

Kryptografia i bezpieczeństwo danych

- przegląd zagadnień

dr inż. Sławomir Samolej

email: ssamolej@kia.prz.edu.pl

WWW: ssamolej.kia.prz.edu.pl

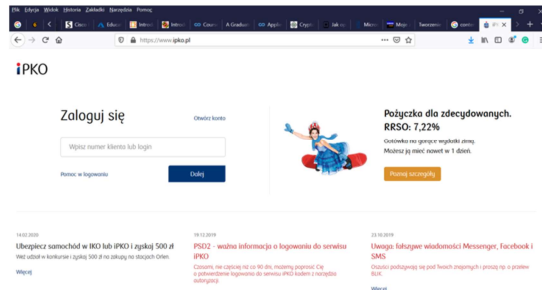
Literatura

- Kurs internetowy „Cryptography I”
<https://www.coursera.org>,
- Książka w wolnym dostępie: Dan Boneh and Victor Shoup, „A Graduate Course in Applied Cryptography”, 2020,
<https://toc.cryptobook.us/>
- Jean-Philippe Aumasson, Nowoczesna Kryptografia, Praktyczne wprowadzenie do szyfrowania, PWN 2018
- Marcin Karbowski, Podstawy Kryptografii, Wydanie II, Helion 2014
- Wiliam Stallings, Lawrie Brown, Bezpieczeństwo Systemów Informatycznych, Zasady i Praktyka, Tom 1 i 2, Helion 2019

Kryptografia jest wszędzie...

- **Ochrona przesyłu danych**
 - Chroniony dostęp do stron: HTTPS
 - Sieci bezprzewodowe: 802.11i WPA2, GSM, Bluetooth
- **Szyfrowanie danych na dyskach:** EFS, TrueCrypt
- **Ochrona informacji:** (np. DVD, Blu-ray): CSS, AACs
- **Autentykacja użytkowników**
... i wiele innych

Bezpieczna komunikacja



HTTPS



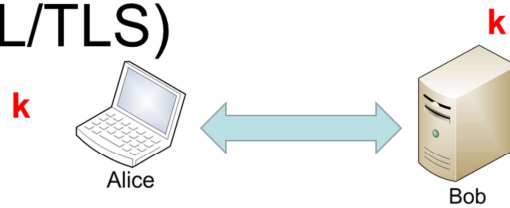
Bob



Alice

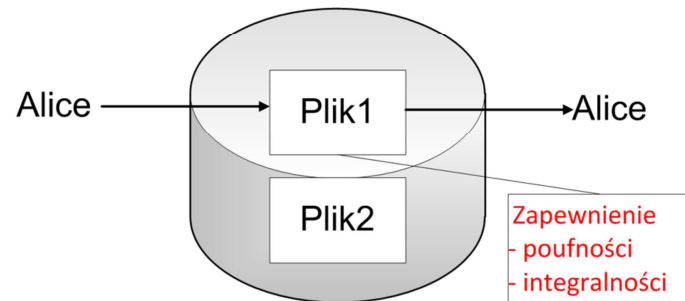


Secure Sockets Layer / Transport Layer Security (SSL/TLS)



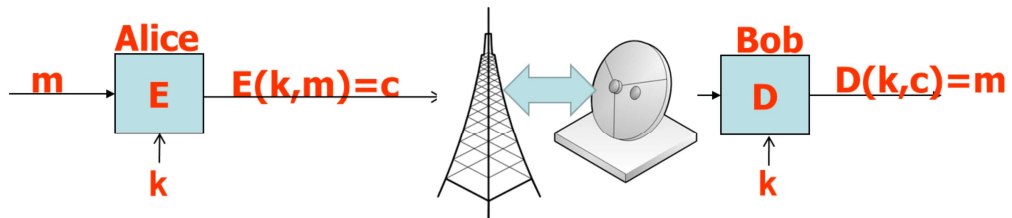
- Dwie podstawowe części:
 - Protokół „handshake” (uzgodnienie):
Ustanowienie wspólnego tajnego klucza z zastosowaniem kryptografii klucza publicznego
 - Warstwa „record” (ustalenie formatu przesyłu pakietów): **Transmisja danych z zastosowaniem tajnego klucza symetrycznego**
Zapewnienie poufności i integralności danych

Ochrona danych dyskowych



Problem szyfrowania dysków jest analogiczny do chronionej komunikacji, kiedy Alice wysyła wiadomość do Alice

Schemat kryptografii symetrycznej



E, D: algorytm szyfrujący/deszyfrujący
k: tajny symetryczny klucz (np. 128 bitowy)
m, c: wiadomość, zaszyfrowana wiadomość

- Algorytm szyfrowania jest **upubliczniony**
- Nie używamy własnych szyfrów
- Jedynym tajnym elementem jest klucz

Tryby użycia klucza

Jednorazowe użycie klucza: (one time key)

- Klucz jest jednorazowo zastosowany do zaszyfrowania wiadomości
 - np. szyfrowanie poczty: nowy klucz jest generowany dla każdej wiadomości

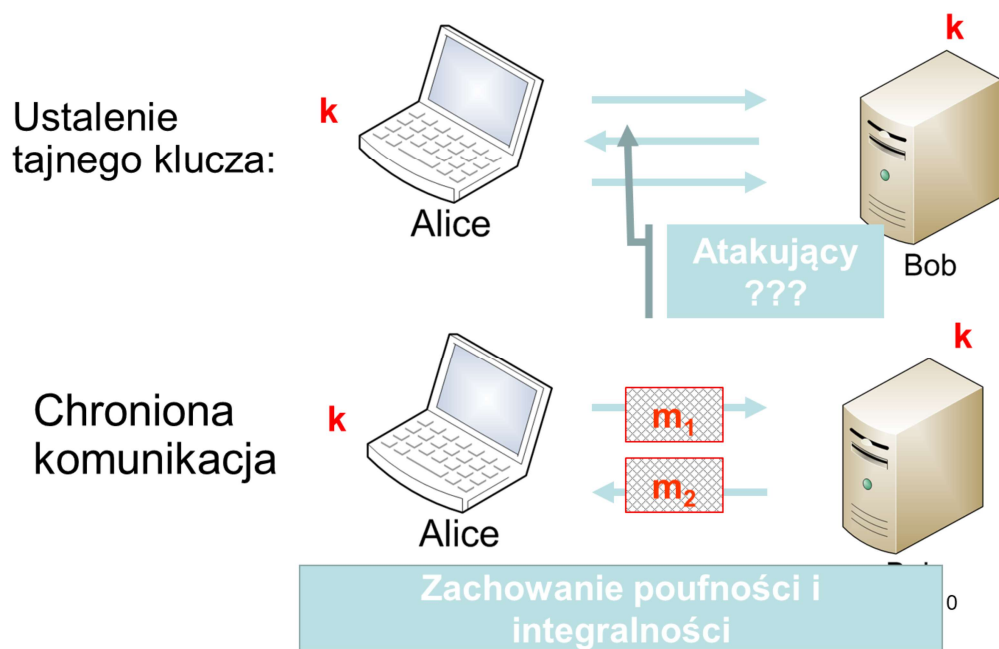
Wielokrotne używanie klucza: (many time key)

- Ten sam klucz jest stosowany do szyfrowania wielu wiadomości
 - np. szyfrowanie plików: ten sam klucz służy do szyfrowania wielu plików
- W takim przypadku będziemy potrzebowali dodatkowych zabiegów, żeby zapewnić bezpieczeństwo

Uwagi na tym etapie

- Kryptografia jest:
 - Doskonałym narzędziem do ochrony informacji w komputerach
 - Podstawą do stworzenia wielu mechanizmów ochrony przesyłu informacji i przechowywania danych
- Kryptografia nie jest:
 - Rozwiązaniem wszystkich problemów ochrony (np. błędy oprogramowania, ataki stosujące socjotechniki)
 - Zabezpieczeniem, jeśli jest stosowana w niewłaściwy sposób (np. WEP)
 - Czymś, co trzeba wymyślić samemu
 - Jest bardzo wiele przykładów złamanych systemów szyfrowania powstałych at-hoc

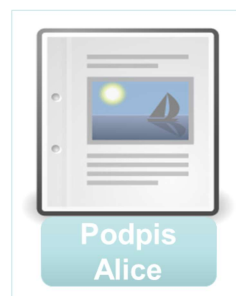
Jądro kryptografii



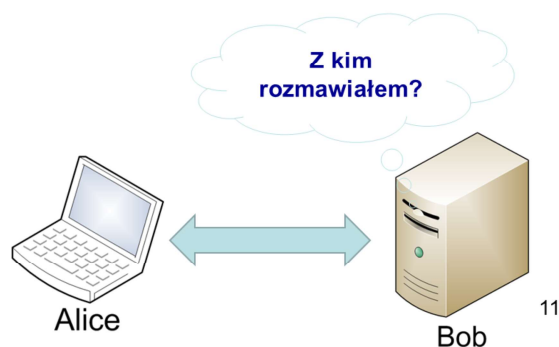
Kryptografia umożliwia więcej...

(1)

- Podpis elektroniczny



- Anonimowa komunikacja (mix net)



Kryptografia umożliwia więcej...

(2)

- Anonimowa **cyfrowa** waluta
 - Czy mogę wydać „cyfrową monetę” z zachowaniem anonimowości?
 - Jak zapobiec podwójnemu płaceniu tą samą monetą?



Internet
anonimowa komunikacja



Kto to był?

