Część 5

Kryptografia bezkluczowa
Zastosowania funkcji skrótu w różnych mechanizmach kryptograficznych

Kryptografia bezkluczowa („unkeyed cryptography”)

System (algorytm, mechanizm) kryptografii bezkluczowej nie wymaga do realizacji usługi kryptograficznej wykorzystania żadnego tajnego parametru.

Zasadniczo rozważa się dwa dominujące zastosowania kryptografii bezkluczowej:

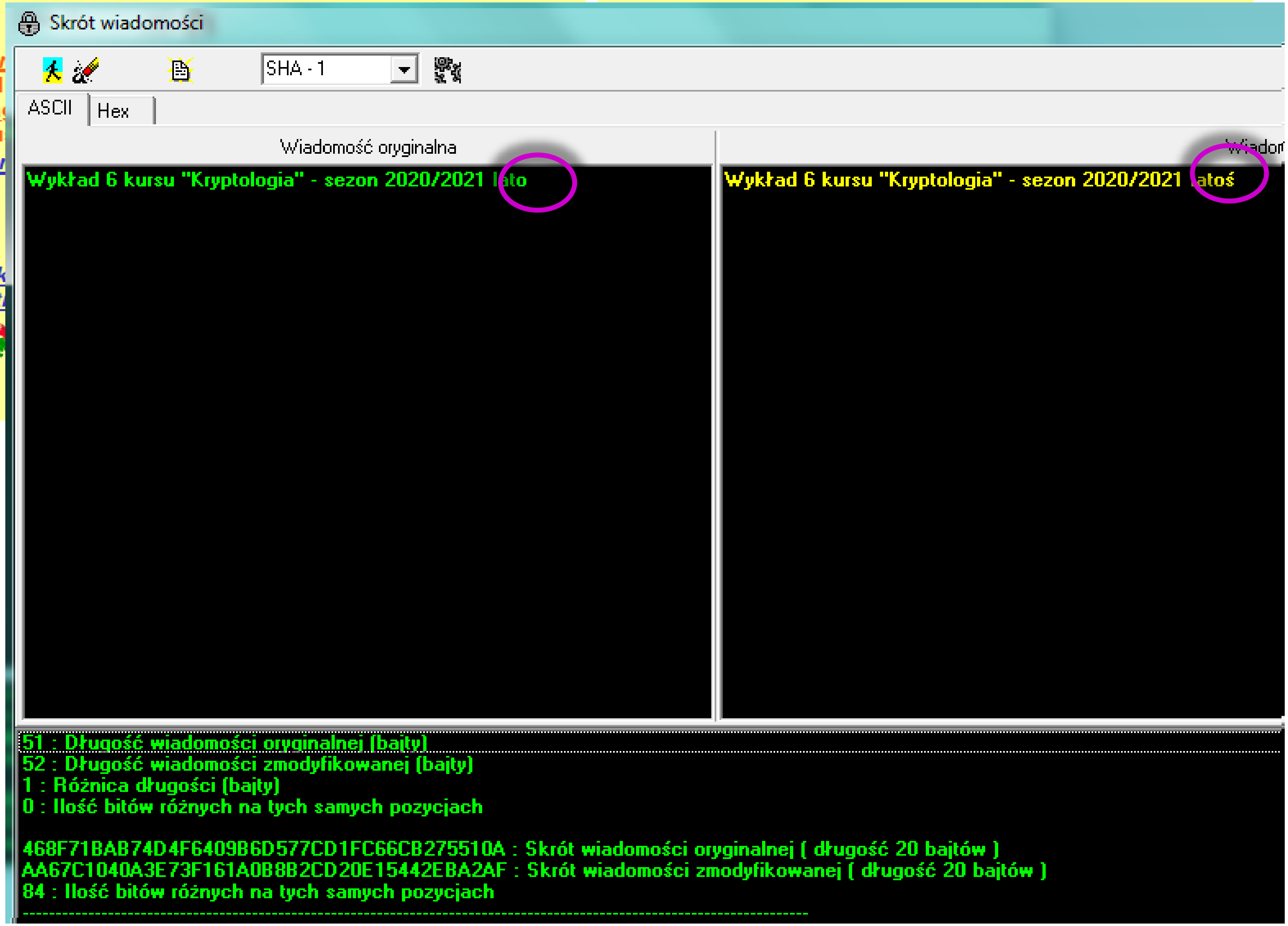
- **generowanie binarnych ciągów (pseudo)losowych o „dobrych” właściwościach kryptograficznych;**
- **obliczanie wartości funkcji skrótu („hash function”) jako narzędzie do kontroli integralności obiektów danych (domyślnie: ciągów binarnych).**

W obu przypadkach bezpośrednio lub pośrednio wykorzystuje się koncepcję funkcji jednokierunkowej.

Jednokierunkowość może być uzależniona od „braku” pewnej dodatkowej informacji („trapdoor”), której znajomość czyni z funkcji jednokierunkowej funkcję odwracalną (bliskie pokrewieństwo z koncepcją kryptografii klucza publicznego).

„Bezkluczowość” obu ww. mechanizmów jest „zwodnicza”, bowiem z reguły oba mechanizmy wykorzystywane są jako część mechanizmów kryptografii symetrycznej albo asymetrycznej.

Czym jest funkcja skrótu (przypomnienie)?



Czym jest funkcja skrótu (przypomnienie)?

Kryptograficzne funkcje skrótu

Funkcja skrótu (hash function) - efektywna obliczeniowo funkcja jednokierunkowa odwzorowująca ciągi binarne o dowolnej długości na ciągi binarne o ustalonej długości

Funkcja skrótu bez klucza: $h = f(x)$

Funkcja skrótu z kluczem (kryptograficzna funkcja kontrolna): $h = f(x, k)$

58

"Dobra" funkcja skrótu musi być *pseudolosowa*, tzn. dowolna wartość funkcji skrótu powinna być jednakowo prawdopodobna, zaś zmiana jednego bitu argumentu funkcji skrótu („skracanej" wiadomości) powinna powodować zmianę około połowy bitów wartości funkcji dla tego nowego argumentu

Kolizja - dwie różne wiadomości x_1 i x_2 są w kolizji wtedy, gdy $h(x_1) = h(x_2)$

Odporność funkcji skrótu na ataki

Pre-image resistance (odporność na wyznaczenie przeciwobrazu, jest to własność każdej funkcji jedno-kierunkowej) – trudne obliczeniowo jest wyznaczenie na podstawie wartości h(x) wartości argumentu x.

2nd pre-image resistance (odporność na wyznaczenie drugiego przeciwobrazu, tzw. „słaba odporność na kolizje"/"weak collision resistance") – trudne obliczeniowo jest wyznaczenie na podstawie wartości argumentu x_1 i $h(x_1)$ wartości argumentu x_2 będącego w kolizji z x_1 (tzn. takiej, że $h(x_1) = h(x_2)$).

Collision resistance (odporność na kolizje, tzw. „silna odporność na kolizje"/"strong collision resistance") – trudne obliczeniowo jest wyznaczenie pary argumentów x_1 i x_2 będących w kolizji (tzn. takiej, że $h(x_1) = h(x_2)$).

Chosen prefix collision attack resistance (odporność na atak polegający na poszukiwaniu takich ciągów p_1 i p_2, że dla różnych x_1 i x_2 w kolizji są odpowiednie konkatenacje, tzn. $h(x_1||p_1) = h(x_2||p_2)$).

Length extension attack resistance (odporność na atak polegający na utworzeniu, na podstawie znajomości $h(x_1)$ i długości x_1, takiego ciągu x_2, że bez znajomości x_1 można wyznaczyć $h(x_1||x_2)$).

Paradoks dnia urodzin – wprowadzenie

Zakładając, że prawdopodobieństwo urodzin w określonym dniu roku dla i-tej osoby ma rozkład jednorodny:

$$P(u_i = d_k) = 1/365 \ (k = 1, 2, 3, \ldots, 365),$$

jakie jest prawdopodobieństwo, że dwie „przypadkowo zgromadzone" osoby obchodzą urodziny tego samego dnia, tzn.:

$$u_1 = u_2 = d_k \ (k = 1, 2, 3, \ldots, 365)?$$

Prawdopodobieństwo faktu, że obie osoby obchodzą urodziny w innych dniach:

$$P(u_1 \neq u_2) = 364/365 \approx 0.99726.$$

Prawdopodobieństwo faktu, że obie osoby obchodzą urodziny tego samego dnia:

$$P_{12} = P(u_1 = u_2) = 1 - P(u_1 \neq u_2) = 1 - 364/365 \approx 0.00274.$$

Do powyższej pary „dochodzi" trzecia osoba. Prawdopodobieństwo faktu, że trzy osoby obchodzą urodziny w innych dniach:

$$P(u_1 \neq u_2) \cdot P((u_3 \neq u_1) \wedge (u_3 \neq u_2)) = 364/365 \cdot 363/365 \approx 0.9918.$$

Prawdopodobieństwo faktu, że co najmniej dwie spośród tych trzech osób obchodzą urodziny tego samego dnia:

$$P_{123} = P((u_1 = u_2) \vee (u_1 = u_3) \vee (u_2 = u_3)) = 1 - 364/365 \cdot 363/365 \approx 0.0082.$$

Paradoks dnia urodzin – wprowadzenie (cd.)

„Dochodzi" czwarta osoba.

$$P_{1234} = 1 - 364/365 \cdot 363/365 \cdot 362/365 \approx 0.01636.$$

„Dochodzi" piąta osoba.

$$P_{12345} = 1 - 364 \cdot 363 \cdot 362 \cdot 361/(365)^4 \approx 0.02714.$$

„Dochodzi" szósta osoba.

$$P_{123456} = 1 - 364 \cdot 363 \cdot 362 \cdot 361 \cdot 360/(365)^5 \approx 0.04046.$$

…

„Dochodzi" dziesiąta osoba.

$$P_{1 \ldots 10} = 1 - 364 \cdot 363 \cdot \ldots \cdot 357 \cdot 356/(365)^9 \approx 0.10468.$$

…

„Dochodzi" dwudziesta osoba.

$$P_{1 \ldots 20} = 1 - 364 \cdot 363 \cdot \ldots \cdot 347 \cdot 346/(365)^{19} \approx 0.40326.$$

…

„Dochodzi" dwudziesta trzecia osoba.

$$P_{1 \ldots 23} = 1 - 364 \cdot 363 \cdot \ldots \cdot 344 \cdot 343/(365)^{22} \approx 0.50045 > 1/2.$$

Paradoks dnia urodzin – uzasadnienie

Załóżmy, że prawdopodobieństwo urodzenia się w określonym dniu roku kalendarzowego każdego z członków grupy liczącej k osób ma rozkład jednorodny, tzn.:

$$P(u_i = d) = 1 / 365,$$

gdzie:

u_i - data urodzin osobnika i-tego ($i = 1, 2, ..., k$);

d - numer kolejnego dnia w roku ($d = 1, 2, ..., 365$).

Spróbujmy oszacować oczekiwaną liczbę par osobników w tej populacji obchodzących urodziny w tym samym dniu roku.

