České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra telekomunikační techniky

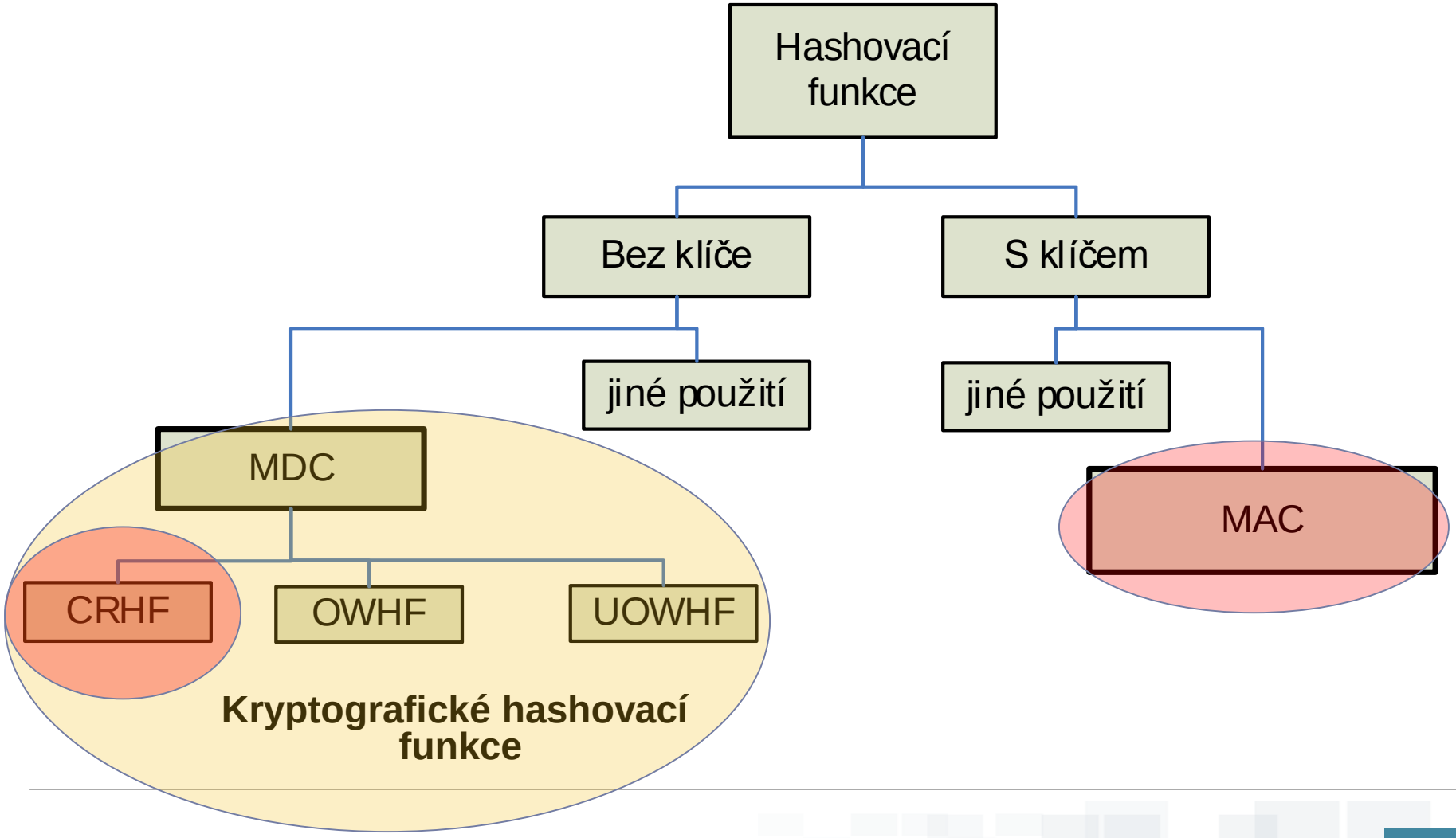
Hashovací funkce

Ing. Tomáš Vaněk, Ph.D.

Obsah

- Obecné rozdělení a principy
- MD5, SHA-1 a nalezení kolizí
- SHA-2
  - SHA-224, SHA-256, SHA-384, SHA-512
  - SHA-512/224, SHA-512/256
- SHA-3
- Whirlpool
- CBC-MAC / HMAC

Rozdělení hashovacích funkcí



Rozdělení hashovacích funkcí

Obecně lze hashovací funkce rozdělit na dvě třídy:

- hashovací funkce bez klíče – mají jediný vstupní parametr – zprávu *M*
- hashovací funkce s klíčem – dva nezávislé vstupy – zprávu *M* a klíč *K*

Hashovací funkce je funkce *h*, která musí splňovat **minimálně** tyto požadavky:

1. **komprese** — funkce *H* mapuje libovolně konečně velký vstup *x*, na výstup *h(x)* pevné délky *n*
2. **jednoduchost výpočtu** - pro danou hashovací funkci *H* a vstup *x*, je jednoduché spočítat *h(x)*

MAC - Message Authentication Code

- skupina funkcí parametrizovaných (tajným) klíčem *k*
- zajišťuje integritu zprávy a zároveň zaručuje i zdroj zprávy (autentizace)
- MAC musí mít následující vlastnosti:

1. **komprese** — funkce h_k mapuje libovolně velký konečný vstup *x*, na výstup $h_k(x)$ pevné délky *n*
2. **jednoduchost výpočtu** - pro danou hashovací funkci *h*, klíč *k* a vstup *x*, je jednoduché spočítat $h(x)$. Výsledek se nazývá MAC nebo MAC-hodnota
3. **výpočetní odolnost** – pro jeden nebo více párů $(x_i; h_k(x_i))$ je výpočetně nemožné spočítat jinou dvojici $(x; h_k(x))$ pro $x \neq x_i$ (zahrnující i možnost $h_k(x)=h_k(x_i)$ pro některé *i*) – tzn. z $h_k(x)$ není možné získat k (označuje se jako „**key non-recovery**“)

MDC – Modification Detection Code

- někdy též MIC (Message Integrity Check)
- účelem je poskytnout reprezentativní vzorek (hash) zprávy, který budu vyhovovat dalším dodatečným podmínkám - viz. dále
- zajišťuje integritu dat

OWHF – One-way Hash Function

- je obtížné najít vstup jež bude mít nějaký předem daný otisk (hash)

CRHF – Collision Resistant Hash Function

- je obtížné nalezení dvou různých vstupních textů se stejným otiskem (hashem)

UOWHF – Universal One-way Hash Function

OWHF, CRHF, UOWHF – kryptografické hashovací funkce

Základní vlastnosti dobré hashovací funkce

1. Odolnost vůči získání předlohy (Preimage resistance)
   Pro všechny výstupy z hashovací funkce je výpočetně nemožné získat vstup, kterému odpovídá daný otisk. Neboli nelze najít předobraz *x* takový, že h(*x*)=y, když známe pouze *h* a *y* a neznáme *x*.

Základní vlastnosti dobré hashovací funkce

2. Odolnost vůči získání jiné předlohy (2nd preimage resistance) – je výpočetně nemožné nalézt ke vstupu *x* druhý vstup x’ takový, že x’≠ x a zároveň h(*x*’)=h(*x*).

3. Odolnost vůči nalezení kolize (Collision resistance) – je výpočetně nemožné systematicky vytvářet dva libovolné různé vstupní texty *x* a *x’* pro které platí, že x ≠ x’ a zároveň h(x)=h(x’)

Slova „jednoduché“ a „výpočetně nemožné“ v předchozích definicích nemají přesnou a striktní definici.

Odolnost neznamená, že kolize nebudou existovat (vždy budou, z principu musí!), ale o obtížnost jejich nalezení.

Narozeninový paradox

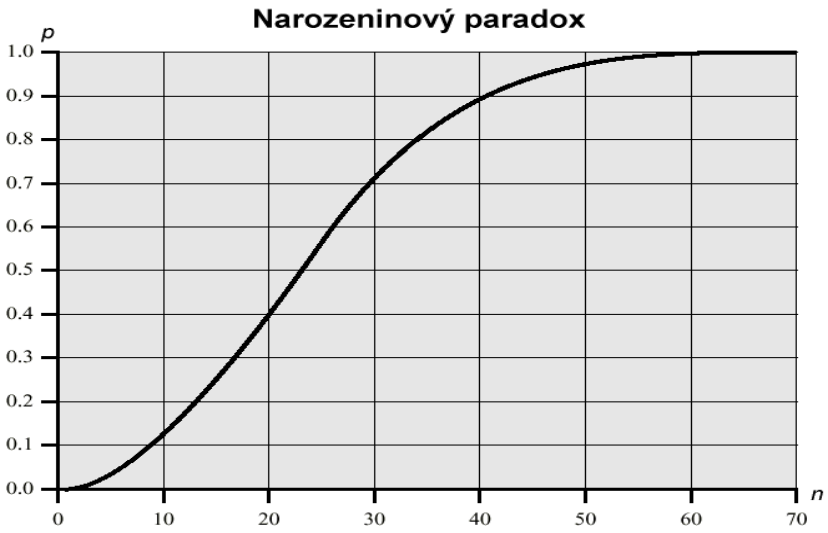
Předpoklad: Nechť M je množina o *n* různých prvcích

Vybírejme náhodně *k* prvků z množiny M. Po každé volbě zaznamenejme výsledek a vraťme prvek zpět.

Tvrzení: Po výběru $k=n^{1/2}$ prvků je pravděpodobnost výběru již jednou vybraného prvku cca 50%.

