Hash function

A function which maps strings of bits to fixed-length strings of bits, satisfying the following two properties:

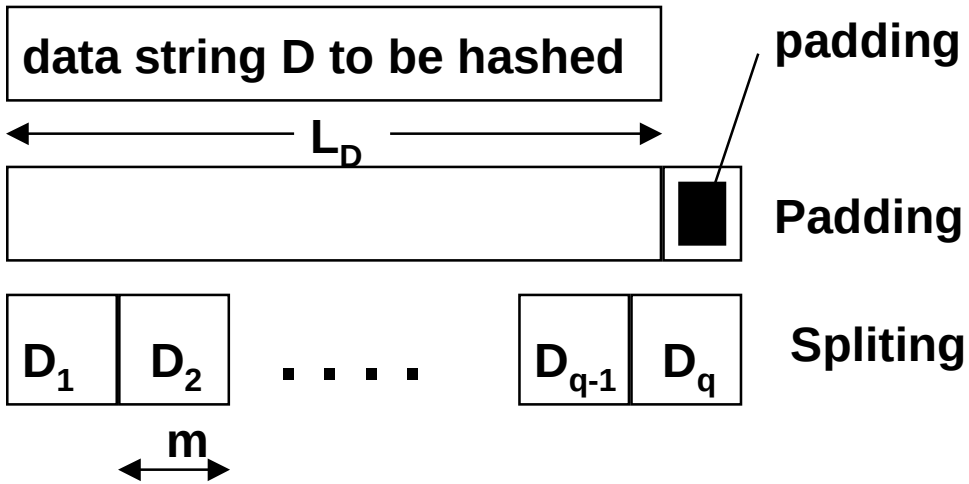
- **it is computationally infeasible to find for a given output an input which maps to this output,**
- **it is computationally infeasible to find for a given input a second input which maps to the same output.**

Collision-resistant hash-function: A hash-function satisfying the following property:

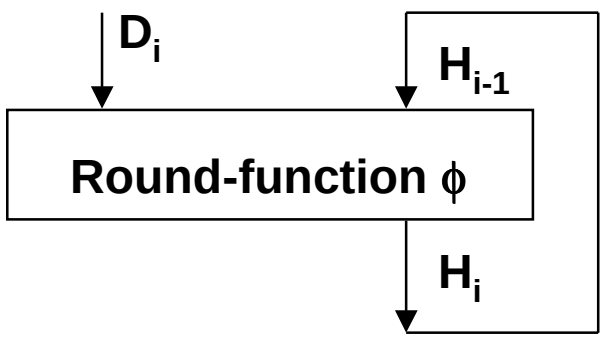
- **it is computationally infeasible to find any two distinct inputs which map to the same output.**

ISO/IEC 10118-3 IT ST Hash functions
Part 3: Dedicated hash-functions

Step 1 - padding. The length of data string should be a multiple of L_X.
Step 2 - splitting.



Step 3 - iteration



$H_0 = IV$

$H_i = \phi(D_i, H_{i-1})$

Step 4 - truncation. The hash-code H is derived by taking the leftmost L_H bits of the final L_Y-bit output string H_q.

Annex A 10118-1
(Normative)
Padding Methods

The calculation of a hash-code, as specified in other parts of ISO/IEC 10118, may require the selection of a padding method. The padding method will always output a padded data string whose length (in bits) is a multiple of L1. Three methods are presented in this annex.

NOTE - Hash-functions using Padding Method 1 may be subject to trivial forgery attacks, where an adversary can add or delete a number of trailing '0' bits of the data string without changing the hash-code. This means that Padding Method 1 shall only be used in environments where the length of the data string D is known to the parties beforehand, or where data strings with a different number of trailing '0' bits have the same semantics. Theoretical results (see, for example, [1]) also indicate that, in environments not satisfying the above condition, Padding Method 3 may offer certain advantages over Padding Method 2.

The padding bits (if any) need not be stored or transmitted with the data. The verifier shall know whether or not the padding bits have been stored or transmitted, and which padding method is in use.

A.1 Method 1

The data for which the hash-code is to be calculated are appended with as few (possibly no) '0' bits as are necessary to obtain the required length.