Jeżeli uznamy, że fakty narodzin każdego z osobników są zdarzeniami niezależnymi, to wówczas dla dowolnej pary osobników (i, j) prawdopodobieństwo zdarzenia polegającego na tym, że urodzili się w tym samym określonym dniu d, wynosi:

$$P(u_i = d, u_j = d) = P(u_i = d) * P (u_j = d) = 1 / (365)^2 .$$

Paradoks dnia urodzin – uzasadnienie (cd.)

W skali całego roku prawdopodobieństwo to wynosi:

$$P(u_i = u_j) = \sum_{d=1}^{365} P(u_i = d,\ u_j = d) = 365 / (365)^2 = 1 / 365.$$

Definiując zmienną losową :

$$x_{ij} = \begin{cases} 1\text{, gdy osoby } i \text{ oraz } j \text{ obchodzą urodziny w tym samym dniu;} \\ 0 \text{ w przypadku przeciwnym,} \end{cases}$$

można określić wartość oczekiwaną tej zmiennej w przypadku konkretnej pary:

$$E(x_{ij}) = 1 * (1 / 365) + 0 * (1 - 1 / 365) = 1 / 365,$$

oraz wartość oczekiwaną liczby par osobników "wspólnie świętujących" w całej populacji (sumowanie obejmuje wszystkie możliwe pary):

$$\sum_{i=2}^{k} \sum_{j=1}^{i-1} E(xij) = \binom{k}{2} (1/365) = \frac{k(k-1)}{2 * 365}$$

Już dla 28 osób wartość ta wynosi 1,05 (a więc powinna się znaleźć w tej grupie przynajmniej jedna taka para).

Paradoks dnia urodzin a funkcja skrótu

Poszukiwanie pary wiadomości dających tą samą wartość n-bitowej funkcji skrótu (poszukiwanie kolizji) równoważne jest zadaniu znalezienia pary osobników urodzonych w tym samym dniu roku liczącego 2^n dni.

Oczekiwana liczba takich par, w sytuacji gdy "przebadano" k "kandydatur" wynosi:

$$\sum_{i=2}^{k} \sum_{j=1}^{i-1} E(xij) = \binom{k}{2}(1/p) = \frac{k(k-1)}{2 * p} \quad ,$$

gdzie: $p = 2^n$.

Jeżeli wartość funkcji skrótu jest ciągiem n-bitowym, to wówczas znalezienie metodą ataku brutalnego wiadomości, która po skróceniu dałaby taką samą wartość skrótu, jak wartość dana, wymagałoby skracania „przeciętnie" 2^{n-1} losowo wybranych wiadomości.

Znalezienie pary wiadomości dających taki sam skrót wymagałoby jedynie skracania $2^{0.5n}$ losowo wybranych wiadomości. Korzyść jest ewidentna (np. jeśli n = 16, wówczas liczba koniecznych operacji wynosi odpowiednio 32768 i 256).

Atak na podpis cyfrowy wykorzystujący paradoks dnia urodzin

Podpis cyfrowy dla wiadomości m:

$$s = D_d(h(m)).$$

Procedura ataku (n – długość wartości funkcji skrótu w bitach):

- wygenerować $2^{n/2}$ „fałszywych wiadomości” m_i ($i = 1, 2, ..., 2^{n/2}$);
- wygenerować losowo $2^{n/2}$ „fałszywych podpisów” s_i ($i = 1, 2, ..., 2^{n/2}$);
- utworzyć listę $E_e(s_i)$ oraz listę $h(m_i)$;
- znaleźć parę (j, k) taką, że $E_e(s_j) = h(m_k)$.

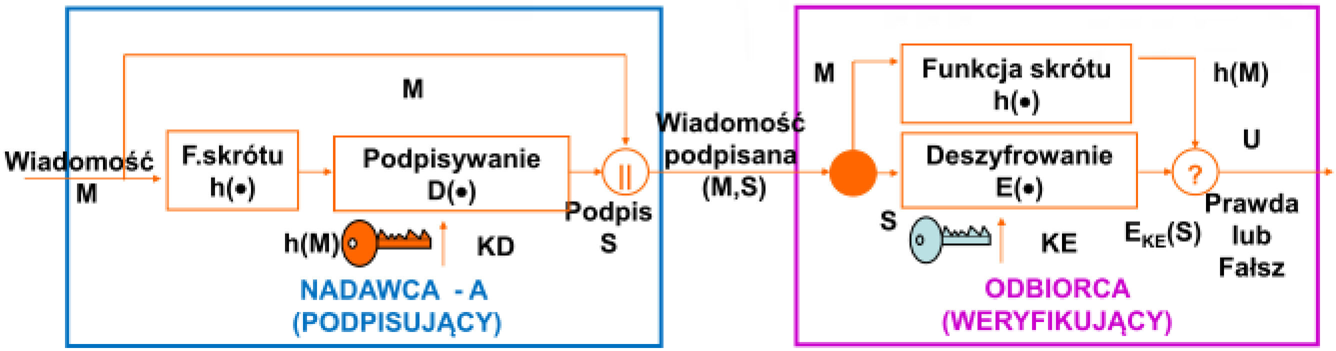
Propozycja zapobiegania atakowi (J.Patarin)

Wykorzystać dwie różne funkcje skrótu h_1 i h_2:

$$s = D_d(h_1(m) + D_d(h_2(m) + D_d(h_1(m)))).$$

Zastosowania funkcji skrótu

Podpis cyfrowy w schemacie podpisu z załącznikiem



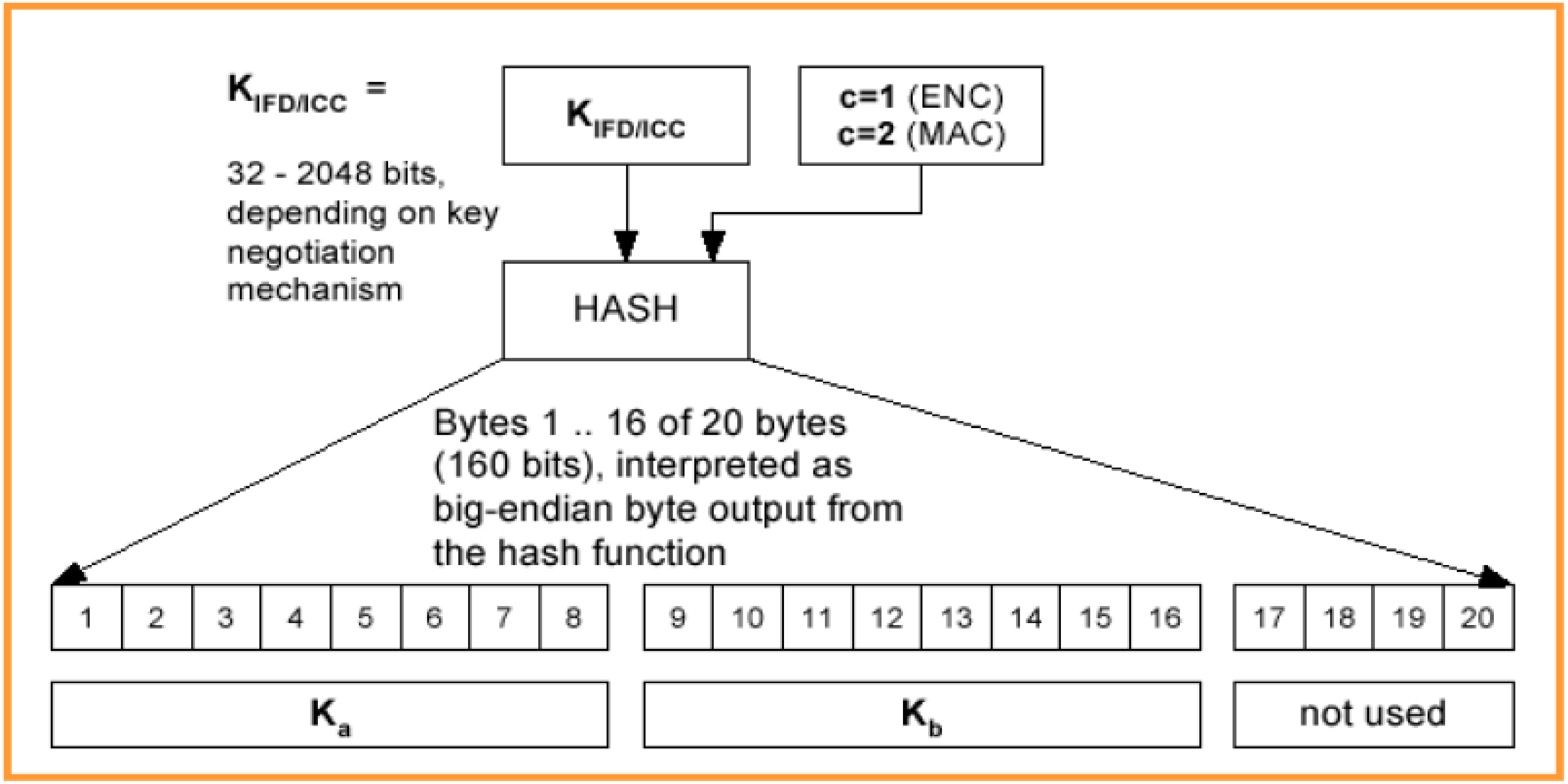
$$S = S_A(\, h\,(M\,)) = D_{KD}(\, h\,(M\,))$$

$$U = V_A(M\,,S) = \begin{cases} \text{prawda, gdy } E_{KE}(S) = h(M\,) \\ \text{fałsz, gdy } E_{KE}(S) \neq h(M\,) \end{cases}$$

Zastosowania funkcji skrótu (cd.)