P(365,23)=0,507

P(365,30)=0,706



Terminologie - alternativní pojmy

odolnost vůči získání předlohy / preimage resistance

=

jednocestná / jednosměrná / one-way

odolnost vůči získání jiné předlohy / 2nd preimage resistance

=

slabá odolnost proti kolizím / odolnost proti kolizím 2. řádu

odolnost vůči nalezení kolize / collision resistance

=

silná odolnost proti kolizím / odolnost proti kolizím 1. řádu

Jednocestná funkce (One-Way Function)

- funkce *f* splňují tyto vlastnosti:
  1) pro všechna *x* je f(x) „snadno“ spočitatelná
  2) z pouhé znalosti *f(x)* je „výpočetně nemožné“ najít *x*

Rozdíly OWF a OWHF:

- Není požadavek na 2nd preimage resistance
- Není požadavek na kompresi vstupu
- Není požadavek na konečnou délku vstupní posloupnosti

Pozn.: Dnes není známa žádná dokazatelně jednocestná funkce. Existují pouze kandidáti na OWF v tom smyslu, že jejich vyřešení vede na NP-úplný problém.
Důkaz existence OWF by byl zároveň důkazem, že P ≠ NP.

MDC – OWHF (One-way Hash Function)

Podmínky:

1. komprese
2. jednoduchost výpočtu
3. odolnost vůči získání předlohy
4. odolnost vůči získání jiné předlohy

Lze nalézt kolizi po provedení $2^{n/2}$ operací , kde n je délka výstupu (paradox společných narozenin)

MDC – CRHF (Collision Resistant Hash Function)

Podmínky:

1. komprese
2. jednoduchost výpočtu
3. odolnost vůči získání předlohy
4. odolnost vůči získání jiné předlohy
5. odolnost vůči nalezení kolize

- CRHF je vždy OWHF

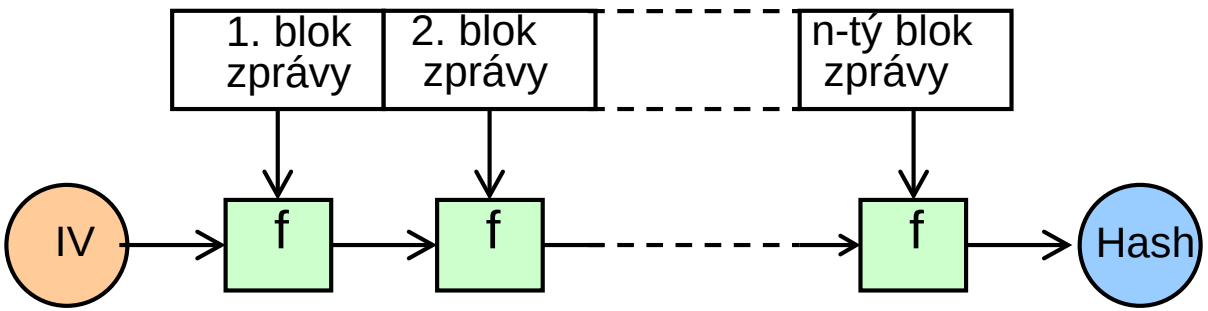
Další obvyklé požadavky na HF zahrnují:

- nekorelovatelnost vstupních a výstupních bitů
  - znemožní statistickou kryptoanalýzu

- odolnost vůči skoro-kolizím (*Near-collision resistance*)
  - je obtížné nalézt x a y taková, že $x \neq y$ a zároveň $H(x)$ a $H(y)$ se liší jen v malém počtu bitů.

- lokální odolnost vůči získání předlohy
  - je obtížné najít i jen část vstupu x ze znalosti $H(x)$

Merkle-Damgårdova struktura hashovací funkce



f – kompresní funkce

IV – inicializační vektor

Hash – výsledný otisk zprávy

- bloky *f* provádějí kompresní funkci
- toto schéma využívá naprostá **většina** moderních hashovacích funkcí – MD-5, SHA-1, RIPEMD-160
- vstup musí být doplněn na celistvý násobek délky bloků
- musí být jednoznačně určitelné kolik se doplnilo
  - jinak by jednoduše vznikala řada kolizí - zpráva doplněná na …0110000 by mohla vzniknout ze zprávy …011, …0110, …01100,…
- Merkle-Damgårdovo zesílení (doplnění výplně posledního bloku o délku zprávy – zvyšuje odolnost vůči generování kolizí)

Merkle-Damgårdova metoda

- v roce 1989 ji nezávisle na sobě navrhli Merkl a Damgård
- nazývá se Merklova meta metoda nebo Merkle-Damgårdova
- je matematicky dokázáno, že pokud je kompresní funkce bezkolizní, bude bezkolizní i iterovaná hashovací funkce realizovaná pomocí Merkle-Damgårdovy metody*
  - stačí „pouze“ najít vhodnou funkci *f*
- **Merkle-Damgård → MD(4,5)**

** Toto neplatí obecně!!!*

Požadavky na kompresní fci

Realizace kompresní funkce v ideálním světě

- **náhodné orákulum**

Náhodné orákulum je hypotetické zařízení, které na každý vstup odpovídá náhodně vybraným výstupem ze svého oboru hodnot, navíc ale s tou podmínkou, že na stejně zadané vstupy odpovídá vždy stejnými odpověďmi.

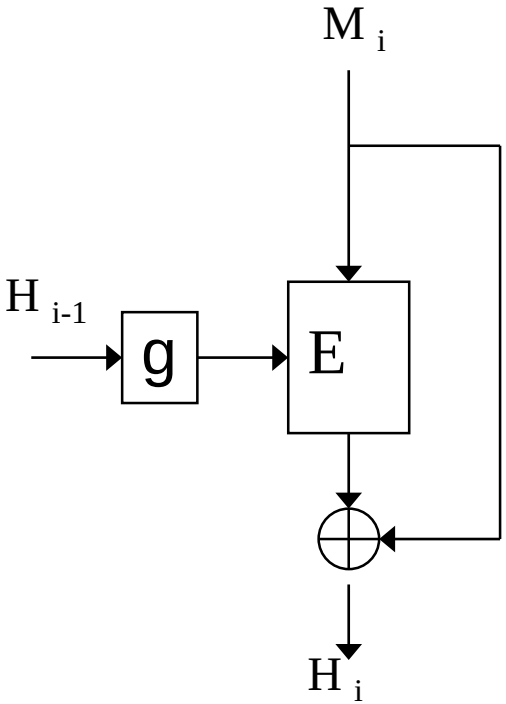
Realizace kompresní funkce v reálném světě

- vhodnou blokovou šifrou
- nelineární funkcí

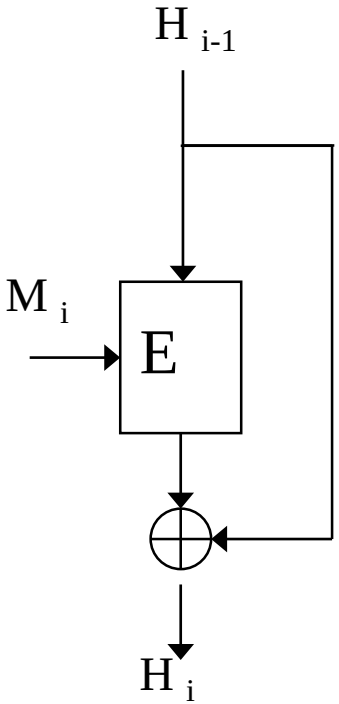
Cílem návrhu kompresní funkce je, aby se taková funkce chovala podobně jako náhodné orákulum.

Realizace kompresní funkce HF pomocí blokové šifry

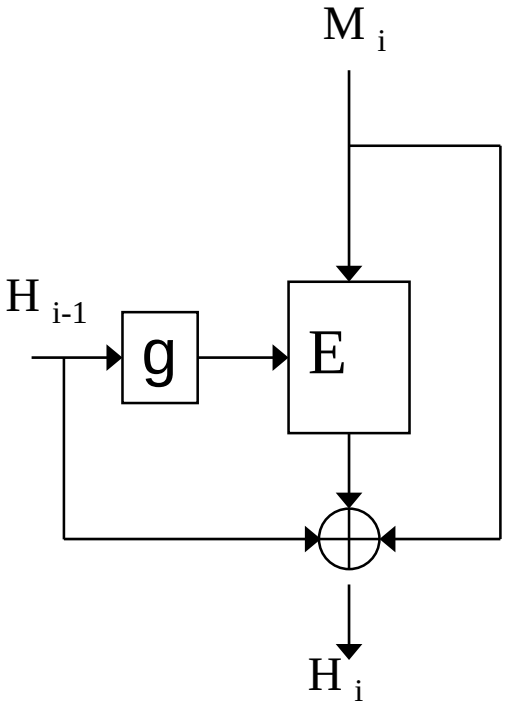
Matyas - Meyer - Oseas



Davies - Mayer



Miyaguchi - Preneel



g – funkce, která upraví hash z předchozího kroku tak, aby ho bylo možné použít jako heslo pro blokovou šifru E (doplnění, vhodná konverze)

