Kryptografia i bezpieczeństwo danych
- Integralność I

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

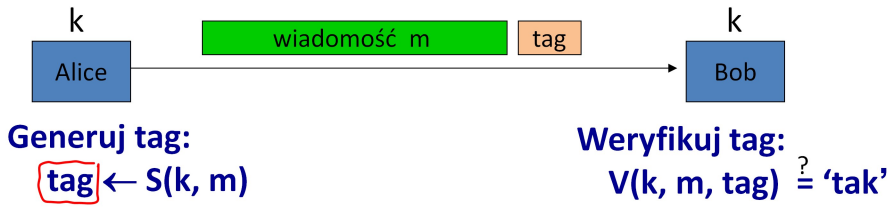
1

Założenia wykładu

- Omawiane będą techniki zapewnienia integralności danych, ale bez ich poufności (połączenie tych zagadnień będzie wprowadzone później).
- Przykłady zastosowań
  - Ochrona danych na dysku (nie chcemy ich szyfrować, ale nie chcemy też, żeby były zmodyfikowane przez wirusa)
  - Ochrona reklam na stronach internetowych (Wytwórca reklam nie martwi się o ich skopiowanie i zamieszczenie gdzie indziej, ale martwi się, żeby ich treść bez jego pozwolenia nie uległa modyfikacji)

2

Integralność wiadomości: MAC (ang. Message Authentication Code)



Generuj tag:
tag ← S(k, m)

Weryfikuj tag:
V(k, m, tag) = ? 'tak'
lub
'Nie'

Definicja: **MAC** I = (S,V) zdefiniowany na trójce zbiorów (K,M,T) jest parą algorytmów:

- S(k,m) zwraca t ze zbioru T
- V(k,m,t) zwraca `Tak' lub `Nie'

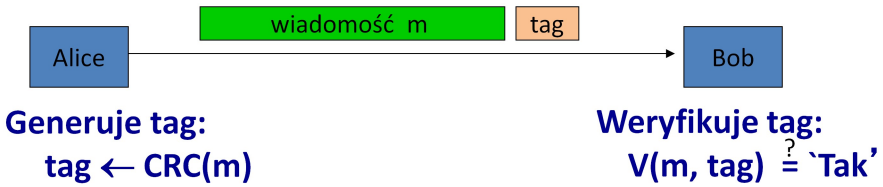
3

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 sprawcza 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 << D(k,E(k,m))=m >>.

Zapewnienie integralności wymaga tajnego klucza



- Atakujący może łatwo zmodyfikować wiadomość i ponownie przeliczyć CRC.
- CRC jest zaprojektowane do wykrywania **przypadkowych** błędów, a nie błędów wprowadzanych rozmyślnie.

4

Należy zwrócić uwagę, że zapewnienie integralności jest możliwe TYLKO, gdy stosujemy rozwiązanie ze współdzielonym kluczem. Na pierwszy rzut oka do zapewnienia integralności możemy zastosować CRC (ang. Cyclic Redundancy Code/Cyclic Redundancy Check) cykliczny kod nadmiarowy/cykliczna kontrola nadmiarowa. Jest to klasyczny algorytm do wykrywania losowych błędów w wiadomości. On też w rezultacie generuje ciąg bitów, który jest „skrótem" wiadomości, a sprawdzenie poprawności wiadomości polega na obliczeniu jej CRC u odbiorcy i porównanie z nadesłanym CRC. Jeśli oba CRC są identyczne, to z bardzo dużym prawdopodobieństwem w wiadomości nie ma błędów.

Rozważmy zastosowanie CRC w zapewnieniu integralności. Niech Alice zamiast generować tag za pomocą klucza, wygeneruje CRC wiadomości (CRC nie wymaga klucza), a następnie dołączy CRC do wiadomości i ją wyśle. Bob odbierze wiadomość i CRC. CRC się zgadza, więc wiadomość jest uznana za niezmodyfikowaną. Zwróćmy jednak uwagę, że taka wiadomość może zostać przejęta przez atakującego, zmodyfikowana, dla niej zostanie policzone CRC i przesłane Bobowi. Odbiorca (Bob) nie może w takim przypadku zauważyć przeprowadzonej próby modyfikacji wiadomości.

Tak się stało, ponieważ nie zastosowano tajnego klucza. Gdy w generowaniu tagu uczestniczy klucz, atakujący nie może tak łatwo zmodyfikować treść jawnej wiadomości. CRC jest zaprojektowane do wykrywania losowych błędów, a nie celowo wprowadzanych przez atakującego.

Bezpieczne MAC

Możliwości atakującego: **atak z wybraną wiadomością**

- dla $m_1, m_2, \ldots, m_q$ atakujący ma $t_i \leftarrow S(k, m_i)$

Cel atakującego: **egzystencjalne fałszerstwo**

- przygotowanie **<u>nowej</u>** „sprawdzalnej” pary wiadomość/tag (m,t).

$$(m,t) \notin \{ (m_1,t_1) , \ldots , (m_q,t_q) \}$$

Co musimy zapewnić:

⇒ atakujący nie może wygenerować „sprawdzalnego” tagu dla nowej wiadomości

⇒ mając (m,t) atakujący nie może nawet spreparować (m,t’) dla $t' \neq t$

5

Jeśli rozważamy bezpieczeństwo z MAC, to musimy rozważyć atak o następującym scenariuszu. Atakujący może przesłać do Alice serię wiadomości i dla każdej otrzyma wygenerowany tag. (Alice zapisuje email od atakującego na dysku, w którym ma włączony mechanizm zachowywania integralności plików. Dysk jest potem kradziony. Wtedy atakujący ma swoje wiadomości wraz z tagami dla nich wygenerowanymi.) To, co będzie próbował wykonać atakujący, to wygenerowanie nowej pary wiadomość/tag, innej niż te przechwycone, ale takiej, która będzie zweryfikowana przez system jako poprawna (będzie wtedy można wysyłać sfałszowane wiadomości). Ważnym warunkiem jest, aby atakujący nie mógł wygenerować ŻADNEJ pary wiadomość/tag, nawet takiej, gdzie wiadomość jest jakimś bełkotem. W systemie musimy mieć absolutną pewność, że wiadomość nie zostanie w jakikolwiek sposób podmieniona. Wtedy mechanizm sprawdzania integralności wiadomości będzie mógł być z powodzeniem stosowany do wymiany kluczy szyfrowania, które wyglądają jak losowe ciągi bitów (czyli bełkot…). Podsumowując, dla zachowania bezpieczeństwa atakujący nie będzie mógł wygenerować żadnego prawidłowego tagu. Drugą ważną właściwością bezpiecznego systemu jest niemożliwość wygenerowania nowego „spreparowanego” tagu dla tej samej wiadomości, który mógłby być prawidłowo zweryfikowany przez system.

Ważne wybrane właściwości sytemu MAC, które muszą być spełnione, aby zapewnić bezpieczeństwo

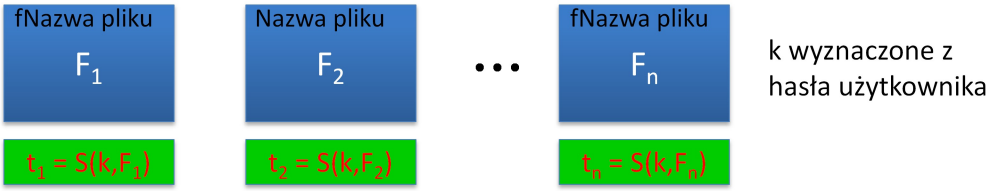
- System nie może dopuścić do zdarzenia, w którym: $S(k,m_0) = S(k,m_1)$
- Tagi nie mogą być zbyt krótkie, bo można by przeprowadzić na system atak podobny do siłowego (jest możliwość łatwego zgadnięcia tagu, jeśli jest dostatecznie krótki), typowo tagi mają długość 64, 96 lub 128 bitów. TLC stosuje tagi 96 bitowe.

6

Na slajdzie zawarto ważne spostrzeżenia dotyczące budowania bezpiecznych systemów z MAC. System nie może być tak skonstruowany, że dwie różne wiadomości dają takie same tagi. Same tagi nie mogą być zbyt krótkie, aby nie można było na nich przeprowadzić swego rodzaju ataku siłowego (próby sprawdzenia wszystkich możliwych par wiadomość/tag).

Przykład: ochrona systemu plików

Zakładamy, że podczas instalacji systemu operacyjnego następuje przetwarzanie:



Później wirus infekuje system i modyfikuje pliki systemowe

Użytkownik startuje z czystego OS i podaje hasło

- Wtedy: dysponując bezpiecznymi MAC dla plików możemy wskazać wszystkie zmodyfikowane pliki

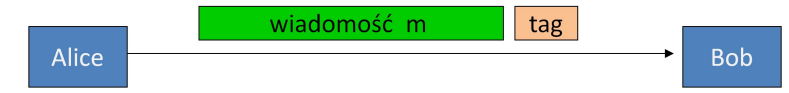
7

Zastanówmy się, jak system MAC mógłby posłużyć do ochrony naszego systemu plików. Załóżmy, że właśnie instalujemy system operacyjny np. Windows. Jednym z elementów instalacji jest podanie hasła. Na podstawie hasła jest generowany klucz k. Dla każdego pliku tworzonego na dysku w czasie instalacji (F_1, F_2, …, F_n) system operacyjny z zastosowaniem klucza k generuje tag (t_1, t_2, …, t_n) i przechowuje go razem z plikiem. Po instalacji klucz jest usuwany.
Załóżmy, że po jakimś czasie komputer jest zainfekowany i wirus próbuje modyfikować pliki systemowe. Powstaje pytanie, czy użytkownik jest w stanie stwierdzić, które pliki zostały zmodyfikowane.
Należy wtedy wystartować system z „czystego” nośnika (USB). Po uruchomieniu systemu można wpisać podane wcześniej hasło. Nowy system może sprawdzić MAC dla każdego z plików systemowych na twardym dysku. Wirus nie zna klucza, więc nie może wygenerować właściwego tagu po modyfikacji pliku. Jedyne, co może zrobić, to zamienić pliki nazwami. Wtedy system może być uszkodzony. Z tym problemem również może sobie poradzić MAC, pod warunkiem, że do obliczania tagu wzięta będzie również nazwa pliku systemowego. Wtedy zamiana nazw również zostanie wykryta, ponieważ wirus nie jest w stanie obliczyć prawidłowego tagu.

