Kryptografia i bezpieczeństwo danych
- SZYFRY BLOKOWE II

Sławomir Samolej
ssamolej.kia.prz.edu.pl
ssamolej@prz.edu.pl

1

Szyfr AES

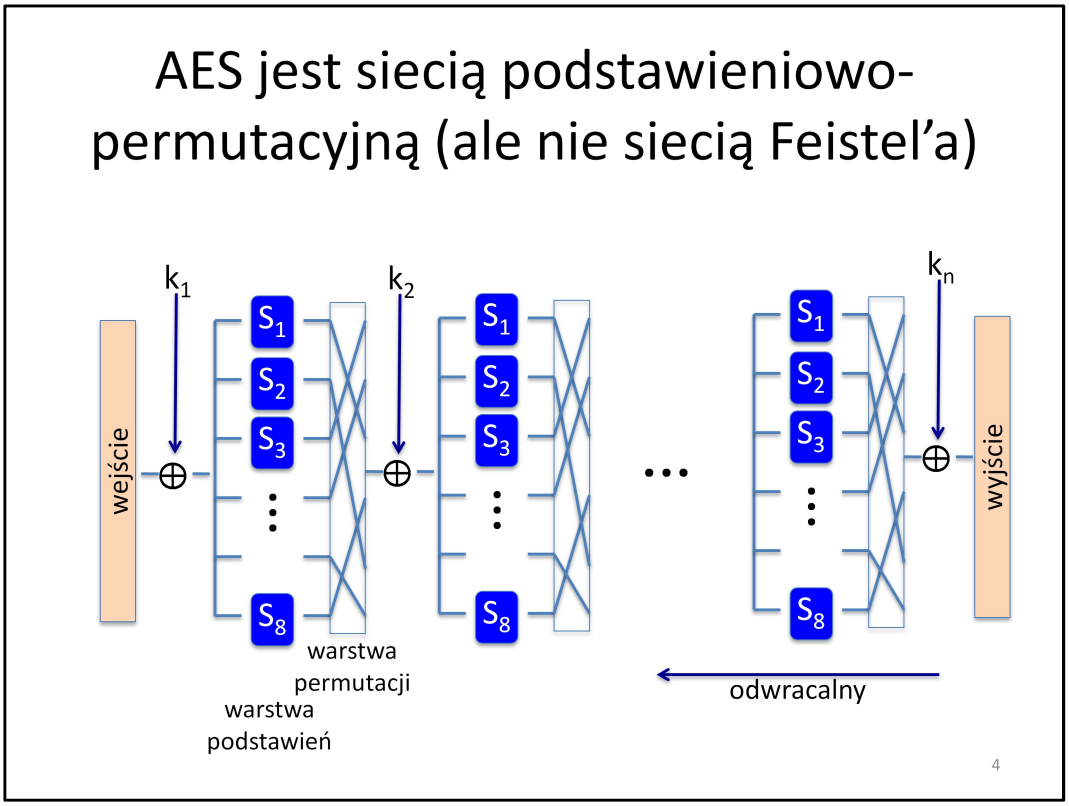
Historia szyfru AES

- 1997: NIST (National Institute of Standards and Technology) ogłasza konkurs na nowy standard szyfrowania blokowego
- 1998: 15 zgłoszeń.
- 1999: NIST wybiera 5 finalistów
- 2000: NIST wybiera szyfr Rijndael jako AES (Advanced Encryption Standard) (zaprojektowany w Belgii)

Rozmiary kluczy: 128, 192, 256 bitów.
Rozmiary bloków: 128 bitów

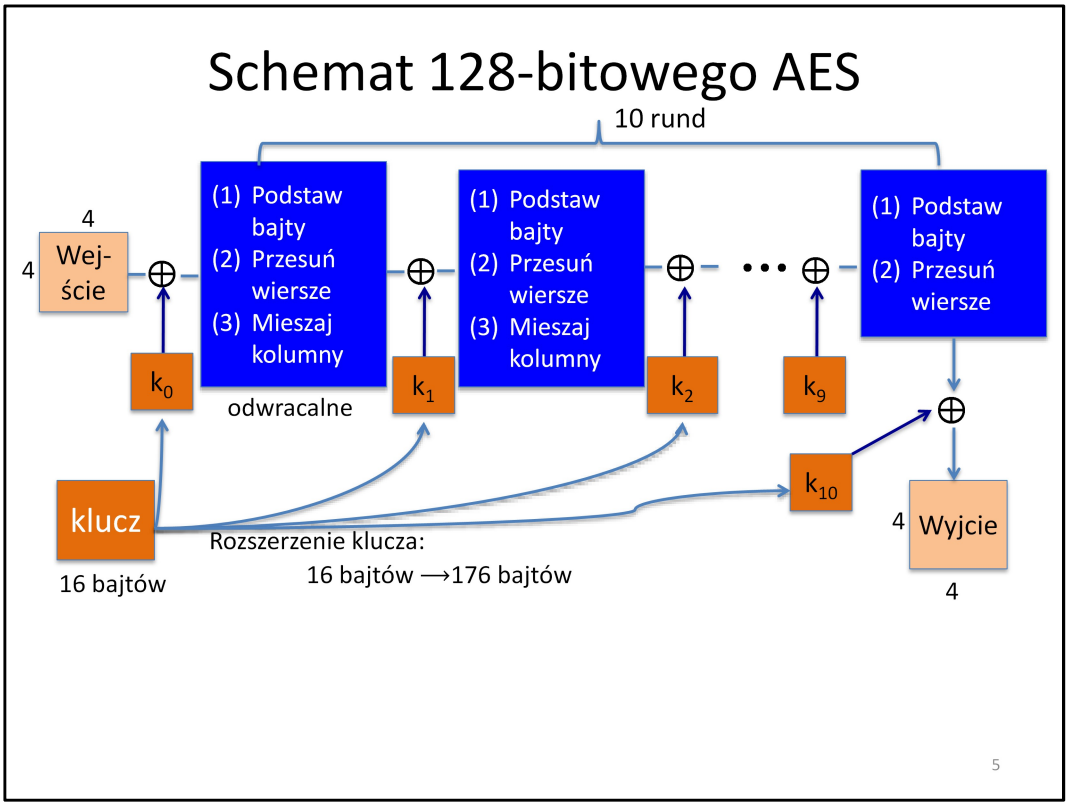
3

Przez lata stosowania DES i 3DES okazało się, że te rozwiązania nie są zbytnio kompatybilne ze współczesnym sprzętem i są zbyt wolne. Organizacja NIST (National Institute of Standards and Technology) w USA rozpoczęła w 1997 roku proces poszukiwania nowego standardu szyfrowania blokowego. W 1997 został rozpisany konkurs, w 1998 roku nadesłano 15 zgłoszeń. Ostatecznie wybrano w roku szyfr Rijndael. Rozmiar bloku szyfru wynosi 128 bitów. Jest zaprojektowany na trzy możliwe długości klucza: 128, 192 i 256 bitów. Założenie jest takie, że im dłuższy klucz, tym bardziej bezpieczny szyfr ale również wolniej działa algorytm.