Hirose

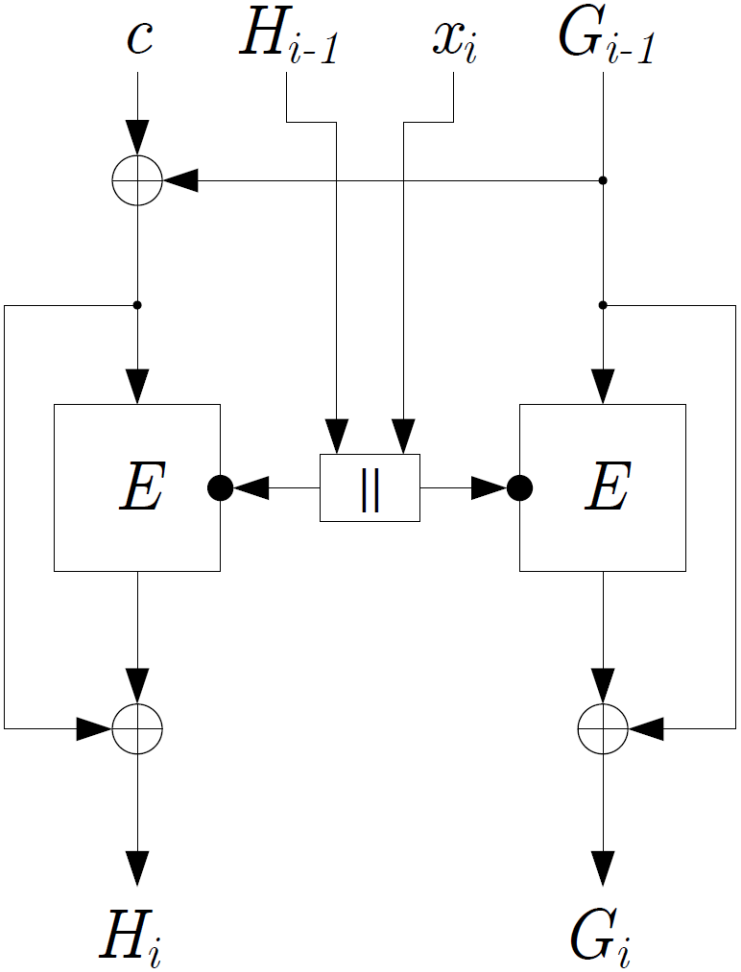
- 2006 - double-block-length OWCF
- vstup délky *n*
- výstup délky *2n*
- bloková šifra + permutace
- klíč *k* > blok *n*
- AES-192, AES-256

$G_i = E_{Ki}(G_{i-1}) \oplus G_{i-1}$

$H_i = E_{Ki}(p(G_{i-1})) \oplus p(G_{i-1})$.

$p(G_{i-1})$ je libovolná permutace s nějakým celým číslem na *n*-bitovém řetězci, typicky definované: $p(x) = x \oplus c$ pro libovolnou nenulovou konstantu *c*.

Šifrování je implementováno ve standardním Davies-Meyerově zapojení.



Hashovací funkce - přehled

MD2 – kompromitovaná, nepoužívá se

MD4 – kompromitovaná, nepoužívá se

MD5 – oblíbená, ale kompromitovaná funkce.

Od srpna 2004 je veřejně znám postup nalezení kolizí a to i pro málo odlišná vstupní data.

RIPEMD-128 – kompromitovaná, nepoužívá se

RIPEMD-160 – může být kompromitována

WHIRLPOOL – považuje se za bezpečnou, jádro modifikovaný AES

TIGER – považuje se za bezpečnou

GRINDAHL – mladá (03/2007), jádro z AES

Hashovací funkce - přehled

SHA-0 – kompromitována, nepoužívá se

SHA-1 – oblíbená, ale již kompromitovaná funkce. V únoru 2005 byl zveřejněn objev algoritmu, který umožňuje nalézt kolizi podstatně rychleji než hrubou silou.

únor 2017 – Google nalezl kolizi v SHA-1

SHA-2 – dosud považována za bezpečnou
– není to jedna hashovací funkce, ale více variant souhrnně označovaných jako SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512)

SHA-3 – standardizována 08/2015, nepoužívá se

MD-5

Vstup: řetězec proměnné délky (neomezeně dlouhé)
Výstup: pevná délka128 bitů

Počet rund: 4

Počet kroků v rundě: 16

Vstup je zpravován po úsecích délky 512 bitů

Zpráva je zarovnána tak, aby byla dělitelná 512.

Zarovnání:
- zpráva se doplní zprava o jeden bit s hodnotou „1“
- zpráva se doplní zprava bity s hodnotou „0“ až do délky o 64 bitů menší než je číslo dělitelné 512
- posledních 64 bitů obsahuje číslo reprezentující původní délku zprávy mod 2^{64}

MD-5

M_i – vstupní zpráva 32bitů

K_i – konstanta 32bitů

– mění se pro každou operaci

F – nelineární funkce

$<<<_s$ – rotace o *s* bitů doleva, hodnota *s* se mění v každé operaci

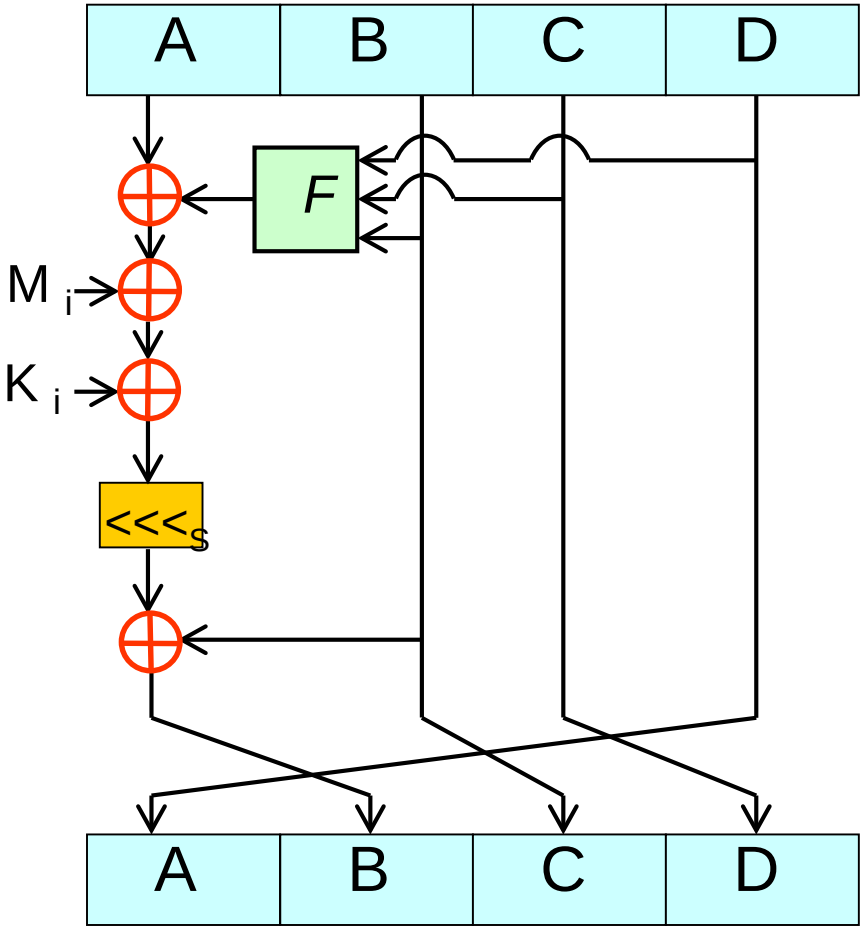
$\oplus$ – operace sčítání mod 2^{32}

A,B,C,D – pomocné registry

– na začátku konstanty

– po poslední rundě obsahují hash

– velikost 128 bitů (4x32)



Jeden krok algoritmu MD-5

MD-5 expanze bloku

- vstupní blok M_i má délku 512 bitů
- před vstupem do vlastního algoritmu je rozdělen na 32 bitové části (16 různých bloků)
- každý blok vstupuje do kompresní funkce čtyřikrát, pokaždé s jinou nelineární funkcí *f*
- celkem tedy 4 kola *16 kroků = 64 rund
- konstanta K_i je spočítána jako celá část z 4294967296 násobku absolutní hodnoty funkce sin(i), kde i je úhel v radiánech

MD-5 - nelineární funkce „F“

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$

$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$

$$H(X,Y,Z) = X \oplus Y \oplus Z$$

$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

$\oplus$ XOR

$\vee$ OR

$\wedge$ AND

$\neg$ NOT

- v každém kole probíhá jiná nelineární operace v bloku F

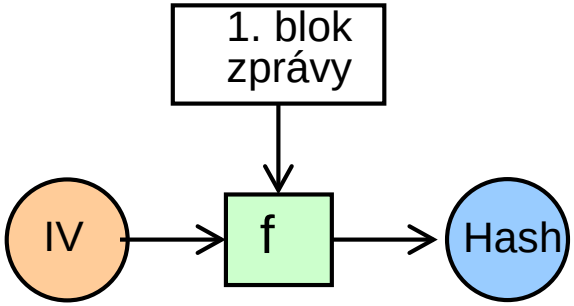
MD5 - počáteční hodnoty IV

- registr A: 0x01234567
- registr B: 0x89abcdef
- registr C: 0xfedcba98
- registr D: 0x76543210

Útoky na hashovací funkci MD5

Kde leží slabiny současných hashovacích funkcí ?

Předpoklad: hashovací funkce musí plnit svoji funkci a zachovávat si všechny vlastnosti i při hashování jediného bloku

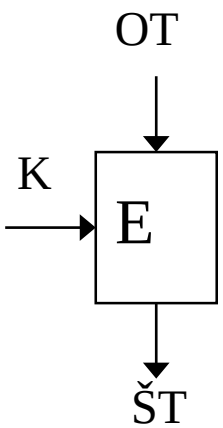


Důsledek: analýza se zjednoduší pouze na zkoumání kompresní funkce.

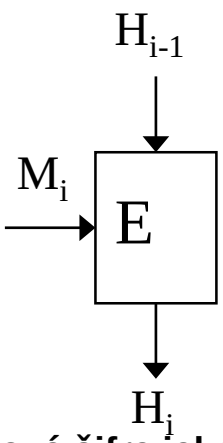
Bezpečnost HF závisí pouze na bezpečnosti *f*

Kde leží slabiny současných hashovacích funkcí ?

1) *f* je tvořena blokovou šifrou



Bloková šifra pro šifrování



Bloková šifra jako základ hashovací funkce

Žádná bloková šifra není připravena na stav, kdy útočník zná klíč ! Navíc klasickou blokovou šifru lze „přepnout“ do režimu dešifrování – ale my požadujeme jednosměrnost!