Generowanie bezpiecznych MAC z zastosowaniem bezpiecznego PRF (ang. Pseudo Random Function)

Dla PRF **F: K × X ⟶ Y** definiujemy MAC $I_F = (S,V)$ jako:

- S(k,m) := F(k,m)
- V(k,m,t): zwraca `Tak' jeśli t = F(k,m) i `Nie' w przeciwnym wypadku.



tag ← F(k,m)

Zaakceptuj wiadomość jeśli **tag = F(k,m)**

8

Sprawę można podsumować następująco. Każdą bezpieczną funkcję generującą ciąg pseudolosowy można potraktować jako poprawny z punktu matematycznego generator tagu. Generowanie podpisu może wtedy polegać na wykonaniu ustalonej bezpiecznej funkcji pseudolosowej z argumentami: klucz k i wiadomość m. Weryfikacja tagu może polegać na uruchomieniu po stronie odbiorcy tej samej funkcji i stwierdzeniu, że otrzymany tag jest identyczny z przesłanym.

Zły przykład

Niech $F: K \times X \longrightarrow Y$ będzie bezpiecznym generatorem pseudolosowym generującym ciągi 10-bitowe:
$Y = \{0,1\}^{10}$

Czy opracowany system MAC I_F jest bezpieczny?

- Tak, MAC jest bezpieczny, bo PRF jest bezpieczny
- Nie, tagi są za krótkie i każdy może zgadnąć tag dla każdej wiadomości
- To zależy od funkcji F

9

Załóżmy, że mamy bezpieczną funkcję pseudolosową, która generuje ciąg 10 bitowy. Czy taka konstrukcja jest bezpieczna? Mam nadzieję, że zaznaczyliście odpowiedź „Nie…" Rzeczywiście, jeśli założymy zbyt krótkie tagi, można je łatwo zgadnąć dla wiadomości. Atakujący po prostu może wziąć wiadomość m i zacząć zgadywać wartości MAC dla niej. Prawdopodobieństwo zgadnięcia wynosi $1/2^{10}$ 1/1024 (wystarczy najwyżej 1024 próby, żeby na pewno zgadnąć MAC).

Dobrą wiadomością jest to, że tylko dla zbyt małych długości MAC system nie jest bezpieczny. Istnieje dowód matematyczny, że przy odpowiednio dużej ilości bitów w MAC system staje się bezpieczny. O bezpieczeństwie mówi się obecnie, gdy ciąg bitowy MAC ma 80 i więcej bitów.

Przykłady

- AES: może być MAC dla 16-bajtowych wiadomości.
- Główne pytanie: jak przekształcić Mały-MAC w Duży-MAC?
- Dwie główne konstrukcje stosowane w praktyce:
  - **CBC-MAC** (bankowość – ANSI X9.9, X9.19, FIPS 186-3)
  - **HMAC** (Protokoły internetowe: SSL, IPsec, SSH, …)
- W obu przypadkach następuje przekształcenie małego-PRF w duży-PRF.

10

Rozważania nad AES wskazują, że jest on dobrym przykładem bezpiecznego PRF. Może on być więc dobrym mechanizmem MAC dla 16-bajtowych wiadomości. Powstaje więc pytanie, czy dysponując PRF dla małego bloku danych możemy skonstruować PRF dla dużych bloków danych (np. Gigabajtów).
Okazuje się, że tak. Dysponując MAC dla małych wiadomości można zbudować MAC dla wiadomości dużych. Rozważone zostaną dwie praktyczne metody generowania MAC dla dużych wiadomości: CBC-MAC i HMAC. Obie polegają na zastosowaniu PRF wygenerowanych dla małych danych wejściowych i zastosowaniu ich do tworzenia PRF dla wiadomości większych. Metody te stosowane są jednak w różnych kontekstach. CBC-MAC jest stosowany w systemie Automatic Clearing House (ACH) do „czyszczenia czeków", czyli przenoszenia pomiędzy bankami pieniędzy, lub ich ekwiwalentów odnoszących się do kwot wystawionych na czeku. Np. ktoś realizuje czek w jednym banku, dostaje za to pieniądze, a te pieniądze są potem transferowane z banku, gdzie wystawiający czek ma swoje konto. Tam system CBC-MAC jest stosowany do zapewnienia integralności czeków przesłanych pomiędzy bankami. W Internecie, protokoły takie jak SSL, IPsec czy SSH używają dla zapewnienia integralności danych metody HMAC.

Przycinanie MAC opartych na PRF

Można udowodnić, że:

niech $F: \mathbf{K \times X \longrightarrow \{0,1\}^n}$ jest bezpiecznym PRF.

Wtedy także bezpieczne jest

$$\mathbf{F_t(k,m) = F(k,m)[1\ldots t]} \quad \text{dla} \quad 1 \le t \le n$$

$\Rightarrow$ Jeśli (S,V) jest MAC opartym na bezpiecznym PRF wytwarzającym n-bitowe tagi, to przycięty MAC zwracający **w** bitów jest bezpieczny

… dopóki $1/2^w$ jest dostatecznie małe (umowa: $w \ge 64$)

11

Slajd zwraca uwagę, na pewną zależność. Jeśli mamy bezpieczny PRF, to przycinając jego wyjście do pewnej długości także uzyskujemy bezpieczny PRF. Bezpieczeństwo leży w długości „przycięcia”, aby niemożliwe było współczesnymi metodami łatwe zgadnięcie (atak siłowy) MAC. Stąd w konsekwencji ustala się długość MAC na taką wartość, żeby zgadywanie było złożone obliczeniowo. W praktyce granicą jest wartość 2^{64}.

