Y36BEZ – Bezpečnost přenosu a zpracování dat

Róbert Lórencz

5. přednáška

Hašovací funkce, MD5, SHA-x, HMAC

http://service.felk.cvut.cz/courses/Y36BEZ
lorencz@fel.cvut.cz

Obsah přednášky

- Hašovací funkce
- MD5
- SHA-x
- HMAC

Hašovací funkce (4)

Vstup a výstup hašovací funkce

- Hašovací funkce h zpracovává prakticky neomezeně dlouhá vstupní data M na krátký výstupní hašový kód $h(M)$ pevné a předem stanovené délky n bitů.
- Například u hašovacích funkcí MD5/SHA-1/SHA-256/SHA-512 je to 128/160/256/512 bitů.

Hašovací funkce jako náhodné orákulum

Z hlediska bezpečnosti požadujeme, aby se hašovací funkce chovala jako náhodné orákulum. Odtud se odvozují bezpečnostní vlastnosti.

- Orákulum nazýváme libovolný stroj ("podivuhodných vlastností"), který na základě vstupu odpovídá nějakým výstupem. Má pouze vlastnost, že na tentýž vstup odpovídá stejným výstupem.
- Náhodné orákulum je orákulum, které na nový vstup odpovídá náhodným výběrem výstupu z množiny výstupů.

Hašovací funkce (5)

Odolnost proti kolizi 1. a 2. řádu, bezkoliznost 1. a 2. řádu

Definice – Bezkoliznost 1. řádu

Je odolnost proti kolizi a požaduje, aby bylo výpočetně nezvládnutelné nalezení libovolných dvou různých (byť nesmyslných) zpráv M a M' tak, že $h(M) = h(M')$. Pokud se toto stane, říkáme, že jsme nalezli kolizi.

- Bezkoliznost se zásadně využívá k digitálním podpisům.
- Nepodepisuje se přímo zpráva (často dlouhá), ale pouze její haš.
- Bezkoliznost zaručuje, že není možné nalézt dva dokumenty se stejnou haší. Proto můžeme podepisovat haš.

Definice – Bezkoliznost 2. řádu

Hašovací funkce h je odolná proti nalezení 2. vzoru, jestliže pro daný náhodný vzor x je výpočetně nezvládnutelné nalézt 2. vzor $y \neq x$ tak, že $h(x) = h(y)$.

Hašovací funkce (6)

Odolnost proti nalezení kolize 1. řádu

- Pokud se hašovací funkce $f : \{0,1\}^d \rightarrow \{0,1\}^n$ bude chovat jako náhodné orákulum $\Rightarrow$ složitost nalezení kolize s 50% pravděpodobností je $\approx 2^{\frac{n}{2}}$, namísto očekávaných 2^{n-1}
- Tato složitost nalezení kolize $2^{\frac{n}{2}}$ vyplývá z tzv. narozeninového paradoxu (viz dále).
- Například pro 160 bitový hašový kód bychom očekávali 2^{160} zpráv, paradoxně je to pouhých 2^{80} zpráv.
- U náhodného orákula neznáme rychlejší cestu nalézání kolizí.

Odolnost proti nalezení kolize 2. řádu

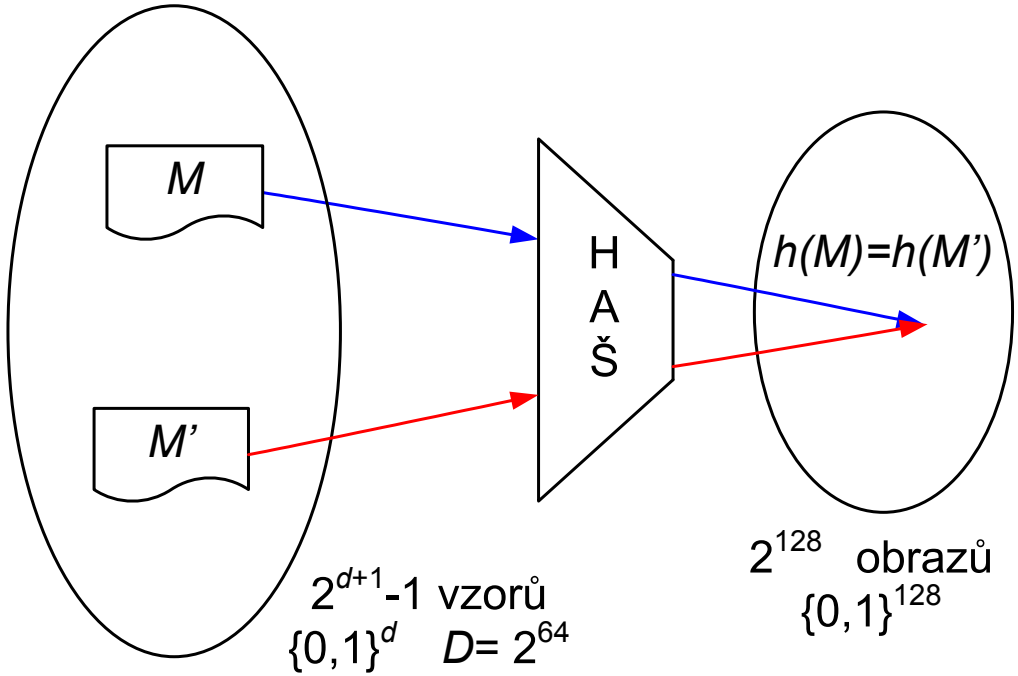
- Pokud se hašovací funkce $f : \{0,1\}^d \rightarrow \{0,1\}^n$ bude chovat jako náhodné orákulum $\Rightarrow$ složitost nalezení 2. vzoru je $\approx 2^n$.

