10 VYBRANÉ TYPY HAŠOVACÍCH FUNKCIÍ

Hašovacie funkcie predstavujú významný prostriedok v oblasti autentizácie používateľov a autorizácie dát, resp. v oblasti digitálnych podpisov. Ich vývoj je veľmi podobný vývoju v oblasti symetrických blokových šifier, ktorý ovplyvnilo najmä zvýšenie účinnosti útokov metódou totálnych skúšok a vývoj metód kryptoanalýzy. To nakoniec viedlo k prelomeniu algoritmu DES a prehodnoteniu bezpečnej dĺžky kľúča v symetrických blokových šifrách.

Analogicky aj v oblasti hašovacích funkcií zvýšenie výpočtovej výkonnosti a pokroky v ich kryptoanalýze priniesli zníženie popularity hašovacích funkcií MD4 a MD5, resp. zväčšenie dĺžky hašovacieho kódu.

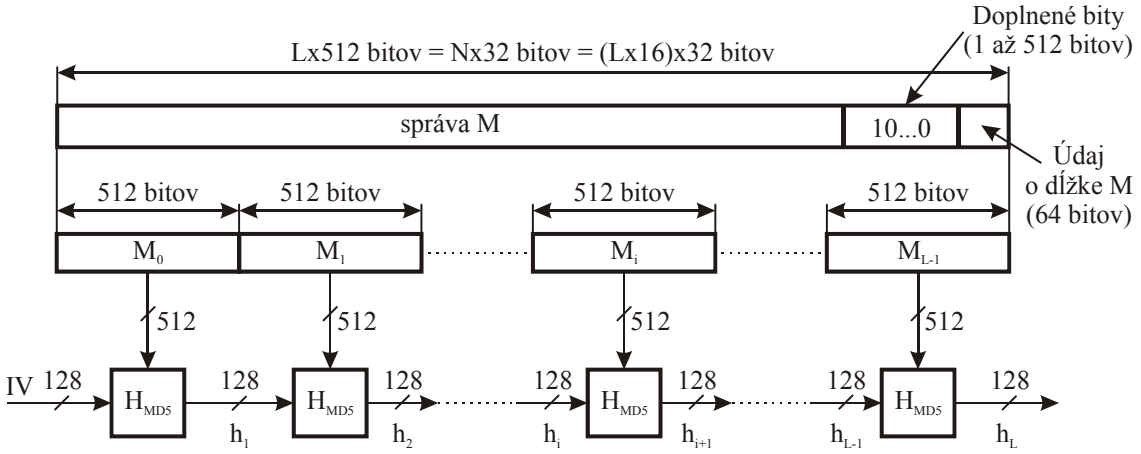
Vývoj symetrických blokových šifier v poslednom období priniesol tiež zmeny v oblasti ich štruktúry. Nové blokové šifry už nepoužívajú štruktúru Feistelovej šifry, ktorá je založená na aplikácii substitúcií a permutácií. Podobne aj nové generácie hašovacích funkcií využívajú nové prvky v ich štruktúre, pričom sa zväčšuje dĺžka hašovacieho kódu.

V ďalšej časti budú analyzované vlastnosti troch najznámejších hašovacích funkcií MD5, SHA–1 a RIPEMD–160 ako aj internetový štandard MAC, ktorý sa označuje ako HMAC a v svojej štruktúre používa hašovaciu funkciu.

10.1 Algoritmus MD5

Algoritmus hašovacej funkcie MD5 spracúva vstupné údaje po blokoch s dĺžkou 512 bitov a generuje hašovací kód s dĺžkou 128 bitov.

Bloková schéma algoritmu MD5 je uvedená na *Obr. 10.1*.



Obr. 10.1 Bloková schéma algoritmu MD5

Algoritmus MD5 pozostáva z týchto krokov:

1. krok: zahrňuje doplnenie správy *M* skupinou bitov tak, aby dĺžka doplnenej správy bola kongruentná 448 modulo 512. Toto doplnenie zabezpečí, že dĺžka správy je vždy o 64 bitov kratšia než celočíselný násobok 512 bitov. Napr. ak dĺžka správy je 448 bitov, doplnenie sa realizuje pridaním 512 bitov, t.j. dĺžka doplnenej správy je 960 bitov, čo je o 64 bitov menej ako je celočíselný násobok 512, t.j. 1024 bitov.

Doplnenie správy *M* sa realizuje vždy, pričom počet doplnených bitov sa môže pohybovať v rozmedzí 1 až 512 bitov a táto skupina bitov obsahuje prvý bit 1, za ktorým nasleduje potrebný počet bitov 0.

2. krok: zabezpečuje doplnenie údaja o dĺžke správy, ktorý sa pridá k výsledku z predošlého kroku. Údaj má dĺžku 64 bitov, t.j. reprezentuje dĺžku správy *M* do 2^{64} bitov. Údaj zabezpečuje len nižších 64 bitov, t.j. udáva dĺžku správy *M* modulo 2^{64}.

Výsledkom prvých dvoch krokov je, že dĺžka doplnenej správy je celočíselným násobkom 512 bitov. Expandovaná správa je reprezentovaná postupnosťou 512 bitových blokov M_0, M_1, M_2 ... M_{L-1} *(Obr. 10.1)*, teda platí, že dĺžka expandovanej správy je $L \times 512$ bitov.

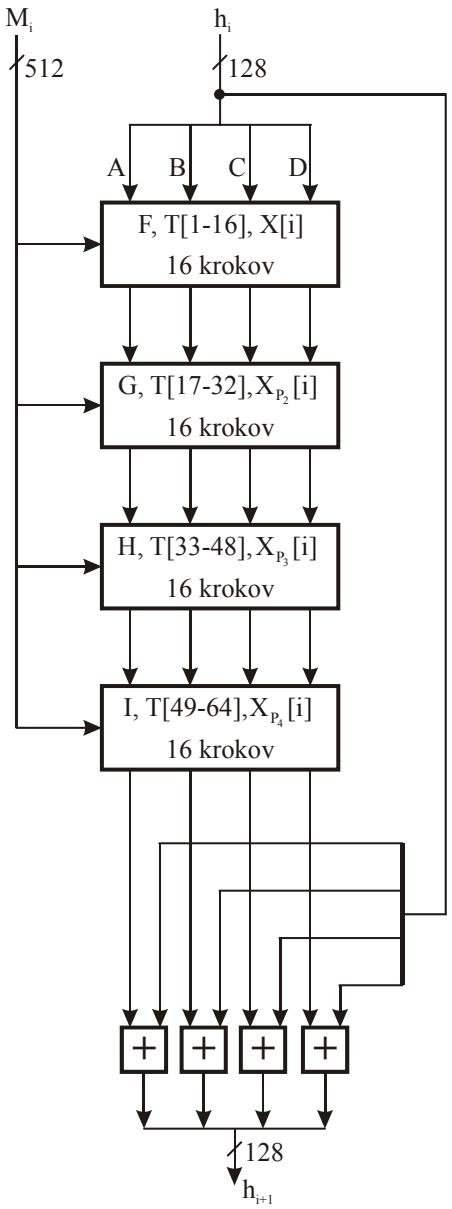
Ak vyjadríme jeden 512 bitový blok v tvare 16×32 bitov, každý blok obsahuje 16 slov o dĺžke 32 bitov. Expandovanú správu možno vyjadriť tiež v tvare $N \times 32$ bitov, kde $N = L \times 16$.