UZGADNIANIE KLUCZY SESYJNYCH DLA SM

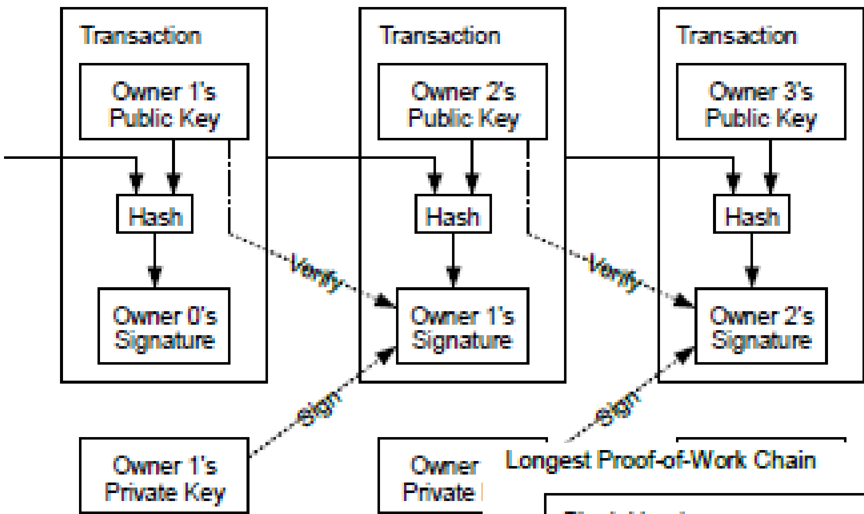
(algorytm 3DES/TDES; źródło: EN 419212-2, ICAO – PKI for MRTD, ETSI TS 102 176-2)



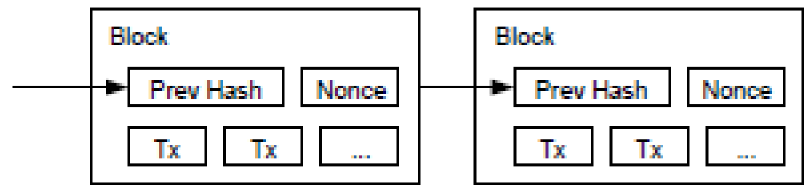
HASH1 = $h_{SM}(K_{IFD/ICC}||c)$ z wartością c = 1
HASH2 = $h_{SM}(K_{IFD/ICC}||c)$ z wartością c = 2 (aktualnie jako h_{SM} wskazywana jest SHA-1)

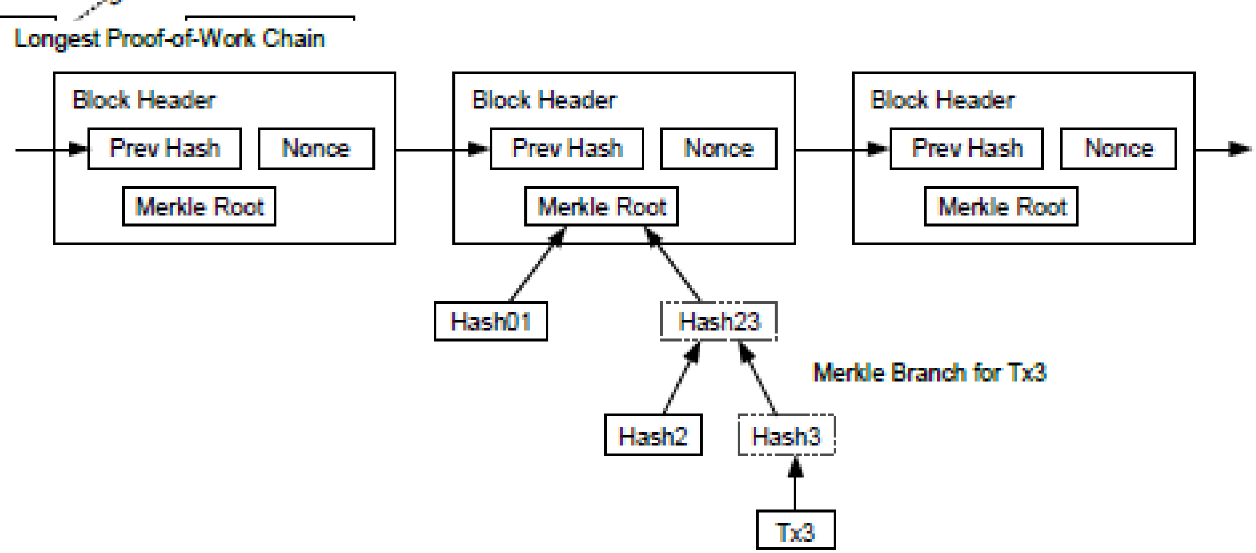
Zastosowania funkcji skrótu (cd.)

Bitcoin: A Peer-to-Peer Electronic Cash System



Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org





Bitcoin

Zastosowania funkcji skrótu (cd.)

Przykład wykorzystania funkcji skrótu do „bezpiecznego” uzgodnienia klucza szyfrującego

A i B chcą uzgodnić m-bitowy klucz szyfrujący k, wykorzystując do tego celu rodzinę zależnych od parametru a funkcji skrótu $h_a(.)$.
Procedura uzgadniania przebiega w następujących krokach :

1. **A wybiera losowo parametr a; następnie losowo wybiera n różnych m-bitowych ciągów r_i ($i = 1, ..., n$), oblicza dla nich $h_a(r_i)$ i zapamiętuje wszystkie pary :**
$$(r_i\ ,\ h_a(\ r_i\ )).$$

2. **A wysyła do B parametr a.**

3. **B wybiera losowo m-bitowy klucz k, oblicza dla tego klucza wartość $h_a(k)$ i wysyła tą wartość do A.**

4. **A sprawdza, czy wśród zapamiętanych przezeń par wartości istnieje takie r_i, że:**
$$h_a(\ r_i\ ) = h_a(k).$$

Jeśli istnieje, wówczas A odsyła do B jednobitową informację 1, potwierdzającą uzgodnienie klucza; jeśli nie, wówczas wysyła jednobitową informację 0, zaś kroki 3 i 4 powtarzane są aż do osiągnięcia uzgodnienia klucza.

Zastosowania funkcji skrótu (cd.)

Przykład wykorzystania funkcji skrótu do „bezpiecznego” uzgodnienia klucza szyfrującego (cd.)

Powyższą procedurę uzgadniania klucza można przyspieszyć modyfikując czynności wykonywane przez strony uzgadniające klucz w krokach 3 i 4 :

3’. *B* wybiera losowo pewną liczbę l m-bitowych kluczy:

$$k_j \ (j = 1, 2, \ldots, l),$$

oblicza dla każdego z kluczy wartości $h_a (k_j)$ i wysyła te wartości w odpowiedniej kolejności do *A* (liczbę l wybiera tak, by prawdopodobieństwo znalezienia się w tablicy par po stronie *A* jednego spośród tych kluczy było względnie wysokie).

4’. *A* sprawdza, czy wśród zapamiętanych przezeń par wartości istnieje takie r_i, że :

$$h_a (r_i) = h_a (k_j).$$

Jeśli istnieje, wówczas *A* odsyła do *B* wartość j wskazującą jednoznacznie uzgodniony klucz; jeśli nie, wówczas wysyła wartość 0, zaś kroki 3’ i 4’ powtarzane są aż do osiągnięcia uzgodnienia klucza.

Zastosowania funkcji skrótu (cd.)

Przykład wykorzystania funkcji skrótu do „bezpiecznego” uzgodnienia klucza szyfrującego (cd.)

Optymalna wartość parametru *n* (ze względu na minimalny oczekiwany czas uzgodnienia klucza) wynosi:

$$n = 2^{0.5\,m}\ .$$

Prawdopodobieństwo *nie uzgodnienia* klucza po *jn* iteracjach kroków 3' i 4' wynosi:

$$(1 - 1/n)^{j\,n} \approx e^{-j},$$

zaś „podsłuchiwacz” usiłujący wyznaczyć klucz metodą ataku brutalnego potrzebuje średnio 2^{m-1} prób do osiągnięcia sukcesu, a zatem zasoby (czasowe i pamięciowe) niezbędne do przełamania tego *protokołu* są proporcjonalne do kwadratu zasobów wystarczających do jego realizacji.

UWAGA: W tym przypadku „birthday paradox” jest „sprzymierzeńcem”!!!

Zastosowania funkcji skrótu (cd.)

Przykład wykorzystania funkcji skrótu do wielokrotnego uwierzytelniania klienta banku

Załóżmy, że klient wydaje dyspozycje (dotyczące jego transakcji) swemu bankowi korzystając z głosowego połączenia telefonicznego albo sms-u. Poniższy protokół wykorzystuje funkcję skrótu (bez klucza) do bezpiecznego wielokrotnego uwierzytelniania.

Faza wstępna:

Bank **generuje losowy ciąg binarny S_0, a następnie iteracyjnie, wykorzystując funkcję skrótu $h(.)$, generuje sekwencję N ciągów binarnych:**

$$S_{i+1} = h(S_i).$$

Bank **przekazuje** ***klientowi*** **fizycznie zabezpieczonym kanałem całą wygenerowaną sekwencję (bądź jedynie S_0, przy założeniu, że klient jest w stanie samodzielnie tą sekwencję odtworzyć); po przekazaniu tych informacji bank niszczy całą informację o wygenerowanych ciągach za wyjątkiem ciągu S_N.**

Przedostatni ciąg S_{N-1} staje się bieżącym kodem identyfikacyjnym klienta.

Zastosowania funkcji skrótu (cd.)

Przykład wykorzystania funkcji skrótu do wielokrotnego uwierzytelniania klienta banku (cd.)

Pierwsza transakcja:

Klient w fazie uwierzytelniania przekazuje **Bankowi** jako swój kod uwierzytelniający ciąg S_{N-1};

Bank sprawdza wiarygodność kodu obliczając:

$$S'_N = h(S_{N-1});$$

jeżeli zachodzi równość $S'_N = S_N$, to uznaje wiarygodność klienta i realizuje wydaną dyspozycję, zapamiętując jako nowy kod uwierzytelniający klienta ciąg binarny S_{N-1}; w przeciwnym wypadku odmawia obsługi;

Klient w przypadku pozytywnego uwierzytelniania przyjmuje jako swoją nową wartość kodu uwierzytelniającego ciąg binarny S_{N-2}.

Kolejne transakcje:

Rolę kodu uwierzytelniającego klienta spełniają kolejne ciągi binarne S_i, do momentu, gdy w ostatniej transakcji kodem uwierzytelniającym będzie S_0; wtedy **Bank** powinien wygenerować kolejny ciąg $\{ S_i \}$ i przekazać go *klientowi*.

Struktura funkcji skrótu

Funkcja skrótu jest funkcją jednokierunkową przetwarzającą ciągi binarne o „dowolnej" długości (w tym ciąg „pusty" o długości 0 bitów) na ciągi binarne o tej samej długości, charakterystycznej dla danej funkcji skrótu.

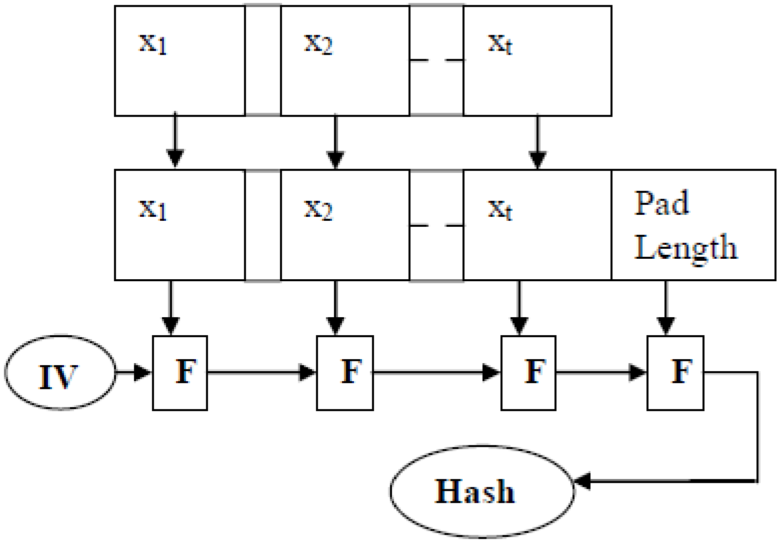
Wejściowy ciąg binarny („skracana" wiadomość) jest dzielony na bloki o długości charakterystycznej dla algorytmu funkcji skrótu i często uzupełniany blokiem (lub częścią bloku), w którym umieszcza się informację o długości wejściowego ciągu binarnego.