Když víme, jak vzory nalézat jednodušeji, a to buď v případě kolizí 1. nebo 2. řádu $\Rightarrow$ hovoříme o prolomení hašovací funkce.

Hašovací funkce (7)

Pojem bezkoliznosti

Pokud máme hašovací funkci MD5 $\Rightarrow$ možných zpráv je mnoho $(1 + 2^1 + \cdots + 2^d)$, kde $d = 2^{64} - 1$ a hašovacích kódů pouze $2^{128} \Rightarrow$ existuje ohromné množství zpráv vedoucích na tentýž hašový kód, v průměru je to řádově 2^{d-127}.



Kolizí existuje velké množství, ale nalezení jen jedné kolize je nad naše výpočetní možnosti, i když zohledníme narozeninový paradox.

Z principu definice hašovací funkce plyne:

- jak existence obrovského množství kolizí,
- ale stejně tak skutečnost, že nalezení těchto kolizí je pro nás příliš těžká a výpočetně neproveditelná úloha.

Hašovací funkce (8)

Narozeninový paradox

Otázka: Jak velká musí být množina náhodných zpráv, aby v ní s nezanedbatelnou pravděpodobností existovaly dvě různé zprávy se stejnou haší?

Narozeninový paradox říká, že pro n-bitovou hašovací funkci nastává kolize s cca 50% pravděpodobností v množině $2^{\frac{n}{2}}$ zpráv, namísto očekávaných 2^{n-1}.

Tvrzení

Mějme množinu M n různých koulí a proveďme výběr k koulí po jedné s vracením do množiny M. Potom pravděpodobnost, že vybereme některou kouli dvakrát nebo vícekrát je

$$P(n,k) = 1 - \frac{n(n-1)\cdots(n-k+1)}{n^k}.$$

Pro $k = O(n^{\frac{1}{2}})$ a n velké je $P(n,k) \approx 1 - \exp(\frac{-k^2}{2n})$.

Hašovací funkce (9)

Důsledek

Pro m velké se ve výběru $k = (2n \ln 2)^{\frac{1}{2}} \approx n^{\frac{1}{2}}$ prvků z M s cca 50% pravděpodobností naleznou dva prvky shodné. Obecně pro hašovací funkce s n bitovým hašovacím kódem postačí zhašovat $2^{\frac{n}{2}}$ zpráv, abychom přibližně s 50% pravděpodobností našli kolizi.

Platí: $P(365, 23) = 0.507. \Rightarrow$ 23 náhodně vybraných lidí postačí k tomu, aby se mezi nimi s cca 50% pravděpodobností našla dvojice, slavící narozeniny tentýž den. U skupiny 30 lidí je $P(365, 30) = 0.706$.

Paradox, protože obvykle to vnímáme ve smyslu "kolik lidí je potřeba, aby se k danému člověku našel jiný, slavící narozeniny ve stejný den". V této podbízející se interpretaci hledáme jedny konkrétní narozeniny, nikoli "jakékoliv shodné" narozeniny. Oba přístupy odráží rozdíl mezi kolizí 1.řádu (libovolní 2 lidé) a 2. řádu (nalezení 2. člověka k danému).

