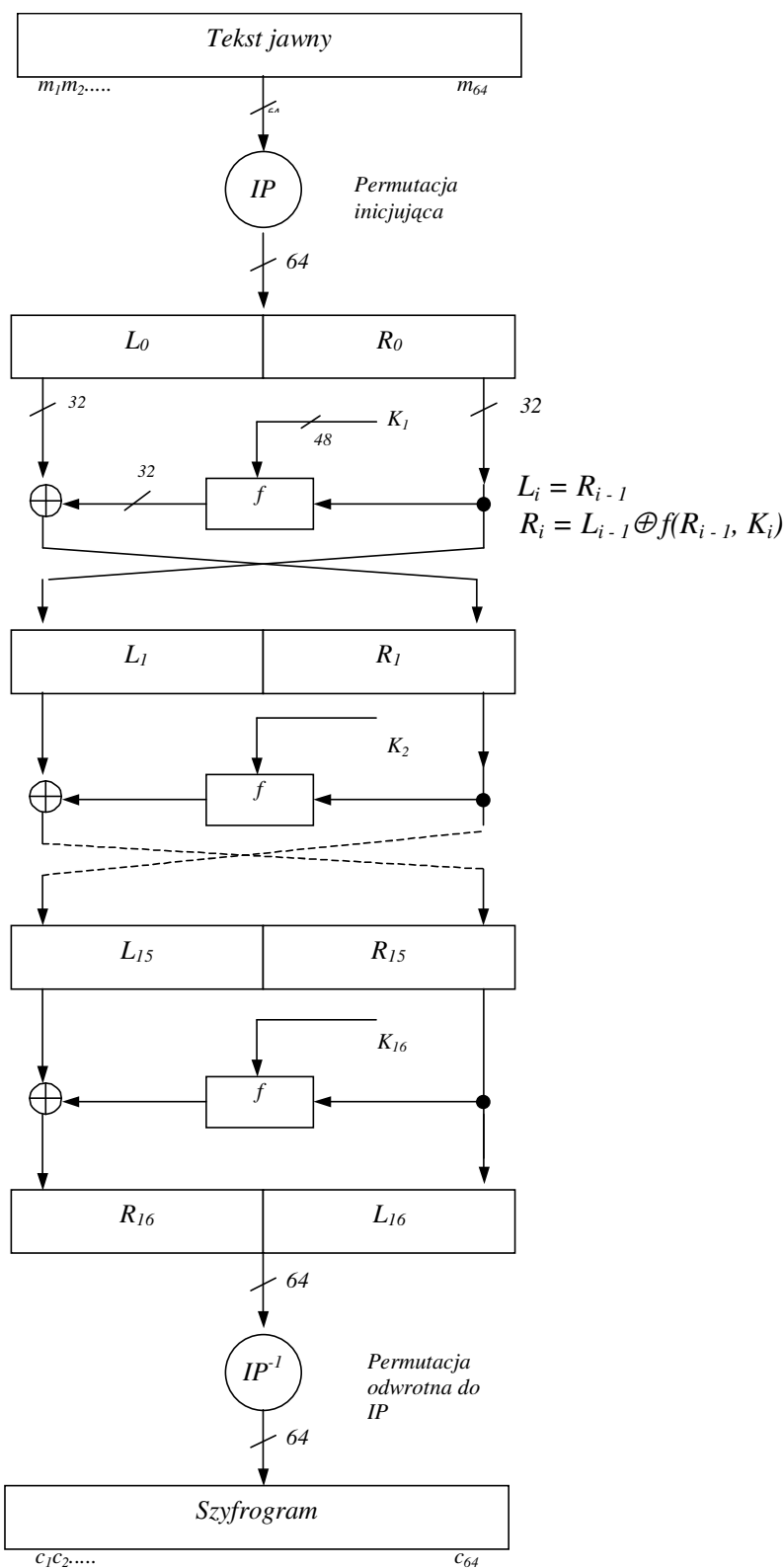
IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Rys. 3.1 Permutacja inicjująca IP

Permutacja początkowa IP oraz końcowa IP^{-1} , są niezależne od klucza, nie wpływają na bezpieczeństwo algorytmu i w wielu implementacjach zarówno softwarowych jak i hardwarowych są pomijane. Nie jest to jednak zgodne ze specyfikacją DES. Dane wychodzące z permutacji mają długość równą 64 bity i są rozdzielane na dwa bloki, oznaczane jako L oraz R , po 32 bity każdy.

- Zawartość bloku R_i (gdzie i oznacza numer cyklu) kopiowana jest do bloku L_{i+1} , oraz poddawana jest transformacji F , parametryzowanej 48 bitowym kluczem cyklu K_i . Zawartość 32 bitowego bloku L_i , wraz z 32 bitami wyjściowymi funkcji F ,

poddawane są operacji sumy XOR. Wynik sumowania kopiowany jest do bloku R_{i+1} . Operacje te są powtarzane odpowiednio w cyklach 1...15



Rys. 3.2 Schemat blokowy algorytmu DES

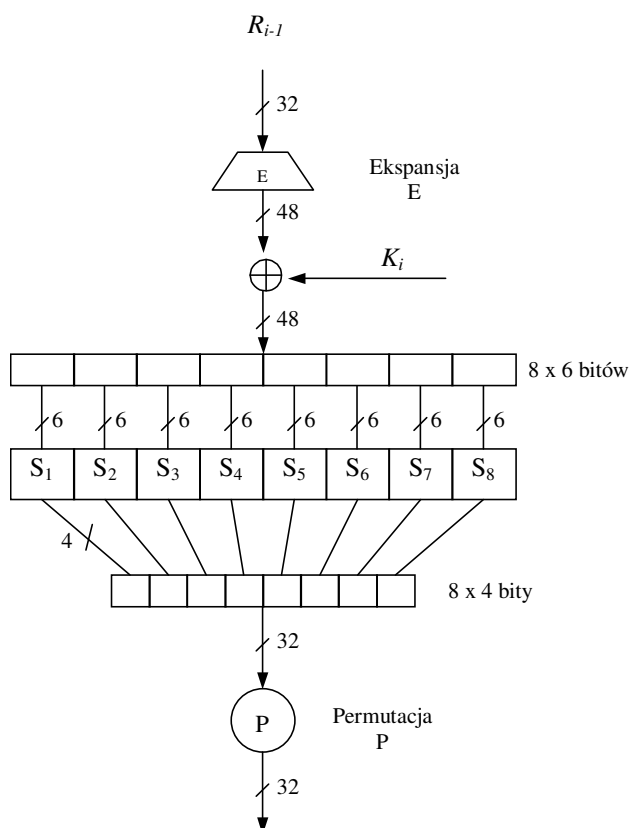
- Cykl 16 przebiega identycznie jak poprzednie, jednakże występuje w nim wymiana zawartości bloków R oraz L .
- Po zakończeniu cyklu szesnastego, 64-bitowy blok poddawane są permutacji określonej jako IP^{-1} , stanowiącej odwrotność transformacji inicjującej IP .

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Rys. 3.3 Permutacja końcowa IP^{-1}

3.1.1 Opis transformacji F

Transformacja F składa się z czterech kroków:



Rys. 3.4 Schemat blokowy funkcji F

1. Na wstępie 32 bity pochodzące z bloku R, poddawane są ekspansji do 48 bitów, zgodnie z poniższą tabelą:

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

Rys. 3.5 Tablica ekspansji bitowej E algorytmu DES

Celem ekspansji jest dostosowanie wielkości danych do wielkości klucza, aby możliwe było sumowanie XOR. Z punktu widzenia kryptograficznego, ekspansja poprawia własności dyfuzyjne cyklu gdyż zwiększa wpływ zmian pojedynczego bitu na wartość słowa wyjściowego, co uwidocznione zostało przy operacjach podstawienia - S-box..

2. Po ekspansji 48 bitów, wraz z 48-bitowym kluczem cyklu poddawane jest operacji XOR.
3. Suma ta, traktowana jako 8 słów 6-bitowych ($B_1B_2...B_8$) poddawana jest operacji podstawienia, realizowanej przez 8 różnych S-boxów, zgodnie z następującą zasadą: Pierwszy i ostatni bit słowa B (dla B_1 są to b_1 i b_6) reprezentują binarną wartość - numer wiersza S-boxa r , z zakresu 0 - 3. Bity pomiędzy nimi ($b_2b_3b_4b_5$) reprezentują binarnie wartości z zakresu 0-15. Niech wartości te będą numerami kolumn c S-boxa. Wartość słowa wyjściowego S-boxa otrzymujemy odczytując odpowiednią kombinację wiersza i kolumny z S-boxa. Ponieważ zastosowano S-boxy nieliniowe, wartości w S-box są z zakresu 0-15, co pozwala skrócić ich zapis z 6 do 4 bitów. Słowu B_i podlega i-ty S-box . Przykładowo dla słowa $B_1=011011$ otrzymujemy: $r=01_2=1$ oraz $c=1101_2=13$. Korzystamy z S_1 , otrzymujemy więc $S(B_1)=5=0101_2$. Na wyjściu S-boxów otrzymujemy więc 32 bity w postaci ośmiu 4-bitowych słów. Operacja podstawienia realizowana przy użyciu S-boxów jest jedyną nieliniową operacją w DES. Jest ona krytyczną operacją DES, najbardziej wpływającą na bezpieczeństwo algorytmu.

numer wiersza	Numer kolumny															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1																
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

numer wiersza	Numer kolumny															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_2																
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

numer wiersza	Numer kolumny															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_3																
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

numer wiersza	Numer kolumny															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_4																
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

numer wiersza	Numer kolumny															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_5																
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

numer wiersza	Numer kolumny															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_6																
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

numer wiersza	Numer kolumny															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_7																
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

numer wiersza	Numer kolumny															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_8																
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Rys 3.6 Tablica funkcji S-box dla $S_1 \dots S_8$ DES

4. Ostatnim krokiem jest permutacja 32 bitów wyjściowych z S-boxów. Przebieg tej permutacji określa poniższa tabela.

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

Rys. 3.7 Permutacja P funkcji F DES

3.1.2 Rozszerzenie klucza głównego

Efektywna długość klucza wynosi w DES 56 bitów. Każdy cykl posiada jednak własny klucz K_i identyfikowany jako klucz cyklu o długości 48 bitów. Wygenerowanie zbioru kluczy cyklu powstaje w wyniku rozszerzenia klucza głównego (*key schedule*). W wyniku działania tej procedury, z 56 bitów klucza otrzymujemy szesnaście 48-bitowych kluczy cyklu K_i ($1 \leq i \leq 16$). Przebieg tej procedury opisany został poniżej.

56 bitów klucza dzielonych jest na dwa bloki C oraz D , po 28 bitów każdy. Następnie z bloków C oraz D dokonywana jest permutacja $PC1$, zgodnie z poniższą:

PC1							
dla C_i	57	49	41	33	25	17	9
	1	58	50	42	34	26	18
	10	2	59	51	43	35	27
	19	11	3	60	52	44	36
dla D_i	63	55	47	39	31	23	15
	7	62	54	46	38	30	22
	14	6	61	53	45	37	29
	21	13	5	28	20	12	4

Rys. 3.8 Permutacja PC1 algorytmu rozszerzenia klucza

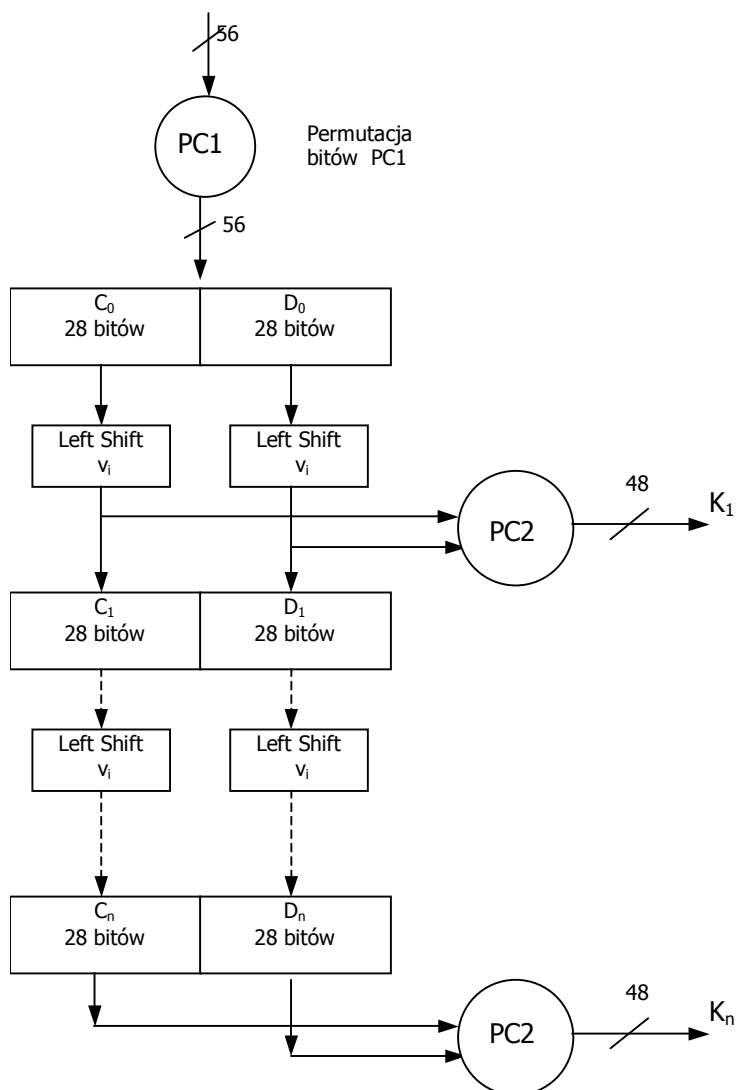
Następnie w zależności od numeru cyklu i , dokonywane jest przesunięcie bitów w lewo o określoną liczbę pozycji - v_i . . Dla $i=1;2;9;16$ dokonywane jest przesunięcie o jedna pozycje - $v_i=1$, dla pozostałych i - o dwie pozycje - $v_i=2$. Klucz cyklu i -tego K_i stanowi przesunięcie w lewo klucza cyklu poprzedniego K_{i-1} , o liczbę v_i pozycji i permutację bitów $PC2$, przedstawioną na poniższym rysunku



PC2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Rys. 3.9 Permutacja PC2 algorytmu rozszerzenia klucza

Schemat blokowy algorytmu rozszerzenia klucza można przedstawić następująco:



Rys 3.10 Schemat blokowy algorytmu rozszerzenia klucza

3.1.3 Deszyfrowanie

Algorytm deszyfrowania za pomocą DES jest identyczny jak algorytm szyfrowania. W procesie deszyfrowania, używamy jednak w odwrotnej kolejności kluczy cyklu K_i , a mianowicie klucz K_{16} używany podczas szyfrowania, podczas deszyfrowania jest kluczem K_1 . Permutacja IP odwraca działanie IP^{-1} . Operacje wykonywane w pierwszym cyklu, odwracają operacje szyfrowania cyklu 16, operacje wykonywane w cyklu 2, odwracają operacje szyfrowania cyklu 15, itd... Stąd też istnieje konieczność użycia odwrotnej kolejności kluczy. W celu uzyskania takiej kolejności kluczy, można użyć algorytmu rozszerzenia klucza (*key schedule*) w odwrotnej kolejności, bądź wygenerować klucze w odwróconym porządku wykorzystując następującą własność, że kiedy zostanie wygenerowany K_{16} , zostaje przywrócona wartość początkowa rejestrów C oraz D (ponieważ zawartość każdego z nich została przesunięta o 28 bitów). Zatem aby w pierw uzyskać wartość odpowiadającą K_{16} , zamiast przesunąć w lewo należy stosować przesunięcia bitów w prawo oraz nadać przesunięciu v_1 wartość 0.

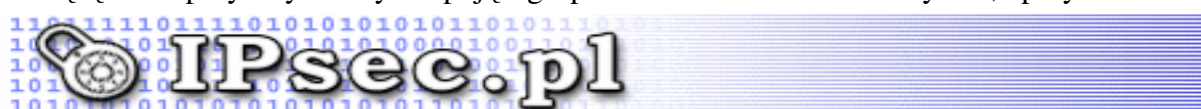
3.2 Tryby pracy DES

Zgodnie ze specyfikacją FIPS81, DES może pracować w trybie ECB, CBC, CFB oraz OFB. W trybie CFB oraz OFB DES może dokonywać szyfrowania danych będącymi blokami o różnych wielkościach, co wynika z własności tych trybów. Wszystkie tryby, z wyjątkiem ECB, mają za zadanie zapobiec ataku na DES z wybranym tekstem jawnym.

3.3 3-DES

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 Frontier Foundation 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 minut¹¹.

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} .

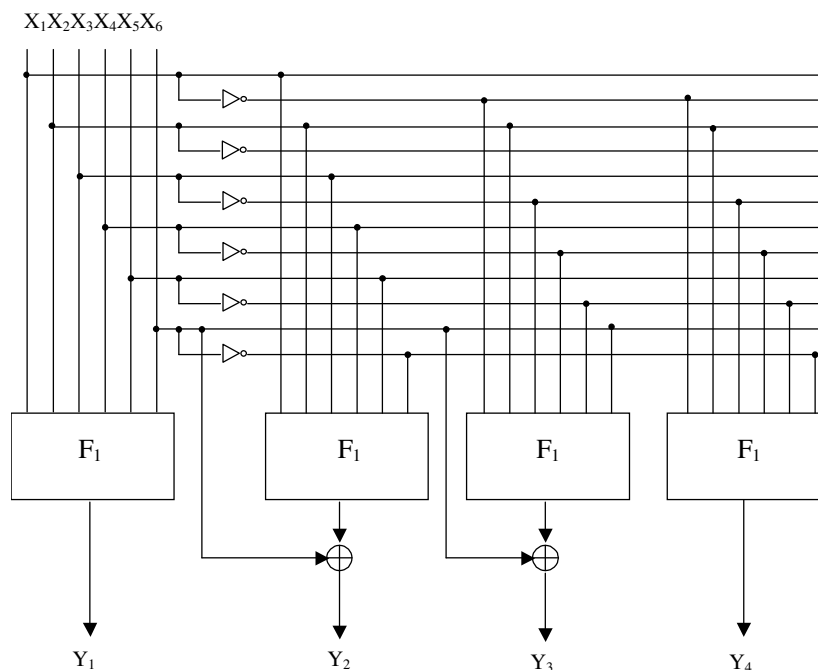
Dokładna analiza bezpieczeństwa poszczególnych trybów przeprowadzona przez E.Bihama¹², wykazała, że niektóre warianty 3DES są bezpieczniejsze od innych.

3.4 Komentarze i obserwacje dotyczące DES

1. Permutacja Inicjująca IP - Załóżmy, że 64-bitowy blok danych jest reprezentowany przez 8 znaków ASCII ($b_0b_1b_2b_3\dots b_{63}$). Bity parzystości zajmują więc pozycje o numerach 0,8,16,24,32,40,48,56 lub 7,15,23,31,39,47,55. Zatem po permutacji, będą one zajmowały pozycje odpowiednio 39,38,37,36,35,34,33,32 lub 31,30,29,28,27,26,25,24. Można więc zauważyć, że bity parzystości łączone są w jeden bajt, oraz to, że tworzą one pierwszy bajt bloku R lub ostatni bajt bloku L.
2. S-boxy. Dokonując dekompozycji S-boxa S_4 widać, że tylko pierwsza z jego funkcji jest nieliniowa, pozostałe trzy są liniowe i można je uzyskać z pierwszej poprzez negację bitów wejściowych oraz negację 2 i 3 wyjścia sterowaną zmienną X_6 . Zmniejsza to liczbę nieliniowych funkcji w DES z 32 do 29.
Każdy z S-boxów charakteryzuje się również tym, że dla niektórych kombinacji wejściowych, negacja dwóch bitów, lub odpowiednia modyfikacja niektórych bitów wejściowych nie ma wpływu na wartość wyjściową S-boxa.
Funkcja F nie jest funkcją typu 1 na 1 jest zatem możliwe, że dla dwóch różnych wartości 32 bitów R, na wyjściu funkcji F otrzymamy jednakową wartość.

¹¹ Na podst. Cracking DES:Secrets of Encryption Research, Wiretap Politics and Chip Design, EFF, July 98

¹² Zob. E.Biham, Cryptanalysis of Triple-Modes of Operation, Technion -Computer Science Department - Technical Report CS0885-1996



Rys. 3.11. Dekompozycja S-boxa \$S_4\$

3. Własność komplementarności. Funkcja szyfrująca powinna być losową funkcją zarówno klucza jak i tekstu jawnego. Nie jest tak w przypadku DES. Jeśli \$X_j'\$ stanowi dopełnienie bitowe bloku \$X_j\$, zaś \$K'\$ - dopełnienie bitowe klucza wówczas:

$$C_j = \text{DES}(X_j; K)$$

$$C_j' = \text{DES}(X_j'; K')$$

Jeśli zatem kryptoanalityk ma do dyspozycji pary \$(X_1; C_1)\$ oraz \$(X_1'; C_2)\$, wówczas dla jakiegokolwiek nieznanego klucza \$K\$ można stwierdzić że: \$C_2 = \text{DES}(K; X_1')\$ oraz \$C_2' = \text{DES}(K'; X_1)\$. Wniosek ten pozwala w przypadku poszukiwania klucza za pomocą ataku wybranym tekstem jawnym zredukować liczbę obliczeń o połowę - z \$2^{55}\$ do \$2^{54}\$.

4. Słabe klucze.

Jeśli obydwa bloki \$C\$ i \$D\$ w algorytmie rozszerzenia klucza będą zawierały same jedynki lub same zera, wówczas, generowany klucz cyklu będzie stały i jednakowy dla każdego cyklu. Zatem zamiana kolejności kluczy nie ma znaczenia, co pozwala stwierdzić, że proces szyfrowania i deszyfrowania stanowi jednakową iterację. Klucze takie nazywane są słabymi kluczami. Dla DES istnieją 4 słabe klucze których wartości przedstawione zostały poniżej.

Słabe klucze (<i>weak keys</i>) - HEX				C_0	D_0
0101	0101	0101	0101	$\{0\}^{28}$	$\{0\}^{28}$
FEFE	FEFE	FEFE	FEFE	$\{1\}^{28}$	$\{1\}^{28}$
1F1F	1F1F	1F1F	1F1F	$\{0\}^{28}$	$\{1\}^{28}$
E0E0	E0E0	E0E0	E0E0	$\{1\}^{28}$	$\{0\}^{28}$

W praktyce oznacza to iż dla takiego klucza $c = DES(k;x)$ oraz że $x=DES(k;DES(k;x))$

5. Pół-słabe klucze (*semi-weak keys*)

Wektor przesunięć rejestru $-v$ w algorytmie rozszerzenia klucza jest następującej postaci:

dla szyfrowania: 1 1 222222 1 222222 1 (w lewo)

dla deszyfrowania: 0 1 222222 1 222222 1 (w prawo)

Jeśli zatem w jednym z rejestrów podczas szyfrowania pojawi się sekwencja bitów 01010101010101010101010101010101 odpowiednio w blokach C oraz D, to analizując procedurę rozszerzenia klucza widać, że klucze cyklu K_1 oraz K_9-K_{15} są identyczne; identyczne są również klucze K_2-K_8 oraz K_{16} . W praktyce oznacza to, że dla pary kluczy ($K_1 ;K_2$) istnieje następująca zależność: $x=DES(K_1; DES(x ; K_2))$, co oznacza anulowanie procesu szyfrowania.

Dla DES istnieje 6 par pół-słabych kluczy.

C_0	D_0	semi-weak key pairs (hex)			
$\{01\}^{14}$	$\{01\}^{14}$	01FE	01FE	01FE	01FE
$\{10\}^{14}$	$\{10\}^{14}$	FE01	FE01	FE01	FE01
$\{01\}^{14}$	$\{10\}^{14}$	1FE0	1FE0	0EF1	0EF1
$\{10\}^{14}$	$\{01\}^{14}$	E01F	E01F	F10E	F10E
$\{01\}^{14}$	$\{0\}^{28}$	01E0	01E0	01F1	01F1
$\{10\}^{14}$	$\{0\}^{28}$	E001	E001	F101	F101
$\{01\}^{14}$	$\{1\}^{28}$	1FFE	1FFE	0EFE	0EFE
$\{01\}^{14}$	$\{1\}^{28}$	FE1F	FE1F	FE0E	FE0E
$\{0\}^{28}$	$\{01\}^{14}$	011F	011F	010E	010E
$\{0\}^{28}$	$\{10\}^{14}$	1F01	1F01	0E01	0E01
$\{1\}^{28}$	$\{01\}^{14}$	E0FE	E0FE	F1FE	F1FE
$\{1\}^{28}$	$\{10\}^{14}$	FEE0	FEE0	FEF1	FEF1

6. Częściowo słabe klucze (*quasi weak keys*).

Na Eurocrypt 94, L.R. Knudsen¹³ przedstawił kolejną grupę 256 kluczy zwaną częściowo słabymi kluczami. Klucze te dają cztery różne podklucze, z których każdy jest wykorzystywane w algorytmie czterokrotnie. L.R. Knudsen udowodnił, że istnieje

¹³ L.R. Knudsen, New Potentially Weak Keys for DES and LOKI, extended abstract, Eurocrypt 94, Springer Verlag 1998

grupa kluczy, różniących się tylko bitami odnoszącymi się do jednego S-boxa, przy użyciu których szyfrując dany tekst jawny uzyskamy identyczne szyfrogramy. Nie stanowią one zagrożenia w ataku DES, mogą jednak stanowić pewne zagrożenie gdy za pomocą DES realizowana jest funkcja haszująca.

7. Dla dwóch kluczy głównych K_1 oraz K_2 , nie istnieje klucz K_3 który byłby im równoważny. Oznacza to, że dla DES nie istnieje klucz K_3 taki że:

$$E_{K_3}(x) = E_{K_2}(E_{K_1}(x))$$

Własność ta oznacza, że DES nie tworzy grupy.

3.5 Szybkość DES

DES jest szyfrem przeznaczonym głównie do implementacji sprzętowych. Na rynku pojawiło się więc wiele układów szyfrujących za pomocą DES. Obecne sprzętowe implementacje DES pozwalają na uzyskanie prędkości rzędu kilkunastu GBit/sec. Poszczególne przypadki zostały omówione w dalszej części pracy, stanowiąc swego rodzaju punkt odniesienia przy porównywaniu prędkości innych szyfrów.

3.6 Siła kryptograficzna i bezpieczeństwo DES

Atak na DES polegający na wyczerpującym poszukiwaniu klucza ma złożoność 2^{56} . Jest to jednak najgorszy z możliwych ataków na DES. Biorąc pod uwagę możliwości obliczeniowe dzisiejszych komputerów, złożoność ta nie stanowi większej przeszkody. Maszyna łamiąca DES, swoje działanie opiera właśnie na tego rodzaju ataku i potrafi złamać DES w ciągu 3,5 godziny, przy rozsądnym nakładzie finansowym.

DES jako standard szyfrowania danych, stanowił przez długie lata cel rozmaitego rodzaju ataków. DES był również czynnikiem sprawczym i inspiracją poszukiwania nowych metod kryptoanalizy. Kryptoanaliza różnicowa DES została przedstawiona w rozdziale 7. Można jednak zauważyć, że DES wykazuje bardzo dużą odporność na tego rodzaju atak (złożoność wynosi 2^{47} lub 2^{55}). Przyczyną tego jest fakt, że kryptoanaliza różnicowa była znana projektantom DES¹⁴, choć do czasu jej „odkrycia i ujawnienia” przez E.Bihama oraz A.Shamira była tajna. W dalszej części pracy została również omówiona kryptoanaliza liniowa DES, będąca obecnie atakiem na DES charakteryzującym się najmniejszym stopniem złożoności. 2^{43} .