Klasické blokové šifry nemohou být současně bezpečné a efektivní funkce *f.*

Kde leží slabiny současných hashovacích funkcí ?

2) *f* je tvořena nelineárními funkcemi
- pouze slabá nelinearita

Historie prolomení MD-5

- v červenci 2004 neexistoval úspěšný útok na MD5
- MD5 byla považována za bezpečnou
- nebyla známa žádná metoda rychlejší než útok hrubou silou se složitostí 2^{64}
- projekt „Rainbow table“

Červenec 2004

- čínský kolektiv oznámil nalezení kolize

Duben 2006

- nalezení kolize na počítači s procesorem Pentium-M 1,6GHz trvá průměrně 31 sekund.
- zrychlení možné díky „tunelům“
  Program na hledání kolizí najdete na
  http://cryptography.hyperlink.cz/2004/kolize_hash.htm

Čínský útok

- V srpnu 2004 byly na konferenci Crypto 2004 prezentovány kolize hashovacích funkcí MD4, MD5, HAVAL-128 a RIPEMD
- demonstrována kolize dvou různých 1024bitových zpráv

Postup vygenerování kolize:

1) naleznou se dvě různé zprávy délky 512 bitů M1, M2 (Číňanům to trvalo cca 60min na multiprocesorovém serveru IBM p690 (32procesorů))

2) Nalezení dvou různých 512bitových půlzpráv N1, N2 (několik sekund) pro které platí, že že složené zprávy (M1, N1) a (M2, N2) mají stejnou hash .

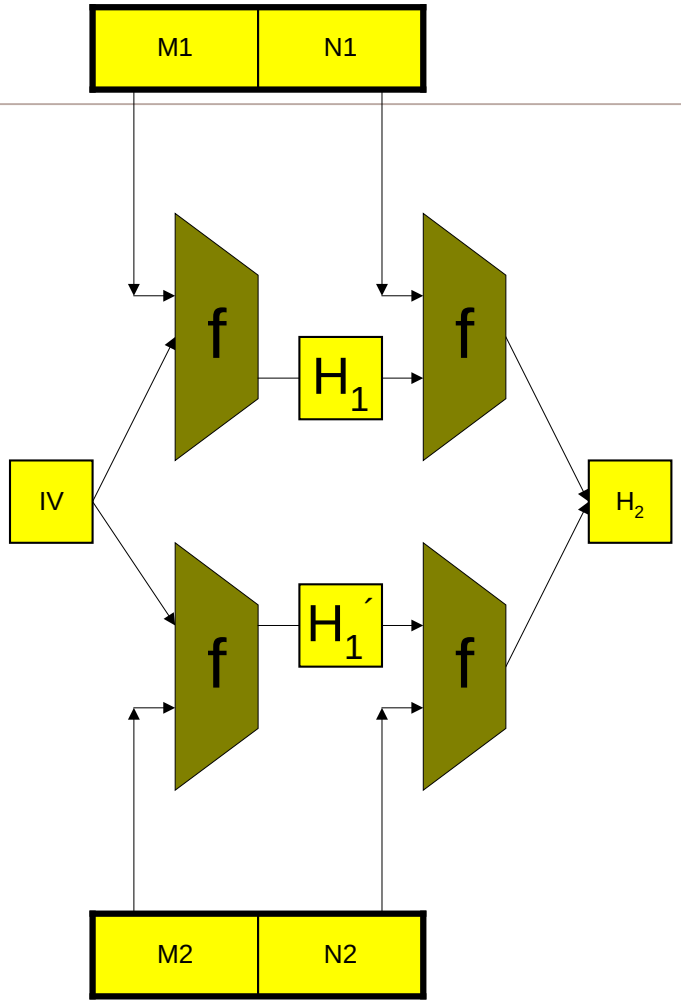
H(M1||N1)=H(M1||N2)

Čínský útok

- M_1 512 bitů
- N_1 512 bitů
- M_2 512 bitů
- N_1 512 bitů

Zprávy M_1 M_2 a N= N= se lišili jen velmi málo. $M_2 = M_1 + C$, $N_2 = N_1 - C$, kde C je 512 bitů velká vhodně zvolená konstanta která má pouze 3 bity nenulové.

Informace o konstrukci konstanty C bylo to nejdůležitější na co Čínané přišli.



Obrázek z prezentace V. Klímy – Hashovaci funkce a čínský útok

Příklad kolize

M1 = 2dd31d1 c4eee6c5 69a3d69 5cf9af98 87b5ca2f ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 5a417125 e8255108 9fc9cdf7 f2bd1dd9 5b3c3780

N1 = d11d0b96 9c7b41dc f497d8e4 d555655a c79a7335 cfdebf0 66f12930 8fb109d1 797f2775 eb5cd530 baade822 5c15cc79 ddcb74ed 6dd3c55f d80a9bb1 e3a7cc35

M2 = 2dd31d1 c4eee6c5 69a3d69 5cf9af98 87b5ca2f ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 5a41f125 e8255108 9fc9cdf7 72bd1dd9 5b3c3780

N2 = d11d0b96 9c7b41dc f497d8e4 d555655a 479a7335 cfdebf0 66f12930 8fb109d1 797f2775 eb5cd530 baade822 5c154c79 ddcb74ed 6dd3c55f 580a9bb1 e3a7cc35

Hash: 9603161f a30f9dbf 9f65ffbc f41fc7ef

Příklad kolize

- http://www.win.tue.nl/hashclash/SoftIntCodeSign/
- dva různé spustitelné soubory se stejným hashem
  - 1. program vytiskl „Hello world“
  - 2. program ve smyčce tiskl „Good bye world“
- každý program měl původně jiný hash
- nalezl se „ocásek“ (832B) takový, že po připojení k oběma programům mají tyto nyní stejný hash

Co je zatím bezpečné

- zatím nelze k dané zprávě vytvořit jinou, se stejnou haší (neumíme kolizi druhého řádu)
- zatím nejsou ohroženy minulé digitální podpisy pomocí MD5. Jsou ohroženy budoucí všude tam, kde útočník vytváří soubor k digitálnímu podpisu (nebezpečí vytvoření druhého falešného souboru).
- zatím nejsou ohroženy funkce HMAC a PRNG využívající slabé hašovací funkce

SHS – Secure Hash Standard

- množina bezpečných kryptografických hashovacích funkcí specifikovaných organizací NIST (National Institute of Standards nad Technology)
- NIST FIPS* 180-4
- specifikuje řadu hashovacích funkcí SHA
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
  - SHA-512/224
  - SHA-512/384

SHA-1

Vstup: řetězec proměnné délky (max. 2^{64} -1 bitů)
Výstup: pevná délka160 bitů

Počet rund: 4

Počet kroků v rundě: 20

Vstup je zpravován po úsecích délky 512 bitů

Zpráva je zarovnána tak, aby byla dělitelná 512.

Zarovnání:
- zpráva se doplní zprava o jeden bit s hodnotou „1“
- zpráva se doplní zprava bity s hodnotou „0“ až do délky o 64 bitů menší než je číslo dělitelné 512
- posledních 64 bitů obsahuje číslo reprezentující původní délku zprávy

SHA-1 expanze bloku

- vstupní blok M_i má délu 512 bitů
- před vstupem do vlastního algoritmu je rozdělen a expandován:

  1) rozdělení na 32 bitové části (16 různých bloků) označených $W_0 \ldots W_{15}$

  2) expanzní funkce $E\{0,1\}^{512} \rightarrow E\{0,1\}^{2560}$ těchto 16 částí rozšíří na 80 podle schématu

  $$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})<<<1$$

  $t = 16..79$

SHA-1

$<<<_5$ – rotace o 5

$<<<_{30}$ – rotace o 30

F – nelineární funkce mění se v každé rundě

$\oplus$ – operace sčítání mod 2^{32}

W_t – vstupní zpráva, 32b

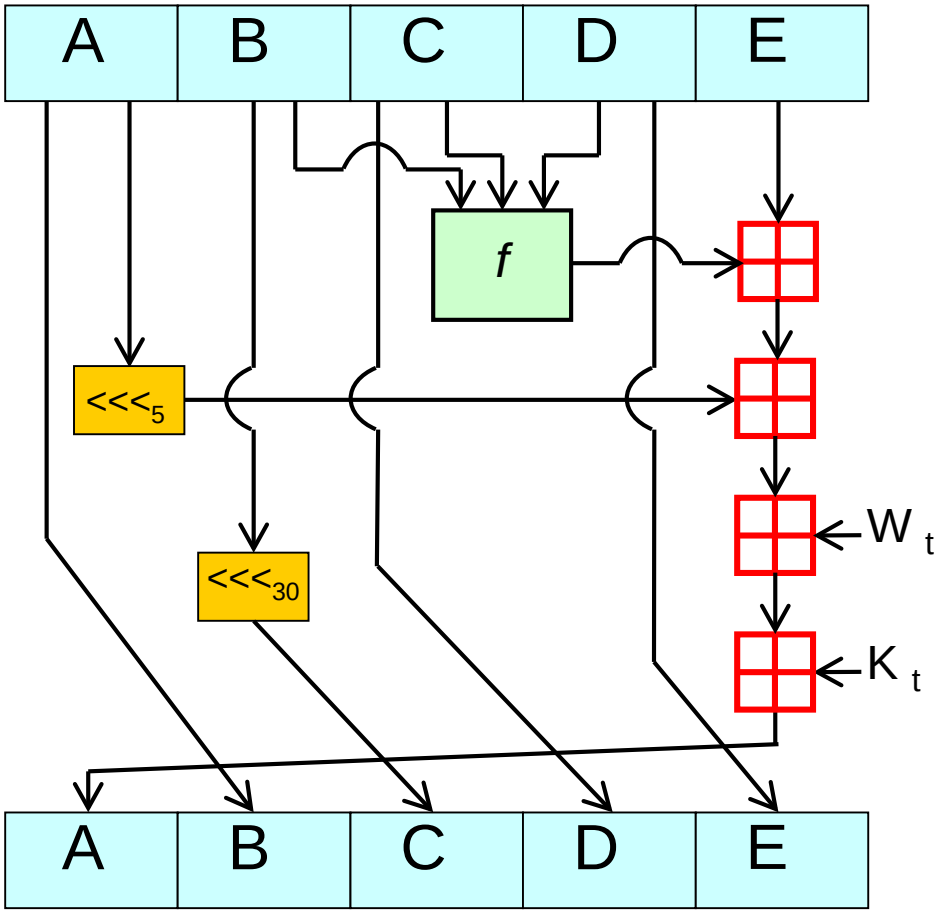
K_t – konstanta, 32b

A,B,C,D,E – vnitřní stavy - 32bitů (každý)