3.krok: v tomto kroku sa realizuje inicializácia registra hašovacieho kódu. Register hašovacieho kódu zapamätáva priebežné výsledky výpočtu 128–bitového hašovacieho kódu $(h_1, h_2, \ldots, h_{L-1})$, resp. výsledný hašovací kód h_L=*h*. Register hašovacieho kódu pozostáva zo 4 registrov s dĺžkou 32 bitov, ktoré sa označujú symbolmi A, B, C, D. Inicializácia pozostáva z naplnenia tých registrov pevnými údajmi, ktoré možno vyjadriť v hexadecimálnom tvare takto:

A=67452301
B=EFCDAB89
C=98BADCFE
D=10325476

Incicializácia registrov A,B,C,D zároveň zodpovedá inicializačnému vektoru IV.

4. krok: v tomto kroku prebieha spracovanie 512 bitových blokov správy *M* v 4 rundách, z ktorých každá pozostáva zo 16 krokov. Štruktúra spracovania 512 bitového bloku je na *Obr. 10.2*. Každá runda má rovnakú štruktúru, ale využíva rôzne vstupné údaje a rôzne logické funkcie, ktoré sú označené symbolmi F, G, H a I. Vstupnými údajmi každej rundy sú aktuálny 512 bitový blok M_i a aktuálny 128 bitový údaj 32 bitových registrov A, B, C, D. Okrem tohto každá runda využíva 16 konštánt z tabuľky *T*, ktorá obsahuje 64 položiek $T[1, 2, \ldots, 64]$. Konštanty s veľkosťou 32 bitov sú odvodené od funkcie absolútnej hodnoty $abs(\sin(i))$, kde *i* je údaj v radiánoch.



Obr. 10.2 Štruktúra spracovania bloku M_i v MD5

5. krok: záverečný krok po spracovaní všetkých *L* 512 bitových blokov poskytuje 128 bitový hašovací kód $h_L=h$.

10.2 Kompresná funkcia MD5

Jadrom každej rundy je kompresná funkcia, ktorá transformuje 512 bitový blok M_i na 128 bitový hašovací kód h_{i+1}. Každá runda pozostáva z postupnosti 16 krokov, pričom každý krok možno zapísať v tvare:

$$a \leftarrow d$$
$$b \leftarrow b + \left(\left(a + g\left(b,c,d\right) + X\left[k\right] + T\left[i\right] \right) <<< RLs \right)$$
$$c \leftarrow b$$
$$d \leftarrow c$$

kde

a, b, c, d – sú aktuálne obsahy registrov A, B, C, D, ktoré sa priebežne aktualizujú počas každého kroku

g – jedna z logických funkcií F, G, H, I

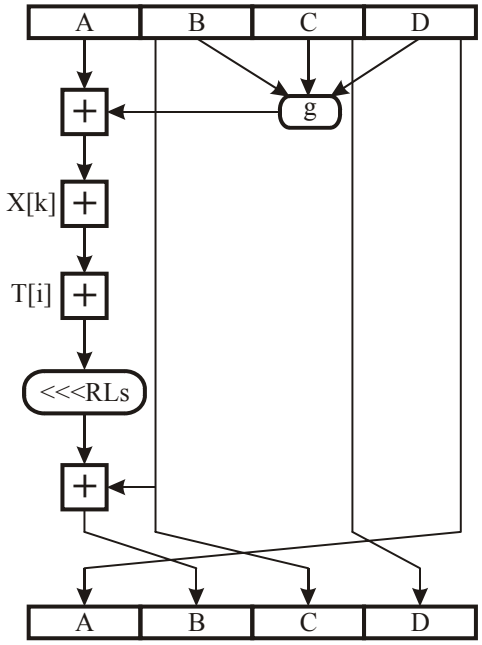
$X[k]$ – 32–bitové slovo z bloku M_i

$T[i]$ – 32–bitová konštanta z tabuľky T

$+$ – sčítanie modulo 2^{32}

$<<<RLs$ – rotácia vľavo o s–bitov, kde s je príslušný krok

Vývojový diagram jedného kroku možno graficky znázorniť schémou na *Obr. 10.3*. Každý krok zahrňuje transformáciu, resp. presun obsahu registrov A, B, C, D tak, že výstupom sú transformované hodnoty týchto registrov.



Obr. 10.3 Vývojový diagram jedného kroku rundy MD5

Kľúčovým prvkom každého kroku je výpočet nového obsahu registra B, t.j. novej hodnoty b, ktorá závisí od hodnôt a, b, c, d, logickej funkcie g, bloku M_i a konštánt $T[i]$.

Logická funkcia g reprezentuje jednu zo štyroch logických funkcií F, G, H, I, pričom logická funkcia F sa uplatní v prvej runde, G v druhej runde, H v tretej runde a I v štvrtej runde.

Každá z uvedených logických funkcií spracúva 32–bitové vstupné slovo a generuje nové 32–bitové slovo, pričom logické operácie sa vykonávajú po bitoch.

Opis logických funkcií F, G, H, I je uvedený v *Tab. 10.1*.

Tab. 10.1 Logické funkcie MD5

Runda	Typ logickej funkcie g	$g(b,c,d)$

1	F(b,c,d)	$(b \wedge c) \vee (\overline{b} \wedge d)$
2	G(b,c,d)	$(b \wedge d) \vee (c \wedge \overline{d})$
3	H(b,c,d)	$b \oplus c \oplus d$
4	I(b,c,d)	$c \oplus (b \vee \overline{d})$

Logické operátory AND, OR, NOT a XOR sú reprezentované symbolmi $\wedge, \vee, \overline{\ }, \oplus$. Pravdivostnú tabuľku logických funkcií F, G, H a I pre jeden bit možno vyjadriť v tvare:

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

V každej runde sa využíva tiež pole 32 bitových slov $X[0,1,\ldots,15]$, ktoré reprezentujú hodnoty aktuálneho 512 bitového bloku M_i. Poradie, v ktorom sa tieto slová využívajú sa však v každej runde mení. V prvej runde sa poradie slov $X[i]$ nemení a zodpovedá originálnemu poradiu v bloku M_i. V ostatných rundách je poradie slov určené permutáciami, ktoré možno vyjadriť v tvare:

$$\left.\begin{aligned} p_2(i) &= (1+5i) \bmod 16 \\ p_3(i) &= (5+3i) \bmod 16 \\ p_4(i) &= 7i \bmod 16 \end{aligned}\right\} \quad pre \quad i = 0,1,2,\ldots,15$$

Koštanty $T[i]$ sú určené tabuľkou a v každej runde sa využíva 16 konštánt z celkového počtu 64.

