Wrocław, 28. listopada 2005

Poufność Informacji i Techniki Kodowania

Politechnika Wrocławska
Wydział Informatyki i Zarządzania
IV rok studiów

Algorytm IDEA
(seminarium)

Autor dokumentu: **STAWARZ Paweł**
Indeks: **125939**
Termin realizacji: **28. listopada 2005**

Prowadzący: **dr inż. Teresa Mendyk-Krajewska**

Krótka historia algorytmu IDEA

Algorytm szyfrowania IDEA był rozwijany od początku lat 90. we wspólnym projekcie Swiss Federal Institute of Technology oraz Ascom AG of Switzerland. Celem projektu było uzyskanie silnego algorytmu szyfrującego, który mógłby zastąpić algorytm DES rozwinięty w latach 70. w Stanach Zjednoczonych. Pierwsza wersja algorytmu IDEA została stworzona w 1990 roku przez Xuejia Lai oraz James Massey i została wówczas nazwana algorytmem PES (Proposed Encryption Standard). W 1991 roku Lai oraz Messey wzmocnili algorytm uodparniając go na kryptoanalizę różnicową i nazwali go IPES (Improved PES). W 1992 nazwa IPES została zmieniona na IDEA.

Ze względu na wysoki poziom bezpieczeństwa oraz szybkość działania, algorytm IDEA stał się pierwszym algorytmem szyfrowania wykorzystanym w stworzonym w 1991 roku przez Phill`a Zimmermann`a produkcie PGP (*Pretty Good Privacy*) służącym do szyfrowania wiadomości tekstowych oraz plików (właściwie był on już drugim algorytmem, ale wcześniejszy, stworzony przez samego Zimmerman`a 'Bass-O-Matic' bardzo szybko okazał się być niebezpiecznym w użyciu).

Algorytm IDEA do dzisiaj stosowany jest w wielu rozwiązaniach dotyczących bezpieczeństwa (szyfrowania danych) – np. w bankowości, czy też płatnej telewizji – wszędzie tam, gdzie istnieje potrzeba ochrony cennych zawartości.

Współczesna IDEA

Charakterystyka

IDEA jest nazwą wypróbowanego, bezpiecznego i uniwersalnie stosowanego opatentowanego symetrycznego algorytmu szyfrowania blokowego, który pozwala efektywnie chronić przesyłane bądź przechowywane dane przed nieautoryzowanym dostępem osób trzecich. Najważniejszymi kryteriami rozwijania algorytmu IDEA były bardzo wysokie wymagania dotyczące bezpieczeństwa idące wraz z łatwością implementacji sprzętowej oraz programowej zapewniającej szybkie wykonanie. Wysoki poziom bezpieczeństwa zapewniony jest m.in. przez 128-bitowy klucz.

Algorytm szyfrowania IDEA:

- **dostarcza wysokiego poziomu bezpieczeństwo;**
- **jest dobrze sprecyzowany i łatwy do zrozumienia;**
- **nadaje się do stosowania w szerokim zakresie aplikacji;**
- **może być z łatwości implementowany w elektronicznych komponentach (VLSI Chip – *Very Large Scale Integration*)**
- **jest dostępny dla wszystkich;**
- **jest chroniony patentami i prawami autorskimi.**

Wydajność

W zależności od sposobu rozwiązania, wydajność algorytmu IDEA waha się w przedziale od kilkuset kbit/sec, do nieco powyżej 3.6 Gbit/sec. Najwolniejszym rozwiązaniem jest wykorzystanie systemów wbudowanych osiągających wydajność rzędu 200 kbit/sec do 2 Mbit/sec. Nieco lepszą wydajność uzyskuje się przy wykorzystaniu języków wysokiego poziomu – od 1Mbit/sec do 100 Mbit/sec. Znaczny wzrost wydajności osiągany jest przy wykorzystaniu języka assemblera – od 100 Mbit/sec do 2 Gbit/sec. Najszybsze rozwiązania oparte są jednak na implementacji sprzętowej, której wydajność przekracza nawet 3.6 Gbit/sec.

