Kryptografia i bezpieczeństwo danych
- Integralność II / odporność na kolizje I

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

1

Kolejny problem

- Konstrukcje ECBC i NMAC są sekwencyjne.
- Czy możemy zbudować **współbieżną** wersję MAC złożoną z małych PRF?

2

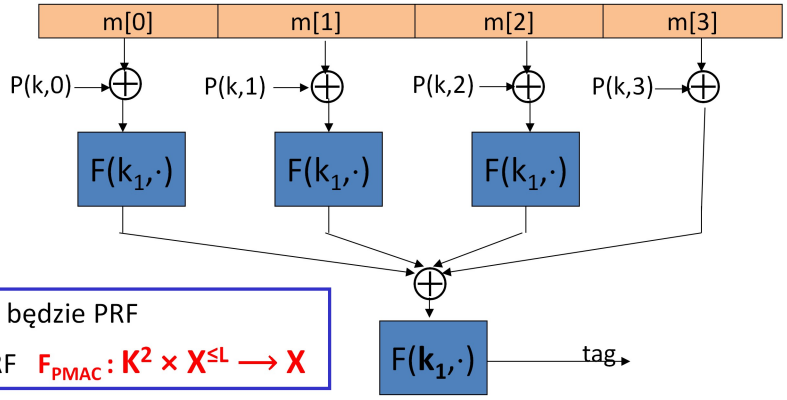
Analizowane na poprzednim wykładzie konstrukcje były sekwencyjne. Wyjście jednego bloku szyfrowania wchodziło na wejście następnego i nie można było procesu generowania tagu zrównoleglić. Powstaje więc pytanie, czy proces podpisywania wiadomości w celu zapewnienia jej integralności można zorganizować w sposób współbieżny?

Konstrukcja 3: PMAC (ang. Parallel MAC)

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

klucz = (k, k_1)

Padding podobny do CMAC



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

Definiuje się nowy PRF $F_{PMAC}: K^2 \times X^{\leq L} \longrightarrow X$

Bezpieczeństwo:

PMAC jest bezpieczny, jeśli: $q*L << |x|^{1/2}$

3

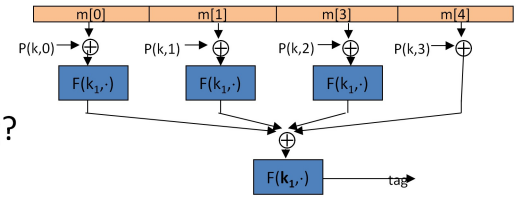
Współbieżne obliczenie MAC oferuje konstrukcja o nazwie PMAC. Podobnie jak wcześniej wiadomość dzielona jest na bloki. Ale tym razem każdy blok wiadomości jest przetwarzany osobno (w oderwaniu od pozostałych bloków). Dla każdego bloku wiadomości następuje wykonanie operacji xor na tym bloku i na rezultacie 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. Zwykłe 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.

PMAC jest przyrostowy (ang. incremental)

Załóżmy, że F jest PRP.

Kiedy m[1] ⟶ m'[1]
czy możemy szybko przeliczyć tag?



- nie, to jest niewykonalne
- tak, wykonaj $F^{-1}(k_1,tag) \oplus F\big(k_1, m'[1] \oplus P(k,1)\big)$
- tak, wykonaj $F^{-1}(k_1,tag) \oplus F\big(k_1, m[1] \oplus P(k,1)\big) \oplus F\big(k_1, m'[1] \oplus P(k,1)\big)$
- tak, wykonaj $tag \oplus F\big(k_1, m[1] \oplus P(k,1)\big) \oplus F\big(k_1, m'[1] \oplus P(k,1)\big)$

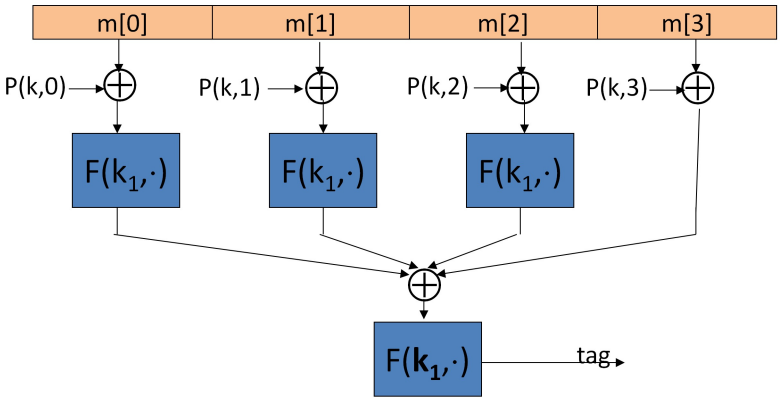
Potem wykonaj $F(k_1, \cdot)$

4

Ciekawą własnością PMAC, jest możliwość „skróconego" jego wykonania, jeśli w wiadomości zmienił się np. pojedynczy blok. Warunkiem możliwości wykonania takiej operacji jest zastosowanie jako funkcji F PRP (pseudolosowej permutacji), która jest odwracalna.

