Hašovacie funkcie

- základný princíp využitia
- štruktúra a vlastnosti hašovacích funkcií z rodiny SHAx
- ekvivalentná bezpečnosť (narodeninový paradox, dôsledky) hašovacích funkcií
- príklady využitia (kontrola integrity dát, digitálne podpisy, zreťazenie hašovacích funkcií)
- príklad využitia v OAEP výplňovej schéme pre RSA šifrovanie

Primárny zdroj informácii k dnešnej prednáške:

[1] prof. Ing. Dušan Levický, CSc.
APLIKOVANÁ KRYPTOGRAFIA
od utajenia správ ku kybernetickej bezpečnosti
Elfa, Košice, 2018 (str.182-206)

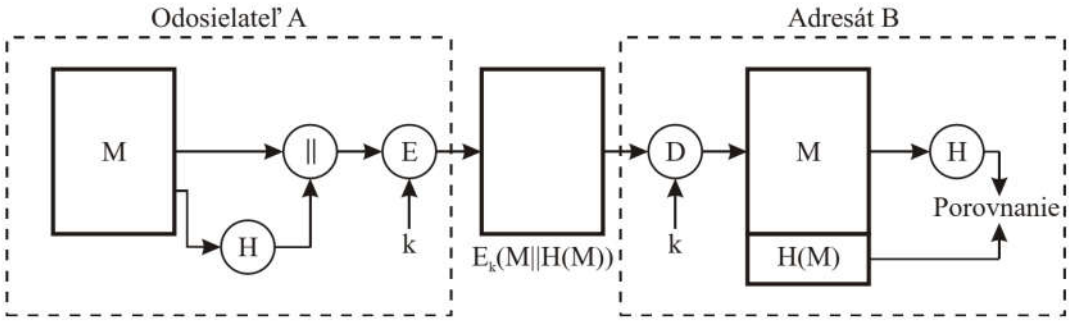
[2] Drutarovský, M.: Kryptografia pre vstavané kryptografické systémy. TUKE, 2017 (str.65-72)

Základný princíp a typické využitie

Údajová štruktúra bežne využívaná v informatike:
Hašovacia tabuľka (https://sk.wikipedia.org/wiki/Ha%C5%A1ovacia_tabu%C4%BEka)

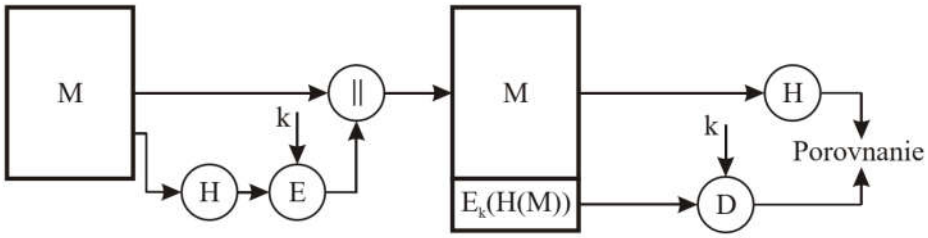
Význam v kryptografii:
Hašovacia funkcia (**hash function**) patrí do kategórie jednocestných funkcií (one-way functions). Hašovacia funkcia, podobne ako funkcia MAC (Message Authentication Code), ktorá bude preberaná v jednej z ďalších prednášok, používa ako **argument správu M s premennou dĺžkou**. Hodnota hašovacej funkcie H(M) sa označuje ako **hašovací kód h**, teda platí h=H(M). Hašovací kód h na rozdiel od MAC **nepoužíva tajný kľúč** a je **závislý len od správy M**. Hašovací kód sa tiež označuje ako **výťah zo správy** (**message digest**), resp. ako hašovacia hodnota a je **funkciou všetkých bitov správy**. Z toho vyplýva, že zmena akéhokoľvek bitu, resp. bitov správy vyvolá zmenu hašovacieho kódu, ktorý má teda vlastnosti **kryptografického kontrolného súčtu**, resp. je vhodný na **zabezpečenie integrity** správy.

Rôzne spôsoby použitia hašovacích funkcií sú uvedené na nasledujúcich obrázkoch (Obr. 9.4).



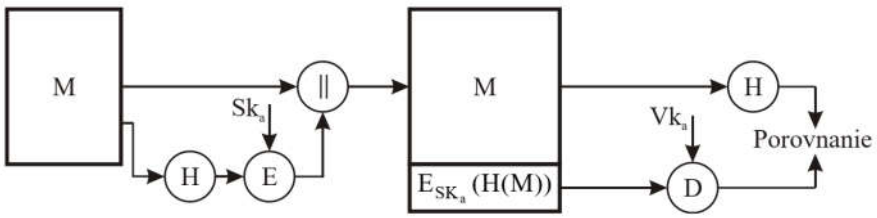
a)

Postup uvedený na *Obr. 9.4a* je založený na použití **symetrického šifrovania tajným kľúčom** *k*, ktoré sa aplikuje na správu *M* a pripojený hašovací kód. **Autentizácia správy** je založená na tom, že iba A a B vlastnia tajný kľúč, teda správa pochádza od A a nebola modifikovaná, ak **test zhodnosti hašovacieho kódu** na prijímacej strane je pozitívny. Pretože šifrovanie sa aplikuje na **celú správu** a hašovací kód je **zaručená aj dôvernosť správy**.