Struktury funkcji skrótu różnią się sposobem „wiązania" bloków (wartość funkcji skrótu musi zależeć od wszystkich bloków) :

- **schemat „klasyczny" Ralpha Merkle'a-Ivana Dåmgarda (najpopularniejszy) – sekwencyjne stosowanie tzw. funkcji kompresji;**
- **schemat drzewa Merkle'a-Dåmgarda – drzewo funkcji kompresji;**
- **schemat „gąbki" (sponge construction) – zastosowanie permutacji oraz naprzemiennego „pochłaniania" (absorbing) wiadomości i „wyciskania" (squeezing) skrótu;**

Struktura funkcji skrótu (cd.)

Schemat Merkle'a-Dåmgarda



WEJŚCIE: wiadomość $M \in \{0, 1\}^*$
WYJŚCIE: skrót $h(M) \in \{0, 1\}^n$

- $M \leftarrow M||1||0z$, gdzie z jest najmniejszą liczbą całkowitą spełniającą warunek $m|(|M| + 1 + z)$; opcjonalnie dodać blok m bitów z zakodowaną informacją o początkowej długości wiadomości M;
- podzielić wiadomość M na bloki o długości m: $M = (x_1, x_2, \ldots, x_{t+1})$;
- wykonać odpowiednią liczbę iteracji funkcji kompresji F:

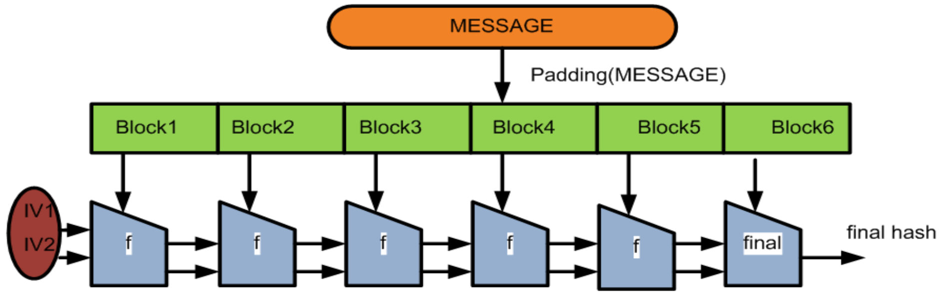
  $H_0 \leftarrow IV$; (często $IV = 0$)
  for($i \leftarrow 0$; $i < t+1$; i++)
  $H_{i+1} \leftarrow F(H_i||x_{i+1})$;
- zwrócić $h(M) = H_{t+1}$.

Struktura funkcji skrótu (cd.)

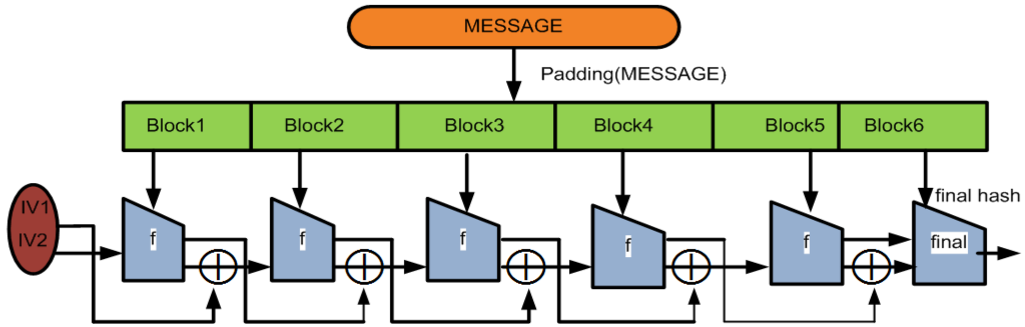
Schemat Merkle'a-Dåmgarda jest podatny na „chosen prefix attack", a dla pewnych funkcji skrótu także na „length extension attack".

Zapobiegać im mają takie konstrukcje jak:

wide-pipe →



fast wide pipe →

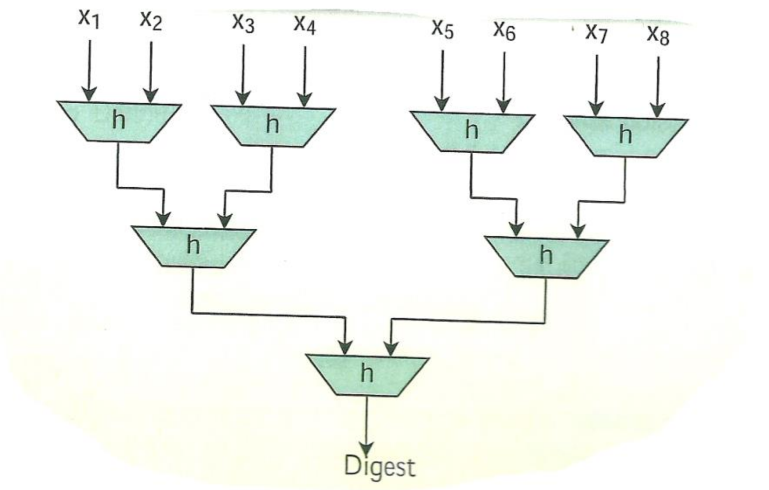


i inne rozwiązania „potokowo-rurowe".

`Źródło: https://en.wikipedia.org/wiki/Merkle-Damgård_construction`

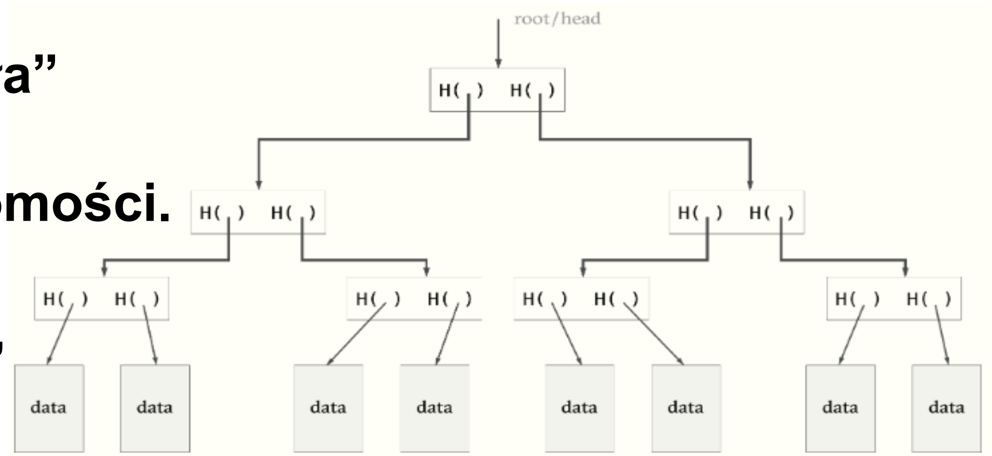
Struktura funkcji skrótu (cd.)

Schemat drzewa Merkle'a-Dåmgarda

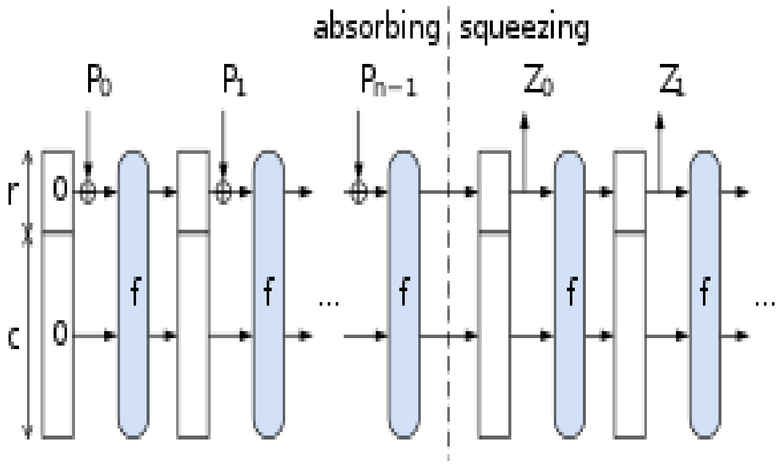


Kolejne bloki skracanej wiadomości to $x_1, x_2, \ldots, x_n$;
h to dowolna funkcja skrótu.

Koncepcja „wiązania bloków" wg. tego schematu „wykroczyła" daleko poza pierwotną rolę funkcji skrótu dla jednej wiadomości. Jako „Merkle-tree" pojawia się w rozwiązaniach „blockchain", i nie tylko.



Struktura funkcji skrótu (cd.)



Schemat „gąbki”

Pomysłodawca - Guido Bertoni (2007)

Kolejne bloki skracanej wiadomości to p_1, p_2, …, $p_{n-1;}$
bloki wyjściowe skrótu to z_0, z_1, …,;
r i c to ciągi binarne stanu wewnętrznego, z których wyłącznie część r jest XOR-owana z blokami skracanej wiadomości, zaś w fazie „wyciskania gąbki” wyprowadzana jako blok wyjściowy;
część c stanu wewnętrznego (tzw.”capacity”) pełni rolę „pochłaniacza”, zaś funkcja f jest zazwyczaj pseudolosową permutacją stanu wewnętrznego.

Struktura funkcji skrótu (cd.)

Rozwiązania dla funkcji kompresji:

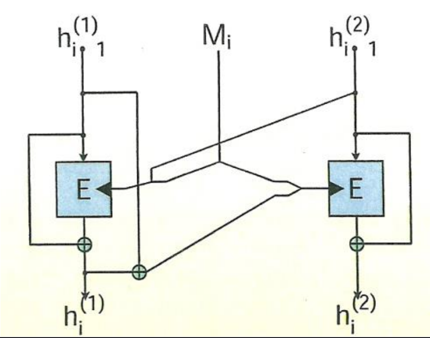
- **Wykorzystanie blokowych algorytmów kryptografii symetrycznej**

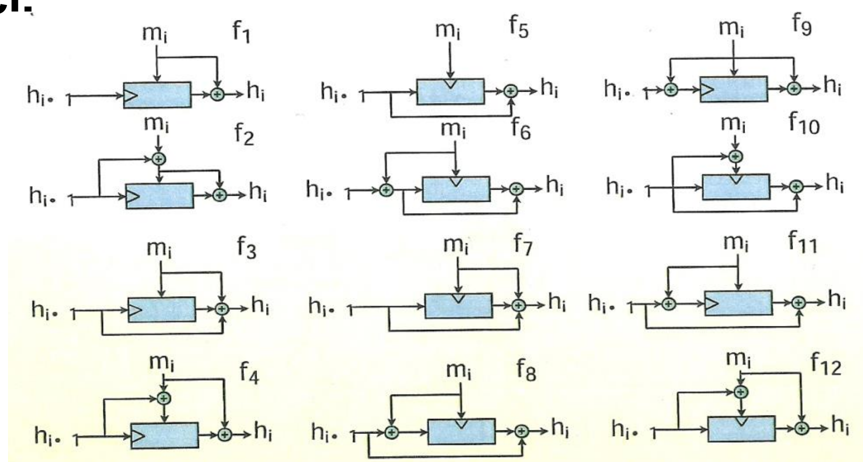
ZALETY

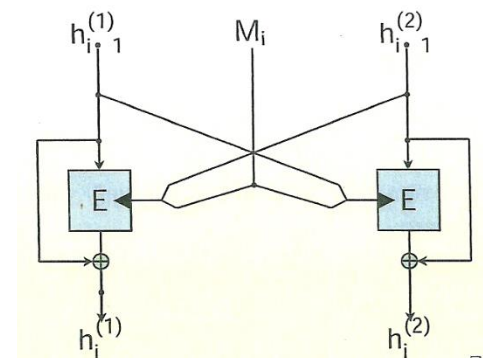
- szyfry blokowe łatwo zaadaptować dla potrzeb funkcji skrótu;
- wykorzystanie istniejących dobrze rozwiniętych technologii;
- dobre algorytmy blokowe wytrzymują „próbę czasu” (np. AES, który stał się podstawą algorytmu „Whirlpool”).