W konsekwencji, jeśli stosujemy np. AES do wygenerowania CBC i ona wynosi 128 bitów, to możemy ją skrócić do wartości 90 lub 80 bitów z zachowaniem bezpieczeństwa naszego rozwiązania.

Założenia w konstruowaniu systemów MAC dla dowolnie długich wiadomości

Mamy bezpieczny PRF **F,** generujemy bezpieczny MAC, dopóki długość wygenerowana przez F jest dostatecznie duża (64, 80, 90 bitów).

$$S(k, m) = F(k, m)$$

Nasz cel:

mając PRF dla krótkich wiadomości (AES)

zbudować PRF dla dużych wiadomości

Odtąd zakładamy $X = \{0,1\}^n$ (e.g. n=128)

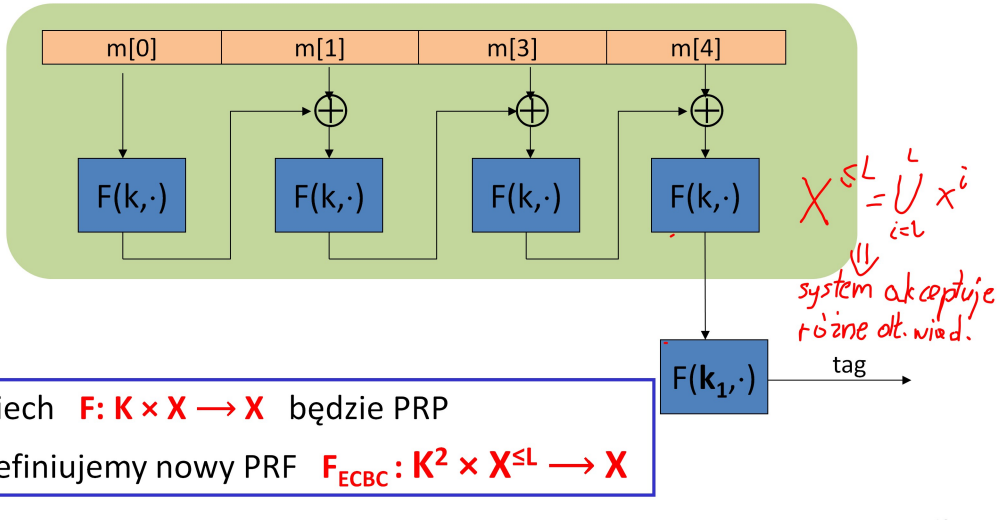
12

Z poprzedniego slajdu wynika, że jeśli mamy bezpieczną PRF, możemy się nią posłużyć jako MAC. Możemy też ją „przyciąć", ale długość przycięcia nie może być zbyt wielka. Musi zostać (64, 80 ,90 nitów). Naszym celem jest teraz opracowanie systemu zastosowania generatorów krótkich PRF (AES) do wygenerowania PRF dla dowolnie dużych wiadomości. W dalszych rozważaniach X będzie zbiorem n-bitowych ciągów podawanych na wejście algorytmu MAC. Jeśli ciągi będą podawane na AES, to n-128.

Konstrukcja 1: szyfrowany CBC-MAC

(ECBC)

surowe CBC

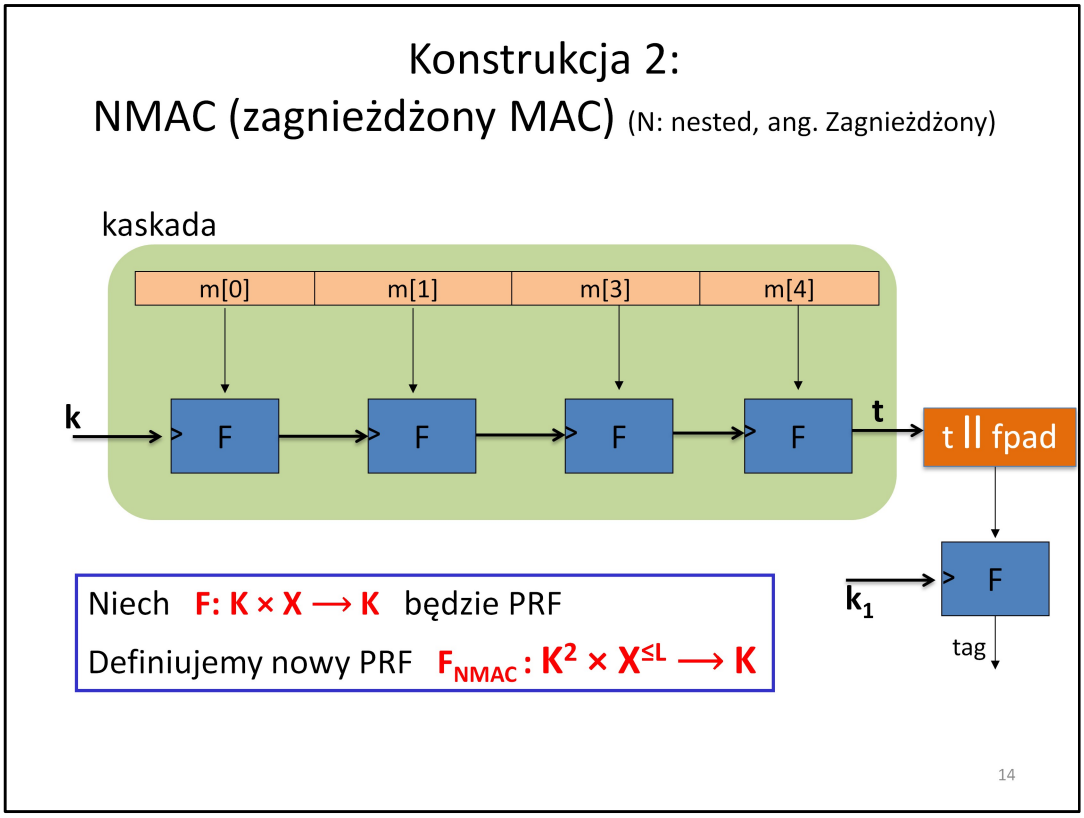


Niech $F: K \times X \longrightarrow X$ będzie PRP

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

13

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.