Załóżmy więc, że jeden z bloków wiadomości uległ zmianie i w podanej konstrukcji chcemy otrzymać szybko tag (gdybyśmy stosowali CBC-MAC, to i tak zawsze trzeba przeliczyć cały łańcuch przekształceń, a czas obliczen jest proporcjonalny do długości wiadomości). Podane są 4 możliwości… Po chwili zastanowienia można zauważyć, że rozwiązanie nr 3 jest poprawne.

PMAC – wyjaśnienie szybkiego przeliczenia 1 bloku



5

Żeby szybko przeliczyć tag dla jednego zmodyfikowanego bloku musimy przyjąć, że F jest odwracalna (PRP). Załóżmy, że zmodyfikowanym blokiem będzie blok o indeksie 1. Będzie w nim teraz zmodyfikowana część wiadomości m'[1]. Najpierw odwracamy ostatnią (dolną) funkcję $F(k_1,\cdot)$ w naszym systemie. W rezultacie otrzymujemy rezultat przedostatniej operacji w PMAC, czyli xor ze wszystkich wyjść funkcji $F(k_1, \cdot)$. Z właściwości funkcji xor wynika, że jeśli na otrzymanym xor z wszystkich wyjść funkcji $F(k_1, \cdot)$ (wykonanych na poszczególnych blokach wiadomości) wykonamy xor z wyrażeniem $F\big(k_1, m[1] \oplus P(k,1)\big)$, to usuwamy z wyniku wartość tego wyrażenia (bo. A xor A = 0). Wtedy na tym wyrażeniu wykonujemy xor z $F\big(k_1, m'[1] \oplus P(k,1)\big)$. Wynikiem takich operacji jest najpierw usunięcie zaszyfrowania bloku m[1], a potem zastąpienie go wynikiem szyfrowania bloku m'[1]. Otrzymaną wartość trzeba ponownie zaszyfrować funkcją $F(k_1, \cdot)$, aby otrzymać uaktualniony tag.

Ma to szczególne znaczenie dla długich wiadomości, w których zachodzą niewielkie zmiany. Wtedy znacznie szybciej, bez konieczności przeliczania całej konstrukcji można uzyskać tag. Oczywiście do przeliczenia tagu potrzebna jest znajomość klucza, oraz specjalna właściwość funkcji F, która musi być odwracalna.

Koncepcja jednorazowego MAC
(analog szyfrowania z kluczem jednorazowym w dziedzinie integralności)

- Chcemy zbudować MAC do zachowania integralności dokładnie jednej wiadomości.
- Za każdym generowaniem MAC zmieniamy więc klucz.
- Okazuje się, że zastosowanie takiego podejścia jest bezpieczne oraz szybsze niż wcześniej omówione metody tworzenia MAC oparte o PRF (funkcje generujące liczby pseudolosowe)

6

Przykład budowania jednorazowego MAC

Niech q będzie dużą liczbą pierwszą (np. $\mathbf{q = 2^{128}+51}$)

key = (k, a) ∈ $\{1,\ldots,q\}^2$ (dwie losowe liczby całkowite z przedziału[1,q])

msg = (m[1], …, m[L]) gdzie każdy blok jest 128 bitową liczbą całkowitą.

$$\mathbf{S(\ key,\ msg\ )\ =\ P_{msg}(k) + a \pmod q}$$

gdzie $P_{msg}(x) = m[L]\cdot x^L + \ldots + m[1]\cdot x$
jest wielomianem o rzędzie L

Pokazujemy, że: mając $\mathbf{S(\ key,\ msg_1)}$ atakujący nie dostaje informacji na temat $\mathbf{S(\ key,\ msg_2\ )}$.