Tab. 10.2 Tabuľka konštánt $T[i]$

$T[1]$ = D76AA478	$T[17]$ = F61E2562	$T[33]$ = FFFA3942	$T[49]$ = F4292244
$T[2]$ = E8C7B756	$T[18]$ = C040B340	$T[34]$ = 8771F681	$T[50]$ = 432AFF97
$T[3]$ = 242070DB	$T[19]$ = 265E5A51	$T[35]$ = 699D6122	$T[51]$ = AB9423A7
$T[4]$ = C1BDCEEE	$T[20]$ = E9B6C7AA	$T[36]$ = FDE5380C	$T[52]$ = FC93A039
$T[5]$ = F57C0FAF	$T[21]$ = D62F105D	$T[37]$ = A4BEEA44	$T[53]$ = 655B59C3
$T[6]$ = 4787C62A	$T[22]$ = 02441453	$T[38]$ = 4BDECFA9	$T[54]$ = 8F0CCC92
$T[7]$ = A8304613	$T[23]$ = D8A1E681	$T[39]$ = F6BB4B60	$T[55]$ = FFEFF47D
$T[8]$ = FD469501	$T[24]$ = E7D3FBC8	$T[40]$ = BEBFBC70	$T[56]$ = 85845DD1
$T[9]$ = 698098D8	$T[25]$ = 21E1CDE6	$T[41]$ = 289B7EC6	$T[57]$ = 6FA87E4F
$T[10]$ = 8B44F7AF	$T[26]$ = C33707D6	$T[42]$ = EAA127FA	$T[58]$ = FE2CE6E0
$T[11]$ = FFFF5BB1	$T[27]$ = F4D50D87	$T[43]$ = D4EF3085	$T[59]$ = A3014314
$T[12]$ = 895CD7BE	$T[28]$ = 455A14ED	$T[44]$ = 04881D05	$T[60]$ = 4E0811A1
$T[13]$ = 6B901122	$T[29]$ = A9E3E905	$T[45]$ = D9D4D039	$T[61]$ = F7537E82
$T[14]$ = FD987193	$T[30]$ = FCEFA3F8	$T[46]$ = E6DB99E5	$T[62]$ = BD3AF235
$T[15]$ = A679438E	$T[31]$ = 676F02D9	$T[47]$ = 1FA27CF8	$T[63]$ = 2AD7D2BB
$T[16]$ = 49B40821	$T[32]$ = 8D2A4C8A	$T[48]$ = C4AC5665	$T[64]$ = EB86D391

Poslednou operáciou je rotácia vľavo o *s*–bitov, kde počet bitov je pre prvú rundu rovný *s*=1, pre druhú rundu *s*=2, atď.

10.3 Algoritmy skupiny hašovacích funkcií SHA

Algoritmus SHA (Secure Hash Algorithm) bol zavedený NIST a publikovaný v norme FIPS 180 v roku 1993. Revidovaná verzia bola vydaná pod označením FIPS–180–1 v roku 1995, pričom sa označuje ako SHA–1.

Algoritmus SHA–1 umožňuje spracovať originálnu správu s maximálnou dĺžkou menšou než 2^{64} bitov a predstavuje výstupný hašovací kód s dĺžkou 160 bitov. Vstupná správa je spracovaná po blokoch s veľkosťou 512 bitov.

Bloková schéma algoritmu SHA–1 je totožná so schémou na *Obr. 10.1*, pričom algoritmus spracovania správy *M* pozostáva z týchto krokov:

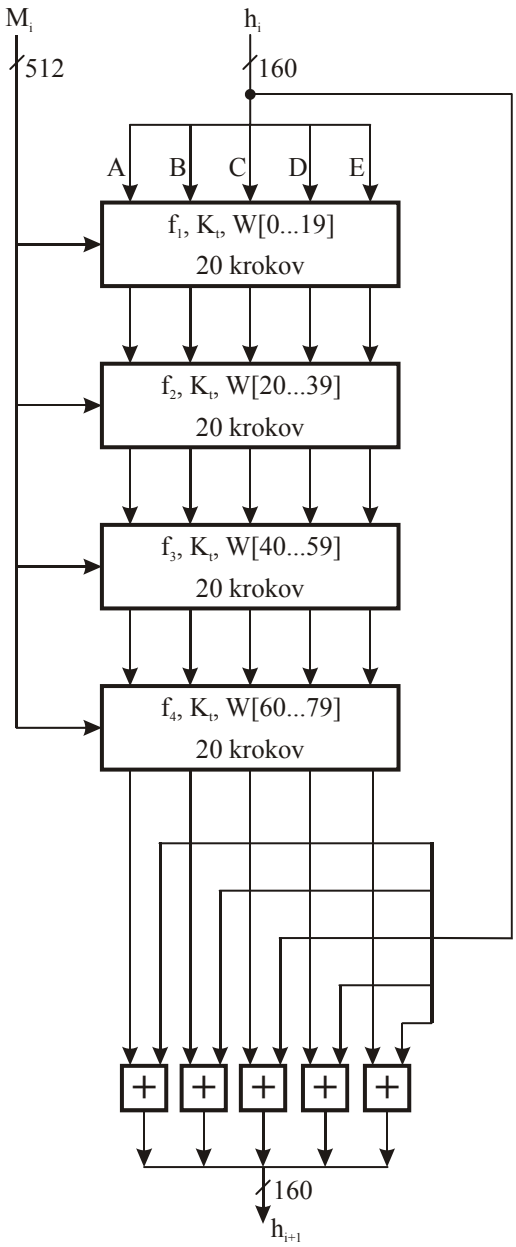
1.krok: doplnenie originálnej správy *M* skupinou bitov tak, aby dĺžka doplnenej správy bola kongruentná 448 modulo 512. Doplnenie sa realizuje vždy, pričom počet doplnených bitov sa mení v rozsahu od 1 do 512 a skupina doplnených bitov má tvar 1 0 0 ... 0.

2.krok: doplnenie bloku s dĺžkou 64 bitov, ktorý obsahuje údaj o dĺžke originálnej správy (pred doplnením), pričom ako prvý je uvedený najviac významový byte.

3.krok: inicializácia 160 bitového registra hašovacieho kódu, ktorý pozostáva z 5 registrov (A, B, C, D, E) s dĺžkou 32 bitov. Inicializácia má tvar v hexadecimálnom tvare:

A=67452301
B=EFCDAB89
C=98BADCFE
D=10325476
E=C3D2E1F0

4.krok: spracovanie expandovanej správy po 512 bitových blokoch resp. 16–bitových slovách. Štruktúra algoritmu SHA–1, ktorá je uvedená na *Obr. 10.4*, realizuje spracovanie jedného 512 bitového bloku a pozostáva zo štyroch rúnd, pričom každá runda má 20 krokov. Jednotlivé rundy majú rovnakú štruktúru, ale každá používa rôzne logické funkcie, ktoré sú označené symbolmi f_1, f_2, f_3 a f_4.



Obr. 10.4 Štruktúra spracovania bloku M_i v SHA–1

Okrem toho každá runda používa v jednotlivých krokoch konštantu K_t, pričom $0 \le t \le 79$, ktorej hodnota je daná tabuľkou:

Číslo kroku	Hexadecimálny tvar K_t	Celočíselná časť hodnoty
$0 \le t \le 19$	K_t=5A927999	$\left(2^{30} \times \sqrt{2}\right)$
$20 \le t \le 39$	K_t=6ED9EBA1	$\left(2^{30} \times \sqrt{3}\right)$
$40 \le t \le 59$	K_t=8F1BBCDC	$\left(2^{30} \times \sqrt{5}\right)$
$60 \le t \le 79$	K_t=CA62C1D6	$\left(2^{30} \times \sqrt{10}\right)$

Ako vyplýva z uvedenej tabuľky, algoritmus SHA–1 používa iba štyri rôzne hodnoty K_t, teda rôznu hodnotu pre každú rundu.

