Hašovací funkce

Bc. Jakub Kolář

Fakulta jaderná a fyzikálně inženýrská
České vysoké učení technické v Praze

16. dubna 2012

Obsah

Obsah

Co je to hašovací funkce?

- $\mathcal{M}$ = množina *zpráv* (konečná nebo spočetná)
- $\mathcal{H}$ = množina *otisků* (konečná)

Co je to hašovací funkce?

- $\mathcal{M} =$ množina *zpráv* (konečná nebo spočetná)
- $\mathcal{H} =$ množina *otisků* (konečná)

Definice

Hašovací funkcí rozumíme zobrazení $h : \mathcal{M} \mapsto \mathcal{H}$.

Co je to hašovací funkce?

- $\mathcal{M}$ = množina *zpráv* (konečná nebo spočetná)
- $\mathcal{H}$ = množina *otisků* (konečná)

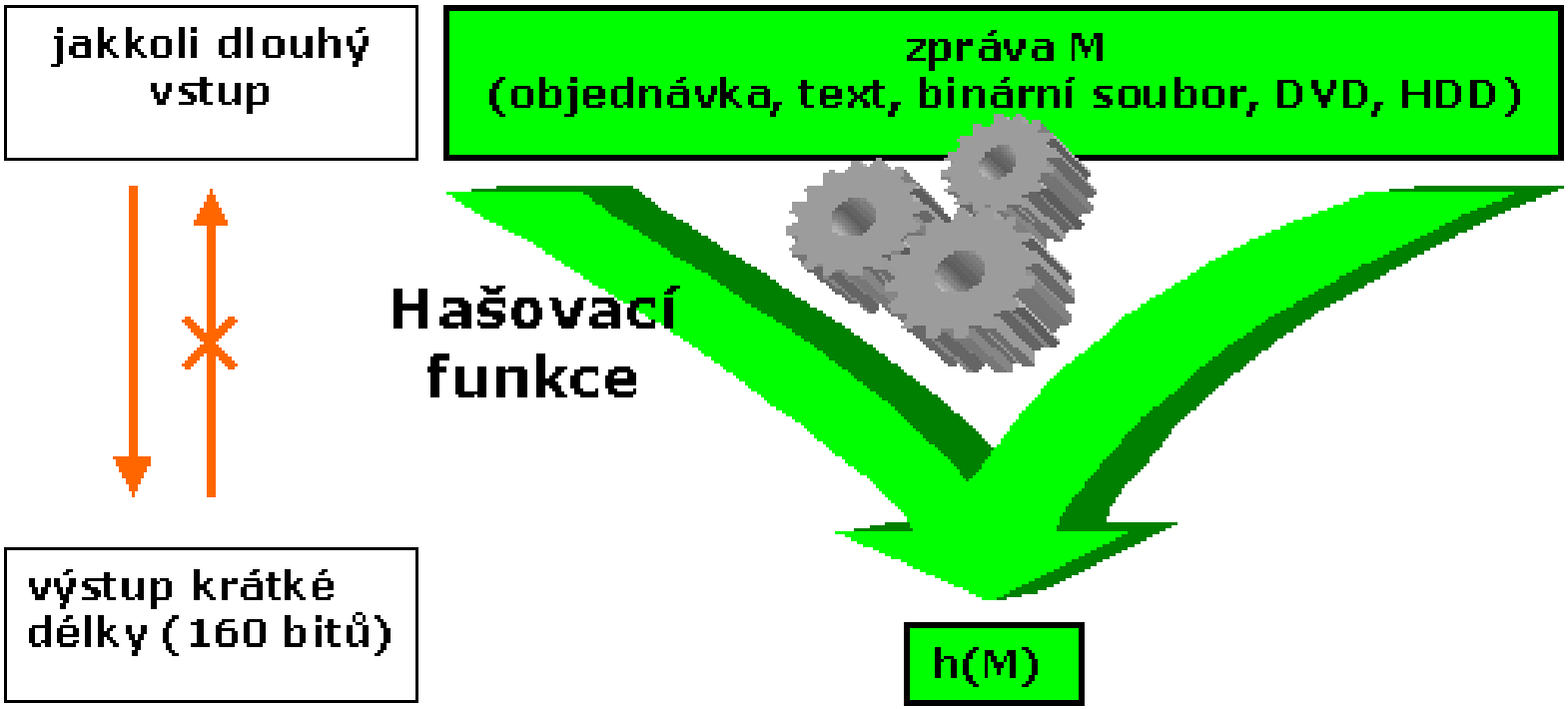
Definice

Hašovací funkcí rozumíme zobrazení $h : \mathcal{M} \mapsto \mathcal{H}$.

- $\mathcal{M}$ obvykle podstatně větší než $\mathcal{H}$
- Když konečné $\Rightarrow$ búno čísla, binárně zapsaná:

$$\mathcal{M} = \{0,1\}^N \quad , \quad \mathcal{H} = \{0,1\}^n$$

Co je to hašovací funkce?



Obrázek: Hašovací funkce schematicky

Hloupý příklad

$h : \mathbb{N}_0 \mapsto \{0, \ldots, p-1\}$

- p prvočíslo
- pro $m \in \mathbb{N}_0$:

$$h(m) = m \mod p$$

Hloupý příklad

$h : \mathbb{N}_0 \mapsto \{0, \ldots, p-1\}$

- p prvočíslo
- pro $m \in \mathbb{N}_0$:

$$h(m) = m \mod p$$

Proč je např. $p = 2^k$ nevhodné?