A.2 Method 2

The data for which the hash-code is to be calculated are appended with a single '1' bit. The resulting data are then appended with as few (possibly none) '0' bits as are necessary to obtain the required length.

Note – Method 2 always requires the addition of at least one padding bit**.**

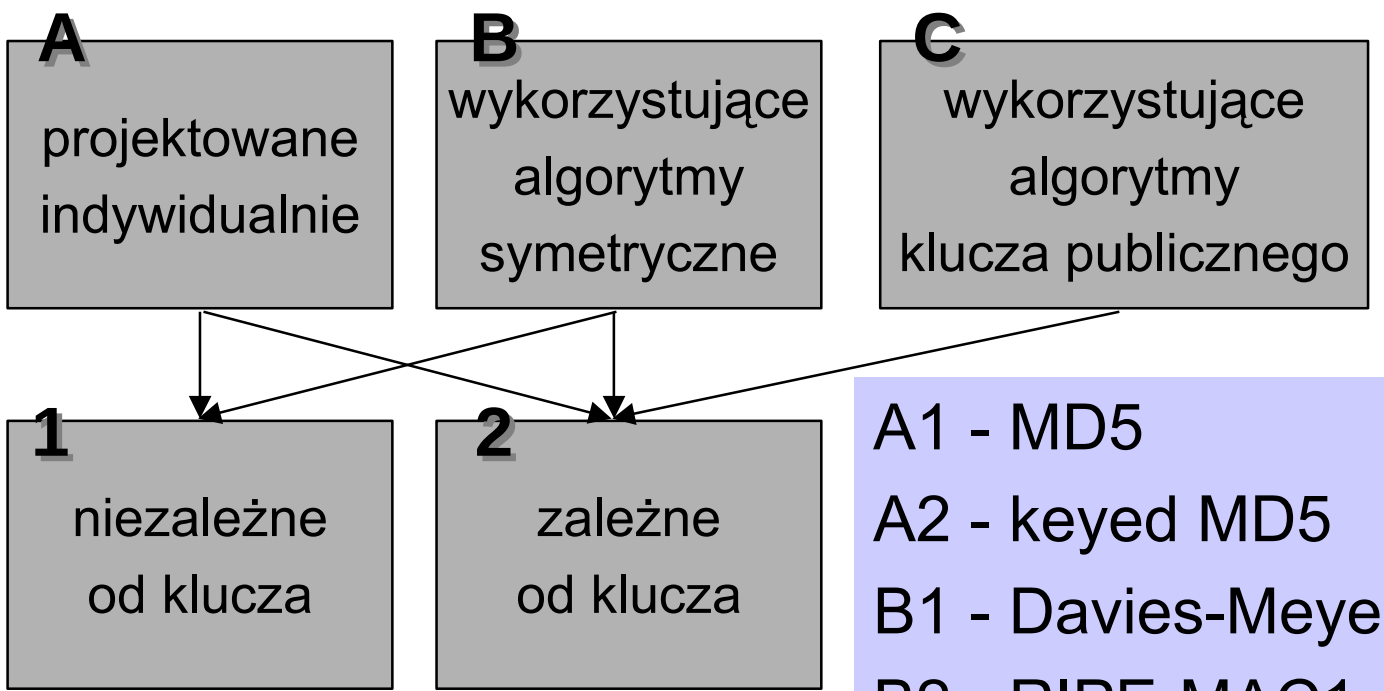
A.3 Method 3

This padding method requires the selection of a parameter r (where $r \leq L$), e.g. r = 64, and a method for encoding the bit length of the data D, i.e. LD, as a bit string of length r. The choice for r will limit the length of D, in that $LD < 2r$.

The data D for which the hash-code is to be calculated is padded using the following procedure.

1. D is concatenated with a single '1' bit.
2. The result of the previous step is concatenated with between zero and L1-1 '0' bits, such that the length of the resultant string is congruent to L1- r modulo L1. The result will be a bit string whose length will be r bits short of an integer multiple of L1 bits (in the case r = L1, the result will be a bit string whose length is an exact multiple of L1 bits).
3. Append an r-bit encoding of LD, using the selected encoding method, yielding the padded version of D.