WADY

- skracanie wolniejsze, niż w przypadku funkcji „specjalnie” projektowanych;
- niezgodność między wyznaczaniem kluczy a rozszerzaniem wiadomości.







Struktura funkcji skrótu (cd.)

Rozwiązania dla funkcji kompresji:

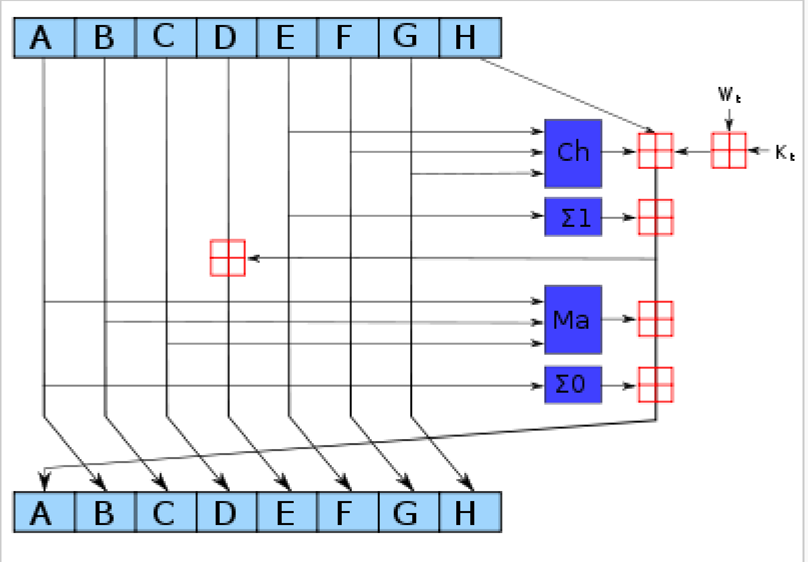
- **Funkcje „specjalnie" projektowane (np.: MD5, SHA-x, RIPEMD-160)**

  ZALETY

  - **specjalnie projektowane dla potrzeb skracania;**
  - **realizowane żądanie dyfuzji;**
  - **podstawowe operacje dobierane pod kątem maksymalnej prędkości przetwarzania.**

  WADY

  - **nie udowodnione bezpieczeństwo;**
  - **rozwój względnie opóźniony w stosunku do kryptografii symetrycznej, z reguły „klasyczna" struktura Merkle'a-Damgårda podatna na współczesne ataki.**



One iteration in a SHA-2 family compression function. The blue components perform the following operations:

$$\mathrm{Ch}(E,F,G) = (E \land F) \oplus (\neg E \land G)$$
$$\mathrm{Ma}(A,B,C) = (A \land B) \oplus (A \land C) \oplus (B \land C)$$
$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$
$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256. The red $\boxplus$ is an addition modulo 2^{32}.

Struktura funkcji skrótu (cd.)

Rozwiązania dla funkcji kompresji:

- **funkcje oparte o trudne problemy obliczeniowe (np.: MASH-1)**

ZALETY

- możliwość dowodzenia bezpieczeństwa („provable secure”);
- elastyczność w doborze wielkości parametrów decydujących o bezpieczeństwie.

WADY

- niskie prędkości przetwarzania;
- podatność na ataki za pomocą algorytmów kwantowych.

Very Smooth Hash (VSH)

Let $p_1 = 2, p_2 = 3, p_3 = 5, \ldots$ be the sequence of primes. Let n be a large RSA composite. Let k, the block length, be the largest integer such that $\prod_{i=1}^{k} p_i < n$. Let m be a be an l-bit message to be hashed, consisting of bits $m_1, m_2, \ldots, m_l$, and assume that $l < 2^k$. To compute the hash of m:

1. Let $x_0 = 1$.
2. Let $L = \lceil l/k \rceil$ the number of blocks. Let $m_i = 0$ for $l < i \leq Lk$ (padding).
3. Let $l = \sum_{i=1}^{k} l_i 2^{i-1}$ with $l_i \in \{0, 1\}$ be the binary representation of the message length l and define $m_{Lk+i} = l_i$ for $1 \leq i \leq k$.
4. For $j = 0, 1, \ldots, L$ in succession compute
$$x_{j+1} = x_j^2 \prod_{i=1}^{k} p_i^{m_{(jk+i)}} \mod n.$$
5. Return x_{L+1}.

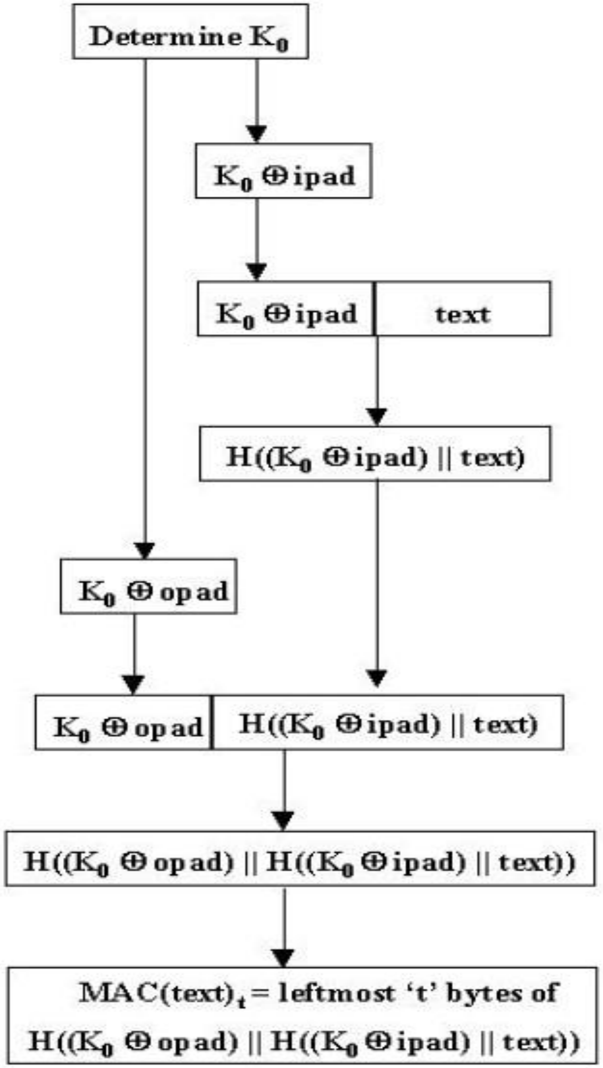
Discrete Logarithm Hash

- 260 to 132 bit contractor:
$$\mathcal{X} = [0, 2^{260} - 1], \mathcal{Y} = [0, 2^{132} - 1]$$
- $\mathcal{K}$ - keys $K = (p, q, \alpha, \beta)$ should satisfy:
  - p and q are prime with $p = 2q + 1$
  - α and β are **primitive** in $\mathbb{Z}_p^*$
  - Bit-lengths: $|p| = 132, |q^2| = 261$
- $\mathcal{H}$ - hash functions defined by
$$h_K(n) = \alpha^{\lfloor \frac{n}{q} \rfloor} \beta^{n \bmod q} \mod p$$

HMAC – funkcja skrótu z kluczem

HMAC (Keyed-Hash Message Authentication Code) – umożliwia jednoczesną realizację usług autentyczności i integralności danych.

(RFC 2104, FIPS PUB 198)



H – funkcja skrótu (SHA-1,SHA-256,RIPEMD-160,…)

B – wielkość bloku wejściowego funkcji skrótu (w bajtach)

L - wielkość bloku wyjściowego funkcji skrótu (w bajtach)

K – **C**-bajtowy tajny klucz współdzielony przez twórcę i weryfikatora HMAC

K_0 – **B**-bajtowy klucz uzyskiwany z klucza **K**

$C = B \Rightarrow K_0 = K$

$C < B \Rightarrow K_0 = K\|0x00\|\ldots\|0x00$

$C > B \Rightarrow K_0 = H(K)\|0x00\|\ldots\|0x00$

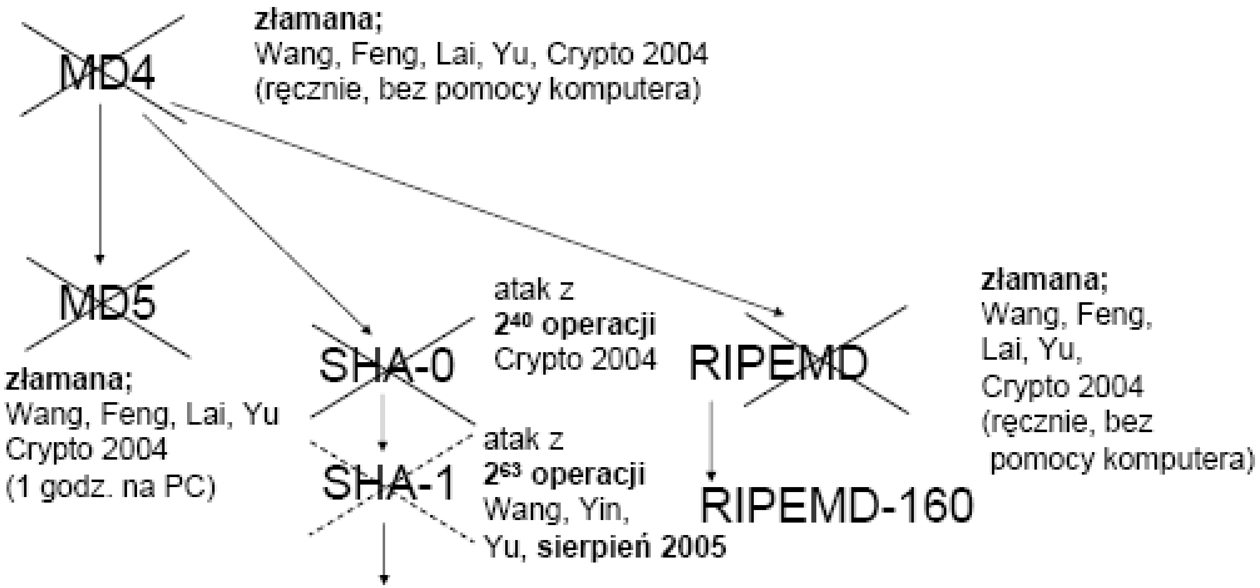
ipad – ciąg **B** bajtów o wartości 0x36 każdy

opad - ciąg **B** bajtów o wartości 0x5C każdy

t – liczba bajtów HMAC ($4 \leq L/2 \leq t \leq L$)

text - uwierzytelniane dane o długości **n** bitów ($0 \leq n < 2^B - 8B$)

„Bezpieczne” funkcje skrótu



SHA-224, SHA-256, SHA-384, SHA-512 = SHA-2

Krzysztof Gaj – Enigma’2007

„Bezpieczne” funkcje skrótu (cd.)