Hašovací funkce (10)

Konstrukce moderních hašovacích funkcí

- U moderních hašovacích funkcí může být zpráva velmi dlouhá, například $d = 2^{64} - 1$ bitů $\Rightarrow$
- zpracovávání po částech, ne najednou. Také v komunikacích zprávu dostáváme po částech a nemůžeme ji z paměťových důvodů ukládat celou pro jednorázové zhašování $\Rightarrow$
- nutnost hašování zprávy po blocích a sekvenční způsob $\Rightarrow$
- nutnost zarovnání vstupní zprávy na celistvý počet bloků před hašováním.
- Zarovnání musí být bezkolizní a také proto musí umožňovat jednoznačné odejmutí.
- Dále budeme předpokládat, že máme hašovací funkci o n-bitovém hašovacím kódu a zprávu M zarovnanou na m-bitové bloky $M_1, \cdots, M_N$.

Hašovací funkce (11)

Zarovnání

- Zarovnání musí být takové, aby umožňovalo jednoznačné odejmutí, jinak by vznikaly jednoduché kolize.
- Například při doplnění zprávy nulovými bity by nešlo rozeznat, kolik jich bylo doplněno, a zda některé z nich nejsou platnými bity zprávy, pokud by zpráva nulovými bity končila.

Příklad: Doplněná zpráva

```
10111100110101000000000000000000000000000000
```

by byla doplněním zpráv

```
10111100110101
101111001101010
1011110011010100
10111100110101000
...
10111100110101000000000000000000000000000000
```

Všechny zprávy by vedly ke kolizi, měly by stejný hašový kód, nastala by dokonce vícenásobná kolize.

Hašovací funkce (12)

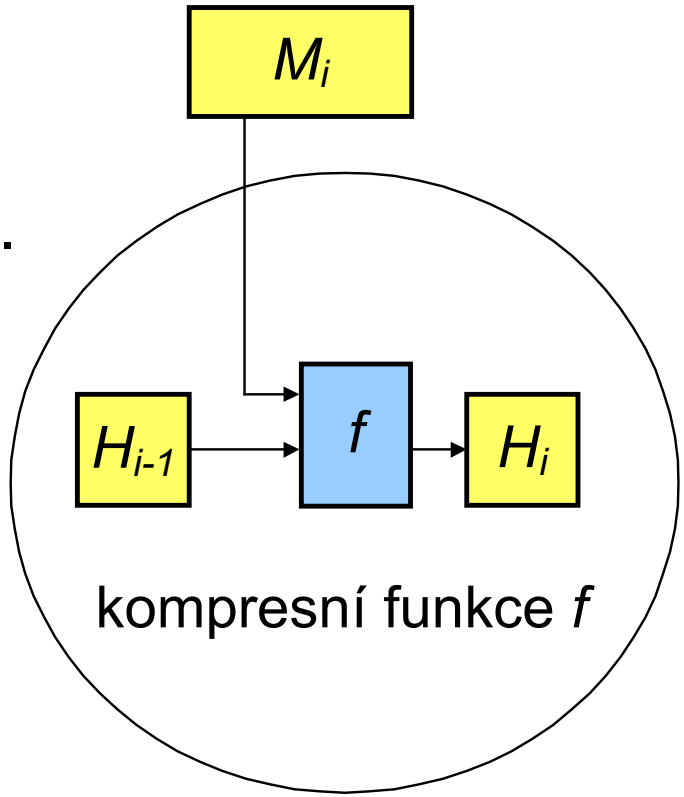
Zarovnání u nových hašovacích funkcí se definuje jako doplnění bitem 1 a poté potřebným počtem bitů 0 $\Rightarrow$ jednoznačné odejmutí doplňku.

Damgard-Merklovo zesílení – doplnění o délku původní zprávy

- Doplnění ilustruje obrázek (viz dále) pro blok délky $n = 512$, který je použit u MD4, MD5, SHA-0, SHA-1, SHA-256.
- Zpráva M je nejprve doplněna bitem 1 $\Rightarrow$ bity 0 (může jich být 1 — 512) tak, aby do celistvého násobku 512 bitů zbývalo ještě 64 bitů.
- 64 bitů je vyplněno 64 bitovou hodnotou # bitů původní zprávy M.
- Délka zprávy je součástí hašovacího procesu. 64 bitů na délku zprávy umožňuje hašovat zprávy až do délky $d = 2^{64} - 1$ bitů.
- Začleněním informace o délce původní zprávy eliminujeme případné útoky.