Bezpieczeństwo

Od wielu lat społeczność kryptograficzna z całego świata testowała algorytm IDEA. Testy te wykazały, że jest on odporny na kryptoanalizę różnicową. W 1994 roku odkryto jedynie niewielką klasę słabych kluczy, ale na tyle niewielką, że w żaden specjalny sposób się ich nie unika. Natomiast przy wykorzystaniu współczesnej mocy

obliczeniowej komputerów oraz wyczerpującego przeszukiwania kluczy z przestrzeni 2^{128} kluczy, odnalezienie właściwego klucza mogłoby zająć nawet 10 bilionów lat. Najlepszą znaną metodą kryptoanalityczną łamania szyfru powstałego w wyniku zastosowania algorytmu IDEA jest wypróbowany w 2003 roku *collision attack*, który potrzebując 2^{24} wybranych tekstów jawnych łamał algorytm IDEA zredukowany do 5 rund ze złożonością 2^{126}.

Patenty

Algorytm IDEA jest chroniony międzynarodowym prawem autorskim i dodatkowo został opatentowany w Europie (1993), w Stanach Zjednoczonych (1993) oraz w Japonii (2001). Patent algorytmu IDEA, jak również jego nazwa są własnością firmy Ascom AG, Switzerland. Ascom AG przekazała wszelkie prawa dystrybucji firmie MediaCrypt AG. W celu rozpowszechnienia silnego algorytmu szyfrowania, MediaCrypt pozwala na darmowe jego wykorzystanie tylko i wyłącznie w celach niekomercyjnych. W przypadku wykorzystania algorytmu IDEA w celach komercyjnych wymagane jest uzyskanie licencji od MediaCrypt.

Licencje

Firmy chcące wykorzystać algorytm IDEA w komercyjnych rozwiązaniach mogą otrzymać licencję za drobną odpłatą. MediaCrypt oferuje kilka rodzajów licencji: dla firm chcących wykorzystać algorytm IDEA przy tworzeniu własnego produktu, dla końcowego użytkownika chcącego korzystać z produktu zawierającego algorytm IDEA, oraz dla niekomercyjnych zastosowań.

Profil MediaCrypt

MediaCrypt jest przedsiębiorstwem zajmującym się zagadnieniami bezpieczeństwa związanymi głównie z komunikacją między komputerami, na przykład wymianą danych, transakcjami finansowymi czy też transmisjami Internetowymi. Współwłaścicielami MediaCrypt, znajdującego się w Szwajcarii w Zurichu, są firmy Ascom oraz Kudelski Group. Mediacrypt rozwija własne oprogramowanie szyfrujące oparte na algorytmie IDEA (*International Data Encryption Algorithm*).