AES jest siecią podstawieniowo-permutacyjną. Nie jest to sieć Feistel'a. W sieci Feistel'a połowa bitów na rundę nie była zmieniana, tutaj w każdej rundzie zmieniane są wszystkie bity. Sieć podstawieniowo-peruntacyjna działa w następujący sposób.

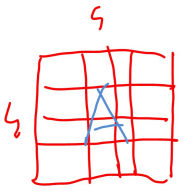
Blok danych do szyfrowania jest szyfrowany kluczem rundy k_1 (wykonywany jest XOR na kluczu i bloku danych). Następnie w wyznaczonych blokach danych następuje podstawienie zgodnie z regułami zawartymi w **tablicach podstawień**. W warstwie permutacji bity są przetasowywane (wykonywana jest na nich permutacja). W kolejnych rundach wyjściowe bity z poprzedniej rundy są szyfrowane z kolejnym kluczem rundy, podstawiane w blokach i permutowane.
Ostatnia runda polega tylko na szyfrowaniu z kluczem ostatniej rundy k_n. Pokazana konstrukcja kryptograficzna jest odwracalna. Deszyfrowanie polega na wzięciu szyfrogramu i wykonanie operacji opisanych w sieci w odwrotnej kolejności. Opisane permutacje i bloki, w których następują podstawienia muszą być odwracalne (XOR z kluczem też jest oczywiście odwracalne).



128-bitowy ciąg (16 bajtów) jest zapisywany w postaci dwuwymiarowej tablicy 4 x 4 bajty. Każda komórka tablicy zawiera 1 bit. Na tablicy bitów jest wykonywana pierwsza runda szyfrowania: XOR z kluczem rundy, a następnie pewna funkcja, która zawiera podstawienia, permutacje i inne operacje na stanie danych. Funkcje muszą być odwracalne aby szyfrogram był do odszyfrowania. Potem wykonywane są kolejne rundy rozpoczynające się od zaszyfrowania bloku (XOR) kolejnym kluczem rundy itd. Całość szyfrowania zawiera 10 rund. Proszę zwrócić uwagę na ostatnią rundę, w której nie następuje mieszanie kolumn. Ostatnią fazą szyfrowania jest XOR z kluczem 11 rundy (k_{10}). W każdej fazie szyfrowania przekształcenia dotyczą tej samej tablicy bitów 4 na 4 bajty. Klucze rund pochodzą z rozszerzenia początkowego 16 bajtowego klucza (128 bitów). Ten klucz jest rozszerzany do długości 176 bajtów (11 rund po 16 bajtów).

Funkcje rundy

- **Podstawianie bajtów**: 1 bajtowy S-box. 256 bajtowa tablica
  (łatwa do przetworzenia)



$\forall i,j : A[i,j] \leftarrow S[A[i,j]]$

Modyfikacja tablicy następuje przez podmienienie bajtów na podstawie pomocniczej 256 bajtowej tablicy

6

Zasada działania funkcji rund zostanie przedstawiona tylko na ogólnym poziomie. Osoby zainteresowane szczegółami proszę o sięgnięcie do materiałów w Internecie.

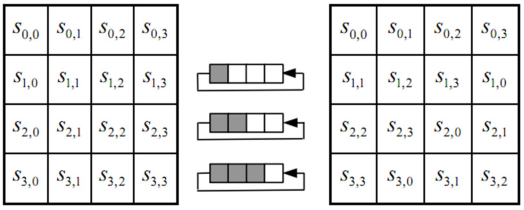
Podstawianie bajtów ma postać pojedynczego S-box'a. Jest dana tablica pomocnicza o rozmiarze 256 bajtów. Na podstawie jej stanu następuje podmiana bajtów w tablicy wejściowej.

Funkcje rundy

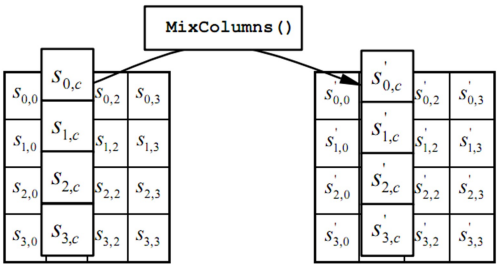
- **Podstawianie bajtów**: 1 bajtowy S-box. 256 bajtowa tablica

  (łatwa do przetworzenia)

- **Przesuwanie wierszy**:



- **Mieszanie kolumn**:



7

Drugi etap to permutacja polegająca na cyklicznym przesuwaniu bajtów w wierszach. Pierwszy wiersz nie jest przesuwany, drugi przesuwany jest o jeden bajt, trzeci o 2 bajty, a czwarty o trzy bajty (oczywiście bajty , które zostają „wypchnięte” z wiersza są wstawiane z powrotem na jego początek – tak działa cykliczne przesuwanie).

Trzeci etap to mieszanie kolumn. Jest to liniowa transformacja wykonana na każdej z kolumn. Istnieje tablica, przez którą przemnaża się daną kolumnę i otrzymuje się jej nową postać.

Rozmiar kodu/zagadnienia wydajnościowe

	Rozmiar kodu	Wydajność
Wstępnie obliczone funkcje rund (24KB lub 4KB)	największy	najszybsze: Przeglądanie tablic i operacje XOR
Wstępnie obliczony S-box (256 bajtów)	mniejszy	wolniejsze
Rozwiązanie bez obliczeń wstępnych	najmniejszy	najwolniejsze

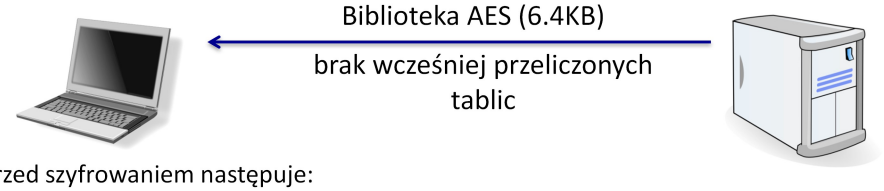
8

Warto zwrócić uwagę, że przesuwanie wierszy i mieszanie kolumn jest bardzo proste do implementacji programowej i intuicyjnie szybkie w przetwarzaniu. Implementacja może zajmować mniej niż 256 bajtów kodu. Można nawet przygotować kod, który w momencie uruchamiania szyfrowania generuje tablice służące do przetwarzania. Taka implementacja kodu jest najwolniejsza, ale możliwa do uruchomienia na urządzeniach o ograniczonej pamięci, np. kartach chipowych.