- celkem 160 bitů

- na konci procesu tam je hash

- na počátku konstanty



SHA-1

Funkce *F* v jednotlivých rundách algoritmu SHA-1

$$f(t,B,C,D) = (B \wedge C) \vee (\neg B \wedge D) \qquad (0 \le t \le 19)$$

$$f(t,B,C,D) = B \oplus C \oplus D \qquad (20 \le t \le 39)$$

$$f(t,B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) \qquad (40 \le t \le 59)$$

$$f(t,B,C,D) = B \oplus C \oplus D \qquad (60 \le t \le 79)$$

Konstanta K_t

K_t = 0x5A827999 (0 ≤ t ≤ 19)
K_t = 0x6ED9EBA1 (20 ≤ t ≤ 39)
K_t = 0x8F1BBCDC (40 ≤ t ≤ 59)
K_t = 0xCA62C1D6 (60 ≤ t ≤ 79)

Počáteční hodnoty registrů A..E

A = 0x67452301 B = 0xEFCDAB89
C = 0x98BADCFE D = 0x10325476
E = 0xC3D2E1F0

SHA-256

- po přijetí standardu AES vznikly další modifikace SHA-1
- označují se jako SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512)
- nové vstupní hodnoty (IV) - odpovídají desetinným částem druhých odmocnin prvních osmi prvočísel
- v každém kole se používají odlišné aditivní konstanty
  - odpovídají desetinným částem třetích odmocnin prvních 64 prvočísel
- opět konvence big-endian
- SHA-224 je realizována jako SHA-256 s jinými počátečními hodnotami a příslušně oříznutým výstupem

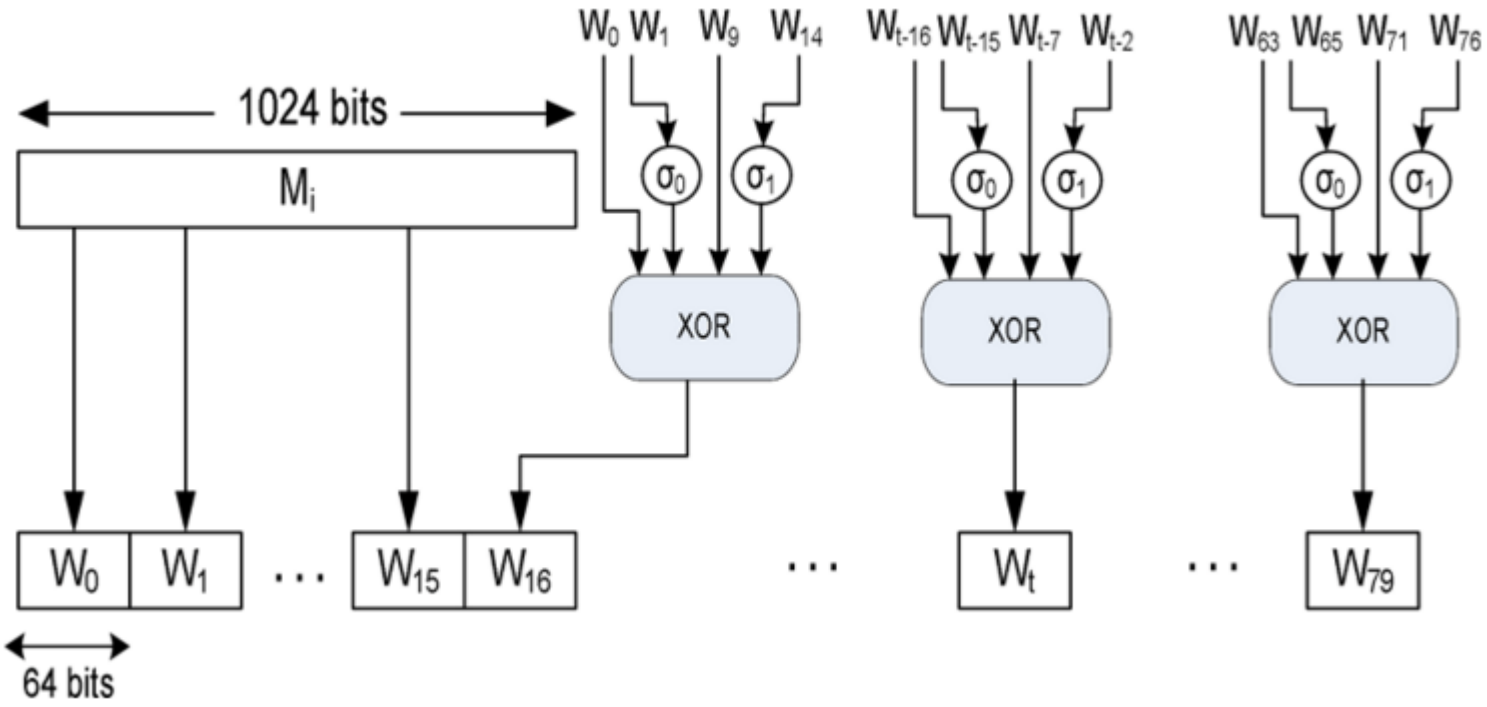
SHA-512

- pracuje s 64bitovými slovy (konstanta K_t i zpráva W_t má délku 64b)
- vstupní text se zpracovává po 1024bitových částech
- počáteční hodnoty registrů odpovídají desetinným částem druhých odmocnin 9. - 16. prvočísel
- délka výstupu 512 bitů

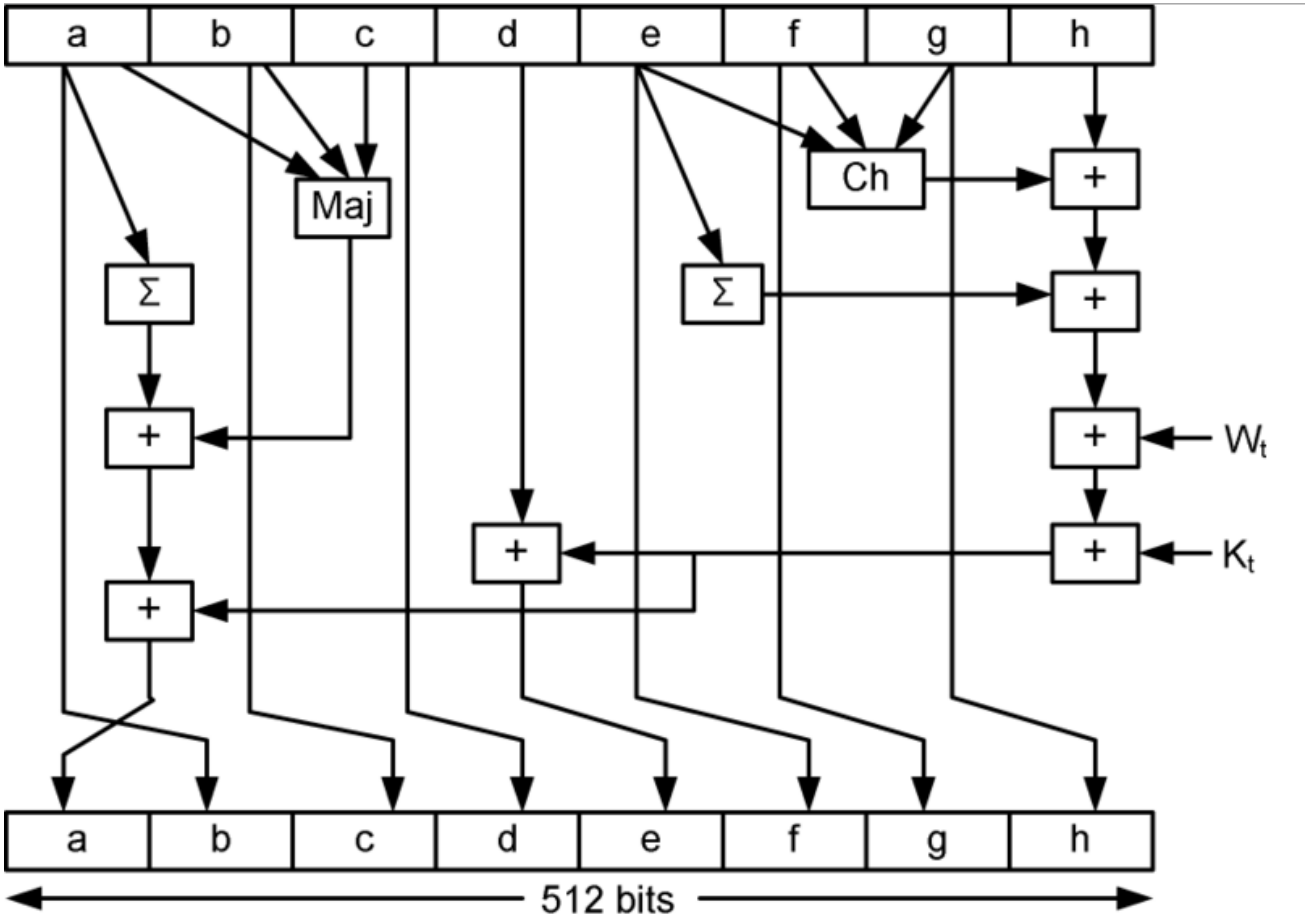
- SHA-384 vychází ze SHA-512
  - jiné aditivní konstanty
  - jiné počáteční hodnoty registrů
  - výsledný hash je zleva zkrácen z 512 bitů na 384 bitů