¹⁴ Kryptoanaliza różnicowa w latach 70. była tajna. Zob. B. Schneier, Applied Cryptography, John Wiley&sons, 1994, Jako lekturę uzupełniającą można również polecić D. Coppersmith, "The Data Encryption Standard (DES) and its Strength Against Attacks," IBM Journal of Research and Development, v. 38, n. 3, May 1994, pp. 243-- 250

Rozdział 4

Inne szyfry blokowe

4.1 IDEA

W 1990 r. na konferencji Eurocrypt 90, X.Lai oraz J.L.Masses przedstawili projekt nowego szyfru blokowego¹⁵, 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 modyfikacji¹⁶ (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 (rys. 4.1), 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*). W procesie szyfrowania stosowane są trzy grupy operacji algebraicznych zdefiniowane odpowiednio:

⊕ - suma modulo 2 - xor

⊞ - dodawanie liczb całkowitych w ciele skończonym $GF(2^{16})$

⊙ - mnożenie liczb całkowitych w ciele skończonym $GF(2^{16}+1)$, przy czym element zerowy - 0, traktowane jest jako 2^{16} , tj.

$$0000_B \odot 1000_B = 2^{16} 2^3 \bmod 2^{16}+1 = 2^{13}+1$$

¹⁵ Zob. X.Lai, J.L.Masses, A Proposal for a New Block Encryption Standard, Eurocrypt90, Springer Verlag 1998,

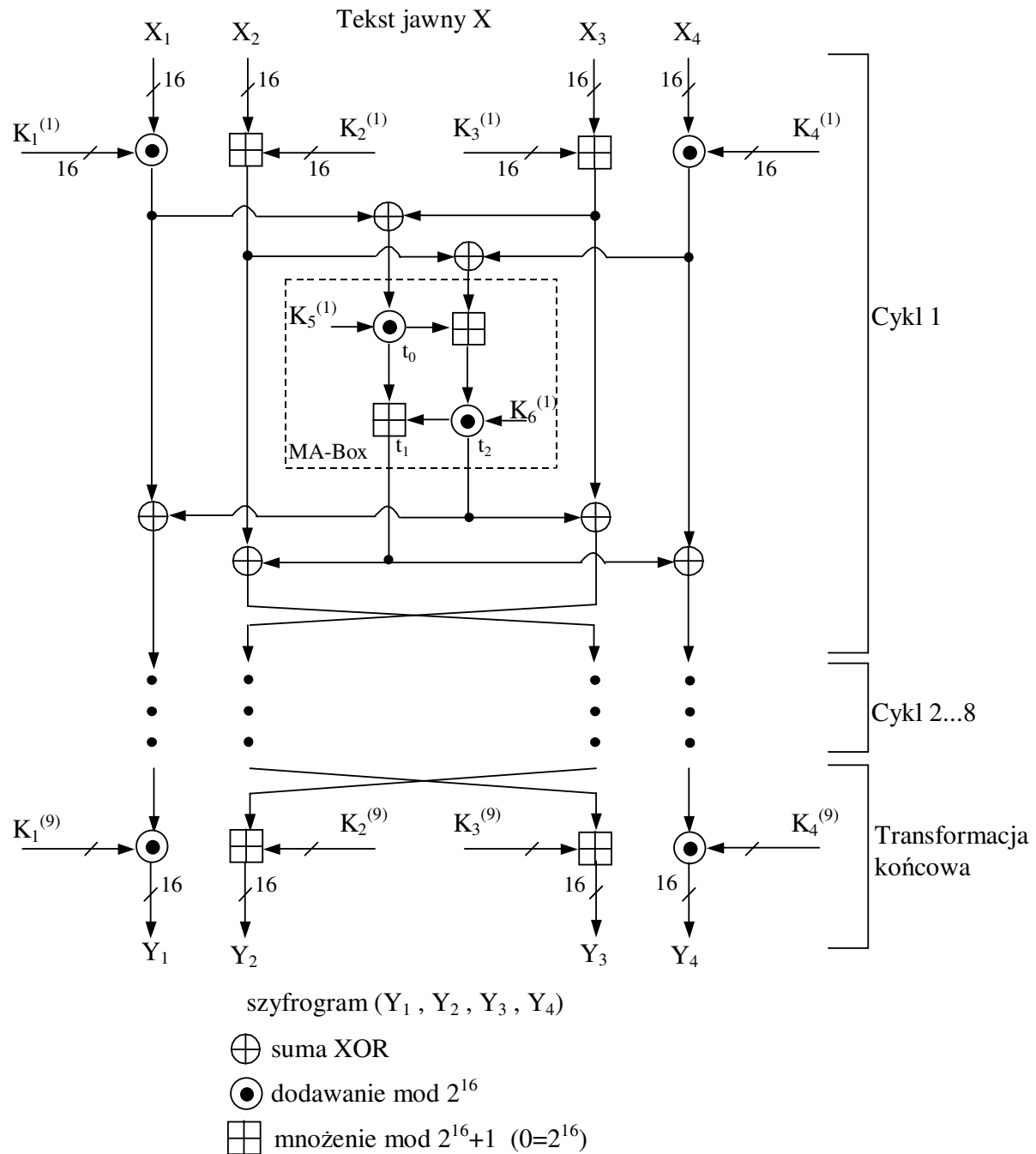
¹⁶ X.Lai, J.L.Masses, Markov Ciphers and Differential Cryptanalysis, Eurocrypt 91, Springer Verlag 1998,

4.1.1 Przebieg procesu szyfrowania

Schemat pojedynczego cyklu pracy przedstawia rys. 4.1

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} ...



Rys. 4.1 Cykl pracy IDEA

3. Następnie przez kolejnych 8 cykli wykonywane są następujące operacje:

$$X_1 = X_1 * K_1^{(r)} \text{ mod } 2^{16} + 1$$

$$X_4 = X_4 * K_4^{(r)} \text{ mod } 2^{16} + 1$$

$$X_2 = X_2 + K_2^{(r)} \text{ mod } 2^{16}$$

$$X_3 = X_3 + K_3^{(r)} \text{ mod } 2^{16}$$

$$t_0 = K_5^{(r)} * (X_1 \text{ xor } X_3) \text{ mod } 2^{16} + 1$$

$$t_1 = K_6^{(r)} * (t_0 + (X_2 \text{ xor } X_4) \text{ mod } 2^{16}) \text{ mod } 2^{16} + 1$$

$$t_2 = t_0 + t_1 \text{ mod } 2^{16}$$

$$X_1 = X_1 \text{ xor } t_1$$

$$X_4 = X_4 \text{ xor } t_1$$

$$a = X_2 \text{ xor } t_2$$

$$X_2 = X_3 \text{ xor } t_1$$

$$X_3 = a$$

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

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

$$Y_1 = X_1 * K_1^{(9)} \text{ mod } 2^{16} + 1$$

$$Y_4 = X_4 * K_4^{(9)} \text{ mod } 2^{16} + 1$$

$$Y_2 = X_3 + K_2^{(9)} \text{ mod } 2^{16}$$

$$Y_3 = X_2 + K_3^{(9)} \text{ mod } 2^{16}$$

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

4.1.2 Rozszerzenie klucza

Rozszerzenie klucza w IDEA odbywa się w następujący sposób:

128 bitowy klucz główny $K = [k_1 k_2 \dots k_{128}]$, 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 $K_7^{(1)}$, z ósmego bloku - klucz $K_8^{(1)}$.

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.

4.1.3 Deszyfrowanie

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	$(K_1^{(10-r)})^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$(K_4^{(10-r)})^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
$2 \leq r \leq 8$	$(K_1^{(10-r)})^{-1}$	$-K_3^{(10-r)}$	$-K_2^{(10-r)}$	$(K_4^{(10-r)})^{-1}$	$K_5^{(9-r)}$	$K_6^{(9-r)}$
r=9	$(K_1^{(10-r)})^{-1}$	$-K_2^{(10-r)}$	$-K_3^{(10-r)}$	$(K_4^{(10-r)})^{-1}$

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

4.1.4 Przesłanki projektowe IDEA

Jednym z podstawowych założeń podczas projektowania szyfru, jest bardzo silna zależność wartości szyfrogramu od wartości tekstu jawnego oraz klucza, przy czym zależność ta nie może być łatwa do dedukcji (*confusion layer*). Projektanci IDEA osiągnęli ten cel poprzez połączenie trzech różnych grup operacji algebraicznych, w ten sposób, że wartość wyjściowa operacji jednego typu, nigdy nie stanowi wartości wejściowej następnej operacji tego samego typu. Dodatkowo żadna para z tych trzech operacji nie spełnia prawa rozłączności względem mnożenia oraz przemienności względem dodawania.

Kolejnym założeniem jest silna propagacja (*diffusion layer*) - każdy bit tekstu jawnego powinien mieć jak największy wpływ na wartości wszystkich bitów szyfrogramu. Ta sama zasada odnosi się do bitów klucza. Analiza komputerowa dokonana przez twórców szyfru pokazuje, że warunki te są spełnione już po pierwszym cyklu. Propagację tą realizuje transformacja oznaczona na schemacie blokowym jako MA-Box. - będąca transformacją sumowo-iloczynową (*multiplication-addition box*), przekształcającą dwa 16-bitowe bloki pod wpływem dwóch kluczy cyklu. MA-Box charakteryzuje się następującymi własnościami:

- dla każdego z kluczy - K_5 oraz K_6 transformacja $MA(; ; K_5; K_6)$ jest transformacją odwracalną; transformacja ta jest odwracalna również dla każdego z bloków tekstu jawnego.
- transformacją zapewnia pełną propagację, w tym sensie, każdy blok wyjściowy zależy od każdego z bloków wejściowych; przy użyciu jednego zestawu kluczy K_5 oraz K_6 , nie ma możliwości aby dla dwóch różnych wartości bloku wejściowego na wyjściu pojawiła się jednakowa wartość.

Podobieństwo procesu szyfrowania i deszyfrowania osiągnięte zostało dzięki odwracalności poszczególnych operacji algebraicznych bądź transformacji.

4.1.5 Siła i bezpieczeństwo oraz obserwacje IDEA

1. Pierwsza wersja szyfru, wprowadzona w roku 90 jako PES była nieco mniej bezpieczna niż IDEA, zdefiniowana w roku 91. Mimo, iż w momencie projektowania PES, kryptoanaliza różnicowa była jeszcze nieznaną, PES, a tym samym IDEA, są szyframi odpornymi na tego rodzaju atak. Dowiedli tego X.Lai i J.Masses na Eurocrypt 91, wprowadzając koncepcję szyfrów Markova¹⁷, wśród których niektóre posiadające pewne własności odporne są na kryptoanalizę różnicową.
2. Konstruując szyfr, Lai i Masses użyli trzech różnych grup algebraicznych, takich, aby pary tych grup nie spełniały praw przemienności względem dodawania i rozłączności względem mnożenia. Po wykonaniu dokładniejszej analizy matematycznej przez Willi Meiera¹⁸ okazało się że para operacji dodawania w $GF(2^{16})$ oraz mnożenia w $GF(2^{16}+1)$ jest częściowo rozłączna względem mnożenia i przemienna względem dodawania. Oznacza to że spełnione jest równanie

$$a \odot (b \boxplus \text{delta}) = a \odot b \boxplus a \odot \text{delta}$$

¹⁷ X.Lai, J.L.Masses, Markov Ciphers and Differential Cryptanalysis, Eurocrypt 91, Springer Verlag 1998

¹⁸ W.Meier, On Security of The Idea Block Cipher, Eurocrypt 93, Springer Verlag 1998

w następujących przypadkach:

- jeśli $a=1$ to r -nie to jest spełnione dla każdych wartości b oraz δ ,
- jeśli $a \neq \{0,1\}$ oraz $b \neq 0$ i $\delta \neq 0$ to równanie to jest spełnione gdy:

$$b + \delta \leq 2^n \text{ oraz } a \odot b + a \odot \delta \leq 2^n$$

Można zauważyć że równania te będą spełniane z określonymi prawdopodobieństwami. Spostrzeżenie to pozwoliło na kryptoanalizę liniową i różnicową jednego lub dwóch cykli IDEA o stopniu złożoności mniejszym niż wyczerpujące szukanie klucza.

3. J. Daemen analizując algorytm rozszerzenia klucza¹⁹ zdefiniował kilka klas kluczy zwanych słabymi kluczami w IDEA. Definicja słabego klucza w IDEA ma nieco inną postać niż w przypadku DES. J. Daemen wykazał, że 2^{23} kluczy IDEA pozwala uzyskać liniowe równania opisujące jeden cykl szyfrowania, co ma znaczenie w przypadku ataku za pomocą liniowej kryptoanalizy. Co więcej, 2^{35} kluczy pozwala uzyskać w kryptoanalizie różnicowej charakterystyki z prawdopodobieństwem 1.

J. Daemen przedstawił również klasę 2^{51} kluczy, dla których 2 procesy szyfrowania oraz rozwiązanie układu 16 nieliniowych równań logicznych z 12 niewiadomymi pozwala zidentyfikować przynależność klucza do tej klasy. Jeśli klucz taki znajduje się w niej, wówczas część jego bitów może zostać w sposób efektywny znaleziona. Użycie tych kluczy nie pogarsza znacząco bezpieczeństwa algorytmu. Ich słabość polega na tym, że można stosunkowo łatwo wydedukować, czy zostały one użyte, czy też nie, a jeśli tak, oznacza to zmniejszenie złożoności w ataku z wyczerpującym poszukiwaniem klucza. Aby temu zapobiec Daemen zaproponował modyfikacje algorytmu rozszerzenia klucza pozbawioną powyższej wady.

4. Ataki na zredukowaną liczbę cykli w IDEA, bazujące na kryptoanalizie różnicowej, oraz połączeniu kryptoanalizy liniowo-różnicowej zostały przeprowadzone przez L.R.Knudsen, V.Rijmena oraz J.Borsta²⁰. Pierwszy z nich po wykonaniu 3,5 cykli potrafił znaleźć klucz mając do dyspozycji 2^{56} wybranych tekstów jawnych, oraz dokonując 2^{67} szyfrowań. Drugi z ataków potrafił odnaleźć klucz po trzech cyklach, mając do dyspozycji 2^{29} wybranych par tekstów jawnych oraz dokonując 2^{44} szyfrowań. Ataki te okazywały się być efektywniejsze w stosunku do ataku z wyczerpującym sprawdzaniem kluczy, tylko dla trzech cykli. Przy przeprowadzaniu

¹⁹ J. Daemen, Weak Keys For Idea, Crypto 93, Springer Verlag 1998, także J.Daemen, Cipher and hash function design strategies based on linear and differential cryptanalysis, Doctorial Dissertation, March 1995, K.U.Leuven

²⁰ L.R. Knudsen, V.Rijmen, J.Borst, Two Attacks on reduced Idea - extended abstracts, Eurocrypt 97, Springer Verlag 1998

pełnej kryptoanalizy, charakteryzowały się one zbyt dużym stopniem skomplikowania oraz zbyt wielką złożonością. IDEA wykazuje odporność na kryptoanalizę różnicową już po 4 z 8 cykli.

Na podstawie powyższych danych, można wywnioskować, że najlepszym rodzajem ataku jest atak polegający na sprawdzeniu słabych kluczy. Atak polegający na wyczerpującym poszukiwaniu klucza ma złożoność 2^{128} .

4.1.6 Implementacje IDEA

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. Szyfr ten można bardzo łatwo i efektywnie zaimplementować na drodze programowej, dlatego też IDEA 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 multimedialnych²¹.

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.

²¹ H.Lipmaa, Idea: A Cipher For Multimedia Architectures ?, unpublished

4.2 RC5

RC5 został zaprojektowany w 1994 roku przez Ronalda Rivesta, jednego z twórców algorytmu RSA. Konstruując RC5 Rivest miał na uwadze następujące aspekty²²:

- RC5 ma być szyfrem symetrycznym,
- RC5 ma być przeznaczony do implementacji zarówno sprzętowej jak i programowej. Oznacza to, że RC5 jako swych prymitywów ma używać jedynie instrukcji dostępnych w standardowych mikroprocesorach,
- RC5 ma być szybki i możliwy do implementacji w procesorach pracujących z różnymi długościami słowa, dlatego też jednym z parametrów RC5 ma być ilość bitów słowa.
- struktura szyfru ma być strukturą iteracyjną z różną liczbą cykli, mającą stanowić zarówno parametr jak i kompromis pomiędzy szybkością a bezpieczeństwem,
- RC5 powinien pracować z różnej długości kluczami,
- RC5 powinien być prosty i łatwy w opisie. Prostota pozwala w łatwy sposób dokonać analizy bezpieczeństwa szyfru.

4.2.1 Opis algorytmu

RC5 jest szyfrem zorientowanym na długość słowa. Dla wszystkich podstawowych operacji zarówno długość słowa wejściowego jak i wyjściowego wynosi w bitów.

Wielkość bloku wejściowego, podobnie jak wielkość bloku wyjściowego wynosi $2w$ bitów. Zazwyczaj $w=32$, co oznacza że wielkości bloków tekstu jawnego i szyfrogramu wynoszą 64 bity. RC5 jest bardzo elastyczny i może operować na dowolnych długościach słowa w , jednakże w celach normalizacyjnych proponuje się długości słowa 16, 32 oraz 64 bit.

Kolejnym parametrem RC5 jest liczba cykli r , od której zależy również liczba wygenerowanych kluczy cyklu za pomocą procedury rozszerzenia klucza głównego. Zależność pomiędzy liczbą wygenerowanych kluczy cyklu t a liczbą cykli przedstawia się następująco:

$$t = 2(r + 1)$$

Większa liczba cykli r , zapewnia większy poziom bezpieczeństwa ale zmniejsza szybkość działania szyfru.

²² R.L.Rivest, The RC5 Encryption Algorithm, RSA Laboratories 1994

Oprócz różnych długości słów, różnej liczbie cykli, RC5 ma możliwość operowania na kluczach K o różnej długości. Wielkość klucza określona jest w bajtach za pomocą parametru b , zaś odpowiednie bajty klucza głównego poprzez K_0, K_1, \dots, K_{b-1} .

Różnorodność wielkości powyższych parametrów spowodowała, że wprowadzona została notacja dotycząca algorytmu, której postać przedstawiona została poniżej:

$$\text{RC-5 } w / r / b$$

gdzie:

w - wielkość słowa w bitach,

r - liczba cykli,

b - długość klucza głównego K w bajtach.

Przykładowo: RC5 - 32 / 16 / 10 jest szyfrem przetwarzającym 64 bitowe bloki, operującym na kluczu o długości 80 bitów, oraz wykonującym 16 cykli.

Należy zauważyć, że nie wszystkie wartości parametrów oznaczają bezpieczeństwo: dla liczby cykli wynoszącej 0, proces szyfrowania w zasadzie nie ma miejsca, dla $r = 1$, szyfr może zostać łatwo złamany. Wybranie długości klucza równej 0 bajtów nie daje żadnego bezpieczeństwa. Wybór wielkości parametrów RC5 powinien więc stanowić kompromis pomiędzy bezpieczeństwem a szybkością, oraz być optymalny dla określonej implementacji, lub też powinien być optymalizowany pod kątem zastosowań (dla niektórych aplikacji - np. zarządzających kluczami publicznymi, nie jest zbyt ważna szybkość szyfru, ale bezpieczeństwo, stąd też sugerowana liczba cykli wynosi w nich 32).

Operacjami algebraicznymi wykonywanymi w trakcie szyfrowania i deszyfrowania przez RC5 są

1. suma dwóch liczb w ciele skończonym ciele $GF(2^w)$, oznaczana znakiem dodawania '+' oraz odwrotność tej operacji - oznaczana '-'
2. xor - suma modulo 2 dwóch liczb całkowitych oznaczana \oplus
3. cykliczne przesunięcia w lewo; słowo x przesunięcie słowa x o y bitów oznaczane jest jako $x \lll y$. Wartość y interpretowana jest jako liczba modulo w , więc kiedy w stanowi potęgę liczby 2, wówczas tylko $\lg_2(w)$ mniej znaczących bitów używanych jest do wyznaczenia wartości przesunięcia. Operacja odwrotna - cykliczne przesunięcie w prawo oznaczona jest jako: $x \ggg y$.

4.2.2 Przebieg procesu szyfrowania

W każdym cyklu używane są dwa klucze cyklu S , wygenerowane za pomocą procedury rozszerzenia klucza głównego K , oznaczane jako S_{2i} oraz S_{2i+1} . Zakłada się, że blok wejściowy o wielkości $2w$ bitów rozdzielany jest na dwa bloki o długościach w bitów, do rejestrów A oraz B . Algorytm szyfrowania można przedstawić następująco:

```
A=A+S0 ;  
B=B+S1 ;  
for i=1 to r do  
    A= ( (A⊕B) <<<B) +S2i ;  
    B= ( (A⊕B) <<<A) +S2i+1 ;
```

Na początku działania wartości rejestrów A i B poddawane są dodawaniu w ciele $GF(2^W)$ z wartościami kluczy S_0 oraz S_1 . Następnie przez kolejnych r cykli wykonywane są: sumowania XOR zawartości rejestrów A i B , cykliczne przesunięcia, oraz sumowanie w ciele $GF(2^W)$ przesuniętych wartości z kluczami cyklu S .

Należy zwrócić uwagę na fakt, że operacja przesunięcia jest jedyną nieliniową operacją algorytmu. Wartość przesunięcia jest wyznaczana przez wartość znajdującą się w innym rejestrze. W jednym cyklu zmieniają się wartości obu rejestrów. Nie jest to zatem szyfr bazujący na strukturze Feistela, ale typowy szyfr iteracyjny.

4.2.3 Deszyfrowanie

Proces deszyfrowania za pomocą RC5 polega na przeprowadzeniu operacji odwrotnych do szyfrowania, i w odwrotnej kolejności. Algorytm deszyfrowania w postaci pseudokodu, można zatem przedstawić następująco:

```
for i=r downto 1 do  
    B= ( (B-S2i+1) >>>A) ⊕A ;  
    A= ( (A-S2i) >>>B) ⊕B ;  
B = B - S1 ;  
A = A - S0 ;
```

4.2.4 Rozszerzenie klucza

Podobnie jak większość szyfrów, również RC5 w celu wygenerowania różnych kluczy dla każdego cyklu - iteracji, korzysta z algorytmu rozszerzenia klucza głównego. RC5 używa w tym celu dwóch liczb P_w oraz Q_w o wielkości słowa w . Są one zdefiniowane następująco:

$$P_w = \text{Odd}((e - 2) 2^w)$$

$$Q_w = \text{Odd}((\phi - 1) 2^w)$$

gdzie:

$$e = 2.718281828459\dots \text{ (podstawa logarytmu naturalnego)}$$

$$\phi = 1.618033988749\dots \text{ (wartość złotej liczby)}$$

$\text{Odd}(x)$ jest funkcją zaokrąglającą wartość x w górę do najbliższej liczby całkowitej.

Wartości P_w oraz Q_w dla długości słowa $w=16, 32$ oraz 64 bity przedstawione zostały w poniższej tabelce.

w	P_w (hex)	Q_w (hex)
16	b7e1	9e37
32	b7e15163	9e3779b9
64	b7e151628aed2a6b	9e3779b97f4a7c15

Algorytm ten wykonywany jest w trzech krokach:

1. konwersji bajtów klucza K do postaci słów - bity klucza głównego $K[0\dots b-1]$ kopiowane są do tablicy $L[]$ o wymiarach $8b/w$ słów w -bitowych. W niewypełnione pozycje tablicy L wpisywane są wartości 0. Algorytm procedury realizującej powyższy krok przedstawia się następująco:

```
for i = b - 1 downto 0 do
    L[8i/w] = (L[8i/w] << 8) + K[i] ;
```

2. inicjalizacji tablicy kluczy cyklu S pseudolosową sekwencją bitową, z użyciem wartości Q_w oraz P_w . Działania arytmetyczne wykonywane w tym kroku są działaniami wykonywanymi zgodnie z arytmetyką skończonego ciała $\text{GF}(2^w)$.

```
S[0] = P_w ;
for i=1 to t - 1 do
    S[i] = S[i - 1] + Q_w ;
```

3. pomieszania wartości klucza K z wartościami tablicy S . Mieszanie odbywa się w trzech identycznych krokach. Ponieważ tablice S oraz L są różnej wielkości, większa z

tych tablic jest przetwarzana trzykrotnie. Algorytm tej operacji przedstawia się następująco:

```
i = j = 0 ;
A = B = 0 ;
do 3 * max (t, c) times
  A= S[i] = (S[i] + A +B) <<< 3 ;
  B= L[j] = (L[j] + A +B) <<< (A + B) ;
  i = (i +1) mod (t) ;
  j = (j +1) mod (c) ;
```

4.2.5 Implementacje RC5.

Algorytm RC5 jest prosty w implementacji zarówno sprzętowej jak i programowej, co jest zawdzięczane w bardzo dużej mierze elastyczności algorytmu, pozwalającej doskonale zoptymalizować implementacje pod kątem używanego sprzętu. Brak użycia S-boxów pozwala zredukować wielkość pamięci, zaś użycie prostych operacji algebraicznych sugeruje bardzo dużą szybkość działania algorytmu na procesorach 8 bitowych, 32 bitowych a także na strukturach FPGA oraz kartach pamięci *smartcard*. Wyniki wykonane przez twórców szyfru przedstawiają się następująco:

kompilator 16 bit Borland C++, CPU 486 50MHz: prędkość szyfrowania: 100KB/sec

Assembler, CPU 486SLC 50MHz: prędkość szyfrowania: 1,2MB/sec

Sparc 5, kompilator gcc: prędkość szyfrowania: 2,4MB/sec

4.2.6 Siła kryptograficzna RC5

Istotną rzeczą w rozważaniach na temat siły kryptograficznej szyfru jest jego odporność na liniową i różnicową kryptoanalizę. Wyniki badań dotyczące tej odporności zostały przedstawione podczas Crypto 95, przez B.S. Kaliski Jr. oraz Y.L. Yin²³. Potwierdzona została teza Rivesta, że RC5 jest w pełni odporny na atak typu DC i LC po 12 cyklach. B.S. Kaliski Jr. oraz Y.L. Yin pokazują, że atak DC na RC5 jest nieefektywny i nieopłacalny już dla liczby cykli wynoszącej 6, gdyż wymaga 2^{31} par tekstów jawnych. Należy w tym miejscu również dodać, że podczas owej kryptoanalizy używano 5 charakterystyk, w tym jednej z prawdopodobieństwem równym 1 zaś pozostałych z $P > 1/2w$.

²³ Zob. B.S.Kaliski, Y.L.Yin, On Differential and Linear Cryptanalysis of the RC5 Encryption Algorithm, Crypto 95, Springer Verlag 1998, także B.S.Kaliski, Y.L.Yin, On the Security of the RC5 Encryption Algorithm, RSA Laboratories Technical Report TR602, version 1.0-September 1998

Dla 12 cykli liczba tekstów jawnych wynosi 2^{62} . Przeprowadzając kryptoanalizę liniową w/w znaleźli aproksymację liniową dla połowy cyklu. Na podstawie tego, pokazano, że liczba tekstów jawnych dla LC RC5 wynosi $N=w*4w^{2(r-1)}$, co dla długości słowa $w=32$ daje: $N=2^{37}$ dla czterech cykli, $N=2^{47}$ dla pięciu cykli oraz $N=2^{57}$ dla sześciu cykli. Analiza ta jest więc efektywna tylko dla małej liczby cykli, co w przypadku większej liczby cykli czyni ją całkowicie bezużyteczną.

Kluczem do uzyskania odporności na LC i DC są cykliczne przesunięcia w rejestrach.

Na Crypto 96, W. Meier oraz L.R. Knudsen przedstawili²⁴ nieco inną propozycję ataku DC na RC5. Poprzez analizę przesunięć cyklicznych, złożoność przedstawiona przez nich DC, charakteryzowała się 512 krotnie mniejszą złożonością. Poprzez analizę algorytmu rozszerzenia klucza, pokazali oni również, że RC5 ma słabe klucze, dzięki którym DC może okazać się jeszcze bardziej skuteczniejsza.

Kolejne postępy w kryptoanalizie DC szyfru RC5 zostały przedstawione na Eurocrypt 98. Biryukow i Kushilevitz zaprezentowali²⁵ atak za pomocą DC który dla 64 bitowej długości słowa i 12 cykli wymagał użycia 2^{44} par tekstów jawnych.

Powyższy atak, jednocześnie pokazuje postęp jaki dokonał się w kryptoanalizie różnicowej w ciągu zaledwie trzech lat. Jest to również najlepszy z przeprowadzonych ataków na RC5, dowodzący, że dla 64 bitowej długości słowa oraz 12 cykli, RC5 jest tak samo bezpieczny jak 16 cykli DES. Analizując powyższe dokonania, można stwierdzić, że RC5 jest nawet silniejszy od DES, w następującym aspekcie: - jest on szybszy, łatwiejszy we wszelakiego rodzaju implementacjach. Czas wykonania 16 cykli RC5 jest przybliżony do 16 cykli DES, przy zwiększonym w stosunku do 12 cykli RC5 bezpieczeństwie.

²⁴ L.R.Knudsen, W.Meier, Improved Differential Attacks of RC5, Crypto 96, Springer Verlag 1998

²⁵ A.Biryukov, E.Kushilevitz, Improved Cryptanalysis of RC5

4.3 RC6

Mimo dużego bezpieczeństwa RC5, oraz faktu, że do roku 1998 nie został przeprowadzony żaden praktyczny atak na ten szyfr, dostępne analizy pokazały ewentualne drogi, którymi taki atak mógł zostać w przyszłości przeprowadzony - oparte na fakcie, że nie wszystkie bity wpływają na wartość przesunięć w rejestrach. Kiedy 2 stycznia 1997r ogłoszono konkurs na szyfr mający stać się nowym standardem szyfrowania danych, mającym zastąpić DES, rozważano kandydaturę RC5. NIST określił wymagania jednoznacznie. Nowy szyfr ma być bardziej bezpieczny niż 3DES, ma pracować z długościami klucza 128, 192 oraz 256 bitów, a wielkość przetwarzanego bloku ma wynosić min. 128 bitów.