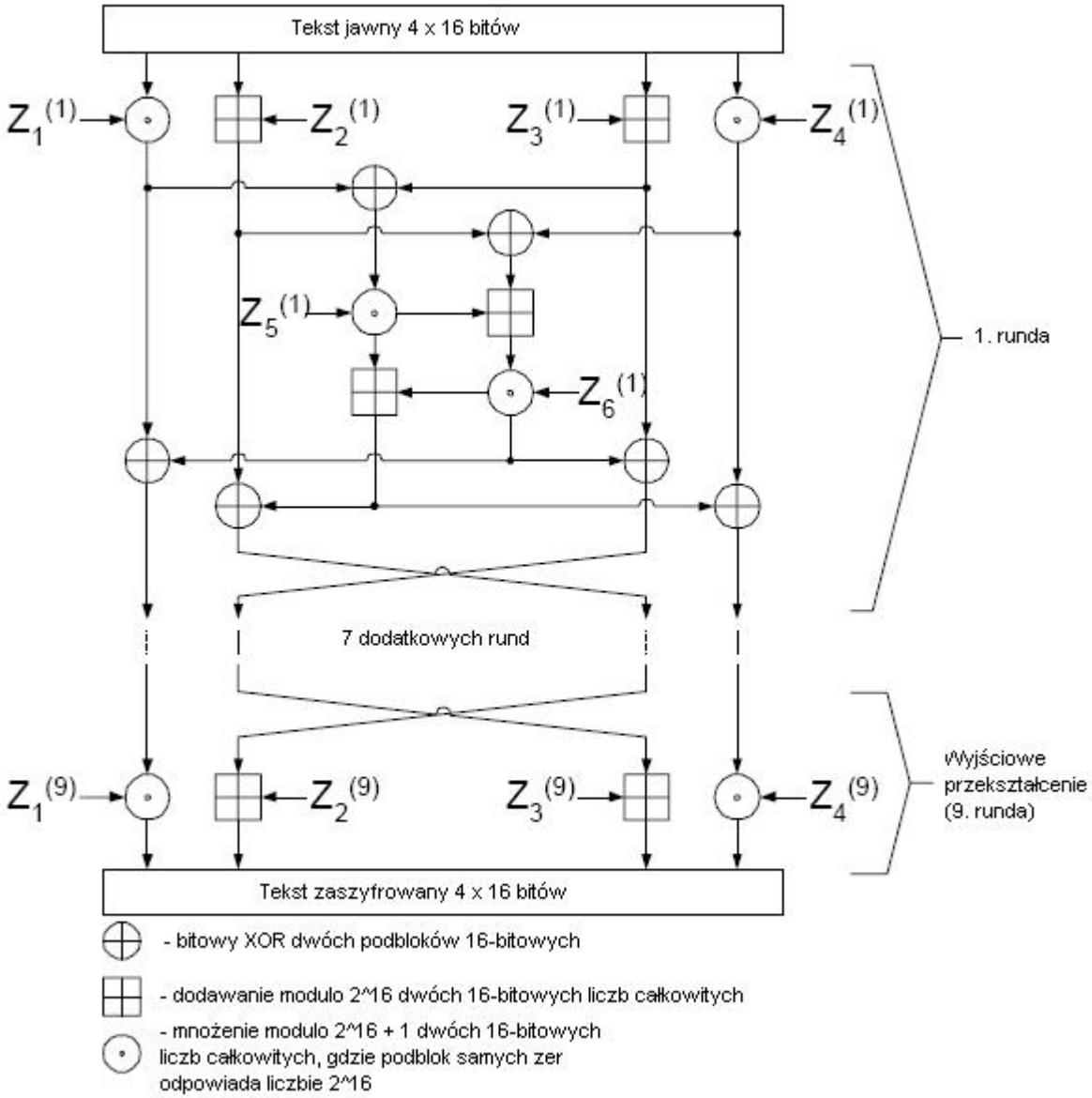
Opis techniczny algorytmu IDEA

Informacje podstawowe

Symetryczny szyfr blokowy IDEA wykorzystuje 128-bitowy klucz i operuje 64-bitowymi blokami tekstu jawnego i zaszyfrowanego. Zasadniczą innowacją w realizacji tego algorytmu jest zastosowanie operacji z trzech różnych grup algebraicznych. Zrezygnowano całkowicie ze stosowania permutacji oraz S-box`ów, tj. w przypadku algorytmu DES. Struktura algorytmu została tak zaprojektowana, aby proces szyfrowania był identyczny z procesem deszyfrowania. Wyjątek stanowi jedynie wykorzystanie różnych podkluczy.

Struktura szyfrowania

Struktura szyfrowania algorytmu IDEA opisana w reprezentacji funkcyjnej:



Proces szyfrowania składa się z ośmiu identycznych kroków szyfrowania, nazywanych również rundami szyfrowania, po których następuje wyjściowa transformacja. Struktura pierwszej rundy pokazana jest szczegółowo.

Ponieważ wszystkie operacje algebraiczne wykorzystywane w procesie szyfrowania operują na liczbach 16-bitowych, więc na wstępie 64-bitowy blok tekstu jawnego dzielony jest na cztery 16-bitowe podbloki. Dla każdej rundy szyfrowania (z wyjątkiem ostatniej rundy) ze 128-bitowego klucza generowanych jest sześć 16-bitowych podkluczy. W sumie w całym procesie szyfrowania ze 128-bitowego klucza generowane są 52 różne podklucze – sześć podkluczy dla każdej z ośmiu rund oraz cztery podklucze dla wyjściowej transformacji (Na przykładowym schemacie znajduje się 10 z 52 różnych 16-bitowych podkluczy).