SHA-512 – expanze bloku

- Expanze vstupního bloku 1024b → 5120b
- V každém kroku se použije 64b



SHA-512



SHA-512 – význam operací

- $+$ sčítání mod 2^{32}
- $C(E,F,G) = (E \wedge F) \oplus (\neg E \wedge \neg G)$
- $Maj(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$
- $\Sigma(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$
- $\Sigma(E) = (A \ggg 6) \oplus (A \ggg 11) \oplus (A \ggg 25)$

Srovnání hashovacích funkcí SHA-x

	SHA-0	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Délka hashe	128	160	224	256	384	512
Délka zprávy	$<2^{64}$	$<2^{64}$	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Velikost bloku	512	512	512	512	1024	1024
Velikost slova	32	32	32	32	64	64
Počet kroků algoritmu	80	80	64	64	80	80
Ekvivalentní Bezpečnost[b]	64	80	112	128	192	256
Nalezení Kolize	Ano	Teoret. 2^{63}	Ne	Ne	Ne	Ne

Srovnání MD-5, SHA-1, RIPEMD-160

Hashovací funkce	**MD5**	**SHA1**	**RIPEMD-160**
Délka hashe [b]	128	160	160
Velikost zpracovávaného bloku [b]	512	512	512
Počet rund	4	4	5 párů
Počet kroků v rundě	15	20	16
Maximální velikost vstupu [b]	∞	$2^{64}-1$	$2^{64}-1$
Počet logických funkcí	4	4	5
Počet konstant	64	4	10

Srovnání rychlostí algoritmů

- Intel Core 2
  - 2 jádra
  - 1.83 GHz
- 32bitový OS
  - Windows Vista
- x86/MMX/SSE2

Algortimus	MiB/s	Cyklů/Byte
CRC32	253	6.9
MD5	255	6.8
SHA-1	153	11.4
SHA-256	111	15.8
SHA-512	99	17.7
Tiger	214	8.1
Whirlpool	57	30.5
RIPEMD-160	106	16.5
RIPEMD-320	110	15.9
RIPEMD-128	153	11.4

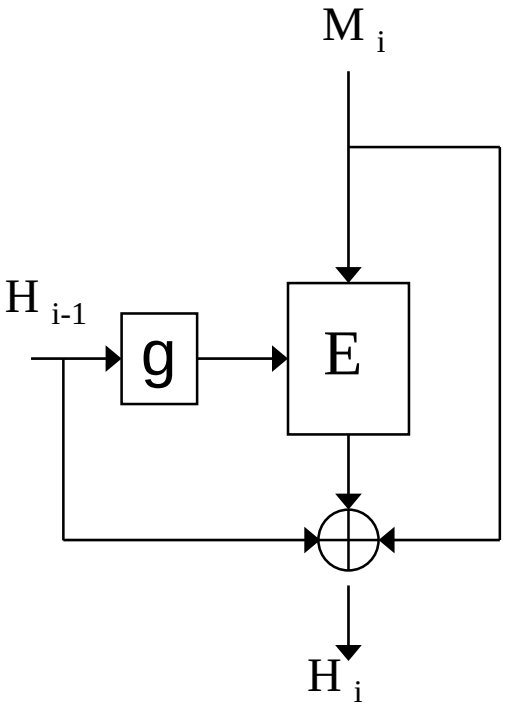
Whirlpool

- není patentovaná
- standard ISO/IEC 10118-3:2004
- kompresní funkce je tvořena speciální blokovou šifrou
  - upravený AES
  - 10 rund
  - jeden speciální S-box
- vstup $<2^{256}$ bitů
- výstup 512 bitů
- vnitřní struktura WHIRPOOLu má Miyaguchi-Preneelovo schéma
- pojmenována po spirálové galaxii v souhvězdí Honících se psů
- bezpečná
- používá se např. v FreeOTFE, Truecrypt/Veracrypt

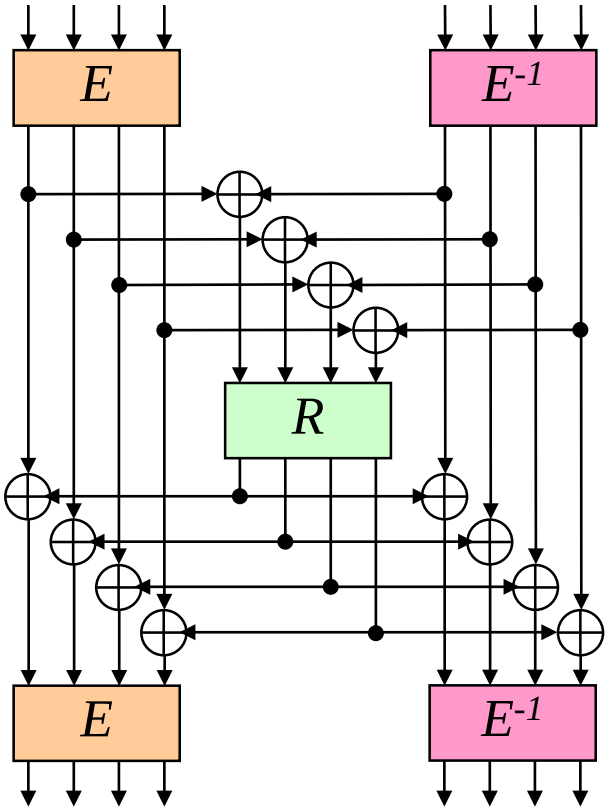
Whirlpool – rozdíl

	RIJNDAEL	***Whirlpool***
Velikost bloku [b]	128, 160, 192, 224, 256	vždy 512
Počet rund	10, 11, 12, 13, or 14	vždy 10
Příprava klíčů	Dáno specifickým algoritmem	Vlastní rundová funkce
GF(2^8) nedělitelný polynom	$x^8 + x^4 + x^3 + x + 1$ (`0x11B`)	$x^8 + x^4 + x^3 + x^2 + 1$ (`0x11D`)
Původ S-boxu	mapování $u \rightarrow u^{-1}$ nad GF(2^8), + afinní transformace	rekurzivní struktura (viz níže)
Původ konstant v rundě	polynomy x^i nad GF(2^8)	Následující záznamy z S-boxu
Difúzní vrstva	násobení zleva pomocí matice 4×4 circulant MDS cir(2,3,1,1)	násobení zprava pomocí matice 8×8 circulant MDS cir(1,1,4,1,8,5,2,9)

Whirlpool



Miyaguchi-Preneelova kompresní funkce



S-box – kryptografické jádro

Whirlpool

E mini S-box:

u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E(*u*)	1	B	9	C	D	6	F	3	E	8	7	4	A	2	5	0

E^{-1} mini S-box:

u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$E^{-1}(u)$	F	0	D	7	B	E	5	A	9	2	C	1	3	4	8	6

R mini S-box:

u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
R(*u*)	7	C	B	D	E	4	9	F	6	3	8	A	2	5	1	0

Hledání SHA-3 - plán

- 06/2006 – představen časový plán výběru nové hashovací funkce SHA-3 neboli AHS (Advanced Hash Standard)
  - analogický proces s hledáním AES
  - do roku 2012 by měl být nalezen kandidát aby mohl být obsažen v aktualizace standardu FIPS 180-2 (plánované aktualizace jsou v 2007 a 2012)
- 02/2007 – představeny minimální akceptační požadavky kandidátů a hodnotící kritéria
- 12/2007 – finální verze požadavků a kritérií. Začíná přijímání kandidátských návrhů.
- 3Q/2008 – Konec přihlašování nových funkcí do soutěže. Začátek procesu zkoumání kandidátů.
- 4Q/2009 – Ukončení komentářů k vybraným kandidátům.
- 1Q/2010 – Výběr finalistů. Poslední úpravy funkcí, které postoupili do finále.
- 2Q/2011 – Ukončení finálového kola výběru. Poslední konference – výběr vítěze.
- 2012 – Zveřejnění oficiální specifikace AHS.