12

Protokoły

- Wybory



Protokoły

- Aukcja w systemie **Vickreya**



Protokoły

- Bezpieczne obliczenia wielostronne



Cel: obliczyć $f(x_1, x_2, x_3, x_4)$

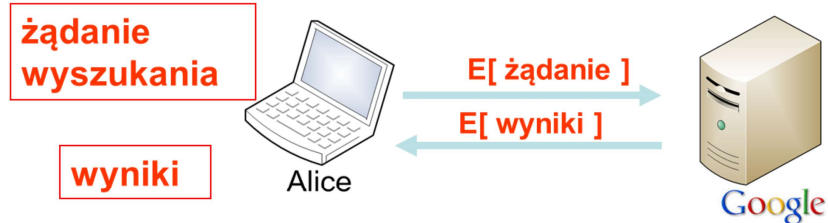
Zaufany
urząd

“Twierdzenie:” cokolwiek może być wykonane z zastosowaniem zaufanego urzędu może być wykonane bez niego

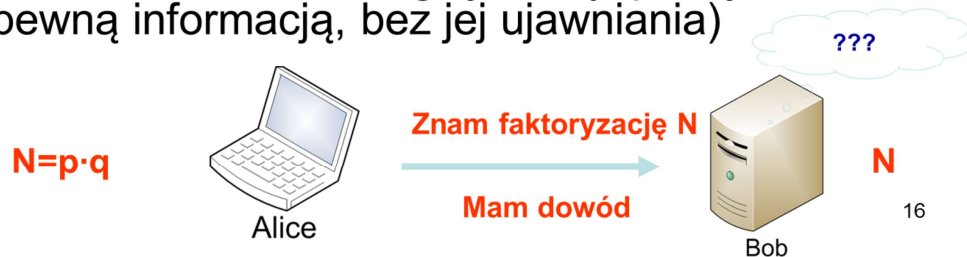
15

Magia kryptografii

- Prywatne zlecenia zdalnych obliczeń



- Dowód z wiedzą zerową (jedna ze stron potrafi udowodnić drugiej, że dysponuje pewną informacją, bez jej ujawniania)



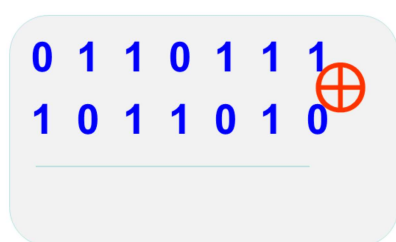
Precyzyjna nauka

Trzy kroki w kryptografii:

- Precyzyjny model zagrożenia
- Propozycja konstrukcji
- Udowodnienie, że złamanie konstrukcji pod względem zagrożenia jest bardzo trudnym problemem

Podstawowy aparat matematyczny

**Operacja
bitowa XOR**

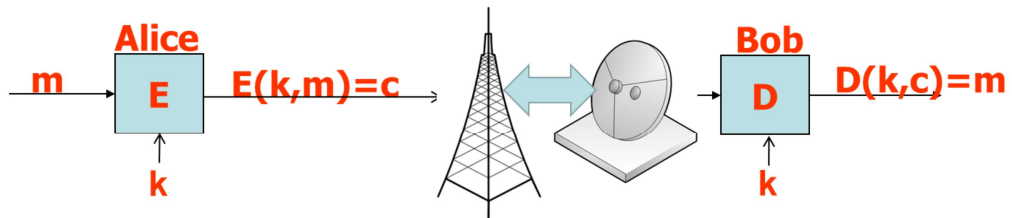


Wejście		WYJŚCIE
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Ważna właściwość funkcji XOR

- (Twierdzenie, nieformalnie)
Jeśli klucz szyfrujący jest ciągiem losowym, to wykonanie operacji $\oplus$ (XOR) na dowolnym ciągu bitowym i tym kluczu daje dalej ciągiem losowym.

Schemat kryptografii symetrycznej



E, D: algorytm szyfrujący/deszyfrujący
k: tajny symetryczny klucz (np. 128 bitowy)
m, c: wiadomość, zaszyfrowana wiadomość

- Algorytm szyfrowania jest **upubliczniony**
- Nie używamy własnych szyfrów
- Jedynym tajnym elementem jest klucz

Szyfr z kluczem jednorazowym (One Time Pad) (Vernam 1917)

- Pierwszy przykład „bezpiecznego” szyfru.
- Przestrzeń wiadomości ($\mathcal{M}$) i szyfrów ($\mathcal{C}$) jest n -bitowym ciągiem.
- Klucz jest losowym ciągiem bitów o długości wiadomości.

21

Uwaga: przestrzeń wiadomości ma taki sam rozmiar co przestrzeń kluczy są to n -bitowe ciągi.

Definicja szyfru z kluczem jednorazowym

- $C = E(k, m) = k \oplus m$
- $D(k, m) = k \oplus c$

msg: 0 1 1 0 1 1 1
key: 1 0 1 1 0 1 0 $\oplus$
CT:

22

Własności operacji xor:

- 1) $m \text{ xor } k \rightarrow c$, równocześnie $c \text{ xor } k \rightarrow m$ (mamy uniwersalny sposób szyfrowania i deszyfrowania z kluczem tajnym)
- 2) Jeśli weźmiemy dowolny ciąg losowy i niezależny od niego jednorodny ciąg losowy to po wykonaniu na nich xor otrzymamy jednorodny ciąg losowy.

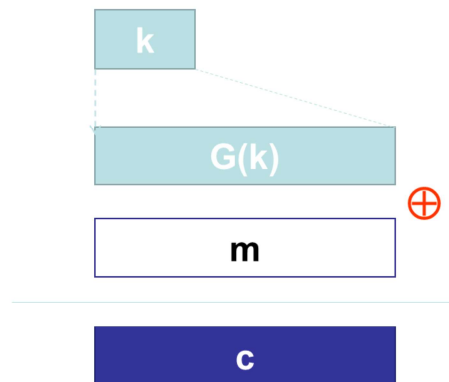
Szyfr strumieniowy: przekształcenie pomysłu na szyfr z kluczem jednorazowym na coś praktycznego

- Pomysł: należy zastąpić „losowy” klucz kluczem generowanym przez algorytm „pseudolosowy”.
- Do generowania kluczy służą generatory liczb pseudolosowych (**PRG – pseudo-random generator**): algorytm, który na podstawie pewnego ciągu zero-jedynkowego (**ziarna**) generuje znacznie dłuższy ciąg mający charakter pseudolosowy.
- Ciąg musi być efektywnie „wyliczalny” i generowany przez deterministyczny algorytm.
- Wygenerowany ciąg musi „wyglądać” jak losowy.

Szyfr strumieniowy zasada działania

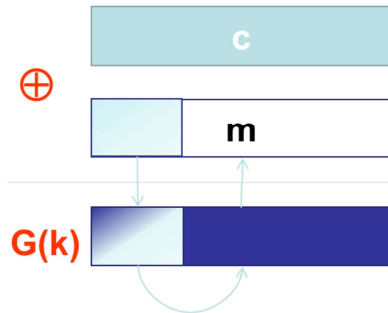
$$C = E(k,m) = m \oplus G(k)$$

$$D(k,c) = c \oplus G(k)$$



PRG musi być nieprzewidywalny

- Nieodpowiedni algorytm PRG pozwala na określenie kolejnej sekwencji (pseudolosowych) bitów na podstawie pewnej początkowej sekwencji



Formalnie: PRG jest nieprzewidywalny, jeśli następny element ciągu pseudolosowego jest możliwy do obliczenia na podstawie wcześniej obliczonych wartości z bardzo małym prawdopodobieństwem (np. $1/2^{30}$)

25

Jeśli znamy część wiadomości (np. wiadomo, że jest to zaszyfrowany email, wtedy zawsze zaczyna się od „From:”) i algorytm generowania liczb pseudolosowych jest przewidywalny (to znaczy, że na podstawie pierwszych i -bitów ciągu losowego można obliczyć pozostałe n -bitów), to wystarczy wykonać xor z pierwszej części szyfrogramu i znanej nam części wiadomości. Rezultatem będzie pierwsza część ciągu losowego wygenerowanego przez przewidywalny algorytm. Na jej podstawie można obliczyć resztę tego ciągu i wiedząc, że jest kluczem odszyfrować resztę wiadomości.

Problem występuje nawet wtedy, gdy bylibyśmy w stanie przewidzieć nawet 1 bit po i -bitach wcześniej wygenerowanych.

Atak 1 na szyfr z kluczem jednorazowym

- Zastosowanie dwa razy tego samego klucza nie jest bezpieczne!!!

$$C_1 \leftarrow m_1 \oplus \text{PRG}(k)$$

$$C_2 \leftarrow m_2 \oplus \text{PRG}(k)$$

- Atakujący wykonuje:

$$C_1 \oplus C_2 \rightarrow$$

- Jest wystarczająca liczba powtarzających się znaków, żeby:

$$m_1 \oplus m_2 \rightarrow m_1, m_2$$

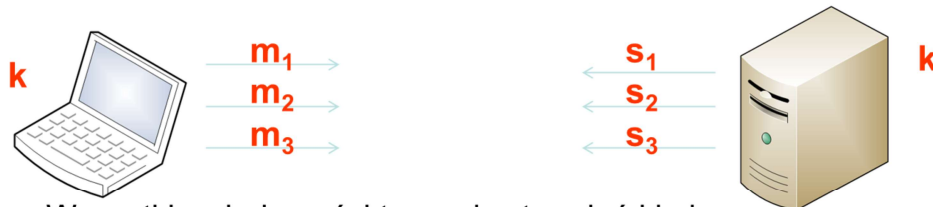
- **WNIOSEK: w szyfrze z kluczem jednorazowym i w szyfrach strumieniowych nie wolno dwa razy zastosować tego samego klucza!!!**

Rzeczywiste przykłady (1)

- Projekt Venona (Związek Radziecki, 1941-1946).
 - szyfrowanie z kluczem jednorazowym
 - klucz był uzyskiwany przez notowanie rzutem kostką
 - ponieważ było to pracochłonne, to używano tego samego klucza do kodowania wielu wiadomości
 - Rząd Stanów Zjednoczonych odczytał około 3000 zaszyfrowanych wiadomości

Rzeczywiste przykłady (2)

- MS-PPTP (Point to Point Tunneling Protocol Windows NT)