RC5 spełniał te wymagania - dla odpowiedniej ilości cykli był zapewniał określony poziom bezpieczeństwa, mógł przetwarzać powyższe wielkości bloków. Problemem jednak była szybkość działania szyfru oraz ograniczone możliwości implementacyjne dla takich parametrów. Minimalna długość słowa wynosiła by wówczas 64 bity, co pozwoliło by efektywnie zaimplementować RC5 jedynie na 64 bitowych procesorach. Zwiększeniu uległa by również liczba cykli, co dodatkowo zmniejszyło by szybkość pracy szyfru. Stało się oczywiste, że aby spełnić powyższe wymagania przy zapewnieniu odpowiedniego poziomu bezpieczeństwa i szybkości pracy należy zaprojektować nowy szyfr.

Szyfrem takim został zaprojektowany²⁶ przez R. Rivesta, M.J.B Robshawa, R. Sidneya oraz Y.L.Yin szyfr RC6. Głównymi założeniami postawionymi przy projektowaniu RC6 były identyczne założenia jak dla RC5 oraz spełnienie wymogów postawionych przez NIST. RC6 stanowi więc rozwinięcie szyfru RC5. Podobnie jak jego poprzednik, oparty jest na cyklicznych przesunięciach danych w obrębie rejestrów. Główne zmiany wprowadzone w RC6 obejmują użycie czterech rejestrów roboczych oraz wprowadzenie dodatkowej operacji algebraicznych w ciele $GF(2^w)$ - mnożenia. Mnożenie to, doskonale poprawia charakterystyki warstwy dyfuzyjnej, co zwiększa bezpieczeństwo szyfru.

4.3.1 Opis algorytmu

Podobnie jak RC5, RC6 jest elastycznym algorytmem, który jest opisany identycznym parametrami jak RC5, których notacja jest następująca:

RC6 - $w / r / b$

gdzie:

²⁶ R.L.Rivest, M.J.B Robshaw, R.Sidney, Y.L.Yin, The RC6 Block Cipher, RSA Laboratories , version1.1, August 1998

w - wielkość słowa w bitach,

r - liczba cykli,

b - długość klucza głównego K w bajtach.

Parametry te mają identyczne wielkości jak RC5. Ponieważ jednak RC6 pracuje na czterech rejestrach roboczych, dla długości słowa $w=32$ bity, wielkość przetwarzanego bloku wynosi 128 bitów.

Dla długości klucza sprecyzowanych przez NIST, tzn. 128, 192, 256 bitów, parametr b przyjmuje odpowiednio wartości $b = 16, 24$ oraz 32 bajty.

RC6 używa rozszerzonej w stosunku do RC5 liczby operacji algebraicznych. Ich notacja przedstawia się następująco:

$a + b$	-suma dwóch liczb w ciele skończonym ciele $GF(2^W)$
$a - b$	-różnica dwóch liczb w ciele skończonym ciele $GF(2^W)$
$a \oplus b$	-suma xor - modulo 2, dwóch liczb całkowitych
$a \times b$	- mnożenie całkowite liczb w skończonym ciele $GF(2^W)$
$a \lll b$	- cykliczne przesunięcie bitów w lewo w rejestrze A, o wartość daną przez 5 mniej znaczących bitów rejestru b
$a \ggg b$	- cykliczne przesunięcie bitów w prawo w rejestrze A, o wartość daną przez 5 mniej znaczących bitów rejestru b

Algorytm rozszerzenia klucza w RC6 jest identyczny jak algorytm w RC5. Jediną różnicą jest fakt, że lista kluczy cyklu jest dłuższa - w każdym cyklu używane są cztery klucze, a nie dwa, jak w RC5 .

4.3.2 Szyfrowanie

RC6 wykorzystuje podczas szyfrowania cztery w -bitowe rejestry robocze oznaczane jako A, B, C, D , zawierające na początku procesu szyfrowania tekst jawny, a na końcu szyfrowania szyfrogram. Pierwszy bit zarówno tekstu jawnego jak i szyfrogramu umieszczany jest jako najmniej znaczący bit A, ostatni bit - jako najbardziej znaczący bit rejestru D. Algorytm szyfrowania za pomocą RC6 można przedstawić następująco:

```

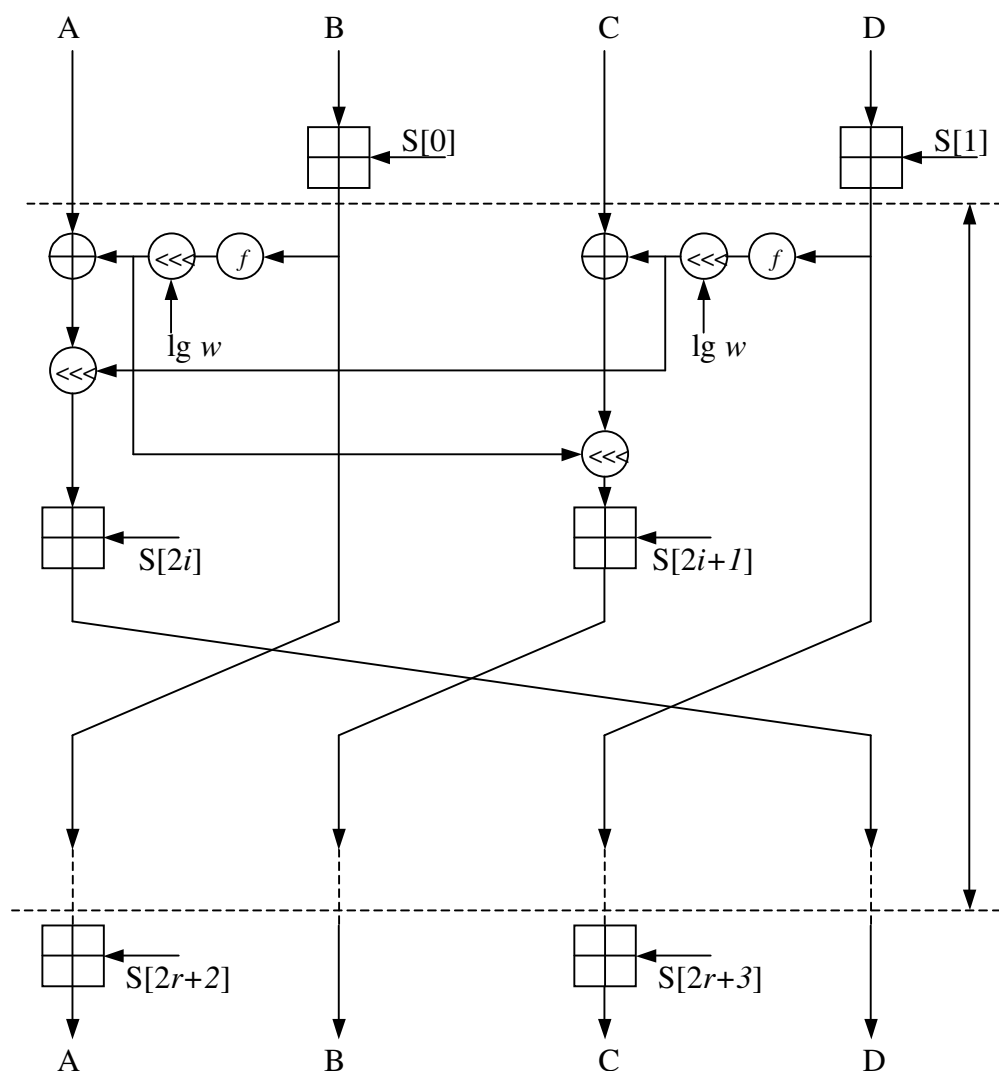
B = B + S[0]
D = D + S[1]
for i = 1 to r do
{
    t = (B x (2B + 1)) <<< lg2(w)
    u = (D x (2D + 1)) <<< lg2(w)

```



$$\begin{aligned}
 A &= ((A \oplus t) \lll u) + S[2i] \\
 C &= ((C \oplus u) \lll t) + S[2i + 1] \\
 (A, B, C, D) &= (B, C, D, A) \\
 & \} \\
 A &= A + S[2r + 2] \\
 C &= C + S[2r + 3]
 \end{aligned}$$

Schemat blokowy cyklu pracy algorytmu, w formie graficznej, przedstawiony został poniżej.



Rys. 4.2 Schemat cyklu szyfrowania za pomocą RC6

4.3.3 Deszyfrowanie

Deszyfrowanie za pośrednictwem RC6 jest procedurą wykorzystującą odwrotne operacje algebraiczne w stosunku do operacji wykorzystywanych podczas szyfrowania. Operacje te przeprowadzane są również w odwrotnej kolejności. Algorytm deszyfrowania przedstawia się więc następująco:

```

C = C - S[2r + 3]
A = A - S[2r + 2]
for i = r downto 1 do
  {
    (A, B, C, D) = (D, A, B, C)
    u = (D x (2D + 1)) << lg2(w)
    t = (B x (2B + 1)) << lg2(w)
    C = ((C - S[2i + 1] >>> t) ⊕ u
    A = ((A - S[2i] >>> u) ⊕ t
  }
D = D - S[1]
B = B - S[0]

```

4.3.4 Przesłanki projektowe

Głównymi przesłankami projektowymi dla RC6 były:

- prostota,
- bezpieczeństwo,
- wydajność

Analizując budowę RC6, można ją przyrównać w stosunku do RC5 jako do:

1. wykonania połowy cyklu RC5
2. równoległego wykonania dwóch cykli RC5, jednego cyklu na rejestrach A i B , drugiego na rejestrach C i D z następującym wyjątkiem: zamiast zamiany zawartości A i B oraz C i D , dokonania permutacji: $(A,B,C,D)=(B,C,D,A)$
3. dokonania pomieszania par rejestrów A , B oraz C,D z uwzględnieniem wartości współczynników cyklicznych rotacji.

Szyfr został jednak zaprojektowany w nieco inny sposób. Próby dokonania kryptoanalizy różnicowej RC5, mimo iż okazały się być nieefektywne, pokazały, że jako par różnicowych najlepiej używać par nie zmieniających wartości współczynników rotacji w rejestrach.

Aby zapobiec łatwemu wyborowi takich par, wartości współczynników cyklicznej rotacji, które określane są za pomocą ostatnich pięciu bitów rejestrów B oraz D powinny być zależne od każdego z bitów słowa wejściowego. Należy więc wprowadzić dodatkową transformację, dzięki której powyższa własność, oznaczająca polepszenie własności warstwy dyfuzyjnej szyfru, będzie spełniona. Zadanie to realizuje funkcja kwadratowa $f(x) = x(2x+1) \bmod 2^w$. Funkcja ta, postaci $f(x) = x(ax+b) \bmod 2^w$, dla parzystej wartości a oraz nieparzystej wartości b jest odwzorowaniem typu 1 do 1, stanowi ona więc permutację. Analizując własności algebraiczne tej funkcji, można zauważyć, że zmiana jednego z bitów wejściowych, powoduje zmianę wartości wielu bitów wyjściowych w niemożliwych do łatwego wyliczenia

pozycjach. Zastosowanie funkcji $f(x) = x(2x+1) \bmod 2^w$ daje więc doskonały efekt lawinowy, w aspekcie dyfuzji bitowej. Dzięki temu prawdopodobieństwo, zdarzenia w którym dwa losowo wybrane teksty jawne, spowodują wystąpienie identycznych wartości współczynników rotacji w rejestrach, jest znikome.

Poprzez poprawę własności warstwy dyfuzyjnej, znacząco poprawiona została odporność szyfru na kryptoanalizę różnicową.

Aby uodpornić szyfr na działanie kryptoanalizy liniowej, zdecydowano wprowadzić stałą liczbę bitów 5 definiującą wartość cyklicznej rotacji rejestrów²⁷.

4.3.5 Wydajność oraz możliwości implementacji RC6

Pomiary wydajności RC6 32/16/16, dokonane przez jego twórców zostały przedstawione w poniższej tabeli:

Środowisko	operacja	cykli/blok	bloków/sek	MBytes/sec
ANSI C	szyfrowanie	616	325000	5,19
	deszyfrowanie	566	353000	5,65
JAVA	szyfrowanie	1010	197000	3,15
	deszyfrowanie	955	209000	3,35
ASM	szyfrowanie	254	787000	12,6
	deszyfrowanie	254	788000	12,6

Dla porównania przedstawione zostały wartości uzyskane dla RC5 32/16/16 przy szyfrowaniu

ANSI C	szyfrowanie	328	610000	4,9
JAVA	szyfrowanie	1140	175000	1,4
ASM	szyfrowanie	148	1350000	10,8

Powyższe pomiary zostały przeprowadzone w następujących środowiskach:
ANSI C, ASM: PII 200Mhz, 32MB RAM, Windows 95, C++ Borland Dev. Suite 5.0
JAVA: PPro 200Mhz, 64 MB RAM, WinNT4.0+SrvPack4, Javasoft JDK 1.16 +JIT2.1

W przeciwieństwie do wielu szyfrów, implementacja RC6 nie pociąga za sobą konieczności stosowania tablic, co pozwala na znaczną redukcję pamięci. Wszelkie dane mogą być więc przechowywane w pamięci cache, co pozwala znacznie przyspieszyć prędkość działania szyfru.

RC6 jako jeden z nielicznych może zostać również zaimplementowany w strukturach FPGA rodziny XC4000 firmy Xilinx (rodziny prostych układów FPGA). Dla wielkości klucza oraz bloku równych 128 bitów prędkość szyfrowania wyniosła 44Mbit/sec. Przy zastosowaniu

²⁷ Dokładne przesłanki projektowe oraz analizę bezpieczeństwa można znaleźć w R.L.Rivest, M.J.B Robshaw, R.Sidney, Y.L.Yin, The Security of the RC6 Block Cipher, RSA Laboratories, version 1.0, August 1998

układów Xilinx Virtex XCV1000BG560-54 prędkość ta wyniosła 103,9 Mbit/s²⁸. Przy implementacjach wielopotokowych na tych samych układach, RC6 32/16/16 pracujący w trybie bez sprzężenia zwrotnego, w implementacji zorientowanej pod względem uzyskania maksymalnej szybkości, osiągnął prędkość 2,4Gbit/s, zaś w trybie ze sprzężeniem zwrotnym 126,5 Mbit/s²⁹.

4.3.6 Siła i bezpieczeństwo RC6

Ponieważ, tak jak RC5, tak i RC6 należy do szyfrów bardzo elastycznych, jego siła i bezpieczeństwo w dużej mierze zależą od wyboru parametrów. Dla długości klucza równej 128 bitów, atak polegający na wyczerpującym poszukiwaniu klucza charakteryzuje się zbyt wielką złożonością obliczeniową.

Odporność na kryptoanalizę różnicową

Dzięki zastosowaniu funkcji kwadratowej przyspieszony został bardzo efekt lawinowy. Konkatenacja efektów działania funkcji kwadratowej oraz rotacji bitowych o współczynnikach wyznaczanych przez 5 bitów rejestrów B oraz D, bardzo utrudnia tworzenie dobrych charakterystyk różnicowych. RC6 wykazuje pełną odporność na kryptoanalizę różnicową po 20 cyklach. Ilości tekstów par jawnych przeznaczonych do dokonania tego rodzaju kryptoanalizy, przy użyciu jednej z najlepszych dla RC6 sześć-cyklowych charakterystyk iteracyjnych, spełnianej z prawdopodobieństwem $p=2^{-91}$, w zależności od liczby cykli przedstawione zostały poniżej. Miarą zróżnicowania tekstów jawnych była w tym przypadku nie suma XOR, ale bezpośrednia różnica arytmetyczna.

Rodzaj charakterystyki	Liczba cykli				
	8	12	16	20	24
iteracyjna	2^{93}	2^{151}	2^{214}	2^{279}	2^{337}
charakterystyka 6 cykli powstała z połączenia różnych charakterystyk nieiteracyjnych	2^{56}	2^{117}	2^{190}	2^{238}	2^{290}

Rys. 4.3 Liczba wymaganych tekstów jawnych potrzebnych do przeprowadzenia ataku DC szyfru RC6, w zależności od rodzaju użytych charakterystyk.

²⁸ Na podstawie K.Gaj, P.Chodowicz, Comparison of the hardware performance of the AES candidates using reconfigurable hardware, AES3 Proceedings, April 2000

²⁹ Na podstawie: J Elbirt1, W Yip, B Chetwynd, C Paar, An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists, AES3 Proceedings, April 2000

Odporność RC6 na kryptoanalizę liniową

W przypadku RC5, kryptoanaliza liniowa dokonana³⁰ przez B.S. Kaliski Jr. oraz Y.L. Yin pokazuje, że liczba danych potrzebnych do przeprowadzenia takiego ataku może przekroczyć liczbę dostępnych danych już po sześciu cyklach. przy bloku długości 64 bitów. W przypadku bloku długości 128 bitów, kryptoanaliza liniowa, choć nieefektywna, okazała by się możliwa. RC6 musi być zatem odporniejszy na działanie LC niż RC5.

Istnieją dwie metody dokonania kryptoanalizy liniowej szyfru RC6. Pierwsza z nich polega na aproksymacji samej rotacji w obrębie rejestru, zaś druga na aproksymacji funkcji f oraz rotacji. Najefektywniejsze aproksymacje można znaleźć dla pierwszej z metod. Aproksymacje te są efektywne tylko dla małej liczby cykli, gdyż ich współczynnik liniowości spada gwałtownie wraz ze wzrostem liczby iteracji. Używając efektywnych charakterystyk, w postaci zespołu aproksymacji oraz innych bardzo zaawansowanych technik LC (*linear hulls*), dokonanie takiego ataku jest teoretycznie możliwe, chociaż wysoce nieefektywne, z uwagi na ilość danych potrzebnych do jego przeprowadzenia. Ilość ta, w zależności od rodzaju użytych technik oraz liczby cykli została pokazana poniżej.

Wariant LC	Liczba cykli				
	8	12	16	20	24
podstawowy (jedna aproksymacja)	2^{62}	2^{102}	2^{142}	2^{182}	2^{222}
podstawowy z wieloma aproksymacjami	2^{51}	2^{91}	2^{131}	2^{171}	2^{211}
bardzo zaawansowany	2^{47}	2^{83}	2^{119}	2^{155}	2^{191}

Rys. 4.4 Liczba wymaganych tekstów jawnych potrzebnych do przeprowadzenia ataku LC szyfru RC6, w zależności od rodzaju użytych charakterystyk.

Na podstawie powyższej tabeli można jednoznacznie stwierdzić, że RC6 jest w pełni odporny na LC (czyniąc ją niemożliwą) po 20 cyklach. Tak wysoka odporność zawdzięczana jest w największej mierze dwóm transformacjom: funkcji f , oraz cyklicznej rotacji o stałą liczbę 5 pozycji.

³⁰ Zob. R.L.Rivest, M.J.B Robshaw, R.Sidney, Y.L.Yin, The Security of the RC6 Block Cipher, RSA Laboratories, version 1.0, August 1998

Bezpieczeństwo algorytmu rozszerzenia klucza.

Ponieważ dla RC5 nie znaleziono dotychczas żadnej słabości w procedurze rozszerzenia klucza związanej ze słabymi kluczami oraz nie została zanotowana próba przeprowadzenia ataku metodą kluczy powiązanych, można stwierdzić, że algorytm rozszerzenia klucza RC6 jest w pełni bezpieczny³¹.

Analizując budowę szyfru, można jednak zauważyć, że w przypadku implementacji sprzętowych RC6, należy zwrócić uwagę, na podatność tego szyfru na atak za pomocą różnicowego poboru mocy. Obserwując charakterystyki prądowe można dokonać odczytu wartości współczynnika rotacji w rejestrach, co może ułatwić późniejszą kryptoanalizę.

³¹ Stwierdzenie to zostało oparte na S. Contini, R.L. Rivest, M.J.B Robshaw, Y.L.Yin, On Security of the RC6 Block Cipher v.1.0, RSA Laboratories 1998

Rozdział 5

Rijndael - Advanced Encryption Standard

5.1 Wstęp

DES, szyfr używany dotychczas jako standard szyfrowania danych okazał się być na dzisiejsze czasy za mało bezpieczny. Dysponując odpowiednim nakładem finansowym, możliwe jest zbudowanie maszyny łamiącej DES w czasie kilku godzin. Wraz ze wzrostem rozwoju technologicznego, czas ten ulegnie dalszemu skróceniu. Problem ten, był czynnikiem sprawczym poszukiwania następcy DES, poszukiwania nowego szyfru gwarantującego odpowiedni poziom bezpieczeństwa, szyfru który mógłby zostać uznany za standard przez najbliższą dekadę.

2 stycznia 1997 NIST ogłosił rozpoczęcie poszukiwań kandydata na nowy standard szyfrowania danych AES, określając minimalne wymagania następująco:

- dokumentacja szyfru musi być powszechnie dostępna,
- szyfr ma należeć do grupy blokowych szyfrów symetrycznych,
- projekt szyfru musi zakładać możliwość rozszerzenia długości klucza, w razie takiej potrzeby,
- szyfr musi być łatwy do implementacji zarówno sprzętowej jak i programowej,
- nowy szyfr ma być bardziej efektywny oraz bardziej bezpieczny niż 3-DES, co implikuje następujące wymagania:
 - o długość klucza określona jako 128, 192 lub 256 bitów
 - o wielkość bloku równa 128, 192 lub 256 bitów

Rozpatrując odpowiednią kandydaturę, brano również pod uwagę następujące aspekty:

- zapewniany poziom bezpieczeństwa w odniesieniu do struktury oraz złożoności,
- koszty implementacji oraz jej możliwości,
- elastyczność dotycząca nie tylko parametrów szyfru, ale i przystosowania do pełnienia innych kryptograficznych funkcji: szyfrów strumieniowych, funkcji haszujących..

Spośród piętnastu kandydatów rundy pierwszej, do ścisłego finału zakwalifikowało się 5 szyfrów:

- Mars - szyfr zaprojektowany przez IBM,
- RC6 - RSA Laboratories,
- Rijndael - Joan Daemen, Vincent Rijmen,
- Serpent - Ross Anderson, Eli Biham, Lars Knudsen,
- Twofish - Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson

Wśród tych pięciu finalistów przeprowadzono serię dodatkowych badań: kryptoanaliz oceniających bezpieczeństwo, pomiarów szybkości w poszczególnych implementacjach, możliwości do zaadoptowania w nowych rozwiązaniach.

Spośród wszystkich kandydatów, najlepiej został oceniony Rijndael, stając się nowym standardem szyfrowania danych. Z uwagi na ten fakt, obok nazwy Rijndael, obowiązująca nazwa brzmi AES - Advanced Encryption Standard

5.2 Opis algorytmu

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 zależy zarówno od długości klucza jak i od długości bloku. Zależności te, dla wartości uznanych jako standardowe przedstawiono w poniższej tabelce

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

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.

5.3 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 wygenerowaniem 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*.

5.4 Opis poszczególnych transformacji

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

5.4.1 SubByte

SubByte jest transformacją, podstawiającą 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

$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad (1)$$

dla $0 \leq i < 8$ gdzie b_i jest i -tym bitem bajtu b zaś c_i jest i -tym bitem bajtu 01100011. Transformację tą wygodnie jest przedstawić w postaci macierzowej jako:

$$\begin{bmatrix} b_7' \\ b_6' \\ b_5' \\ b_4' \\ b_3' \\ b_2' \\ b_1' \\ b_0' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

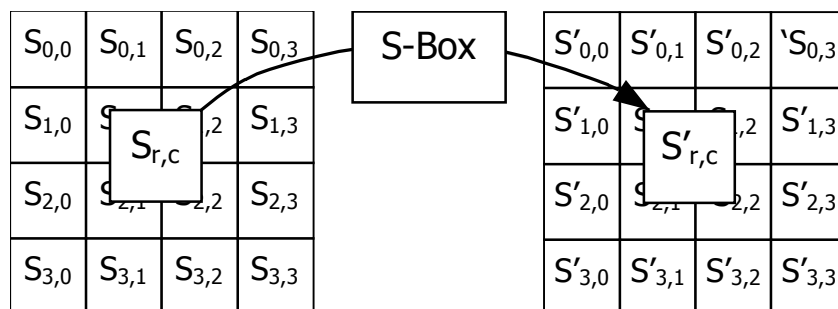
Możliwe jest zatem wyliczenie powyższych przekształceń dla każdej z wartości bajtów i stworzenie odpowiedniej tablicy podstawień, określanej jako S-box.

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	c9	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	d6	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	aa	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	c4	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	c2	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

Rys. 5.1 S-box Rijndaela używany w transformacji SubByte



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 odpowiednią kolumnę zdefiniowaną przez y . Przykładowo $B=\{4F\}$, po transformacji SubByte przyjmie wartość równą $B=\{84\}$. Efekt działania funkcji SubByte przedstawia poniższy rysunek:



Rys 5.2 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 (2), 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	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	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	a8	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

Rys 5.3 S-box realizujący przekształcenie odwrotne w stosunku do SubByte

Operację odwrotną do (1) można zapisać jako:

$$b'_i = b_{(i+2) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus d_i \quad (2)$$

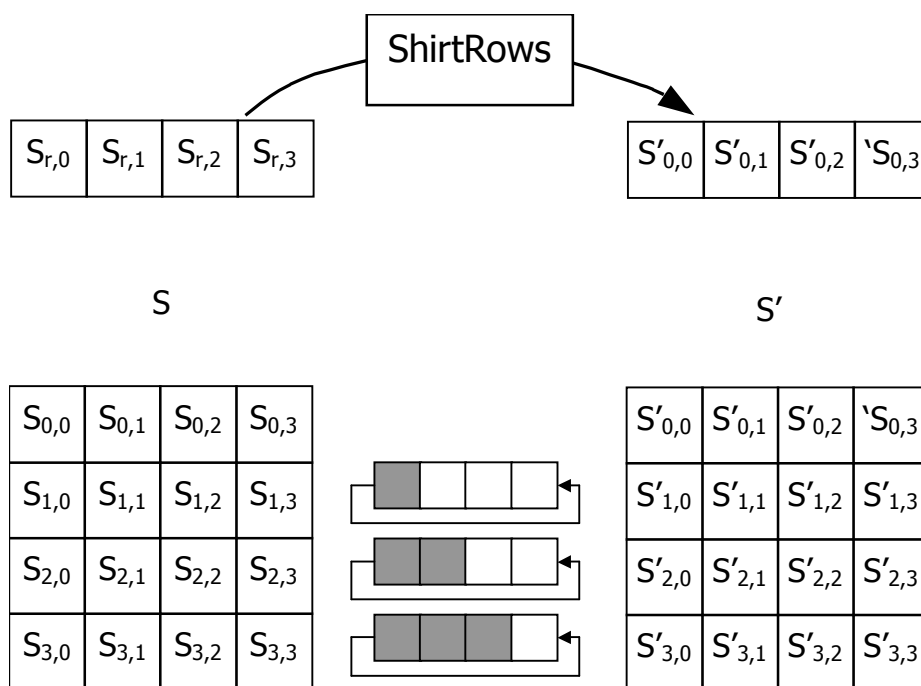
dla $0 \leq i < 8$ gdzie b_i jest i -tym bitem bajtu b zaś d_i jest i -tym bitem bajtu $d=05_x = 00000101$

Transformację InvSubByte można również przedstawić za pomocą tablicy podstawień, InvSBox będącej odwrotnością tablicy S-box.

W opisie tym cała tablica jest traktowana jako jeden S-box. W rzeczywistości jednak mamy do czynienia z transformacją złożoną z 256 S-boxów. Wartość bajtu w macierzy *State*, określa aktywny dla niego S-box.

5.4.2 ShiftRow

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



Rys. 5.4 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.

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

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 $(j+N_B-C_i) \bmod N_B$.