Jeśli algorytm jest uruchamiany na komputerach o większej ilości pamięci warto rozważyć wstępne przygotowanie tablic już w pamięci programu. Wtedy obliczanie algorytmu sprowadza się do szybkiego przeglądania tablic i wykonywanie operacji xor na ciągach bitowych. Implementacja algorytmu staje się wtedy znacznie szybsza, kosztem większego zapotrzebowania na wielkość pamięci na kod programu.

Implementacja AES w Javascript

AES w przeglądarce:



Przed szyfrowaniem następuje:
wygenerowanie lokalnie tablic

Potem następuje szyfrowanie z zastosowaniem tych tablic

Do pobrania: http://crypto.stanford.edu/sjcl/

9

Przykładowa implementacja AES na platformę przeglądarki może być pobrana ze strony uniwersytetu w Stanford (http://crypto.stanford.edu/sjcl/). Ona zakłada wysłanie małej ilości danych przez sieć (6,4 KB), następnie wygenerowanie pomocniczych tablic po stronie klienta (odbiorcy) a dopiero potem szyfrowanie z zastosowaniem tych pomocniczych tablic. Tak uzyskuje się kompromis pomiędzy przesyłaniem niewielkich programów, a zwiększeniem wydajności po stronie klienta.

AES w implementacjach sprzętowych

Instrukcje AES w procesorach Intel Westmere (Core i3, Core i5, Core i7, Pentium, Celeron and Xeon):

- **aesenc, aesenclast**: wykonaj jedną rundę AES
  128-bit rejestry: xmm1=stan, xmm2=klucz rundy
  aesenc xmm1, xmm2; przekaż wynik do rej. xmm1
- **aeskeygenassist**: wykonuje rozszerzenie klucza AES
- Intel twierdzi, że daje to 14 – krotne przyspieszenie w działaniu biblioteki OpenSSL na tym samym sprzęcie

Podobne instrukcje zaimplementowano w architekturze AMD Bulldozer

10

Algorytm szyfrowania AES ma też rozwiązania sprzętowe wprost zaimplementowane w struktury współczesnych mikroprocesorów. Procesory Intela poczynając od mikroarchitektury Westmere mają wbudowane gotowe instrukcje (assembler) do dokonywania szyfrowania z zastosowaniem algorytmu AES.
Instrukcja **aesenc** wykonuje jedną rundę algorytmu AES. Przyjmuje 2 argumenty 128 bitowe: wektor wejściowy i klucz rundy, wynik rundy jest przekazywany do rejestru xmm1. Instrukcja **aesenclast** wykonuje ostatnią rundę (inną niż pozostałe) algorytmu. Instrukcja **aeskeygenassist** dokonuje rozszerzenia klucza. Pełna implementacja szyfrowania AES polega na 9-krotnym wykonaniu instrukcji **aesenc**, 1-krotnym wykonaniu **aesenclast**.
Twórcy implementacji twierdzą, że uzyskali w ten sposób 14-krotne przyspieszenie szyfrowania w odniesieniu do „tradycyjnej" implementacji nieużywającej tych instrukcji sprzętowych.
Procesory AMD również proponują odpowiadające instrukcje zrealizowane w sprzęcie.

Ataki

zamiast klucza 128-bit trzeba myśleć jak mamy 126-bit

Najlepszy atak na odzyskanie klucza: czterokrotnie szybszy niż siłowe przeszukiwanie [BKR'11]

luka w rozszerzaniu klucza

Atak na powiązane klucze na AES-256: [BK'09]

Mając 2^{99} par wiadomość/szyfrogram dla czterech powiązanych kluczy szyfru AES-256 pozwala na odzyskanie klucza w czasie $\approx 2^{99}$

11

Najlepszy atak na odzyskiwanie klucza na AES-128 pokazał, że jest 4-krotnie szybszy niż siłowe przeszukiwanie. Stąd, jeśli mówimy, że klucz jest długości 128 bitów, to szyfrowanie AES jest bezpieczne na poziomie na 126 bitów (czas złamania tym atakiem $\approx 2^{126}$)

W przypadku AES-256 wykazano, że algorytm rozszerzania klucza ma słabość. Można go zaatakować atakiem na „powiązane klucze". Trzeba dysponować 2^{99} par wiadomość/szyfrogram i czterema „powiązanymi kluczami" (takimi, że różnią się kilkoma bitami), to wtedy z czasem 2^{99} można odzyskać klucze. Złożoność takiego ataku jest wciąż taka duża, że nie można jej wykonać z zastosowaniem współczesnego sprzętu.
Pogarsza to jakość tej wersji szyfrowania, ale trzeba zaznaczyć, że opracowana metoda wymaga specyficznego zestawu kluczy i dużej ilości par wiadomość/szyfrogram i praktycznie nie ma zastosowania.

Zastosowanie szyfrów blokowych

Zastosowania pseudo-losowych permutacji i pseudo-losowych funkcji (**jednokrotne stosowanie tego samego klucza**)

Aplikacje: szyfrowanie wiadomości pocztowych, zawsze nowy klucz dla każdej wiadomości.

Cel: zbudowanie "bezpiecznego" systemu szyfrowania z zastosowaniem pseudo-losowych permutacji [PRP (ang. Pseudo-Random Permutation), np. AES].

Założenie w tej części wykładu: **stosujemy klucze jednorazowe**

1. Możliwości atakującego:
   Atakujący widzi tylko pojedynczy szyfrogram (stosujemy szyfrowanie z kluczem jednorazowym)

2. Cel atakującego:
   Odzyskanie wiadomości zaszyfrowanej z szyfrogramu
   (sprawdzenie bezpieczeństwa semantycznego)

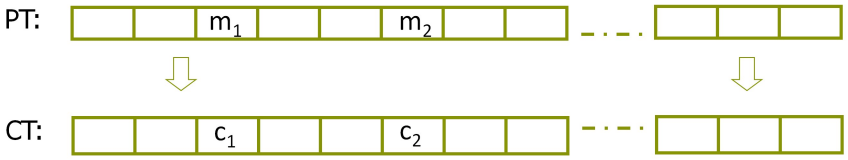
W następnym segmencie wykładu rozważymy: klucze wielokrotnego użytku

13

W tej części wykładu zajmiemy się zbudowaniem bezpiecznego systemu szyfrowania z zastosowaniem pseudolosowych permutacji i pseudolosowych funkcji (permutacje są odwracalne, a funkcje nie muszą być). Mamy np. do dyspozycji metodę szyfrowania AES, za każdym razem stosujemy pojedynczy klucz szyfrowania. Pytamy się, czy atakujący może odzyskać wiadomość z pojedynczego szyfrogramu (pojedynczy, bo za każdym razem stosujemy inny klucz). Jest to de facto sprawdzenie, czy nasz system jest semantycznie bezpieczny.
W drugiej części wykładu rozważymy bezpieczeństwo systemu, w którym klucz będzie stosowany wielokrotnie.