5.krok: Výsledok spracovania bloku M_i v štyroch rundách (80 krokov) poskytuje h_{i+1} s dĺžkou 160 bitov. Po spracovaní všetkých 512–bitových blokov správy M v L cykloch sa získa výsledný hašovací kód $h_L=h$.

10.4 Kompresná funkcia SHA–1

Kompresná funkcia realizuje transformáciu 512 bitového bloku M_i na 160 bitový hašovací kód h_{i+1}, ktorá sa vykonáva v 80 krokoch, resp. štyroch rundách po 20 krokoch.

Každú rundu možno zapísať v tvare *(Obr. 10.5)*.

$$\mathrm{A} \leftarrow \mathrm{E} + f(t,\mathrm{B},\mathrm{C},\mathrm{D}) + S^5(\mathrm{A}) + W_t + K_t$$
$$\mathrm{B} \leftarrow \mathrm{A}$$
$$\mathrm{C} \leftarrow S^{30}(\mathrm{B})$$
$$\mathrm{D} \leftarrow \mathrm{C}$$
$$\mathrm{E} \leftarrow \mathrm{D}$$

kde

A,B,C,D,E – päť 32–bitových registrov hašovacieho kódu

t – číslo kroku $0 \le t \le 79$

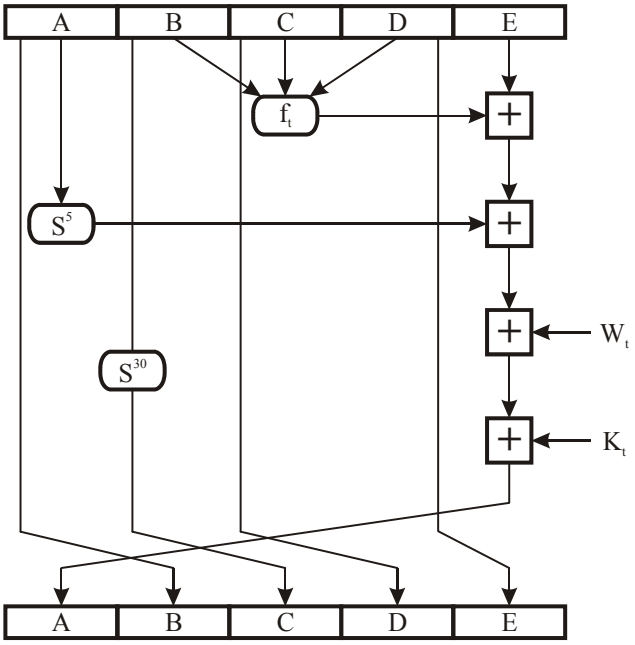
$f(t,\mathrm{B},\mathrm{C},\mathrm{D})$ – logická funkcia pre krok t

S^k – rotácia vľavo 32–bitového slova o k bitov

W_t – 32–bitové slovo z 512–bitového bloku M_i

K_t – 32–bitová konštanta

$+$ – sčítanie modulo 2^{32}



Obr. 10.5 Kompresná funkcia SHA–1

Logická funkcia $f(t,A,B,C,D)$ pracuje s troma 32–bitovými vstupmi a produkuje 32–bitové výstupné slovo, pričom logické operácie sa realizujú po bitoch podľa tabuľky *Tab. 10.3*.

Tab. 10.3 Logické funkcie SHA–1

Číslo kroku	Typ logickej funkcie	$f(t,A,B,C,D)$
$0 \le t \le 19$	$f_1(t,B,C,D)$	$(B \wedge C) \vee (\overline{B} \wedge D)$
$20 \le t \le 39$	$f_2(t,B,C,D)$	$B \oplus C \oplus D$
$40 \le t \le 59$	$f_3(t,B,C,D)$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$60 \le t \le 79$	$f_4(t,B,C,D)$	$B \oplus C \oplus D$

Logické operátory AND, OR, NOT a XOR sú reprezentované symbolmi $\wedge, \vee, \overline{\ }, \oplus$ a logické funkcie sú pre jednotlivé rundy označené symbolmi f_1, f_2, f_3 a f_4. Ako vyplýva z *Tab. 10.3*, funkcia $f_2=f_4$, teda kompresná funkcia SHA–1 používa iba tri rôzne logické funkcie. Pravdivostná tabuľka logických funkcií je daná *Tab. 10.4*.

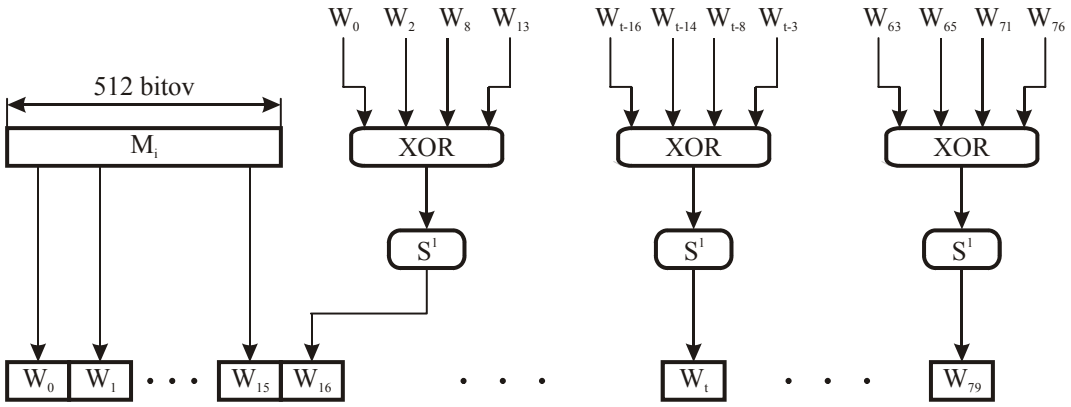
Tab. 10.4 Pravdivostná tabuľka logických funkcií

B	C	D	$f_{0–19}$	$f_{20–39}$	$f_{40–59}$	$f_{60–79}$
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	0	0	1	0
1	1	0	1	0	1	0
1	1	1	1	1	1	1

Kompresná funkcia SHA–1 využíva aj 32–bitové slovo W_t, ktoré je odvodené z 512–bitového bloku M_i. Generovanie tohto slova sa realizuje dvoma spôsobmi. Prvých 16 slov W_t predstavuje priamo 16 slov bloku M_i. Ostatné hodnoty W_t možno získať z rovnice

$$W_t = S^1\left(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}\right), \quad \text{pre } t = 16,17,\ldots,79 \tag{10.1}$$

Z uvedenej rovnice vyplýva, že zvyšných 64 hodnôt W_t sa získa rotáciou o jeden bit vľavo hodnoty, ktorá sa získa operáciou XOR štyroch hodnôt W_t. Graficky je postup znázornený na *Obr. 10.6*.



Obr. 10.6 Generovanie slov W_t

Z aktuálneho bloku M_i sa teda priamo získa 16 slov W_0 až W_{15} a rovnica (10.1) opisuje generovanie 64 slov W_{16} až W_{79}.

V roku 1981 sa inovoval štandard FIPS 180–1, ktorý dostal označenie FIPS–180–2, a ktorý zaviedol tri nové hašovacie algoritmy SHA–256, SHA–384 a SHA–512. Základné vlastnosti uvedených algoritmov vrátane algoritmu SHA–1 sú uvedené v *Tab. 10.5*.