Collisions for Hash Functions

MD4, MD5, HAVAL-128 and RIPEMD

Xiaoyun Wang[1], Dengguo Feng[2], Xuejia Lai[3], Hongbo Yu[1]

The School of Mathematics and System Science, Shandong University, Jinan250100, China[1]

Institute of Software, Chinese Academy of Sciences, Beijing100080, China[2]

Dept. of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai, China[3]

xywang@sdu.edu.cn[1]

revised on August 17, 2004

„Bezpieczne" funkcje skrótu (cd.)

1 Collisions for MD5

MD5 is the hash function designed by Ron Rivest [9] as a strengthened version of MD4 [8]. In 1993 Bert den Boer and Antoon Bosselaers [1] found pseudo-collision for MD5 which is made of the same message with two different sets of initial value. H. Dobbertin[3] found a free-start collision which consists of two different 512-bit messages with a chosen initial value IV_0'.

$$IV_0' : A_0' = 0x12AC2375,\ B_0' = 0x3B341042, C_0' = 0x5F62B97C\ , D_0' = 0x4BA763ED$$

Our attack can find many real collisions which are composed of two 1024-bit messages with the original initial value IV_0 of MD5:

$$IV_0 : A_0 = 0x67452301, B_0 = 0xefcdab89, C_0 = 0x98badcfe, D_0 = 0x10325476$$

$$M' = M + \Delta C_1, \Delta C_1 = (0,0,0,0,2^{31},\ldots,2^{15},\ldots,2^{31},0)$$

$$N_i' = N_i + \Delta C_2, \Delta C_2 = (0,0,0,0,2^{31},\ldots,-2^{15},\ldots,2^{31},0)$$

(non-zeros at position 4,11 and 14)

such that

$$MD5(M, N_i) = MD5(M', N_i').$$

On IBM P690, it takes about one hour to find such M and M', after that, it takes only 15 seconds to 5 minutes to find N_i and N_i', so that (M, N_i) and (M', N_i') will produce the same hash same value. Moreover, our attack works for any given initial value.

„Bezpieczne” funkcje skrótu (cd.)

3 Collisions for MD4

MD4 is designed by R. L. Rivest[8] . Attack of H. Dobbertin in Eurocrypto'96[2] can find collision with probability $1/2^{22}$. Our attack can find collision with hand calculation, such that

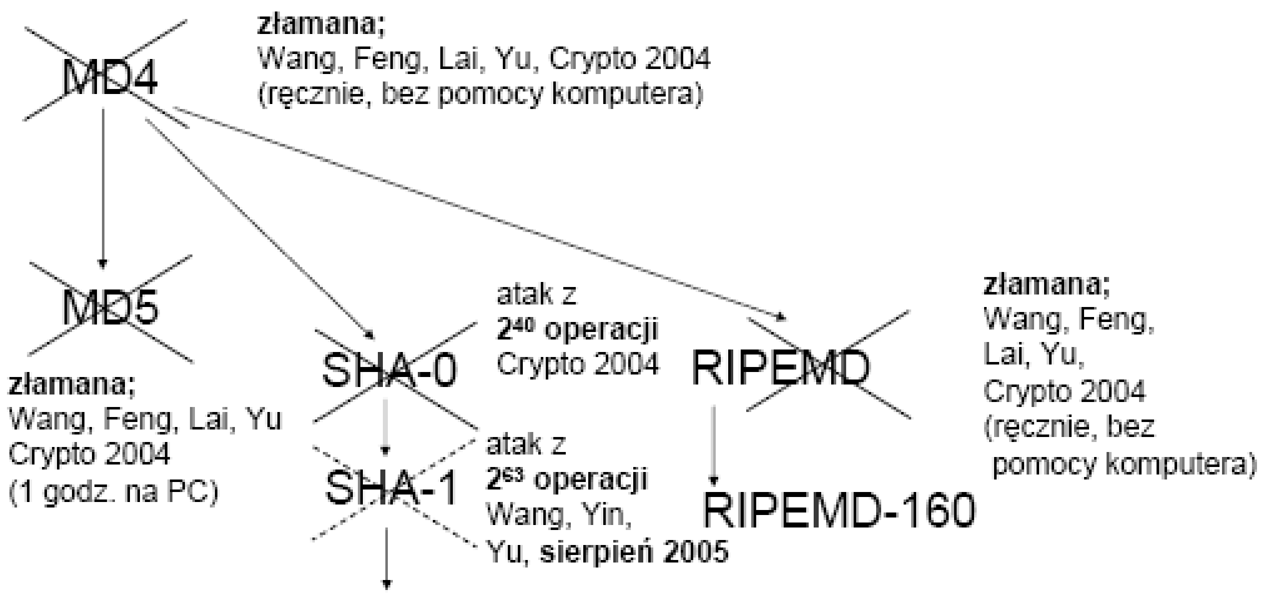
$$M' = M + \Delta C, \Delta C = (0, 2^{31}, -2^{28} + 2^{31}, 0,0,0,0,0,0,0,0,0, -2^{16}, 0,0,0)$$

and $MD4(M) = MD4(M')$.

M_1	4d7a9c83	56cb927a	b9d5a578	57a7a5ee	de748a3c	dcc366b3	b683a020	3b2a5d9f
	c69d71b3	f9e99198	d79f805e	a63bb2e8	45dd8e31	97e31fe5	2794bf08	b9e8c3e9
M_1	4d7a9c83	d6cb927a	29d5a578	57a7a5ee	de748a3c	dcc366b3	b683a020	3b2a5d9f
	c69d71b3	f9e99198	d79f805e	a63bb2e8	45dc8e31	97e31fe5	2794bf08	b9e8c3e9
H	5f5c1a0d	71b36046	1b5435da	9b0d807a				
M_2	4d7a9c83	56cb927a	b9d5a578	57a7a5ee	de748a3c	dcc366b3	b683a020	3b2a5d9f
	c69d71b3	f9e99198	d79f805e	a63bb2e8	45dd8e31	97e31fe5	f713c240	a7b8cf69
M_2	4d7a9c83	d6cb927a	29d5a578	57a7a5ee	de748a3c	dcc366b3	b683a020	3b2a5d9f
	c69d71b3	f9e99198	d79f805e	a63bb2e8	45dc8e31	97e31fe5	f713c240	a7b8cf69
H	e0f76122	c429c56c	ebb5e256	b809793				

Table 3 Two pairs of collisions for MD4

„Bezpieczne” funkcje skrótu (cd.)



SHA-224, SHA-256, SHA-384, SHA-512 = SHA-2

Krzysztof Gaj – Enigma’2007

2 listopada 2007 – NIST „wzywa” do zgłaszania kandydatur na SHA-3

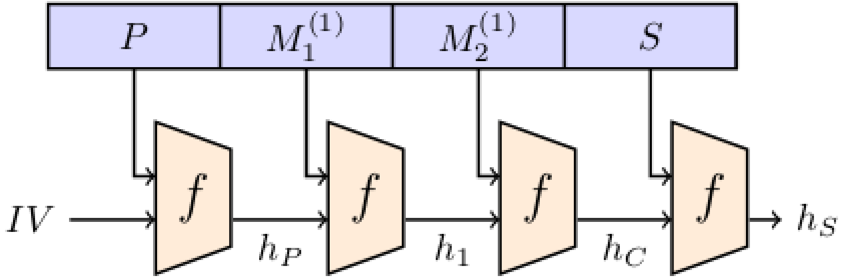
2 października 2012 – wybór zwycięzcy (SHA-3 = Keccak)

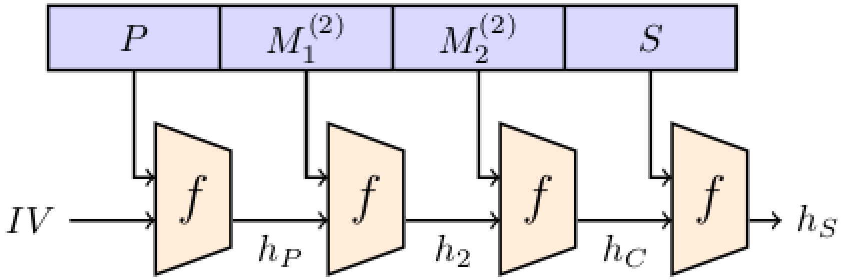
February 23, 2017

A team from Google and CWI Amsterdam just announced it: they produced the first SHA-1 hash collision. The attack required over 9,223,372,036,854,775,808 SHA-1 computations, the equivalent processing power as 6,500 years of single-CPU computations and 110 years of single-GPU computations. While this may seem overwhelming, this is a practical attack if you are, lets say, a state-sponsored attacker. Or if you control a large enough botnet. Or if you are just able to spend some serious money on cloud computing. It's doable. Make no mistake, this is not a brute-force attack, that would take around 12,000,000 single-GPU years to complete.

the SHAttered Attack

Because of the Merkle–Damgård construction of SHA-1, one can choose to alter the differences between the iterations in order to improve the differences to match his needs. In the case of the SHAttered Attack, they chose an initial prefix (P), then later on the next blocks they introduce a difference ($M_1^{(1)}$,$M_1^{(2)}$) and remove it ($M_2^{(1)}$ and $M_2^{(2)}$). At this point they already have their collision. They just need to continue with the same following blocks (S), leading to a collision on the whole input.