- Wszystkie wiadomości tworzyły strumień i były szyfrowane kluczem: $[m_1 || m_2 || m_3 || \dots] \oplus \text{PRG}(k)$
- Niestety odpowiedzi na wiadomości były szyfrowane tym samym kluczem: $[s_1 || s_2 || s_3 || \dots] \oplus \text{PRG}(k)$
- Taki system można zaatakować tak jak na slajdzie nr 15.
- Wniosek: Przy szyfrowaniu komunikacji Klient-Serwer i Serwer-Klient potrzeba stosować różne klucze**
- Klucz do komunikacji jest wtedy parą kluczy i obie strony muszą znać oba klucze.**

Rzeczywiste przykłady (3)

802.11b WEP:



- Ramka jest szyfrowana szyfrem strumieniowym z uzgodnionym kluczem k
- Klucz każdej ramki uzyskuje się łącząc go z 24 bitowym ciągiem IV
- Można sobie wyobrazić, że ten ciąg rozpoczyna się od 0 i co ramkę jest zwiększany o jeden
- Pomysł gwarantował zmianę klucza co ramkę. Odbiorca znał klucz i dostawał wraz z ramką wartość IV
- Problemem okazała się długość IV : jest tylko 2^{24} IV , w przybliżeniu 16 milionów i potem następuje reset IV .
- **Problem 1:** W dostatecznie długim strumieniu ramek są więc dwie zaszyfrowane tak samo
- **Problem 2:** Po resecie **niektórych** kart bezprzewodowej komunikacja startowała od nowa z $IV == 0$

29

Rzeczywiste przykłady (4)

– należy unikać „powiązanych ze sobą kluczy”

802.11b WEP:



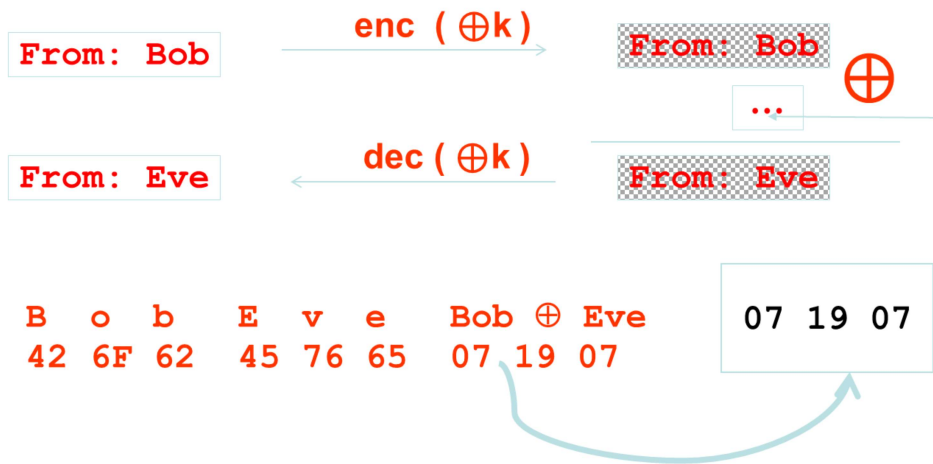
- Problem 3: kolejne klucze uzyskuje się przez połączenie stałego klucza i wartości IV zwiększonej o 1.
 - Klucze są do siebie podobne
- Problem 4: zastosowany algorytm do generowania liczb pseudolosowych RC4 był źle stosowany (udany atak opublikowany 2001 roku)

30

Pr. 3: PRG zastosowany w WEP nie generował bezpiecznych ciągów losowych, kiedy dane wejściowe niewiele się od siebie różniły.

Pr. 4: Znając właściwości szyfrowania strumienia danych w WEP (bliskie klucze) i algorytm RC4 jako PRG opublikowano atak pokazujący, że po podsłuchaniu 10^6 ramek można odkryć klucz. Późniejsze prace pokazały, że wystarczy tylko 40 000 ramek do przeprowadzenie takiego ataku (czas ataku: minuty!!!).

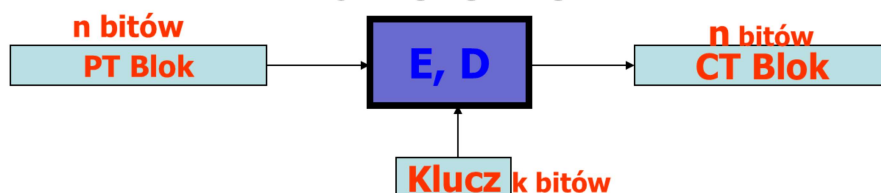
Atak 2: brak integralności (2)



Modyfikacja wiadomości zaszyfrowanej jest **niewykryta** i ma **przewidywalny wpływ na treść wiadomości**.

31

Szyfry blokowe – zasada działania



Popularne przykładowe algorytmy:

1. 3DES: $n = 64$ bitów, $k = 168$ bitów

2. AES: $n = 128$ bitów, $k = 128, 192, 256$ bitów

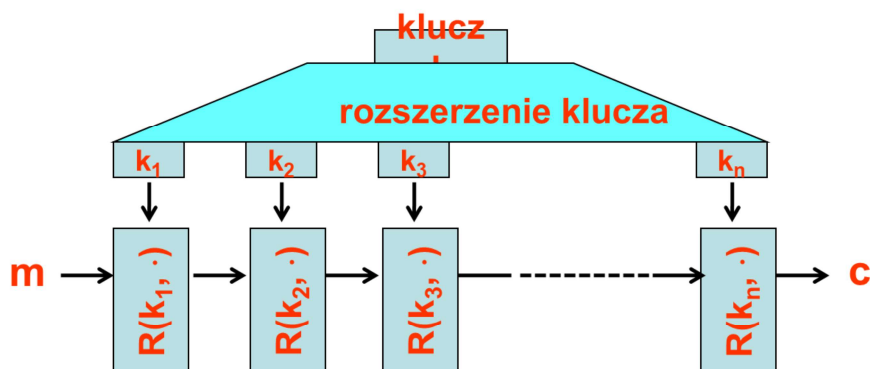
32

Szyfr blokowy składa się z dwóch algorytmów: E i D. Są to odpowiednio algorytmy szyfrujący i deszyfrujący. Oba algorytmy pobierają jako wejście klucz K. Zasada działania szyfru blokowego polega na tym, że bierze on N bitów danych do zaszyfrowania i wylicza dokładnie N bitów danych zaszyfrowanych. Następuje „zamapowanie” N bitów wejściowych na dokładnie N bitów wyjściowych.

W dolnej części slajdu zamieszczono dwa reprezentatywne przykłady szyfrów blokowych. Pierwszym z nich jest „potrójny DES” (3DES, Triple Data Encryption Algorithm). W algorytmie potrójny DES jednorazowo jest szyfrowany blok 64 bitów a szyfrowanie odbywa się z zastosowaniem klucza o długości 168 bitów (algorytm potrójny DES jest uznany za przestarzały od 2018 roku i jego stosowanie będzie zabronione od 2023 roku).

Algorytm AES (Advanced Encryption Standard) jest nowszy. Został przyjęty jako standard w 2001 roku. Posiada inne parametry niż 3DES. Szyfruje pojedynczy blok o długości 128 bitów. Dopuszcza się 3 długości klucza: 128, 192 lub 256 bitów. Co do zasady im dłuższy jest klucz, tym dłużej wykonuje się algorytm, ale tym trudniejszy do złamania (bardziej bezpieczny).

Szyfry blokowe są budowane z zastosowaniem iteracji



$R(k, m)$ jest nazywana funkcją rundy

dla 3DES ($n=48$), dla AES-128 ($n=10$)

33

Szyfry blokowe działają przez wykonanie szeregu iteracji. W pierwszej kolejności klucz K jest rozszerzany na klucze K_1 do K_n nazywane kluczami rundy. Następnie wiadomość do zaszyfrowania jest wielokrotnie szyfrowana przez funkcje rundy. Wejściem każdej z funkcji jest kolejny klucz K_n oraz wyjście poprzednie funkcji rundy. Pierwszym wejściem algorytmu jest oczywiście wiadomość do zaszyfrowania i pierwszy klucz. Po każdej rundzie otrzymuje się blok wyjściowy zawsze o takiej samej długości jak dane wejściowe. Liczba rund zależy od algorytmu. W 3DES jest 48 rund, w AES – 10.

Uwagi o bezpieczeństwie - nieformalnie

- Wykonywanie poszczególnych faz szyfru blokowego (funkcje rund) można nazwać generowaniem kolejnych pseudolosowych permutacji (permutacja - ustawianie elementów zbioru w pewnej kolejności)
- Bezpieczeństwo szyfrów blokowych opiera się na tym, że wygenerowane permutacje muszą być nierozróżnialne od ciągów losowych.

Porównanie wydajności

Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

	<u>Szyfr</u>	<u>Blok/roz. klucza</u>	<u>Szybkość (MB/sec)</u>
strumieniowy	RC4		126
	Salsa20/12		643
	Sosemanuk		727
blokowy	3DES	64/168	13
	AES-128	128/128	109

35

Porównanie wydajności poszczególnych algorytmów na tym samym komputerze pokazano na obecnym slajdzie. Można zauważyć, że szyfry blokowe są wolniejsze od strumieniowych. Jednak w dalszej części wykładu zostanie pokazane, że z zastosowaniem szyfrów strumieniowych pewne problemy mogą być rozwiązane bardziej efektywnie. Pozwalają one między innymi zagwarantować integralność i wiele innych właściwości wymaganych we współczesnych systemach ochrony danych.