Použití

- hašovací tabulka

Použití

- hašovací tabulka
- kontrolní otisk – detekce chyb

Použití

- hašovací tabulka
- kontrolní otisk – detekce chyb
- porovnávání dat

Použití

- hašovací tabulka
- kontrolní otisk – detekce chyb
- porovnávání dat
- kryptografické hašovací funkce
  - odhalení úmyslného poškození dat
  - digitální podpisy
  - PRNG

Obsah

Jednoduchý příklad použití

Ukládání a kontrola přihlašovacích hesel

- heslo $= m \in \mathcal{M}$
- otisk $= h(m) \in \mathcal{H}$
- přihlašování uživatele – na serveru uloženo pouze $h(m)$

Jednoduchý příklad použití

Ukládání a kontrola přihlašovacích hesel

- heslo $= m \in \mathcal{M}$
- otisk $= h(m) \in \mathcal{H}$
- přihlašování uživatele – na serveru uloženo pouze $h(m)$

Útočník

- chce se přihlásit
- pokud možno získat m
- server kompromitován $\Rightarrow$ útočník získá $h(m)$

Jednosměrnost

Jednosměrná funkce

S danými výpočetními prostředky a v daném čase není pro žádné $H \in \mathcal{H}$ možné vypočíst nějaké $m \in \mathcal{M}$ tak, aby

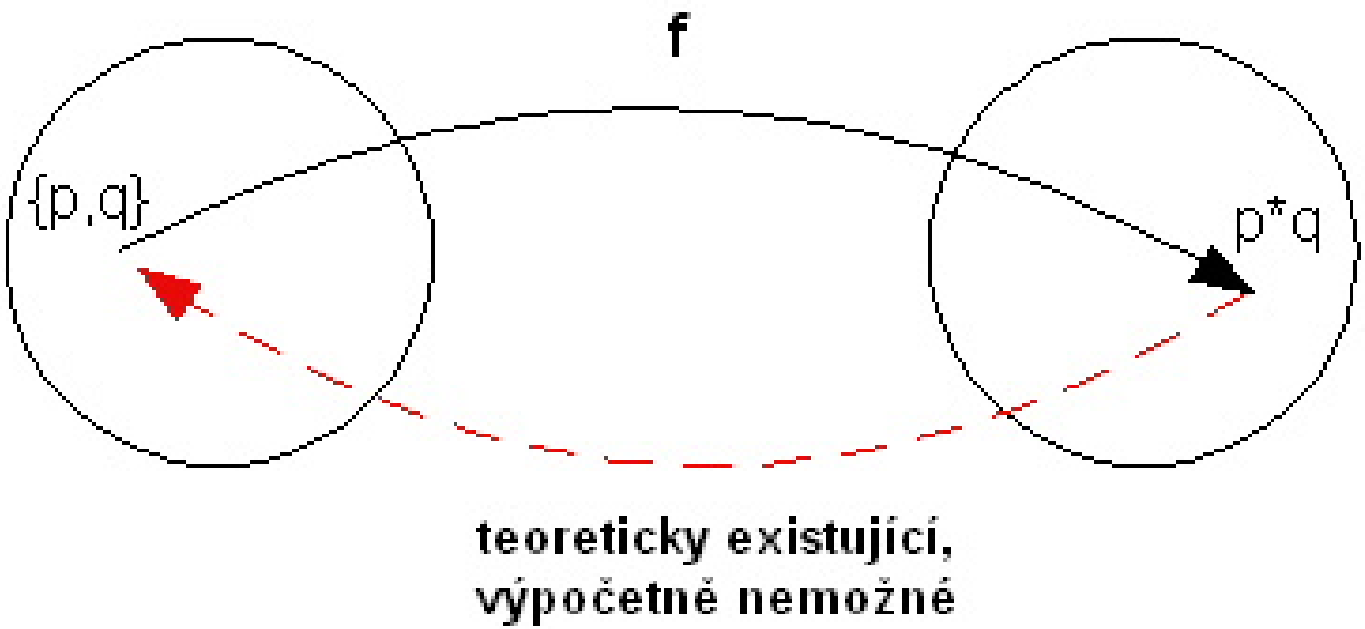
$$h(m) = H \ .$$

Jednosměrnost

Jednosměrná funkce

S danými výpočetními prostředky a v daném čase není pro žádné $H \in \mathcal{H}$ možné vypočíst nějaké $m \in \mathcal{M}$ tak, aby