Tab. 10.5 Základné vlastnosti hašovacích funkcií

	SHA–1	SHA–256	SHA–384	SHA–512
Dĺžka hašovacieho kódu	160	256	384	512
Dĺžka správy	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Veľkosť bloku	512	512	1024	1024
Veľkosť slova	32	32	64	84
Počet krokov algoritmu	80	80	80	80
Ekvivalentná bezpečnosť v bitoch	80	128	192	256

Najvýznamnejšie rozdiely v uvedených hašovacích algoritmoch spočívajú v dĺžke hašovacieho kódu, ktorý určuje bezpečnosť algoritmu proti metóde totálnych skúšok, pričom štruktúra uvedených algoritmov je takmer rovnaká.

10.5 Algoritmus RIPEMD–160

Hašovací algoritmus RIPEMD–160 bol navrhnutý v rámci európskeho projektu RIPE (Race Integrity Primitives Evaluation) skupinou výskumníkov, ktorí realizovali čiastočne úspešné útoky na hašovacie algoritmy MD4 a MD5. Skupina navrhla pôvodne verziu RIPEM so 128–bitovým hašovacím kódom, ktorá bola neskoršie upravená na verziu RIPEMD–160.

Algoritmus RIPEMD–160 umožňuje spracovať originálnu správu s maximálnou dĺžkou 2^{64} bitov a produkuje výstupný hašovací kód s dĺžkou 160 bitov, pričom vstupná správa je spracovaná po 512–bitových blokoch.

Algoritmus spracovania originálnej správy *M* pozostáva z týchto krokov:

1.krok: doplnenie originálnej správy *M* skupinou bitov tak, aby dĺžka doplnenej správy bola kongruentná 448 modulo 512. Doplnenie sa realizuje vždy, pričom počet doplnených bitov sa mení v rozsahu 1 až 512 a skupina doplnených bitov má tvar 1 0 0 ... 0.

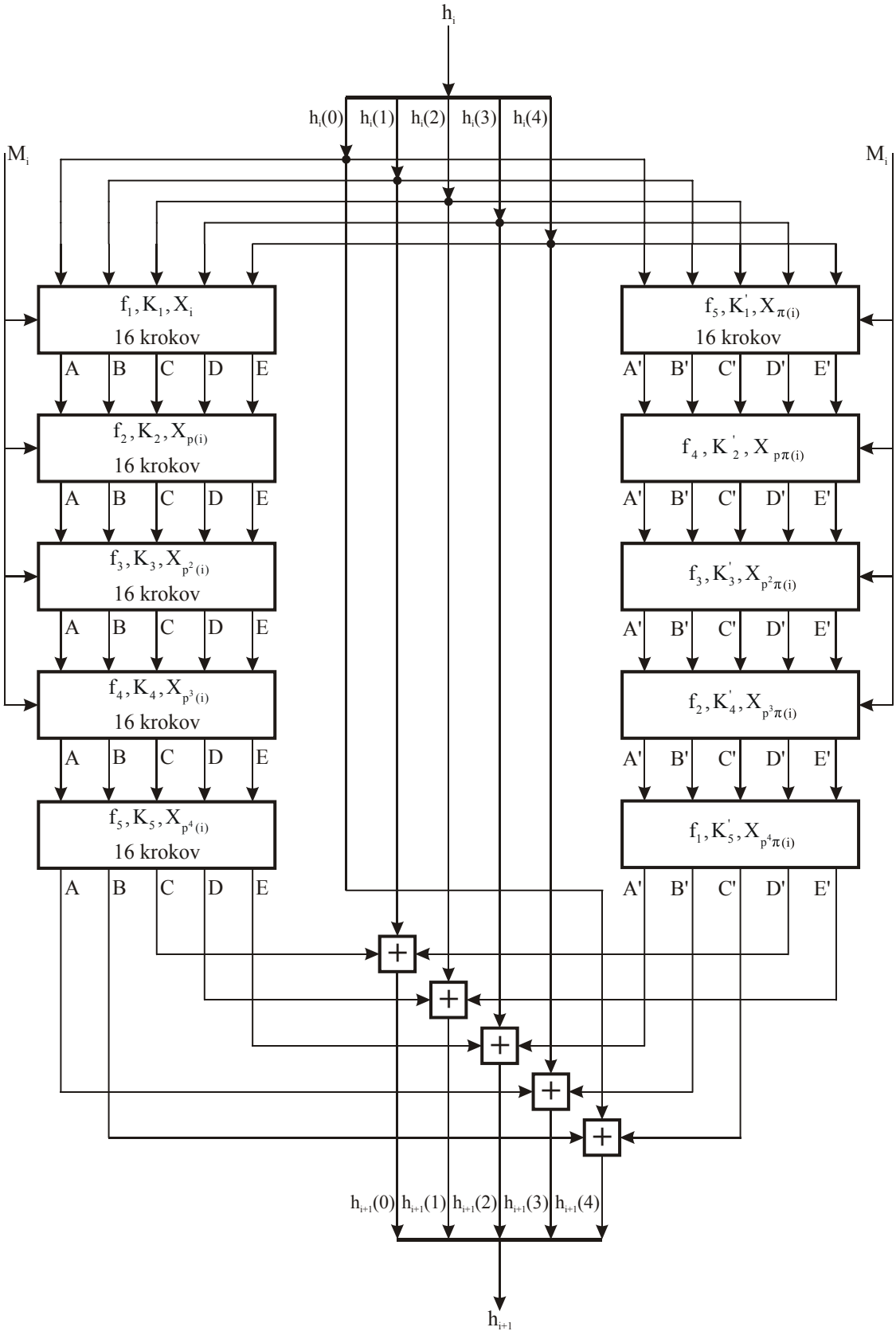
2.krok: doplnenie blokom s dĺžkou 64 bitov, ktorý obsahuje údaj o dĺžke originálnej správy, pričom ako prvý je uvedený najviac významový byte.

3.krok: inicializácia 160–bitového registra hašovacieho kódu, ktorý pozostáva z 5 registrov (A,B,C,D,E) s dĺžkou 32 bitov. Inicializácia má tvar

A=67452301
B=EFCDAB89
C=98BADCFE
D=10325476
E=C3D2E1F0

Uvedené hodnoty pre A, B, C, D sú zhodné s hodnotami, ktoré sú použité v MD5 a hodnota E je zhodná s hodnotou E v SHA–1.

4.krok: spracovanie expandovanej správy po 512–bitových blokoch, resp. po 16–bitových slovách. Štruktúra algoritmu RIPEMD–160, ktorá je uvedená na *Obr. 10.7*, používa 10 rúnd (každá má 16 krokov) rozdelených do dvoch paralelných vetiev po 5 rúnd. Všetky rundy majú rovnakú štruktúru ale každá používa inú logickú funkciu. Logické funkcie sú označené symbolmi f_1, f_2, f_3, f_4 a f_5, pričom poradie ich použitia v jednotlivých vetvách je opačné. Každá runda tiež používa inú konštantu. Prehľad použitých konštánt v jednotlivých rundách je znázornený v *Tab. 10.6*.