7

Na slajdzie pokazano jedną z możliwości konstruowania jednorazowego MAC. Podejście rozpoczyna się od wybrania liczby pierwszej, która jest „nieco” dłuższa od rozmiaru bloku. W naszym wypadku jest to liczba $q=2^{128}+51$. Teraz nasz klucz będzie parą losowych liczb z przedziału od 1 do wartości naszej liczby pierwszej [1,q]. Naszą wiadomość dzielimy na 128-bitowe bloki i traktujemy każdy blok jako liczbę z zakresu od 0 do $2^{128}-1$.
MAC jest definiowany w następujący sposób: Bierzemy nasze bloki wiadomości i konstruujemy z nich wielomian o stopniu L, gdzie wartość wyrazu wolnego wynosi 0. Bierzemy wielomian odpowiadający wiadomości i obliczamy jego wartość dla argumentu wynoszącego **k**, która jest połową naszego tajnego klucza. Do otrzymanej wartości dodajemy wartość **a**, która jest drugą połową naszego klucza. I tak powstaje MAC. Otrzymany rezultat jest skracany modulo **q**.
Taka konstrukcja uznawana jest za bezpieczną. Można wykazać, że obliczając MAC dla jednej wiadomości nie ujawniamy jakiejkolwiek informacji na temat MAC obliczonej dla wiadomości innej. Należy pamiętać, że za każdym razem musi być zmieniany klucz. Jeśli o tym zapomnimy, cała konstrukcja przestaje być bezpieczna.

Generowanie wielokrotnego MAC na bazie jednokrotnego MAC

Niech (S,V) będzie bezpiecznym jednokrotnym MAC zdefiniowanym na $(K_I, M, \{0,1\}^n)$.

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

Wolne, ale krótkie wej. Szybkie, ale długie wej.

Carter-Wegman MAC: $CW((k_1,k_2), m) = (r, F(k_1,r) \oplus S(k_2,m))$

dla losowego $r \longleftarrow \{0,1\}^n$.

Tw: Jeśli (S,V) jest bezpiecznym jednorazowym MAC i F jest bezpieczną PRF,

to CW jest bezpiecznym MAC generującym tagi o rozmiarze $\{0,1\}^{2n}$

Weryfikacja CW MAC:

$V(k_2, m, F(k_1, r) \oplus t))$

8

Na slajdzie pokazano technikę tworzenia wielokrotnego MAC na bazie jednokrotnego. Zakładamy, że dysponujemy bezpiecznym jednokrotnym (funkcje S i V) MAC generującym taki o długości n. Dodatkowo posiadamy funkcję F będącą bezpiecznym PRF, która również generuje n bitowe wyjście. Konstrukcja generująca bezpieczny MAC może mieć postać, jak to zaproponowali Carter i Wegman: W pierwszej kolejności stosowany jest jednokrotny MAC z kluczem $\mathbf{k_2}$ dla wiadomości **m**, a następnie jest on szyfrowany z zastosowaniem funkcji **F**. Parametrami funkcji szyfrującej są klucz $\mathbf{k_1}$ i wartość **r**. Wartość **r** jest wybierana losowo. Elegancja rozwiązania polega na tym, że zastosowaliśmy szybki jednokrotny algorytm generowania MAC do generowania tagów dla długich wiadomości, wolny algorytm PRF służy do wygenerowania ciągu pseudolosowego służącego do zaszyfrowania tagu.

Można też udowodnić, że jeśli jednokrotny MAC jest bezpieczny i i PRF jest bezpieczna, to otrzymaliśmy metodę wielokrotnego generowania MAC na podstawie metody generowania jednokrotnego MAC i cała konstrukcja jest bezpieczna. Warto zwrócić uwagę, że wyjście podanej metody generuje MAC o długości 2n w stosunku do tego co generowały F i PRF.

Pokazane podejście do generowania tagów jest składnikiem systemów zaakceptowanych przez NIST do wykonywania szyfrowania z zachowaniem integralności.

Carter-Wegman MAC jest również dobrym przykładem losowego generowania tagów, pod warunkiem, że wartość nonce (r) jest za każdym razem wybierana losowo. Wtedy tag dla tej samej wiadomości obliczony dwukrotnie dla różnych wartości r jest różny. Jest to też z matematycznego punktu widzenia rozwiązanie nie będące PRF, ponieważ jedna wiadomość może mieć wiele tagów, które są wszystkie prawidłowe dla danej wiadomości.

Na dole slajdu pokazano formułę sprawdzenia, że para (r,t) jest poprawnym tagiem wygenerowanym za pomocą (CW MAC) dla danej wiadomości.