Hašovací funkce (13)

- Současné prakticky používané hašovací funkce používají Damgard-Merklův (DM) princip iterativní hašovací funkce s využitím kompresní funkce f.
- f zpracovává aktuální blok zprávy M_i a výsledek je určitá hodnota, tzv. kontext a označuje H_i.
- Hodnota H_i nutně tvoří vstup do f v dalším kroku. f má tedy dva vstupy: H_{i-1} a aktuální M_i. Výstupem je nové H_i.
- Damgard-Merklova konstrukce
  Tato iterativní konstrukce je popsána vzorcem
  $H_i = f(H_{i-1}, M_i)$,
  $H_0 = \text{IV}$,
  která se nazývá Damgard-Merklova metoda.

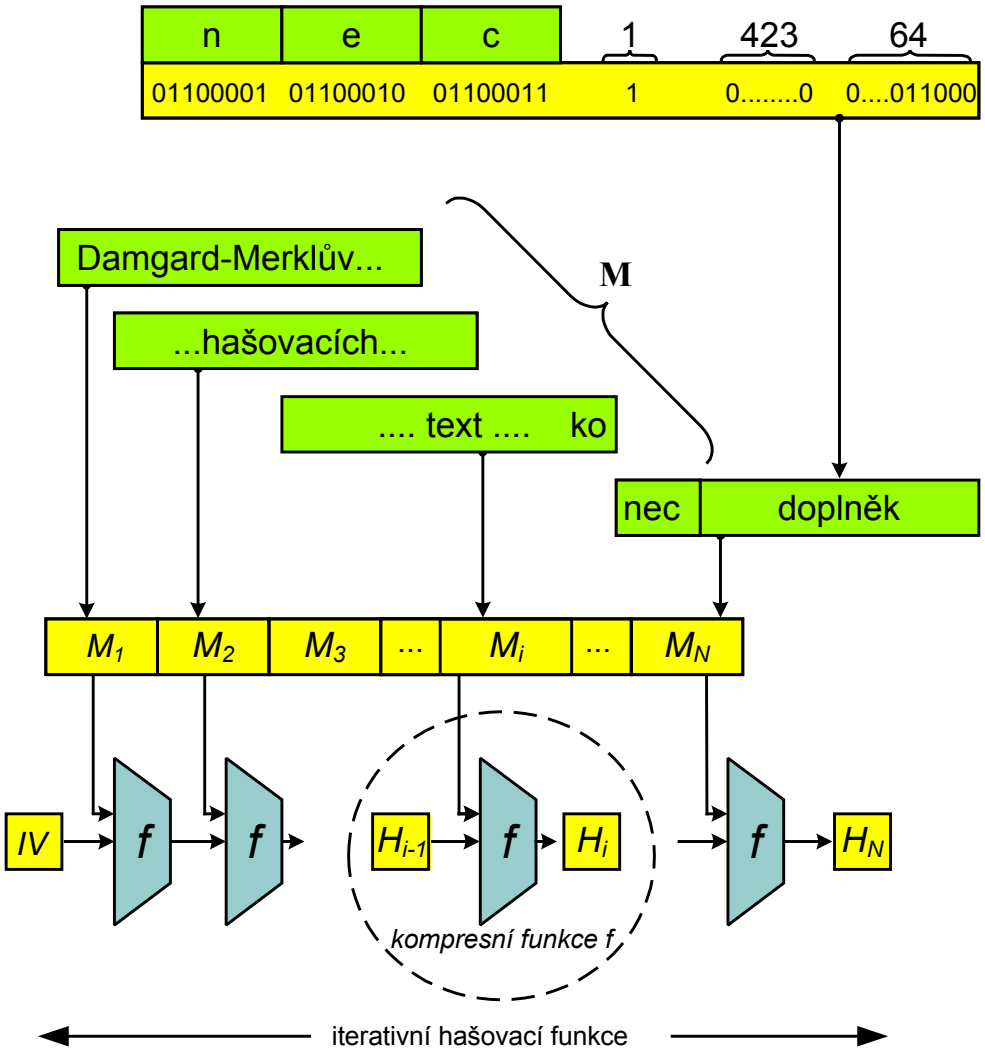


Hašovací funkce (14)