Historia szyfru AES

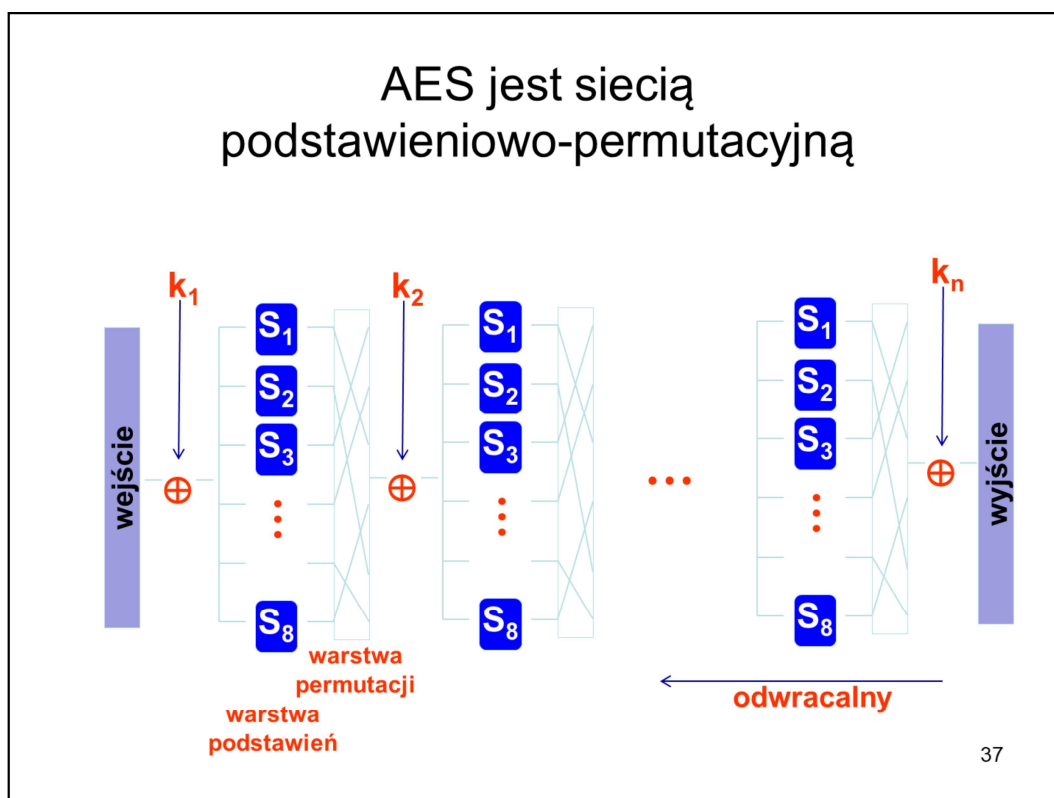
- 1997:NIST (National Institute of Standards and Technology) ogłasza konkurs na nowy standard szyfrowania blokowego
- 1998:15 zgłoszeń.
- 1999: NIST wybiera 5 finalistów
- 2000: NIST wybiera szyfr Rijndael jako AES (Advanced Encryption Standard) (zaprojektowany w Belgii)

Rozmiary kluczy: 128, 192, 256 bitów.

Rozmiary bloków: 128 bitów

36

Przez lata stosowania DES i 3DES okazało się, że te rozwiązania nie są zbyt 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-permutacyjna 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).

AES w implementacjach sprzętowych

Instrukcje AES w procesorach Intel Westmere (Core i3, Core i5, Core i7, Pentium, Celeron i 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

38

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.

Pełne przeszukiwanie klucza dla szyfrów blokowych

Dla dwóch par des DES $(m_1, c_1 = \text{DES}(k, m_1))$, $(m_2,$
 $c_2 = \text{DES}(k, m_2))$

prawdopodobieństwo znalezienia klucza $\approx 1 - 1/2^{71}$

Dla AES-128: przy posiadaniu dwóch par
klucz/szyfrogram, prawdopodobieństwo znalezienia klucza
 $\approx 1 - 1/2^{128}$

⇒ wystarczy dwie pary wiadomość - szyfrogram, żeby
znaleźć klucz

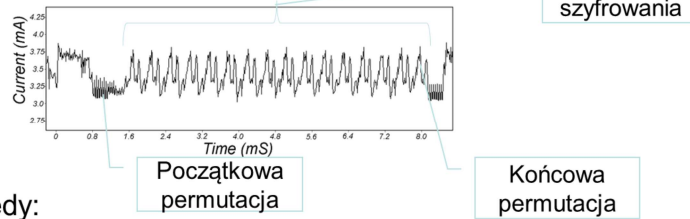
Atak siłowy w celu uzyskania klucza szyfru blokowego

- Zostało wykazane matematycznie, że dysponując dwiema parami: wiadomość/szyfr można dokonać ataku siłowego.
- Przeprowadzenie ataku siłowego oznacza więc odkrycie klucza na podstawie co najmniej dwóch par wiadomość – zaszyfrowana wiadomość
- Dokonuje się tego przez podawanie na wejście wszystkich możliwych kluczy.
- Problem jest do rozwiązania. Stawianym pytaniem jest w **jakim czasie i przy pomocy jakich środków** można tego dokonać.

Ataki na implementację

1. Ataki przez boczny kanał:

- Pomiar **czasu** podczas pracy algorytmu,
- Pomiar **mocy** podczas pracy algorytmu



2. Ataki na błędy:

- Spowodowanie błędu w ostatniej rundzie ujawnia ukryty klucz

⇒ Proszę nie **implementować** struktur kryptograficznych samodzielnie ...
Należy posługiwać się gotowymi bibliotekami/urządzeniami, których kod i implementacja były opublikowane i poddane atakom i te ataki były opublikowane

41

Ataki na szyfry mogą przebiegać nie tylko na drodze matematycznej i informatycznej. Pierwszy przykład ataku zakłada, że blokowy algorytm szyfrowania jest zaimplementowany na karcie chipowej. Karta może być zastosowana do np. do płatności. Może zawierać wbudowany ukryty klucz do uwierzytelnienia płatności podczas podłączenia karty do terminalu. Osoba atakująca może przejąć taką kartę i zabrać do laboratorium. Karta może być zastosowana do szyfrowania/deszyfracji jakiś danych. W warunkach laboratoryjnych można dokładnie zmierzyć np. czas potrzebny karcie do przeprowadzenia szyfrowania i deszyfrowania. Czas tych operacji zależy od długości zastosowanego klucza i w ten sposób można wejść w posiadanie pewnych informacji o kluczu, a nawet wyekstrahować klucz z karty. Jest wiele przykładów takich procedur, gdzie mierząc czas operacji algorytmów szyfrowania uzyskuje się tajny klucz.

Innym typem ataków są ataki na błędy (fault attacks). Atakując kartę chipową można doprowadzić do jej nieprawidłowego działania, np. poprzez przyspieszenie zegara, lub podgrzewanie. Zmusza się wtedy procesor do popełniania błędów i generowania błędnych danych wyjściowych. Okazuje się, że jeśli doprowadzi się do błędu w ostatniej rundzie szyfrowania blokowego z otrzymanego szyfrogramu można wyodrębnić klucz. Ochrona algorytmu szyfrującego przed takimi atakami polega na uzupełnieniu oprogramowania/sprzętu w taki sposób, że sprawdza się, czy otrzymano prawidłowy wynik. Nie jest to zagadnienie trywialne, choćby z tej przyczyny, że nie wiadomo, czy nie wystąpił błąd w algorytmie sprawdzającym poprawność wyniku...

Innym podejściem jest mierzenie ilości mocy konsumowanej przez kartę podczas jej pracy. Na podstawie pomiarów można opracować wykresy. Ponieważ karty nie są zbyt szybkie można w zasadzie zilustrować pobór mocy w każdym cyklu pracy karty jak to pokazano dla algorytmu DES na slajdzie. Na wykresie widać, kiedy algorytm wykonywał początkową permutację, następnie 16 rund szyfrowania blokowego, oraz końcową permutację. Powiększenie uzyskanego wykresu pozwala na odczytanie klucza bit po bicie.

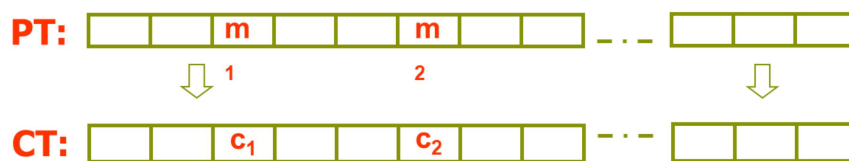
Nawet karty zabezpieczone przed takim atakiem (ograniczające ujawnienie konsumpcję mocy podczas obliczeń) nie są bezpieczne na inną odmianę ataku związanego z mierzaniem poboru mocy. Nazywa się on różnicowym atakiem z pomiarem mocy. W tym ataku dokonuje się pomiaru poboru mocy podczas bardzo wielu procesów kryptograficznych. Jeśli tylko istnieją bardzo niewielkie różnice w poborze prądu mogą one zostać wydobyte i posłużyć do odkrycia klucza. Te ataki zostały odkryte i opublikowane przez Paula Kochera i jego kolegów w instytucji Cryptography Research i obecnie dosyć pokaźną gałąź przemysłu związana z ochroną danych zajmująca się tylko zapobieganiu takim atakom.

Jeśli chodzi o ataki związane z czasem, to są one stosowane nie tylko na urządzeniach typu karty chipowe. Można sobie np. wyobrazić procesor wielordzeniowy, na którym na jednym rdzeniu wykonywany jest algorytm szyfrujący, a na drugim zainstalowano kod osoby atakującej. Rdzenie współdzieli ten sam obszar pamięci cache. Atakujący może mierzyć i podglądać przypadki nietrafienia pamięci podręcznej (cache misses) <<sytuacja, gdy program odwołuje się do pamięci cache, ale nie ma w niej potrzebnej danej lub instrukcji i musi ona być sprowadzona z innego poziomu pamięci (cache wyższego poziomu lub RAM)>> jakie wywołał algorytm szyfrujący. Okazuje się, że tylko śledząc nietrafienia pamięci cache można zgadnąć klucz stosowany przez algorytm. Jeden rdzeń może wydobywać informację od drugiego rdzenia obserwując tylko nietrafienia pamięci cache. W konsekwencji implementacje szyfrów blokowych są subtelnym zagadnieniem, które musi brać pod uwagę nie tylko siłowe ataki ale również ATAKI Z ZASTOSOWANIEM BOCZNEGO KANAŁU.