Hledání SHA-3 - požadavky

- rychlost
  - SHA-1 7,5 cyklu procesoru/byte
  - SHA-2 20 cyklů procesoru/byte)
  - nároky na paměť (stovky bytů)

- 64 algoritmů od 191 kryptoolgů
  - Microsoft, Sony, RSA, Intel, IBM, MIT, PGP, Hitachi
  - Rivest, Schneier, Bernstein,…
  - Češi – 2 kandidáti
    - EDON-R – Drápal, Klíma, Grigoroski
    - BMW – Klíma, Gligoroski

SHA-3

- přehled algoritmů kandidujících na SHA-3:

http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo

- 64 přihlášených algoritmů
- 56 je dnes veřejně dostupných

Z nich:

- 15 algoritmů vyřazeno ještě před prvním kolem
  - stáhli sami autoři nebo
  - nalezena kolize nebo
  - 2nd-preimage kolize
- 27 algoritmů vyřazeno před druhým kolem
  - včetně MD-6

Hledání SHA-3

- 07/2009 - 14 algoritmů postoupilo do druhého kola
  - BLAKE, BLUE MIDNIGHT WISH, CubeHash, ECHO, Fugue, Grøstl, Hamsi,JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, Skein
- 09/2010 – NIST dosud neví kolik ani podle jakých pravidel vybrat finalisty
  - mělo být známo v Q1/2010!
  - výběr 4-6 algoritmů do konce 2010
  - různé typy konstrukcí (wide-pipe, narrow-pipe, sponge, AES-based)
- 9.12.2010 – NIST oznámil 5 finalistů

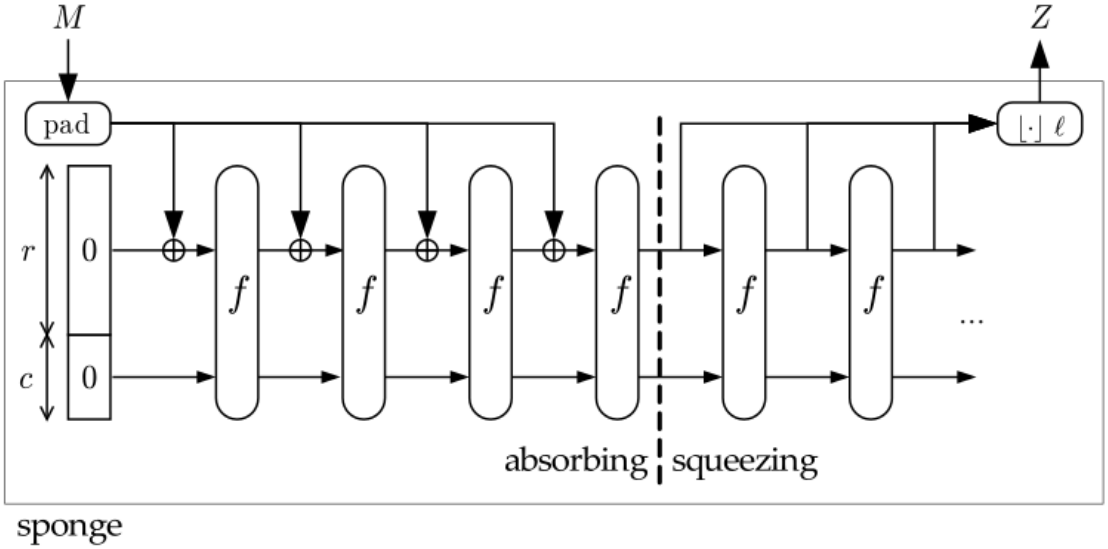
64 bitový procesor, 256 bitový hašový kód, rychlost v cyklech/byte				64 bitový procesor, 512 bitový hašový kód, rychlost v cyklech/byte		
1	Blue Midnight Wish	7.55		1	Blue Midnight Wish	3.88
2	Skein	7.6		2	Skein	6.1
3	Shabal	8.03		3	Shabal	8.03
4	BLAKE	8.19		4	BLAKE	9.29
5	Keccak	10		5	CubeHash	11
6	CubeHash	11		6	SIMD	12
7	SIMD	11		7	SHA-512	12.59
8	Luffa	13.4		8	JH	16.8
9	SHA-256	15.34		9	Keccak	20
10	JH	16.8		10	Luffa	23.2
11	Grøstl	22.2		11	Hamsi	25
12	Hamsi	25		12	Grøstl	30.5
13	SHAvite-3	26.7		13	SHAvite-3	38.2
14	Fugue	28		14	ECHO	53.5
15	ECHO	28.5		15	Fugue	56

SHA-3 finále

- 5 algoritmů, které postoupilo do finále
  - BLAKE
  - Grøstl
  - JH
  - Keccak (5 autorů – Joan Daemen)
  - Skein (B. Schneier)
- překvapivě vyřazeny „silní“ kandidáti
  - BMW
  - SIMD
  - Shabal
- 2.10.2012 – oznámen vítězný algoritmus **Keccak** [kɛtʃak]

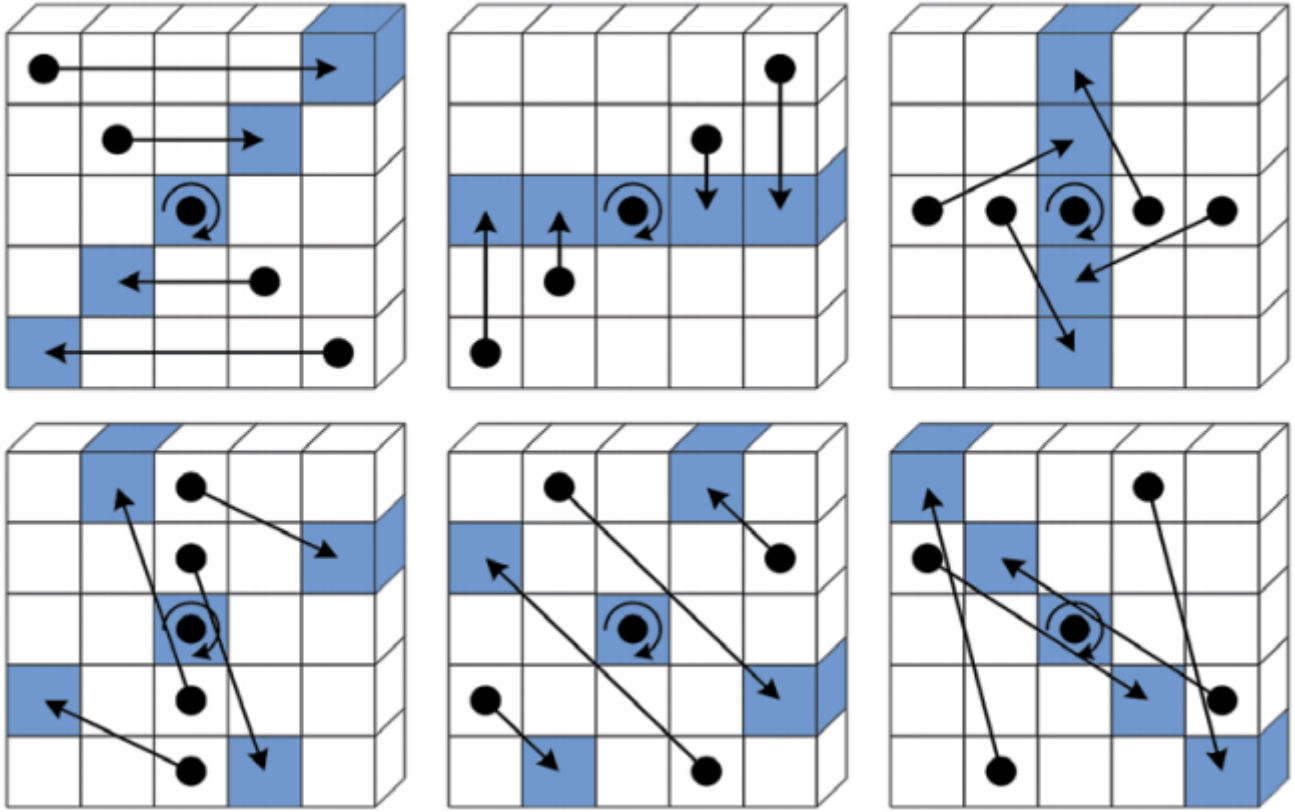
SHA3 - Keccak

- výborný výkon v HW implementacích
- BLAKE a Skein lepší v SW implementacích, ale Keccak lze velmi efektivně paralelizovat
- houbovitá (sponge) struktura
  - http://sponge.noekeon.org/
  - fáze absorpce a fáze zmáčknutí



SHA3 - Keccak

- detail operací v *f*



SHA-3 - aktuální stav

- SHA512/224 = trunc_{224}(SHA-512)
- SHA512/256 = trunc_{256}(SHA-512)
  - proč ?
  - protože SHA512 je na 64bitových CPU výrazně rychlejší než SHA256
  - velké posílení bezpečnosti

Předpoklady

  - než se rozšíří SHA3 bude naprostá většina počítačů 64 bitových
  - většina počítačů bude mít HW instrukce pro AES / vektorové výpočty

- SHA-3
  - standard publikován 5.8.2015
  - FIPS 202 - http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

Použití hashovacích funkcí - obecně

- zajištění integrity dat
- digitální podpis
- ukládání hesel
  - salt (sůl)
  - v praxi také funkce PBKDF2, bcrypt, scrypt
- porovnání obsahu dvou kopií dat
- generování pseudonáhodných posloupností (PRNG)

Hashovací funkce jako základ PRNG

- chování jako náhodné orákulum
  - změna 1 bitu na vstupu -> změna každého výstupního bitu s pravděpodobností 50 %

- PKCS#1 v.2.1 (RSA Cryptography Standard) definuje MGF1 (Mask Generation Function)
  - h(seed||0x00000001), h(seed||0x00000002), h(seed||0x00000003),…

  kde 0x… je hexadecimální vyjádření daného čísla a *h* je hashovací funkce
- DSA (Digital Signature Algorithm)
- podpisové schéma definuje náhodný generátor

  $x_0 := K$, $x_{j+1} := h((K + 1 + x_j) \bmod 2^m)$,

  kde *K* je počáteční vstup (klíč) a *m* délka bloku hašovací funkce *h*

Použití hashovacích funkcí – konkrétně

- **Digitální / elektronický podpis** – zajištění integrity
- **PKI** - integrita X.509 certifikátů a seznamů CRL
- **Časové značky –** integrita časové značky
- **Kerberos -** generování klíčů
- **IEEE 802.1X EAP :** EAP-FAST, EAP-TLS, EAP-TTLS… používají TLS protokol, který používá hashovací funkce
- **RADIUS -** integrita dat
- **IPsec (IKE, AH, ESP)** – integrita zpráv, generování pseudonáhodných posloupností.
- **SSL/TLS -** handshake protokol používá hashovací funkce kvůli tvorbě HMAC a při generování klíčů a IV
- **SSH** – HMAC a integrita přenášených dat
- **S/MIME** - použití hashovacích funkcí v digitálním podpise