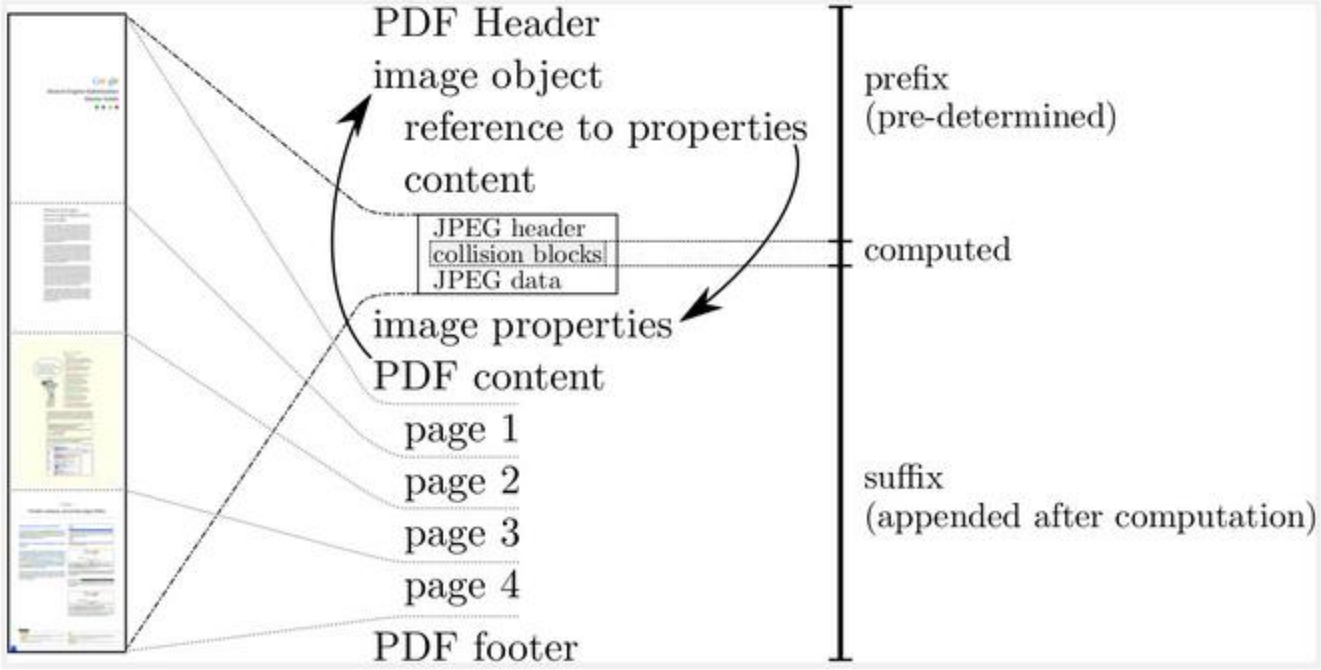




$$\mathrm{SHA}-1(P||M_1^{(1)} \parallel M_2^{(1)} \parallel S) = \mathrm{SHA}-1(P||M_1^{(2)} \parallel M_2^{(2)} \parallel S)$$

CV_0	4e	a9	62	69	7c	87	6e	26	74	d1	07	f0	fe	c6	79	84	14	f5	bf	45
$M_1^{(1)}$			7f	46	dc	93	a6	b6	7e	01	3b	02	9a	aa	1d	b2	56	0b		
			45	ca	67	d6	88	c7	f8	4b	8c	4c	79	1f	e0	2b	3d	f6		
			14	f8	6d	b1	69	09	01	c5	6b	45	c1	53	0a	fe	df	b7		
			60	38	e9	72	72	2f	e7	ad	72	8f	0e	49	04	e0	46	c2		
$CV_1^{(1)}$	8d	64	d6	17	ff	ed	53	52	eb	c8	59	15	5e	c7	eb	34	f3	8a	5a	7b
$M_2^{(1)}$			30	57	0f	e9	d4	13	98	ab	e1	2e	f5	bc	94	2b	e3	35		
			42	a4	80	2d	98	b5	d7	0f	2a	33	2e	c3	7f	ac	35	14		
			e7	4d	dc	0f	2c	c1	a8	74	cd	0c	78	30	5a	21	56	64		
			61	30	97	89	60	6b	d0	bf	3f	98	cd	a8	04	46	29	a1		
CV_2	1e	ac	b2	5e	d5	97	0d	10	f1	73	69	63	57	71	bc	3a	17	b4	8a	c5

CV_0	4e	a9	62	69	7c	87	6e	26	74	d1	07	f0	fe	c6	79	84	14	f5	bf	45
$M_1^{(2)}$			73	46	dc	91	66	b6	7e	11	8f	02	9a	b6	21	b2	56	0f		
			f9	ca	67	cc	a8	c7	f8	5b	a8	4c	79	03	0c	2b	3d	e2		
			18	f8	6d	b3	a9	09	01	d5	df	45	c1	4f	26	fe	df	b3		
			dc	38	e9	6a	c2	2f	e7	bd	72	8f	0e	45	bc	e0	46	d2		
$CV_1^{(2)}$	8d	64	c8	21	ff	ed	52	e2	eb	c8	59	15	5e	c7	eb	36	73	8a	5a	7b
$M_2^{(2)}$			3c	57	0f	eb	14	13	98	bb	55	2e	f5	a0	a8	2b	e3	31		
			fe	a4	80	37	b8	b5	d7	1f	0e	33	2e	df	93	ac	35	00		
			eb	4d	dc	0d	ec	c1	a8	64	79	0c	78	2c	76	21	56	60		
			dd	30	97	91	d0	6b	d0	af	3f	98	cd	a4	bc	46	29	b1		
CV_2	1e	ac	b2	5e	d5	97	0d	10	f1	73	69	63	57	71	bc	3a	17	b4	8a	c5

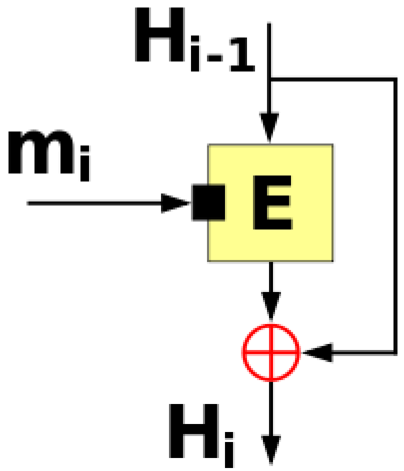

PDF Header
image object
reference to properties
content
JPEG header
collision blocks
JPEG data
image properties
PDF content
page 1
page 2
page 3
page 4
PDF footer
prefix
(pre-determined)
computed
suffix
(appended after computation)

From Collisions to Chosen-Prefix Collisions
Application to Full SHA-1

Gaëtan Leurent[1] and Thomas Peyrin[2,3]

[1] Inria, France
[2] Nanyang Technological University, Singapore
[3] Temasek Laboratories, Singapore

gaetan.leurent@inria.fr, thomas.peyrin@ntu.edu.sg



On 24 April 2019 a paper by Gaëtan Leurent and Thomas Peyrin presented at Eurocrypt 2019 described an enhancement to the previously best chosen-prefix attack in Merkle-Damgård-like digest functions based on Davies-Meyer block ciphers

„Bezpieczne” funkcje skrótu (cd.)

Stan wiedzy dotyczącej „odporności” pewnych funkcji skrótu

- Nie udało się zademonstrować żadnego skutecznego ataku
- Atak zademonstrowany „w teorii”
- Atak zademonstrowany „w praktyce”

Collision resistance [edit]

Main article: Collision attack

Hash function	Security claim	Best attack	Publish date	Comment
MD5	2^{64}	2^{18} time	2013-03-25	This attack takes seconds on a regular PC. Two-block col
SHA-1	2^{80}	$2^{61.2}$	2020-01-08	Paper by Gaëtan Leurent and Thomas Peyrin[2]
SHA256	2^{128}	31 of 64 rounds ($2^{65.5}$)	2013-05	
SHA512	2^{256}	24 of 80 rounds ($2^{32.5}$)	2008-11	
SHA-3	Up to 2^{512}	6 of 24 rounds (2^{50})	2017	
BLAKE2s	2^{128}	2.5 of 10 rounds (2^{112})	2009-05	
BLAKE2b	2^{256}	2.5 of 12 rounds (2^{224})	2009-05	

Chosen prefix collision attack [edit]

Hash function	Security claim	Best attack	Publish date
MD5	2^{64}	2^{39}	2009-06-16
SHA-1	2^{80}	$2^{63.4}$	2020-01-08
SHA256	2^{128}		
SHA512	2^{256}		

Źródło: https://en.wikipedia.org/wiki/Hash_function_security_summary

„Bezpieczne” funkcje skrótu (cd.)

Stan wiedzy dotyczącej „odporności” pewnych funkcji skrótu

- Nie udało się zademonstrować żadnego skutecznego ataku
- Atak zademonstrowany „w teorii”
- Atak zademonstrowany „w praktyce”

Collision resistance [edit]

Hash function	Security claim	Best attack	Publish date
GOST	2^{128}	2^{105}	
HAVAL-128	2^{64}	2^{7}	
MD2	2^{64}	$2^{63.3}$ time, 2^{52} memory	
MD4	2^{64}	3 operations	
PANAMA	2^{128}	2^{6}	
RIPEMD (original)	2^{64}	2^{18} time	
RadioGatún	Up to 2^{608}[21]	2^{704}	

Preimage resistance [edit]

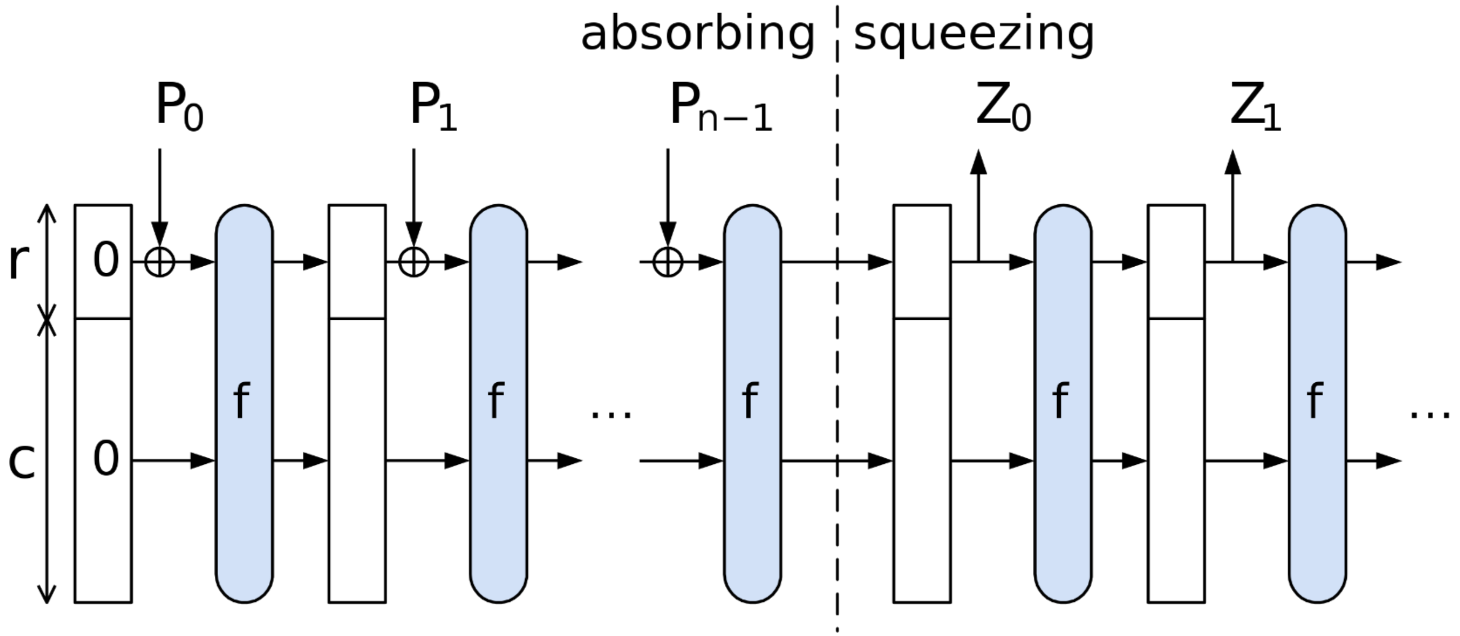
Hash function	Security claim	Best attack	Publish date
GOST	2^{256}	2^{192}	2008-08-18
MD2	2^{128}	2^{73} time, 2^{73} memory	2008
MD4	2^{128}	2^{102} time, 2^{33} memory	2008-02-10
RIPEMD (original)	2^{128}	35 of 48 rounds	2011
RIPEMD-128	2^{128}	35 of 64 rounds	2011
RIPEMD-160	2^{160}	31 of 80 rounds	2011
Streebog	2^{512}	2^{266} time, 2^{259} data	2014-08-29
Tiger	2^{192}	$2^{188.8}$ time, 2^{8} memory	2010-12-06