Inną klasyczną konstrukcją służącą do generowania skróconych MAC z dłuższych wiadomości jest tzw. zagnieżdżony MAC. Proszę zwrócić uwagę, że ta konstrukcja zwraca jako wyjście ciąg bitowy o długości odpowiadającej długości klucza k.
Wiadomość, jak na poprzednim slajdzie jest dzielona na bloki. Długość bloku odpowiada długości danych przyjmowanych na wejście przez funkcję F. Blok wiadomości jest przekształcany przez funkcję F z zastosowaniem klucza k. Rezultatem jest ciąg bitowy odpowiadający długości kluczowi i jest on z kolei przekazywany na wejście (klucz) kolejnej instancji funkcji F przetwarzającej kolejny blok widomości m. Czynność jest powtarzana do uzyskania ostatecznego wyjścia **t**. Jeśli zatrzymalibyśmy się w tym miejscu algorytmu, to wykonalibyśmy tak zwaną kaskadę. Ale nie daje to bezpiecznego MAC. Do otrzymanego klucza, który zwykle jest krótszy niż pojedynczy blok wiadomości dołącza się odpowiednie uzupełnienie, a następnie traktuje się tę wartość jako jeszcze jeden blok danych do przetworzenia przez funkcję F, przy czym ostatnie przetworzenie odbywa się przy pomocy osobnego klucza k_1. Ostatecznie uzyskuje się wyjście o długości kluczka k, należące do przestrzeni kluczy k (jedna z możliwych wartości klucza k).

Po co stosuje się dodatkowe szyfrowanie w ostatnim kroku ECBC-MAC i NMAC?

NMAC: załóżmy, że definiujemy MAC I = (S,V) gdzie

S(k,m) = kaskada(k, m)

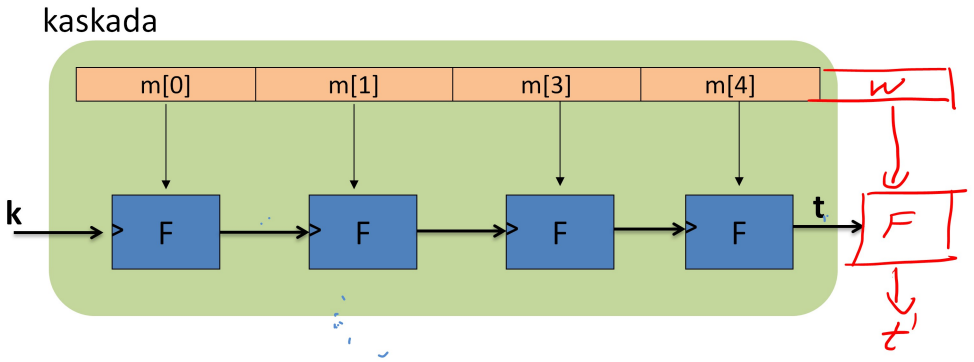
- Ten MAC jest bezpieczny
- Ten MAC może by sfałszowany bez zażądania przesłania wiadomości
- Ten MAC może być sfałszowany jeśli uzyskamy tylko 1 wiadomość
- Ten MAC może być sfałszowany, jeśli uzyskamy tylko 2 wiadomości

Mogę mając kaskada(k,m) obliczyć kaskada(k, m||w) dla dowolnego w.

15

Teraz rozważamy, dlaczego w obu wcześniej pokazanych schematach na koniec dokładany jest dodatkowy blok szyfrowania z osobnym kluczem. Zastanówmy się, w jakiej sytuacji moglibyśmy już złamać kaskadę. Okazuje się, że dysponując tylko jedną wiadomością i jej MAC system jest do złamania.
Jeśli schemat zawiera tylko kaskadę można z wyniku obliczeń kaskada(k,m) wyprowadzić możliwość obliczenia kaskada(k, m||w). Czyli można posłużyć się pojedynczą wiadomością z takim MAC, żeby móc wygenerować prawidłowe (z punktu widzenia tej wadliwej konstrukcji) MAC dla wiadomości o wydłużonej treści. Daje to możliwość sfałszowania kolejnych wiadomości i nie zapewnia bezpieczeństwa.

Jak sfałszować kaskadę?