Niewłaściwe użycie PRP

Elektroniczna Książka Kodowa (ECB):
(ang. Electronic Code Book)



Problem:

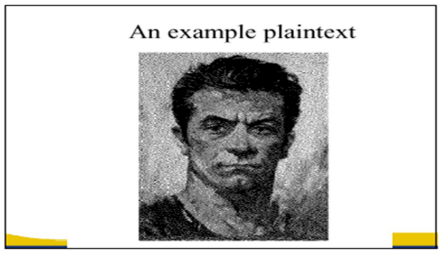
– jeśli $m_1=m_2$ to $c_1=c_2$

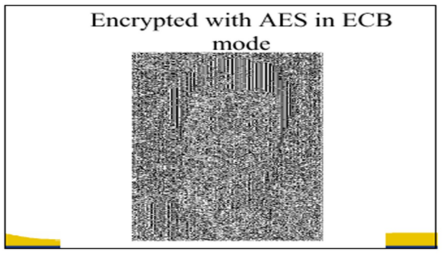
14

Klasycznym błędem zastosowania szyfrów blokowych jest tzw. Elektroniczna Książka Kodowa. Trzeba mieć świadomość, że istnieją „produkty kryptograficzne”, które działają według tej zasady i zostały łatwo złamane. Przykład ma na celu ostrzeżeni Państwa przed popełnianiem takich błędów.

Pierwszym podejściem, które się nasuwa w szyfrowaniu danych za pomocą szyfru blokowego jest podzielenie wiadomości na bloki. Każdy blok jest tak duży jak blok pojedynczego szyfrowania (np. 128 bitów). W przypadku AES użylibyśmy 16-bajtowych porcji wiadomości, a następnie szyfrowali każdą porcję osobno. Takie rozwiązanie ma bardzo mało wspólnego z bezpieczeństwem. Jeśli okażę się, że 2 bloki wiadomości do szyfrowania są takie same, to przy takim zastosowaniu szyfru blokowego dwie części szyfrogramu też będą takie same. Atakujący może od razu nie wie jak odszyfrować dane, ale już wie że te dwa bloki są identyczne.

Ilustracja na obrazie





(Udostępnione przez B. Preneel)

15

Jeśli nie przemawia do Państwa wcześniejsze „abstrakcyjne” omówienie problemu, proszę porównać obraz i jego kryptogram zaszyfrowany za pomocą elektronicznej książki kodowej. Być może nie da się rozpoznać twarzy, ale jej zarys tak oraz obszar, gdzie były kodowane włosy czy ciemna koszula. Dzieje się tak dlatego, że takie same bloki danych kodowane metodą elektronicznej książki kodowej dają takie same szyfrogramy.

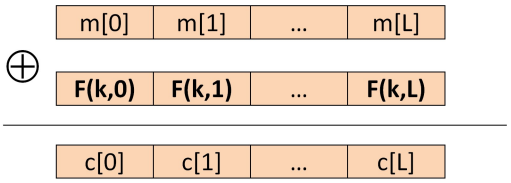
Można udowodnić matematycznie, że elektroniczna książka kodowa nie jest bezpieczna.

Powstaje więc pytanie, w jaki sposób poprawnie kodować większe porcje danych z zastosowaniem szyfrów blokowych.

Budowanie bezpiecznej konstrukcji I

Tryb z deterministycznym licznikiem i pseudolosową funkcją (PRF) F:

- E_{DETCTR} (k, m) =



⇒ Szyfr strumieniowy zbudowany z zastosowaniem funkcji pseudo-losowej (np. AES, 3DES)

⇒ Można udowodnić, że taka konstrukcja jest semantycznie bezpieczna

16

Funkcją pseudolosową może być algorytm AES. Każdy blok szyfrujemy z innym kluczem powstającym z połączenia kolejnych wartości licznika i klucza szyfrowania. W taki sposób przekształcamy szyfr blokowy w szyfr strumieniowy.

Istnieje matematyczny dowód, że taka konstrukcja jest semantycznie bezpieczna.

Semantyczne bezpieczeństwo dla systemów, w których **wielokrotnie używamy tego samego klucza**

Aplikacje: szyfrowanie plików (ten sam klucz AES do zaszyfrowania wielu plików)
IPsec (ten sam klucz AES zastosowany do szyfrowania wielu pakietów)

Klucz jest użyty więcej niż raz ⇒ atakujący widzi wiele szyfrogramów zakodowanych za jego pomocą

Nowa możliwość atakującego: atak z wybranym tekstem jawnym (CPA – ang. chosen-plaintext attack)

- Może on otrzymać szyfrogram wskazanego przez niego tekstu.
  (realny model ataku)

Cel atakującego:
Złamać semantyczne bezpieczeństwo konstrukcji kryptograficznej

17

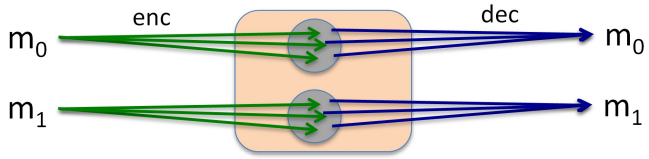
Jeśli myślimy o systemach, w których wielokrotnie stosujemy ten sam klucz, mamy na myśli systemy szyfrowania plików albo ciągu pakietów w transmisji sieciowej.

Jeśli używamy wielokrotnie tego samego klucza, to atakujący widzi wiele szyfrogramów zaszyfrowanych za pomocą tego samego klucza. W konsekwencji definiując bezpieczeństwo takiej konstrukcji kryptograficznej musimy dopuścić ze strony atakującego nowy typ ataku: atak z wybranym tekstem jawnym. Atakujący wysyła znany mu tekst jawny i może przechwycić jego szyfrogram.

Okazuje się, że taki model ataku jest powszechny i wykonalny w realnym świecie. Np. ktoś wysyła do nas wiadomość, która jest przechowywana na naszym dysku. Dysk jest zaszyfrowany. Ktoś przechwytuje ten dysk i ma: wiadomość i jej zaszyfrowaną postać.

Rozwiązanie 1: losowe szyfrowanie (ang. randomized encryption)

- E(k,m) jest algorytmem z losowaniem:



⇒ szyfrowanie tej samej wiadomości daje różne rezultaty (z dużym prawdopodobieństwem)

⇒ szyfrogram musi być dłuższy niż tekst do zaszyfrowania

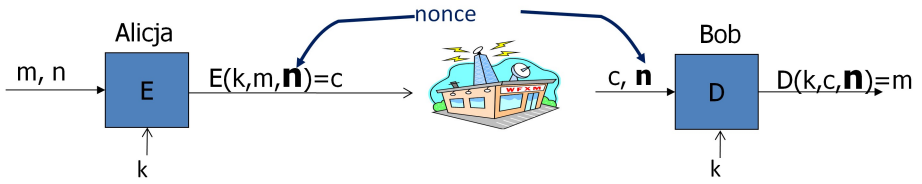
W uproszczeniu:
rozmiar szyfrogramu = rozmiar tekstu + liczba losowych bitów

18

Algorytm wybiera pewien losowy tekst i stosuje go do zaszyfrowania wiadomości „m". Wtedy pewna wiadomość m_0 może zostać zaszyfrowana jako wiele różnych wiadomości.