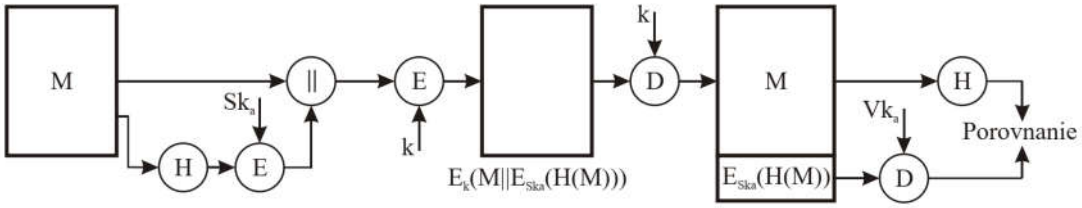
b)

V postupe na *Obr. 9.4b* sa šifrovanie aplikuje iba na hašovací kód, čo zabezpečuje autentizáciu správy. To redukuje výpočtovú náročnosť, čo je výhodné v aplikáciách, kde sa nevyžaduje zabezpečenie dôvernosti správy. Je potrebné tiež poznamenať, že kombinácia hašovacej funkcie a šifrovania vytvára vlastne MAC, pretože výraz $E_k\left(H\left(M\right)\right)$ je funkciou správy *M* s premennou dĺžkou a poskytuje výstup s konštantnou dĺžkou.



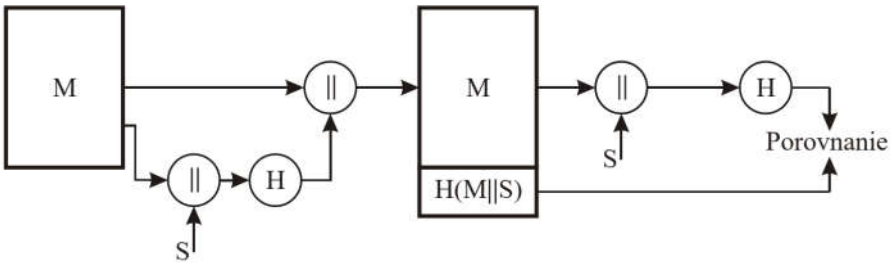
c)

Postup uvedený na *Obr. 9.4c* využíva princíp šifrovania s verejným kľúčom, ktorý na šifrovanie hašovacieho kódu používa súkromný kľúč odosielateľa. Podobne ako v predošlom postupe na *Obr. 9.4b*, tento postup zabezpečuje autentizáciu a funkciu digitálneho podpisu, pretože iba odosielateľ môže vytvoriť zašifrovaný hašovací kód. Uvedený postup je najviac rozšíreným postupom v oblasti autentizácie správ a v digitálnych podpisoch.



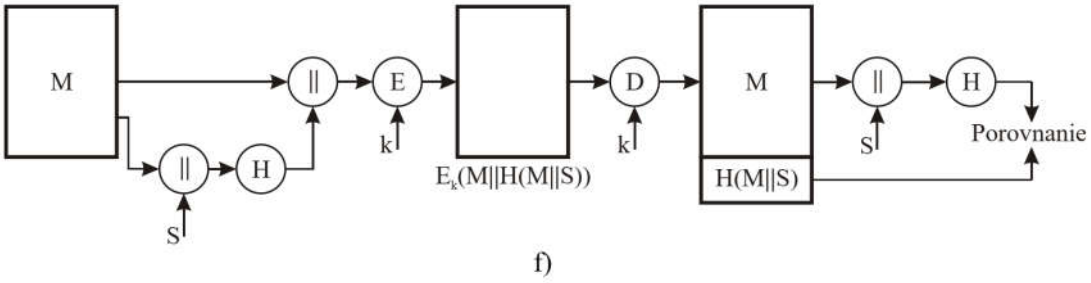
d)

Postup uvedený na *Obr. 9.4d* zabezpečuje dôvernosť správy a funkciu digitálneho podpisu. Hašovací kód sa podpisuje súkromným kľúčom odosielateľa a dôvernosť správy sa zabezpečuje symetrickým šifrovaním tajným kľúčom, ktoré sa aplikuje na správu *M* a zašifrovaný hašovací kód.



e)

V postupe, ktorý je uvedený na *Obr. 9.4e*, sa na autentizáciu využíva hašovacia funkcia, ale nepoužíva sa šifrovanie správy. V tejto technike dve komunikujúce strany poznajú spoločnú tajnú hodnotu *S*, ktorá sa pripája k správe *M*. Hašovací kód sa vytvára so spojenia *H*||*S*, teda je rovný $H(M \| S)$. Adresát prijme $H(M \| S)$, a pretože pozná *S* môže modifikovať $H(M \| S)$. Hodnota *S* sa neprenáša, preto nepovolaná osoba nemôže modifikovať správu, resp. generovať falošnú správu.



Obr. 9.4 Základné spôsoby použitia hašovacích funkcií

Ak je potrebné zabezpečiť dôvernosť správy predošlý postup sa doplní šifrovaním tajným kľúčom, ktoré sa aplikuje na hašovací kód a správu *M*. Tento postup je uvedený na *Obr. 9.4f*.

Z uvedených postupov autentizácie na báze hašovacích funkcií sú stále aktuálne postupy, kde sa nevyžaduje zabezpečenie dôvernosti správy šifrovaním, čo znižuje nároky na výpočtovú náročnosť *(Obr. 9.4b, c, e)*.