Konstrukcja 4: HMAC (Hash-MAC)

- Najpopularniejszą metodą generowania MAC w Internecie jest HMAC
- … ale żeby ją omówić trzeba wprowadzić jeszcze funkcje hashujące (skrótu)…

Podsumowanie na tym etapie

Jak dotąd, omówiliśmy 4 konstrukcje MAC:

PRFs:

- **ECBC-MAC, CMAC** : zwykle stosowane w połączeniu z AES (np. 802.11i)
- **NMAC** : podstawa do stworzenia HMAC (zaprezentowane dalej)
- **PMAC**: równoległy MAC

randomized MAC:

- **Carter-Wegman MAC**: zbudowany na bazie jednokrotnego MAC

Dalej zostanie pokazany: MACs wywodzący się z odporności na kolizje.

10

Odporność na kolizje

Niech H: M $\rightarrow$T będzie funkcją skrótu (hash) (|M| >> |T|)

Kolizją dla H jest para wiadomości m_0 , $m_1 \in M$ takich, że:

$H(m_0) = H(m_1)$ i $m_0 \neq m_1$

Funkcja H jest **odporna na kolizje** jeśli prawdopodobieństwo wygenerowania identycznych funkcji skrótu dla różnych wiadomości jest pomijalne.

Przykładowo: SHA-256 (zwraca 256 bitów)

11

Funkcja hashująca przekształca przestrzeń wiadomości w przestrzeń tagów. Przestrzeń wiadomości jest znacznie większa niż przestrzeń tagów. Wiadomość może mieć długości rzędu GB, a hash – 160 bitów. Kolizja dla funkcji H zachodzi jeśli mamy dwie różne wiadomości, ale funkcje hashujące dla nich są identyczne. Mówimy, że funkcja H jest odporna na kolizje, jeśli jest trudno znaleźć kolizję.

Sytuacja wydaje się mało intuicyjna, ponieważ widać, że przestrzeń wiadomości jest znacząco większa od przestrzeni funkcji hash, do powinno powodować , że jest bardzo wiele wiadomości dających ten sam hash. W rozważanym zagadnieniu pytamy się dokładnie, czy stosując zbiór jawnych (explicit) efektywnych algorytmów poszukujących kolizje jesteśmy w stanie te kolizję znaleźć.

Co oznacza, że algorytm jest jawny (explicit)? Nie wystarcza powiedzieć, że algorytm istnieje i wyświetla kolizje. On musi być opisany i możliwy do uruchomienia na komputerze, aby wygenerować te kolizje. Dobrym przykładem takiego algorytmu jest SHA-256, który przekształca dowolnie długą wiadomość w 256 hash (skrót). Rozmiar wiadomości może być z perspektywy komputerów długi, nawet GB. Jak dotąd nikt nie zdołał znaleźć kolizji dla tego algorytmu.

Budowanie MAC z odporności na kolizje (1)

Niech I = (S,V) będzie MAC dla krótkiej wiadomości na zbiorach (K,M,T) (np. AES)
Niech H: $M^{big} \rightarrow M$

Definicja: $I^{big} = (S^{big}, V^{big})$ nad zbiorami (K, M^{big}, T) jako:

$$S^{big}(k,m) = S(k,H(m)) \quad ; \quad V^{big}(k,m,t) = V(k,H(m),t)$$

Tw.: jeśli I jest bezpiecznym MAC i H jest odporna na kolizje
to I^{big} jest bezpiecznym MAC.

Przykład: $S(k,m) = AES_{2\text{-block-cbc}}(k, \text{SHA-256}(m))$ jest bezpiecznym MAC.

12

Załóżmy, że mamy MAC dla krótkich wiadomości (np. AES). Zakładamy również, że dysponujemy funkcją hash, która przekształca dużą wiadomość (M^{big})w jej hash (skrót) M. Można wtedy zdefiniować nowy MAC (nazywany I^{big}) jako „mały" MAC policzony dla hash (skrótu) z wiadomości. Sprawdzenie MAC polegałoby na policzeniu skrótu z wiadomości i poddaniu go weryfikacji algorytmem V dla małego MAC.
Tak więc można rozszerzyć omówione wcześnie konstrukcje MAC dla krótkich wiadomości do zapewnienia integralności dowolnie dużych wiadomości.
Twierdzenie o bezpieczeństwie takiego systemu nie jest trywialne, ale ostatecznie można udowodnić, że jeśli dysponujemy bezpiecznym małym MAC i funkcję odporną na kolizje, to kombinacja tych obiektów jest również bezpiecznym MAC.
Na przykład dysponując algorytmem AES włożonym w konstrukcję CBC-MAC dla dwóch 16 bajtowych bloków danych i dysponując funkcję SHA-256 i konstruując z nich MAC dla dużych wiadomości otrzymujemy bezpieczne rozwiązanie.