5.4.3 MixColumn

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

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

Niech $s'(x) = a(x) \cdot s(x)$, 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:

$$b(x) = '0B'x^3 + '0D'x^2 + '09'x + '0E'$$

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:

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

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

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

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

5.4.4 AddRoundKey

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

5.4.5 Procedura rozszerzenia klucza - *Key Expansion*

Klucze cyklu K_i są generowane za pomocą procedury rozszerzenia klucza głównego K . Całkowita liczba wygenerowanych kluczy cyklu K_i zależna jest od ilości cykli, co w przypadku Rijndaela związane jest z wielkością bloku N_B oraz długością klucza głównego N_K ; wyrażona jest ona w postaci iloczynu: $N_B(N_r+1)$. Długość klucza cyklu równa jest



długości bloku. Klucz cyklu można zatem przedstawić za pomocą macierzy o wymiarach $[4 \times N_B]$, równej wymiarami macierzy *State*. Generowany klucz cyklu rozpatrywać należy nie w kategorii bajtów, lecz w kategorii słów o długości 32 bitów. Każde słowo ma oznaczenie W_i , gdzie $i \bmod N_B$ odpowiada numerowi kolumny w macierz reprezentującej klucz.

Przebieg procedury rozszerzenia klucza zależy od długości klucza głównego K . Jeśli ma on długość mniejszą bądź równą 192 bity ($N_K \leq 6$), jej przebieg jest następujący:

```

KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)])
{
  for(i = 0; i < Nk; i++)
    W[i] = (Key[4*i], Key[4*i+1], Key[4*i+2], Key[4*i+3]);
  for(i = Nk; i < Nb * (Nr + 1); i++)
  {
    temp = W[i - 1];
    if (i % Nk == 0)
      temp = SubWord(RotWord(temp)) ^ Rcon[i / Nk];
    W[i] = W[i - Nk] ^ temp;
  }
}

```

Funkcja *SubWord* jest funkcją zwracającą 4 bajtowe słowo, powstałe w wyniku podstawienia każdemu z bajtów składowych wartości określonej przez S-box transformacji *SubBytes*. *RotWord* jest funkcją dokonującą cyklicznej rotacji bajtów, formułujących 32 bitowe słowo, w lewo o jedną pozycję, tak że słowo wejściowe postaci {a,b,c,d} pod wpływem *RotWord* przyjmuje postać {b,c,d,a} *Rcon[i]* jest stałym wielomianem, przyjmującym inną wartość dla każdego cyklu, zdefiniowanym następująco:

$$Rcon[i/N_K] = [Rc[i], \{00\}, \{00\}, \{00\}]$$

gdzie $Rc[i]$ jest potęgą liczby 02^{i-1}_x w $GF(2^8)$, zaś i jest numerem cyklu tak, że:

$$Rc[1] = 01_x,$$

$$Rc[i] = 02 \bullet (Rc[i-1])$$

Jeśli klucz główny ma długość 256 bitów, wówczas procedura rozszerzenia klucza składa się z następujących transformacji:

```

KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)])
{
  for(i = 0; i < Nk; i++)
    W[i] = (key[4*i], key[4*i+1], key[4*i+2], key[4*i+3]);
  for(i = Nk; i < Nb * (Nr + 1); i++)
  {
    temp = W[i - 1];
    if (i % Nk == 0)
      temp = SubWord(RotWord(temp)) ^ Rcon[i / Nk];
    else if (i % Nk == 4)

```

```
temp = SubWord(temp);  
W[i] = W[i - Nk] ^ temp;  
}  
}
```

Analizując obydwie listingi, można zauważyć, że pierwsze N_K słów rozszerzonego klucza składa się z klucza głównego, zaś każde z następnym słów W_i jest równe sumie xor słowa poprzedniego W_{i-1} oraz słowa pozycji N_K wcześniejszej W_{i-N_K} . Dla słów których numery są krotnościami N_K dokonywany jest zespół transformacji zdefiniowanych przez SubWord oraz RotWord.