$$h(m) = H \ .$$

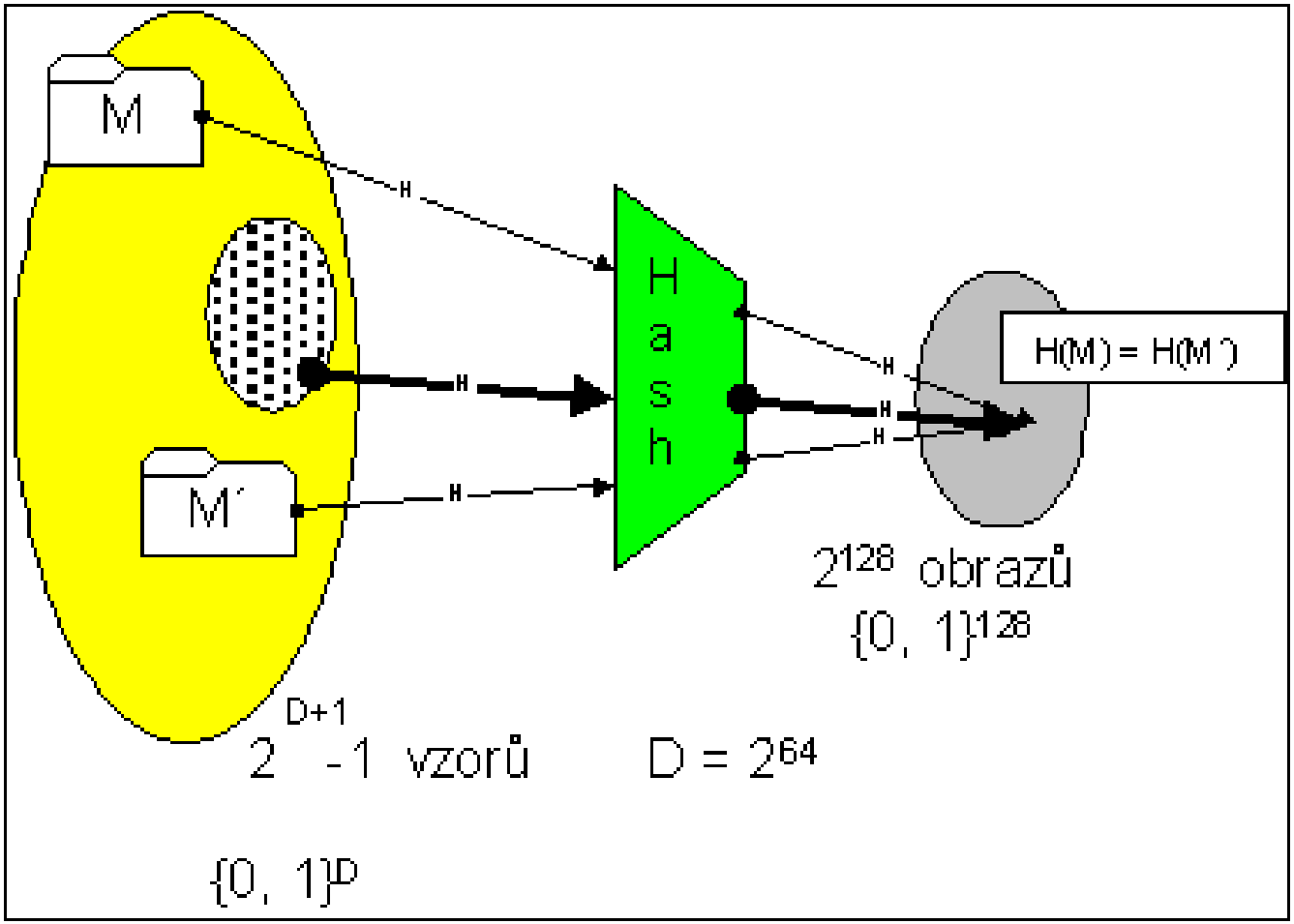


Kolize

- *kolize* zpráv $m, m' \in \mathcal{M}$ nastává, jestliže $h(m) = h(m')$
- kolizí je mnoho

Kolize

- *kolize* zpráv $m, m' \in \mathcal{M}$ nastává, jestliže $h(m) = h(m')$
- kolizí je mnoho



Odolnost proti kolizím

Bezkoliznost prvního řádu

S danými výpočetními prostředky a v daném čase je výpočetně nemožné nalézt dvojici kolidujících zpráv.

Odolnost proti kolizím

Bezkoliznost prvního řádu

S danými výpočetními prostředky a v daném čase je výpočetně nemožné nalézt dvojici kolidujících zpráv.

Bezkoliznost druhého řádu

S danými výpočetními prostředky a v daném čase je pro danou zprávu $m \in \mathcal{M}$ výpočetně nemožné nalézt zprávu $m' \in \mathcal{M}$ tak, aby

$$h(m) = h(m') \ .$$

Zpět k jednoduchému příkladu

Kolize

- nejsou žádoucí
- $h(m)$ není jednoznačný identifikátor
- uživatel pomocí svého hesla na cizí účet

Zpět k jednoduchému příkladu

Kolize

- nejsou žádoucí
- $h(m)$ není jednoznačný identifikátor
- uživatel pomocí svého hesla na cizí účet

Útok na jednosměrnost

- *rainbow tables*

Využití kolizí

Příklad

Využití kolizí

Příklad

- Vývojář $\Rightarrow$ program $= m \in \mathcal{M}$
- uveřejní společně s otiskem $h(m)$ nebo s digitálním podpisem
- uživatel si stáhne, ověří a nainstaluje

Využití kolizí

Příklad

- Vývojář $\Rightarrow$ program $= m \in \mathcal{M}$
- uveřejní společně s otiskem $h(m)$ nebo s digitálním podpisem
- uživatel si stáhne, ověří a nainstaluje

Útočník

- $m' =$ program $+$ škodlivý virus
- pokud $h(m') = h(m)$, uživatel nepozná

Využití kolizí

Využití

Využití kolizí

Využití

- kde se kontroluje pravost dat

Využití kolizí

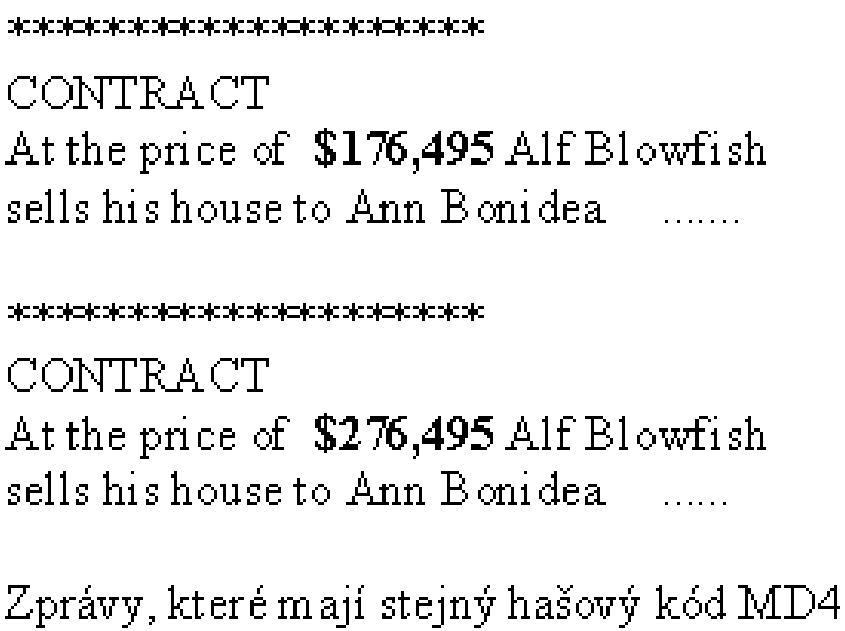
Využití

- kde se kontroluje pravost dat
- „falšování“ digitálního podpisu

Využití kolizí

Využití

- kde se kontroluje pravost dat
- „falšování“ digitálního podpisu
- nutno porušení bezkoliznosti *druhého řádu*



Využití kolizí

Bezkoliznost *prvního řádu*?