- Hašování probíhá postupně po jednotlivých blocích M_i v cyklu podle i od 1 do N.
- f v i-tém kroku ($i = 1, \ldots, N$) zpracuje vždy dané H_{i-1} a blok M_i na nové H_i. Šíře H_i je většinou stejná jako šíře výstupního kódu.
- Počáteční hodnota kontextu H_0 se nazývá inicializační hodnota – IV. Je dodefinována jako konstanta daná v popisu každé iterativní hašovací funkce.
- f zpracovává širší vstup (H_{i-1}, M_i) na mnohem kratší $H_i \Rightarrow f$ je skutečně kompresní.
- Blok M_i se promítne do H_i, ale současně dochází ke ztrátě informace (šířka kontextu $H_0, H_1, \ldots, H_i, \ldots$ zůstává stále stejná).
- Kontextem bývá obvykle několik 16b nebo 32b slov.
- Po zhašování posledního bloku M_N dostáváme H_N, z něhož bereme buď celou délku, nebo část jako výslednou haš. (MD5 má šířku kontextu 128 bitů a výslednou haš tvoří všech 128 bitů H_N).

Hašovací funkce (15)

Doplňování, kompresní funkce a iterativní hašovací funkce



Hašovací funkce (16)

Kolize kompresní funkce

Kolize kompresní funkce f spočívá v nalezení inicializační hodnoty H a dvou různých bloků B_1 a B_2 tak, že $f(H, B_1) = f(H, B_2)$.

Bezpečnost Damgard-Merklovy konstrukce

- Pokud je hašovací funkce bezkolizní, vyplývá odtud i bezkoliznost kompresní funkce.
- Důkaz je triviální pro zprávu tvořící jeden blok, ale je-li kompresní funkce bezkolizní, obecně odtud nevyplývá, že je bezkolizní i hašovací funkce.
- U DM konstrukce to bylo dokázáno, že pokud je kompresní funkce bezkolizní $\Rightarrow$ bezkolizní také iterovaná hašovací funkce, konstruovaná z ní výše uvedeným postupem.
- Tato vlastnost DM konstrukce je bezpečnostním základem všech moderních hašovacích funkcí $\Rightarrow$ je možné se soustředit na nalezení kvalitní kompresní funkce.

Hašovací funkce (17)

Konstrukce kompresní funkce

- Kompresní funkce musí být velmi robustní, aby zajistila dokonalé promíchání bitů zprávy a jednocestnost.
- Konstruovat kompresní funkce na bázi blokových šifer splňuje požadavek na jednocestnost a požadavek na jejich chování jako náhodné orákulum.
- Kvalitní bloková šifra $E_k(x)$ je jednosměrná vzhledem k pevnému klíči $k \Rightarrow$ při znalosti jakékoliv množiny vstupů-výstupů, tj. OT-ŠT (x, y), nemůžeme odtud určit (díky složitosti) klíč k.
- Přesněji, pro každé x je funkce $k \longrightarrow E_k(x)$ jednocestná.
- Odtud vyplývá možnost konstrukce kompresní funkce:

$$H_i = f(H_{i-1}, M_i,) = E_{M_i}(H_{i-1}).$$

- Vezměme hašování krátké zprávy, která i se zarovnáním tvoří 1 blok. Máme $H_1 = E_{M_1}(\mathrm{IV}) \Rightarrow$ z hodnoty hašového kódu H_1 nejsme schopni určit M_1, čili máme zaručenu vlastnost jednocestnosti.

Hašovací funkce (18)

Davies-Meyerova konstrukce kompresní funkce

- Davies-Meyerova konstrukce kompresní funkce pak zesiluje vlastnost jednocestnosti ještě přičtením (XOR) vzoru před výstupem:

$$H_i = f(H_{i-1}, M_i) = E_{M_i}(H_{i-1}) \oplus H_{i-1}.$$

- Výstup je zde tedy navíc maskován vstupem, což ještě více ztěžuje případný zpětný chod.
- Blok zprávy M_i je obvykle větší než klíče blokových šifer $\Rightarrow$ se aplikuje bloková šifra několikrát za sebou – v rundách.
- Rundovní klíč je postupně čerpán z bloku M_i.
- Na schématu MD5 vidíme způsob, jakým je blok zprávy M_i v blokové šifře v 16 rundách po částech používán jako rundovní klíč.
- Navíc se tento stavební blok ještě 4 $\times$ opakuje $\Rightarrow$ každá část bloku M_i se projeví na místě klíče dokonce 4 $\times$ (paradoxně se této fázi někdy říká příprava klíče).