Niech $F: K \times X \longrightarrow K$ będzie PRF

Definiujemy nowy PRF $F_{NMAC}: K^2 \times X^{\leq L} \longrightarrow K$

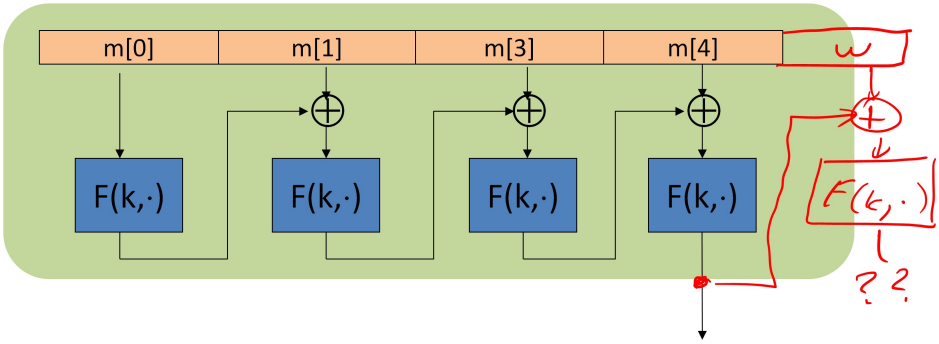
16

Gdyby do generowania MAC zastosowano samą kaskade, to łatwo można dodać do tej konstrukcji kolejny blok tekstu **w** i uruchomić kolejny blok F, aby uzyskać „prawidłowy” tag dla wiadomości wydłużonej o w (m||w).

Taki atak na MAC nazywa się atakiem „rozszerzeniowym” (ang. extension attack). NMAC bez ostatniego etapu szyfrowania z dodatkowym kluczem nie jest więc bezpieczny.

Jak sfałszować CBC-MAC (1)

surowe CBC



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

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

17

Próbna sfałszowania CBC-MAC metodą, jak wcześniej (w przypadku kaskady) skazana jest na niepowodzenie. Można próbować dopisać kolejny blok do przekazania do funkcji F, ale niestety nie znamy k i nie możemy tego sfałszowanego wyjścia policzyć.

Istnieje jednak inny sposób zaatakowania tej konstrukcji… atak z wybraną wiadomością.

Dlaczego trzeba stosować ostatni krok szyfrowania w schemacie ECBC-MAC?

Załóżmy, że definiujemy MAC $I_{RAW} = (S,V)$ gdzie

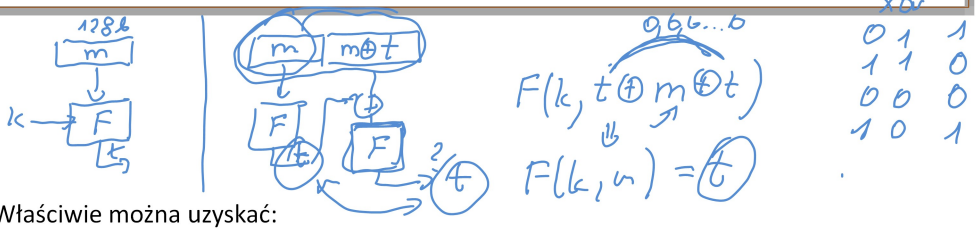
$S(k,m) = rawCBC(k,m)$

rawCBC = surowe CBC

Wtedy I_{RAW} jest łatwe do złamania z zastosowaniem **jednej** wybranej wiadomości

Atakujący pracuje w następujący sposób:

- Wybiera jedną wiadomość o długości 1 bloku $m \in X$
- Żąda wygenerowania dla niej tagu: Pobierz $t = F(k,m)$
- Wyprowadź t jako oszukany tag dla 2-blokowej wiadomości $(m, t \oplus m)$



Właściwie można uzyskać:

$rawCBC(k, (m, t \oplus m)) = F(k, F(k,m) \oplus (t \oplus m)) = F(k, t \oplus (t \oplus m)) = t$

18

Atakujący rozpoczyna atak poprzez poproszenie o obliczenie tagu dla wiadomości o długości równej jednemu blokowi. Tak naprawdę następuje prośba o jednokrotne wykonanie funkcji F. Wtedy atakujący preparuje nową wiadomość postaci $(m, t \oplus m)$ i wysyła ją do obliczenia tagu. Okazuje się wtedy, że t' dla tak skonstruowanej wiadomości jest dokładnie takie samo, jak dla jedno-blokowej wiadomości m. Pokazuje to zależność umieszczona na dole slajdu.

W taki sposób udowodniono, że można doprowadzić do załamania się systemu generowania MAC, w którym zastosujemy tylko surowe CBC.
W konsekwencji dodanie ostatniego bloku szyfrowania jest dopiero gwarantem otrzymania bezpiecznej konstrukcji generowania MAC.

Wyniki analiz matematycznych

- Po podpisaniu
  $|X|^{1/2}$ wiadomości z zastosowaniem ECBC-MAC lub
  $|K|^{1/2}$ wiadomości z zastosowaniem NMAC

  tak skonstruowane MAC przestają być bezpieczne

  (taka zależność wynika z paradoksu dnia urodzin, por. wyk. nr 1)
- W konsekwencji:
  - Jeśli stosujemy AES, to $|X| = 2^{128}$,
    wtedy po 2^{48} wiadomości trzeba zmienić klucz
  - Jeśli stosujemy 3DES, to $|X| = 2^{64}$,
    wtedy po 2^{16} wiadomości trzeba zmienić klucz

19

Analiza matematyczna przedstawionych wcześniej metod generacji bezpiecznych tagów pokazuje, że po $|X|^{1/2}$ tagach z kluczem k dla metody ECBC-MAC i po $|K|^{1/2}$ tagach generowanych z zastosowaniem metody NMAC należy zmienić klucze. Pierwiastek kwadratowy wynika z analizy matematycznej opierającej się na paradoksie dnia urodzin (proszę porównać wiadomości omówione podczas pierwszego wykładu).