Využití kolizí

Bezkoliznost *prvního řádu*?

- útočník musí připravit obě zprávy sám

Využití kolizí

Bezkoliznost *prvního řádu*?

- útočník musí připravit obě zprávy sám
- jednoduchý příklad:

  Zaměstnanec dává nadřízenému podepsat žádost o dovolenou, se kterou mu „náhodou“ koliduje příkaz na zvýšení platu.

Využití kolizí

Bezkoliznost *prvního řádu*?

- útočník musí připravit obě zprávy sám
- jednoduchý příklad:

  Zaměstnanec dává nadřízenému podepsat žádost o dovolenou, se kterou mu „náhodou“ koliduje příkaz na zvýšení platu.

- oklamání CA $\Rightarrow$ vydání certifikátu na dva různé podpisové klíče

Využití kolizí

K využití potřeba *smysluplné kolize* (meaningful collisions)

- m a m' dávají smysl
- při útoku hrubou silou není splněno, ale lze obejít

Využití kolizí

K využití potřeba *smysluplné kolize* (meaningful collisions)

- m a m' dávají smysl
- při útoku hrubou silou není splněno, ale lze obejít

Další požadavek na hašovací funkci

- Pro „blízká“ $m, m' \in \mathcal{M}$ je $h(m) \neq h(m')$

Využití kolizí

K využití potřeba *smysluplné kolize* (meaningful collisions)

- m a m' dávají smysl
- při útoku hrubou silou není splněno, ale lze obejít

Další požadavek na hašovací funkci

- Pro „blízká“ $m, m' \in \mathcal{M}$ je $h(m) \neq h(m')$

alternativně

Využití kolizí

K využití potřeba *smysluplné kolize* (meaningful collisions)

- m a m' dávají smysl
- při útoku hrubou silou není splněno, ale lze obejít

Další požadavek na hašovací funkci

- Pro „blízká“ $m, m' \in \mathcal{M}$ je $h(m) \neq h(m')$

alternativně

- je-li $h(m) = h(m')$, pak spolu m a m' „nesouvisí“

$\Rightarrow h(m)$ je „v podstatě“ jednoznačný identifikátor m

Hledání kolizí

$h : \{0,1\}^N \mapsto \{0,1\}^n$, kde $N \gg n$

Hledání kolizí

$h : \{0,1\}^N \mapsto \{0,1\}^n$, kde $N \gg n$
Útok hrubou silou:

- náhodně vybíráme různá $m_i \in \mathcal{M}$
- maximálně $2^n + 1$ výběrů do nalezení kolize

Hledání kolizí

$h : \{0,1\}^N \mapsto \{0,1\}^n$, kde $N \gg n$
Útok hrubou silou:

- náhodně vybíráme různá $m_i \in \mathcal{M}$
- maximálně $2^n + 1$ výběrů do nalezení kolize

Dobrá funkce h:

- náhodně nepredikovatelně rozděluje obrazy $h(m_i)$ na $\mathcal{H}$

Hledání kolizí

Vylepšení – *birthday attack*

Hledání kolizí

Vylepšení – *birthday attack*

- chceme, aby $P(\text{kolize po } k \text{ výběrech}) \geq p_0$
- p_0 dostatečně velké
- jak velké k?
- $k \sim \text{konst} \cdot 2^{\frac{n}{2}}$

Obsah

CRC - Cyklický redundantní součet

- založeno na cyklických samoopravných kódech
- snadná implementace, i na hardwaru
- zbytek po dělení polynomů nad $GF(2)$

CRC - Cyklický redundantní součet

- založeno na cyklických samoopravných kódech
- snadná implementace, i na hardwaru
- zbytek po dělení polynomů nad $GF(2)$

Použití

CRC - Cyklický redundantní součet

- založeno na cyklických samoopravných kódech
- snadná implementace, i na hardwaru
- zbytek po dělení polynomů nad $GF(2)$

Použití

- $n = 16, 32, 64$

CRC - Cyklický redundantní součet

- založeno na cyklických samoopravných kódech
- snadná implementace, i na hardwaru
- zbytek po dělení polynomů nad $GF(2)$

Použití

- $n = 16, 32, 64$
- např. Ehthernet, komprese (Gzip, Bzip2)
- vhodné jako ochrana proti běžných chybám při přenosu

CRC - Cyklický redundantní součet

- založeno na cyklických samoopravných kódech
- snadná implementace, i na hardwaru
- zbytek po dělení polynomů nad $GF(2)$

Použití

- $n = 16, 32, 64$
- např. Ehthernet, komprese (Gzip, Bzip2)
- vhodné jako ochrana proti běžných chybám při přenosu
- nevhodné pro kryptografické účely
  - nesplňují bezkoliznost druhého řádu
  - jedna ze slabin protokolu WEP

Moderní hashovací funkce

Společné rysy:

Moderní hashovací funkce

Společné rysy:

- hashování po blocích $m_i \in \mathcal{M}_0$ ($\Rightarrow$ zarovnání)

Moderní hashovací funkce

Společné rysy:

- hashování po blocích $m_i \in \mathcal{M}_0$ ($\Rightarrow$ zarovnání)
- Damgard-Merklův iterativní postup