Obr. 10.7 Štruktúra spracovania bloku M_i v RIPEMD–160

Tab. 10.6 Konštanty RIPEMD–160

Číslo kroku	Ľavá vetva		Pravá vetva	
	Hexadecimálny tvar	Celočíselná časť hodnoty	Hexadecimálny tvar	Celočíselná časť hodnoty
$0 \le j \le 15$	$K_1=K(j)=$ 00000000	0	$K_1^{'} = K^{'}(j) =$ 5DA28BE6	$2^{30} \times \sqrt[3]{2}$
$16 \le j \le 31$	$K_2=K(j)=$ 5A827999	$2^{30} \times \sqrt{2}$	$K_2^{'} = K^{'}(j) =$ 5C4DD124	$2^{30} \times \sqrt[3]{3}$
$32 \le j \le 47$	$K_3=K(j)=$ 6ED9EBA1	$2^{30} \times \sqrt{3}$	$K_3^{'} = K^{'}(j) =$ 6D703EF3	$2^{30} \times \sqrt[3]{5}$
$48 \le j \le 63$	$K_4=K(j)=$ 8F1BBCDC	$2^{30} \times \sqrt{5}$	$K_4^{'} = K^{'}(j) =$ 7A6D76E9	$2^{30} \times \sqrt[3]{7}$
$64 \le j \le 79$	$K_5=K(j)=$ A953FD4E	$2^{30} \times \sqrt{7}$	$K_5^{'} = K^{'}(j) =$ 00000000	0

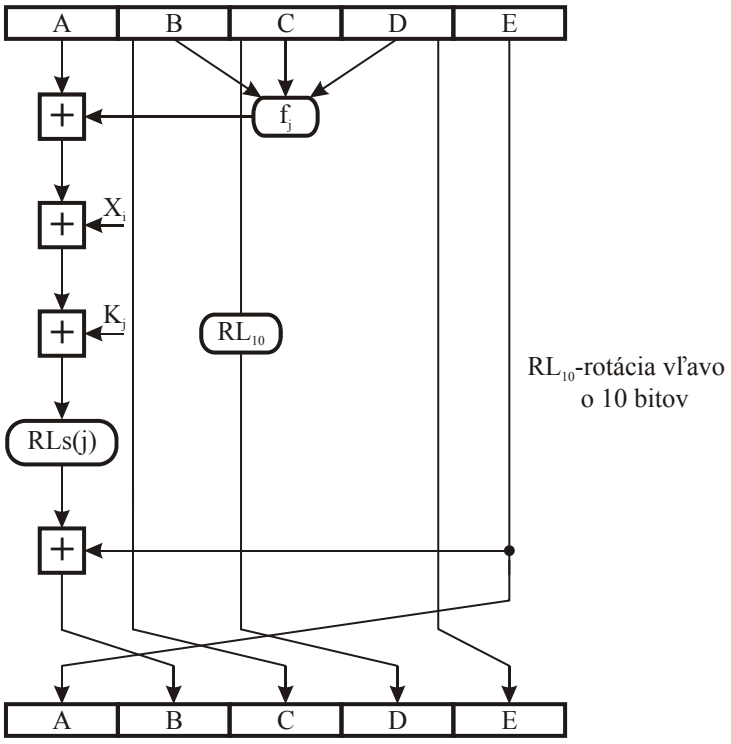
Výstup piatej rundy (80. krok) sa pričíta k vstupu prvej rundy (h_i), čím sa získa hašovací kód h_{i+1}. Sčítanie sa realizuje nezávisle pre každé 32–bitové slovo operáciou modulo 2^{32} a možno ho vyjadriť v tvare

$$h_{i+1}(0) = h_i(1) + C + D'$$
$$h_{i+1}(1) = h_i(2) + D + E'$$
$$h_{i+1}(2) = h_i(3) + E + A'$$
$$h_{i+1}(3) = h_i(4) + A + B'$$
$$h_{i+1}(4) = h_i(0) + B + C'$$

5.krok: po spracovaní všetkých L 512–bitových blokov správy M sa získa výsledný hašovací kód $h_L=h$.

10.6 Kompresná funkcia RIPEMD–160

Kompresná funkcia, ktorá transformuje 512–bitový blok M_i na 160–bitový hašovací kód sa vykonáva v 10 rundách, pričom každá obsahuje 16 krokov. Elementárny krok algoritmu RIPEMD je uvedený na *Obr. 10.8*.



Obr. 10.8 Kompresná funkcia RIPEMD–160

V každej z piatich rúnd sa využíva iná logická funkcia, ktorej vstupy sú tri 32–bitové slová (B,C,D) a výstupom je 32–bitové slovo. Logické operácie sa realizujú po bitoch podľa *Tab. 10.7*.

Tab. 10.7 Logické funkcie RIPEMD–160

Číslo kroku	Typ logickej funkcie	$f(j,A,B,C,D)$
$0 \le j \le 15$	$f_1=f(j,B,C,D)$	$B \oplus C \oplus D$
$16 \le j \le 31$	$f_2=f(j,B,C,D)$	$(B \wedge C) \vee (\overline{B} \wedge D)$
$32 \le j \le 47$	$f_3=f(j,B,C,D)$	$(B \vee \overline{C}) \oplus D$
$48 \le j \le 63$	$f_4=f(j,B,C,D)$	$(B \wedge D) \vee (C \wedge \overline{D})$
$64 \le j \le 79$	$f_5=f(j,B,C,D)$	$B \oplus (C \vee \overline{D})$

Logické operátory AND, OR, NOT, XOR sú reprezentované symbolmi $\wedge, \vee, \overline{\ }, \oplus$, pričom pravdivostná tabuľka logických funkcií je uvedená v *Tab. 10.8*.

Tab. 10.8 Pravdivostná tabuľka logických funkcií

B	C	D	f_1	f_2	f_3	f_4	f_5
0	0	0	0	0	1	0	1
0	0	1	1	1	0	0	0
0	1	0	1	0	0	1	1
0	1	1	0	1	1	0	1
1	0	0	1	0	1	0	0
1	0	1	0	0	0	1	1
1	1	0	0	1	1	1	0

1	1	1	1	1	0	1	0

Blok M_i s dĺžkou 512–bitov sa v každej runde interpretuje ako pole $X_i = X[0,1,\ldots,15]$, pričom poradie použitia jednotlivých slov sa mení v každej runde a je dané permutáciou p a π. Uvedené permutácie sú definované *Tab. 10.9*. Permutáciu π možno vyjadriť v tvare $\pi(i) = (9i+5) \bmod 16$.

Tab. 10.9 Permutácie v RIPEMD–160

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$p(i)$	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
$\pi(i)$	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12

Vetva	Runda 1	Runda 2	Runda 3	Runda 4	Runda 5
Ľavá	bez zmeny	p	p^2	p^3	p^4
Pravá	π	$p\pi$	$p^2\pi$	$p^3\pi$	$p^4\pi$

Elementárny krok v každej runde používa operáciu rotácie vľavo *RLs*(*j*), ktorá pre každý krok používa rôzny počet bitov posunutia. Počet posunutí je daný *Tab. 10.10*.

Tab. 10.10 Tabuľka posunutí *RLs*(*j*)

Runda	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}
1	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8
2	12	13	11	15	6	9	9	7	12	15	11	13	7	8	7	7
3	13	15	14	11	7	7	6	8	13	14	13	12	5	5	6	9
4	14	11	12	14	8	6	5	5	14	12	15	14	9	9	8	6
5	15	12	13	13	9	5	8	6	15	11	12	11	8	6	5	5

Hašovací algoritmus RIPEMD–160 je veľmi podobný algoritmom MD5 a SHA–1, pretože všetky tri uvedené algoritmy sú odvodené od algoritmu MD4. Porovnanie uvedených algoritmov je uvedené v *Tab. 10.11*.