Aktuálnosť uvedených postupov je motivovaná najmä tým, že softvérová implementácia šifrovania je stále relatívne pomalá, cena hardvéru na šifrovanie nie je zanedbateľná a hardvér je optimalizovaný vzhľadom na určitú veľkosť bloku dát. Navyše šifrovacie algoritmy sú obvykle patentované a na ich využívanie je potrebné žiadať udelenie licencie, ktoré sú prísne kontrolované.

Vlastnosti hašovacích funkcií

Hašovacie funkcie predstavujú určitú triedu funkcií $H(M)$, ktoré spracovávajú správu M **prakticky neobmedzenej dĺžky**, pričom ich výstupom je hodnota, resp. hašovací kód h pevnej a **relatívne malej dĺžky**, teda

$$h=H(M)$$

kde h má **pevnú dĺžku desiatky až stovky bitov**.

Hašovací kód h predstavuje **identifikačný údaj** analogický „**s odtlačkom prsta**" súboru správy, resp. bloku dát. Pre účely autentizácie správ musia hašovacie funkcie spĺňať tieto **základné podmienky**:
• jednocestnosť
• odolnosť voči kolízii.

Jednocestnosť hašovacích funkcií možno formulovať takto:
• pre dané M je **jednoduché vypočítať** $h=H(M)$,
• pre dané h je **veľmi ťažké „vypočítať"** M.

Odolnosť voči kolízii spočíva v tom, že:
• **ak je dané *M*** je veľmi ťažké nájsť **iné *M'***, teda $M\neq M'$, tak aby $H(M)=H(M')$. Táto vlastnosť sa označuje ako **slabá odolnosť voči kolízii** (**weak collision resistance**, alebo tiež **second-preimage resistance**),
• je veľmi ťažké nájsť **akýkoľvek pár** (M, M') taký, aby $H(M)=H(M')$. Táto vlastnosť sa označuje ako **silná odolnosť voči kolízii** (**strong collision resistance**).

K slabej a silnej odolnosti voči kolízii hašovacích funkcii pozri napr. tiež napr. diskusiu (spomenuté je aj využitie **narodeninového paradoxu**):
https://crypto.stackexchange.com/questions/33622/weak-and-strong-collision-resistance-again

Hašovacie funkcie plnia v **oblasti autentizácie dve úlohy**. Sú to:
• kompresia
• hašovanie.

Funkcia kompresie spočíva v tom, že veľmi dlhý vstup, resp. argument funkcie $H(M)$, ktorým je správa M je transformovaný na veľmi krátky výstup. Napr. **vstup má dĺžku 2^64–1 bitov** a **výstup má dĺžku 160 bitov**. Význam kompresie má v tomto prípade odlišný význam než kompresia v klasickom ponímaní. Pôvodná informácia nie je vo výstupnom kóde obsiahnutá celá, takže **z výstupného hašovacieho kódu nie je možné získať pôvodnú správu**.

Funkcia hašovania spočíva v tom, že výstupný hašovací kód h jednoznačne reprezentuje vstupnú správu M. Zmena správy M vedie k zmene hašovacieho kódu. Hodnota h sa označuje ako **výťah zo správy *M*** (Message digest).

Narodeninový paradox

Čo si bežne predstavíme pod paradoxom? Na prednáške bude opísaný jednoduchý paradox s povrázkom, aby sme si uvedomili, ako zradné sú paradoxy ...

V **oblasti útokov** na hašovacie funkcie hraje dôležitú úlohu **narodeninový paradox**. Na základe narodeninového paradoxu je v praktických kryptografických aplikáciách **volená veľkosť hašovacieho kódu** (odtlačku, výťahu zo správy). Zjednodušene povedané, volíme **také hašovacie funkcie**, ktoré majú **veľkosť hašovacieho kódu dva-krát väčšiu ako je veľkosť kľúča použitej symetrickej šifry**. V takom prípade budú mať uvedené algoritmy porovnateľnú ekvivalentnú bezpečnosť (predpokladáme, že na uvedené algoritmy **neexistuje iný efektívny útok**, ako **prehľadávanie priestoru kľúčov** u použitej symetrickej šifry a **náhodné hľadanie kolízií** použitej hašovacej funkcie. Vidno to aj z tabuľky, ktorá bola uvedená v predchádzajúcej prednáške:

Tab. 11.6 Ekvivalentná bezpečnosť kryptografických algoritmov

Ekvivalentná bezpečnosť [b]	Symetrické algoritmy	Algoritmy DSA DH	Algoritmus RSA	Eliptické krivky	Hašovacie funkcie
80	2DES	VK=1024 SK=160	$n = 1024$	$n = 160 - 223$	SHA-1
112	3DES	VK=2048 SK=224	$n = 2048$	$n = 224 - 255$	SHA-224
128	AES–128	VK=3072 SK=256	$n = 3072$	$n = 256 - 383$	SHA-256
192	AES–192	VK=7680 SK=384	$n = 7680$	$n = 384 - 511$	SHA-384
256	AES–256	VK=15360 SK=512	$n = 15360$	$n > 512$	SHA-512

Prvý stĺpec *Tab. 11.6* udáva ekvivalentnú bezpečnosť kryptografických algoritmov v rozsahu 80 až 256 bitov.

Poznámka:
Útok na hašovacie funkcie s využitím narodeninového paradoxu je tzv. **generický útok**, t.j. nevyužíva **žiadnu slabinu** hašovacej funkcie. Preto sú v rámci štandardov hašovacie funkcie **prispôsobené tomuto typu útoku** a preto napr. v rodine hašovacích funkcií SHA **existujú** hašovacie funkcie s **hašovacím kódom 224, 256, 384 a 512 bitov**.