5.5 Deszyfrowanie

Zgodnie ze strategią proponowaną przez J.Daemena³² 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])  
}
```

5.6 Aspekty i wskazówki dotyczące implementacji

Rijndael został zaprojektowany w taki sposób, aby możliwe było jak najlepsze zaimplementowanie szyfru na szerokiej gamie procesorów i sprzęcie specjalnego przeznaczenia. Rijndael bardzo dobrze nadaje się również do implementacji w rozwiązaniach równoległych.

³² Strategia wide trail strategy, która jest fundamentem Rijndaela, stanowi jedynie fragment strategii projektowania szyfrów blokowych. Dokładne informacje, kryteria odwracalności transformacji, itd. można znaleźć w: J.Daemen, Cipher and hash function design strategies based on linear and differential cryptanalysis, Doctorial Dissertation, March 1995, K.U.Leuven

Implementacji transformacji SubBytes oraz transformacji do niej odwrotnej najlepiej dokonać używając tablic definiujących S-box oraz InvSbox. Każda z tych tablic wymaga 256 bajtów pamięci.

5.6.1 Procesory 8-bit

Na 8-bitowych procesorach jedyną trudnością może być efektywna implementacja transformacji MixColumn. Problem ten można rozwiązać poprzez odpowiednie przekształcenie algebraiczne poszczególnych równań definiujących transformację, oraz wykorzystanie faktu, że mnożenie przez wartość 02, stanowi zwykle przesunięcie bitów w lewo. Transformację MixColumn można więc opisać jako:

$$\begin{aligned} t &= c[0] \oplus c[1] \oplus c[2] \oplus c[3] \\ u &= c[0] \oplus t \oplus ((c[0] \oplus c[1]) \lll 1) \\ c[1] &= c[1] \oplus t \oplus ((c[1] \oplus c[2]) \lll 1) \\ c[2] &= c[2] \oplus t \oplus ((c[2] \oplus c[3]) \lll 1) \\ c[3] &= c[3] \oplus t \oplus ((c[3] \oplus c[0]) \lll 1) \\ c[0] &= u \end{aligned}$$

Znacznie gorsza jest realizacja operacji odwrotnej InvMixColumns, co wynika z faktu, iż wykorzystuje ona do mnożenia inny wielomian. Współczynniki tego wielomianu: '09', '0E', '0B' oraz '0D' powodują, że poszczególne operacje mnożenia są znacznie bardziej czasochłonne, przez co obserwuje się znaczny spadek wydajności.

5.6.2 Procesory 32-bit.

Traktując każdą z kolumn macierzy stanów jako wektor 32-bitowy, przebieg cyklu w postaci macierzowo wektorowej można opisać następująco:

po SubBytes

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} S[S_{0,c}] \\ S[S_{1,c}] \\ S[S_{2,c}] \\ S[S_{3,c}] \end{bmatrix}$$

po ShiftRows

$$\begin{bmatrix} S''_{0,c} \\ S''_{1,c} \\ S''_{2,c} \\ S''_{3,c} \end{bmatrix} = \begin{bmatrix} S[S_{0,c}] \\ S[S_{1,[c+h(1,Nk)] \bmod Nk}] \\ S[S_{2,[c+h(2,Nk)] \bmod Nk}] \\ S[S_{3,[c+h(3,Nk)] \bmod Nk}] \end{bmatrix} = \begin{bmatrix} S[S_{0,c(0)}] \\ S[S_{1,c(1)}] \\ S[S_{2,c(2)}] \\ S[S_{3,c(3)}] \end{bmatrix}$$

gdzie $c(r) = [c+h(r,Nc)] \bmod Nc$, jest funkcją opisującą wartość cyklicznego przesunięcia

po MixColumns

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

po AddRoundKey

$$\begin{bmatrix} S''''_{0,c} \\ S''''_{1,c} \\ S''''_{2,c} \\ S''''_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[S_{0,c(0)}] \\ S[S_{1,c(1)}] \\ S[S_{2,c(2)}] \\ S[S_{3,c(3)}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,c} \\ k_{1,c} \\ k_{2,c} \\ k_{3,c} \end{bmatrix}$$

Traktując powyższe operacje jako jedną złożoną, można ją opisać w postaci wektorów jako:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = S[S_{0,c(0)}] \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S[S_{1,c(1)}] \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S[S_{2,c(2)}] \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus S[S_{3,c(3)}] \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \oplus \begin{bmatrix} k_{0,c} \\ k_{1,c} \\ k_{2,c} \\ k_{3,c} \end{bmatrix}$$

Jeżeli wprowadzone zostaną cztery tablice, każda wielkości 256 32-bitowych słów ($0 \leq x < 256$) jak poniżej:

$$T_0[x] = \begin{bmatrix} 02 S[x] \\ S[x] \\ S[x] \\ 03 S[x] \end{bmatrix} \quad T_1[x] = \begin{bmatrix} 03 S[x] \\ 02 S[x] \\ S[x] \\ S[x] \end{bmatrix} \quad T_2[x] = \begin{bmatrix} S[x] \\ 03 S[x] \\ 02 S[x] \\ S[x] \end{bmatrix} \quad T_3[x] = \begin{bmatrix} S[x] \\ S[x] \\ 03 S[x] \\ 02 S[x] \end{bmatrix}$$

Wówczas równanie wektorowe można wyrazić w następującej formie:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = T_0[S_{0,c(0)}] \oplus T_1[S_{1,c(1)}] \oplus T_2[S_{2,c(2)}] \oplus T_3[S_{3,c(3)}] \oplus k_{\text{round},c}$$

Każda kolumna macierzy stanów może być obliczona przy użyciu czterech instrukcji XOR

Powyższe równanie odnosi się do wszystkich cykli, z wyjątkiem ostatniego, ponieważ ostatni cykl pozbawiony jest transformacji MixColumn. Brak tej transformacji oznacza konieczność użycia innych tablic dla ostatniego cyklu które są postaci jak poniżej:

$$U_0[x] = \begin{bmatrix} S[x] \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad U_1[x] = \begin{bmatrix} 0 \\ S[x] \\ 0 \\ 0 \end{bmatrix} \quad U_2[x] = \begin{bmatrix} 0 \\ 0 \\ S[x] \\ 0 \end{bmatrix} \quad U_3[x] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ S[x] \end{bmatrix}$$

Stosowanie tablic, wymaga dodatkowych 4 Kb pamięci. Wartość ta ulega podwojeniu w przypadku zastosowania również tablic dla ostatniego cyklu. Wielkość ta może jednak ulec redukcji kosztem zwiększenia ilości obliczeń związanych z cykliczną rotacją poszczególnych elementów tablicy. W identyczny sposób można zaimplementować algorytm deszyfrowania. Należy jednak wziąć w tym przypadku pod uwagę fakt, że tablice będą miały inną zawartość.

Opisane powyżej rozwiązanie pozwala na przetwarzanie 32-bitowych bloków, co znacznie wpływa na szybkość działania szyfru.

5.6.3 Wydajność

W trakcie testów mających na celu pomiar prędkości działania szyfru, uzyskano następujące rezultaty:

1. Dla Pentium Pro 200MHz, 64MB RAM i kompilatorze GCC 2.8.1 pod Linuxem, przy wielkości bloku $n=128$ bitów otrzymano następujące wyniki³³:

	szyfrowanie Kbit/s	deszyfrowanie Kbit/s
Rijndael 128	42602	47754
Rijndael 192	36175	35562
Rijndael 256	31551	30969

2. Dla Sun 2x 360MHz UltraSparc -II 4MB Cache, 256MB RAM i Sun Workshop Compiler4.2: otrzymano:³⁴

	szyfrowanie Kbit/s	deszyfrowanie Kbit/s
Rijndael 128	59522,7	61260
Rijndael 192	50866	52454
Rijndael 256	44409	45612

3. Przy implementacjach opartych na FPGA Xilinx Virtex XCV1000BG560-54, zoptymalizowanych pod względem szybkości, w trybie bez sprzężenia zwrotnego Rijndael osiągnął prędkość szyfrowania 1,939 GBit/sec, zaś w trybach ze sprzężeniem: 300 MBit/sec

35

³³ Na podst. Lawrence E. Bassham III, Efficiency Testing of ANSI C Implementations of Round 2 Candidate Algorithms for the Advanced Encryption Standard, AES3 Proceedings April 2001

³⁴ .Na podst: jak wyżej

³⁵ Na podstawie: J Elbirt1, W Yip, B Chetwynd, C Paar, An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists, AES3 Proceedings, April 2000

4. Dla 8 bitowego procesora MC6805, czas szyfrowania jednego bloku długości 128 bitów , przy długości klucza 128 bitów, wyniósł 9464 cykli, zaś deszyfrowania 13538 cykli. Dla porównania: RC6: 37731 cykli, DES 17458 cykli³⁶

5.7 Przesłanki projektowe

1. Wielomian redukcyjny $m(x)$

Wielomian $m(x) = 11B_x$ używany do mnożenia w $GF(2^8)$ został wybrany ponieważ jest on pierwszym na liście wielomianów pierwszych ósmego stopnia.³⁷

2. Kryteria projektowe S-box

Projektując S-Box J. Daemen oraz V. Rijmen, mieli na uwadze dwa aspekty - odporność na LC oraz DC, ataki oparte na algebrze takie jak: ataki interpolacyjne, oraz aby S-box spełniał następujące kryteria:

- a. odwracalności,
- b. minimalnej wartości największego współczynnika korelacji pomiędzy liniową kombinacją bitów wejściowych oraz liniową kombinacją bitów wyjściowych
- c. prostoty opisu.

Spośród licznych transformacji matematycznych mogących stanowić S-box. wybrane zostało odwzorowanie polegające na przypisaniu określonej wartości wejściowej, jej multiplikatywnej odwrotności w skończonym ciele $GF(2^8)$. Ponieważ odwzorowanie to może być podatne na atak oparty na interpolacjach algebraicznych, jest ono zmodyfikowane poprzez dodatkowe połączenie z afiniczną i odwracalną transformacją algebraiczną - modułowym mnożeniem w ciele $GF(2^8)$. Opis algebraiczny S-boxa przedstawia się zatem następująco:

$$b(x) = (x^7 + x^6 + x^2 + x) + a(x)(x^7 + x^6 + x^5 + x^4 + 1) \bmod x^8 + 1$$

Wielomian $x^7 + x^6 + x^5 + x^4 + 1$ został wybrany tak by być wzajemnie pierwszym wraz z wielomianem $x^8 + 1$.

S-box ten może zostać zamieniony na dowolny inny spełniający kryteria odwracalności. Ponieważ struktura Rijndaela oraz odpowiednia liczba cykli zapewniają odpowiednią odporność na atak za pomocą kryptoanalizy różnicowej lub liniowej, zamienny S-box nie musi spełniać postawionych wymagań dotyczących współczynników korelacji.

3. Transformacja MixColumn.

³⁶ Na podst. G. Keating, Performance Analysis of AES candidates on the 6805 CPU core, AES2 Submission Proceedings 1999

³⁷ Wg. R. Lidl, H Niederreiter, Introduction to finite fields and their applications, Cambridge University Press, 1986, cyt. [w]: J. Daemen, V. Rijmen, AES Proposal: Rijndael,, Document version 2, date: 03/09/1999

Transformacja ta należy do grupy transformacji liniowych spełniających kryteria:

- a. odwracalności,
- b. liniowości w $GF(2)$
- c. dobrych własności dyfuzyjnych
- d. szybkości na procesorach ośmiobitowych,
- e. symetrii oraz regularności
- f. prostoty opisu.

Spełnienie kryteriów b,e,f przyczyniło się do wyboru transformacji opierającej się na mnożeniu wielomianów modulo x^4+1 . Odwracalność, dobre własności dyfuzyjne oraz szybkość pociągnęły za sobą konieczność aby mnożone wielomiany były wielomianami ze współczynnikami.

Najważniejszym spośród tych sześciu kryteriów jest wymóg dobrych własności dyfuzyjnych, czyli aby różnica jednego bitu wejściowego wpływała na jak najwięcej bitów wyjściowych.

4. Transformacja ShiftRows.

Dokonując przeglądu transformacji oraz ich implementacji używanych w szyfrach blokowych³⁸, V. Rijmen, zaleca, aby każdą formę permutacji, którą jest niewątpliwie transformacja ShiftRows charakteryzowała pewna symetria oraz regularność. Regularność pozwala na łatwą implementację zarówno sprzętową jak i programową nie tylko z punktu widzenia zapotrzebowania na pamięć, ale i z punktu widzenia algorytmicznego. Permutacje w których każdy bit rozmieszczony jest indywidualnie, pociągają za sobą konieczność stosowania tablic, co jest rozwiązaniem nieefektywnym, dlatego też ShiftRows używa stałych wektorów przesunięć poszczególnych bitów, odrębnych dla każdego wiersza macierzy stanów.

5. Algorytm rozszerzenia klucza.

Projektując algorytm rozszerzenia klucza, kierowano się następującymi kryteriami:

- a. odpornością na atak w którym część klucza głównego jest znana kryptoanalitykowi,
- b. odpornością na atak metodą kluczy powiązanych³⁹

Algorytm ten, gra również dużą rolę w eliminacji symetrii cyklu, traktującej w jednakowy sposób wszystkie bajty macierzy stanów. Wprowadzając stałe w algorytmie rozszerzenie klucza, wpływ tej symetrii zanika. Zanika również symetria

³⁸ patrz: J.Daemen, „Cryptoanalysis and design of iterated block ciphers”, Doctorial Dissertation, October 1997, K.U. Leuven

³⁹ E. Biham - „New types of cryptanalytic attacks using related keys”, Advances in cryptology, Proceedings Eurocrypt '93, LNCS 765, T. Helleseth, Ed., Springer-Verlag, 1993, pp. 398-409 cyt [w]: p.powyżej

zespołu cykli (transformacje są identyczne dla każdego cyklu) poprzez wprowadzenie stałych zależnych od poszczególnych cykli.

Kolejnymi przesłankami projektu algorytmu rozszerzenia klucza były:

- a. odwracalność,
- b. szybkość pracy na wielu różnych rodzinach mikroprocesorów,
- c. jak największa wartość współczynnika dyfuzji różnic klucza głównego na klucze cyklu,
- d. znajomość części klucza głównego lub pewnych kluczy cyklu nie może w żadnym stopniu przyczynić się do znalezienia pozostałych kluczy cyklu
- e. zapewnić wysoką nieliniowość, aby uniemożliwić wyznaczenie różnic w kluczach cyklu na podstawie różnic w kluczu głównym.
- f. prostotą opisu.

Nieliniowość zapewnia operacja SubByte, której opis znajduje się w pozostałej części rozdziału.

6. Liczba cykli.

Liczba cykli została wyznaczona jako suma dwóch wartości: największej liczby cykli dla których możliwy byłby atak bardziej efektywny niż wyczerpujące szukanie klucza, oraz marginesu bezpieczeństwa.

Dla Rijndaela, pracującego z długością bloku równą 128 bitów oraz 128 bitowym kluczem, przeprowadzenie ataku efektywniejszego niż wyczerpujące poszukiwanie klucza, było możliwe po maksymalnie sześciu cyklach. Wartość marginesu bezpieczeństwa określono jako 4 cykle ponieważ: dwa cykle zapewniają „pełną dyfuzję” w następującym sensie: zmiana wartości jednego z bitów macierzy stanów, wpływa na zmianę połowy bitów macierzy stanów po dwóch cyklach. Dodanie czterech cykli jest więc krokiem zapewniającym pełną dyfuzję na początku i na końcu szyfru.

Bardzo dobre własności dyfuzyjne Rijndaela wynikają z jego struktury, której projekt oparty jest na pracy V. Rijmena. W swojej rozprawie ⁴⁰ twierdzi on, że budowa szyfru oparta na strukturze Feistela jest niepraktyczna gdyż w strukturze takiej, tylko połowa bajtów z wewnętrznej macierzy stanów podlega transformacjom. Fakt ten był wykorzystywany w wielu rodzajach ataków. Zamiast struktury Feistela, V. Rijmen proponuje cykl złożony z kilku prostych zróżnicowanych transformacji

⁴⁰ J.Daemen, „Cryptoanalysis and design if iterated blok ciphers”, Doctorial Dissertation, October 1997, K.U. Leuven

charakteryzujących się odwracalnością oraz symetrią, operujących na całej macierzy stanów. Dalsze rozważania na temat struktury szyfru podyktowane są koncepcją *wide trail strategy*, opisaną w poprzedniej części pracy.

Dla wersji Rijndaela pracujących z dłuższymi kluczami, liczba cykli jest zwiększana o jeden dla każdego dodatkowych 32 bitów klucza głównego. Przyczyną tego jest zwiększenie odporności na ataki o stopniu złożoności mniejszym niż wyczerpujące przeszukiwanie klucza, jakimi są ataki metodą kluczy powiązanych, oraz ataki na algorytm rozszerzenia klucza.

Liczba cykli zwiększana o jeden jest również dla każdego dodatkowych 32 bitów długości bloku.. Przyczyną jest konieczność zapewnienie pełnej dyfuzji - wzrost długości bloku zwiększa liczbę cykli po których zachodzi pełna dyfuzja.

5.8 Wide Trail Strategy

5.8.1 Korelacja i jej wpływ na kryptoanalizę liniową

Fundamentem działania kryptoanalizy liniowej jest korelacja między liniową kombinacją bitów wejściowych oraz liniową kombinacją bitów wyjściowych, w poszczególnym stanie szyfru.

Dla dwóch liniowych funkcji boolowskich $f(a)$ oraz $g(a)$, wartość współczynnika korelacji można zdefiniować jako:

$$C(f,g) = 2\text{Prob}(f(a) = g(a)) - 1$$

Współczynnik ten może przybierać wartości z przedziału $(-1,1)$. Jeżeli jego wartość różna jest od zera, wówczas funkcje są ze sobą skorelowane.

Wektorem selekcji w nazywamy binarny wektor selekcyjny wszystkie i -te komponenty wektora dla których $w_i=1$.

Liniowa kombinację wektora w oraz wektora a określona jest jako $w^t a$. Kombinacja ta określa więc indeksy, dla których wektor a przyjmuje wartość 1. Górny indeks t oznacza transpozycję. Prościej mówiąc wektor selekcji w wskazują jedną z możliwych wartości wektora a . Każdą funkcję boolowską można wyrazić za pomocą wektora selekcji w .

Każdej funkcji boolowskiej $f(a)$ można również przyporządkować funkcję $\hat{f}(a)$ przyjmującą wartości rzeczywiste, taką że: $\hat{f}(a)=-1$ gdy $f(a)=1$ oraz $\hat{f}(a)=1$ gdy $f(a)=0$. Wówczas wartości współczynników korelacji $C(f(a), w^t a)$, dla wszystkich możliwych a można wyrazić w postaci:

$$F^{\wedge}(w) = \sum_a f^{\wedge}(a)(-1)^{w^t a}$$

Postać ta jest określana jako transformata Walsh-Hadamarda, ozn: $F^{\wedge}(w) = W(f(a))$. Transformacja Walsh-Hadamarda, stanowi najdogodniejszą postać wyrażania korelacji pomiędzy poszczególnymi transformacjami cyklu.

Każdy element szyfru, włączając S-boxy, permutacje można określić jako odwzorowanie przestrzeni binarnej n -wymiarowej w przestrzeń binarną m -wymiarową. Często $m = n$. Odwzorowanie $h: \{0,1\}^n \rightarrow \{0,1\}^m$ może zostać rozłożone na m funkcji boolowskich (h_0, h_1, \dots, h_m). Każda z tych funkcji składowych h_i ma swoją własną transformatę Walsh-Hadamarda H_i^{\wedge} . Wektor złożony z składników H_i^{\wedge} oznaczany jest jako H^{\wedge} i może on być traktowany jako transformata Walsh-Hadamarda przekształcenia h . Każde z takich odwzorowań, może zostać przedstawione za pomocą macierzy korelacji. Współczynniki korelacji pomiędzy liniowymi kombinacjami bitów wejściowych oraz bitów wyjściowych przekształcenia h stanowią macierz korelacji C^h o wymiarach $2^m \times 2^n$. Element C_{uw}^h w wierszu u oraz kolumnie w jest równy współczynnikowi korelacji $C(u^t h(a), w^t a)$. Wartości elementów macierzy korelacji są krotnościami 2^{1-n} .

Macierz korelacji przekształcenia $h = h_1 \times h_2$ jest równa iloczynowi macierzy przekształceń składowych:

$$C^{h_1 \times h_2} = C^{h_1} * C^{h_2}$$

Każde odwzorowanie boolowskie, którego macierz korelacji C^h jest macierzą ortogonalną, jest przekształceniem odwracalnym.

Rozważając transformację składającą się z bitowej sumy wektora k , widać, że:

$$h(a) = a + k$$

Ponieważ wartość $u^t h(a)$ jest równa sumie ($u^t a + u^t k$), macierz korelacji jest macierzą diagonalną, taką że $C_{uu} = (-1)^{u^t k}$. Efekt sumy bitowej wektora k przed lub po transformacji h , powoduje więc pomnożenie niektórych wierszy lub kolumn macierzy korelacji przez (-1) . Analogicznie, rozważając transformację składającą się z równoległego zastosowania l odwzorowań (S-boxów) ze zbioru $\{0,1\}^n$ w zbiór $\{0,1\}^m$ z założeniem, że: $\sum_i n_i = n$ oraz $\sum_i m_i = m$ otrzymujemy następujące odwzorowanie: $b_{(i)} = h_{(i)}(a_{(i)})$ dla $0 \leq i \leq l$. Jeśli każdy S-box $h_{(i)}$ ma swą własną macierz korelacji o wymiarach $2^{m_i} \times 2^{n_i}$ oznaczaną $C^{(i)}$, wówczas korelacja odnosząca się do selekcji wejściowej w oraz selekcji wyjściowej u stanowi iloczyn korelacji we-wy poszczególnych S-boxów.

Opisane powyżej zjawiska, można zastosować w odniesieniu do transformacji szyfrujących formułujących cykl pracy szyfru. Transformacje te są postaci:

$$\beta = \rho_m \dots \rho_2 \rho_1$$

W dziedzinie transformacji Walsh-Hadamarda, stały cykl transformacji odnosi się do macierzy korelacji o wymiarach $2^n \times 2^n$, będącej iloczynem macierzy korelacji poszczególnych transformacji cyklu:

$$C^\beta = C^{\rho_m} \dots C^2 C^1$$

Kryptoanaliza liniowa wykorzystuje występowanie dużych wartości elementów w powyższych iloczynach. Ślad liniowy m-tego cyklu ma postać:

$$\Xi = (\xi_0 \leftarrow \xi_1 \leftarrow \xi_2 \leftarrow \dots \xi_{m-1} \leftarrow \xi_m)$$

i zawiera łańcuch korelacji postaci $C(\xi_i \uparrow \rho_i(a), \xi_{i-1} \uparrow a)$.

Do każdego takiego liniowego śladu można odnieść wartość współczynnika wpływu korelacji danego śladu, określanego jako:

$$C_p(\Xi) = \prod_i C_{\xi_i \xi_{i-1}}^{\rho_i}$$

Łączna wartość współczynnika korelacji wartości wyjściowej cyklu z wartością wejściową można określić jako sumę współczynników wpływu korelacji wszystkich śladów liniowych tworzonych przez poszczególne kombinacje danych wejściowych.

$$C(u \uparrow \beta(a), w \uparrow a) = \sum_{\xi_0=w, \xi_m=u} C_p(\Xi)$$

Istotnym jest wprowadzenie współczynnika wagowego korelacji śladu liniowego, określanego jako $w_c = -\log_2 |C_p(\Xi)|$, będącego sumą współczynników wagowych poszczególnych kroków (korelacji) tworzących dany ślad.

Kolejność transformacji cyklu często jest wyznaczana przez klucz, lub inną wartość zależną od pewnej wartości wejściowej. Ogólnie można powiedzieć, że wartości elementów macierzy korelacji transformacji ρ_i zależą od wartości klucza cyklu $K^{(i)}$. Dla niektórych szyfrów nawet, silna zależność własności korelacyjnych i propagacyjnych stawiana jest jako jedno z głównych kryteriów projektowych. W takim przypadku analiza korelacji oraz propagacji różnic musi być powtarzana dla każdej wartości klucza, czyniąc liniową lub różnicową analizę niemożliwą. Typowym problemem pojawiającym się w takim rozwiązaniu jest jednak fakt, że mimo iż odporność na LC i DC może być dobra w odniesieniu do wartości średniej, pewne klasy kluczy mogą spowodować wystąpienie śladów liniowych lub różnicowych z nadmiernymi wartościami korelacji. Problemu takiego można

uniknąć projektując transformacje cyklu tak, aby amplitudy elementów macierzy korelacji były niezależne od wartości klucza. Na podstawie rozważań dot. macierzy korelacji można stwierdzić, że jest to tylko możliwe, gdy cykl zawiera stałą transformację ρ poprzedzoną lub zakończoną bitowym sumowaniem wartości klucza i macierzy stanów. W takim przypadku amplitudy współczynników wpływu korelacji są niezależne od wartości klucza. Jednakże od wartości klucza zależą ich znaki⁴¹.

W kryptoanalizie liniowej wprowadzono pojęcie odchylenia od liniowości aproksymacji liniowej, jako wartości będącej odchyleniem od wartości prawdopodobieństwa równej 0,5. Im większa wartość tego odchylenia, tym większa podatność na kryptoanalizę. Analizując tablice aproksymacji liniowych, można jednoznacznie stwierdzić, że odchylenie od liniowości jest w wysokim stopniu związane z korelacją. Jest zatem wysoce pożądane, aby wartości korelacji były jak najmniejsze.

Przyglądając się własnościom korelacyjnym cyklu należy rozpatrzyć dwa aspekty. Pierwszy aspekt dotyczy liniowych śladów. Dla szyfrów blokowych należy wziąć pod uwagę że wartość maksymalna współczynnika wpływu korelacji C_p ma wpływ nie na jeden, ale na wiele cykli. Efektywny cykl to taki cykl, w który łączy mały nakład pracy z C_p w taki sposób, że wraz ze wzrostem ilości cykli, wartość C_p gwałtownie maleje. Drugi aspekt dotyczy faktu w jaki sposób liniowe ślady mają wpływ na korelacje wielu cykli. Interferencja wielu liniowych śladów z małymi wartościami współczynników wpływu korelacji, po wykonaniu wielu cykli, może objawić się silną korelacją. Dla dobrze zaprojektowanych cykli, współczynniki wpływu korelacji odnoszące się do liczby cykli (wg. J.Daemena oraz V. Rimena, liczba ta wynosi 2 lub 3), nie mogą być większe niż $2^{-n/2}$.

5.8.2 Propagacja różnic i jej wpływ na kryptoanalizę różnicową

Niech a oraz a^* będą n -wymiarowymi wektorami, z różnicą bitową $a' = a \oplus a^*$. Wówczas $b=h(a)$, $b^*=h(a^*)$, oraz $b + b^* = b'$. Różnica a' wpływa zatem na różnicę b' poprzez transformację h , ozn: $(a' \rightarrow b')$. W rzeczywistości różnica b' nie jest wyznaczana przez a' , ale zależy od wartości a lub a^* .

Współczynnik R_p propagacji różnicowej $(a' \rightarrow b')$ można wyrazić jako:

$$R_p(a' \rightarrow b') = 2^{-n} \sum_a \delta(b' + h(a + a') + h(a))$$

gdzie: $\delta(p)=1$ gdy $p=0$ oraz $\delta(p)=0$ w przeciwnym razie.

⁴¹ Dowód matematyczny znajduje się w J.Daemen, „Cryptoanalysis and design if iterated blok ciphers”, Doctorial Dissertation, October 1997, K.U. Leuven

Wartości R_p należą do przedziału $(0,1)$ i są liczbami całkowitymi będącymi krotnościami 2^{n-1} . Współczynnik propagacji różnicowej określa jaki ułamek z wszystkich kombinacji wejściowych mających różnicę równą a' , daje na wyjściu różnicę równą b' . Jeśli $R_p=0$, wówczas propagacja różnic nie zachodzi, czyli różnica a' nie ma wpływu na różnicę b' poprzez transformację h . Można łatwo stwierdzić, że

$$\sum_{b'} R_p(a' \rightarrow b') = 1$$

Ograniczeniem wagowym propagacji różnicowej jest $w_r = -\log_2 R_p(a' \rightarrow b')$. Wartość ograniczenia wagowego, określa ilość informacji (w bitach), zawartej na wyjściu transformacji h , o wartości a . Jeśli h stanowi grupę równoległych l odwzorowań (S-boxów), wówczas wartość współczynnika propagacji R_p transformacji h , stanowi iloczyn współczynników propagacji $R_{p(i)}$ poszczególnych S-boxów S_i , zaś wartość ograniczenia wagowego w_r transformacji, sumę ograniczeń wagowych poszczególnych S-boxów.

Z własności funkcji logarytmicznej można zauważyć, że małe wartości współczynnika propagacji dają dużą wartość ograniczenia wagowego propagacji i odwrotnie. Im większa będzie wartość współczynnika propagacji, tym łatwiej będzie więc przewidzieć różnicę bitów wyjściowych przy danej różnicy bitów wejściowych.

Odnosząc powyższe stwierdzenia do całego cyklu pracy szyfru ślad różnicowy m -tego cyklu jest określony jako:

$$\Omega = (\omega_0 \rightarrow \omega_1 \rightarrow \omega_2 \dots \omega_{m-1} \rightarrow \omega_m)$$

Ślad ten zawiera łańcuch propagacji różnicowych postaci $(\omega_{i-1} \rightarrow \omega_i)$. Nazywane są one różnicowymi krokami śladu. Współczynnik propagacji śladu określony jako $R_p(\Omega)$ stanowi część ze zbioru wszystkich wartości a_0 , powodującą określony ślad różnicowy. Ograniczeniem wagowym śladu różnicowego cyklu jest suma ograniczeń wagowych poszczególnych transformacji wchodzących w jego skład.

$$w_r(\Omega) = \sum_i w_r(\omega_{i-1} \xrightarrow{p_i} \omega_i)$$

Wartość współczynnika propagacji R_p , można aproksymować za pomocą wag jako:

$$R_p(\Omega) \approx 2^{-w_r(\Omega)}$$

przy założeniu, że odpowiednie ograniczenia wagowe nie są ze sobą skorelowane.

W praktyce, w przypadku DES, charakterystyka może mieć wysokie prawdopodobieństwo, jeśli współczynnik propagacji ma bardzo dużą wartość, co implikuje bardzo małą wartość ograniczenia wagowego, znacznie mniejszą niż n . Jeśli wartość $w_r(\Omega)$ jest rzędu n lub większe, wartość współczynnika propagacji jest bardzo mała, tym samym nie

jest możliwe uzyskanie odpowiedniej dla DC charakterystyki szyfru. Aby zapewnić odporność szyfru na kryptoanalizę różnicową łączna wartość współczynnika propagacji kilku cykli szyfru, nie może więc przekraczać 2^{1-n} . Dobrze więc zaprojektowane transformacje cyklu, powodują, że przy małym nakładzie obliczeniowym, wraz ze wzrostem ilości cykli, gwałtownie zmniejsza się współczynnik propagacji różnicowej.

5.8.3 Definicja Wide Trail Strategy

W tej sekcji przedstawiona zostanie strategia projektowania transformacji cyklu, w taki sposób, aby uniknąć wielocyklowych liniowych i różnicowych śladów których współczynniki propagacji i korelacji charakteryzują się dużymi wartościami.

Dla obu typów śladów, waga jest dana jako suma wag ich poszczególnych kroków. Niech zatem cykl zawiera trzy transformacje:

- odwracalną transformację nieliniową γ ,
- odwracalną transformację liniową θ ,
- transformację której działanie polega na sumowaniu wartości macierzy stanów z wartościami klucza.

Jeśli γ jest transformacją złożoną z S-boxów, wówczas współczynnik korelacji $C_u^{\gamma_w}$ jest iloczynem współczynników korelacji we-wy każdego z S-boxów. Waga korelacji liniowego kroku jest równa sumie wag korelacji odnoszących się do we-wy. poszczególnych S-boxów. Podobnie, ograniczenie wagowe propagacji różnicowej jest sumą ograniczeń wagowych propagacji różnicowych poszczególnych S-boxów.

S-box jest w danym cyklu aktywny względem liniowego śladu, jeśli, dla tego liniowego śladu, wektor selekcji jego wartości wyjściowej jest różny od zera. Analogicznie S-box jest aktywny względem różnicowego śladu, gdy wektor selekcji jego wartości wejściowej jest różny od zera, dla określonego śladu różnicowego.

Zarówno dla liniowego jak i różnicowego śladu, wagę śladu można postrzegać jako liczbę aktywnych S-boxów, co sugeruje dwie możliwości eliminacji śladów z małymi wagami:

- wybór S-boxów z propagacjami różnicowymi mającymi dużą wagę oraz dużą wagę korelacji we-wyj.
- taki projekt cyklu, aby występowały tylko ślady pochodzące z wielu S-boxów.

Wide trail strategy kładzie nacisk na drugi z tych mechanizmów. Cykl musi zostać zaprojektowany w taki sposób, aby liniowe (różnicowe) kroki z małą liczbą aktywnych S-

boxów występowały przed liniowymi (różnicowymi) krokami z dużą liczbą aktywnych S-boxów. Koncepcja ta, jest zbliżona do koncepcji dyfuzji wprowadzonej przez Shannona⁴² i odnosi się do rozprzestrzeniania dużej ilości informacji. Jedynym wymogiem jakim pociąga za sobą *wide trail strategy* jest to, że każdy z S-boxów musi mieć pewną minimalną wagę korelacji, oraz pewną minimalną wagę różnicową.

Wide trail strategy nie ogranicza nieliniowego kroku do transformacji złożonej z wielu S-boxów. Może to być dowolna transformacja stała względem przesunięcia⁴³.

Koncepcja *wide trail strategy* stanowi duży kontrast w stosunku do strategii przyjętych przez większość projektantów szyfrów. Tradycyjna koncepcja projektowa jest oparta o strukturę Feistela i koncentruje się na S-boxach, co objawia się w przypadku DES małą szerokością liniowych i różnicowych śladów. Najszerszy ślad różnicowy w DES, w dwóch cyklach, obejmuje tylko trzy S-boxy, zaś liniowy - 3 S-boxy - w czterech cyklach.

Zazwyczaj S-boxy są ulokowane w funkcji F, struktury Feistela lub stanowią część struktury podstawieniowo-permutacyjnej (substitution permutation network), zaproponowanej również przez Feistela. S-boxy te muszą zostać wówczas tak zaprojektowane, aby wyeliminować nisko wagowe ślady. W praktyce oznacza to takie kryteria projektowe dla S-boxów, jak: najmniejsza maksymalna korelacja wejścia z wyjściem i najmniejszy maksymalny współczynnik propagacji oraz dobre własności dyfuzyjne. Wraz ze wzrostem wielkości S-boxów, coraz łatwiej spełnić każde z powyższych założeń, co skłania wielu projektantów do wniosku, że najlepsze zabezpieczenie przeciw DC oraz LC stanowią S-boxy dużych rozmiarów. Ten jednostronny jednak punkt widzenia całkowicie pomija potencjalnie wysoką dyfuzję, jaką może zapewnić dobrze zaprojektowany przebieg cyklu.

Dla Rijndaela S-box ma współczynnik propagacji 2^{-6} , zaś współczynnik korelacji 2^{-3} .

5.9 Odporność i bezpieczeństwo szyfru

Najlepszym rodzajem ataku mającego na celu wyznaczenie bitów klucza w przypadku Rijndaela jest atak polegający na wyczerpującym poszukiwaniu klucza. Dla klucza o długości n bitów, złożoność takiego ataku wynosi 2^n .

Dzięki zastosowaniu nieliniowych transformacji w algorytmie rozszerzenia klucza, wyeliminowana została całkowicie możliwość wystąpienia słabych kluczy.

⁴²Zob. C.E.Shannon, Communication Theory of Secret Systems, Bell System Technical Journal v. 28, n. 4, 1949

⁴³Opis transformacji, kryteria, omówione zostały w J.Daemen, „Cryptanalysis and design of iterated blok ciphers”, Doctorial Dissertation, October 1997, K.U. Leuven