W pierwszej rundzie szyfrowania, pierwsze cztery 16-bitowe podklucze łączone są z dwoma 16-bitowymi blokami tekstu jawnego przy wykorzystaniu dodawania modulo 2^{16} **oraz z kolejnymi dwoma 16-bitowymi blokami tekstu jawnego przy wykorzystaniu mnożenia modulo** $2^{16}+1$**. Wyniki przekazywane są dalej (jak na rysunku), gdzie w połączeniu z dwoma kolejnymi 16-bitowymi podkluczami wchodzą w działanie bitowego XOR`a. Na koniec pierwszej rundy szyfrowania wyprodukowane są cztery 16-bitowe wartości, które następnie w częściowo zmienionej kolejności wykorzystane są jako wejście do drugiej rundy szyfrowania. Proces odnoszący się do pierwszej rundy jest powtarzany w każdej z 7 kolejnych rund, przy czym dla każdej rundy wykorzystywane są różne podklucze. W wyjściowej transformacji cztery 16-bitowe wartości wygenerowane w 8 rundzie szyfrowania łączone są z ostatnimi czterema podkluczami przy wykorzystaniu dodawania modulo** 2^{16} **oraz mnożenia modulo** $2^{16}+1$**, uzyskując w rezultacie cztery 16-bitowe bloki tekstu zaszyfrowanego. Jako ciekawostkę można zauważyć, że w żadnym miejscu procesu szyfrowania nie wykorzystuje się pod rząd tej samej grupy algebraicznej. Dodatkową szczególną cechą mnożenia dwóch 16-bitowych podbloków modulo** $2^{16}+1$ **jest fakt, że podblok zawierający 16 zerowych bitów nie jest interpretowany jako 0, lecz jako** 2^{16}**.**

Struktura deszyfrowania

Proces deszyfrowania tekstu zaszyfrowanego jest zasadniczo taki sam jak w przypadku procesu szyfrowania tekstu jawnego i w związku z tym wcześniej przedstawiony graf przedstawiający strukturę szyfrowania jest również prawdziwy dla opisu struktury deszyfrowania. Jedyną różnicą w stosunku do procesu szyfrowania jest inny sposób generowania 52 podkluczy. Mówiąc ściślej każdy 16-bitowy podklucz wykorzystywany przy deszyfrowaniu jest odwrotnością podklucza wykorzystywanego przy szyfrowaniu w odniesieniu do zastosowanej grupy algebraicznej. Dodatkowo, podczas deszyfrowania, podklucze muszą być wykorzystane w odwrotnej kolejności, aby odwrócić proces szyfrowania.

Generowanie podkluczy

52 16-bitowe podklucze generowane są ze 128-bitowego klucza w następujący sposób:

- **128-bitowy klucz dzielony jest na osiem 16-bitowych podbloków, które bezpośrednio wykorzystywane są jako pierwsze osiem podkluczy;**
- **128-bitowy klucz jest następnie cyklicznie przesuwany w lewą stronę o 25 pozycji, tak, że powstały 128-bitowy blok ponownie dzielony jest na osiem 16-bitowych podbloków, które ponownie bezpośrednio wykorzystywane są jako kolejnych osiem podkluczy;**
- **procedura cyklicznego przesunięcia opisana wyżej jest powtarzana tak długo, aż zostaną wygenerowane wszystkie potrzebne 16-bitowe podklucze.**

Podklucze wykorzystywane do szyfrowania i deszyfrowania w określonych rundach pokazane są w poniższych tabelach:

Podklucze w szyfrowaniu

Round 1	$Z_1^{(1)}\ Z_2^{(1)}\ Z_3^{(1)}\ Z_4^{(1)}\ Z_5^{(1)}\ Z_6^{(1)}$
Round 2	$Z_1^{(2)}\ Z_2^{(2)}\ Z_3^{(2)}\ Z_4^{(2)}\ Z_5^{(2)}\ Z_6^{(2)}$
Round 3	$Z_1^{(3)}\ Z_2^{(3)}\ Z_3^{(3)}\ Z_4^{(3)}\ Z_5^{(3)}\ Z_6^{(3)}$
Round 4	$Z_1^{(4)}\ Z_2^{(4)}\ Z_3^{(4)}\ Z_4^{(4)}\ Z_5^{(4)}\ Z_6^{(4)}$
Round 5	$Z_1^{(5)}\ Z_2^{(5)}\ Z_3^{(5)}\ Z_4^{(5)}\ Z_5^{(5)}\ Z_6^{(5)}$
Round 6	$Z_1^{(6)}\ Z_2^{(6)}\ Z_3^{(6)}\ Z_4^{(6)}\ Z_5^{(6)}\ Z_6^{(6)}$
Round 7	$Z_1^{(7)}\ Z_2^{(7)}\ Z_3^{(7)}\ Z_4^{(7)}\ Z_5^{(7)}\ Z_6^{(7)}$
Round 8	$Z_1^{(8)}\ Z_2^{(8)}\ Z_3^{(8)}\ Z_4^{(8)}\ Z_5^{(8)}\ Z_6^{(8)}$
Output Transform	$Z_1^{(9)}\ Z_2^{(9)}\ Z_3^{(9)}\ Z_4^{(9)}$