W konsekwencji można wyprowadzić maksymalne liczby tagów, które można wygenerować bez zmiany klucza, jeśli generowanie tagów odbywa się odpowiednio z zastosowaniem szyfrów blokowych AES i 3DES.

Porównanie

ECBC-MAC jest powszechnie stosowany jako generator MAC w oparciu o algorytm AES.

- Np. tryb szyfrowania CCM (stosowany 802.11i <WPA2>)
- Jest też ustandaryzowany przez NIST i nazywany wtedy CMAC

NMAC nie jest zwykle stosowany w powiązaniu z algorytmami AES lub 3DES

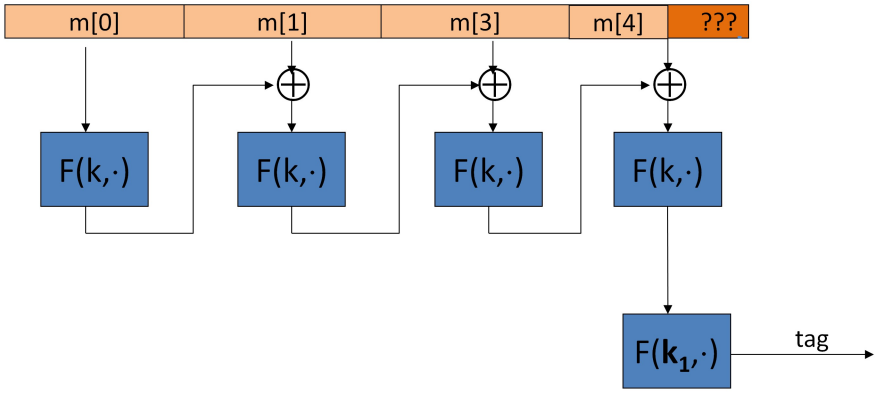
- Główne przyczyny: potrzeba zmiany klucza AES za każdym przejściem od bloku do bloku, a ten algorytm nie działa wydajnie przy tak częstych potrzebach obliczania rozszerzenia kluczy. Stosowane są tedy inne szyfry blokowe, w których rozszerzenia kluczy działają wydajniej.
- jednak NMAC jest podstawą dla popularnej konstrukcji MAC nazywanej HMAC (zostanie ona pokazana później)

20

Schemat RCBC-MAC jest często stosowany z powiązaniem z algorytmem AES. Stosowany jest do kontroli integralności pakietów w sieciach bezprzewodowych (standard WAP2). Jest też standardem NIST o nazwie CMAC.

Schemat NMAC bardzo rzadko stosuje się w powiązaniu z AES. Probelem jest tu konieczność podawanie na wejście poszczególnych bloków szyfrowania nowych kluczy, a więc konieczność obliczania ich rozszerzeń. W tym miejscu ujawnia się „słabość" AES, ponieważ proces rozszerzania klucza trwa w nim relatywnie długo. W tym schemacie stosuje się inne algorytmy szyfrowania blokowego. Warto zwrócić uwagę, że schemat NMAC jest podstawą innej popularnej konstrukcji zapewniającej integralność o nazwie HMAC. Zostanie ona omówiona później.

Co jeśli wiadomość do obliczenia MAC nie ma długości równej wielokrotności bloku?



21

Do tej pory przykłady omawiały sytuację idealną, kiedy wiadomość miała długość o wartości wielokrotności bloku szyfru blokowego. Teraz zastanowimy się, w jaki sposób powinien działać algorytm obliczania CBC dla wiadomości, w której ostatni blok tylko częściowo „obsadzony” jest treścią wiadomości.

CBC MAC padding (1)

Zły pomysł: uzupełnijmy m zerami

Postaje pytanie, czy otrzymany MAC jest bezpieczny?

- Tak, MAC jest bezpieczny
- Zależy od zastosowanej konstrukcji MAC
- Nie, mając tag dla wiadomości

  m atakujący otrzymuje tag dla wiadomości **mll0**

Problem: pad(m) = pad(mll0)

22

W pierwszym odruchu można by pomyśleć, że wystarczy brakując miejsce uzupełnić wartościami 0. Trzeba sobie jednak zadać pytanie, czy takie uzupełnienie (ang. padding) jest bezpieczne…
Niestety odpowiedź na to pytanie brzmi NIE. Okazuje się, że w takim systemie można łatwo uzyskać takie same tagi dla różnych wiadomości. Wiadomość m uzupełniona trzema zerami i wiadomość m_1 zawierająca na końcu 0 i uzupełniona dwoma zerami będą miały takie same tagi! Gdyby te ostatnie wartości wiadomości zawierały np. nasz dług, to nie można by było rozróżnić, czy jesteśmy winni 10, czy 1000 zł…

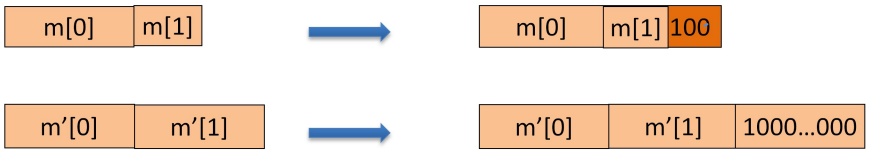
CBC MAC padding (2)

Dla zapewnienia bezpieczeństwa pady muszą być odwracalne!

$$m_0 \neq m_1 \quad \Rightarrow \quad pad(m_0) \neq pad(m_1)$$

ISO: pad w postaci "1000…00". Dodaj „atrapowy" blok, jeśli potrzeba.