Powyższe przykłady ataku powinny przekonać słuchaczy tego kursu, że próby wymyślania własnych szyfrów blokowych i samodzielna implementacja algorytmów kryptograficznych to nie jest najlepszy pomysł. Po pierwsze trzeba wtedy zadbać, żeby w systemie kryptograficznym niemożliwe były ataki bocznymi kanałami. Po drugie implementacja musi być odporna na ataki na błędy. Dobrą praktyką jest stosowanie standardowych bibliotek, np. OpenSSL, czy innych.

Niewłaściwe użycie PRP

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



Problem:

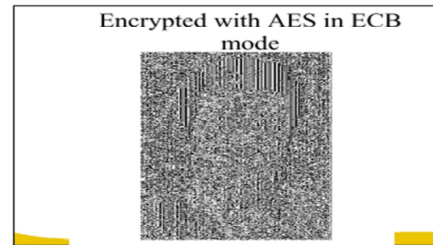
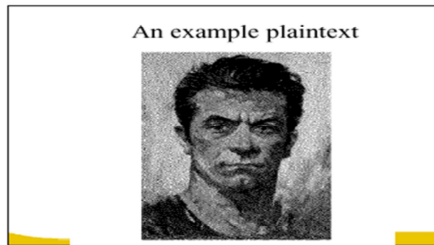
– **jeśli** $m_1 = m_2$ to $c_1 = c_2$

42

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 okaże 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)

43

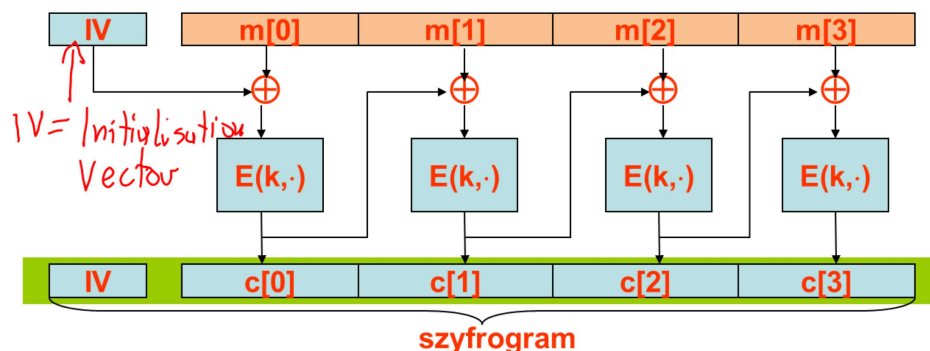
Jeśli nie przemawia do Państwa wcześniejsze „abstrakcyjne” omówienie problemu, proszę porównać obraz i jego kryptogram zaszyfrowany za pomocą elektronicznej książki kodowej. Być może nie da się rozpoznać twarzy, ale jej zarys tak oraz obszar, gdzie były kodowane włosy czy ciemna koszula. Dzieje się tak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

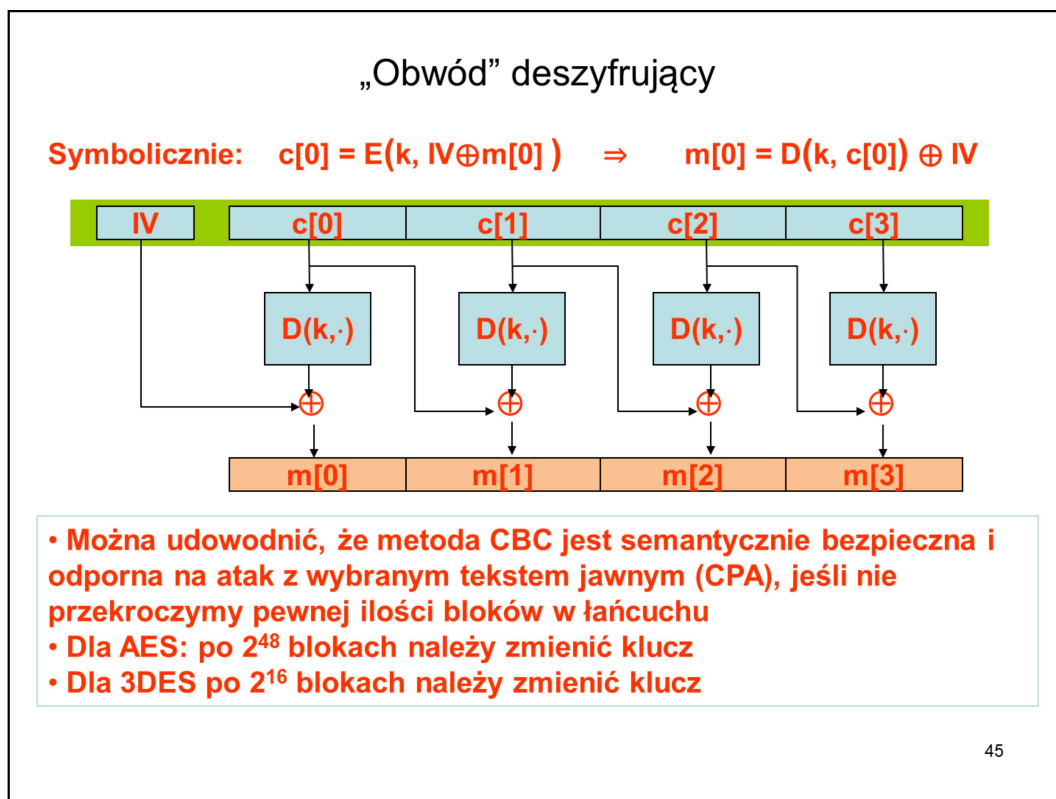
Konstrukcja I: CBC z losowym IV

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



44

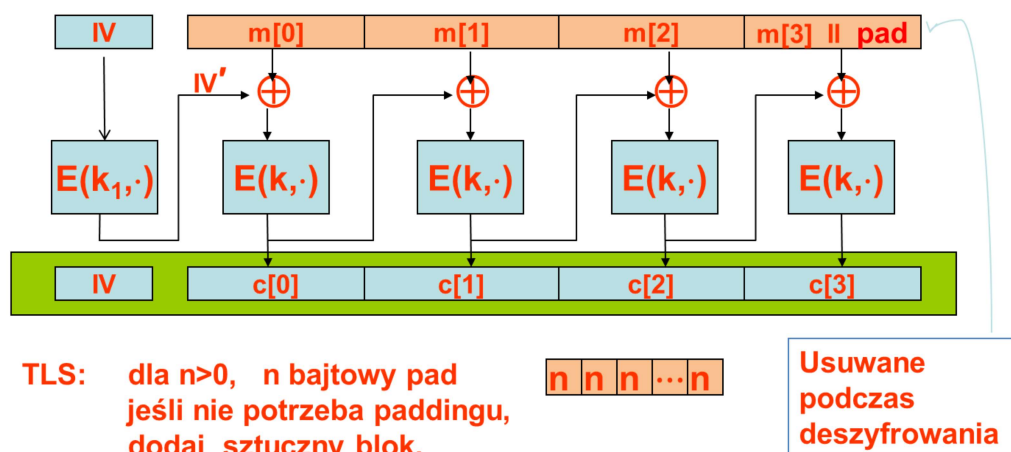
Szyfrowanie rozpoczyna się od wybrania jednego **losowego** bloku IV o długości pojedynczego bloku szyfru AES: 128 lub 192 lub 256). Na pierwszej części danych do zaszyfrowania jest wykonywany XOR z IV i ta wartość jest szyfrowana z zastosowaniem szyfru blokowego i otrzymujemy pierwszy blok szyfrogramu. 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.



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).

Techniczna uwaga do CBC - padding



46

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 wiadomości 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.

Ważna uwaga

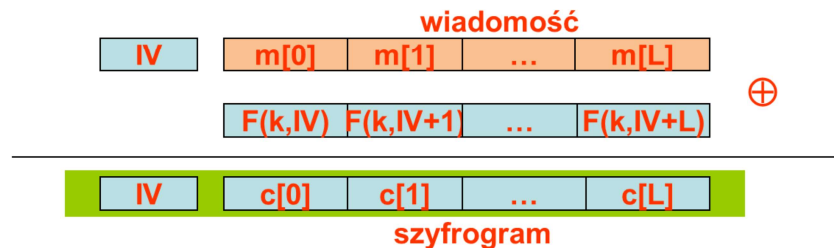
- CBC przestaje być bezpieczne, jeśli atakujący może przewidzieć wartość IV.
 - Istniał błąd w specyfikacji SSL/TSL 1.1, gdzie okazało się, że IV dla rekordu „i” był wartością ostatniego bloku poprzedniego szyfrogramu „i-1”.
 - Atakujący mając ostatni blok poprzedniego szyfrogramu mógł wyliczyć IV dla 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 niemożności przewidzenia kolejnego IV na podstawie poprzedniego.