Typy funkcji skrótu



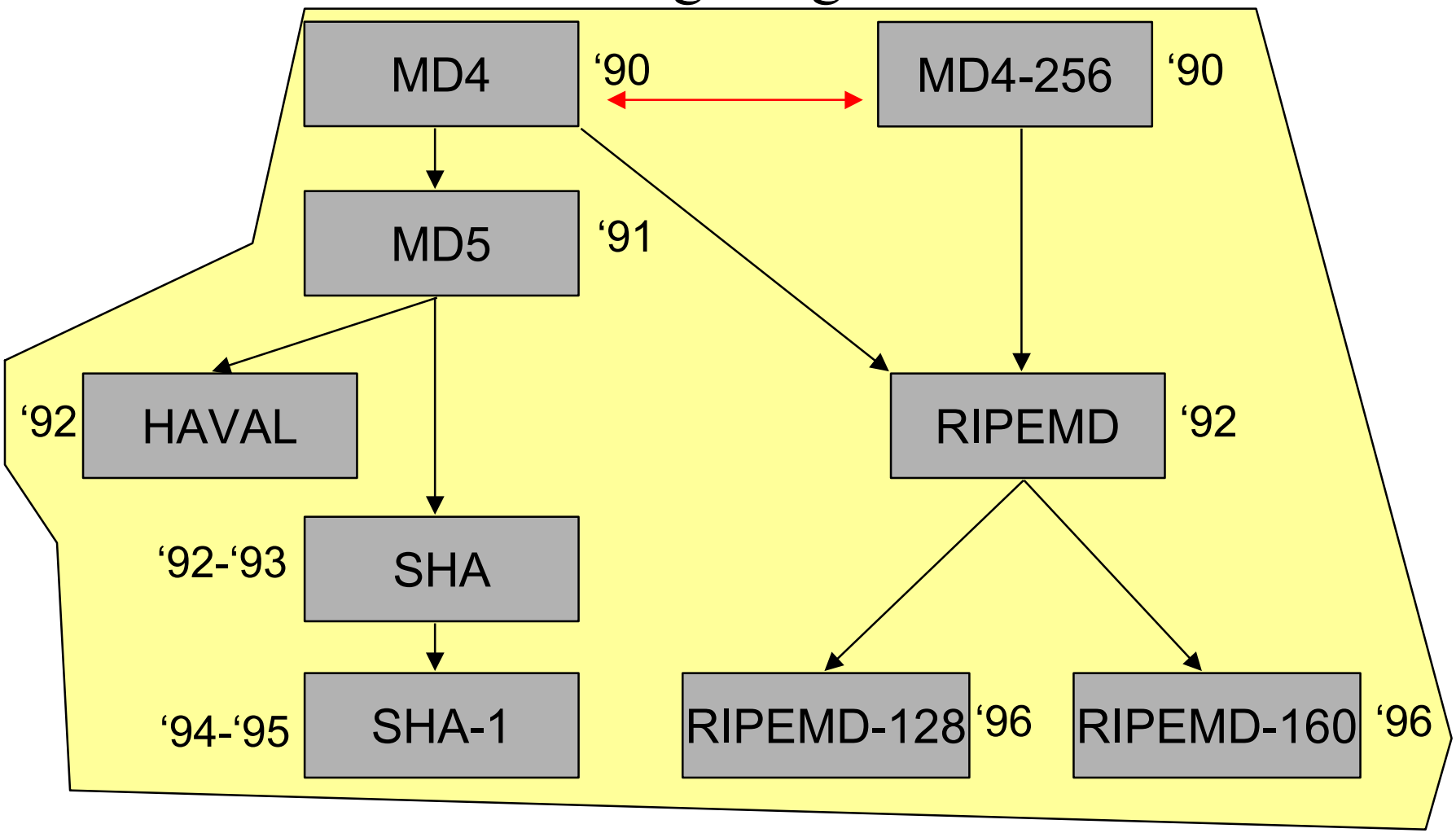
A1 - MD5
A2 - keyed MD5
B1 - Davies-Meyer using DES
B2 - RIPE-MAC1
C - RSA