Hašovací funkce (19)

Kompresní a hašovací funkce MD5

- Kontext H_i (128 b) tvoří 4 32b slova A, B, C a D (obrázek). M_i je 512b blok zprávy a je rozdělen na 16 32b slov $m_0, m_1, \ldots, m_{15}$.
- Ze schématu MD5 vidíme, že v kompresní funkci se kontext "zašifruje" vždy jedním 32b slovem m_i. Na místě dílčí funkce F v obrázku se po 16 rundách střídají 4 různé (nelineární i lineární) funkce (F, G, H, I) a v každé rundě se využívá jiná konstanta K_i.
- Tato posloupnost je opakována 4x za sebou v různých permutacích m_i z M_i. Po 64 rundách je přičten původní kontext H_{i-1} k výsledku podle Davies-Meyerovy konstrukce (XOR je nahrazen arit. součtem modulo 2^{32}). Tak vznikne nový kontext H_i.
- Pokud by zpráva M měla jen 1 blok, byl by kontext (A, B, C, D) celkovým výsledkem, jinak se pokračuje stejným způsobem v hašování 2. bloku zprávy M_2 jakoby se vstupní hodnotou H_1. Po zpracování bloku M_N máme v registrech výsledný 128b haš H_N.

Hašovací funkce (20)

Permutace m_i v bloku M_i pro rundy 16 až 63

$$\begin{aligned} p(i) &= |1 + 5i|_{16} \\ p(i) &= |5 + 3i|_{16} \\ p(i) &= |7i|_{16} \end{aligned}$$

Pro hodnoty $i = 0, 1, 2, \ldots, 15 \Rightarrow m_i \leftarrow m_{p(i)}$.

Logické funkce F, G, H, I

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

- Hodnota s, která udává počet rotací doleva je číslo rundy.
- Všechny modulární operace součtu jsou vykonáváné modulo 2^{32}.
- Pro každou rundu je dána konstanta K_i , pro $i = 1, 2, \ldots, 64$.

Hašovací funkce (21)

Algoritmus SHA-1

- Algoritmus SHA (Secure Hash Algorithm) byl zavedený NIST a publikovaný v normě FIPS 180 (1993). Revidovaná verze SHA–1 v normě FIPS–180–1 (1995).
- SHA–1 umožňuje zpracovat zprávu M s maximální délkou $2^{64} - 1$ bitů a představuje výstupní hašoví kód s délkou 160b. Vstupní zpráva M je zpracována po blocích z velkosí 512b.
- Z obrázku můžeme vidět, že způsob předzpracování zprávy je téměř shodný se způsobem zpracování zprávy u MD5. Jediný rozdíl je v šířce kontextu, tj. ve velikosti hašového kódu, který je u SHA-1 160b.
- Na rozdíl od MD5 má SHA-1 o jeden vstupně/výstupní 32b registr E kompresní funkce f navíc.
- Na schématu SHA-1 vidíme zapojení kompresní funkce SHA-1, která zpracovává jeden 512b blok M_i zprávy M.

Hašovací funkce (22)

- Blok zprávy M_i je zpracováván v 4×20 rundách. 160 bitový kontext (opět začínáme s inicializačním vektorem IV) je postupně "zašifrováván" 32b slovem m_i pro rundy $0, 1, \ldots, 15$ a pro rundy $16, 18, \ldots, 79$ 32b slovem w_i, pro které platí:

$$w_i = <<<_1 (w_{i-16} \oplus w_{i-14} \oplus w_{i-8} \oplus w_{i-3}), \text{ kde } i = 16, 17, \ldots, 79.$$

- Znovu na místě funkce F v obrázku se po 20 rundách střídají 4 různé funkce (F, G, H, I) a v každé rundě se využívá jiná konstanta K_i. Pro výpočet hodnot funkcí F, G, H, I platí:

Logické funkce F, G, H, I

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

Hašovací funkce (23)

- Po 80 rundách je i u této kompresní funkce přičten modulo 2^{32} původní kontext H_{i-1} k výsledku podle Davies-Meyerovy konstrukce. Přičitání je prováděno po 32b slovech.
- Po zpracování bloku M_N máme v registrech A, B, C, D, E, F výslednou 160b haš H_N.
- Pro konstantu K_i platí:

Runda	Hexadecimální K_i	Celočíselná část
$0 \leq i \leq 19$	K_i=5A927999	$(2^{30} \times \sqrt{2})$
$20 \leq i \leq 39$	K_i=6ED9EBA1	$(2^{30} \times \sqrt{3})$
$40 \leq i \leq 59$	K_i=8F1BBCDC	$(2^{30} \times \sqrt{5})$
$60 \leq i \leq 79$	K_i=CA62C1D6	$(2^{30} \times \sqrt{10})$

- Inicializační vektor IV má hodnoty (hexadecimálně):
  A= 67452301, B = EFCDAB89, C= 98BADCFE
  D= 10325476, E= C3D2E1F0

Hašovací funkce (24)

- Inovace standardu FIPS 180-1 na FIPS 180-2 byly zavedeny 3 nové hašovací algoritmy: SHA-256, SHA-384 a SHA-512.