Použití hashovacích funkcí – konkrétně

- Ukládání hesel v Linuxu
  - /etc/shadow
- Řádek = uživatel

tom:1Etg2ExUZ$F9NTP7omafhKIlqaBMqng1:15651:0:99999:7:
jan:6vb1tLY1qiY$M.1ZCqKtJBxBtZm1gRi8Bbkn39KU0YJW1cuMFzTRANcNKFKR4RmAQVk4rqQQCkaJT6wXqjUkFcA/qNxLyqW.U/:15405:0:99999:7::

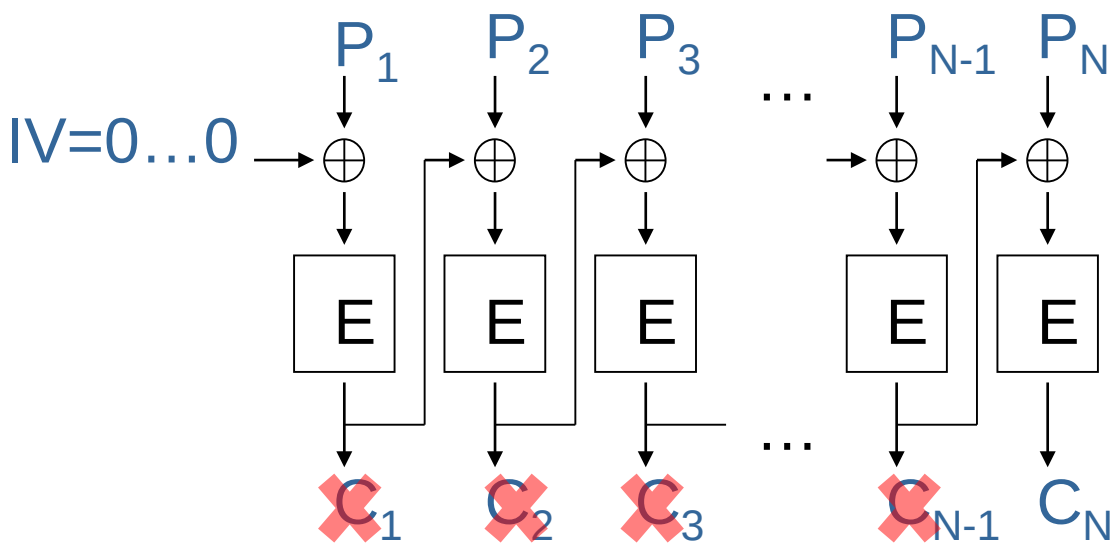
- Číslo mezi 1.$ a 2.$ určuje algoritmus
  - $1 = MD5
  - $2a = Blowfish
  - $2y = Blowfish se správným zpracováním 8bitových znaků
  - $5 =SHA-256
  - $6 =SHA-512
- Řetězec mezi 2.$ a 3.$ je sůl (salt) - Etg2ExUZ
- **Hash = H(**PASSWORD**||**SALT**)**

CBC-MAC

- bloková šifra v režimu CBC
- IV= 000…000
- tajný klíč K_{MAC}
  - tento klíč by měl být jiný, než na šifrování zpráv
- ze zprávy M={m_1, …, m_n} se vygeneruje ŠT={c_1, …, c_n}
- MAC je tvořen pouze posledním blokem c_n
  - v praxi se z posledního bloku obvykle bere jen určitá část o délce potřebné k vytvoření odolného zabezpečovacího kódu
- ostatní bloky ŠT se průběžně zahazují (neodesílají se !)
- MAC zajišťuje autentizaci původu dat
- chrání proti úmyslným modifikacím dat během přenosu

CBC-MAC

- MAC nezajišťuje nepopiratelnost (protože to je symetrická technika a nelze s ní rozlišit data od účastníka A a B)
- Alice pošle Bobovi $\{m_1, \ldots, m_n, c_n\}$
- Bob zrekonstruuje z přijatých bloků $\{m1, \ldots, m_n\}$ hodnotu c'_n
- pokud nebyla zpráva při přenosu změněna, pak se musí $c_n = c'_n$



$C_1 = E(P_1 \oplus IV)$

$C_i = E(P_i \oplus C_{i-1})$ pro i=2..N

HMAC – základní údaje

- Keyed-hash Message Authentication Code
- kryptografický kontrolní součet, kryptografická hašovací funkce, klíčovaná haš,…

- označení podle toho, typu použité hašovací funkce např. HMAC-SHA-1, HMAC-MD5

- zpracovává nejen data, ale i klíč
- výsledek je závislý na klíči a tudíž "nepadělatelný„
- ověřuje integritu dat a zdroj dat

- **odlišuje se od MAC, funkčně podobný, ale používá jiné stavební prvky a je kryptograficky silnější**

HMAC – základní údaje

- doporučení RFC 2104, FIPS 198.1
- nejpoužívanější HMAC jsou HMAC-SHA-1 a HMAC-SHA256

HMAC-SHA-1:
blok B=64 bajtů, klíč *K*, H = SHA-1

ipad = řetězec B bajtů „0x36“

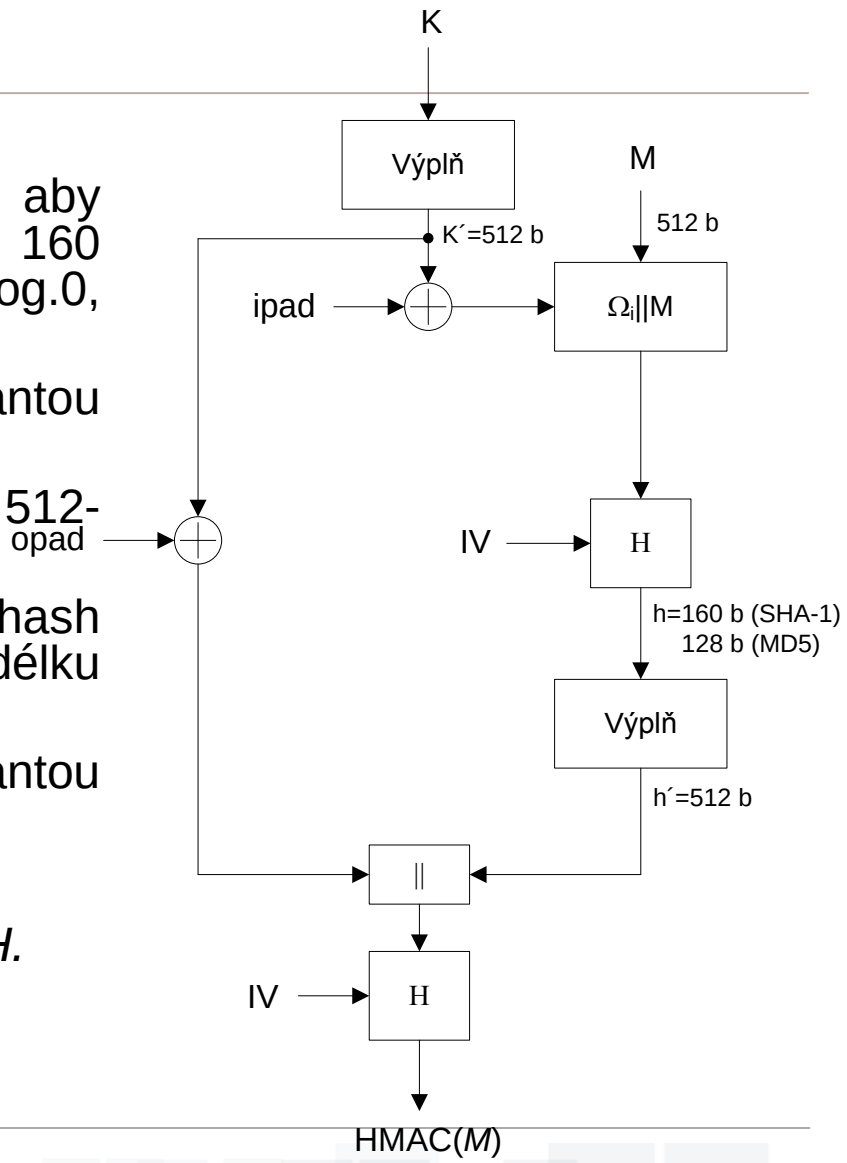
opad = řetězec B bajtů „0x5C“

klíč K se doplní nulovými bajty do plného bloku délky B

$$HMAC(M) = H((K \oplus opad) \,||\, H((K \oplus ipad) \,||\, M))$$

HMAC - princip

- Klíč *K* se doplní bity s hodnotou log.0 tak, aby měl délku 512 bitů (pokud má klíč délku např. 160 bitů, připojí se na konec 352 bitů s hodnotou log.0, tím vznikne klíč *K´*).
- Provede se operace XOR klíče *K´* s konstantou *ipad*.
- K výsledku předchozího kroku se připojí 512-bitová zpráva *M*.
- Na blok vytvořený v kroku 3 se aplikuje hash funkce *H*, výsledek *h* je doplněn nulami na délku 512 bitů (vznikne *h´*).
- Provede se operace XOR klíče *K´* s konstantou *opad*.
- K výsledku bodu 5 se připojí *h´.*
- Na výsledek bodu 6 se aplikuje hash funkce *H.*



Odkazy

- Xiaoyun Wang, Dengguo Feng , Xuejia Lai, Hongbo Yu: Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, rump session, CRYPTO 2004, *Cryptology ePrint Archive*, Report 2004/199,http://eprint.iacr.org/2004/199.pdf

Detailní informace o kolizích v MD5 (česky)

- http://crypto-world.info/klima/2005/cryptofest_2005.htm
- http://cryptography.hyperlink.cz/md5/Vlastimil_Klima_MD5_kolize.pdf
- http://cryptography.hyperlink.cz/2006/tunely.pdf

- The first collision for full SHA-1 https://shattered.io/static/shattered.pdf

- RFC 6234 – US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)
- RFC 2104 – HMAC Key Hashing for Message Authentication

Dotazy