A, B, C - budowa

1, 2 - funkcjonowanie

Rodzina funkcji skrótu MD4

MD- Message Digest



ISO/IEC 9797 IT ST Data Integrity Mechanism

MAC - Message Authentication Code

MAC calculation

1. Padding
2. Blocking $D_1, D_2, \ldots, D_q$ - the n-bit blocks of the data
3. Iteration

$I_1 \leftarrow D_1$

$I_i = A(I_{i-1},K)$ $i=2,3, \ldots ,q$

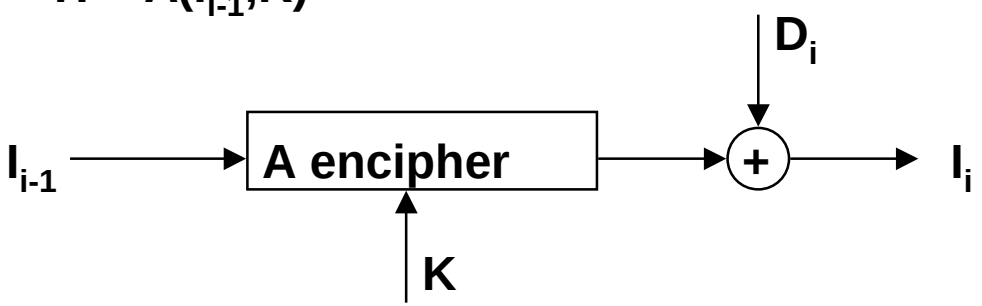
A - cryptographic algorithm using key K (may be DEA)

K - should be randomly or pseudorandomly generated

4. Truncation.

The MAC - the m leftmost bits of the final n-bit block, m<n

$H = A(I_{i-1},K)$



5. Optional processes

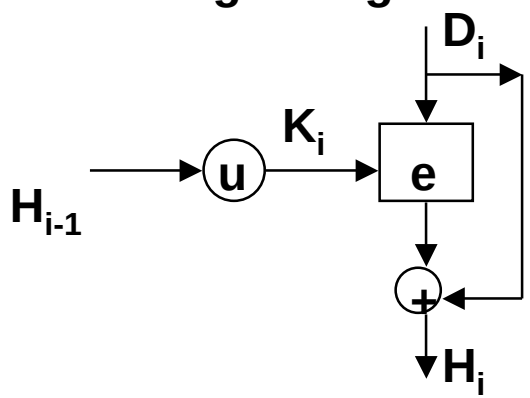
ISO/IEC 10118-2 IT ST Hash functions
Part 2: Hash-functions using an n-bit block cipher algorithm

Step 1 - splitting. $D_1, D_2, \ldots, D_q$ **- the n-bit blocks of the data**

Step 2 - padding.

Step 3 - iteration.