Niedogodnością tego rozwiązania, jest konieczność przesłania do odbiorcy tego pomocniczego losowego tekstu, lub jakiegoś jego obrazu, co wydłuża szyfrogram. Jeśli do bardzo dużej wiadomości dodajemy jakiś niewielki ciąg bitowy, to nie ma to specjalnego znaczenia. Inaczej ma się sprawa, gdy wysyłane są krótkie wiadomości, wtedy blok losowych danych dodawanych do szyfrogramu może mieć podobną długość jak sama wiadomość i trudno tu mówić o wydajności takiego sposobu szyfrowania.

Rozwiązanie 2: szyfrowanie z wartością „nonce” („nonce” = jednokrotna)



- nonce n: wartość zmieniana za każdym razem, gdy szyfrujemy wiadomość.
  (k,n) taka para musi być użyta tylko raz
- Metoda 1: nonce jest **licznikiem** (np. numer pakietu)
  - Stosowane, gdy szyfrator utrzymuje pewien stan w trakcie generowania szyfrogramów
  - Jeśli deszyfrator też utrzymuje stan, to nie ma potrzeby przesyłania wartości nonce (tu: licznika) w szyfrogramie
- Metoda 2: szyfrator wybiera **losową wartość nonce**, $n \leftarrow \mathbb{N}$
- Można udowodnić, że szyfrowanie z wartością nonce jest semantycznie bezpieczne o ile zapewnimy, że wartość nonce nigdy się nie powtórzy w czasie jednej sesji szyfrowania.

19

Algorytm szyfrujący bierze jako wejście 3 wartości: wiadomość m, klucz k i „nonce” n (jednokrotna wartość, taka, że para (k,n) jest użyta w szyfrowaniu tylko raz). Algorytm deszyfrujący także stosuje 3 wartości: szyfrogram c, klucz k, oraz wartość jednokrotna n.

Wartość nonce (jednokrotna) nie musi być ukryta przed atakującym, ani losowa. Jedynym warunkiem jest, żeby para (k,n) była zastosowana do zaszyfrowania tylko jednej wiadomości. Zamiast stosować różne klucze stosuje się różne wartości nonce (jednorazowe).

Dobór nonce może być różny. W protokołach sieciowych to może być licznik. Może być on zwiększany o jeden po przesłaniu danego pakietu.

Jeśli nadawca i odbiorca potrafią utrzymać pewien stan (sesję), to w czasie przesyłania pakietów nie trzeba nawet przesyłać ich numerów, bo nadawca i odbiorca wie, jaki jest numer kolejnego pakietu. Na takiej zasadzie działa np. protokół HTTPS. Niższe warstwy przesyłania danych gwarantują kolejność przesyłania porcji danych w tym protokole, stąd nie ma potrzeby przesyłania stanu licznika. Natomiast w protokole IPsec nie ma gwarancji przesyłania danych w określonej kolejności (bazuje on na pakietach IP, które mogą dochodzić do odbiorcy różnymi drogami i w różnej kolejności), dlatego każdy pakiet ma dołączoną wartość licznika.

Innym sposobem jest wybranie nonce w sposób losowy. Trzeba wtedy zapewnić, że przestrzeń losowych wartości nonce jest wystarczająco duża, żeby zmniejszyć prawdopodobieństwo wylosowania dwóch identycznych wartości na czas życia klucza (na czas sesji szyfrowanego przesyłania danych). Stosowanie takich nonce występuje, kiedy koordynacja między nadawcą a odbiorcą jest utrudniona, albo jeśli szyfrujemy dane tym samym kluczem na różnych urządzeniach (np. na laptopie i na smartfonie). Wtedy nie musimy uzgadniać wartości nonce i z dużym prawdopodobieństwem ten sam tekst zostanie zaszyfrowany w inny sposób, co zapobiega atakom z wybranym tekstem jawnym.

Można udowodnić, że szyfrowanie z nonce jest semantycznie bezpieczne, jeśli zapewnimy, że w czasie jednej sesji szyfrowania ta wartość się nie powtórzy.

Semantyczne bezpieczeństwo dla wielokrotnego szyfrowania wiadomości za pomocą tego samego klucza

Wnioski po analizach matematycznych:

- Jeśli dopuścimy, że takie same wiadomości, pliki etc. Zaszyfrowane tym samym kluczem będą dawały takie same szyfrogramy, to ten model szyfrowania nie jest semantycznie bezpieczny
- Musimy zbudować taki system szyfrowania, który, pomimo wielokrotnego stosowania tego samego klucza będzie dawał różne wyniki dla takich samych danych wejściowych.

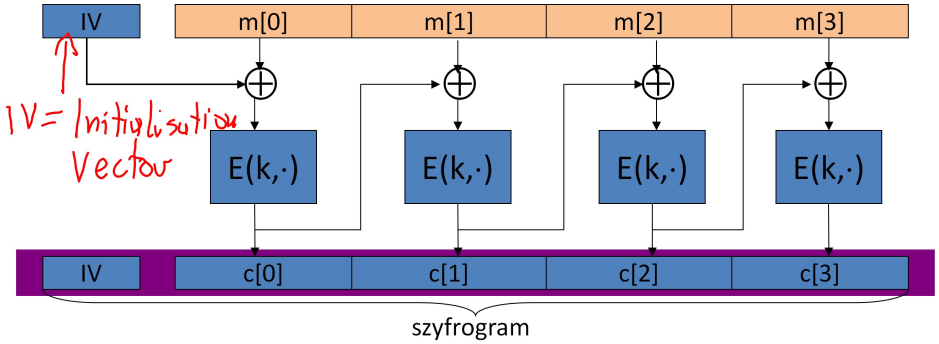
20

Szyfrowanie z łańcuchem bloków (CBC – Cipher Block Chaining)

Aplikacje: System plików – szyfrowanie plików tym samym kluczem AES
IPSec – szyfrowanie pakietów tym samym kluczem AES

Konstrukcja I: CBC z losowym IV

Niech (E,D) będzie PRP. $E_{CBC}(k,m)$: wybierz **losowe** IV∈X i wykonuj:

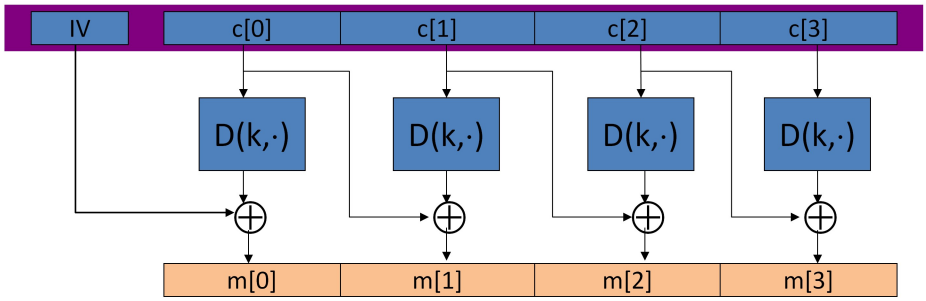


22

Szyfrowanie rozpoczyna się od wybrania jednego **losowego** bloku IV o długości pojedynczego bloku szyfru AES: 128 lub 192 lub 256). Na pierwszej części danych do zaszyfrowania jest wykonywany XOR z IV i ta wartość jest szyfrowana z zastosowaniem szyfru blokowego i otrzymujemy pierwszy blok szyfrogramu. Ten blok szyfrogramu jest brany jako wejście do szyfrowania kolejnej porcji danych. Na kolejnej porcji danych do zaszyfrowania wykonywany jest XOR z poprzednim blokiem szyfrogramu i tak skonstruowany ciąg bitowy jest podawany na wejście blokowego algorytmu szyfrowania. I tak dalej… Proces szyfrowania tworzy łańcuch, w którym na początku bierzemy IV, ale potem IV dla kolejnych etapów szyfrowania staje się blok szyfrogramu z poprzedniej sesji. Ostatecznie szyfrogram składa się z początkowej wartości IV i kolejnych bloków otrzymanych z szyfru blokowego.

„Obwód” deszyfrujący

Symbolicznie: $c[0] = E(k, IV \oplus m[0])$ $\Rightarrow$ $m[0] = D(k, c[0]) \oplus IV$



- Można udowodnić, że metoda CBC jest semantycznie bezpieczna i odporna na atak z wybranym tekstem jawnym (CPA), jeśli nie przekroczymy pewnej ilości bloków w łańcuchu
- Dla AES: po 2^{48} blokach należy zmienić klucz
- Dla 3DES po 2^{16} blokach należy zmienić klucz

23

Żeby odszyfrować pierwszy blok danych wykonujemy na nim algorytm deszyfracji, a następnie XOR z IV. Kolejny blok odszyfrowujemy z zastosowaniem algorytmu deszyfracji oraz operacji XOR na wcześniejszym bloku szyfrogramu itd…

Jeśli chodzi o matematyczne rozważania, to dla takiej konstrukcji kryptograficznej wykazano, że jest semantycznie bezpieczna i odporna na atak z wybranym tekstem jawnym (CPA), jeśli nie przekroczona zostanie pewna wyliczalna liczba elementów w łańcuchu. Pomijając zagadnienia teoretyczne, można wyliczyć, że ta metoda szyfrowania jest bezpieczna o ile nie przekroczymy 2^{64} bloków w łańcuchu dla AES i 2^{16} bloków w łańcuchu dla 3DES (dla 3DES co ½ MB danych trzeba zmienić klucz).

Ważna uwaga

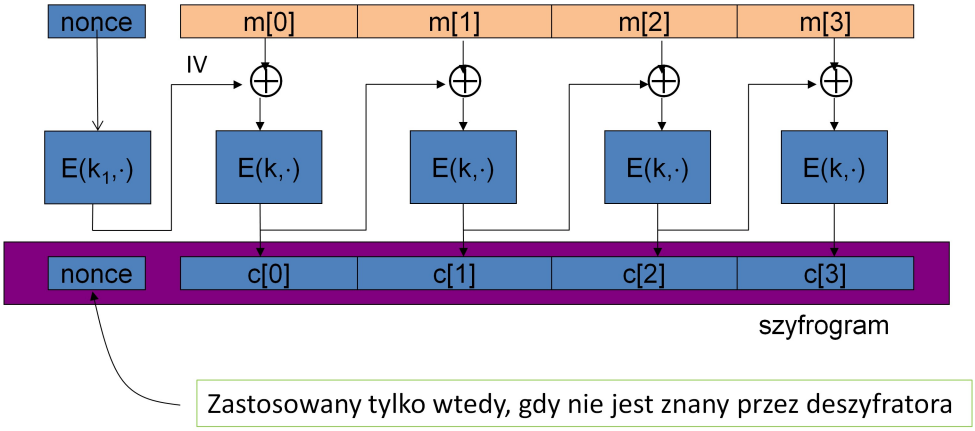
- CBC przestaje być bezpieczne, jeśli atakujący może przewidzieć wartość IV.
  - Istniał błąd w specyfikacji SSL/TSL 1.1, gdzie okazało się, że IV dla rekordu „i” był wartością ostatniego bloku poprzedniego szyfrogramu „i-1”.
  - Atakujący mając ostatni blok poprzedniego szyfrogramu mógł wyliczyć IV dla następnego
  - Przeprowadzono spektakularny atak na taką konstrukcję i wykazano, że nie jest bezpieczna
  - Przy stosowaniu takiej konstrukcji trzeba pamiętać, że jej bezpieczeństwo opiera się na losowości IV i niemożliwości przewidzenia kolejnego IV na podstawie poprzedniego.

24

Konstrukcja 1': zastosowanie CBC opartym na nonce

- Łańcuch bloków z <u>unikatowym</u> nonce: klucz = (k,k_1)

unikatowy oznacza:, że para (klucz, n) jest zastosowana tylko dla jednej widomości



Zastosowany tylko wtedy, gdy nie jest znany przez deszyfratora

25

Zaletą tego rozwiązania jest jeśli nadawca i odbiorca mogą ustalić wartość nonce, to nie musi ona być dołączona do szyfrogramu. Klucz dla takiego rozwiązania musi się składać z dwóch wartości k i k_1. k jest stosowany do wykonywania szyfrowania w algorytmie szyfrowania blokowego, natomiast k_1 musi posłużyć do zaszyfrowania wartości nonce, aby została przekształcona w wartość o charakterze losowym. Bez tego etapu szyfrowania taki łańcuch szyfrów blokowych nie jest bezpieczny. Wykazano równie, że gdybyśmy zastosowali do zaszyfrowania nonce klucz k, to konstrukcja także nie byłaby bezpieczna.

Okazuje się, że w wielu rozwiązaniach zapomniano o szyfrowaniu nonce za pomocą osobnego klucza.