Moderní hashovací funkce

Společné rysy:

- hashování po blocích $m_i \in \mathcal{M}_0$ ($\Rightarrow$ zarovnání)
- Damgard-Merklův iterativní postup
- iterace *kompresní funkce* f

Moderní hashovací funkce

Společné rysy:

- hashování po blocích $m_i \in \mathcal{M}_0$ ($\Rightarrow$ zarovnání)
- Damgard-Merklův iterativní postup
- iterace *kompresní funkce* f
- *kontext* = výstup H_{i-1} z předchozí iterace

$$H_0 = c$$
$$H_i = f(H_{i-1}, m_i)$$

Moderní hashovací funkce

Společné rysy:

- hashování po blocích $m_i \in \mathcal{M}_0$ ($\Rightarrow$ zarovnání)
- Damgard-Merklův iterativní postup
- iterace *kompresní funkce* f
- *kontext* = výstup H_{i-1} z předchozí iterace

$$H_0 = c$$
$$H_i = f(H_{i-1}, m_i)$$

- většinou $H_i \in \mathcal{H}$

Moderní hashovací funkce

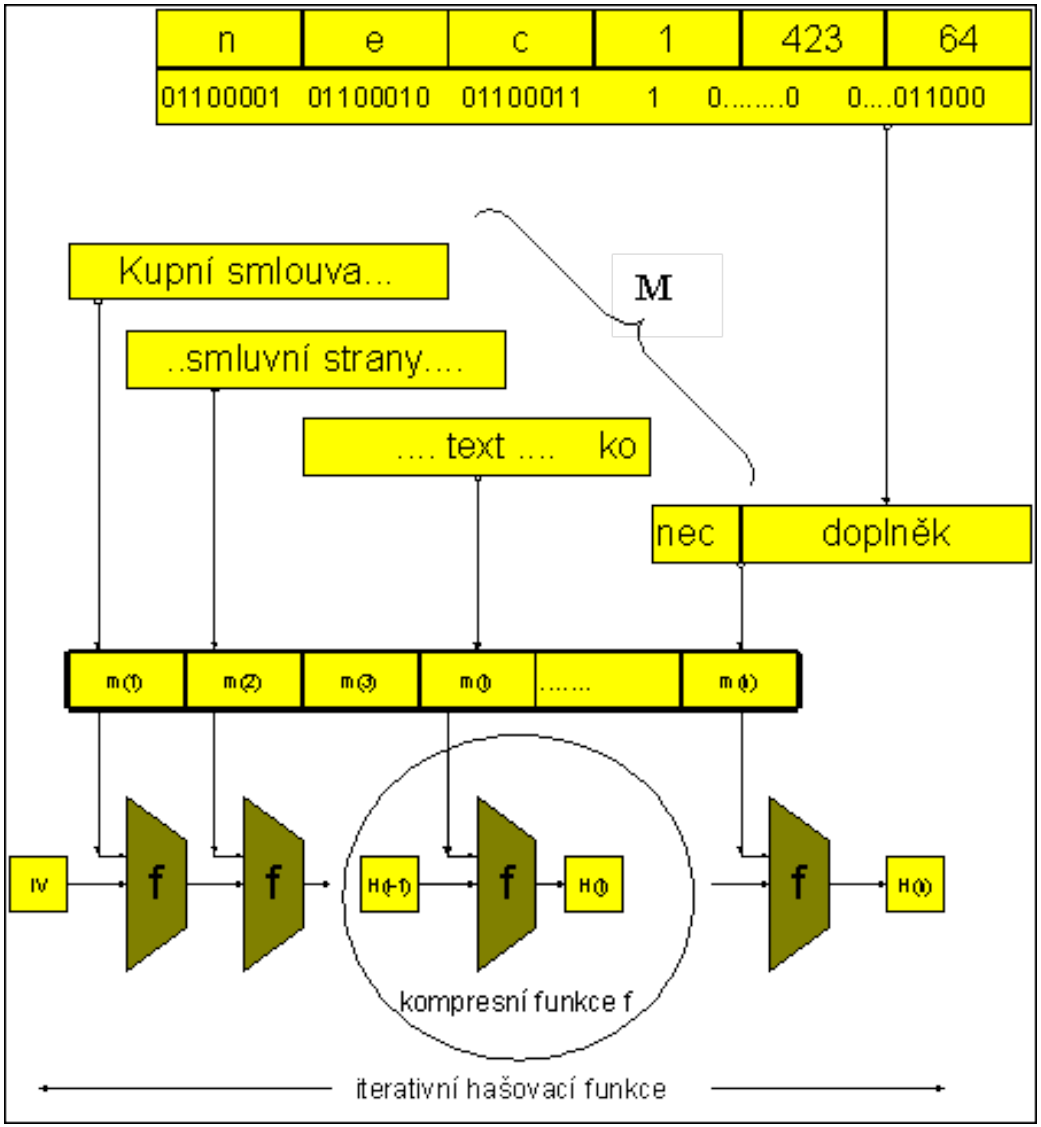
Společné rysy:

- hashování po blocích $m_i \in \mathcal{M}_0$ ($\Rightarrow$ zarovnání)
- Damgard-Merklův iterativní postup
- iterace *kompresní funkce* f
- *kontext* = výstup H_{i-1} z předchozí iterace

$$H_0 = c$$
$$H_i = f(H_{i-1}, m_i)$$

- většinou $H_i \in \mathcal{H}$
- pro $M = m_1 \ldots m_N$ se bere $h(M) = H_N$

Moderní hashovací funkce



Kolize

Kolize funkce f

Kolize pro kompresní funkci nastává pro $H \in \mathcal{H}$, a $m, m' \in \mathcal{M}_0$ takové, že

$$f(H, m) = f(H, m') \ .$$

Kolize

Kolize funkce f

Kolize pro kompresní funkci nastává pro $H \in \mathcal{H}$, a $m, m' \in \mathcal{M}_0$ takové, že

$$f(H, m) = f(H, m') \ .$$

Dokázáno:

- bezkoliznost f $\iff$ bezkoliznost h

Konstrukce kompresní funkce

Používá se Davies-Meyerova konstrukce

$$f(H, m) = E_m(H) \oplus H$$

Konstrukce kompresní funkce

Používá se Davies-Meyerova konstrukce

$$f(H, m) = E_m(H) \oplus H$$

- E_m kvalitní bloková šifra s klíčem m

Konstrukce kompresní funkce

Používá se Davies-Meyerova konstrukce

$$f(H, m) = E_m(H) \oplus H$$

- E_m kvalitní bloková šifra s klíčem m
- xor kvůli zesílení jednocestnosti

Konstrukce kompresní funkce

Používá se Davies-Meyerova konstrukce

$$f(H, m) = E_m(H) \oplus H$$

- E_m kvalitní bloková šifra s klíčem m
- xor kvůli zesílení jednocestnosti
- bloky m obvykle velké (např. 512 bitů) $\Rightarrow$ další rozdělení

$$m = M_1 \dots M_k$$

Konstrukce kompresní funkce

Používá se Davies-Meyerova konstrukce

$$f(H, m) = E_m(H) \oplus H$$

- E_m kvalitní bloková šifra s klíčem m
- xor kvůli zesílení jednocestnosti
- bloky m obvykle velké (např. 512 bitů) $\Rightarrow$ další rozdělení

$$m = M_1 \dots M_k$$

- M_i pak místo m
- tzv. *rundy* (round)

MD5 (Message-Digest algorithm)

Známá hašovací funkce

MD5 (Message-Digest algorithm)

Známá hašovací funkce

- R. Rivest 1991

MD5 (Message-Digest algorithm)

Známá hašovací funkce

- R. Rivest 1991
- $n = 128$

MD5 (Message-Digest algorithm)

Známá hašovací funkce

- R. Rivest 1991
- $n = 128$
- 128 bitů $\Rightarrow$ 64 bitů pro birthday attack

MD5 (Message-Digest algorithm)

Známá hašovací funkce

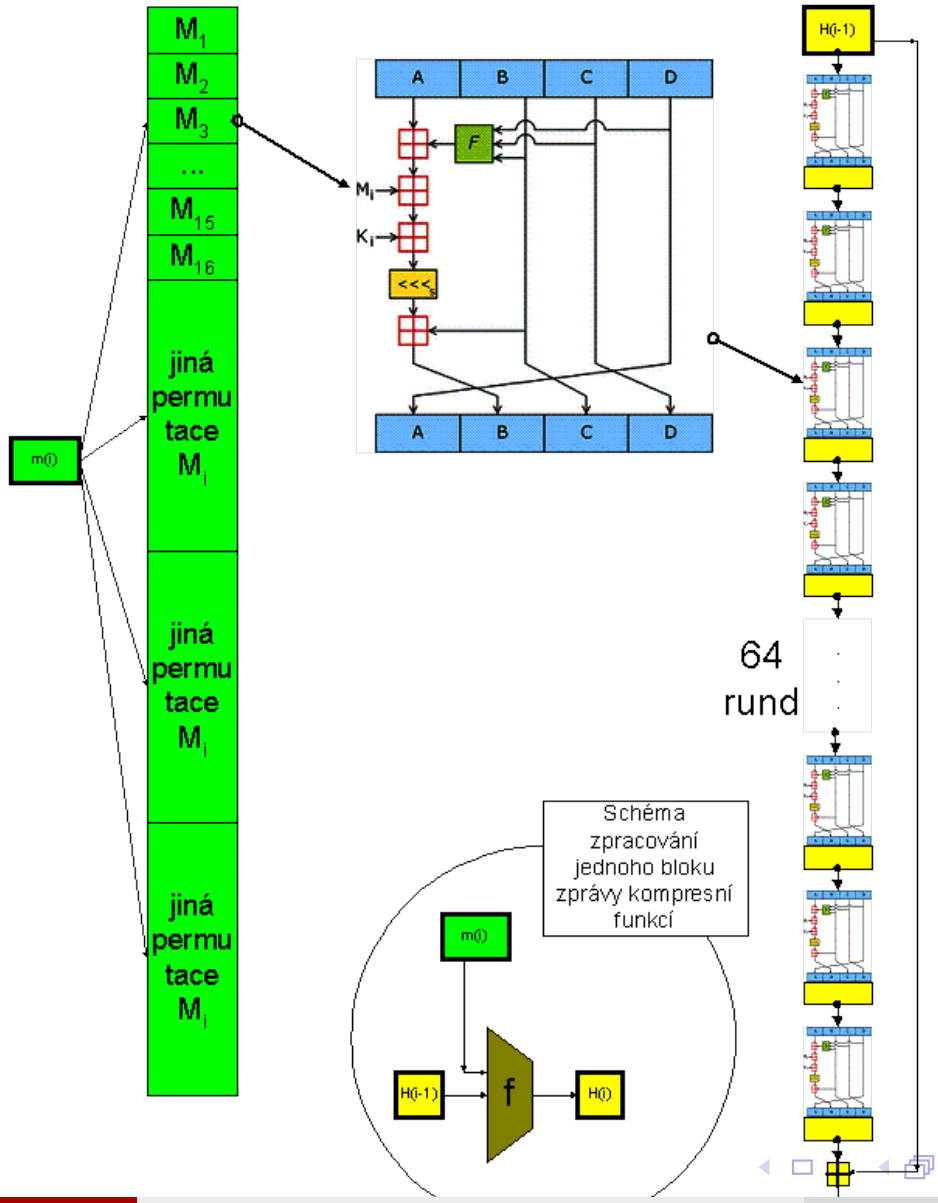
- R. Rivest 1991
- $n = 128$
- 128 bitů $\Rightarrow 64$ bitů pro birthday attack
- 2004 Wangová a kol. nalezení kolize za 8 hodin na superpočítači