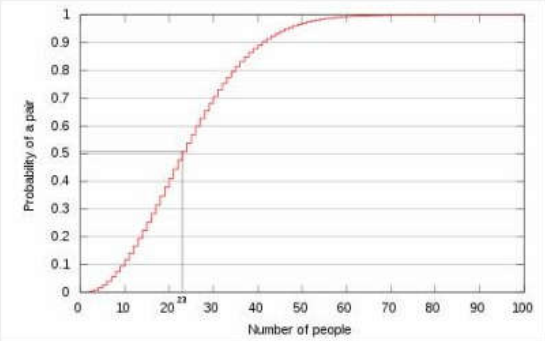
https://www.investicniweb.cz/2012-11-23-tyto-triky-s-cisly-zmeni-vas-pohled-na-svet/

2. Narozeninový paradox

V kanceláři je 23 zaměstnanců. Jaká je pravděpodobnost, že budou mít dva zaměstnanci narozeniny ve stejný den? (Pro zjednodušení neuvažujme přestupný rok.)

Odpověď je zhruba 50 %.

Jakmile by bylo v kanceláři 366 lidí, bylo by jisté, že mají dva narozeniny ve stejný den, protože rok má pouze 365 dní. Protože neuvažujeme 29. února, pravděpodobnosti narozenin jsou pro všechny dny v roce rovnocenné. Kdyby bylo v kanceláři 57 lidí, byla by šance na společnou oslavu narozenin dvou kolegů 99 %.



Než abychom počítali pravděpodobnost, že dva lidé mají narozeniny ve stejný den, kalkulujme opak. Výsledek dostaneme, když získanou pravděpodobnost odečteme od jedné.

Šance dvou náhodně vybraných zaměstnanců, že nebudou mít narozeniny společně, je 364/365. Pravděpodobnost u třetí osoby bude 363/365. Pokud projdeme celou kancelář a vynásobíme zlomky, zjistíme, že pravděpodobnost, aby se narozeniny zaměstnanců ani v jednom případě nesešly ve stejný den, je 49,27 %:

365/365 x 364/365 x 363/365 x 362/365 x ... x 343/365 = 0,4927

Nyní číslo odečteme od jedné a přijdeme na to, že námi hledaná pravděpodobnost je 50,73 %.

Realizácia hašovacích funkcií

Z hľadiska **spôsobu realizácie** možno hašovaciu funkciu rozdeliť do dvoch **prakticky využívaných** kategórií. Sú to:

• zreťazené hašovacie funkcie
• iteračné hašovacie funkcie.

Zreťazené hašovacie funkcie využívajú režim CBC blokových šifier, ale bez použitia tajného kľúča. Vstupná správa M sa rozdelí na bloky $M_1,M_2,...,M_n$ s konštantnou veľkosťou a výpočet hašovacieho kódu h sa realizuje algoritmom, ktorý môžeme opísať takto:

$h_0 =$ počiatočná hodnota
$h_i = E_{M_i}(h_{i-1})$
$h = h_n$

Blok M_i sa teda **používa ako kľúč** pri šifrovaní, ktoré produkuje hodnotu h_i. Po n krokoch, resp. **zašifrovaní všetkých blokov** sa získa **výsledný hašovací kód** h. Ak sa na zašifrovanie použije **algoritmus DES**, dĺžka blokov M_i aj dĺžka hašovacieho kódu je **rovná 64 bitov**. Je potrebné však poznamenať, že uvedená dĺžka je **málo bezpečná**.

V praxi sa využívajú aj iné **špecializované zapojenia blokových šifier** ako napr. ([2, str. 63-71):

• konfigurácia typu Davies-Meyer
• konfigurácia typu Matyas-Meyer-Oseas
• konfigurácia typu Miyaguchi-Preneel
• Hiroseho konfigurácia s dvojnásobnou dĺžkou.

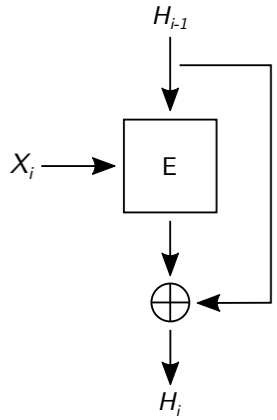
Napr. konfigurácia typu Davies-Meyer je opísaná Algoritmom 5.1 a zobrazená na Obr. 5.2.

Kompresná funkcia typu Davies – Meyer je tvorená zapojením blokovej šifry `E` zobrazenom na obr. 5.2. Vstupný blok X_i je aktuálne použitý kľúč a výstup hašovacej funkcie pre H_{i-1} pre predchádzajúci blok X_{i-1} je vstupom blokovej šifry. Výpočet hašovacej funkcie pre celý blok vstupných dát je vyjadrený algoritmom 5.1.