a. A single length hash-code

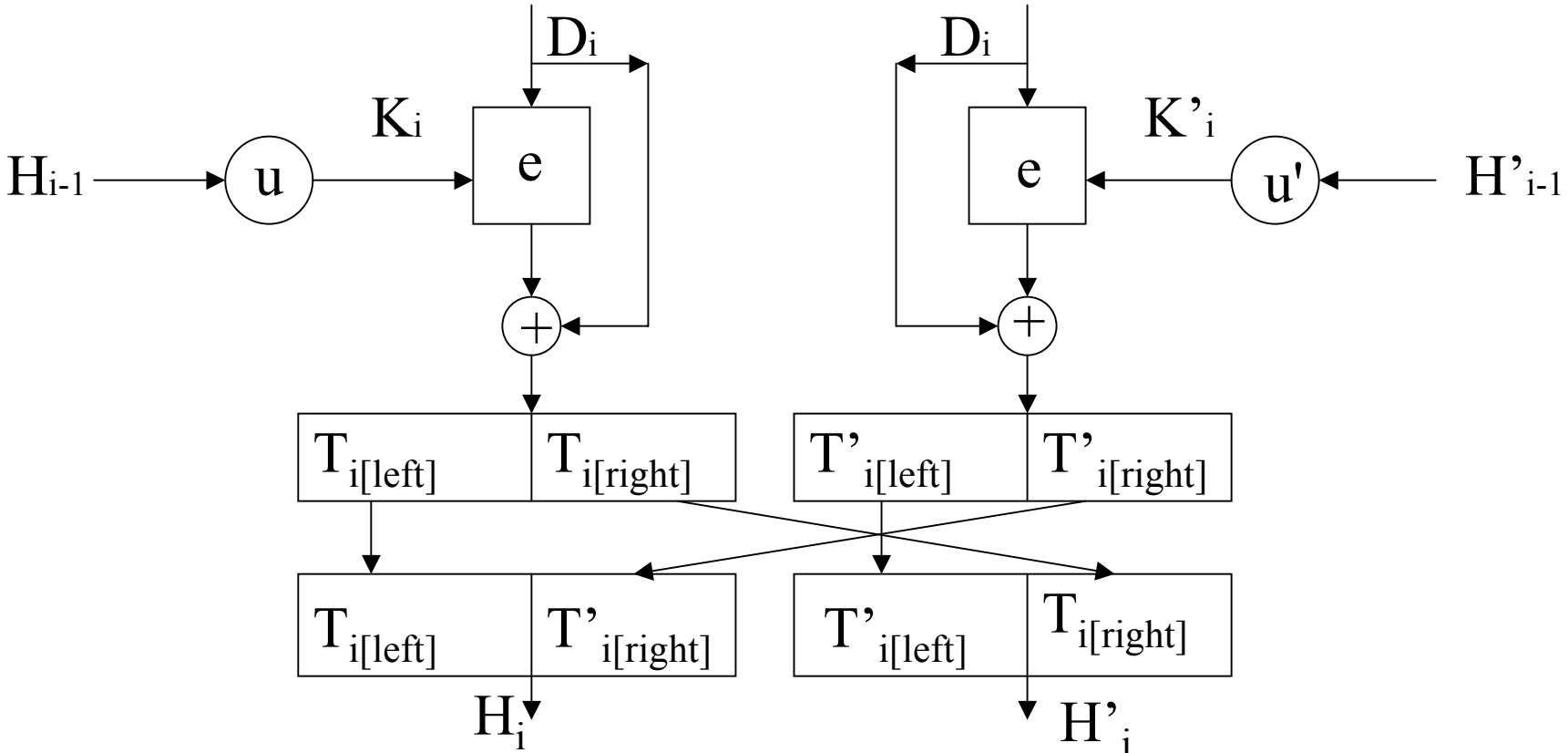


H_0 - **a block equal to IV**

$K_i = u(H_{i-1})$

$H_i = eK_i(D_i)$

b. A double length hash-code



H_0, H'_0 - two blocks equal to IV and IV' respectively

$K_i = u(H_{i-1})$ $\quad K'_i = u'(H'_{i-1})$,

$T_i = eK_i(D_i)$ xor D_i $\quad T'_i = eK'_i(D_i)$ xor D_i

$H_i = T_{i[left]}||T'_{i[right]}$ $\quad H'_i = T'_{i[left]}||T_{i[right]}$

} for i = 1,2,...,q

$T_{[left]}$ - n/2 leftmost bits of T when n - even, (n+1)/2 - n - odd

$T_{[right]}$ - n/2 righmost bits of T when n - even, (n+1)/2 - n - odd

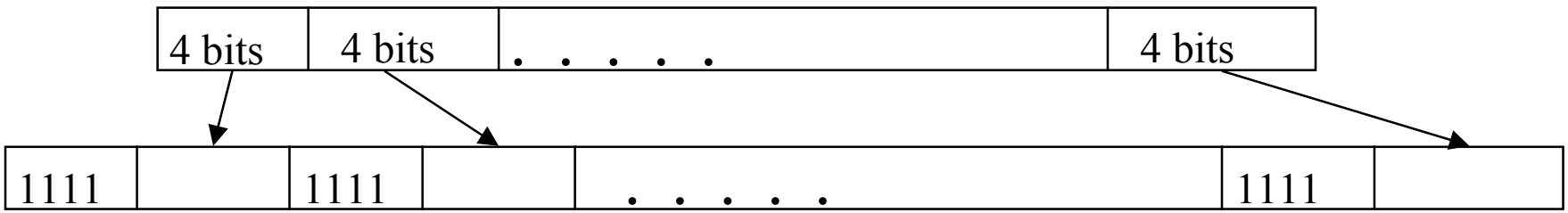
Step 4 - trucation

ISO/IEC 10118-4 IT ST Hash functions
Part 4: Hash functions using modular arithmetic