Podklucze w deszyfrowaniu

Round 1	$Z_1^{(9)-1}\ -Z_2^{(9)}\ -Z_3^{(9)}\ Z_4^{(9)-1}\ Z_5^{(8)}\ Z_6^{(8)}$
Round 2	$Z_1^{(8)-1}\ -Z_3^{(8)}\ -Z_2^{(8)}\ Z_4^{(8)-1}\ Z_5^{(7)}\ Z_6^{(7)}$
Round 3	$Z_1^{(7)-1}\ -Z_3^{(7)}\ -Z_2^{(7)}\ Z_4^{(7)-1}\ Z_5^{(6)}\ Z_6^{(6)}$
Round 4	$Z_1^{(6)-1}\ -Z_3^{(6)}\ -Z_2^{(6)}\ Z_4^{(6)-1}\ Z_5^{(5)}\ Z_6^{(5)}$
Round 5	$Z_1^{(5)-1}\ -Z_3^{(5)}\ -Z_2^{(5)}\ Z_4^{(5)-1}\ Z_5^{(4)}\ Z_6^{(4)}$
Round 6	$Z_1^{(4)-1}\ -Z_3^{(4)}\ -Z_2^{(4)}\ Z_4^{(4)-1}\ Z_5^{(3)}\ Z_6^{(3)}$
Round 7	$Z_1^{(3)-1}\ -Z_3^{(3)}\ -Z_2^{(3)}\ Z_4^{(3)-1}\ Z_5^{(2)}\ Z_6^{(2)}$
Round 8	$Z_1^{(2)-1}\ -Z_3^{(2)}\ -Z_2^{(2)}\ Z_4^{(2)-1}\ Z_5^{(1)}\ Z_6^{(1)}$
Output Transform	$Z_1^{(1)-1}\ -Z_2^{(1)}\ -Z_3^{(1)}\ Z_4^{(1)-1}$

Przykład szyfrowania

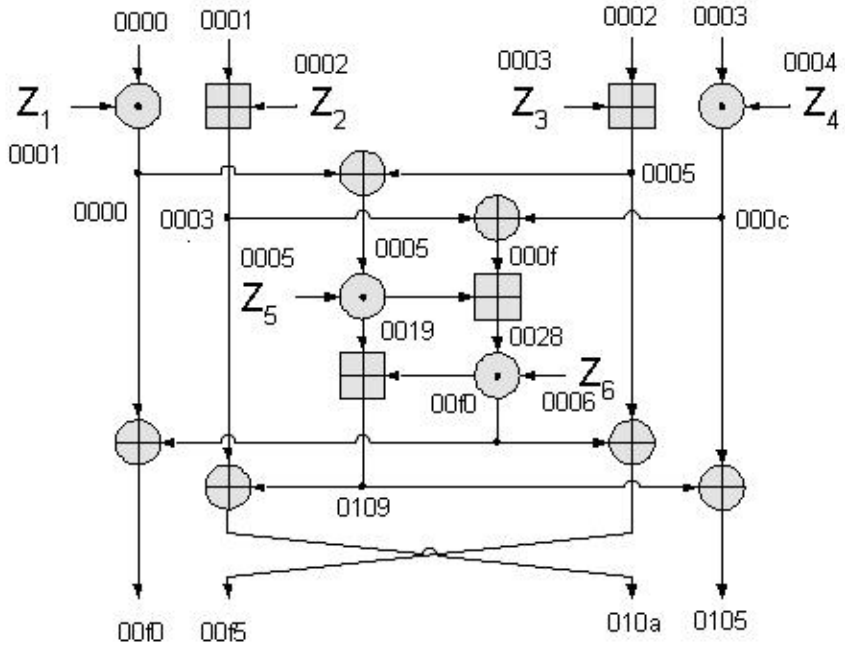
Poniżej znajduje się przykładowy proces szyfrowania 64-bitowego tekstu jawnego M przy wykorzystaniu 128-bitowego klucza K. Wszystkie wartości zostały przedstawione w postaci hexadecymalnej:

128-bit key $K = (1,2,3,4,5,6,7,8)$							64-bit plaintext $M = (0,1,2,3)$			
r	$K_1^{(r)}$	$K_2^{(r)}$	$K_3^{(r)}$	$K_4^{(r)}$	$K_5^{(r)}$	$K_6^{(r)}$	X_1	X_2	X_3	X_4
1	0001	0002	0003	0004	0005	0006	00f0	00f5	010a	0105
2	0007	0008	0400	0600	0800	0a00	222f	21b5	f45e	e959
3	0c00	0e00	1000	0200	0010	0014	0f86	39be	8ee8	1173
4	0018	001c	0020	0004	0008	000c	57df	ac58	c65b	ba4d
5	2800	3000	3800	4000	0800	1000	8e81	ba9c	f77f	3a4a
6	1800	2000	0070	0080	0010	0020	6942	9409	e21b	1c64
7	0030	0040	0050	0060	0000	2000	99d0	c7f6	5331	620e
8	4000	6000	8000	a000	c000	e001	0a24	0098	ec6b	4925
9	0080	00c0	0100	0140	—	—	11fb	ed2b	0198	6de5

Odpowiednia deszyfracja powstałego tekstu zaszyfrowanego C:

$K = (1,2,3,4,5,6,7,8)$							$C = (11fb,ed2b,0198,6de5)$			
r	$K'^{(r)}_1$	$K'^{(r)}_2$	$K'^{(r)}_3$	$K'^{(r)}_4$	$K'^{(r)}_5$	$K'^{(r)}_6$	X_1	X_2	X_3	X_4
1	fe01	ff40	ff00	659a	c000	e001	d98d	d331	27f6	82b8
2	fffd	8000	a000	cccc	0000	2000	bc4d	e26b	9449	a576
3	a556	ffb0	ffc0	52ab	0010	0020	0aa4	f7ef	da9c	24e3
4	554b	ff90	e000	fe01	0800	1000	ca46	fe5b	dc58	116d
5	332d	c800	d000	fffd	0008	000c	748f	8f08	39da	45cc
6	4aab	ffe0	ffe4	c001	0010	0014	3266	045e	2fb5	b02e
7	aa96	f000	f200	ff81	0800	0a00	0690	050a	00fd	1dfa
8	4925	fc00	fff8	552b	0005	0006	0000	0005	0003	000c
9	0001	fffe	fffd	c001	—	—	0000	0001	0002	0003

W celu przećwiczenia i lepszego zrozumienia algorytmu IDEA warto sobie przeanalizować chociażby pierwszą rundę procesu szyfrowania:



Oczywiście dalsza część szyfrowania przebiega analogicznie z zastosowaniem odpowiednio generowanych kolejnych podkluczy (wykorzystując wspomniane wcześniej bitowe przesunięcie cykliczne klucza w lewo o 25 bitów).

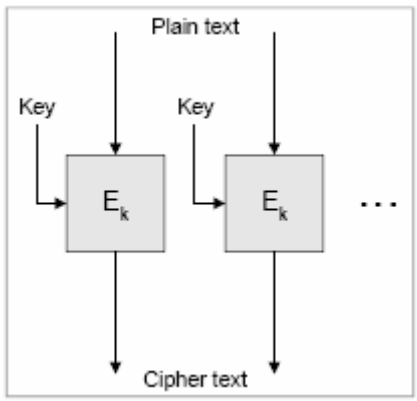
Tryby kodowania

Informacje wstępne

IDEA wspiera wszystkie tryby kodowania opisane przez NIST (National Institute of Standards and Technology) w publikacji FIPS 81 (USA Federal Information Processing Standard). Szyfr blokowy szyfruje i deszyfruje tekst w blokach stałej długości bitów – najczęściej 64-bitowe oraz 128-bitowe. Tekst jawny przekraczający określoną długość dzielony jest po prostu na bloki, tak, aby każdy z nich można było oddzielnie zaszyfrować. Najbardziej znane metody dzielenia tekstu jawnego na bloki to: Electronic Code Book (ECB), Cipher Blok Chaining (CBC), Cipher Feedback (CFB) oraz Output Feedback (OFB).