Tab. 10.11 Porovnanie algoritmov vybraných typov hašovacích funkcií

	MD5	SHA–1	RIPEMD–160
Dĺžka hašovacieho kódu [bit]	128	160	160
Veľkosť bloku spracovanie [bit]	512	512	512
Počet krokov	64 (4 rundy po 15 krokoch)	80 (4 rundy po 20 krokoch)	160 (5 párov rúnd po 16 krokoch)
Maximálna veľkosť správy [bit]	∞	$2^{64}-1$	$2^{64}-1$
Počet logických funkcií	4	4	5
Počet konštánt	64	4	10

10.7 Algoritmus HMAC

Koncepcia autentizačného kódu správy MAC je založená na využití symetrických blokových šifier, ktoré používajú tajný kľúč. V poslednom období sa však zvýšená pozornosť venuje MAC, ktorý je odvodený od hašovacích funkcií a označuje sa skratkou HMAC.

Dôvody zvýšeného záujmu o túto modifikáciu MAC možno zhrnúť takto:

1. hašovacie funkcie MD5 a SHA–1 v softvérovej implementácii sú vo všeobecnosti rýchlejšie než blokové šifry, napr. DES

2. knižnica hašovacích funkcií je plne dostupná

3. na hašovacie funkcie nie sú aplikované obmedzenia, najmä zo strany USA, resp. iných krajín, na rozdiel od obmedzení týkajúcich sa symetrických šifier

Na zabudovanie tajného kľúča do existujúcich hašovacích funkcií možno použiť viaceré postupy, z ktorých uvedieme HMAC definovanú v odporúčaní RFC2104. Tento typ algoritmu HMAC sa využíva v oblasti bezpečnosti IP sietí, resp. iných internetových protokoloch, ako napr. SSL. Pripravuje sa aj odporúčanie FIPS pre algoritmus HMAC.

Požiadavky na algoritmus HMAC podľa RFC2104 možno formulovať takto:

- použitie dostupných hašovacích funkcií bez ich modifikácie
- možnosť náhrady použitej hašovacej funkcie inou rýchlejšou, resp. bezpečnejšou hašovacou funkciou
- zachovanie pôvodnej výkonnosti, resp. bezpečnosti použitej hašovacej funkcie bez jej významnej degradácie
- jednoduché použitie kľúča
- ľahký odhad bezpečnosti a odolnosti voči kryptoanalýze

Štruktúra algoritmu HMAC je uvedená na *Obr. 10.9*. V štruktúre sú použité tieto symboly:

H – vložená hašovacia funkcia (napr. MD5, SHA–1, RIPEMD–160)

IV – inicializačný vektor

M – originálna správa

M_i – i–tý blok M, pre $0 \le i \le L-1$

L – počet blokov v správe M

b – počet bitov v bloku M_i

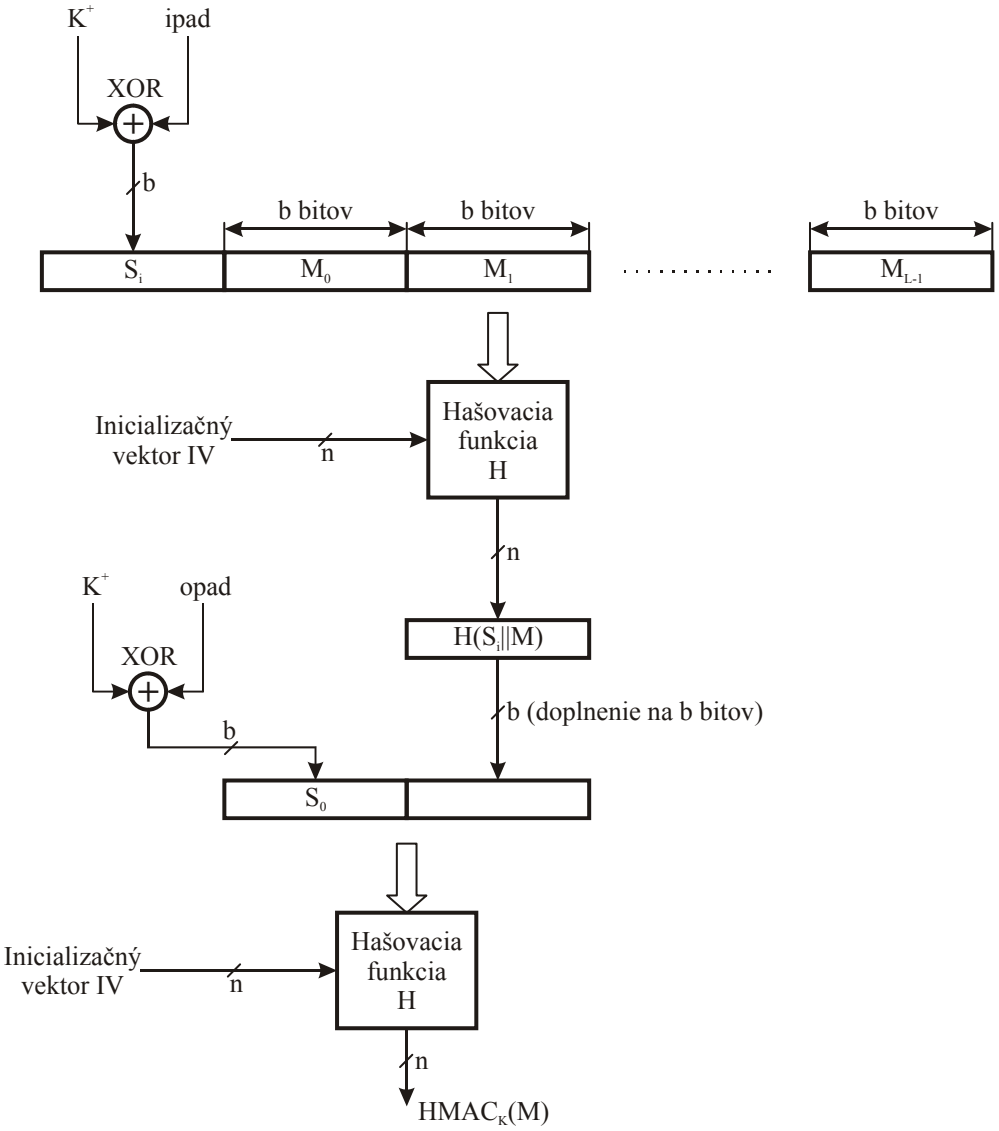
n – dĺžka hašovacieho kódu vloženej hašovacej funkcie, pričom obvykle $n<b$

K – tajný kľúč; dĺžka kľúča by mala byť väčšia, resp. rovná n

K^+ – doplnený kľúč K, pričom doplnenie na b–bitov sa realizuje 0 vľavo od posledného MSB bitu kľúča K

ipad – konštanta s veľkosťou 00110110 (36_H), ktorá sa opakuje $b/8$ krát

opad – konštanta s veľkosťou 01011100 ($5C_H$), ktorá sa opakuje $b/8$ krát



Obr. 10.9 Štruktúra algoritmu HMAC

Generovanie výsledného kódu HMAC možno vyjadriť v tvare

$$HMAC_K(M) = H\left[(K \oplus opad) \| H\left[(K^+ \oplus ipad) \| M\right]\right]$$