Step 1 - splitting. **Step 2 - padding.**

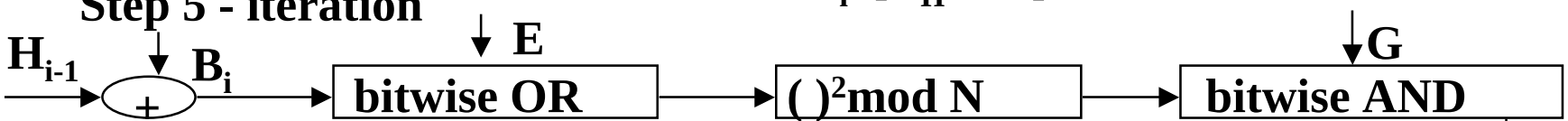
Step 3 - additional substring - contains the length of the original data string D headed by binary zeroes

Step 4 - expansion D_i [L_H/2 bits]



B_i [L_H bits]

Step 5 - iteration



E = 1111000....0 - a constant bit string of length L_H

G = 1111.....1 - a constant bit string of 1-s of length L_H

N - modulus, either a prime number or a product of two primes. Length: $L_H+1 <= L_N <= L_H+16$

H_0=IV $H_i=((((H_{i-1} \text{xor} B_i) \text{or} E)^2 \text{mod } N) \text{and } G)$

Step 6 - the hash-code H is the final block H_{m+1}

a. Dedicated Hash-Function 1 - RIPEMD (Race Integrity Primitives Evaluation Message Digest)

Initializing Value: Y_0=67452301 Y_1=EFCDAB89 Y_2=98BADCFE Y_3=10325476

Each 512-bit data blck D_i is treated as a sequence of 16 words, Z_0,Z_1, ... ,Z_{15}.

The round-function ϕ operates as follows.

1. $\phi(Z,Y)$
2. $X_i:=Y_i$ $X'_i:=Y_i$ i=0,1,2,3
3. For i:=0 to 47
   a. $W:=S^{ti}(X_0+g_i(X_1,X_2,X_3)+Z_{ai}+C_i)$;
   b. $X_0:=X_3$; $X_3:=X_2$; $X_2:=X_1$; $X_1:=W$;
   c. $W:=S^{ti}(X'_0+g_i(X'_1,X'_2,X'_3)+Z_{ai}+C'_i)$;
   d. $X'_0:=X'_3$; $X'_3:=X'_2$; $X'_2:=X'_1$; $X'_1:=W$;

where:

$S^n()$ - circular left shift by n bit positions,

$g_i(X_0,X_1,X_2) = (X_0$ and $X_1)$ or (not X_0 and $X_2)$, (0<= i <= 15)

$g_i(X_0,X_1,X_2) = (X_0$ and $X_1)$ or (X_0 and X_2) or (X_1 and X_2), (16<= i <= 31)

$g_i(X_0,X_1,X_2) = X_0$ xor X_1 xor X_2, (32<= i <= 47)

C_i, C'_i, t_i, a_i i=0,1 ... ,47 - constatnts defined in Standard.

4. Let

$W:=Y_0$ $Y_0:=Y_1+X_2+X'_3$ $Y_1:=Y_2+X_3+X'_0$ $Y_2:=Y_3+X_0+X'_1$ $Y_3:=W+X_1+X'_2$

5. The 128-bit string Y_0, Y_1, Y_2, Y_3 then represents the output of the round-function ϕ.

b. Dedicated Hash-Function 1 - Secure Hash Algorithm SHA-1 (US NIST)

Initializing Value:

Y_0=67452301 Y_1=EFCDAB89 Y_2=98BADCFE Y_3=10325476 Y_4=C3D2E1F0

Each 512-bit data blck D_i is treated as a sequence of 16 words, $Z_0, Z_1, \ldots, Z_{15}$.

The round-function ϕ operates as follows.