Przykładowe Crypto API (OpenSSL)

```
void AES_cbc_encrypt(
        const unsigned char *in,
        unsigned char *out,
        size_t length,
        const AES_KEY *key,
        unsigned char *ivec,    ← użytkownik podaje IV
        AES_ENCRYPT or AES_DECRYPT);
```

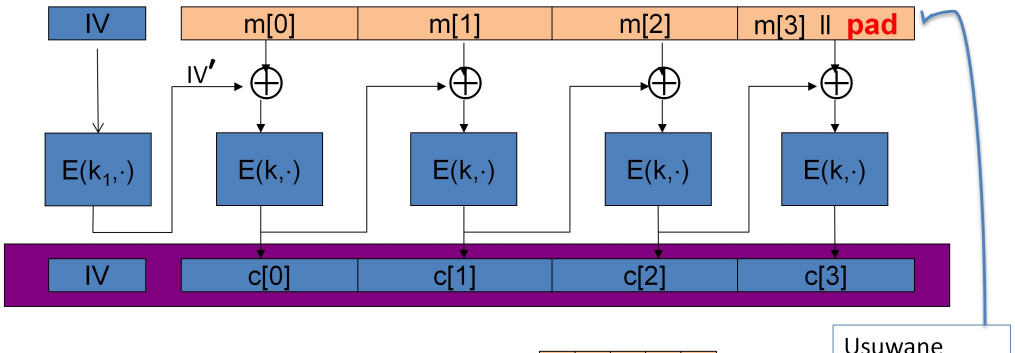
Kiedy nonce nie jest wartością losową, musi zostać zaszyfrowane przed zastosowaniem w wywołaniu tej funkcji.

26

Warto zwrócić uwagę, że w niektórych implementacjach struktur kryptograficznych API jest tak skonstruowane, że prowadzi do nieprawidłowego rozwiązania.

Na złączonym slajdzie jednym z parametrów funkcji realizującej szyfrowanie z łańcuchem bloków jest podanie IV. Funkcja używa IV wprost jako wejście, nie poddaje go szyfrowaniu. Jeśli użytkownik poda tu jakąś wartość, która nie jest losowa, to bezpieczeństwo systemu jest załamane. Tak więc trzeba wiedzieć, że w implementacji OpenSSL, albo we wskazanym miejscu należy podać wartość losową, albo zaszyfrować ją z zastosowaniem osobnego klucza i dopiero podać na wejście funkcji realizującej szyfrowanie z łańcuchem bloków.

Techniczna uwaga do CBP - padding



TLS: dla n>0, n bajtowy pad
jeśli nie potrzeba paddingu,
dodaj sztuczny blok.

n	n	n	•••	n

Usuwane podczas deszyfrowania

27

Jeśli nasza wiadomość nie jest wielokrotnością długości pojedynczego bloku szyfrowania, to trzeba dodać do niej sztucznie tyle bajtów, aby miała taką długość. Jednym z popularniejszych sposobów uzupełniania wiadomości jest uzupełnienie ostatniego bloku serią bajtów, z których każdy zawiera liczbę bajtów, którą należy odrzucić z wiadomości po odszyfrowaniu (TSL). Przykładowo, jeśli do widomości należy dołączyć 5 bajtów, to ostatnie dodatkowe pięć bajtów jest zakodowaną wartością 5: 55555. Program deszyfrujący sprawdza, ile wynosi ostatni bajt i tyle bajtów odcina. Ciekawa sytuacja zachodzi, gdy wiadomość ma długość dokładnie równą wielokrotności 16 bajtów. Wtedy na koniec wiadomości dołącza się 16 bajtowy blok z wartościami 16 w każdym bajcie. Algorytm deszyfrujący po prostu odrzuca cały taki blok.

Konsekwencją takiego postępowania jest wydłużenie szyfrogramów nawet o 16 bajtów, co przy małych porcjach danych może okazywać się znaczącą wartością.

Szyfrowanie w trybie licznikowym (CTR – Counter)

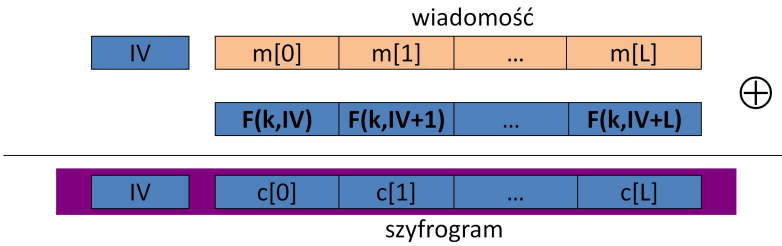
Aplikacje: System plików – szyfrowanie plików tym samym kluczem AES
IPSec – szyfrowanie pakietów tym samym kluczem AES

28

Konstrukcja II: losowy tryb licznikowy

Niech F: $K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ będzie bezpieczną PRF.

E(k,m): wybierz losowy $IV \in \{0,1\}^n$ i wykonuj:



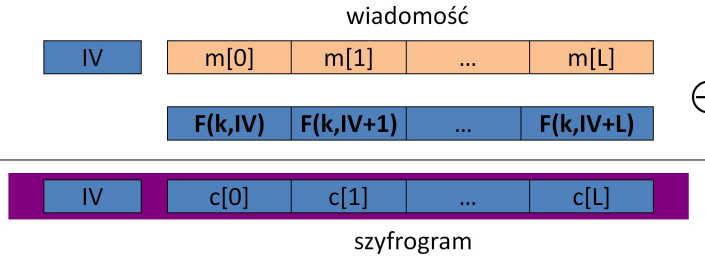
uwaga: możliwe do zrównoleglenia (w przeciwieństwie do CBC)

29

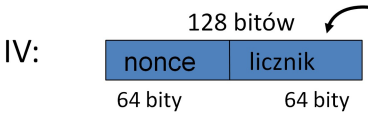
Pokazane rozwiązanie jest lepsze od wcześniejszego (CBC). Stosuje ono funkcję pseudolosową F, ponieważ nigdy nie musi być odwracalna (funkcja pseudolosowa w odniesieniu do permutacji pseudolosowej ma znacznie większą liczbę wyjść – możliwych ciągów losowych). To rozwiązanie nie potrzebuje nawet blokowego algorytmu szyfrowania (ale teraz będziemy go stosować właśnie z powiązaniem z szyframi blokowymi). Zakładamy więc , że F jest bezpieczną funkcją pseudolosową i przetwarza ona n-bajtowe bloki danych. Jeśli myślimy o zastosowaniu algorytmu AES, to bloki będą 128 bitowe.

Konstrukcja zaczyna od wylosowania wartości IV (ang. Initialisation Vector). Ta wartość (w przypadku AES 128 bitowa) jest początkową wartością licznika pracującego w konstrukcji. Kolejne bloki danych są szyfrowane (XOR) za pomocą funkcji F generującej pseudolosowe ciągi na podstawie **klucza** oraz **IV** zwiększanego o jeden dla kolejnych bloków danych. Takie rozwiązanie można odnieść do generowania losowego długiego ciągu danych i szyfrowania za jego pomocą kolejnych bloków danych. Proszę zwrócić uwagę, że IV jest tutaj włączony do szyfrogramu. Przy szyfrowaniu nowych wiadomości za każdym razem następuje wylosowanie nowego IV. Wtedy szyfrowanie tej samej wiadomości daje różne szyfrogramy. Dodatkowo pokazane rozwiązanie jest łatwe do zrównoleglenia, co nie było możliwe w przypadku konstrukcji CBC (tam żeby obliczyć szyfrogram z bloku 5 trzeba było mieć wyliczone szyfrogramy z wszystkich poprzednich bloków danych, a więc nawet jeśli dysponowalibyśmy możliwością współbieżnego szyfrowania AES, to nie moglibyśmy jej w schemacie CBC zastosować). W obecnej konstrukcji (CRT) każdy blok może być szyfrowany współbieżnie z pozostałymi. Trzeba mu tylko podać na wejście wartość IV powiększoną o odpowiednią wartość licznika i oczywiście ten sam klucz.