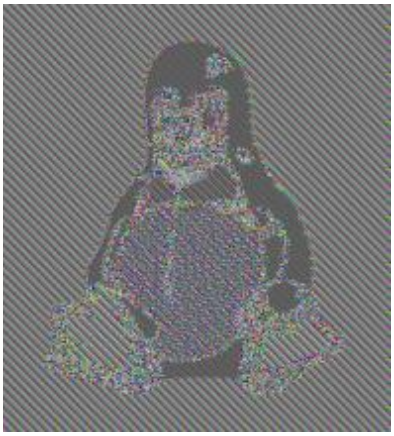
Electronic Code Book (ECB)

Charakterystyczną cechą tego trybu kodowania jest fakt, że każdy blok tekstu jawnego ma zdefiniowaną odpowiadającą mu wartość tekstu zaszyfrowanego i na odwrót. Innymi słowy ta sama wartość tekstu jawnego zawsze zwróci tę samą wartość tekstu zaszyfrowanego. Tryb ECB polega na najprostszym podziale całego tekstu jawnego na fragmenty odpowiadające określonej długości bloków, które następnie szyfrowane są niezależnie od siebie. W szczególności, tryb ECB wspiera szyfrowanie wykorzystujące różne klucze dla różnego typu bloków. Niestety tryb ECB nie zapewnia wysokiego poziomu bezpieczeństwa. W przypadku względnie dużych wiadomości tekstowych, pewne słowa oraz

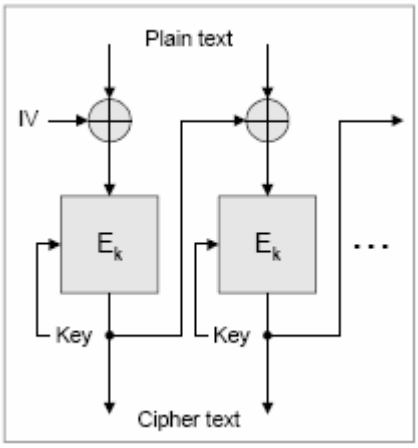


$$M_i = D_k(C_i) \quad C_i = E_k(M_i)$$

wyrażenia często używane mogą spowodować, iż w tekście zaszyfrowanym pojawią się powtarzające się części podbloków, które z kolei mogą stać się podstawą do łamania szyfru przy pomocy ataku słownikowego. Problem ten można chociażby rozwiązać dodając losowe znaki w określonych miejscach bloku, ale mimo wszystko ze względów bezpieczeństwa nie zaleca się stosować tego trybu. Warto jednak zauważyć, iż jakikolwiek błąd w bloku tekstu zaszyfrowanego wpływa tylko i wyłącznie na deszyfrację tego bloku. Przykład obrazu sławnego pingwinka wraz z jego zaszyfrowaniem w trybie ECB (nie podano samej metody szyfrowania):



Cipher Block Chaining (CBC)