1. $\phi(Z,Y)$
2. For i=16 to 79 let $Z_i := S^1(Z_{i-3} \text{ xor } Z_{i-8} \text{ xor } Z_{i-14} \text{ xor } Z_{i-16})$
3. $X_i := Y_i$ i=0,1,2,3,4

4. For i=0 to 79 do the following two steps
   a. $W := S^5(X_0) + f_i(X_1,X_2,X_3) + X_4 + Z_i + C_i$;
   b. $X_4 := X_3$; $X_3 := X_2$; $X_2 := S^{30}(X_1)$; $X_1 := X_0$; $X_0 := W$;

where:

$f_i(X_0,X_1,X_2) = (X_0 \text{ and } X_1) \text{ or } (\text{not } X_0 \text{ and } X_2)$, $(0 <= i <= 19)$

$f_i(X_0,X_1,X_2) = X_0 \text{ xor } X_1 \text{ xor } X_2$, $(20 <= i <= 39)$

$f_i(X_0,X_1,X_2) = (X_0 \text{ and } X_1) \text{ or } (X_0 \text{ and } X_2) \text{ or } (X_1 \text{ and } X_2)$, $(40 <= i <= 59)$

$f_i(X_0,X_1,X_2) = X_0 \text{ xor } X_1 \text{ xor } X_2$, $(60 <= i <= 79)$

C_i i=0,1 ... ,79 - constatnts defined in Standard.

5. Let $Y_i := Y_i + X_i$ i=0,1, ...4.

6. The 160-bit string Y_0, Y_1, Y_2, Y_3, Y_4 then represents the output of the round-function ϕ.

Paradoks dnia urodzin

Ile średnio osób musi być w pomieszczeniu

aby znalazła się druga osoba obchodząca urodziny tego samego dnia co ja?

183

~365/2

aby znalazły się dwie osoby obchodzące urodziny tego samego dnia?

23

Birthday Atack

Birthday Attack

Wyobraźmy sobie tablicę, na której napisano n różnych dni roku. Każdy student wchodzi skreśla dzień swoich urodzin.

Liczymy prawdopodobieństwo, że r. student nie zastanie swojego dnia przekreślonego:
dla 1. - prawdopodobieństwo 1=n/n
dla 2. - (n-1)/n
dla r. - $p=n(n-1)(n-2)...(n-r+1) / n^r = n! / [(n-r)! n^r]$

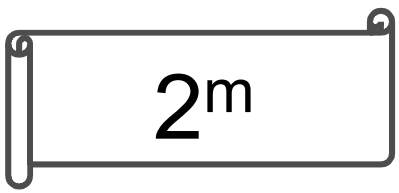
Dla n=365, r=23, p=0,493 - skreślenie już jest dla 23 studentów i wynosi q=1-p

Ataki na funkcję skrótu

Rozważamy funkcję skrótu H dającą skrót o dł. m

(niezależne od algorytmu)

dla danego M szukamy M',
dla którego H(M)=H(M')

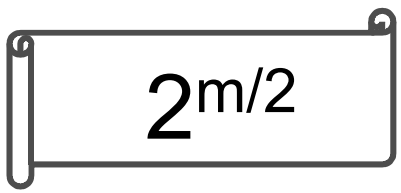


dla m=64 ~1,845e+19

58.5 mln lat

szukamy dowolnych M i M',
dla których H(M)=H(M')

birthday/square root attack



dla m/2=32 ~4,295e+9

5 dni

przy zał. 10.000 prób na sekundę

Przykład wykorzystania Birthday Attack [Yuval]

Zał. wykorzystujemy podpis cyfrowy oparty na funkcji skrótu

1. Ania przygotowuje dwie wersje kontraktu, który ma być podpisany z Basią - jedną fałszywą i drugą prawdziwą.
2. Ania dokonuje niewidocznych okiem zmian edytorskich w dokumentach - generuje w ten sposób $2^{m/2}$ wersji każdego z kontraktów.
3. Ania dokonuje skrótów dokumentów i szuka wspólnej pary.
4. Basia podpisuje „prawdziwy” kontrakt. „Fałszywy” kontrakt również staje się obowiązujący.

Rzut monetą

1. Asia losuje liczbę **x** i zlicza **h=H(x)**
2. Asia wysyła **h** do Bartka
3. Bartek zgaduje czy **x** jest parzyste, czy też nie i wysyła swoją odpowiedź do Asi
4. Jeśli Bartek zgadł, rezultat rzutu to orzeł, jeśli nie - reszka. Asia obwieszcza wynik i wysyła **x** do Bartka
5. Bartek sprawdza, czy **h=H(x)**

Szybkość implementacji