47

Konstrukcja II: losowy tryb licznikowy

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

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



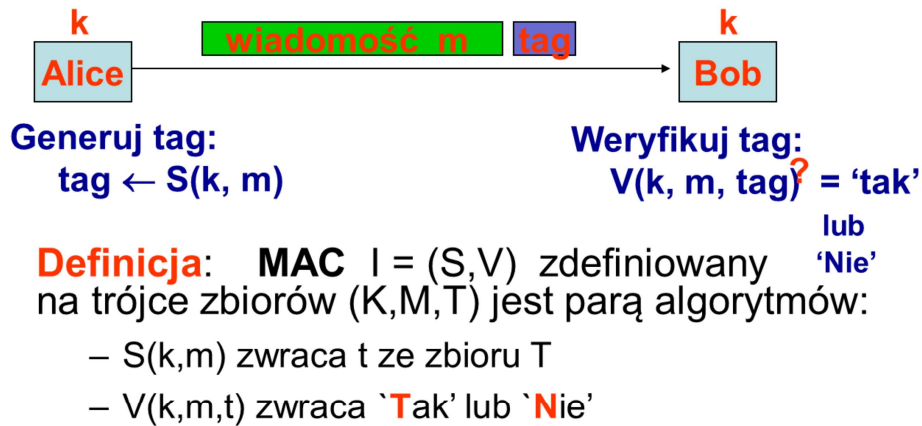
uwaga: możliwe do zrównoleglenia (w przeciwieństwie do CBC)

48

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 (CTR) 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.

Integralność wiadomości: MAC (ang. Message Authentication Code)



49

Zakładamy, że Alice i Bob chcą wymienić wiadomość i chcą, żeby zapewniona była dla niej integralność. Podstawowym mechanizmem w takich przypadkach jest zastosowanie MAC (ang. Message Authentication Code) – kodu uwierzytelniającego wiadomość.

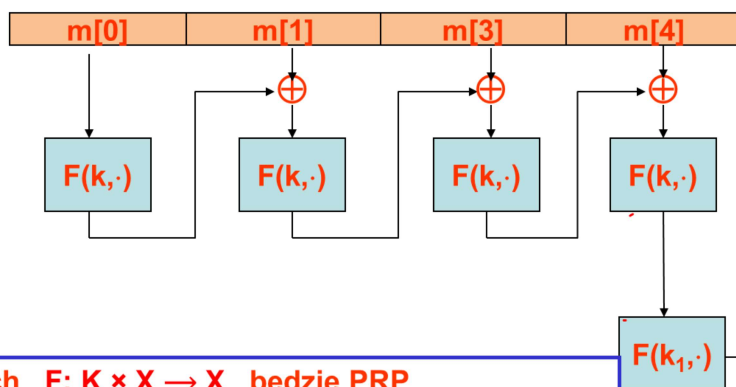
Bob i Alice dysponują współdzielonym, znanym tylko sobie tajnym kluczem k i chcą wymienić jawną wiadomość m . Chcą też, aby wiadomość nie została zmodyfikowana przez atakującego. Po stronie nadawcy (Alice) stosowany jest algorytm podpisujący S stosujący MAC: $S(k, m)$, który bierze jako wejście wiadomość i klucz. Generuje on krótki TAG (etykietę/identyfikator?), o długości 90 lub 100 bitów, nawet, jeśli wiadomość jest wielkości rzędu kilka GB. Następnie wiadomość i TAG są wysyłane do odbiorcy (Bob). Bob odbiera wiadomość i TAG i uruchamia algorytm weryfikujący V biorący jak wejście klucz k , wiadomość m i TAG tag . Algorytm zwraca „Tak”, jeśli wiadomość zachowała integralność, lub „Nie” jeśli stwierdzona została próba jej zmanipulowania.

Nieco bardziej formalnie: MAC (kod uwierzytelniający wiadomość) I jest parą algorytmów S (podpisujący) i V (weryfikujący). Są one zdefiniowane na zbiorach możliwych kluczy, wiadomości i możliwych tagów. Algorytm podpisujący generuje na podstawie klucza i wiadomości tag, a algorytm weryfikujący sprawdza na podstawie wiadomości, klucza i tagu integralność wiadomości i zwraca „Tak” w przypadku sukcesu lub „Nie” w przypadku porażki.

Dla każdego klucza i dla każdej wiadomości jeśli policzymy zależność: $V(k, m, S(k, m))$ otrzymamy zawsze „Tak” (warunek spójności, podobny do warunku z szyfrowaniem i deszyfrowaniem we wcześniejszych wykładach $\ll D(k, E(k, m)) = m \gg$).

Szyfrowany CBC-MAC

(ECBC)



$X^{\leq L} = \bigcup_{i=L}^L x^i$
 $\Downarrow$
 system akceptuje
 różne dł. wiad.

Niech $F: K \times X \rightarrow X$ będzie PRP

Definiujemy nowy PRF $F_{\text{ECBC}}: K^2 \times X^{\leq L} \rightarrow X$

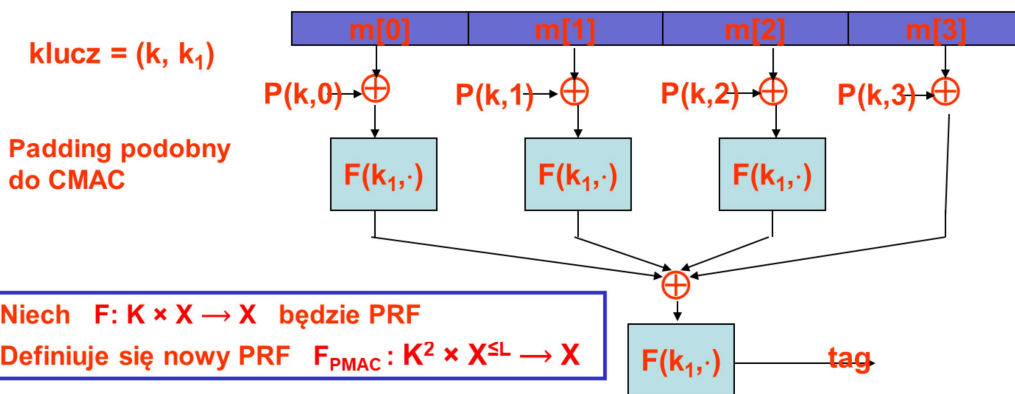
50

Pierwszą prezentowaną konstrukcją CBC dla długich wiadomości jest szyfrowany CBC-MAC, lub ECBC w skrócie. ECBC bierze wiadomość podzieloną na n bloków i zwraca tag o długości pojedynczego bloku. Schemat generowania MAC dla wiadomości jest podobny do szyfrowania zgodnie z CBC (por. poprzedni wykład). Różnica jest taka, że po zaszyfrowaniu kolejnych bloków nie następuje generowanie szyfrogramu, tylko przekazanie zaszyfrowanego bloku na wejście kolejnego bloku szyfrującego. Nie ma też szyfrowania wartości początkowej (IV). Natomiast następuje zaszyfrowanie wyjścia łańcucha za pomocą osobnego klucza k_1 . Wyjściem jest tag o długości odpowiadającej wyjściu pojedynczej instancji szyfru blokowego. Można go jeszcze skrócić, jeśli wymaga tego implementacja (oczywiście tylko do długości uznawane za bezpieczną (80, 90, 64? bitów)). Wygenerowanie ECBC wymaga podania na wejście 2 kluczy oraz wiadomość i zwraca ciąg danych o ustalonej długości.

Skomentowania wymaga zastosowanie ostatniego dodatkowego szyfrowania z innym kluczem. Łańcuch szyfrowań za pomocą pierwszego klucza nazywa się surowym ECBC. Bez tego ostatniego szyfrowania surowe ECBC nie jest bezpieczne.

PMAC (ang. Parallel MAC)

$P(k, i)$: łatwa do wyliczenia na komputerze funkcja



Bezpieczeństwo:

PMAC jest bezpieczny, dopóki: $q \cdot L \ll |X|^{1/2}$

51

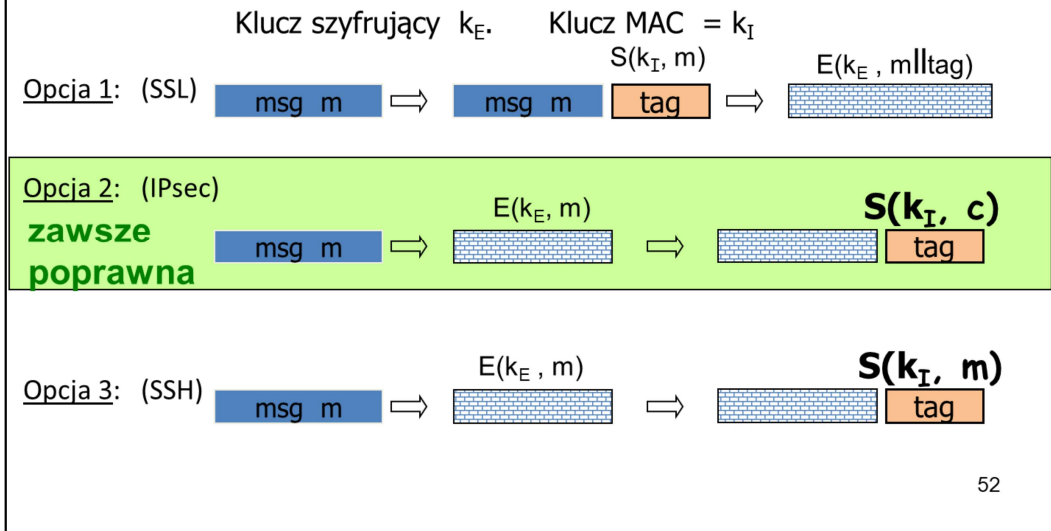
Współbieżne obliczenie MAC oferuje konstrukcja o nazwie PMAC. Podobnie jak wcześniej wiadomość dzielona jest na bloki. Ale tym razem każdy blok wiadomości jest przetwarzany osobno (w oderwaniu od pozostałych bloków). Dla każdego bloku wiadomości następuje wykonanie operacji xor na tym bloku i na rezultacie działania pewnej funkcji P , której wejściem jest klucz szyfrowania k oraz licznik. Tak przetworzony blok wiadomości przekształcany jest przez funkcję F (PRF), której argumentem jest inny klucz k_1 . Na rezultatach przetworzenia wszystkich bloków wiadomości wykonywana jest operacja xor i podawana na jeszcze jeden blok szyfrowania. Rezultatem jest obliczony tag (MAC). Z technicznego względu nie ma potrzeby przekształcania ostatniego bloku przez funkcję F .