Budowanie MAC z odporności na kolizje (2)

$S^{big}(k, m) = S(k, H(m))$; $V^{big}(k, m, t) = V(k, H(m), t)$

Odporność na kolizje jest konieczna dla zapewnienia bezpieczeństwa:

Załóżmy, że atakujący może znaleźć $m_0 \neq m_1$ takie, że $H(m_0) = H(m_1)$.

Wtedy: S^{big} nie jest bezpieczne na atak z jedną wybraną wiadomością

Krok 1: atakujący prosi o $t \leftarrow S(k, m_0)$

Krok 2: zwraca (m_1, t) jako fałszerstwo

13

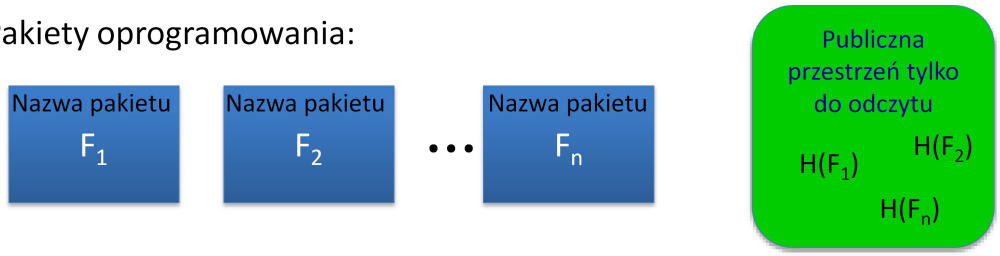
Okazuje się, że zapewnienie odporności na kolizje jest konieczne do utworzenia bezpiecznego systemu. Wystarczy sobie wyobrazić, że jesteśmy w stanie znaleźć kolizję i atakujemy system w taki sposób, że wysyłamy do niego jedną z pary wiadomości, które mają ten sam hash, a potem budujemy fałszywą wiadomość składającą się z drugiej wiadomości z pary i tego samego tagu.

W konsekwencji, jeśli kiedykolwiek ktoś ogłosi, że znalazł kolizję dla algorytmu SHA-256, to wszystkie systemy kryptograficzne, które z niego korzystają przestają być bezpieczne.

Odporność na kolizje jest kolejnym prymitywem kryptograficznym, który okazuje się bardzo przydatny.

Ochrona integralności plików z zastosowaniem C.R. (ang. Collision Resistance) hash

Pakiety oprogramowania:



Kiedy użytkownik pobiera plik, może sprawdzić, czy plik zachował integralność

H jest odporna na kolizje ⇒
atakujący nie może zmodyfikować pakietu danych bez wykrycia tego faktu

Nie potrzeba wtedy kluczy (tzw. mechanizm publicznej weryfikacji), ale przestrzeń, z której następuje pobieranie pakietów musi mieć atrybut „tylko do odczytu”

14

Wyobraźmy sobie, że mamy pewien zbiór plików, które chcemy chronić, np. pakiety, które można doinstalować do systemu operacyjnego rozszerzające jego funkcjonalność (np. sterowniki do nowych urządzeń wej./wyj.). Na slajdzie pokazano trzy różne pakiety. Użytkownik chce pobrać te pakiety, ale chce być równocześnie pewny, że pobrał wersje pakietów niezmodyfikowane przez atakującego. To co dostarczyciel pakietów może zrobić, to posłużyć się niewielką przestrzenią do umieszczania plików, ale posiadającą własność dla pobierających dane „tylko do odczytu” i umieścić tam hash (skróty) z plików zamieszczonych do pobrania (atakujący nie może tam wstawić własnych hash zmodyfikowanych przez niego plików). Odbiorca pakietów może wtedy pobrać pliki, obliczyć dla nich hash i porównać z tym opublikowanym na obszarze „tylko do odczytu”. Jeśli skróty są identyczne, to z bardzo dużym prawdopodobieństwem, graniczącym z pewnością (jeśli zastosowano hash dla którego nie wykryto kolizji) plik nie został zmodyfikowany.
Pokazany przykład warto zestawić z obliczaniem MAC dla wiadomości, które ma zapewnić ich integralność (stosowaliśmy wtedy ukryty klucz). W obecnie pokazanej konstrukcji można zrezygnować ze stosowania kluczy i szyfrowania, dla zapewnienia integralności danych. Jedynym warunkiem poprawności konstrukcji jest zamieszczenie skrótów w przestrzeni, którą nie może zmodyfikować atakujący. Dodatkowo, każdy pobierający dane może sprawdzić ich integralność. Taka konstrukcja jest bardzo powszechnie stosowana do dystrybucji sterowników i pakietów np. dla Linux’a.