Algoritmus 5.1 Výpočet hašovacieho kódu s využitím konfigurácie zapojenia blokovej šifry typu Davies – Meyer

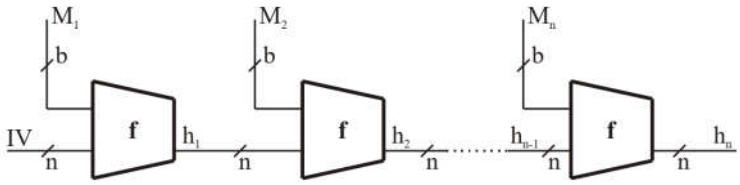
Input: vstupné data $\boldsymbol{X} = \{X_t X_{t-1} \dots X_1\}$ rozdelené na k-bitové bloky X_i, $1 \leqq i \leqq t$, n-bitový IV, `E` – bloková šifra (k-bitový kľúč, n-bitový blok)
Output: n-bitový výstup H_t

1: $H_0 \leftarrow \mathrm{IV}$
2: **for** i from 1 to t **do**
3: $\quad H_i \leftarrow \mathrm{E}_{X_i}(H_{i-1}) \oplus H_{i-1}$
4: **return** H_t



Obr. 5.2: Realizácia kompresnej funkcie $f()$ s využitím konfigurácie zapojenia blokovej šifry typu Davies – Meyer

Určité úspechy v oblasti kryptoanalýzy hašovacích funkcií viedli k vývoju nových typov hašovacích funkcií, ktoré sa označujú ako **iteračné hašovacie funkcie**. Typická štruktúra takýchto funkcií je uvedená na *Obr. 9.6*.



Obr. 9.6 Štruktúra iteračných hašovacích funkcií

Algoritmus na implementáciu iteračných hašovacích funkcií je založený na opakovanom použití **komprimačnej funkcie *f***, ktorá má dva vstupy. Jeden vstup je *n*-bitový vstup z predošlého kroku a druhý vstup predstavuje *b*-bitový blok vstupnej správy *M*. Pretože $n<b$, funkcia *f* sa označuje ako komprimačná. Proces iteračného výpočtu hašovacieho kódu *h* možno vyjadriť v tvare

IV = h_0 = počiatočná hodnota

$h_i = f\left(h_{i-1}, M_i\right)$ pre $1 \le \mathrm{i} \le \mathrm{n}$

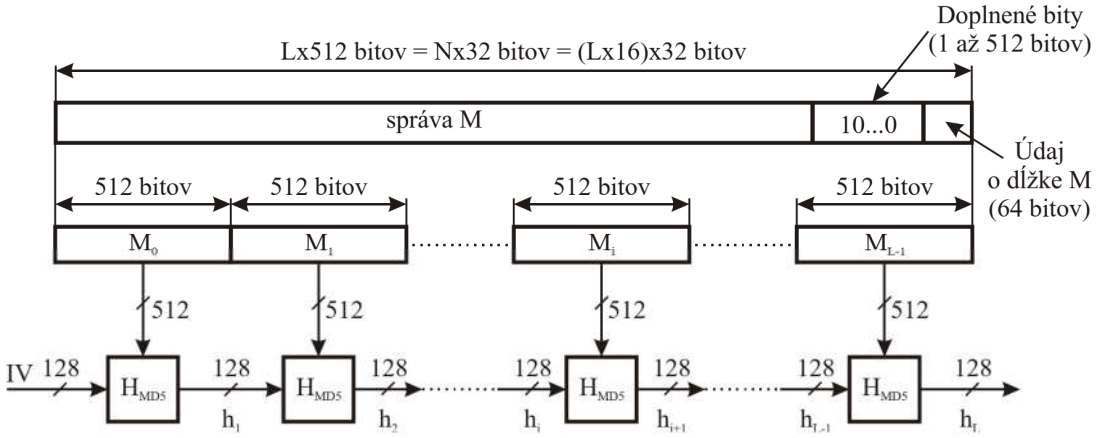
$h = h_n$

Uvedenú štruktúru používa väčšina v súčasnosti používaných hašovacích funkcií, napr. **MD5**, **SHA-1** a **RIPEMD-160** a pod. V ďalšej časti opíšeme len skupinu hašovacích funkcií SHA.

Algoritmy skupiny hašovacích funkcií SHA

Algoritmus SHA (Secure Hash Algorithm) bol zavedený **štandardom NIST** a publikovaný v technickej špecifikácii **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**.



Obr. 10.1 Bloková schéma algoritmu MD5

Bloková schéma algoritmu SHA-1 je **totožná so schémou algoritmu MD5** na *Obr. 10.1*, pričom algoritmus spracovania správy *M* pozostáva z týchto krokov:

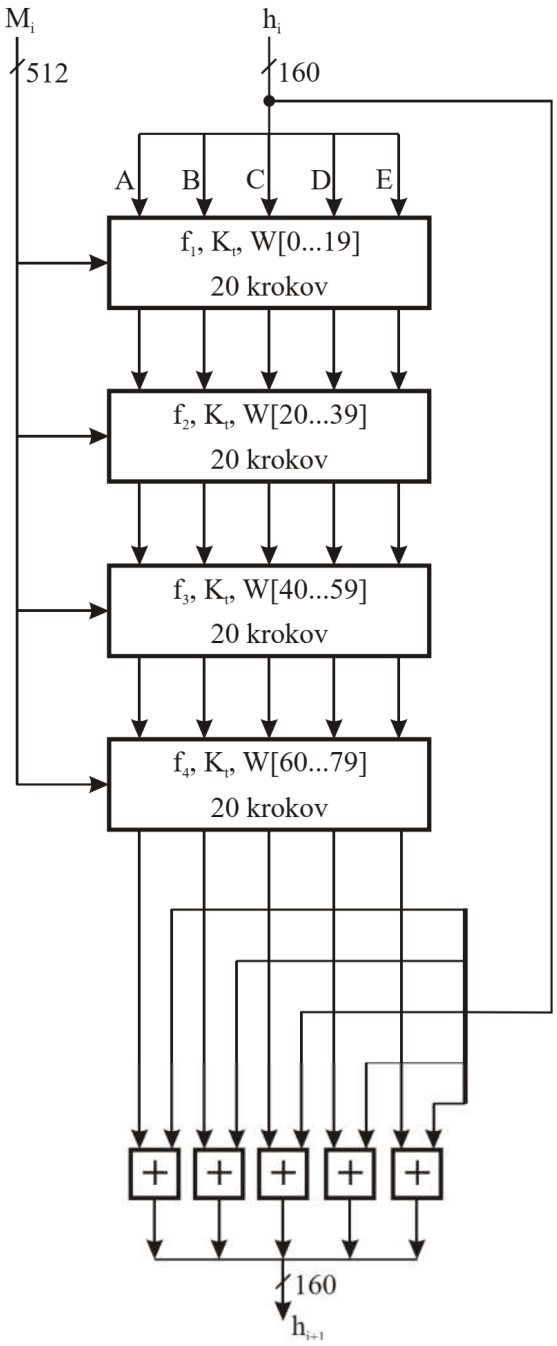
1.krok: doplnenie originálnej správy *M* skupinou bitov tak, aby dĺžka doplnenej správy bola kongruentná **448 modulo 512**. Doplnenie sa **realizuje vždy**, pričom počet doplnených bitov sa **mení v rozsahu od 1 do 512** a skupina doplnených bitov **má tvar 1 0 0 ... 0**.

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

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 $h_L=h$.

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)*.

$$A \leftarrow E + f_1(B, C, D) + (<<< RL5(A)) + W_t + K_t$$
$$B \leftarrow A$$
$$C \leftarrow <<< RL30(B)$$
$$D \leftarrow C$$
$$E \leftarrow D$$

kde

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

t – číslo kroku $0 \le t \le 79$

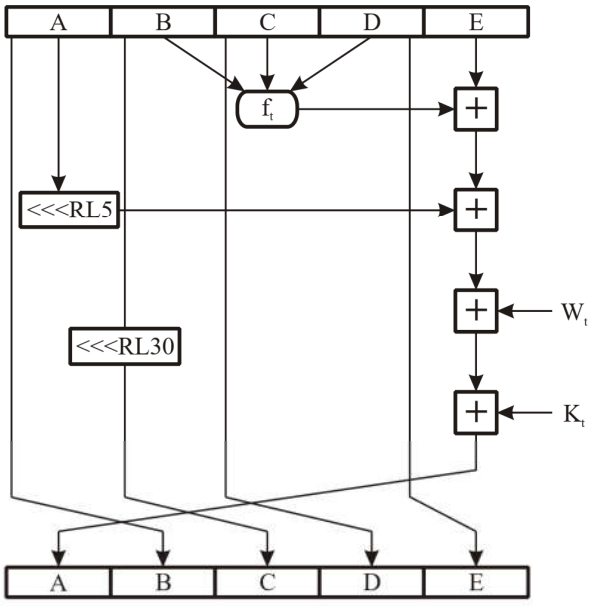
f_l(B,C,D) – logická funkcia pre krok t

<<< RLs – rotácia vľavo 32-bitového slova o s bitov a s=5, resp. s=30

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

K_t – 32-bitová konštanta

$\boxplus$ – sčítanie modulo 2^{32}



Obr. 10.5 Kompresná funkcia SHA–1

Logická funkcia f_1(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

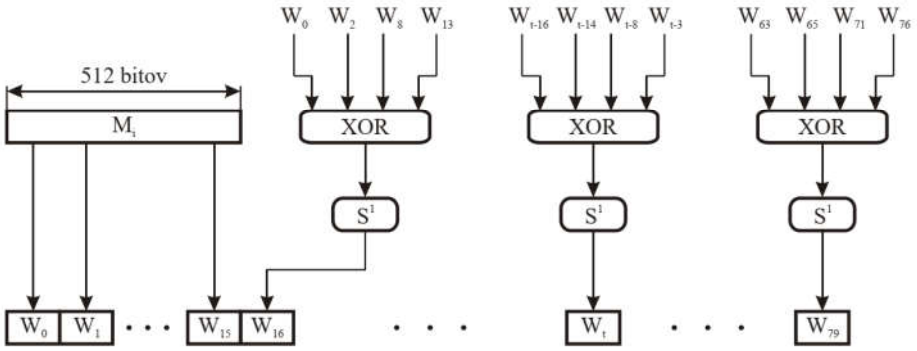
Číslo kroku	Typ logickej funkcie	f(B,C,D)
$0 \le t \le 19$	f_1(B,C,D)	$(B \wedge C) \vee (\bar{B} \wedge D)$
$20 \le t \le 39$	f_2(B,C,D)	$B \oplus C \oplus D$
$40 \le t \le 59$	f_3(B,C,D)	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$60 \le t \le 79$	f_4(B,C,D)	$B \oplus C \oplus D$

Logické operátory AND, OR, NOT a XOR sú reprezentované symbolmi $\wedge, \vee, \bar{}, \oplus$ 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*.

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}.

Ďalšie podrobnosti os SHA-1 je možné nájsť v [1].

V **roku 2002** 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*. Niekedy sa táto skupina hašovacích funkcií **označuje ako skupina SHA2**.

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.