Rijndael, bazujący na koncepcji *wide trail strategy* odporny jest na działanie kryptoanalizy różnicowej. Na wartość propagacji różnicowej mają wpływ różnicowe ślady, której współczynnik propagacji jest sumą współczynników propagacji wszystkich śladów różnicowych mających specyficzną różnicę na początku i końcu cyklu. Aby zapewnić odporność przeciw DC, musi być spełniony warunek, że nie ma śladu różnicowego którego współczynnik propagacji ma wartość większą niż 2^{1-n} . W przypadku Rijndaela nie istnieje ślad, który po czterech cyklach szyfru ma wartość większą niż 2^{-150} , co jest wystarczającą wartością dla wszystkich długości bloku.

Niektóre techniki kryptoanalizy różnicowej wykorzystują inną miarę różnicy danych niż XOR (przykładem może być DC szyfru RC6). Dla Rijndaela jednak, XOR jest najlepszą miarą zróżnicowania danych, co pozwala stwierdzić, że nie istnieje dla tego szyfru możliwość wykorzystania lepszej techniki DC.

Dzięki koncepcji *wide trail strategy*, Rijndael jest także odporny na LC. Analizując istotę *wide trail strategy*, można zauważyć, że możliwość LC jest wówczas, kiedy wartości współczynników korelacji we-wy dla wszystkich cykli szyfru, przekraczają $2^{n/2}$. Znak wartości współczynników korelacji w Rijndaelu zależy od wartości klucza cyklu. W przypadku Rijndaela nie ma śladów liniowych, dla których wartości współczynników są większe niż 2^{-75} , po czterech cyklach szyfru. Wraz ze wzrostem ilości cykli, dzięki dobrym własnościom propagacyjnym, zapewnianym w dużej mierze przez transformację MixColumn, liczba ta ulega dalszemu zmniejszeniu (po czterech cyklach minimalna liczba aktywnych S-boxów wynosi 25).

Autorzy szyfru twierdzą, że najlepszym atakiem na Rijndaela jest *square attack*. Atak ten, należący do ataków z wybranym tekstem jawnym, wykorzystujący zorientowaną bajtowo strukturę szyfru, został dokonany na szyfrze *square*, szyfrze będącym poprzednikiem Rijndaela. Ponieważ Rijndael ma wiele cech wspólnych ze *square*, atak ten jest również możliwy. Atak ten jest jednak szybszy od wyczerpującego poszukiwania klucza tylko dla sześciu cykli Rijndaela (do jego dokonania potrzebne jest wówczas 2^{32} tekstów jawnych, zaś liczba procesów szyfrowania wynosi $2^{40} - 2^{72}$), co czyni jego bezużytecznym.

Według projektantów inne rodzaje kryptoanalizy i ataków, opierających się na interpolacjach lub powiązanych kluczach są również niemożliwe. Stwierdzenie to, przyczyniło się w znacznej mierze do dokładniejszej analizy szyfru, dokonanej przez B.Schneier, D.Wagner⁴⁴ Wyniki jej przedstawiają się następująco: poprzez zmodyfikowanie i

⁴⁴ Zob. B.Schneier, N.Ferguson, R.Wagner, Improved Cryptanalysis of Rijndael,

ulepszenie *square attack* możliwy jest atak na sześć cykli Rijndaela o stopniu złożoności 2^{44} , dla siedmiu cykli złożoność wynosi 2^{155} dla 192-bitowego klucza oraz 2^{172} dla 256 bitowego klucza. Analiza ta może więc pokazywać ewentualne drogi dalszych ataków na AES. Kolejnym wnioskiem wypływającym z powyższej analizy, jest fakt, że procedura rozszerzenia klucza Rijndaela charakteryzuje się znacznie gorszymi parametrami dyfuzyjnymi niż sam szyfr. W/w autorzy, pokazali, że istnieje możliwość ataku na 9 cykli Rijndaela z 256-bitowym kluczem, metodą kluczy powiązanych, przy użyciu 2^{77} tekstów jawnych oraz 256 kluczy. Atak ten charakteryzuje się złożonością 2^{224} , jest on więc wysoce niepraktyczny, aczkolwiek teoretycznie możliwy.

Wart odnotowania jest rodzaj ataku przeprowadzony przez H. Gilbert i M.Minier.⁴⁵ Za pomocą nieco innego spojrzenia na transformacje cyklu, oraz wprowadzenia odpowiadających im zależności, w/w udowodnili, że niezależnie od długości klucza, istnieje możliwość ataku na siedem cykli Rijndaela, o złożoności 2^{140} , używając 2^{32} tekstów jawnych. Atak ten jednak uznany został za niepraktyczny, gdyż jego dokonanie dla 10 cykli Rijndaela jest niemożliwe z punktu widzenia teoretycznego.

Implementacje Rijndaela, najczęściej oparte są na odczycie z tablic oraz na przesunięciach. V.Rijmen, dokonując analizy odporności szyfrów pod względem ataków opartych na implementacjach - SPA, DPA, pomiar czasu pokazał⁴⁶, że Rijndael, właściwie zaimplementowany, jest odporny na tego rodzaju ataki. W dalszej części pracy przedstawiony został jednak sposób ataku bazującego na pomiarze czasu na Rijndaela, na niewłaściwie dokonanej implementacji.

⁴⁵ H.Gilbert, M.Minier, A Collision Attack on 7 Rounds of Rijndael, AES3 Proceedings related Papers

⁴⁶ V.Rijmen, J.Daemen, Resistance Against Implementation Attacks, A Comparative Study of The AES Proposals, February 1999

Rozdział 6

Systemy szyfrowania z kluczem publicznym

6.1 Wstęp

W systemach z kluczem symetrycznym, ktoś, kto wie jak zaszyfrować wiadomość, wie również jak ją odszyfrować. Do obydwu tych operacji używany jest jeden klucz. Istotną wadą takiego rozwiązania jest jednak fakt, że para użytkowników, która chce porozumieć się w sposób poufny, musi również w sposób poufny wymienić klucz.

Oblicze kryptografii radykalnie się zmieniło, kiedy w 1976 roku Diffie i Hellman przedstawili nowy algorytm szyfrowania danych oparty na kluczach publicznych. Odkrycie to oznaczało, że każdy mógł zaszyfrować i przesłać wiadomość do danego użytkownika, szyfrując ją przy za pośrednictwem dostępnego klucza publicznego, jednakże tylko właściwy adresat wiadomości mógł ją odszyfrować przy użyciu swojego tajnego klucza prywatnego. Nie ma żadnej potrzeby kontaktu pomiędzy nadawcą a odbiorcą w celu wymiany klucza.

Wprowadzenie kryptografii z kluczem publicznym w drastyczny sposób powiększyło rolę algebry i teorii liczb w kryptografii. Zastanawiający jest jednak fakt, dlaczego kryptografia z kluczem publicznym została wynaleziona tak późno, mimo iż opiera się ona na matematyce znanej już Eulerowi w XVIIIw. i nic co potrzebne było do jej wynalezienia i skonstruowania kryptosystemów z kluczami publicznymi, nie wymagało współczesnej matematyki. Według N. Koblitz, wielkiego matematyka i autorytetu w dziedzinie kryptografii, przypuszczalnym powodem może być to, że do lat 70. kryptografii używano jedynie w celach militarnych i dyplomatycznych, w celach, w których systemy z kluczem symetrycznym idealnie spełniały swoją rolę. Ekspansja komputerów w sfery życia codziennego spowodowała pojawienie się sytuacji i przypadków, w których spotykamy się z szeroką i płynną grupą użytkowników, sytuacji w których stosowanie systemów z kluczem symetrycznym było by kłopotliwe oraz mało efektywne. Kryptografia oparta o klucz publiczny powstała więc tak późno, gdyż dopiero wtedy zaistniała konieczność jej wynalezienia.⁴⁷

Najczęstszymi przypadkami w których stosowana jest kryptografia z kluczem publicznym są:

- poufny przekaz informacji,
- wymiana kluczy,
- dzielenie sekretów,
- „rzut monetą na odległość”
- dowód wiedzy zerowej.

⁴⁷ Na podst. N. Koblitz, Algebraiczne aspekty kryptografii, WNT Warszawa 2000.

Systemy z kluczem publicznym, mimo iż nie wymagają stosowania poufnego kanału wymiany kluczy, wymagają autentyfikacji, czyli potwierdzenia, że klucz publiczny przy użyciu którego została zaszyfrowana wiadomość, należy rzeczywiście do osoby która ma być właściwym odbiorcą wiadomości. W przeciwnym razie atakujący mógłby podszyć się za którąś z jednostek i wysłać swój klucz publiczny i za pomocą swojego klucza prywatnego odszyfrować treść wiadomości. Sytuację taką przedstawia rysunek 6.1. Problem ten jest rozwiązywany za pomocą specjalnej infrastruktury centrów kluczy publicznych - PKI.

Zakłada się, że wiadomość - tekst jawny, mający być zaszyfrowany i wysłany podzielony jest na bloki w wielkości m . Każdy z bloków może zostać zaszyfrowany niezależnie (analogicznie jak w trybie ECB). Aby jednak uniknąć zamiany kolejności bloków, zaleca się używanie trybu CBC - z łańcuchowym wiązaniem szyfrogramów. Trybu CFB oraz OFB ze względu na specyfikę swego działania nie mogą być w szyfrach z kluczem publicznym używane.

6.2 System RSA

Jednym z najczęściej używanych systemów z kluczem publicznym jest RSA, system nazwany od nazwisk jego twórców: R.Rivesta, A. Shamira oraz L.Adlemana. Za pomocą tego systemu można realizować nie tylko poufną wymianę danych, lecz także implementować techniki podpisu elektronicznego. Bezpieczeństwo RSA opiera się na problemie faktoryzacji liczb całkowitych.

Aby komunikacja dwóch jednostek przy użyciu RSA była możliwa, każda z jednostek musi wygenerować parę kluczy: klucz prywatny oraz klucz publiczny. W celu wygenerowania kluczy, każda z jednostek podejmuje następujące kroki:

1. Generuje losowo dwie różne liczby pierwsze p oraz q , każda zbliżonej wielkości
2. oblicza iloczyn tych liczb $n = pq$, oraz wartość funkcji Eulera $\phi(n) = (p-1)(q-1)$.
3. wybiera losowo liczbę całkowitą e , taką, że: $1 \leq e \leq \phi(n)$. Ponadto liczba e musi być wraz z liczbą $\phi(n)$ wzajemnie pierwsza⁴⁷, tzn: $\gcd(e, \phi(n)) = 1$.
4. używając rozszerzonego algorytmu Euklidesa, oblicza liczbę całkowitą d , taką że: $1 \leq d \leq \phi(n)$ oraz $ed \equiv 1$, co oznacza, że d stanowi multiplikatywną inwersję liczby e w zbiorze liczb całkowitych Z_ϕ (zbiorze w którym działania wykonywane są modulo $\phi(n)$)
5. Kluczem publicznym jest para liczb (n, e) , zaś kluczem prywatnym liczba d .

⁴⁷ gcd - greatest common divisor - największy wspólny dzielnik

Jednostka B - Bob, chcąc wysłać wiadomość do A - Alicji, szyfruje wiadomość m w następujący sposób:

1. zdobywa klucz publiczny Alicji (e, n)
2. wiadomość m przedstawia w postaci liczby całkowitej z przedziału $(0, n-1)$
3. oblicza wartość szyfrogramu c jako $c = m^e \bmod n$
4. wysyła szyfrogram c do Alicji

Alicja odszyfrowuje wiadomość przy użyciu swojego klucza prywatnego d zgodnie z następującym wzorem:

$$m = c^d \bmod n$$

Dowód poprawności systemu RSA znajduje się poniżej:

Ponieważ $ed \equiv 1 \pmod{\phi}$, istnieje liczba całkowita k , taka, że: $ed = 1 + k\phi$. Jeśli $\gcd(m, p) = 1$, to na mocy twierdzenia Fermata:

$$m^{p-1} \equiv 1 \pmod{p}.$$

Podnosząc obie strony tego równania do potęgi $k(q-1)$ i mnożąc przez m otrzymujemy:

$$m^{1+k(p-1)(q-1)} \equiv m \pmod{p}.$$

Jeśli $\gcd(m, p) = p$, wówczas powyższa kongruencja jest również prawdziwa, gdyż każda strona równania jest kongruentna do $0 \pmod{p}$. Dlatego też, we wszystkich przypadkach prawdziwe są kongruencje: $m^{ed} \equiv m \pmod{p}$ oraz $m^{ed} \equiv m \pmod{q}$, z czego wynika, że jeżeli p oraz q są dwoma różnymi liczbami pierwszymi to prawdziwa jest równość:

$$m^{ed} \equiv m \pmod{n} \text{ oraz } c^d \equiv (m^e)^d \equiv m \pmod{n}$$

Przykład szyfrowania RSA dla bardzo małych liczb:

1. Generowanie klucza.

Alicja wybiera liczby pierwsze $p = 2357$, $q = 2551$, oblicza $n = pq = 6012707$.

$\phi = (p-1)(q-1) = 6007800$. Wybiera $e = 3674911$ i używając rozszerzonego algorytmu Euklidesa oblicza multiplikatywną inwersję równą $d = 422191$. Kluczem publicznym Alicji jest więc para liczb $(n = 6012707, e = 3674911)$, zaś kluczem prywatnym liczba $d = 422191$.



2. Szyfrowanie.

Aby wysłać wiadomość $m = 5234673$, Bob oblicza $c = m^e \bmod n = 5234673^{3674911} \bmod 6012707$ używając algorytmu modularnego potęgowania; $c = 3650502$ i wysyła tą wartość do Alicji.

3. Deszyfrowanie

Alicja otrzymując c oblicza wartość m za pomocą klucza prywatnego d w następujący sposób: $c^d \bmod n = 3650502^{422191} \bmod 6012707 = 5234673 = m$

6.2.1 Wielkości kluczy dla RSA

W 1995 roku, jako stosunkowo bezpieczną wielkość modułu n uznawano 512 bitów. Dla zapewnienia maksymalnego bezpieczeństwa zalecano użycie modułu o wielkości 1024 bity. W 1999 roku 512 bitowa liczba określona jako RSA155 została rozłożona na czynniki pierwsze w ciągu siedmiu miesięcy. Wzrost mocy obliczeniowej jaki można zaobserwować od tego czasu, nasuwa więc wniosek, że wielkość modułu wynoszącą 512 bitów nie można już uznać za bezpieczną. Obecnie zaleca się aby informacje o stosunkowo krótkim okresie ważności, szyfrowane były za pośrednictwem 1024 bitowego klucza, zaś ściśle tajne informacje których ważność wynosi kilka lat, za pomocą 2048-bitowego klucza. Do danych mało ważnych można używać klucza o długości 768 bitów, gdyż taka długość wciąż jest poza zasięgiem algorytmów faktoryzujących⁴⁸.

Zaleca się, aby w celu zapewnienia bezpieczeństwa, co 2 lata dokonywać aktualizacji kluczy, oraz dostosowywać ich długość do obecnych standardów. Wciąż dokonywany jest postęp w dziedzinie faktoryzacji liczb i może się okazać, że obecna zalecana długość klucza 1024 bity, nie będzie wkrótce zapewniała należytego poziomu bezpieczeństwa.

Można sobie zadać pytanie: czy RSA, czy w ogóle systemy szyfrowania z kluczem publicznym są równie bezpieczne, czy też może mniej bezpieczne niż systemy symetryczne. Odpowiedź przedstawia Krzysztof Gaj⁴⁹. Uwzględniając postępy w faktoryzacji, pokazuje on, że dla zapewnienia takiego samego bezpieczeństwa co Rijndael 128, długość klucza RSA musiała by wynosić na dzień dzisiejszy ok. 3000 bitów, zaś na rok 2030 - ok. 4000 bitów. Dla Rijndael 256 wielkości te wynoszą odpowiednio 14000 i 18000 bitów. Dla 3DES wielkości te

⁴⁸ Dane na podst. RSA Laboratories, RSA Laboratories Frequently Asked Questions About Today's Cryptography, version 4.1, May 2000, RSA Security Inc.

⁴⁹ Zob. [Mijający rok w kryptografii i kryptoanalizie](#), dr inż. Krzysztof Gaj, George Mason University, USA, Materiały konferencyjne Enigma 2002

wahają się w granicach 2000-3000 bitów. Pozwala to jednoznacznie wysunąć tezę, że systemy z kluczem publicznym są znacznie mniej bezpieczne niż systemy symetryczne.

6.2.2 Bezpieczeństwo RSA

1. Celem atakującego jest znalezienie klucza prywatnego d , mając do dyspozycji jedynie klucz publiczny (n, e) . Jeżeli więc atakujący będzie w stanie dokonać faktoryzacji liczby n w prosty sposób znajdzie wartość funkcji Eulera $\phi(n)$, dzięki czemu obliczenie multiplikatywnej odwrotności liczby e w zbiorze $\phi(n)$, jaką jest liczba d stanowiąca prywatny klucz, będzie stosunkowo prostym zadaniem. Na podstawie tego, można jednoznacznie stwierdzić, że RSA będzie tak długo bezpiecznym systemem szyfrowania, dopóki, dopóty nie zostanie znaleziony efektywny algorytm faktoryzacji liczb całkowitych. Generując pary kluczy, należy więc wygenerować odpowiednio duże liczby pierwsze, tak, aby faktoryzacja liczby n , będącej ich iloczynem, była niewykonywalna.
2. Na podstawie powyższego, determinantami bezpieczeństwa RSA są w dużej mierze liczby pierwsze p oraz q , których iloczyn tworzy moduł n . Jeżeli wielkość modułu ma wynosić 1024 bity, to czynniki powinny mieć długość około 512 bitów. Różnica liczb p oraz q nie może być zbyt mała, w przeciwnym razie można bowiem wysunąć tezę, że $p \approx q$, z czego wynika, że wartość jednego z czynników może być zbliżona wartości pierwiastka modułu n , co znacznie redukuje krąg poszukiwań czynników tego modułu.
3. Kryteria generowania klucza RSA precyzuje ANSI X9.31. Można je podsumować następująco:
 - jeśli publiczny exponent e jest nieparzysty, wówczas powinien on być wzajemnie pierwszy z $(p-1)$ oraz z $(q-1)$
 - jeśli e jest parzyste, wówczas musi być ono wzajemnie pierwsze z $(p-1)/2$, z $(q-1)/2$ oraz $p \neq q \pmod{8}$
 - zarówno p oraz q powinny przejść test pierwszości z błędem mniejszym niż 2^{-100}
 - p oraz q powinny być silnymi liczbami pierwszymi.
 - różnica pomiędzy p
4. Wiele uwagi poświęcono tematowi konieczności użycia silnych liczb pierwszych przy konstrukcji modułu. Zalecenia ANSI X9.31 precyzuje konieczność ich użycia.

Wszelkie rozważania dotyczące konieczności użycia silnych liczb pierwszych miały na uwadze fakt, że za pomocą pewnych algorytmów faktoryzacyjnych (algorytmów Pollarda $p-1$ oraz $p+1$) znacznie trudniej dokonać jest faktoryzacji modułu zbudowanego w oparciu o te liczby. Postęp w dziedzinie faktoryzacji oraz pojawienie się nowych algorytmów (zwłaszcza algorytmów opartych o krzywe eliptyczne) pokazały, że faktoryzacja modułu zbudowanego w oparciu o silne liczby pierwsze charakteryzuje się takim samym stopniem komplikacji jak faktoryzacja modułu zbudowanego o „słabe” liczby pierwsze. Czynnikiem w pełni determinującym bezpieczeństwo RSA jest więc długość i wielkość liczby pierwszej a nie jej długość.

5. Aby przyspieszyć prędkość działania algorytmu, dobrze jest wybrać małą wartość wykładnika e , np $e=3$. Grupa użytkowników może mieć identyczną wartość e , jednakże każdy użytkownik musi mieć własną, odrębną od innych, wartość modułu n . Jeżeli jednostka A zechce wysłać identyczną wiadomość m , do użytkowników których moduły oznaczone są odpowiednio n_1, n_2, n_3 , i których wykładnik jest równy $e=3$, wówczas dla każdego z tych użytkowników oblicza $c_i = m^3 \bmod n_i$. Ponieważ istnieje wysokie prawdopodobieństwo, że moduły są wzajemnie pierwsze, znając c_1, c_2, c_3 używając algorytmu Gaussa znaleźć rozwiązanie układu trzech kongruencji: $x \equiv c_1$; $x \equiv c_2$, $x \equiv c_3$. Ponieważ $m^3 < n_1 n_2 n_3$, na mocy chińskiego twierdzenia o resztach musi być taki przypadek, że $x = m^3$. Obliczając pierwiastek trzeciego stopnia z liczby x można w ten sposób wyznaczyć wartość wiadomości m . Przykład ten jest dowodem na to, że mała wartość wykładnika e nigdy nie powinna być używana do szyfrowania tej samej wiadomości wysyłanej do wielu użytkowników. Aby temu zapobiec, przed rozpoczęciem szyfrowania, do wiadomości często dodaje się pseudolosowo wygenerowaną sekwencję, inną dla każdego z adresatów wiadomości. Oznacza to ekspansję wiadomości. Mała wartość wykładnika e może również stanowić problem dla niewielkich rozmiarów wiadomości. Jeżeli $m < n^e$, wówczas wartość m może być obliczona na podstawie wartości szyfrogramu $c = m^e \bmod n$, poprzez obliczenie pierwiastka e -tego stopnia z c . Dodanie do wiadomości sekwencji pseudolosowej, jak powyżej, również pomaga zapobiec takiemu złamaniu systemu. Częstą wartością publicznego wykładnika jest $e=65537$
6. Wartość klucza prywatnego d również nie może być mała. Jeśli $\text{gcd}(p-1)(q-1)$ jest mały, oraz jeśli liczba bitów wymagana do przedstawienia d jest blisko czterokrotnie mniejsza niż liczba bitów wymagana do przedstawienia modułu n wówczas mając

dany klucz publiczny możliwe jest dokonanie aproksymacji pozwalającej znaleźć klucz prywatny. Aproksymację taką przedstawił D. Boneh.⁵⁰ Aby uniknąć tego rodzaju ataku, zaleca się, aby dla klucza modułu długości 1024 bitów, wartość d wynosiła przynajmniej 256 bitów.

7. RSA charakteryzuje się własnością homomorficzną:

$$(m_1 m_2)^e = m_1^e m_2^e = c_1 c_2 \pmod n$$

Własność ta pozwala na przeprowadzenie ataku adaptacyjnie wybranym szyfrogramem. Załóżmy, że atakujący chce rozszyfrować szyfrogram $c = m^e \pmod n$ przeznaczony dla jednostki A. Wybiera on w tym celu losową liczbę x i za pomocą klucza publicznego A oblicza: $c' = cx^e \pmod n$ po czym wysyła c' do jednostki A. Jednostka A oblicza $m' = (c')^d \pmod n$ po czym odsyła atakującemu m' . Ponieważ:

$$m' = (c')^d = c^d (x^e)^d = mx \pmod n,$$

atakujący może obliczyć $m = m' x^{-1} \pmod n$.

Scenariusz ten, choć trudny do zrealizowania jest możliwy.

6.2.3 Szybkość algorytmu RSA oraz jego implementacje

Algorytm RSA, będący algorytmem z kluczem asymetrycznym, jest znacznie wolniejszy niż systemy szyfrowania z kluczem symetrycznym. Dla przykładu, programowe implementacje DES są stukrotnie szybsze niż RSA. Należy jednak wziąć pod uwagę, że DES został zoptymalizowany pod kątem implementacji sprzętowej i to właśnie w rozwiązaniach sprzętowych pokazuje swoją szybkość. Dlatego też DES, bazujący na rozwiązaniach sprzętowych jest nawet 10000 razy szybszy niż RSA.

Wyjaśnieniem tak słabej efektywności algorytmu jest użycie wielkich liczb - modułów oraz exponentów o wielkościach kilkuset bitów. Na skutek tego każde szyfrowanie/desyfrowanie RSA, powoduje, że mikroprocesor wykonuje modułowe potęgowanie przy użyciu serii modułowych multiplikacji. Od efektywności i implementacji algorytmu modułowego mnożenia, w bardzo dużej mierze zależy więc szybkość działania RSA. Istnieje wiele możliwości implementacji modułowego mnożenia. Ich złożoność oraz wskazówki implementacyjne zostały omówione w dostępnej literaturze⁵¹.

RSA ma szerokie możliwości implementacyjne. Oprócz układów VLSI realizujących sprzętowo RSA możliwe są konstrukcje rozwiązań w oparciu o układy FPGA oraz procesory

⁵⁰ D. Boneh, Twenty Years of Attack on the RSA Cryptosystem, Springer-Verlag 1998

⁵¹ Zob. K.Koc, RSA Hardware Implementation, RSA Laboratories Technical Report TR801, version 1.0-August 1995, także: C.K.Koc, High Speed RSA Implementation, RSA Laboratories Technical Report TR201, version 2.0-November 1994

DSP⁵². Szybkość tych rozwiązań jest bardzo zróżnicowana. Dla DSP wynosi ona kilkadziesiąt kbit/sec, dla układów FPGA - kilkaset kbit/sec.

6.2.4 Warianty RSA

RSA, znalazł wiele zastosowań. Wśród nich najbardziej rozpowszechnionym protokołem wykorzystującym RSA jest SSL służący do ochrony danych w internecie. Zrozumiałą rzeczą jest więc zapewnienie odpowiednich szybkości szyfrowania jak i deszyfrowania programowego. Istnieje wiele rodzajów implementacji programowych RSA, jednakże nawet najszybsze i najdoskonalsze z nich okazują się być zbyt wolne w stosunku do dzisiejszych potrzeb. Fakt ten stał się czynnikiem sprawczym poszukiwań szybszych doskonalszych programowych wariantów RSA, które będą kompatybilne wstecz. Jednym z takich wariantów jest Batch RSA zaproponowany przez A. Fiat. Batch RSA jest szybszy w stosunku do standardu o 300%. Innym wariantem jest wieloczynnikowy RSA (*multifactor RSA*). Wariant ten bazuje na modyfikacji modułu RSA. Jedną z takich modyfikacji jest modyfikacja polegająca na tym, by moduł składał się z iloczynu nie dwóch lecz trzech liczb pierwszych: $n=pqr$, co dla modułu wielkości 1024 bity daje trzy liczby o długości 340 bitów. Operowanie na liczbach o mniejszej długości przyspiesza działanie algorytmu. Druga modyfikacja, zwana Multi-power RSA, polega na tym, że moduł jest tworzony z dwóch liczb p oraz q , lecz ma postać $n=p^2q$, co również znakomicie przyspiesza działanie szyfru. Każda z dwóch modyfikacji charakteryzuje się pełną kompatybilnością wstecz i przyspiesza RSA również o 250-300%. Szczegóły dotyczące obydwu wariantów można znaleźć w literaturze⁵³

⁵² Zob. S.Kaliski, S.R.Dusse, A Cryptographic Library for the Motorola DSP 56000, Eurocrypt 90, Springer Verlag 1998

⁵³ Zob. D.Bonch, H.Shacham, Fast Variants of RSA, CryptoBytes, vol 5, no1, Winter-Spring 2002

Rozdział 7

Podstawowe ataki na szyfry

7.1 Wstęp

A Kerckhoff w 1883 r. powiedział „The security of a cipher must not depend on anything that cannot be easily changed”. Oznacza to, że szyfr powinien opierać swe bezpieczeństwo tylko na tajności klucza, nie na tajemnicy stosowanych przezeń procedur i algorytmów. Analizując bezpieczeństwo szyfru, zakłada się więc, że znajoma jest jego struktura oraz przebieg procesu szyfrowania. Celem ataku jest zatem zdobycie klucza. Na mocy stwierdzenia A. Kerckhoffa można wyróżnić następujące rodzaje ataków:

- atak ze znajomością szyfrogramu (*ciphertext only attack*) - kryptoanalityk dysponuje jedynie szyfrogramem i na jego podstawie próbuje znaleźć klucz bądź tekst jawny
- atak ze znajomością tekstu jawnego (*known plaintext attack*) - w tym przypadku kryptoanalityk dysponuje parą tekst jawny i odpowiadający jej szyfrogram. Na podstawie zawartości tej pary, próbuje wyznaczyć klucz.
- atak z wybranym tekstem jawnym (*chosen plaintext*) - kryptoanalityk może użyć dowolnego tekstu jawnego i na jego podstawie wygenerować odpowiadający mu szyfrogram, gdyż ma do dyspozycji urządzenie szyfrujące. Celem ataku jest zdobycie klucza.
- atak z wybranym szyfrogramem (*chosen ciphertext attack*) - atakujący ma do dyspozycji szyfrogramy oraz urządzenie deszyfrujące. Celem ataku jest zdobycie klucza na podstawie analizy otrzymanych w procesie deszyfrowania, z użyciem różnych nieznanymi kluczy, tekstów jawnych.

Dokonując ataku na dany szyfr należy wziąć pod uwagę złożoność określonego ataku (*attack complexity*). Złożoność taką można zdefiniować jako nakład pracy włożonej przez kryptoanalityka w celu dokonania udanego ataku na szyfr. Na złożoność ataku mają wpływ następujące czynniki:

- liczba tekstów jawnych, szyfrogramów bądź par tekst jawny - szyfrogram, wymagana do przeprowadzenia ataku (*data complexity*),
- wielkość pamięci niezbędnej do przeprowadzenia ataku (*storage complexity*),
- ilość operacji matematycznych, wykonywanych w celu przetworzenia danych (*processing complexity*).

Jeśli n oznacza długość bloku, to mając do dyspozycji 2^n par tekst jawny - szyfrogram, możemy, przy użyciu stałego klucza, w pełni scharakteryzować operację szyfrującą, gdyż stanowi ona permutację. Na podstawie tego można wysunąć następującą implikację dotyczącą

bezpieczeństwa szyfru blokowego - wzrost wielkości bloku może powoduje wzrost złożoności ataku na szyfr, co zwiększa bezpieczeństwo szyfru.

Mając do dyspozycji jedną parę tekst jawny - szyfrogram, można dokonać ataku na szyfr przy użyciu wyczerpującego poszukiwania wartości klucza. Atak taki charakteryzuje się złożonością 2^k . Im większa jest długość klucza, tym większa jest odporność szyfru na tego rodzaju atak.

7.2 Techniki kryptoanalizy

Analizując bezpieczeństwo szyfru, stosuje się różne techniki, których zasadniczym celem jest znalezienie możliwości innego ataku na szyfr niż atak z wyczerpującym poszukiwaniem klucza. Podstawowymi technikami stosowanymi we współczesnej kryptoanalizie szyfrów są:

- analizy statystyczne - badające korelację pomiędzy tekstem jawnym, szyfrogramem oraz kluczem. Prawidłowo zaprojektowany szyfr powinien stanowić całkowicie losową permutację.
- kryptoanaliza różnicowa⁵⁴ - prekursorami tego rodzaju analizy byli E.Biham oraz A.Shamir. Kryptoanaliza różnicowa polega na obserwacji różnic w wartościach szyfrogramów przy znanych różnicach, odpowiadającym tym szyfrogramom, tekstu jawnego. Różnicom tym można przyporządkować określone wartości prawdopodobieństwa i na ich podstawie określić najbardziej prawdopodobna wartość klucza. Kryptoanaliza różnicowa zostanie szczegółowo omówiona w dalszej części pracy przy omawianiu szyfru DES.
- kryptoanaliza liniowa⁵⁵ - w prowadzona przez M. Matsui, polega na aproksymacji szyfru za pomocą liniowej funkcji boolowskiej. Na podstawie tej aproksymacji, przy znajomości par tekst jawny - szyfrogram, możliwe jest znalezienie bitów klucza. Kryptoanaliza liniowa zostanie szczegółowo omówiona w dalszej części pracy przy omawianiu szyfru DES.
- Analiza różnicowa mocy - wprowadzona przez P. Kochera⁵⁶, używana jako metoda kryptoanalizy przy implementacjach sprzętowych, oparta jest na pomiarze wartości różnic poboru mocy przez poszczególne części urządzenia szyfrującego, podczas

⁵⁴ Zob. E.Biham, A.Shamir, Differential Cryptanalysis of DES-like cryptosystems, The Weizman Institute of Science Department of Applied Mathematics, July 1990

⁵⁵ M.Matsui, Linear Cryptanalysis Method for DES Cipher, Eurocrypt 93, Springer Verlag 1993

⁵⁶ P.C.Kocher, J.Jaffe, B.jun, Differential Power Analysis, Crypto 1999

Dane wejściowe funkcji F pochodzące z prawego bloku R , mają długość 32 bitów. Poddawane są ekspansji do 48 bitów. Te 48 bitów jest sumowane modulo 2 z 48 bitowym kluczem cyklu. Zsumowana wartość stanowi wejście dla grupy S-boxów. Otrzymane 32 bitowe słowo wyjściowe z S-Boxów jest poddawane permutacji P .

W rozdziale tym wprowadzone zostaną następujące oznaczenia:

n_x	dowolna wartość hexadecymalna
$P(X)$	permutacja końcowa P funkcji F
$E(X)$	ekspansja bitów
P	64-bitowy blok tekstu jawnego. Wraz z P^* tworzy on parę tekstów jawnych używanych w kryptonalizie. Ich suma XOR oznaczana jest jako $P' = P \oplus P^*$. Zakłada się, że permutacja inicjująca IP oraz końcowa IP^{-1} zostały pominięte z uwagi fakt, iż nie odgrywają one roli w bezpieczeństwie szyfru.
T, T^* i T'	odpowiadające tekstom jawnym wartości szyfrogramów.
$a...j$	odpowiadające numerom cyklu 32 bitowe słowa wejściowe funkcji F . ($a=R_0, b=R_1...$)
$A...J$	jw., lecz słowa wyjściowe
S_i	S-box o numerze i ($i=1...8$)
(L, R)	zawartości rejestrów L i R . odnoszące się do tekstu jawnego
(l, r)	jw., lecz do szyfrogramów
S_{iX}	słowo wejściowe S-boxa o numerze i , w cyklu X ; $S_{iX} = S_{iEX} \oplus S_{iKX}$
S_{iOX}	jw., lecz słowo wyjściowe
S_{iEX}	wartość 6 bitów po ekspansji E w funkcji F , które wraz z kluczem, będą formułowały słowo wejściowe S-boxa o numerze i
S_{iKX}	wartość 6 bitów klucza w funkcji F , które będą, które wraz z bitami wejściowymi formułowały słowo wejściowe S-boxa o numerze i

Dodatkowo można wprowadzić definicję niezależnego klucza, będącego zbiorem kluczy każdego cyklu, niekoniecznie otrzymanych w procesie rozszerzenia klucza głównego. Aby uprościć procedurę analizy prawdopodobieństw, poczynione zostanie założenie, że każdy klucz jest niezależny.

Mając daną wejściową sumę XOR pary wejściowej funkcji F , łatwo można wyznaczyć XOR po ekspansji bitowej $-E$ jako:

$$E(X) \oplus E(X^*) = E(X \oplus X^*)$$



Należy zauważyć, że sumowanie XOR pary wraz z bitami klucza, również nie ma wpływu na wartość XOR pary gdyż:

$$(X \oplus K) \oplus (X^* \oplus K) = X \oplus X^*$$

Dane wyjściowe z poszczególnych S-boxów poddawane są permutacji. Permutacja ta również nie wpływa na sumę XOR pary tekstów jawnych, gdyż:

$$P(X) \oplus P(X^*) = P(X \oplus X^*)$$

Można zatem powiedzieć, że funkcja X jest liniowa względem XOR pary tekstów jawnych. Liniowość ta odnosi się również do poszczególnych cykli, tzn.:

$$(X \oplus Y) \oplus (X^* \oplus Y^*) = (X \oplus X^*) \oplus (Y \oplus Y^*)$$

Jedynymi funkcjami nieliniowym wchodzącymi w skład funkcji są S-boxy, i w tym przypadku znajomość XOR wejściowego, wcale nie oznacza znajomości XOR wyjściowego.

W DES dla każdego S-Box istnieją 64×64 pary wejściowe XOR. Każda z nich ma przyporządkowaną sobie odpowiednią wartość wyjściową, wyznaczoną przez S-box. Całkowita ilość par XOR we-wy ($P'; T'$) dla każdego z S-Box w DES wynosi $64 \times 16 = 1024$, dlatego można stwierdzić, że każdemu zbiorowi ($P'; T'$) przyporządkować można 4 XOR wejściowe. Nie wszystkie jednak zbiory we-wy są możliwe do uzyskania, a te możliwe mają rozkład niejednostajny.

Tablicę pokazującą rozkład XOR wejściowych oraz XOR wyjściowych wszystkich możliwych par we-wy danego S-boxa nazywamy *Tablicą rozkładu par XOR S-boxa*. W tablicy tej, każdy wiersz odnosi się do danego XOR wejściowego, każda kolumna do danego XOR wyjściowego, zaś elementy tabeli stanowią liczby możliwych par wejściowych o XOR określonych numerem wiersza, dających XOR określony numerem kolumny. Sumując wartości elementów każdego wiersza widać, że liczba możliwych w wierszu par wynosi 64, zaś jej średnia 4. Analizując element (0;0) tabeli, widać, że dla zerowego XOR wejściowego (oznaczającego identyczność tekstów jawnych pary wejściowej), XOR wyjściowy musi być również zerowy. Liczba takich par XOR wynosi więc 64.