Základní vlastnosti hašovacích funkcí SHA-x

	SHA-1	SHA-256	SHA-384	SHA-512
Délka haš. kódu	160	256	384	512
Délka zprávy	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Velikost bloku	512	512	1024	1024
Velikost slova	32	32	64	84
# rund f	80	80	80	80
Bezpečnost v bitech	80	128	192	256

- Nejvýznamnější rozdíly jsou v délce hašového kódu, který určuje odolnost hašového kódu vůči nalezení kolizí 1. a 2. řádů.
- Na druhé straně struktura hašovacích funkcí (kompresních funkcí) je téměř stejná.

Hašovací funkce (25)

Kryptografické využití hašovacích funkcí

- Kontrola integrity (kontrola shodnosti velkých souborů dat).
- Automatické dešifrování (souboru, disku apod.).
- Ukládání a kontrola přihlašovacích hesel. Pro vyloučení slovníkových útoků se používá ještě metoda solení. S otiskem hesla se vygeneruje náhodný řetězec "sůl", který je dohromady hašován s heslem. Databáze obsahuje dvojici: (sůl, haš(heslo, sůl)).
- Prokazování autorství.
- Jednoznačná identifikace dat (jednoznačná reprezentace vzoru, digitální otisk dat, jednoznačný identifikátor dat, to vše zejména pro digitální podpisy).
- Prokazování znalosti.
- Autentizace původu dat.
- Nepadělatelná kontrola integrity.
- Pseudonáhodné generátory, derivace klíčů.

Hašovací funkce (26)

Algoritmus HMAC

- Klíčované hašové autentizační kódy zpráv HMAC zpracovávají hašováním nejen zprávu M, ale spolu s ní i nějaký tajný klíč K.
- Jsou proto podobné autentizačnímu kódu zprávy MAC, ale místo blokové šifry využívají hašovací funkci.
- Používají se jak k nepadělatelnému zabezpečení zpráv, tak k autentizaci (prokazováním znalosti tajného klíče K).
- HMAC je obecná konstrukce, která využívá obecnou hašovací funkci. Podle toho, jakou hašovací funkci používá konkrétně, se označuje výsledek, například HMAC-SHA-1(M, K) používá SHA-1, M je zpráva a K je tajný klíč.
- HMAC je definován ve standardu FIPS-PUB-198. Definice závisí na tom, kolik bajtů má blok kompresní funkce. Například u SHA-1 je 64B (512b), u SHA- 384 a SHA-512 to je 128B (1024b).

Hašovací funkce (27)