MD5 (Message-Digest algorithm)

Známá hašovací funkce

- R. Rivest 1991
- $n = 128$
- 128 bitů $\Rightarrow$ 64 bitů pro birthday attack
- 2004 Wangová a kol. nalezení kolize za 8 hodin na superpočítači
- 2006 Klíma – kolize během jedné minuty na notebooku

MD5 (Message-Digest algorithm)



SHA (Secure Hash Algorithm)

Několik hašovacích funkcí

SHA (Secure Hash Algorithm)

Několik hašovacích funkcí

- NIST = National Institute of Standards and Technology

SHA (Secure Hash Algorithm)

Několik hašovacích funkcí

- NIST = National Institute of Standards and Technology
- FIPS = U.S. Federal Information Processing Standard

SHA (Secure Hash Algorithm)

Několik hašovacích funkcí

- NIST = National Institute of Standards and Technology
- FIPS = U.S. Federal Information Processing Standard
- 1993 SHA-0 (hned stažena)

SHA (Secure Hash Algorithm)

Několik hašovacích funkcí

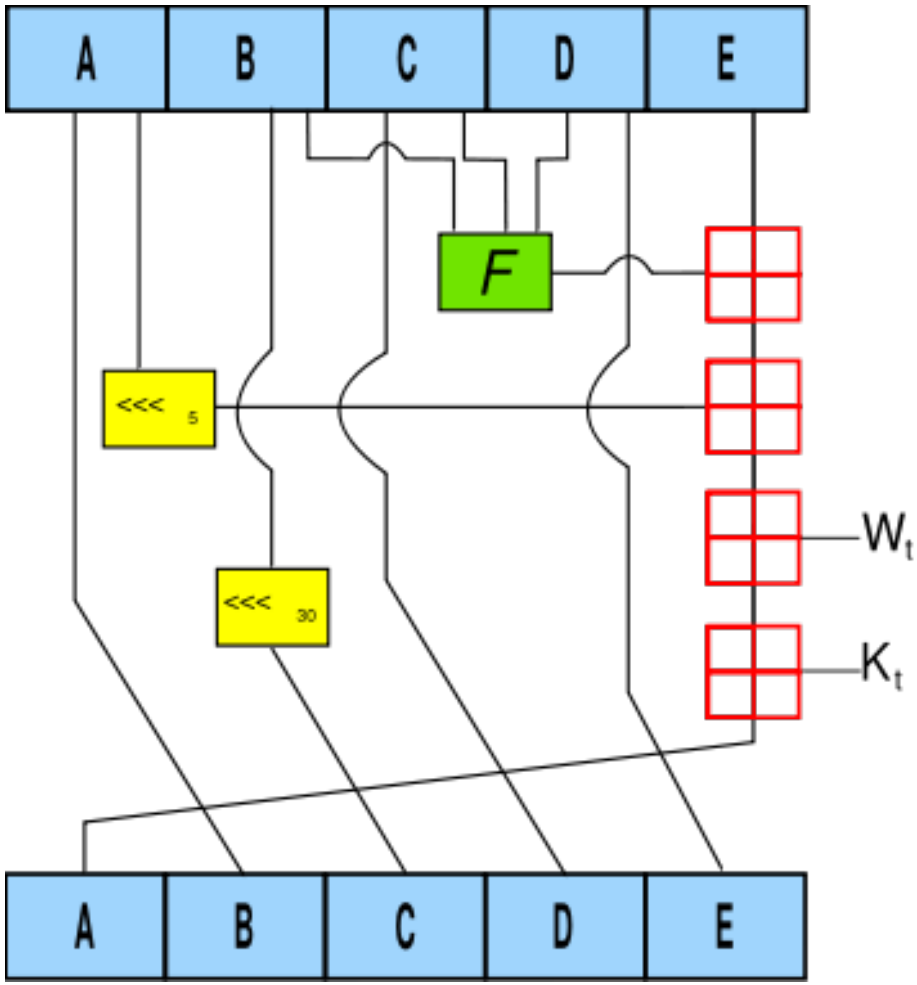
- NIST = National Institute of Standards and Technology
- FIPS = U.S. Federal Information Processing Standard
- 1993 SHA-0 (hned stažena)
- 1995 SHA-1, 160 bitů stejně jako SHA-0

SHA (Secure Hash Algorithm)

Několik hašovacích funkcí

- NIST = National Institute of Standards and Technology
- FIPS = U.S. Federal Information Processing Standard
- 1993 SHA-0 (hned stažena)
- 1995 SHA-1, 160 bitů stejně jako SHA-0
- 2001 SHA-2 – několik funcí, např. SHA-256, SHA-512

SHA-1



SHA-3

- už v roce 2005 známy jisté teoretické problémy iterativního postupu

SHA-3

- už v roce 2005 známy jisté teoretické problémy iterativního postupu
- možná vhodná změna konceptu

SHA-3

- už v roce 2005 známy jisté teoretické problémy iterativního postupu
- možná vhodná změna konceptu
- 2007 NIST vyhlásilo otevřenou soutěž na SHA-3

SHA-3

- už v roce 2005 známy jisté teoretické problémy iterativního postupu
- možná vhodná změna konceptu
- 2007 NIST vyhlásilo otevřenou soutěž na SHA-3
- podobný proces jako AES

SHA-3

- už v roce 2005 známy jisté teoretické problémy iterativního postupu
- možná vhodná změna konceptu
- 2007 NIST vyhlásilo otevřenou soutěž na SHA-3
- podobný proces jako AES
- nebude založeno na SHA-2