Później omówione zostaną podpisy elektroniczne, które będą brały po trochę funkcjonalność stosowaną do publikacji plików i po trochę szyfrowanie z kluczem aby nie potrzebować specjalnego obszaru „tylko do odczytu” dla zapewnienia integralności wiadomości.

Uogólniony atak na funkcje odporne na kolizje

Niech $H: M \rightarrow \{0,1\}^n$ będzie funkcją hashującą ($|M| >> 2^n$)

Ogólny alg. do znalezienia kolizji w czasie **$O(2^{n/2})$** może mieć postać

Algorytm:
1. Wybierz **$2^{n/2}$** losowych wiadomości z dziedziny M: $m_1, \ldots, m_{2^{n/2}}$ (różne z wysokim prawdopodobieństwem)
2. Dla i = 1, …, $2^{n/2}$ oblicz $t_i = H(m_i)$ $\in\{0,1\}^n$
3. Poszukuj kolizji ($t_i = t_j$). Jeśli się nie udało, idź do kroku 1.

Powstaje pytanie, jak dobrze takie rozwiązanie działa?

15

Jak dotąd zajmowaliśmy się atakami na szyfry blokowe. Najprostszym był atak siłowy polegający na przeszukiwaniu przestrzeni kluczy. On wymuszał minimalną długość klucza na wartość 128 bitów.
Podobnie, w przypadku odporności na kolizje istnieje taki uogólniony schemat ataku zwany atakiem urodzinowym. Zachowanie bezpieczeństwa na taki atak wymusza oczywiście zalecane długości wyjść funkcji hashujących.
Taki uogólniony atak można opisać w następujący sposób. Dysponujemy funkcją hashującą, przekształcającą wiadomości ze zbioru M na n-bitowe ciągi. Przestrzeń wiadomości jest znacznie większa niż przestrzeń skrótów. Pokazujemy ogólny algorytm, który próbuje znaleźć kolizje w czasie proporcjonalnym do pierwiastka kwadratowego z długości hash. Zaczynamy od wylosowania $2^{n/2}$ losowych wiadomości z przestrzeni wszystkich możliwych wiadomości. Nazwijmy je m_1 do $m_{2^{n/2}}$. Ponieważ przestrzeń wiadomości jest olbrzymia, to z bardzo dużym prawdopodobieństwem wylosowane wiadomości są różne. Następnie dla każdej z wylosowanych wiadomości jest obliczana funkcja hashująca i w rezultacie otrzymujemy zbiór hashów (skrótów). Kolejnym krokiem algorytmu jest poszukiwanie kolizji, czyli szukanie, czy w otrzymanym zbiorze tagów istnieją dwa takie same. Jeśli kolizja nie jest znaleziona możemy powrócić do pierwszego kroku algorytmu. Powstaje pytanie w jakim czasie realizując taki algorytm znajdziemy kolizję?
Okazuje się, że znalezienie kolizji nie wymaga wielu iteracji głównych kroków algorytmu i czas znalezienia kolizji można przybliżyć wartością $2^{n/2}$.