Algoritmus HMAC

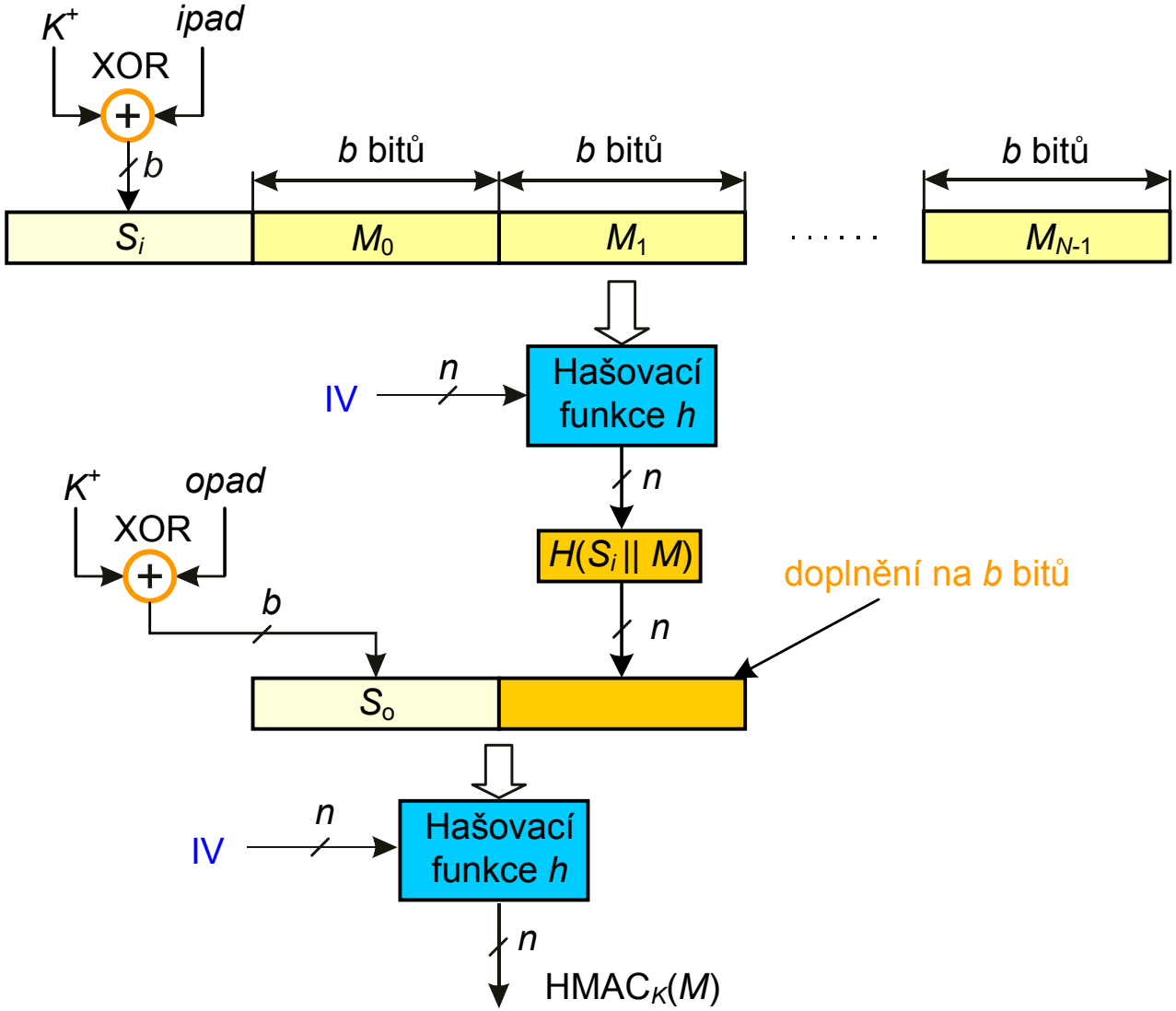
- Definujeme konstantní řetězce *ipad* jako řetězec $b/8$ bajtů s hodnotou `0x36 (0011 0110)` a *opad* jako řetězec $b/8$ bajtů s hodnotou `0x5C (0101 1100)`.
- Klíč K ($\log_2 K \geq n$, kde n je délka hašového kódu) doplníme bity 0 vlevo od MSB bitu klíče do délky b-bitů a označíme ho K^+.
- Definujeme hodnotu $HMAC_K(M)$ jako

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

  kde $\|$ označuje zřetězení.
- Operace "doplnění na b bitů" (viz. obrázek) je prováděna stejným způsobem, jako doplnění zprávy u hašovací funkce do celého bloku.
- Doplnění bude vždy jenom do 1 bloku, protože při použití MD5 nebo SHA-x platí: *délka hašového kódu* $<$ *velikost bloku* -64 [b].

Hašovací funkce (28)

Struktura algoritmu HMAC



Hašovací funkce (29)

Nepadělatelný integritní kód, autentizace původu dat

- Zabezpečovací kód $HMAC_K(M)$ pokud je připojen za zprávu M, detekuje neúmyslnou chybu při jejím přenosu.
- Zabraňuje útočníkovi změnit zprávu a současně změnit HMAC, protože bez znalosti klíče K nelze nový HMAC vypočítat $\Rightarrow$
- HMAC – nepadělatelný integritní kód (neposkytuje samotná haš).
- Pro komunikujícího partnera je správný HMAC autentizací původu dat, protože odesílatel musel znát hodnotu tajného klíče K.

Průkaz znalosti při autentizaci entit

- HMAC může být použit jako průkaz znalosti tajného sdíleného tajemství klíč K při autentizaci entit.
- Dotazovatel odešle náhodnou výzvu (řetězec) *challenge* a od prokazovatele obdrží odpověď $response = HMAC_K(challenge) \Rightarrow$
- prokazovatel zná tajný klíč K. Útočník na komunikačním kanálu z hodnoty response klíč K nemůže odvodit.