SHA-3

- už v roce 2005 známy jisté teoretické problémy iterativního postupu
- možná vhodná změna konceptu
- 2007 NIST vyhlásilo otevřenou soutěž na SHA-3
- podobný proces jako AES
- nebude založeno na SHA-2
- 2008 64 přihlášených, vybráno 51 do 1. kola

SHA-3

- už v roce 2005 známy jisté teoretické problémy iterativního postupu
- možná vhodná změna konceptu
- 2007 NIST vyhlásilo otevřenou soutěž na SHA-3
- podobný proces jako AES
- nebude založeno na SHA-2
- 2008 64 přihlášených, vybráno 51 do 1. kola
- 2009 14 kandidátů do 2. kola

SHA-3

- už v roce 2005 známy jisté teoretické problémy iterativního postupu
- možná vhodná změna konceptu
- 2007 NIST vyhlásilo otevřenou soutěž na SHA-3
- podobný proces jako AES
- nebude založeno na SHA-2
- 2008 64 přihlášených, vybráno 51 do 1. kola
- 2009 14 kandidátů do 2. kola
- 2010 5 finalistů

SHA-3

- už v roce 2005 známy jisté teoretické problémy iterativního postupu
- možná vhodná změna konceptu
- 2007 NIST vyhlásilo otevřenou soutěž na SHA-3
- podobný proces jako AES
- nebude založeno na SHA-2
- 2008 64 přihlášených, vybráno 51 do 1. kola
- 2009 14 kandidátů do 2. kola
- 2010 5 finalistů
- jaro 2012 „final SHA-3 Candidate Conference“

SHA-3

- už v roce 2005 známy jisté teoretické problémy iterativního postupu
- možná vhodná změna konceptu
- 2007 NIST vyhlásilo otevřenou soutěž na SHA-3
- podobný proces jako AES
- nebude založeno na SHA-2
- 2008 64 přihlášených, vybráno 51 do 1. kola
- 2009 14 kandidátů do 2. kola
- 2010 5 finalistů
- jaro 2012 „final SHA-3 Candidate Conference“
- vyhlášení vítěze později v roce 2012

SHA-3

Čeští kandidáti

- EDON-R

SHA-3

Čeští kandidáti

- EDON-R
  - autoři: Danilo Gligoroski, Rune Steinsmo Ødegård, Marija Mihova, Svein Johan Knapskog, Ljupco Kocarev, Aleš Drápal, Vlastimil Klíma

SHA-3

Čeští kandidáti

- EDON-R
  - autoři: Danilo Gligoroski, Rune Steinsmo Ødegård, Marija Mihova, Svein Johan Knapskog, Ljupco Kocarev, Aleš Drápal, Vlastimil Klíma
  - nedostal se do 2. kola

SHA-3

Čeští kandidáti

- EDON-R
  - autoři: Danilo Gligoroski, Rune Steinsmo Ødegård, Marija Mihova, Svein Johan Knapskog, Ljupco Kocarev, Aleš Drápal, Vlastimil Klíma
  - nedostal se do 2. kola
- BMW

SHA-3

Čeští kandidáti

- EDON-R
  - autoři: Danilo Gligoroski, Rune Steinsmo Ødegård, Marija Mihova, Svein Johan Knapskog, Ljupco Kocarev, Aleš Drápal, Vlastimil Klíma
  - nedostal se do 2. kola
- BMW
  - autoři: Danilo Gligoroski, Vlastimil Klíma, Svein Johan Knapskog, Mohamed El-Hadedy, Jørn Amundsen, Stig Frode Mjølsnes

SHA-3

Čeští kandidáti

- EDON-R
  - autoři: Danilo Gligoroski, Rune Steinsmo Ødegård, Marija Mihova, Svein Johan Knapskog, Ljupco Kocarev, Aleš Drápal, Vlastimil Klíma
  - nedostal se do 2. kola
- BMW
  - autoři: Danilo Gligoroski, Vlastimil Klíma, Svein Johan Knapskog, Mohamed El-Hadedy, Jørn Amundsen, Stig Frode Mjølsnes
  - nedostal se do 3. kola mezi finalisty

SHA-3

Čeští kandidáti

- EDON-R
  - autoři: Danilo Gligoroski, Rune Steinsmo Ødegård, Marija Mihova, Svein Johan Knapskog, Ljupco Kocarev, Aleš Drápal, Vlastimil Klíma
  - nedostal se do 2. kola
- BMW
  - autoři: Danilo Gligoroski, Vlastimil Klíma, Svein Johan Knapskog, Mohamed El-Hadedy, Jørn Amundsen, Stig Frode Mjølsnes
  - nedostal se do 3. kola mezi finalisty
- Oba alg. velmi rychlé

SHA-3

Čeští kandidáti

- EDON-R
  - autoři: Danilo Gligoroski, Rune Steinsmo Ødegård, Marija Mihova, Svein Johan Knapskog, Ljupco Kocarev, Aleš Drápal, Vlastimil Klíma
  - nedostal se do 2. kola
- BMW
  - autoři: Danilo Gligoroski, Vlastimil Klíma, Svein Johan Knapskog, Mohamed El-Hadedy, Jørn Amundsen, Stig Frode Mjølsnes
  - nedostal se do 3. kola mezi finalisty
- Oba alg. velmi rychlé
- EDON-R jisté slabiny

SHA-3 2. kolo

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

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

Obrázek: Kandidáti ve 2. kole a rychlost

SHA-3 finále

Finalisté

- BLAKE, Grøstl, JH, Keccak a Skein

SHA-3 finále

Finalisté

- BLAKE, Grøstl, JH, Keccak a Skein

Rozhodování

- „Security: We preferred to be conservative about security, and in some cases did not select algorithms with exceptional performance, largely because *something about them made us 'nervous'*, even though we knew of no clear attack against the full algorithm.“

Obsah

Závěrem

Děkuji za pozornost