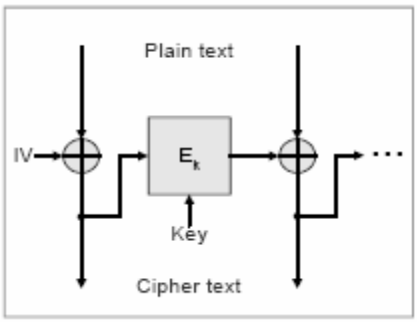


Kluczową cechą trybu kodowania CBC jest wykorzystanie łańcuchowego mechanizmu, który powoduje, że deszyfrowanie bloku tekstu zaszyfrowanego zależy od wszystkich poprzedzających go bloków tekstu zaszyfrowanego. Warto przy tym zauważyć, że błąd pojedynczego bitu w bloku tekstu zaszyfrowanego wpływa na deszyfrowanie reszty. Wymieszanie kolejności bloków tekstu zaszyfrowanego powoduje zepsucie całej deszyfracji. Działanie trybu CBC opiera się na szyfrowaniu zXORowanego bloku tekstu jawnego z poprzednim blokiem tekstu zaszyfrowanego. Przy czym do zaszyfrowania pierwszego bloku tekstu jawnego wykorzystuje się losowy ciąg bitów dołączany do wiadomości, tzw. wektor inicjujący (IV). Identyczne bloki tekstu zaszyfrowanego mogą pojawić się tylko i wyłącznie w przypadku gdy szyfrowany jest ten sam blok tekstu jawnego przy użyciu tego samego klucza oraz wektora inicjującego. Zaletą trybu CBC w stosunku do trybu EBC jest ukrycie wzorców tekstu jawnego. W przypadku idealnym, wektor inicjujący powinien być różny dla dowolnych dwóch wiadomości szyfrowanych tym samym kluczem.

$$M_i = D_k(C_i) \oplus C_{i-1}$$
$$C_i = E_k(M_i \oplus C_{i-1})$$

Ciphertext Feedback (CFB)

Podobnie jak w trybie CBC, tryb CFB wykorzystuje wektor inicjujący (IV) pewnej określonej długości. W trybie CFB, każdy blok tekstu zaszyfrowanego powstaje w wyniku zXORowania bieżącego bloku tekstu jawnego z zaszyfrowanym poprzednim blokiem tekstu zaszyfrowanego. W tym przypadku tryb CFB, podobnie jak w przypadku trybu CBC, ukrywa wzorce tekstu jawnego. Podobnie również wpływa zmiana wektora inicjującego na ten sam blok tekstu jawnego, która powoduje zwracanie różnych wartości wyjściowych. Deszyfracja jednego bloku tekstu zaszyfrowanego zależy od deszyfracji bloku poprzedniego.

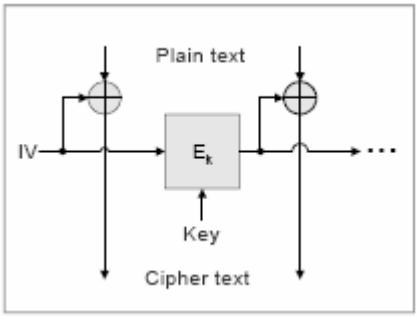


$$M_i = C_i \oplus E_k(C_{i-1})$$

$$C_i = E_k(C_{i-1}) \oplus M_i$$

Output Feedback (OFB)

Mechanizm trybu OFB jest zbliżony do mechanizmu trybu CFB, z jedyną różnicą, że sprzężone jest wyjście funkcji szyfrującej, a nie tak jak w trybie CFB blok tekstu zaszyfrowanego. W związku z tym każda wartość funkcji XOR dla bieżącego bloku tekstu jawnego generowana jest niezależnie od innych bloków tekstu jawnego oraz zaszyfrowanego. Tryb ten, dzięki wykluczeniu zależności łańcuchowej, wykorzystywany jest w przypadku, gdy niedopuszczalna jest dalsza propagacja błędu. Podobnie jak w przypadku trybu CFB, tryb OFB wykorzystuje wektor inicjujący, którego zmiana dla tego samego bloku tekstu jawnego wpływa na zmianę tekstu zaszyfrowanego.



$$C_i = M_i \oplus O_i \quad O_i = E_k(O_{i-1})$$

Przykład zastosowania różnych typów kodowania

Obraz oryginalny	Szyfrowanie w trybie ECB	Szyfrowanie w trybie CBC
		

Literatura

1. **http://www.mediacrypt.com – oficjalna strona firmy MediaCrypt, na której znajdują się m.in. szczegółowe opisy działania algorytmu szyfrowania IDEA – referat w większej części został zrealizowany na podstawie zaczerpniętych stąd dokumentów oraz obrazków.**
2. **http://en.wikipedia.org – wolna encyklopedia Wikipedia, w której można znaleźć między innymi opis trybów kodowania, podział szyfrów kryptograficznych, a także opis samego algorytmu IDEA**
3. **A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, *Chapter 7.6 IDEA*, CRC Press, 1996 – szczegółowy opis algorytmu szyfrowania IDEA wraz z przykładem wykorzystanym w referacie.**