V **roku 2004** bol FIPS 180-2 doplnený o hašovaciu **funkciu SHA-224** s dĺžkou **hašovacieho kódu 224 bitov**. Táto hašovacia funkcia je založená na hašovacej funkcii SHA-256, pričom výpočet SHA-224 pozostáva z dvoch krokov. V prvom kroku sa realizuje výpočet SHA-256 a v druhom kroku sa jej výstupný kód skráti na 224 bitov. Hašovacia funkcia **SHA-224** poskytuje **ekvivalentnú kryptografickú bezpečnosť 112 bitov**, čo zodpovedá **bezpečnosti algoritmu 3DES**.

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	64
Počet krokov algoritmu	80	80	80	80
Ekvivalentná bezpečnosť v bitoch	80	128	192	256

Skupina hašovacích funkcií SHA3

SHA-3 je skupina hašovacích funkcií využívajúca základné **kryptografické operácie** (cryptographic primitive) **z rodiny Keccak**, ktoré boli navrhnuté skupinou autorov Guido Bertoni, Joan Daemen, Michael Peeters a Gilles Van Assche.

V **októbri 2012** hašovacia funkcia **SHA-3 zvíťazila v súťaži hašovacích funkcií**, ktorú organizovala **NIST**. Zároveň je potrebné poznamenať, že hašovacia funkcia **SHA-3 nemá zatiaľ*) úlohu nahradiť skupinu** hašovacích funkcií SHA-2, ale predstavuje alternatívu k existujúcim hašovacím funkciám.

*Pozri napr. diskusiu na stránke:
https://www.csoonline.com/article/3256088/why-arent-we-using-sha3.html

SHA-3 používa **novú vnútornú štruktúru** (sponge construction), ktorá využíva logické operácie (XOR) a permutácie blokov vstupnej správy. Parametre hašovacích funkcií SHA-3 sú uvedené v *Tab. 10.6*

Tab. 10.6 Parametre hašovacích funkcií SHA-3

	SHA3–224	SHA3–256	SHA3–384	SHA3–512
Dĺžka hašovacieho kódu	224	256	384	512
Veľkosť bloku	1152	1088	832	576
Veľkosť slova	64	64	64	64
Počet krokov algoritmu	24	24	24	24

SHA-3 splňuje podmienku **kompatibility** hašovacieho kódu so **skupinou hašovacích funkcií SHA-2**, ktoré majú tento parameter 224, 256, 384 a 512 bitov pri maximálnej dĺžke vstupnej správy 2^64-1 a **využíva 64-bitové slová**, čo je **výhoda** pri implementácii na **platformách nových 64-bitových CPU.**

Poznámka:
V **praxi je aktuálne možné používať** SHA-2 a SHA-3, **SHA-1 nie je v súčasnosti považovaná za bezpečnú**.

Okrem vyššie uvedených verzií SHA3 funkcií, ktoré **generujú pevné dĺžky výstupných hašovacích kódov** (224, 256, 383 a 512 bitov) sú v štandarde FIPS PUB 202 (https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf) definované aj varianty SAH3 s **rozšíriteľnou výstupnou funkciou** (Extendable-Output Function) – SHA3 XOFs (SHAKE128 a SHAKE256). Ich typické využitie je v aplikáciách, kde je potrebné generovať maskovacie hodnoty (napr. rôzne výplňové schémy ako napr. RSA OAEP, pseudonáhodné generovanie parametrov šifrovacích algoritmov ako napr. rôzne post-kvantové kryptografické algoritmy, a pod.). Výhodou je možnosť priamo využiť štandardizovaný algoritmus bez nutnosti vytvárať pseudonáhodný výstup zapojením viacerých stavebných blokov.

Historicky prehľad najčastejšie používaných hašovacích funkcií je zobrazený v nasledujúcej tabuľke (https://crypto.stackexchange.com/questions/68307/what-is-the-difference-between-sha-3-and-sha-256):

History. There has been a long line of hash functions standardized by NIST in FIPS 180, the Secure Hash Standard, and later FIPS 202, the SHA-3 Standard: Permutation-Based and Extendable-Output Functions. More details and history, including the related MD4 and MD5 hash functions on which SHA-0, SHA-1, and SHA-2—but not SHA-3—were based:

hash	year	coll. res.	size (bits)	design	broken?
MD4	1990	64	128	32-bit ARX DM	1995
SHA-0 (SHA)	1993	80	160	32-bit ARX DM	1998
MD5	1993	64	128	32-bit ARX DM	2004
SHA-1	1995	80	160	32-bit ARX DM	2005
SHA-256 (SHA-2)	2002	128	256	32-bit ARX DM	
SHA-384 (SHA-2)	2002	192	384	64-bit ARX DM	
SHA-512 (SHA-2)	2002	256	512	64-bit ARX DM	
SHA-224 (SHA-2)	2008	112	224	32-bit ARX DM	
SHA-512/224	2012	112	224	64-bit ARX DM	
SHA-512/256	2012	128	256	64-bit ARX DM	
SHA3-224	2013	112	224	64-bit Keccak sponge	
SHA3-256	2013	128	256	64-bit Keccak sponge	
SHA3-384	2013	192	384	64-bit Keccak sponge	
SHA3-512	2013	256	512	64-bit Keccak sponge	
SHAKE128	2013	$\leq$128	any	64-bit Keccak sponge	
SHAKE256	2013	$\leq$256	any	64-bit Keccak sponge	