– "1" oznacza rozpoczęcie padu.



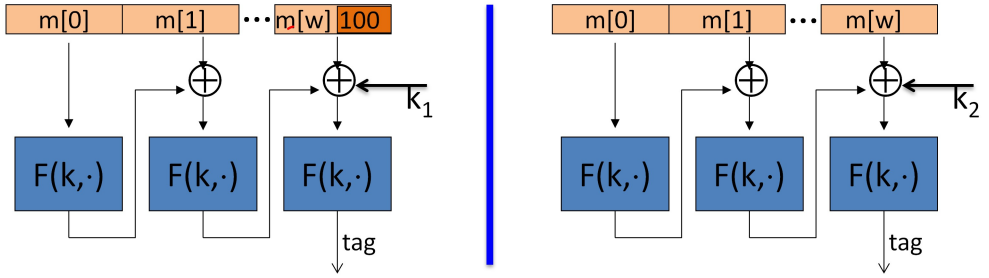
23

Pierwszy wniosek z poprzedniego slajdu jest taki, że funkcja dodająca brakujące bity do wiadomości musi być odwracalna. Co oznacza, że dwie różne wiadomości muszą dawać dwa różne uzupełnienia.
Został zaproponowany standard spełniający to wymaganie. Proponuje on uzupełnienie wiadomości ciągiem 1000….000 i ew. dodaniem „atrapowego" (ang. dummy) bloku.
Algorytm usuwania padu polega na czytaniu wiadomości od tyłu i usunięcia z niej wszystkich zer i pierwszej jedynki. Po usunięciu takiego ciągu uzyskuje się oryginalną wiadomość.
W pierwszym przykładzie do ostatniego bloku wiadomości dopisuje się ciąg „100".
Algorytm odczytujący wiadomość, po usunięciu wszystkich zer od tyłu i pierwszej jedynki uzyskuje wiadomość oryginalną. Jeśli zaś wiadomość jest długości wielokrotności bloków, żeby zachować zgodność z algorytmem usuwającym pad, do wiadomości dodaje się cały dodatkowy blok zaczynający się od 1 a dalej wypełniony zerami. Tutaj przy odczycie wiadomości znowu odcina się od niej n-bitów o wartości 0 i pierwszy bit o wartości 1 i w ten sposób odzyskuje się wiadomość oryginalną.
Okazuje się, że powstało wiele produktów kryptograficznych, które nie uwzględniły dodawania tego bloku i okazały się podatne na przekłamanie wiadomości. Załóżmy, że nie dołączyliśmy tego „atrapowego" bloku na zakończenie naszej wiadomości. I że nasza oryginalna wiadomość kończy się ciągiem bitów „100". Wtedy nasza wiadomość nie jest do odróżnienia od wiadomości, do której dodano pad „100". Nie można jednoznacznie odtworzyć informacji i bezpieczeństwo systemu nie jest zachowane.
Powstaje pytanie, czy zawsze trzeba dodawać aż cały blok danych w celu prawidłowego przeprowadzenia sprawdzania integralności wiadomości.

CMAC (standard NIST)

Wariant CBC-MAC gdzie stosuje się 3 klucze key = (k, k_1, k_2)

- Brak ostatniego bloku szyfrowania (atak udaremniony przez xor z kluczem)
- Brak „atrapowego” bloku (rozstrzygnięcie z zastosowaniem różnych kluczy)



24

Odpowiedzią na pytanie zadane w poprzednim slajdzie jest rozwiązanie o nazwie CMAC pokazujące, jak zastosować funkcję losową dodawania padu, co w konsekwencji nie wymaga dodawania „atrapowego” bloku danych na końcu wiadomości.
Jest to wariant konstrukcji CBC-MAC, która stosuje trzy klucze k, k_1 i k_2. Klucz k stosowany jest wielokrotnie w kolejnych blokach szyfrowania, natomiast klucze k_1 i k_2 są stosowane w ostatnim bloku szyfrowania. Klucze te są otrzymywane przez rozszerzenie klucza k. Obliczanie ostatniego bloku tagu zależy od tego, czy wiadomość ma długość równą wielokrotności bloku. Jeśli nie, to do ostatniego bloku wiadomości dodawany jest pad zgodnie ze standardem ISO (por. poprzedni slajd), ale przede szyfrowaniem na tym bloku wykonywana jest operacja XOR z kluczem k_1 (oczywiście nieznanym przez atakującego). Jeśli tak, to do wiadomości nie dodaje się żadnego padu, ale na ostatnim bloku wiadomości wykonuje się xor z kluczem k_2 (znowu ten klucz nie jest znany atakującemu). Pokazany zabieg zapobiega atakowi na rozszerzenie wiadomości (istnieje na to dowód matematyczny).
Zastosowane podejście ma dwie zalety: 1) w konstrukcji nie ma potrzeby wykonania końcowego bloku szyfrowania (bo jest on zastąpiony xor z kluczem), 2) konstrukcja nie wymaga dodawania „atrapowegp” bloku danych, ponieważ dla widomości z pad i bez pad (długość wielokrotności bloku) stosuje się różne klucze.
Warto zwrócić uwagę, że CMAC jest standardem NIST (USA) i w rzeczywistości, jeśli chce się zastosować strukturę CBC-MAC, to stosuję się jej wersję CMAC. Standard CMAC zakłada również, że blokiem szyfrującym jest AES.