Slovne možno štruktúru HMAC opísať takto:

1. pridanie nulových bitov zľava ku kľúču K pred bit MSB tak, aby sa vytvoril reťazec K^+ s dĺžkou b–bitov (ak napr. kľúč K má 160 bitov a b=512, potom je potrebné pridať zľava 352 nulových bitov, t.j. 44 bytov)
2. vykonanie operácie XOR reťazcov K^+ a *ipad*, čím sa vytvorí blok S_i s dĺžkou b–bitov
3. pridanie správy M k bloku S_i
4. aplikovanie hašovacej funkcie H na reťazec vytvorený v kroku 3, pričom sa v prvom kroku použije inicializačný vektor IV s dĺžkou n–bitov. Výsledný hašovací kód $H(S_i \| M)$ sa doplní na b–bitov

5. vykonanie operácie XOR reťazcov K^+ a *opad*, čím sa vytvorí blok S_0 s dĺžkou *b*–bitov
6. pridanie bloku S_0 k výsledku v kroku 4
7. aplikovanie hašovacej funkcie *H* na reťazec vytvorený v kroku 6, čím vznikne výsledný reťazec $HMAC_K(M)$ s dĺžkou *n*–bitov

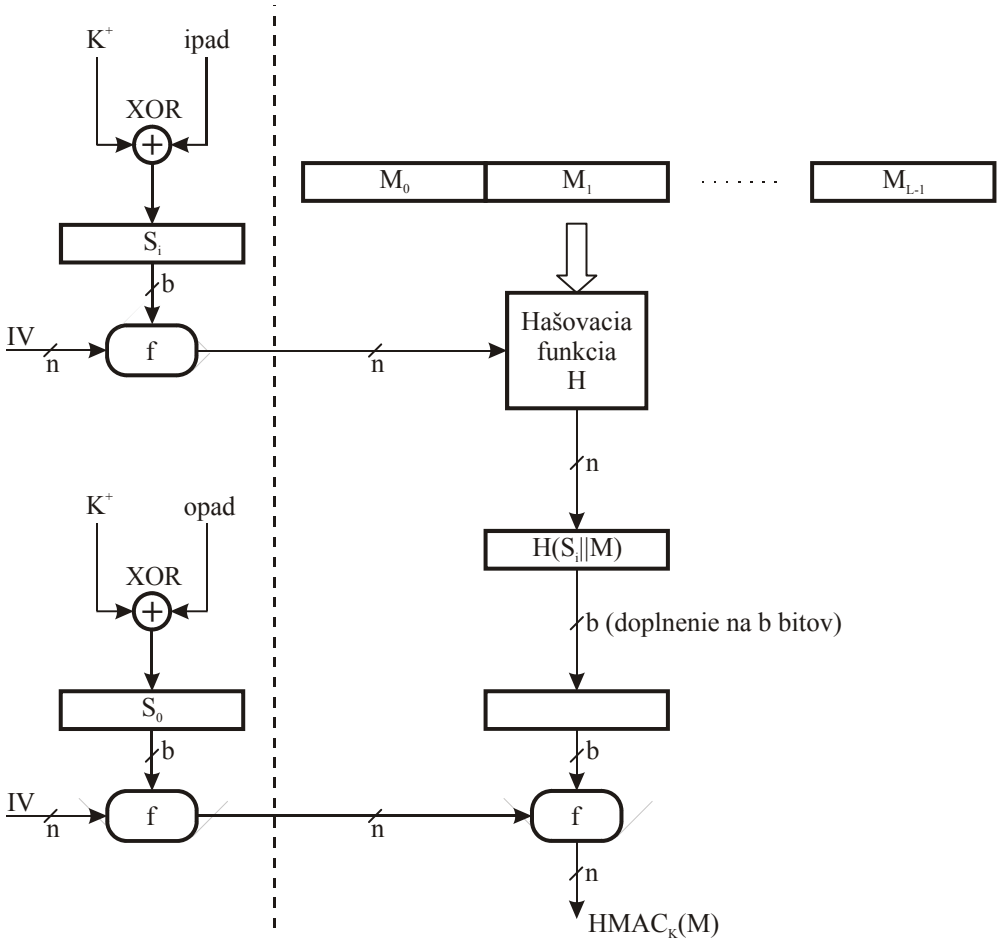
Je potrebné tiež poznamenať, že vytvorením blokov S_0 a S_1 a ich spracovaním hašovacou funkciou *H* sa náhodne vygenerovali dva kľúče z kľúča *K*.

Algoritmus HMAC má pre dlhé správy zhruba rovnaké časové nároky ako realizácia vloženej hašovacej funkcie *H*.

Efektívnejší spôsob implementácie algoritmu HMAC je zobrazený na *Obr. 10.10*, v ktorom sa v predstihu vypočítajú dve hodnoty

$$f\left(IV,\left(K^+ \oplus ipad\right)\right)$$
$$f\left(IV,\left(K^+ \oplus opad\right)\right)$$

Uvedené hodnoty je potrebné vypočítať, iba ak sa mení kľúč a fakticky reprezentujú inicializačný vektor pre hašovaciu funkciu.



Obr. 10.10 Efektívnejšia implementácia algoritmu HMAC

Tento efektívnejší spôsob implementácie algoritmu HMAC je zvlášť výhodný pre krátke správy *M*.

Bezpečnosť funkcií MAC je založená na aplikácii hašovacích funkcií resp. ju možno vyjadriť ako pravdepodobnosť vytvorenia dvoch hodnôt MAC tým istým kľúčom pri únosnej miere úsilia (čas, náklady).

Pravdepodobnosť úspešného útoku na algoritmus HMAC je ekvivalentná jednému z týchto útokov na vloženú hašovaciu funkciu:

1. nepovolaná osoba je schopná vypočítať výstupnú hodnotu kompresnej funkcie ak inicializačný vektor *IV* je náhodný, utajený a neznámy nepovolanej osobe
2. nepovolaná osoba nájde kolízie v hašovacej funkcii ak inicializačný vektor *IV* je náhodný a neznámy

Prvý typ útoku sa zakladá na myšlienke, že kompresná funkcia je ekvivalentná hašovacej funkcii aplikovanej na správu obsahujúcu jeden *b*–bitový blok. V tomto útoku sa inicializačný vektor *IV* nahradí náhodnou postupnosťou *n*–bitov. Útok na túto hašovaciu funkciu možno realizovať buď metódou totálnych skúšok, ktorý vyžaduje 2^n pokusov alebo metódou „narodeninového paradoxu“, ktorý je špeciálnym prípadom útoku druhého typu.

Druhý typ útoku vyhľadáva dve správy *M* a *M'*, ktoré produkujú rovnaký hašovací kód *H*(*M*)=*H*(*M'*). V tomto prípade ako bude uvedené v ďalšej časti sa uplatní efekt útoku na báze narodeninového paradoxu, ktorý ukazuje, že pre dĺžku hašovacieho kódu *n*–bitov je počet potrebných pokusov rovný $2^{n/2}$, čo by pre hašovaciu funkciu MD5 bolo už v súčasnosti kritické (2^{64} pokusov). Analýza tohto typu útoku však ukazuje, že vzhľadom na použitie kľúča *K* je aj použitie hašovacej funkcie MD5 s dĺžkou hašovacieho kódu 128 bitov v algoritme HMAC vyhovujúce.