ARX DM je skratka pre štruktúru využívajúcu **A**dd-**R**otate-**X**or operácie a štruktúru Davies-Mayer (https://en.wikipedia.org/wiki/One-way_compression_function#Davies%E2%80%93Meyer)

V praxi sa využívajú aj **iné typy hašovacích funkcií**, ktoré majú **výrazne odlišné využitie** ako napr. hašovacie funkcie z rodiny SHA. Ako typické príklady môžeme napr. uviesť (podrobnejšie sa im budeme venovať v inžinierskom štúdiu v predmete BIKS):

SipHash (https://en.wikipedia.org/wiki/SipHash)
Hašovacia funkcia optimalizovaná pre spracovanie krátkych paketov, využívaná napr. aj v moderných VPN sieťach a implementáciách serverov ako protiopatrenie voči "hash flooding" denial-of-service attacks (HashDoS) útokom.

Scrypt (https://en.wikipedia.org/wiki/Scrypt),
Argon2 (https://en.wikipedia.org/wiki/Argon2), …
Hašovacie funkcie špecializované pre hašovanie hesiel a implementácie využívané v kryptomenách.

Blake 3 (https://github.com/BLAKE3-team/BLAKE3)
Rýchla kryptografická hašovacia funkcia využívajúca paralelizmus použitej datovej štruktúry (Merkelov strom - https://en.wikipedia.org/wiki/Merkle_tree) a SIMD paralelizmus na úrovni inštrukcií.

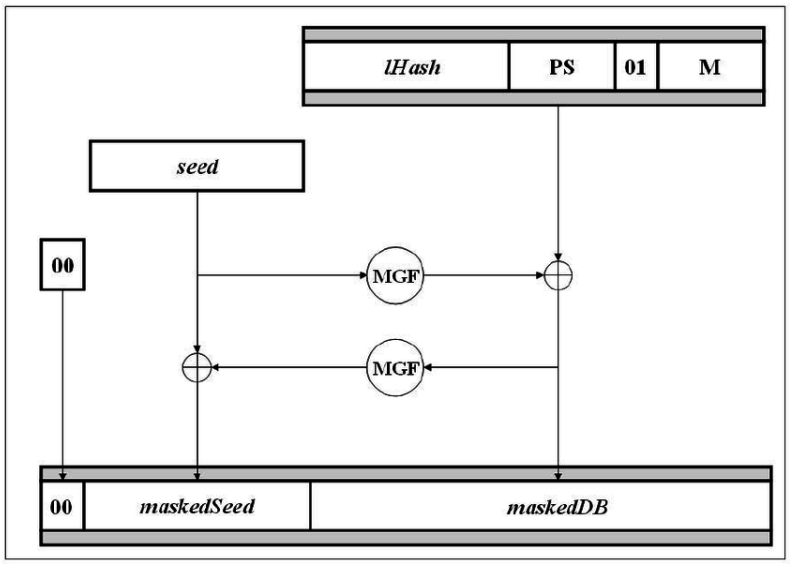
...

Príklad využitia hašovacích funkcií v móde RSA OAEP (Optimal Asymmetric Encryption Padding – optimálna asymetrická výplňová schéma)

- definovaná v štandarde PKCS 1 (https://en.wikipedia.org/wiki/PKCS_1)
- rieši bezpečnostný problém **„školskej implementácie“ RSA**. RSA šifrovanie má tzv. **multiplikatívnuu vlastnosť**, ktorú je možné bez protiopatrení zneužiť na dešifrovanie správy bez znalosti dešifrovacieho exponentu **postupným podsúvaním modifikovaných zašifrovaných textov, ktoré súvisia s pôvodnou správou**.

Detailnejší opis (vrátane základného opisu tzv. **Mangerovho útoku**) je možné nájsť napr. v dokumente: http://crypto-world.info/klima/2001/chip-2001-11-172-175.pdf

!!! Je dôležité používať nie len **bezpečné algoritmy**, ale tiež používať ich v **„správnom zapojení“** a **správne implementované**.



Využitie **Feistelovej štruktúry** v OAEP maskovaní správy **M pred RSA šifrovaním** (zdroj: PKCS1 štandard)

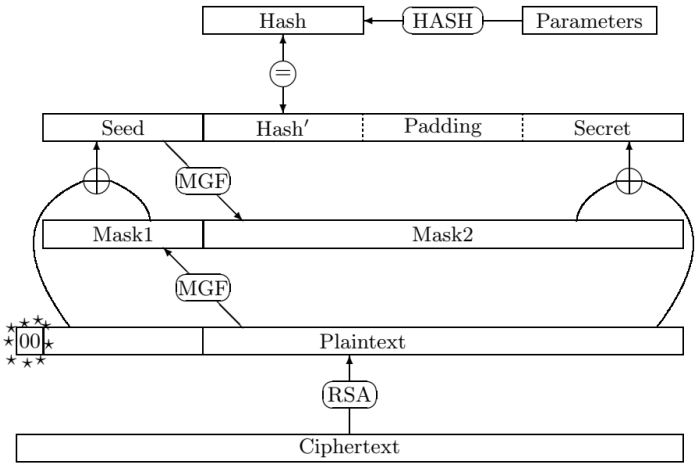


Fig. 1. RSAES-OAEP Decoding

Znázornenie krokov pri demaskovaní správy **M** (Secret) **po RSA dešifrovaní** (zdroj: https://www.iacr.org/archive/crypto2001/21390229.pdf)