Paradoks dania urodzin - przypomninie

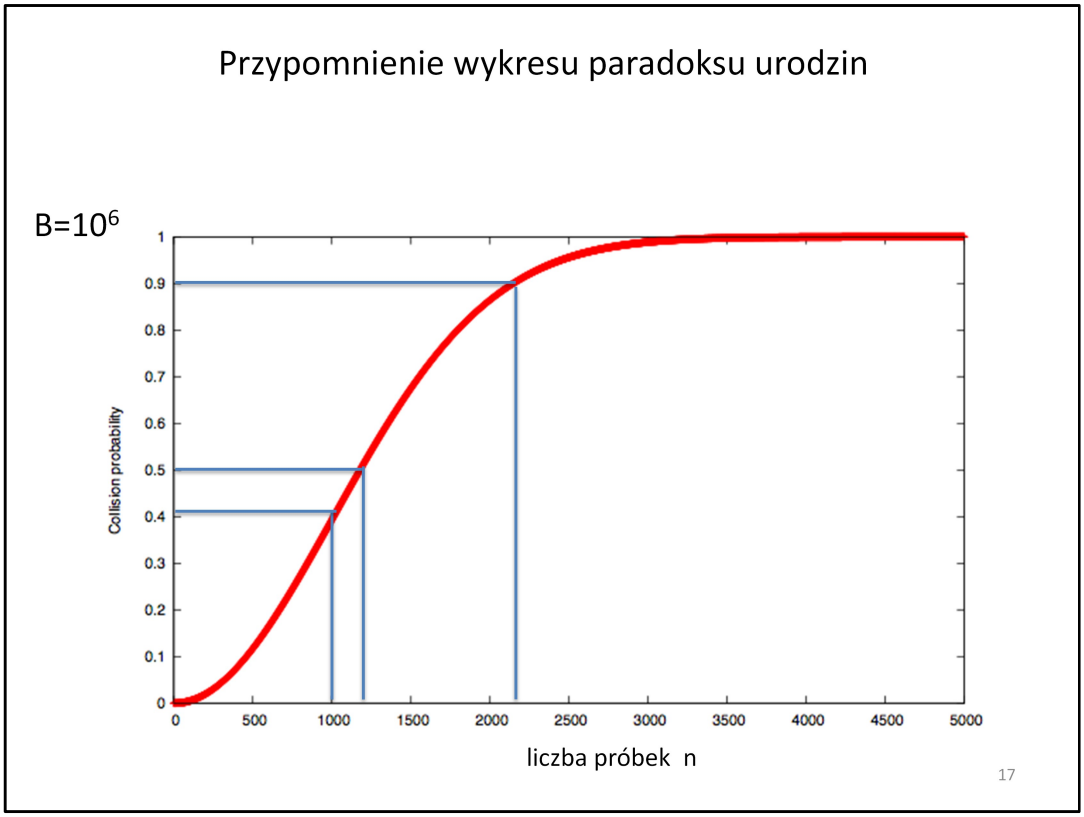
Niech $r_1, \ldots, r_n \in \{1,\ldots,B\}$ będą niezależnymi równomiernie rozłożonymi liczbami całkowitymi.

Tw: kiedy $n = 1.2 \times B^{1/2}$ wtedy

$$\Pr[\exists i \neq j: \; r_i = r_j] \geq \tfrac{1}{2}$$

16

Twierdzenie o paradoksie dnia urodzin mówi. Jeśli weźmiemy n niezależnych, równomiernie rozłożonych liczb całkowitych (można tu myśleć o losowaniu liczb, ale prawdopodobieństwo wylosowania każdej z nich jest zawsze takie samo) o zbiorze wartości z zakresu 1 do B, to po przeanalizowaniu $1.2 \times B^{1/2}$ liczb znajdziemy parę identycznych liczb z prawdopodobieństwem ½. Dowodu twierdzenia nie będę przytaczał. Nazwa paradoksu pochodzi od zastosowania zależności do sprawdzenia, ile osób w jednym miejscu należy zgromadzić, żeby z prawdopodobieństwem ½ dwie z nich urodziły się tego samego dnia roku. Zapisując zależność $1.2 \times 365^{1/2}$ otrzymujemy liczbę 23, która na pierwszy rzut oka jest zaskakująca, ale oparta na precyzyjnych matematycznych wyliczeniach. Co więcej, jeśli rozkład prawdopodobieństwa zbioru losowych liczb jest inny niż równomierny, to to prawdopodobieństwo tylko wzrasta.



Naszkicujmy wykres odzwierciedlający zależność prawdopodobieństwa znalezienia dwu identycznych liczb w ciągu równomiernie losowanych liczb całkowitych z zakresu od 1 do B. W naszym wypadku B = 1000000. Na osi poziomej zawarto liczbę losowań, na pionowej prawdopodobieństwo, że znajdziemy kolizję (dwie takie same liczby w grupie liczb wylosowanych). Widać, że prawdopodobieństwo 0.5 osiągamy po przeanalizowaniu $1.2\times B^{1/2}$ próbek, w przybliżeniu dla n=1200. Jeśli wzięlibyśmy dokładnie pierwiastek kwadratowy z 1000000, to prawdopodobieństwo „kolizji" wynosiłoby ok. 0.41. Zwróćmy również uwagę, że krzywa prawdopodobieństwa dosyć szybko rośnie i po przeanalizowaniu nieco więcej niż $2\times B^{1/2}$ ($\approx$2200) próbek prawdopodobieństwo znalezienia kolizji dochodzi do 0.9. Analizując prawdopodobieństwo znalezienia kolizji dla liczby próbek poniżej $B^{1/2}$ losowań zauważamy także ta liczba szybko schodzi od 0.