Jeśli X oznacza słowo 6-bitowe, zaś Y - słowo 4-bitowe, można powiedzieć, że X może spowodować wystąpienie Y w danym S-box, jeśli wystąpi XOR wejściowy równy X , który na wyjściu S-box da XOR wyjściowy równy Y . Oznacz. $X \rightarrow Y$.

Analizując poniższa tabelę, można zauważyć, że dla wejściowego XOR równego 34_x , może wystąpić 8 różnych Y ($1_x; 2_x; 3_x; 4_x; 7_x; 8_x; D_x; F_x$) Jednocześnie widać, że istnieje 16 par wejściowych, których XOR jest równy 34_x , dających na wyjściu S-box XOR wyjściowy równy 2_x . Można więc przyjąć, że $X \rightarrow Y$ z prawdopodobieństwem p , odzwierciedlającym

Z tablicy wartości S_1 odczytać można, że dla wartości 13_x oraz 27_x na wyjściu S-box S_1 otrzyma się wartości 6_x oraz 2_x . Znając parę wejściową oraz XOR wyjściowy można znaleźć wartości klucza używanego podczas cyklu.

XOR wyjściowy (S_{1o}')	Możliwe wartości wejściowe (S_{1i})
1_x	03, 0F, 1E, 1F, 2A, 2B, 37, 3B
2_x	01, 05, 0e, 11, 12, 14, 1A, 1B, 20, 25, 26, 2E, 2F, 30, 31, 3A
3_x	01, 02, 15, 21, 35, 36
4_x	13, 27
7_x	00, 08, 0D, 17, 18, 1D, 23, 29, 2C, 34, 39, 3C
8_x	09, 0C, 19, 2D, 38, 3D
D_x	06, 10, 16, 1C, 22, 24, 28, 32
F_x	07, 0A, 0B, 33, 3E, 3F

Rys 7.3 Tablica wartości wejściowych dla XOR wejściowego 34_x dla S_1 .

Rozważmy S-box S_1 i niech parę wejściową stanowią $S_{IE}=I_x$ oraz $S_{IE}^*=35_x$. Niech wartość odpowiadających im bitów klucza wynosi 23_x . Dokonując sumowania modulo 2 powyższych wartości otrzymujemy:

$$S_{II} = S_{IE} \oplus S_{IK} = I_x \oplus 23_x = 22_x$$

oraz

$$S_{II}^* = S_{IE}^* \oplus S_{IK} = 35_x \oplus 23_x = 16_x.$$

Z S-box S_1 można odczytać, że dla $S_{II}=22_x$ otrzymamy $S_{IO}=I_x$ i odpowiednio dla $S_{II}^*=16_x$ otrzymamy $S_{IO}^*=C_x$. Suma XOR $S_{IO}' = S_{IO} \oplus S_{IO}^*$ ma wartość $S_{IO}'=D_x$.

Założmy więc, że znamy $S_{IE}=I_x$ i $S_{IE}^*=35_x$ oraz $S_{IO}'=D_x$ chcemy wyznaczyć klucz S_{IK} .

Niezależnie od wartości klucza prawdziwe jest: $S_{IE} \oplus S_{IE}^* = S_{IE}' = 34_x = S_{II}'$. Z tablicy rozkładów par XOR odczytujemy, że dla wartości wejściowej $S_{II}'=34_x$ oraz wartości wyjściowej $S_{IO}'=D_x$ wystąpić może 8 różnych wartości XOR tekstów jawnych co implikuje również i klucza 8 różnych wartości klucza ($S_K = S_I \oplus S_E$). Klucz może przyjąć dwie wartości: $S_{IK} = S_{II} \oplus S_{IE}$ lub $S_{IK} = S_{II}^* \oplus S_{IE}^*$. Wartości S_{II} oraz S_{II}^* dające XOR wejściowy równy $S_{II}'=34_x$ przy XOR wyjściowym S_1 równym $S_{IO}'=D_x$, przedstawia poniższa tabela:

$S_{11}'=34_x$	S_{11}	S_{11}^*
		6
dla	32	6
	10	24
$S_{10}'=D_x$	24	10
	16	22
	22	16
	28	1C
	1C	28
	1C	28

Rys. 7.4 Tablica możliwych wartości S_{11} dających XOR wej.= 34_x przy XOR wyj.= D_x

Należy teraz po kolei obliczyć sumy XOR $S_{1K} = S_{11} \oplus S_{1E}$ oraz $S_{1K} = S_{11}^* \oplus S_{1E}^*$ w celu wyznaczenia wartości klucza. Dla wartości $S_{1E}=I_x$ oraz $S_{1E}^*=35_x$ wyniki zostały przedstawione poniżej:

S_{11} lub S_{11}^*	$S_{1K} = S_{1E} \oplus S_{11}$ lub $S_{1K} = S_{1E}^* \oplus S_{11}^*$
06, 32	07, 33
10, 24	11, 25
16, 22	17, 23
1C, 28	1D, 29

Rys.7.5 Tablica możliwych wartości bitów klucza.

Każdy wiersz tej tabeli zawiera 2 pary z użyciem identycznych danych wejściowych ale w odwróconym porządku bitów. Każdej z par można przyporządkować jedną wartość klucza, zatem w każdym wierszu tabeli znajdują się dwa możliwe klucze. Jednym z tych kluczy jest szukany klucz. Używając dodatkowych par, można znaleźć dodatkowe możliwe wartości kluczy, w analogiczny sposób. Za każdym razem jednak, w tabeli możliwych kluczy pojawi się wartość, która będzie występowała we wszystkich przypadkach - dla wszystkich par (przykładowo: Dla $S_{1E}=2I_x$ oraz $S_{1E}^*=15_x$, $S_{10}' = 3_x$ otrzymamy $S_{1K}=03_x, 37_x, 00_x, 34_x, 17_x, 23_x$ - wartością powtarzającą się jest 17_x oraz 23_x). Wartość tą stanowi szukany klucz.

Powyższe rozważania dotyczące XOR we X , XOR wy Y oraz prawdopodobieństwa p można odnieść do całej funkcji F w cyklu DES. Wartości X oraz Y będą jednak w tym przypadku 32 bitowe. Można zatem wprowadzić definicję:

Niech X oraz Y stanowią 32 bitowe wartości. Mówimy, że X może spowodować wystąpienie po transformacji F wartości Y , z prawdopodobieństwem p , jeżeli istnieje pewna liczba p par ze zbioru wszystkich możliwych par, szyfrowanych przy użyciu wszystkich

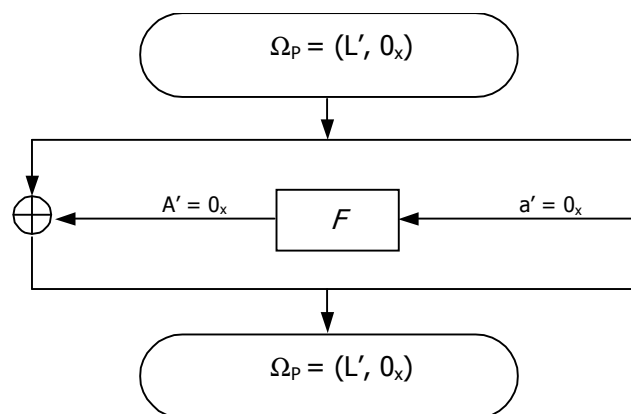
możliwych kluczy, dla których XOR we. równe jest X , a XOR wyj. równe jest Y . Jeśli taka liczba p istnieje oznaczamy ten fakt jako $X \rightarrow Y$.

Prawdopodobieństwo całkowite $X \rightarrow Y$ jest iloczynem poszczególnych prawdopodobieństw z poszczególnych S-boxów odpowiadających im bitów wchodzących w skład X oraz Y .

Dokonując po jednym cyklu szyfrowania DES, dwóch wartości tekstów jawnych, można zauważyć, że operacji takiej można przyporządkować następujące wartości:

- XOR tekstów jawnych X
- XOR szyfrogramów Y
- XOR wartości wejściowych funkcji F
- XOR wartości wyjściowych funkcji F

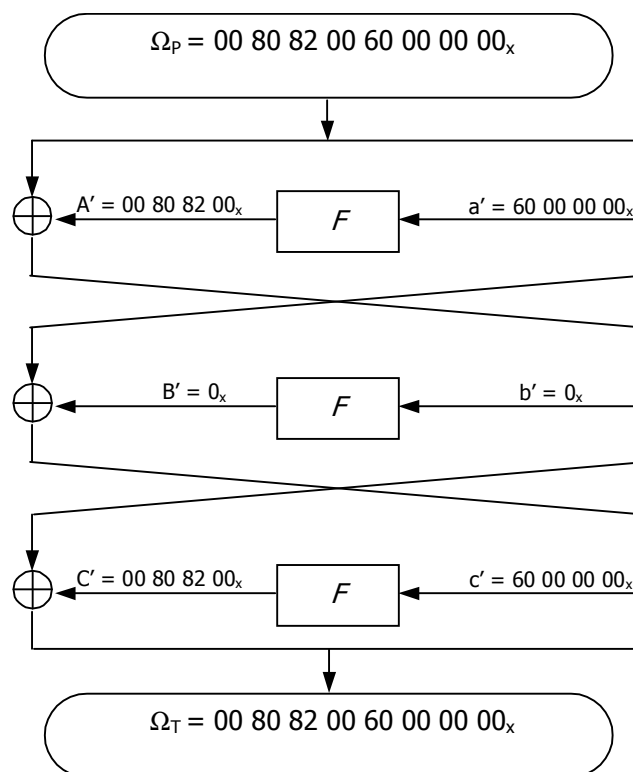
Wartości te tworzą zbiór zwany charakterystyką cyklu. Charakterystyce tej można przypisać odpowiednią wartość prawdopodobieństwa, będącego prawdopodobieństwem, że dla danych wartości wejściowych otrzymamy dane wartości wyjściowe. XOR wartości wejściowych (tekstów jawnych) charakterystyki oznaczona jest jako Ω_p , zaś XOR wartości wyjściowych (szyfrogramów) jako Ω_T . Przykładowa charakterystyka jednego cyklu z prawdopodobieństwem $p=1$ została przedstawiona poniżej.



Rys. 7.6 Charakterystyka jednego cyklu DES z prawdopodobieństwem $p=1$

Charakterystyka ta ma prawdopodobieństwo $p=1$, ponieważ suma XOR pary wejściowej jest równa 0_x (co oznacza że składniki teksty jawne są identyczne) i na wyjściu suma XOR szyfrogramów ma wartość 0_x (identyczność szyfrogramów dla identycznych tekstów jawnych).

N charakterystyk poszczególnych cykli można łączyć, otrzymując w ten sposób charakterystykę n -cykli. Prawdopodobieństwo charakterystyki łącznej jest wówczas iloczynem prawdopodobieństw poszczególnych charakterystyk.



Rys.7.7 Przykładowe połączenie charakterystyk

Prawdopodobieństwo charakterystyki pierwszej i trzeciej równe jest $p=0,21$, prawdopodobieństwo charakterystyki drugiej wynosi $p=1$, zatem prawdopodobieństwo konkatencji charakterystyk wynosi $(0,21)^2=0,05$. Przykład ten pokazuje, że jeśli dwa teksty jawne różnią się pięcioma bitami, ich szyfrogramy po 3 cyklach, będą różniły się trzema bitami, z prawdopodobieństwem $p=0,05$.

Para dla której XOR we. jest równa Ω_p i której XOR wyjściowa jest równa Ω_T nazywana jest parą właściwą. Pozostałe pary to pary niewłaściwe. Analiza wszystkich właściwych par pozwala wyznaczyć najbardziej prawdopodobną wartość klucza. Im większa liczba właściwych par, tym wyznaczenie poprawnego klucza jest łatwiejsze.

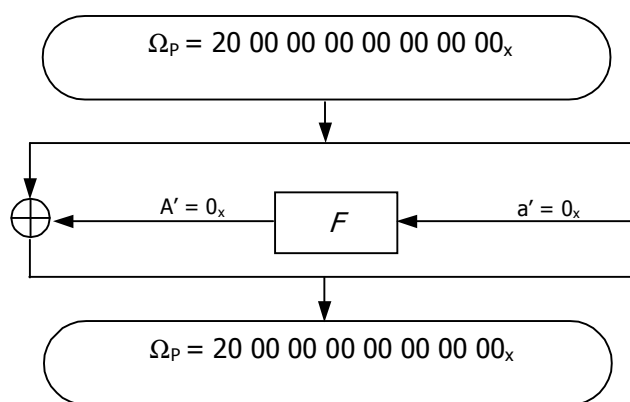
Charakterystyka, w której zamiana wartości rejestrów formułujących Ω_p tworzy Ω_T , jest charakterystyką iteracyjną. Charakterystyki te pozwalają na budowę charakterystyk wielu cykli charakteryzujących się tą własnością, że ich prawdopodobieństwo łączne maleje z każdym dodatkowym cyklem o stałą wartość, a nie jak w przypadku użycia charakterystyk nieiteracyjnych lawinowo. Jednymi ze składowych takich charakterystyk są najczęściej charakterystyki z prawdopodobieństwem $p=1$.

Istnieje wiele rodzajów charakterystyk, jednakże najczęściej używanymi są najprostsze, bazujące na niezerowej XOR wejściowej, mogące formułować zerową XOR wyjściową, co

jest możliwe w DES, dzięki specyficznym własnościom S-boxów⁵⁷ w funkcji F. Najlepsza taka charakterystyka (z $X = 19\ 60\ 00\ 00_x$) ma prawdopodobieństwo $=1/234=0,0043$ i używana jest do łamania DES w 9 i więcej cyklach.

7.3.1 DES zredukowany do 4 cykli

W powyższej sekcji przedstawione zostało ogólne wprowadzenie do kryptoanalizy różnicowej. W tej części zostanie podjęta próba złamania DES zredukowanego do czterech cykli. W tym celu zostanie użyta charakterystyka 1-cyklu z prawdopodobieństwem równym $p=1$.



Rys. 7.8 Charakterystyka jednego cyklu DES z prawdopodobieństwem $p=1$

W drugim cyklu następuje zamiana zawartości bloków. W bloku r' znajduje się więc wartość $20\ 00\ 00\ 00_x$ i ta wartość stanowi daną wejściową dla funkcji F.

$$20\ 00\ 00\ 00_x = 1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_B.$$

XOR wchodzący do S-box S_1 ma wartość 010000 , zaś do pozostałych S-boxów - $S_2 \dots S_8$ - 0_x . Zgodnie z wytycznymi dotyczącymi projektowania S-box zamiana jednego bitu wejściowego S-box musi powodować zmianę co najmniej dwóch bitów wyjściowych S-box. Na wyjściu S_1 otrzymamy więc 0011 (zgodnie z tabelą dla S_1). Najbardziej prawdopodobnym faktem jest fakt, że te dwa bity będą wpływały na dane wyjściowe trzech S-boxów w cyklu trzecim. Na wyjściu funkcji F cyklu trzeciego wystąpi więc różnica 6 bitów. Te bity poddane XOR wraz ze znaną różnicą bitów z cyklu drugiego powodują, że zawartość bloku I na początku cyklu czwartego różni się liczbą siedmiu bitów. Te siedem bitów na skutek ekspansji w funkcji F zmieni zawartość 11 bitów słowa wejściowego S-boxów funkcji F w czwartym cyklu. Jest

⁵⁷ Dokładna analiza S-boxów DES została przeprowadzona przez Y. Desmedt, M. Davio, Dependence of Output on Input in DES: Small avalanche characteristics, Crypto 84, Springer Verlag 1998

Ponieważ 28 bitów B' , oraz a' oraz l' są znane, znanych jest również odpowiadających im 28 bitów D' . Te 28 bitów stanowi wyjście S-boxów $S_2 \dots S_8$ w czwartym cyklu. Oznacza to też znajomość wartości S_{Ed} , S_{Ed}^* oraz S_{Od} . Mając dane zaszyfrowane pary, dla każdego z S-boxów można dokonać sprawdzenia 64 możliwych wartości S_{Kd} poprzez sprawdzenie prawdziwości równania:

$$S_{Id} = S_{Ed} \oplus S_{Kd} \text{ oraz } S_{Id}^* = S_{Ed}^* \oplus S_{Kd}$$

co łącznie daje:

$$S_{Od} \oplus S_{Od}^* = S_{Od}'$$

Dla każdej wartości klucza policzyć można liczbę par spełniających powyższy test. Dla prawdziwego klucza powyższe równości są spełniane dla wszystkich właściwych par. Błędne wartości klucza mogą wystąpić w pozostałych parach.

W ten sposób znalezionych zostało $7 \cdot 6$ bitów klucza K_4 . Jeśli klucze cyklu rozszerzane są według algorytmu DES, wówczas te 42 bity stanowią bity klucza głównego DES. 14 spośród 56 jego bitów jest nadal nieznanymi, co daje już tylko 2^{14} możliwości przy wyczerpującym poszukiwaniu klucza.

Atak za pomocą kryptoanalizy różnicowej może zostać przeprowadzony również dla większej ilości cykli DES. W swojej publikacji⁵⁸ E. Biham i A. Shamir pokazują możliwości przeprowadzenia takiego ataku również dla 6, 8, 9 a także dowolnej ilości cykli, za pomocą różnych charakterystyk. Podsumowanie dotyczące kryptoanalizy różnicowej DES w zależności od liczby cykli zostało przedstawione w formie tabelarycznej poniżej.

ilość cykli	ilość potrzebnych par	ilość par użytych	ilość znalezionych bitów klucza	ilość użytych charakterystyk	prawdopodobieństwo charakterystyk	komentarz
4	2^3	2^3	42	1	1	
6	2^7	2^7	30	3	0,0625	
8	2^{15}	2^{13}	30	5	$9,5 \cdot 10^{-5}$	
9	2^{25}	2^{24}	30	6	10^{-6}	
13	2^{43}	2^{19}	48	11	2^{-40}	
16	2^{57}	2^5	18	15	2^{-56}	wolniejsza niż wyczerpujące poszukiwanie klucza.

E. Biham i A. Shamir pokazują również sposoby przeprowadzenia tego rodzaju ataku na różne modyfikacje DES.

⁵⁸ E. Biham, A. Shamir, Differential Cryptanalysis of DES-like cryptosystems, The Weizman Institute of Science Department of Applied Mathematics, July 1990

Analizy DES, mające na celu uodpornienie szyfru na ten atak pokazały, że modyfikacja permutacji P, porządku S-boxów, XOR nie przyczynia się do zwiększenia odporności szyfru na tego rodzaju kryptoanalizę, a może nawet zmniejszyć bezpieczeństwo szyfru.

Za pomocą kryptoanalizy różnicowej możliwe jest również złamanie szyfru Lucifer, będącego prekursorem DES, oraz wielu innych szyfrów takich jak LOKI, FEAL.

W 1991 roku E. Biham i A. Shamir dokonali kryptoanalizy różnicowej 16 cykli DES ze złożonością mniejszą niż 2^{55} - złożonością wyczerpującego poszukiwania klucza i przedstawili kryptoanalizę różnicową za pomocą której można złamać 16 cykli DES z użyciem 2^{36} szyfrogramów w czasie 2^{37} uzyskanych z puli 2^{47} tekstów jawnych⁵⁹.

7.4 Kryptoanaliza liniowa.

Na konferencji Eurocrypt 93' M. Matsui wprowadził nową metodą kryptoanalizy zwaną kryptoanalizą liniową⁶⁰, używając tej metody do ataku na DES z użyciem 2^{47} znanych tekstów jawnych⁶¹.

Kryptoanaliza liniowa zajmuje się badaniem liniowych związków pomiędzy bitami tekstu jawnego, szyfrogramu oraz klucza. Związki te używane są do znalezienia wartości bitów klucza. Celem tego rodzaju kryptoanalizy jest więc znalezienie liniowej aproksymacji równania opisującego dany szyfr.

Na wstępie należy wprowadzić podstawowe pojęcia jakie używane są w kryptoanalizie liniowej. Postać liniowa funkcji boolowskiej m zmiennych wejściowych X oraz n zmiennych wyjściowych Y może zostać przedstawiona w formie:

$$X_1 \oplus X_2 \oplus \dots \oplus X_n = Y_1 \oplus Y_2 \oplus \dots \oplus Y_m$$

Jeżeli za powyższe wartości X_i oraz Y_j podstawimy w sposób całkowicie losowy wartości ze zbioru $\{0;1\}$, wówczas równanie to będzie spełnione z prawdopodobieństwem $P=0,5$.

Dewiację od wartości prawdopodobieństwa $P=0,5$ nazwać można odchyleniem od liniowości równania (*linear probability bias*); w pracy tej zostanie użyty termin odchylenie ozn ϵ .

Jeśli P_L stanowić będzie wartość prawdopodobieństwa z jakim równanie boolowskie jest spełnione, wówczas dla $P_L = 1$ szyfr będzie szyfrem całkowicie liniowym, zaś dla $P_L = 0$, szyfr będzie oparty na relacjach afinicznych pomiędzy poszczególnymi zmiennymi. Obie te wartości oznaczają słabość szyfru i jego idealną podatność na atak za pomocą kryptoanalizy liniowej.

⁵⁹ E.Biham, A.Shamir, Differential Cryptanalysis of the full 16-round DES, Technion -Computer Science Department - Technical Report CS0708-1991

⁶⁰ M.Matsui, Linear Cryptanalysis Method for DES Cipher, Eurocrypt 93, Springer Verlag 1993

⁶¹ M.Matsui, The First Experimental Cryptanalysis of the Data Encryption Standard, Crypto 94, Springer Verlag 1998

Rozkłady prawdopodobieństwa zmiennych równania można zdefiniować jako:

$$P_R (X_1=X_i) = \{ P_1 \text{ dla } i = 0 ; 1-P_1 \text{ dla } i=1 \}$$

$$P_R (X_2=X_i) = \{ P_2 \text{ dla } i = 0 ; 1-P_2 \text{ dla } i=1 \}$$

Jeżeli zmienne te są niezależne wówczas:

$$P_R (X_1=X_i , X_2=X_j) = \{ P_1P_2 \text{ dla } i = 0, j=0 ; P_1(1-P_2) \text{ dla } i=0, j=1 ; (1-P_1)(1-P_2) \text{ dla } i=1, j=1 \}$$

Można zatem przyjąć, że

$$\begin{aligned} P_R (X_1 \oplus X_2 = 0) &= P_R (X_1 = X_2) = \\ &= P_R (X_1 = 0, X_2 = 0) + P_R (X_1 = 1, X_2 = 1) = \\ &= P_1P_2 + (1-P_1)(1-P_2) \end{aligned}$$

Jeżeli P_1 wyrażone zostanie w postaci odchylenia wówczas: $P_1 = 0,5 + \varepsilon_1$. P_2 przyjmuje postać analogiczną. Zatem:

$$P_R (X_1 \oplus X_2 = 0) = 0,5 + 2\varepsilon_1\varepsilon_2$$

M. Matsui pokazał, że:

$$P_R (X_1 \oplus X_2 \oplus \dots \oplus X_n = 0) = 0,5 + 2^{n-1} \prod_{i=1}^n \varepsilon_i$$

Wprowadzając kryptoanalizę liniową, Matsui wprowadził ją na przykładzie DES, dlatego też wszelkie odwołania i opis tego rodzaju kryptoanalizy przedstawiony zostanie dla DES. Wszystkie transformacje DES, z wyjątkiem S-boxów, są operacjami liniowymi. Aproksymacja liniowa cyklu DES sprowadza się więc do aproksymacji liniowej S-boxów w funkcji F.

Związki pomiędzy wartościami zmiennych wejściowych a wartościami wyjściowymi danego S-boxa można określić poprzez sprawdzenie XOR określonej funkcji wejściowej jest równy XOR określonej funkcji wyjściowej.

Niech zatem:

X stanowi wektor zmiennych wejściowych S-box, postaci $X_6X_5X_4X_3X_2X_1$,

Y wektor zmiennych wyjściowych S-box postaci $Y_4Y_3Y_2Y_1$, zaś:

α - sześciobitowa liczba całkowita. Poszczególne bity tej liczby ozn. są jako $\alpha_1 \dots \alpha_6$

β - czterobitowa liczba całkowita. Poszczególne bity tej liczby ozn. są jako $\beta_1 \dots \beta_4$

Dla danego S-box S_i ($i=1 \dots 8$), oraz dla danych $1 \leq \alpha \leq 63$ i $1 \leq \beta \leq 15$, można wprowadzić wartość $NS_i(\alpha; \beta)$, jako liczbę sekwencji wejściowych danego S_i , dla których suma XOR

bitów wejściowych X maskowanych przez α , równa jest sumie XOR bitów wyjściowych Y maskowanych przez β , tzn.

$$NS_i(\alpha;\beta) = N\{x \mid 0 \leq x < 64 \quad (\bigoplus_{i=1}^6 (X_i \text{ AND } \alpha_i)) = (\bigoplus_{j=1}^4 (Y_j \text{ AND } \beta_j))\}$$

Przykładowo $NS_5(16, 15) = 12$. Oznacza to że $X_5 = Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4$ dla 12 z 64 możliwych funkcji zmiennych wejściowych S-boxa S_5 . Oznacza to też, że wartość X_5 S-boxa S_5 wpływa na XOR bitów wyjściowych tego S-boxa z prawdopodobieństwem $P=12/64=0,19$.

Jeżeli liczba $NS_i(\alpha;\beta)$ jest równa 32, wówczas S-box jest nieliniowy. Każde odchylenie od wartości 32, oznacza pewną liniowość. Aby przedstawić liniowość w prosty sposób Matsui wprowadził tabele aproksymacji liniowych każdego S-boxa, dla S-box S_5 .

Elementy tej tabeli stanowią ilość przypadków dla których XOR zmiennych wejściowych określonych przez α , jest równe XOR zmiennych wyjściowych Y określonych przez β , pomniejszoną o wartość 32. Im zatem większa wartość w poszczególnym elemencie tej tabeli, tym bardziej liniowy jest S-box. Wartości ujemne oznaczają relacje afiniczne pomiędzy zmiennymi.

Po analizie tabeli aproksymacji można zauważyć, że największa wartość występuje w polu (16, 15) i jest równa ona 12. Oznacza to, że $X_5 = Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4$ jest najlepszą liniową aproksymacją S_5 , gdyż jej odchylenie $\epsilon = 12/64 = 0,19$.

Biorąc pod uwagę wpływ permutacji P oraz ekspansji E w funkcji F cyklu DES, równanie $X_5=Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4$ dla S_5 przyjmuje postać:

$$X[15] + K[22] = F[7] \oplus F[18] \oplus F[24] \oplus F[29].$$

gdzie:

$X[15]$ - piętnasty bit rejestru R ,

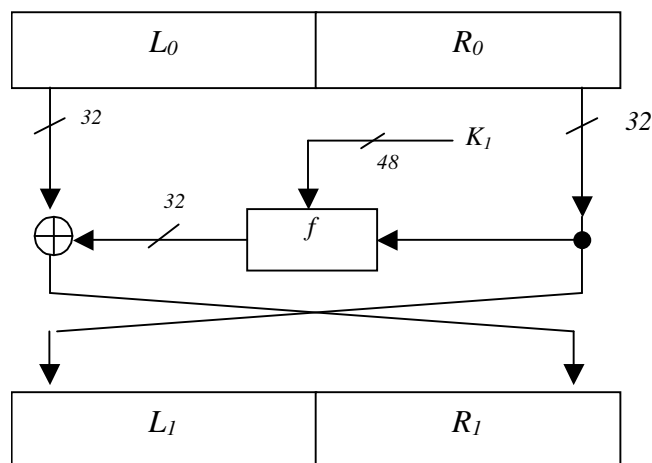
$K[22]$ - bitem klucza w pozycji 22-ej,

$F[i]$ - bity wyjściowe na i -tych pozycjach funkcji F w cyklu.

β	α															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2_x	0	4	-2	2	-2	2	-4	0	4	0	2	-2	2	-2	0	-4
3_x	0	0	-2	6	-2	-2	4	-4	0	0	-2	6	-2	-2	4	-4
4_x	0	2	-2	0	0	2	-2	0	0	2	2	4	-4	-2	-2	0
5_x	0	2	2	-4	0	10	-6	-4	0	2	-10	0	4	-2	2	4
6_x	0	-2	-4	-6	-2	-4	2	0	0	-2	0	-2	-6	-8	2	0
7_x	0	2	0	2	-2	8	6	0	-4	6	0	-6	-2	0	-6	-4
8_x	0	0	2	6	0	0	-2	-6	-2	2	4	-12	2	6	-4	4
9_x	0	-4	6	-2	0	-4	-6	-6	6	-2	0	-4	2	-6	-8	-4
A_x	0	4	0	0	-2	-6	2	2	2	2	-2	2	4	-4	-4	0
B_x	0	4	4	4	6	2	-2	-2	-2	-2	2	0	-8	-4	0	0
C_x	0	2	0	-2	0	2	4	10	-2	4	-2	-8	-2	4	-6	-4
D_x	0	8	0	2	0	-2	4	-10	-2	0	-2	4	-2	8	-6	0
E_x	0	-2	-2	0	-2	4	0	2	-2	0	4	2	-4	6	-2	-4
F_x	0	-2	-2	8	6	4	0	2	2	4	8	-2	8	-6	2	0
10_x	0	2	-2	0	0	-2	-6	-8	0	-2	-2	-4	0	2	10	-20
11_x	0	2	-2	0	4	2	-2	-4	4	2	2	0	-8	-6	2	4
12_x	0	-2	0	-2	2	-4	-2	-8	4	6	4	6	-2	4	-6	0
13_x	0	-6	0	2	-2	4	2	0	4	-6	4	2	-6	4	-2	0
14_x	0	4	-4	0	0	0	0	0	-4	-4	4	4	0	4	-4	0
15_x	0	4	0	-4	-4	4	-8	-8	0	0	-4	4	8	4	0	4
16_x	0	0	6	6	2	-2	4	0	4	0	6	2	2	2	0	0
17_x	0	4	-6	-2	6	-2	-4	4	4	-4	-6	2	-2	2	0	4
18_x	0	6	0	2	4	-10	-4	2	2	0	-2	0	2	4	-2	-4
19_x	0	2	4	-6	0	-2	4	-2	6	8	6	4	10	0	2	-4
$1A_x$	0	2	2	-8	-2	4	0	2	-2	0	4	2	0	-2	-2	0
$1B_x$	0	2	6	-4	-6	0	0	2	6	8	0	-2	-4	-6	-2	0
$1C_x$	0	6	-2	2	4	0	-6	2	-2	6	-4	0	2	-2	0	0
$1D_x$	0	4	-2	6	-8	0	-2	2	10	-2	-8	-8	2	2	0	4
$1E_x$	0	-4	-8	0	-2	-2	-2	2	-2	2	-2	6	4	4	4	0
$1F_x$	0	-4	8	-8	2	-6	-6	-2	-2	2	-2	-2	-8	0	0	-4
20_x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21_x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22_x	0	-4	-2	2	-2	2	-4	8	-4	0	-6	6	2	-2	-16	-12
23_x	0	0	-2	-2	6	-2	-4	4	0	0	-2	-2	-2	6	4	-4
24_x	0	-2	6	4	0	6	-2	4	4	-6	-2	4	0	14	2	0
25_x	0	6	2	0	0	6	2	0	-4	-6	2	-8	0	-2	6	-4
26_x	0	2	4	-2	-2	0	2	-4	4	-2	-4	-2	6	0	-2	0
27_x	0	-10	0	-2	6	4	6	-4	0	6	-12	2	2	0	6	-4
28_x	0	4	-2	-2	0	4	-6	2	2	-6	4	0	6	-2	-4	0
29_x	0	0	2	6	0	0	6	2	2	-2	-8	0	-2	-6	0	0
$2A_x$	0	0	-4	-8	6	6	6	-6	6	2	-2	-2	-8	4	-4	4
$2B_x$	0	8	0	4	6	-2	-6	6	2	6	-2	6	-4	0	4	4
$2C_x$	0	2	4	-6	0	-6	0	6	-2	-4	2	-4	-2	4	6	0
$2D_x$	0	-2	-4	-2	0	-2	-8	2	-2	0	-6	-8	-2	0	-2	4
$2E_x$	0	6	2	-4	6	4	4	-2	-10	-8	0	-2	4	-2	2	0
$2F_x$	0	6	-6	-4	6	-4	4	-2	2	4	4	-6	0	2	-2	-4
30_x	0	2	-2	0	-4	-6	-2	-4	4	2	2	0	0	2	2	4
31_x	0	2	-2	0	0	-2	2	0	0	-2	-2	-4	0	2	2	4
32_x	0	6	0	-2	-2	8	2	4	0	10	0	2	-2	4	2	0
33_x	0	-6	0	10	2	0	-2	-4	0	6	0	-10	2	4	-2	0
34_x	0	0	-12	4	-4	0	4	-8	-4	0	-4	0	-4	-4	0	0
35_x	0	-8	0	0	8	-4	4	0	0	-4	-4	0	4	4	-4	4
36_x	0	4	-2	-6	-2	-2	8	0	4	-4	-2	-2	6	2	-4	0
37_x	0	-8	-6	-6	-6	6	0	4	12	0	2	-2	2	2	4	-4
38_x	0	2	4	-6	0	-2	4	-2	-6	4	-6	0	6	4	-2	0
39_x	0	-2	8	2	-4	6	-4	-6	-2	-4	2	4	-2	0	2	0
$3A_x$	0	6	-10	0	2	4	0	-2	6	-4	0	2	4	-2	-2	-4
$3B_x$	0	-2	-6	-4	10	0	-8	-2	-10	4	4	-2	0	2	-2	4
$3C_x$	0	-8	-6	-2	0	-4	2	2	-6	2	4	0	10	-2	4	4
$3D_x$	0	4	2	2	4	4	-2	2	-2	10	0	0	2	2	4	0
$3E_x$	0	-4	4	-4	2	2	-2	2	2	-2	-2	4	-4	0	4	4
$3F_x$	0	-4	-4	-4	14	6	-6	-2	2	-2	6	-2	0	0	-4	0

Rys. 7.10 Tablica aproksymacji liniowych S-boxa S_5 .

Ponieważ pojedynczy cykl DES przedstawić można jako:



Rys.7.11 Pojedynczy cykl DES

Na podstawie powyższego rysunku, aproksymację liniową S_5 w odniesieniu do całego cyklu można opisać:

$$R_1 [7,18,24,29] \oplus L_0 [7,18,24,29] = K_1[22] \oplus R_0[15]$$

Aproksymacja ta ma prawdopodobieństwo równe 0,19. W analogiczny sposób można opisać trzecią iterację DES, jako:

$$R_1 [7,18,24,29] \oplus L_2 [7,18,24,29] = K_2[22] \oplus R_2[15]$$

Dokonując konkatencji równań otrzymujemy aproksymację liniową trzech cykli DES, postaci:

$$L_0 [7,18,24,29] \oplus L_2 [7,18,24,29] \oplus R_0[15] \oplus R_2[15] = K_1[22] \oplus K_2[22]$$

Aproksymacja ta spełniona jest z prawdopodobieństwem $p=(0,19)^2 + (1-0,19^2) = 0,7$.

Mając do dyspozycji odpowiednią liczbę tekstów jawnych N , oraz prawdopodobieństwa poszczególnych aproksymacji, z bardzo dużym prawdopodobieństwem można wyliczyć wartości odpowiednich bitów klucza.

Każda z takich aproksymacji tworzy określoną liniową charakterystykę cyklu. Poprzez połączenie określonych aproksymacji, odnosząc je do poszczególnych cykli, można budować liniowe charakterystyki wielu cykli. Dla 5 cykli DES, taką charakterystyką, bazującą na równaniu używanym w przykładzie poprzednim, oraz równaniu aproksymującym S_1 postaci:

$$X[27] \oplus X[28] \oplus X[30] \oplus X[31] \oplus F[15] = K[42] \oplus K[43] \oplus K[45] \oplus K[46]$$

jest charakterystyka opisana zależnością:



$$\begin{aligned} L_0[15] \oplus R_0[7,18,24,27,28,29,30,31] \oplus L_4[15] \oplus R_4[7,18,24,27,28,29,30,31] = \\ = K_1[42,43,45,46] \oplus K_2[22] \oplus K_4[22] \oplus K_5[42,43,45,46] \end{aligned}$$

Charakterystyka ta powstała z zastosowania aproksymacji S-boxa S_5 użytej w iteracji drugiej i czwartej, oraz aproksymacji S_1 w iteracji pierwszej i ostatniej. Prawdopodobieństwo tej charakterystyki wynosi 0,519. M.Matsui pokazuje⁶², że dla 2800 par tekst jawny - szyfrogram, w 97,7% znaleźć można bity klucza występujące w prawej stronie równania opisującego charakterystykę.

Wraz ze wzrostem liczby cykli, rośnie złożoność aproksymacji liniowych oraz liczba par tekst jawny - szyfrogram, niezbędnych do prawidłowego rozwiązania aproksymacji liniowej. Dla 16-cykli DES, Matsui przedstawia charakterystyki umożliwiające znalezienie 14 bitów klucza z użyciem 2^{47} tekstów jawnych. Pozostałe 42 bity znajdowane są za pomocą wyczerpującego szukania klucza. Na Crypto 94 Matsui zaprezentował atak⁶³ na 16 cykli DES, z użyciem dwóch aproksymacji liniowych obejmujących 14 cykli. Każda z tych aproksymacji pozwoliła na znalezienie 13 bitów klucza. Atak ten pozwolił więc znaleźć łącznie 26 bitów klucza z użyciem 2^{43} tekstów jawnych. Pozostałe 30, znalezione zostało za pomocą poszukiwania wyczerpującego. Używając 12 komputerów HP9735/PA-RISC 99MHz, DES został w pełni złamany poprzez znalezienie 56 bitów klucza po 50 dniach obliczeń, z czego w ciągu 40 dni generowane były pary tekst jawny - szyfrogram, a tylko 10 dni trwało wyczerpujące poszukiwanie klucza..

Do dalszego rozwoju kryptoanalizy liniowej, a tym samym efektywniejszego złamania DES przyczynili się L.R.Knudsen i M.R.B.Robshaw, prezentując⁶⁴ na EuroCrypt 96 możliwość zastosowanie równań aproksymacji nieliniowych w kryptoanalizie liniowej. Przedstawili oni znacznie lepsze aproksymacje S-boxów, niż te proponowane przez Matsui.

Wprowadzona przez Matsui technika kryptoanalizy liniowej, uogólnionej przez L.R.Knudsen i M.R.B.Robshaw stała się jedną z najbardziej zaawansowanych oraz efektywnych technik kryptoanalitycznych. Pozwoliła ona złamać nie tylko DES, ale i takie algorytmy jak FEAL, oraz LOKI, w czasie znacznie krótszym niż wyczerpujące przeszukiwanie klucza. Stała się czynnikiem sprawczym oraz inspiracją do projektowania i konstrukcji nowych odpornych na tą technikę algorytmów.

⁶² M.Matsui, Linear Cryptanalysis Method for DES Cipher, Eurocrypt 93, Springer Verlag 1993

⁶³ M.Matsui, The First Experimental Cryptanalysis of the Data Encryption Standard, Crypto 94, Springer Verlag 1998

⁶⁴ L.R. Knudsen, M.R.B.Robshaw, Non-Linear Approximations in Linear Cryptanalysis, Eurocrypt 96, Springer Verlag 1998

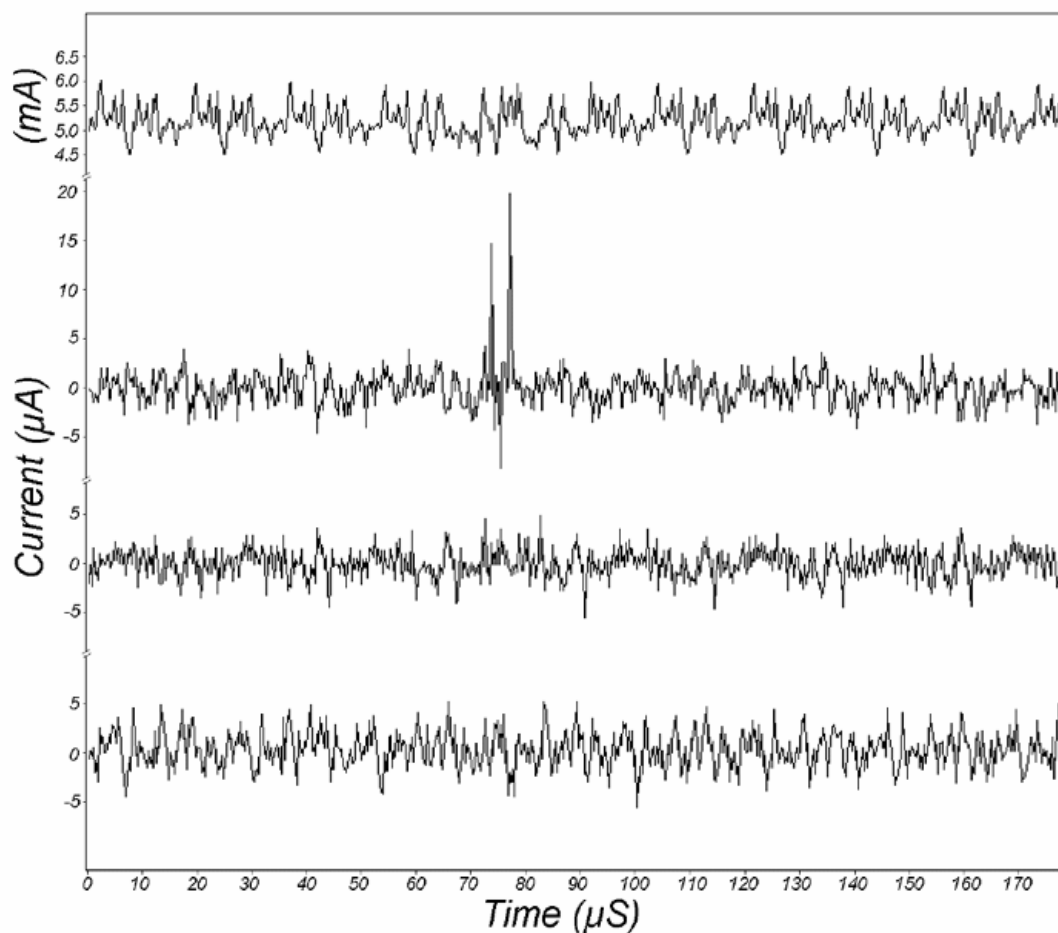
- permutacje DES - implementacje DES dokonują wielu permutacji. Warunkowe obejścia w wykonywanym kodzie przyczyniają się do różnych charakterystyk prądowych dla różnych wartości bitów
- porównania - porównania łańcuchów lub obszarów pamięci zazwyczaj realizowane są przy użyciu warunkowych obejść.
- potęgowania- zwykle potęgowanie modularne powoduje podnoszenie wartości do kwadratu przy wykonaniu każdej iteracji, co powoduje powtarzanie charakterystyk dla każdej iteracji. Na podstawie ich kształtu oraz liczby można zatem oszacować wielkość wykładnika.

Drugą techniką jest różnicowa analiza poboru mocy DPA. Atak na DES za pomocą DPA składa się z dwóch faz. W pierwszej fazie należy dokonać pomiarów charakterystyk ok. 1000 operacji DES. Druga faza opiera się na tworzeniu charakterystyk różnicowych powyższych pomiarów i ich analizie. W tym celu konstruowana jest funkcja wybierająca $D(K_i, C, b)$, zależna od wartości klucza oraz szyfrogramu. Wartość tej funkcji zależy wartości bitu $0 \leq b \leq 32$ w bloku L na początku szesnastego cyklu deszyfrowania szyfrogramu C, zaś 6 bitów klucza wchodzących do S-boxa przyporządkowanego bitom b jest liczbą $0 \leq K_i \leq 64$. Następnie generowany jest przebieg różnicowy, poprzez znalezienie różnicy w przebiegach, dla których funkcja selekcji ma wartość zero oraz w przebiegach, dla których funkcja selekcji ma wartość 1. Jeżeli wartość K_i jest niepoprawna wartość bitu obliczona za pomocą funkcji wybierającej różni się dla połowy używanych szyfrogramów. Przebieg różnicowy ma wówczas wartość zbliżoną do zera z pewnymi fluktuacjami. Jeżeli K_i jest poprawne, wówczas wartość obliczona za pomocą funkcji wybierającej jest równa szukanemu bitowi b . Funkcja selekcji jest więc skorelowana z wartością bitu b podlegającego manipulacji w szesnastym cyklu. Rezultatem tego jest wzrost poboru mocy w momencie przetwarzania tego bitu. Ponieważ wartość poboru mocy zależy od wartości jaką mają przetwarzane bity, na przebiegu różnicowym w momentach przetwarzania tych bitów zauważyć można skoki gwałtowne skoki pobory prądu. Na podstawie tego można zidentyfikować poprawne wartości klucza.

Poniżej znajduje się rysunek (rys. 7.15) obrazujący cztery przebiegi sporządzone przy badaniach karty chipowej szyfrującej DES. Pierwszy z nich przedstawia średni pobór prądu przy operacjach DES. Przebiegi nr 2, 3, 4 to przebiegi różnicowe. Przebieg nr2 został zdjęty przy użyciu właściwej wartości klucza K_i , przebiegi nr3 i nr4 - przy użyciu niepoprawnej wartości K_i .

DPA może zostać użyta nie tylko do kryptoanalizy DES. Uniwersalność jej oraz możliwości adaptacyjne czynią ją dobrym narzędziem do kryptoanalizy algorytmów asymetrycznych przy ich implementacjach hardwarowych. Według jej twórców⁶⁵, możliwe jest za jej pomocą dokonywanie *reverse engineeringu* nieznanymi szyfrów lub protokołów. Dodatkowym atutem jest niski koszt jej przeprowadzenia, oraz względna prostota.

Istnieją techniki pozwalające DPA w wysokim stopniu skomplikować. Są to techniki zarówno programowe jak i sprzętowe. Wpływają one jednak znacząco na złożoność a tym samym koszt rozwiązań kryptograficznych. Pozwala to wysunąć tezę, że o sile rozwiązania kryptograficznego decyduje nie tylko doskonałość zastosowanych technik kryptograficznych, ale także ich odpowiednia implementacja.



Rys. 7.15 Oscylogram pracy karty realizującej szyfrowanie DES - 4 warianty.

⁶⁵ Zob. P.C.Kocher, J.Jaffe, B.Jun, Differential Power Analysis, Crypto 1999

7.6 Atak bazujący na pomiarze czasu wykonywanych operacji

Obok analizy poboru mocy, istnieje jeszcze drugi rodzaj ataku, związany z bezpośrednią implementacją szyfru, atak oparty na pomiarze czasu wykonywanych operacji i dedukcji bitów klucza na podstawie otrzymanych charakterystyk czasowych.

Wiadomym jest fakt, iż w kryptosystemach czas przetwarzania danych, często w dużej mierze zależy od ich wartości. Składa się na to wiele przyczyn, składających się na optymalizację szybkości działania, takich jak: obejścia, instrukcje warunkowe, rodzaj zastosowanego algorytmu; oraz przyczyn wynikających bezpośrednio z konstrukcji określonego sprzętu - różnego czasu wykonywania poszczególnych instrukcji. Znając zatem pewne charakterystyki czasowe, możliwe jest na ich podstawie uzyskanie pewnych informacji o danych wejściowych - np. kluczu, takich jak. waga Hamminga czy też przybliżony rząd wielkości klucza. Informacje te w znacznym stopniu przyczyniają się do ułatwienia ostatecznego ataku.

Najczęstszym celem ataku na RSA jest wartość prywatnego klucza k . Operacją związaną z użyciem prywatnego klucza jest deszyfrowanie, lub też podpisywanie wiadomości opisane $m^k \bmod n$, w przypadku gdy za pomocą RSA dokonujemy podpisu elektronicznego. Zakłada się, że atakujący zna projekt i implementację algorytmiczną atakowanego systemu.

Założmy więc, że obliczenie $m^k \bmod n$ jest wykonywane z użyciem algorytmu Montgomery, za pomocą mnożenia i podnoszenia do kwadratu:

```
x = m
FOR i = n - 2 DOWNT0 0
    x = x2
    IF (ki = 1) THEN
        x = x * m
ENDFOR
RETURN x
```

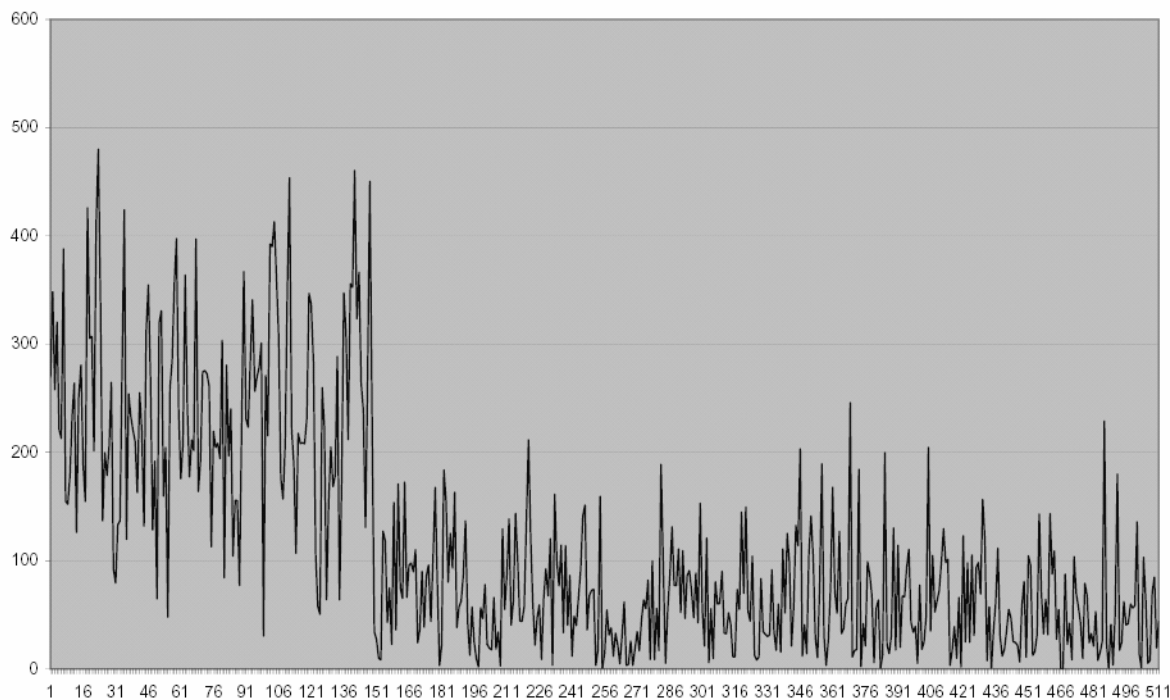
Czas mnożenia za pomocą algorytmu Montgomery jest stały, niezależny od czynników, chyba, że pośredni rezultat mnożenia jest większy niż moduł n , wówczas na końcu mnożenia wykonywane jest dodatkowe odejmowanie, zwane redukcją. Oznacza to, że dla niektórych czynników operacja mnożenia będzie wykonywana nieznacznie dłużej. Można więc dokonać ataku bazującego na pomiarze czasu wykonywanych operacji, w oparciu o mnożenie liczb w algorytmie.

Założmy, że pierwszy bit klucza k_1 ma wartość 1. Atak można zacząć więc od drugiego bitu klucza k_2 . Obserwując algorytm krok po kroku, można zauważyć, że jeśli wartość tego bitu będzie wynosić 1, wykonywana będzie instrukcja $x=x*m$, a tym samym obliczana będzie wartość $m*m^2$. Dla niektórych wiadomości m (dla tych dla których pośredni rezultat mnożenia będzie większy niż moduł n), podczas mnożenia będzie wykonywana dodatkowa redukcja, dla niektórych zaś nie. Jeżeli zbiór badanych wiadomości M będzie wystarczająco liczny, wówczas badane próbki można podzielić na dwa podzbiory: M_1 - zbiór wiadomości dla których przy mnożeniu $m*m^2$ dodatkowa redukcja będzie wykonywana, oraz M_2 - zbiór wiadomości dla których, przy mnożeniu $m*m^2$ redukcja nie będzie wykonywana. Jeżeli wartość bitu k_2 będzie rzeczywiście 1, wówczas można się spodziewać, że czasy obliczeń dla wiadomości z M_1 będą odpowiednio wyższe niż czasy obliczeń wiadomości z M_2 .

Z drugiej strony, jeśli wartość bitu k_2 będzie równa zero, wówczas mnożenie $m*m^2$ nie będzie wykonywane. Podział na dwa zbiory jest wówczas bezsensowny. Nie ma powodu, dla którego m powodowało by redukcję przy wykonywaniu $m*m^2$, dlatego też dwa zbiory będą wyglądały losowo i nie będzie możliwe zauważenie znaczących różnic w czasach wykonywania obliczeń. Analizując przebiegi czasowe można więc wydedukować na ich podstawie bit klucza k_i . Analogicznie zestaw powyższych operacji można przeprowadzić w celu znalezienia kolejnych bitów klucza.

Pomiaru charakterystyk czasowych można również dokonywać dla innych operacji algorytmu, takich jak np. podnoszenie do kwadratu. Charakterystyki te mogą się okazać bardziej miarodajne. Przytoczony przykład oparty na mnożeniu miał na celu tylko charakter zapoznawczy.

Interesującą własnością tego rodzaju ataku jest własność detekcji błędów. Atak ten polega na symulacji obliczeń do pewnego momentu. Następnie na podstawie analizy podejmowana jest decyzja jaką ma wartość szukany bit klucza. Każdy krok ataku jest w pewnym stopniu związany z poprzednim. Założmy więc, że dla bitu klucza k_i została podjęta błędna decyzja. W kolejnym kroku, obliczenia nie będą mogły zostać przeprowadzone prawidłowo i nie będzie można stwierdzić, czy dodatkowa redukcja była wykonywana czy też nie. Kryterium decyzyjne stanowią średnie różnice czasów obliczeń dla wiadomości z dwóch zbiorów. Po podjęciu nieprawidłowej decyzji, charakterystyki czasowe dla obydwu zbiorów będą zbliżone. Sytuację taką przedstawia poniższy rysunek (rys 7.16).



Rys. 7.16 Detekcja błędu dla 512 bitowego klucza.

Analizując powyższy wykres, widać że błąd w wyznaczeniu wartości bitu wystąpił około 150-ego bitu klucza. Z powyższej charakterystyki również widać, że w niektórych przypadkach, mimo iż właściwie określono wartości bitów, niektóre z wartości różnic czasu, stanowiących kryterium decyzyjne, są bardzo małe. Sugeruje to, że do detekcji błędów, znacznie lepiej używać kryteriów zbudowanych w oparciu o wiele bitów, niż w oparciu o jeden bit. Aby zatem prawidłowo wyznaczyć wartości bitów, liczba próbek badanych powinna być stosunkowo duża. Dla klucza długości 128 bitów, powinna ona wynosić ok 20000, dla klucza długości 512 bitów - 350000, co znacznie zwiększa nakład pracy koniecznej do przeprowadzenia ataku. Korzystną rzeczą może być więc zastosowanie w takim przypadku specjalnej procedury korekcji błędów. Dla G.Hachez i F.Koeune⁶⁶ zastosowanie takiej procedury pozwoliło zredukować liczbę badanych próbek blisko dwukrotnie.

Atak bazujący na pomiarze czasu wykonywanych operacji możliwy do przeprowadzenia jest również dla innych szyfrów. Poniżej zostanie przedstawiony tego rodzaju atak na szyfr Rijndael, szyfr będący nowym standardem szyfrowania danych.

⁶⁶ G.Hachez, F.Koeune, J.J.Quisquater, Timing Attack: What can be achieved by a powerful adversary ?, UCL Crypto Group, 1999

Każdy cykl szyfru AES, z wyjątkiem ostatniego składa się z czterech transformacji: SubBytes, ShiftRows, MixColumn oraz AddRoundKey, opisanych w rozdziale piątym. Warto jedynie przypomnieć postać macierzową MixColumn

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

Wielomian $c(x)$ jest równy $c(x) = 03x^3 + 01x^2 + 01x + 02$. Mnożenie odbywa się zgodnie z zasadami mnożenia w $GF(2^8)$, czyli modulo nieredukowalny wielomian $x^8 + x^4 + x^3 + x + 1$ ($11B_x$). Redukcja współczynników iloczynu wielomianu jest modulo $x^4 + 1$. Patrząc jednak na wartości współczynników można zauważyć, że $03 = 02 + 01$, dlatego też w praktyce istnieje konieczność wykonania tylko jednego mnożenia - przez 02_x . Mnożenie przez 02_x w $GF(2^8)$ może być wykonywane w dwóch krokach:

1. przesunięcia bitów o jedną pozycję w lewo,
2. w przypadku gdy stopień wielomianu wynikowego przekracza 7 (występuje przeniesienie), sumy XOR wyniku z wartością $11B_x$ - redukcji.

W przypadku gdy implementacja szyfru zostanie wykonana, jak powyżej. wówczas czas wykonania powyższej operacji nie będzie stały. Kiedy wystąpi konieczność redukcji stopnia wyniku (sumy XOR z $11B_x$), operacja mnożenia będzie wykonywana dłużej.

Przy takiej implementacji, w transformacji MixColumn, bajty będą przynajmniej raz mnożone przez wartość 02 . Zdarzenie to oznaczone zostanie jako ξ . Jak zostało zauważone powyżej proces ten może trwać dłużej, jeśli pierwszy bit bajtu ma wartość równą jeden.

Atak składa się z dwóch faz:

1. Fazy inicjacji - zbudowania macierzy w której dla każdej możliwej wartości pierwszego bajtu klucza i dla N różnych możliwych wartości pierwszego bajtu tekstu jawnego⁶⁷, której elementy będą zależne od tego, czy mnożenie ξ wymaga redukcji (dodatkowej XOR), czy też nie:

$\forall 0 \leq i \leq 255, 0 \leq j < N, T[i][j] = 1$ jeśli pierwszy bit $\text{ByteSub}(i \text{ XOR } j) = 1$, 0 w przeciwnym razie

Każdy wiersz i odnosi się do możliwej wartości pierwszego bajtu klucza cyklu R_1 , każda kolumna j odnosi się do różnych wartości pierwszego bajtu tekstu jawnego.

⁶⁷ Możliwe jest zbudowanie tej macierzy dla $N=256$. Eksperymenty przeprowadzone przez autorów (zob. przypis 67) pokazały jednak, że atak może zostać przeprowadzony już dla $N=20$

2. Fazy pomiarów

W tym celu zbudowane zostanie N zbiorów składających się z M wiadomości, w których pierwszy bajt każdej wiadomości z danego podzbioru S_i jest równy wartości i , pozostałe bajty są losowe. Dzięki temu, dla każdej wiadomości z podzbioru S_i , mnożenie ξ będzie identyczne.

Niech wiadomości te zostaną zaszyfrowane i zmierzony zostanie czas obliczeń. Jeśli M jest dość duży, można się spodziewać, że średni czas obliczeń dla podzbioru S_i odzwierciedla czas mnożenia ξ . Tym sposobem, można orzec, z pewnym prawdopodobieństwem błędu, czy dla i w $[0, N-1]$ pierwszy z bitów operacji $\text{ByteSub}(i \text{ XOR } j)$ ma wartość równą jeden.

Aby wyznaczyć R_1 , należy wynik porównać z całą tablicą T . Wiersz, który w najlepszy sposób odzwierciedla przewidywane wartości, odnosi się do poprawnej wartości pierwszego bajtu R_1 . Pozostałe bajty pierwszego klucza cyklu można uzyskać w identyczny sposób. Specyficzny algorytm rozszerzenia klucza pozwala, dzięki znajomości pierwszych N_K bitów kluczy cykli, wygenerować pozostałe klucze dla cykli pozostałych.

Analiza ta, dokonana przez F.Koeune⁶⁸, pozwoliła dla Rijndaela operującego na bloku i kluczu długości 128 bitów, uzyskać, przy ilości 3000 próbek na bajt klucza, wszystkie bity klucza głównego, przy rozsądnym nakładzie kosztów. Atak ten był możliwy do przeprowadzenia, tylko i wyłącznie dzięki nierozsądnej implementacji algorytmu szyfrującego.

Powyższy atak stanowi kolejny dowód, oprócz doskonałego bezpieczeństwa i wysokiej odporności szyfru na teoretyczne ataki, o sile kryptosystemu decyduje również odpowiednia implementacja. Atak ten, zaprezentowany przez P. Kochera⁶⁹ może również być używany do ataków na implementacje DES (procedura rozszerzenia klucza), RC6, oraz różnego rodzaju systemy z kluczem publicznym.

⁶⁸ F.Koeune, J.J.Quisquater, A Timing Attack Against Rijndael, UCL Crypto Group, Technical Report CG-1999/1

⁶⁹ P.C.Kocher, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems, 1998

Rozdział 8

Implementacja programowa AES

8.1 Wstęp

Jedno z założeń projektowych niniejszej pracy stanowiła implementacja programowa szyfru Rijndael - AES (Advanced Encryption Standard). Dokonanie implementacji programowej każdego szyfru blokowego składa się z dwóch części:

- implementacji właściwego szyfru, tj. poszczególnych jego transformacji oraz algorytmu rozszerzenia klucza. Od jakości oraz sposobu tej implementacji, optymalizacji algorytmów i procedur, bezpośrednio zależy szybkość działania szyfru.
- implementacji interfejsu użytkownika oraz trybów pracy - w zależności od określonych wymagań, może to być szyfrowanie plików, szyfrowanie strumieniowe (tryb CFB), szyfrowanie pakietów itp.

Implementując algorytm, głównym założeniem była 100% zgodność z zaleceniami AES, tj:

- praca na wielkościach bloków 128, 192, 256 bitów.
- praca na kluczach o długościach 128, 192, 256 bitów.
- praca w trybach ECB oraz CBC

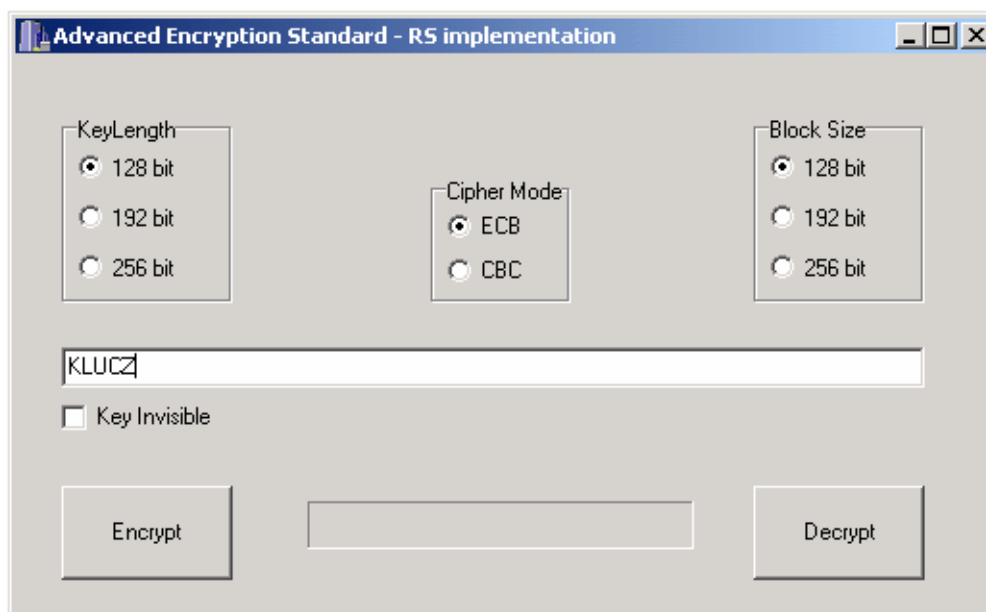
Wszystkie powyższe założenia zostały spełnione. Ograniczenie jakie nakłada program, stanowi jedynie minimalna wielkość pliku, na którym dokonywane będą operacje szyfrowania lub deszyfrowania - musi ona wynosić co najmniej długość bloku, a więc 16, 24 lub 32 bajty.

Dla plików których wielkość nie stanowi krotności długości bloku, zaimplementowana została omawiana wcześniej technika cipher stealing, zaś dla trybu CBC - stały wektor inicjujący.

Implementacji dokonano w środowisku Borland C++ Builder Enterprise Suite v5.0. Pisząc procedury i transformacje szyfru zachowano zgodność ze standardem ANSI C. Jedynie interfejs użytkownika wykorzystuje funkcje środowiska obiektowego Borlanda. Program został napisany na podstawie kodu odniesienia dostępnego w FIPS 197. Implementacja taka, mimo iż stanowi najwolniejszy przykład, jest najbardziej przejrzysta dla celów dydaktycznych. Zachowanie przejrzystości kodu, uniemożliwiło dokonania optymalizacji programu, omawianej w rozdziale 5. Listingi zostały zamieszczone w dodatku A.

8.2 Instrukcja obsługi programu

Program charakteryzuje się prostotą obsługi. Poniżej został przedstawiony ekran roboczy programu.



Rys. 8.2 Ekran roboczy programu

Chcąc zaszyfrować wybrany plik, należy wpierw zaznaczyć żadaną długość klucza - KeyLength, wielkość bloku -Block Size oraz tryb - Cipher Mode. Domyślnie program uruchamia się z wielkościami klucza i bloku równymi 128 bitów, oraz trybem ECB.

Następnie należy podać klucz. Klucz stanowić musi ciąg znaków w systemie heksadecymalnym, tj. 0-9, A-F. Dla klucza długości 128 bitów, liczba wprowadzonych znaków wyniesie więc 32, dla 192 bitów - 48, zaś dla 256 bitów - 64 znaki.

Chcąc zabezpieczyć klucz przed widokiem osób trzecich podczas wprowadzania, należy zaznaczyć pole Key Invisible.

Po naciśnięciu przycisku Encrypt (szyfrowanie) lub Decrypt (deszyfrowanie), przy założeniu, że parametry zostały wprowadzone poprawnie, otwarte zostanie okno dialogowe, w którym należy zaznaczyć plik do przetworzenia. Następnie otwarte zostanie okno dialogowe, w którym należy wprowadzić nazwę pliku wynikowego.

W trakcie szyfrowania/deszyfrowania, pomiędzy przyciskami pokazywany jest stopień zaawansowania wykonywanej operacji.