Jak działa funkcja P ? Jeśli wyobrazimy sobie konstrukcję bez funkcji P , to okazuje się, że obliczony MAC nie jest bezpieczny. Zwykle przestawienie bloków w takiej konstrukcji dałoby taki sam MAC. W rezultacie atakujący mógłby zażądać obliczenia MAC dla wiadomości, potem przestawić w niej bloki i uzyskać taki sam MAC, co przeczy pojęciu bezpieczeństwa MAC.

To co funkcja P zapewnia, to utrzymanie kolejności bloków danych. Wejściami kolejnych wywołań funkcji jest klucz oraz numer bloku. W rezultacie ciąg wygenerowany przez funkcję P jest za każdym razem inny dla kolejnego bloku wiadomości. Struktura funkcji jest bardzo prosta do wykonania na komputerze. Jest iloczynem pewnego ograniczonego zbioru pól danych. Przy czym nie narusza ona bezpieczeństwa konstrukcji PMAC.

Konstrukcja wymaga zastosowania dwóch kluczy k i k_1 . Jednego stosowanego do funkcji P , a drugiego do F . Rozwiązanie paddingu jest podobne do tego pokazanego w konstrukcji CMAC. Nie ma więc tu potrzeby stosowania dodatkowego „atrapowego” bloku danych.

Łączenie MAC i szyfrowania (CCA)



Na slajdzie pokazano trzy przykłady kombinacji szyfrowania i MAC. W każdym z nich stosuje się osobny klucz do szyfrowania i osobny klucz do generowania MAC. Klucze są niezależne od siebie i są generowane podczas inicjalizacji sesji szyfrowania/uwierzytelniania. Sposób generowania kluczy będzie pokazany później. Pierwszy przykład to protokół SSL. W protokole w pierwszym kroku oblicza się MAC dla wiadomości z zastosowaniem klucza k_I , a następnie wiadomość i dołączony do niej TAG szyfruje się z zastosowaniem algorytmu szyfrującego. W drugim rozwiązaniu (IPsec) najpierw szyfruje się wiadomość, a następnie z szyfrogramu oblicza się MAC i dołącza do szyfrogramu. Trzecie rozwiązanie stosowane w protokole SSH polega na zaszyfrowaniu wiadomości, a następnie dołączenie do niej MAC obliczonego z niezaszyfrowanej wiadomości. Pokazaliśmy trzy różne rozwiązania łączące szyfrowanie i MAC. Powstaje pytanie, które z nich jest bezpieczne.

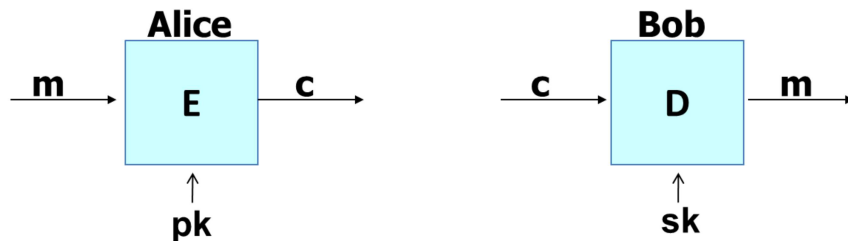
Rozważmy rozwiązanie zastosowane w SSH (3 opcja). Tutaj MAC jest obliczany dla niezaszyfrowanej wiadomości i dołączany do szyfrogramu. Tu pojawia się problem, ponieważ MAC nie są konstruowane do zachowania poufności (do szyfrowania wiadomości). Dołączamy wtedy do szyfrogramu jakąś informację na temat niezaszyfrowanego tekstu. Chociaż więc konstrukcja SSH jest uznana za poprawną nie zaleca się w nowych rozwiązaniach stosowania takiej kombinacji szyfrowania i MAC. Rozważmy pozostałe dwie opcje. Warto wiedzieć, że rekomendowanym rozwiązaniem jest to zastosowane w IPsec. Zaletą rozwiązania jest to, że niezależnie jakie szyfrowanie i jaką metodę generowania MAC zastosujemy, to zawsze otrzymamy rozwiązanie bezpieczne (szyfrowanie z uwierzytelnieniem). Dlaczego? Najpierw szyfrujemy wiadomość. Potem generujemy MAC dla szyfrogramu. W taki sposób zamykamy możliwość wygenerowania innego szyfrogramu, który byłby prawidłowy. Każda modyfikacja szyfrogramu będzie wykryta. Dla konstrukcji zastosowanej w SSL jest kilka patologicznych przykładów kiedy kombinacja szyfrowania odpornego na atak z wybranym tekstem (CPA) i MAC jest podatna na atak z wybranym szyfrogramem. Mogą się pojawić pewne złe interakcje pomiędzy schematem szyfrowania i algorytmem generującym MAC, które można zaatakować atakiem z wybranym szyfrogramem (CCA).

W nowych rozwiązaniach należy więc stosować najbezpieczniejszą kombinację: szyfrowanie wiadomości, a następnie obliczanie MAC dla szyfrogramu.

Stąd nie zawsze jest uznawana za poprawne rozwiązanie szyfrowania z uwierzytelnieniem.

Kryptografia klucza publicznego

Bob: generuje (PK, SK) i przekazuje PK do Alice



53

Czym jest kryptografia klucza publicznego? Podobnie jak w przypadku szyfrowania symetrycznego dysponujemy dwoma algorytmami: szyfrującym (E) i deszyfrującym (D). Tym razem do szyfrowania stosujemy jeden klucz, nazywany kluczem publicznym. W naszym przypadku nazwijmy go pk , ale do odszyfrowywania stosujemy inny klucz, nazywany kluczem tajnym (sk). Te dwa klucze tworzą parę kluczy. Szyfrowanie polega na uruchomieniu algorytmu szyfrującego wiadomość za pomocą klucza publicznego. Odszyfrowywanie polega na uruchomieniu algorytmu odszyfrowującego z zastosowaniem klucza tajnego.

Wyprowadzenie kryptografii klucza publicznego z funkcji zapadkowych

- (G, F, F^{-1}) : bezpieczna funkcja **zapadkowa** $X \rightarrow Y$
- (E_s, D_s) : **system szyfrowania symetrycznego z uwierzytelnieniem** (K, M, C)
- $H: X \rightarrow K$ funkcja hash

$E(pk, m)$: R

$x \leftarrow X, \quad y \leftarrow F(pk, x)$
 $k \leftarrow H(x), \quad c \leftarrow E_s(k, m)$
wyjście (y, c)

$D(sk, (y, c))$:

$x \leftarrow F^{-1}(sk, y),$
 $k \leftarrow H(x), \quad m \leftarrow D_s(k, c)$
wyjście m

54

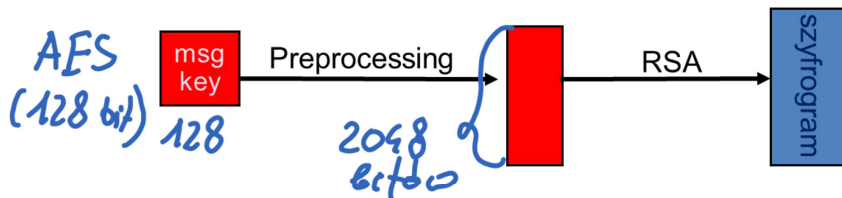
Wejściem podsystemu szyfrującego jest klucz publiczny (pk) i wiadomość (m). Najpierw generujemy losową wartość ze zbioru X , a potem stosujemy funkcję zapadkową F z kluczem pk do obliczenia wartości y . y jest przekształceniem wartości x przez funkcję F z kluczem pk . Teraz z kolei przekształcamy z zastosowaniem funkcji hash wartość x na klucz k (klucz symetryczny do „szybkiej” komunikacji szyfrowanej). Na koniec szyfrujemy wiadomość szyfrem symetrycznym z otrzymanym kluczem k . Wyjściem algorytmu są szyfrogram i wartość y . Zwróćmy uwagę, że funkcja zapadkowa jest zastosowana tylko do przekształcenia losowej wartości x w y . Szyfrowanie zaś odbywa się z zastosowaniem szyfru symetrycznego.

Rozważmy proces odszyfrowywania. Algorytm deszyfrujący bierze jako wejście klucz sekretny i parę wartości y i c . Deszyfrowanie rozpoczyna się od odwrócenia wartości y z zastosowaniem funkcji zapadkowej i klucza sk . Na podstawie otrzymanej wartości x z zastosowaniem funkcji hash oblicza się klucz. Algorytm deszyfrujący kryptografii symetrycznej odszyfrowuje szyfrogram z zastosowaniem „odzyskanego” klucza k .

Kryptografia klucza publicznego w praktyce

Nie stosujemy RSA do bezpośredniego szyfrowania danych czy kluczy!