Źródło: `https://en.wikipedia.org/wiki/Hash_function_security_summary`

KECCAK (SHA-3)

(Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche)

Konstrukcja „gąbki” (*Sponge*)



The KECCAK reference: http://keccak.noekeon.org/

KECCAK (SHA-3)
(Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche)

Konstrukcja „gąbki" (*Sponge*)

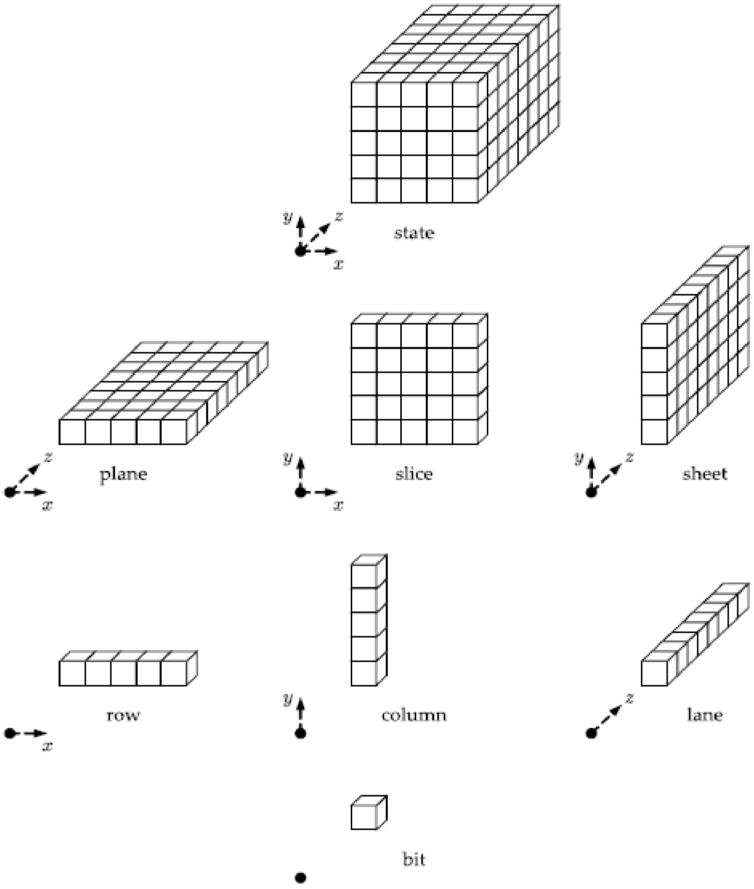
KECCAK-$f[b]$ is an iterated permutation, consisting of a sequence of n_r rounds R, indexed with i_r from 0 to $n_r - 1$. A round consists of five steps:

$$\mathrm{R} = \iota \circ \chi \circ \pi \circ \rho \circ \theta, \text{ with}$$

$$\begin{aligned}
\theta: \quad & a[x][y][z] && \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1], \\
\rho: \quad & a[x][y][z] && \leftarrow a[x][y][z - (t+1)(t+2)/2], \\
& && \quad \text{with } t \text{ satisfying } 0 \le t < 24 \text{ and } \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \text{ in } \mathrm{GF}(5)^{2\times 2}, \\
& && \quad \text{or } t = -1 \text{ if } x = y = 0, \\
\pi: \quad & a[x][y] && \leftarrow a[x'][y'], \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}, \\
\chi: \quad & a[x] && \leftarrow a[x] + (a[x+1] + 1)a[x+2], \\
\iota: \quad & a && \leftarrow a + \mathrm{RC}[i_r].
\end{aligned}$$

The KECCAK reference: http://keccak.noekeon.org/

KECCAK (SHA-3)



The KECCAK reference: http://keccak.noekeon.org/

KECCAK (SHA-3)

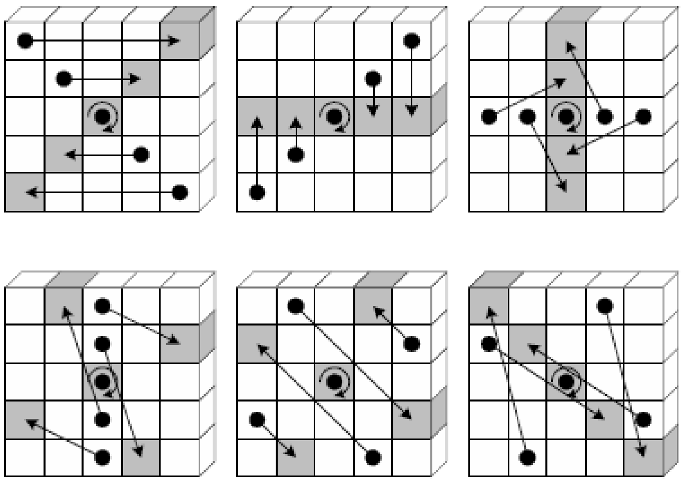


Figure 2.3: π applied to a slice. Note that $x = y = 0$ is depicted at the center of the slice.

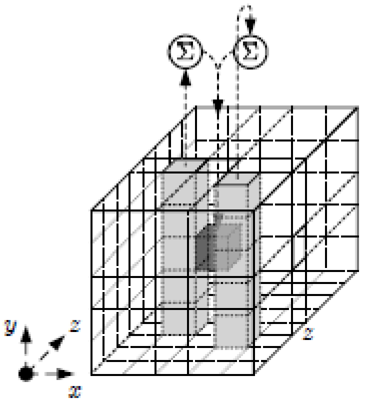


Figure 2.2: θ applied to a single bit

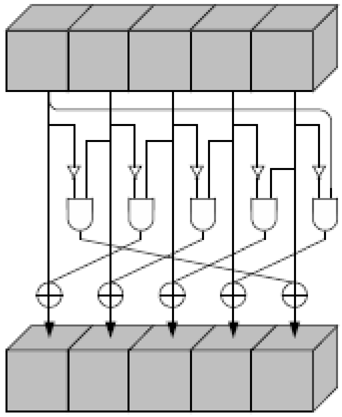


Figure 2.1: χ applied to a single row

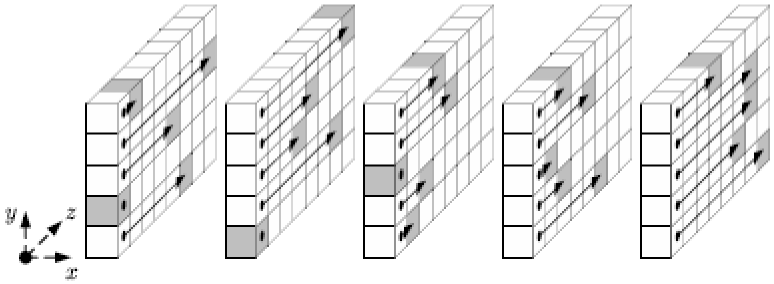


Figure 2.4: ρ applied to the lanes. Note that $x = y = 0$ is depicted at the center of the slices.

The KECCAK reference: http://keccak.noekeon.org/

FIPS PUB 202

FEDERAL INFORMATION PROCESSING STANDARDS
PUBLICATION

SHA-3 Standard: Permutation-Based Hash an
Extendable-Output Functions

CATEGORY: COMPUTER SECURITY SUBCATEGORY: CRYPT

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.FIPS.202

August 2015

Abstract

This Standard specifies the Secure Hash Algorithm-3 (SHA-3) family of functions on binary data. Each of the SHA-3 functions is based on an instance of the KECCAK algorithm that NIST selected as the winner of the SHA-3 Cryptographic Hash Algorithm Competition. This Standard also specifies the KECCAK-*p* family of mathematical permutations, including the permutation that underlies KECCAK, in order to facilitate the development of additional permutation-based cryptographic functions.

The SHA-3 family consists of four cryptographic hash functions, called SHA3-224, SHA3-256, SHA3-384, and SHA3-512, and two extendable-output functions (XOFs), called SHAKE128 and SHAKE256.

Hash functions are components for many important information security applications, including 1) the generation and verification of digital signatures, 2) key derivation, and 3) pseudorandom bit generation. The hash functions specified in this Standard supplement the SHA-1 hash function and the SHA-2 family of hash functions that are specified in FIPS 180-4, the Secure Hash Standard.

Extendable-output functions are different from hash functions, but it is possible to use them in similar ways, with the flexibility to be adapted directly to the requirements of individual applications, subject to additional security considerations.

$$\mathrm{SHAKE128}(M, d) = \mathrm{KECCAK}[256]\,(M \,\|\, 1111, d),$$
$$\mathrm{SHAKE256}(M, d) = \mathrm{KECCAK}[512]\,(M \,\|\, 1111, d).$$

$$\mathrm{RawSHAKE128}(J, d) = \mathrm{KECCAK}[256]\,(J \,\|\, 11, d),$$
$$\mathrm{RawSHAKE256}(J, d) = \mathrm{KECCAK}[512]\,(J \,\|\, 11, d).$$

$$\mathrm{SHAKE128}(M, d) = \mathrm{RawSHAKE128}\,(M \,\|\, 11, d),$$
$$\mathrm{SHAKE256}(M, d) = \mathrm{RawSHAKE256}\,(M \,\|\, 11, d).$$

„Potomstwo” KECCAK’a

FIPS PUB 202 - SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions

August 2015 - The FIPS 202 standard defines:

NIST Special Publication 800-185 - SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash

December 2016 - The SP 800-185 standard defines:

- t
- t
- t
- t ... XOF256),

3GPP TS 35.231 - Specification of the TUAK algorithm set: A second example algorithm set for the 3GPP f1, f1*, f2, f3, f4, f5 and f5*

October 2014 - The 3GPP TS 35.231 standard defines:

...bile telephony, based on Keccak.

KangarooTwelve

August 2016 - This docume...

- the KangarooTwelve...
- the MarsupilamiFour...

CAESAR submission: Ketje v2

September 2016 - This document defines:

- the MonkeyDuplex construction,
- the MonkeyWrap authenticated encryptio...
- the Ketje authenticated encryption schem...

...and so on...

Źródło: `https://keccak.team/specifications.html`

Koniec części 5