Konstrukcja II: tryb licznikowy z „nonce”



Żeby zapewnić, że funkcja F z parametrami (k, x) (F(k,x)) nie jest nigdy używana więcej niż raz, wybiera się IV jako:



Startuje od 0 dla każdej wiadomości

Bezpieczeństwo:
Dla AES po 2^{64} blokach trzeba zmienić klucz.

30

Tryb licznikowy szyfrowania me też wariant pracy z początkową wartością nonce (jednorazową). Wtedy IV nie jest naprawdę losowy, tylko jest to nonce jako wartość inicjalizująca licznik. Jedną z możliwości tworzenia IV jest podzielenie 128 bitowego bloku danych na 2 równe pola. W jednym będzie przechowywana wartość nonce, a w drugim wartość licznika, który będzie startował zawsze od 0 dla nowego nonce. Dobrze jest, jeśli nonce jest dobierane losowo. Warunkiem bezpieczeństwa jest wybranie nowego nonce po „przekręceniu” się licznika (2^{64} możliwych wartości) oraz niedopuszczanie do zerowania się licznika przy szyfrowaniu (reset licznika). Gdyby pozostawić ten sam licznik, w szyfrogramie pojawiły by się bloki zaszyfrowane przy pomocy tej samej pary (nonce, licznik), co przeczy pojęciu bezpieczeństwa.

Porównanie CRT z CBC

	CBC	Tryb licznikowy
Stosuje	PRP	PRF
Możliwość zrównoleglenia	Nie	Tak
Bezpieczeństwo: q- liczba wiadomości zaszyfrowanych k L – długość najdłuższej wiadomości |X| - ilość możliwych kluczy	q^2 L^2 << |X|	q^2 L << |X|
Uzupełnianie brakujących bajtów w bloku	Tak	Nie
Jeśli zastosujemy wiadomości 1 – bajtowe (szyfrowanie z nonce)	16- krotne wydłużenie	Brak wydłużenia

(dla CBC, uzupełnienie bloków może być rozwiązane przez „kradzenie szyfrogramu”)

31

Pokazana tabela jest porównaniem konstrukcji szyfrujących CBC i trybu licznikowego. Od razu można powiedzieć, że tryb licznikowy w każdym aspekcie przewyższa tryb łańcuchowy. We nowych rozwiązaniach kryptograficznych stosuje się najczęściej tryb licznikowy, chociaż CBC wciąż jest stosowany.

Po pierwsze do poprawnego działania tryb CBC musi stosować szyfr blokowy, bo odszyfrowywanie poszczególnych bloków polega na uruchamianiu kolejnych instancji szyfru blokowego. W drugim podejściu nie używamy szyfrów blokowych do odszyfrowywania. Poza tym tryb licznikowy jest ogólniejszy i nie wymaga szyfrowania poszczególnych bloków szyframi blokowymi. Może być do tego np. zastosowany szyfr strumieniowy Salsa20.

Szyfrowanie w trybie licznikowym może być łatwo zrównoleglone, podczas gdy szyfrowanie łańcuchowe w swej naturze jest sekwencyjne.

Tryb licznikowy jest też bardziej bezpieczny. W tabeli podano zależność, która powinna być spełniona dla zapewnienia wysokiego bezpieczeństwa. Na wcześniejszych slajdach podano pewne oszacowania uwzględniające podaną zależność i dla AES można przyjąć, że w trybie CBC po 2^{48} blokach należy zmienić klucz natomiast dla trybu licznikowego po 2^{64} blokach.

Szyfrowanie z CBC wymaga uzupełniania długości bloku do 16 bajtów, oraz dodawania 16 bajtów, nawet kiedy wiadomość jest krotnością 16 bajtów. W szyfrowaniu w trybie licznikowym takiego wymagania niema. Warto wspomnieć (dół slajdu), że istnieje **nieustandaryzowane** rozwiązanie szyfrowania w trybie CBC, które nie wymaga dodawania bajtów. Nosi nazwę „kradzenie szyfrogramu” (ang. ciphertex stealing). Wtedy możemy tę wadę trybu CBC uznać za mniej ważną.

Podsumowanie

- PRP i PRF: są użytecznymi abstrakcjami do modelowania szyfrów blokowych.
- Analizowaliśmy 2 pojęcia bezpieczeństwa: (ochrona przed podsłuchiwaniem)
  1. Semantyczne bezpieczeństwo przed jednokrotnym atakiem z wybranym tekstem (one-time CPA).
  2. Semantyczne bezpieczeństwo przed wielokrotnym atakiem z wybranym tekstem (many time CPA).

  Uwaga: żadne nie zapewnia integralności szyfrogramów.
- Ustalone jak na razie rezultaty bezpieczeństwa można podsumować w tabeli:

Cel \ Moc	Klucz jednorazowy	Klucz wielorazowy (CPA)	CPA i integralność
Semantyczne bezpieczeństwo	Szyfry strumieniowe deterministyczny tryb licznikowy	losowe CBC losowy tryb licznikowy (CTR)	później

32

Podsumowując, spoglądając na zasadę działania szyfrów blokowych dobrymi ich przybliżeniami są pseudolosowe permutacje (PRP) i pseudolosowe funkcje (PRF). W przeprowadzonych rozważaniach wspominaliśmy, że pokazane metody szyfrowania są semantycznie bezpieczne na atak z wybranym tekstem (mamy tekst i jego szyfrogram i chcemy odzyskać klucz). W przypadku konstrukcji z zastosowaniem szyfrów blokowych przytoczyliśmy liczby pokazujące, jak często trzeba zmieniać klucz szyfrowania, żeby zachować bezpieczeństwo. Poroszę zwrócić uwagę, że w dalszym ciągu omawiane konstrukcje nie zapewniają ochrony przed manipulacjami danych (atak na integralność). W konsekwencji nie znamy jeszcze poprawnej metody szyfrowania i ochrony danych. Omówione konstrukcje stanowią jednak bazę do zbudowania systemów, które staną się bezpieczne na ataki na integralność przesyłanych informacji.

Jak dotąd wprowadziliśmy zagadnienia szyfrowania z kluczem jednorazowym, gdzie stosowaliśmy szyfry strumieniowe i szyfry blokowe z deterministycznym trybem licznikowym, oraz systemy z wielokrotnym zastosowaniem tego samego klucza (Losowe CBC oraz losowy tryb licznikowy). W dalszym ciągu nie dysponujemy systemem chroniącym integralność naszych strumieni/bloków danych.