RSA w praktyce (ponieważ standard ISO nie jest często stosowany) :

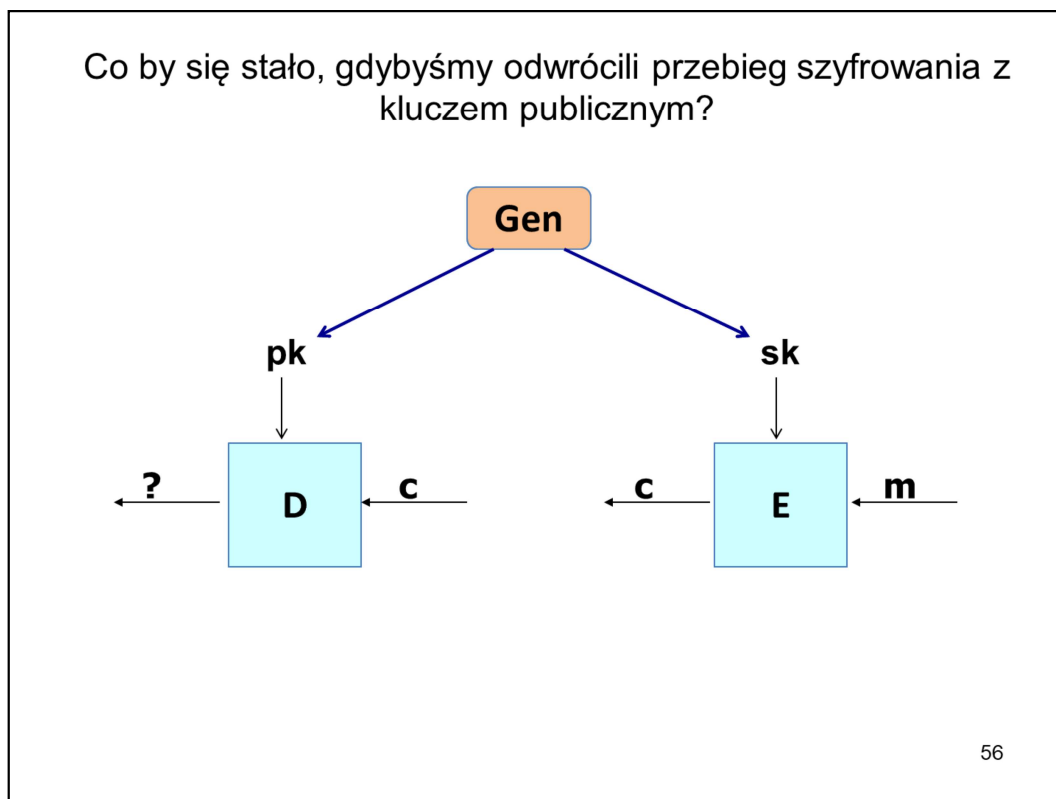


Główne pytania:

- Jak powinien być przeprowadzony „preprocessing”?
- Czy możemy udowodnić bezpieczeństwo takiego systemu?

55

Podany wcześniej ustandaryzowany system szyfrowania z kluczem publicznym rzadko jest używany w powiązaniu z RSA. RSA jest stosowany do czegoś innego. Zwykle system „osobno” generuje klucz kryptografii symetrycznej, a system RSA jest „proszony” o zaszyfrowanie tego klucza. Przykładowo, tworzymy 128-bitowy klucz AES, rozszerzamy do długości 2048 bitów i stosujemy RSA do zaszyfrowania takiego klucza. Powstaje pytanie, jak powinno zachodzić takie przekształcenie? Drugą kwestią jest, czy takie podejście jest bezpieczne?



Na początek zastanówmy się, co by się stało, gdybyśmy dla odmiany zaszyfrowali jakąś wiadomość za pomocą naszego klucza sekretne i pozwolili na jej odszyfrowanie za pomocą klucza publicznego? Na pierwszy rzut oka takie działanie wydaje się dziwne, ale po chwili zastanowienia możemy przeprowadzić następujące rozumowanie. Bob szyfruje wiadomość swoim kluczem sekretnym. Praktycznie każdy (w tym Alice) może tę wiadomość odszyfrować. Uzyskujemy tu jednak nową cechę: ponieważ tylko Bob mógł zaszyfrować tę wiadomość, to znaczy, że pochodzi ona **tylko od niego**, co więcej odpowiednio zorganizowany system szyfrowania może zapewnić **integralność** wiadomości oraz **niezaprzeczalność**, że jest się autorem wiadomości.

Atak na taki system polega na próbie podszycia się pod nadawcę wiadomości i oszukania odbiorcy, że sfałszowana wiadomość pochodzi od danej osoby i zawiera określoną treść. Bezpieczny system podpisu elektronicznego powinien zapobiegać takim atakom.

Znaczenie podpisu elektronicznego

- Weryfikowalność podpisu zależy do możliwości niezawodnego i możliwego do zweryfikowania udostępnienia klucza publicznego.
- Taka procedura jest kosztowna, także ze względów organizacyjnych, ale zwykle wykonywana jest tylko raz, a potem każdy łatwo może uzyskać klucz i zweryfikować podpis.
- Okazuje się, że właśnie schemat podpisu elektronicznego sam w sobie jest doskonałym mechanizmem dystrybucji kluczy publicznych.
- Tak więc publikowanie kluczy publicznych odbywa się z zastosowaniem podpisów elektronicznych i jest podstawą do tzw. Infrastruktury Klucza Publicznego (PKI – Public Key Infrastructure)

57

Definicja podpisu elektronicznego

Podpis elektroniczny: (Gen, Sign, Vrfy)

- Gen: algorytm generacji pary kluczy (pk i sk)

- Sign: algorytm podpisujący
 $(m, ds) \leftarrow \text{Sign}(sk, m)$

- Vrfy: algorytm weryfikujący
 $\text{Vrfy}(pk, m, ds)$

Zwraca 1, gdy podpis jest prawidłowo zweryfikowany, 0
gdy weryfikacja się nie udała.

Dla wszystkich legalnych wiadomości zachodzi:

$$\text{Vrfy}(pk, m, \text{Sign}(sk, m)) = 1.$$

58

Zasada wykonywania podpisu elektronicznego wygląda następująco. Osoba podpisująca generuje parę kluczy (sk i pk) [tak jak w kryptografii klucza publicznego: jeden klucz służy do szyfrowania, a drugi do odszyfrowywania]. Klucz publiczny osoby podpisującej jest udostępniany. Jeśli osoba chce podpisać jakąś wiadomość szyfruje ją z zastosowaniem klucza sekretnego sk i wysyła do odbiorcy parę (m, ds). Odbiorca znając klucz publiczny może odszyfrować podpis i sprawdzić, czy otrzymana wartość jest identyczna z nadesłaną wiadomością m. Wtedy dowiadujemy się 2 rzeczy: 1) wiadomość była od konkretnego nadawcy, 2) nie została zmodyfikowana (odszyfrowana wiadomość jest identyczna z nadesłaną).

Dalej, podobnie jak w przypadku MAC, system nie jest odporny na ataki powtórzeniowe.

W systemie zakłada się również, że istnieje jakiś mechanizm pewnego przekazywania kluczy. To znaczy, że odbiorca otrzymując klucz publiczny jest pewny, że jest to klucz od konkretnej osoby.

Jeśli potrafimy w sposób pewny dostarczyć właściwy klucz publiczny, to po co w ogóle tworzyć podpisy? Okazuje się, że takie pewne dostarczenie kluczy jest procesem trudnym i kosztownym, ale w schemacie podpisu elektronicznego musi być wykonane tylko raz. Potem można przekazywać podpisy nieskończoną liczbę razy.

Poza tym mechanizm podpisu elektronicznego służy do efektywnego przekazywania samych kluczy publicznych (np. PKI – public key infrastructure)

Certyfikat

- Załóżmy, że Charlie wygenerował parę kluczy (pk_C , sk_C)
- Załóżmy też, że Bob wygenerował parę kluczy (pk_B , sk_B)
- Załóżmy, że Charlie wie, że pk_B jest kluczem publicznym Bob'a
- Wtedy Charlie może obliczyć podpis:
 $cert_{C \rightarrow B} = \text{Sign}[sk_C, (\text{"To jest klucz Boba"}, pk_B)]$
i odesłać ten podpis Bob'owi.
- Nazywamy $cert_{C \rightarrow B}$ certyfikatem dla klucza publicznego Bob'a wystawionym przez Charliego.
- W praktyce certyfikat powinien jednoznacznie identyfikować Bob'a, czyli zawierać jego imię i nazwisko, email, stronę WWW i inne atrybuty

59

Pokazaliśmy wybrane bezpieczne konstrukcje stosujące kryptografię z kluczem publicznym. Do tej pory zakładaliśmy, że klucze publiczne są efektywnie rozdystrybuowane, co więcej odszyfrowanie sygnatury za pomocą klucza publicznego miało także potwierdzić, że wiadomość podpisała konkretna osoba (niezaprzeczalność). Okazuje się, że kryptografia klucza publicznego może również posłużyć do efektywnej i niezaprzeczalnej metody dystrybucji kluczy publicznych. To co zostanie pokazane, to że jeden klucz publiczny należący do instytucji zaufanej i rozdystrybuowany w sposób bezpieczny może „uruchomić” bezpieczną dystrybucję wielu innych kluczy publicznych. Zwłaszcza, że problem bezpiecznej dystrybucji kluczy musi być rozwiązany tylko raz.

Kluczowym pojęciem jest tutaj cyfrowy certyfikat, który w istocie jest podpisem łączącym klucz publiczny z jakąś osobą/tożsamością.

Weryfikacja klucza publicznego na podstawie certyfikatu

- Teraz Bob chce skomunikować się z inną osobą/instytucją, np. Alice, która już zna pk_C .
- Może on wysłać do Alice wiadomość $(pk_B, cert_{C \rightarrow B})$, która może zweryfikować, że pk_B rzeczywiście należy do Bob'a, o czym poświadcza pk_C .
- Jeśli sprawdzenie podpisu się udało, to Alice wie, że Charlie podpisał klucz Bob'a
- Jeśli Alice ufa Charlie'mu, to może zaakceptować pk_B jako legalny klucz
- Cała wymiana potwierdzająca pk_B może się odbyć w publicznym i niechronionym kanale komunikacyjnym
- Dopóki Charlie (klucz prywatny Charlie'go) nie zostanie „skompromitowany” tak zweryfikowane klucze są uznawane za legalne i powiązane z daną osobą/instytucją.

Kilka pominiętych detali

- Skąd Alice zna pk_C i mu ufa?
- Skąd Charlie wie, że pk_B należy do Boba?
- Jak Alice decyduje, czy ufać Charlie'mu?
- Pełna specyfikacja, jak rozwiązać pokazane problemy opisana jest w dokumentach dotyczących Infrastruktury Klucza Publicznego (PKI – Public Key Infrastructure)