Wracając do uagólnionego ataku na kolizje…

Niech $H: M \rightarrow \{0,1\}^n$

Algorytm:

1. Wybierz $\mathbf{2^{n/2}}$ losowych wiadomości z dziedziny M: $m_1, \ldots, m_{2^{n/2}}$
2. Dla i = 1, …, $2^{n/2}$ oblicz $t_i = H(m_i) \in \{0,1\}^n$
3. Poszukuj kolizji ($t_i = t_j$). Jeśli się nie udało, idź do kroku 1.

Spodziewana liczba iteracji algorytmu ≈ 2

Czas wykonywania $O(2^{n/2})$ (przestrzeń $O(2^{n/2})$)

18

Zwróćmy uwagę, na fakt, że przy zastosowaniu uogólnionego algorytmu ataku na kolizje w pierwszym przebiegu algorytmu dokonujemy wylosowania $\mathbf{2^{n/2}}$ wiadomości i obliczamy dla nich hash. Otrzymujemy $2^{n/2}$ tagów, które są niezależne. Przy tak dobranej liczbie wylosowanych wiadomości. zgodnie z paradoksem dnia urodzin, prawdopodobieństwo wystąpienia kolizji wynosi ½. Stąd jeśli wykonamy główną pętlę algorytmu w przybliżeniu 2 razy, to z bardzo dużym prawdopodobieństwem znajdziemy kolizję (prawdopodobieństwa się dodadzą…).
Złożoność czasowa takiego rozwiązania wynosi $O(2^{n/2})$ (przy okazji złożoność pamięciowa również wynosi $O(2^{n/2})$).
Z takich ogólnych rozważań wynika, że jeśli długość naszego hash będzie wynosić n, to czas znalezienia kolizji wynosi $2^{n/2}$. Przykładowo, zastosowanie hash o długości 128 bitów oznacza, że kolizję dla takiego hash będzie można znaleźć w czasie 2^{64}, co w chwili obecnej nie jest uważane za wystarczająco bezpieczne.

Przykładowe odporne na kolizje funkcje hash:

Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

	funkcja	długość skrótu(bity)	szybkość (MB/sec)	uogólniony czas ataku
Standardy NIST	SHA-1	160	153	2^{80}
	SHA-256	256	111	2^{128}
	SHA-512	512	99	2^{256}
	Whirlpool	512	57	2^{256}

*najlepszy znany algorytm znalezienia kolizji dla SHA-1 wymaga analizy 2^{51} hashów

19

Dokonajmy przeglądu kilku opublikowanych funkcji hash. Pierwsze trzy są standardami NIST (USA) (SHA-1, SHA-256, SHA-512). Ostatni algorytm (Whirlpool) został zapropnowany przez twórców AES. Algorytm SHA-1 generuje hash o długości 160 bitów. Uogólniony atak na niego może być przeprowadzony w czasie 2^{80}. Uogólnione ataki na SHA-256 i SHA-512 mogą być przeprowadzone w czasach odpowiednio 2^{128} i 2^{256}. Warto zwrócić uwagę, że im większe są długości generowanych hashów, tym algorytmy wolniej działają. W większości wypadków kluczowe znaczenie ma jednak ustalony poziom bezpieczeństwa.

Proszę zwrócić uwagę, że SHA-1 została w tabeli wyróżniona. Wprawdzie nikt jeszcze nie znalazł kolizji dla tego algorytmu, ale w nowo-projektowaych rozwiązaniach nie zaleca się już jej stosowanie. Sugeruje się stosowanie SHA-256,a w rozwiązaniach, w których chcemy zachować zwiększone bezpieczeństwo - SHA-512. W szczególności, został opublikowany teoretyczny atak na SHA-1, który mógłby znaleźć kolizję w czasie 2^{51}. Więc w przypadku tego rozwiązania jest tylko kwestią czasu, kiedy zostanie znaleziona kolizja.

Gdyby zastosować komputery kwantowe…

	Klasyczne algorytmy	**Algorytmy kwantowe**
Szyfry blokowe **E: K × X ⟶ X** pełne przeszukiwanie	$O(\lvert K\rvert)$	$O(\lvert K\rvert^{1/2})$
Funkcja hash **H: M ⟶ T** poszukiwanie kolizji	$O(\lvert T\rvert^{1/2})$	$O(\lvert T\rvert^{1/3})$

20