Hashovací funkce

Andrew Kozlík

KA MFF UK

Hashovací funkce

- *Hashovací funkce* je zobrazení $h : \{0,1\}^* \to \{0,1\}^n$.
- Typicky $n \in \{128, 160, 192, 224, 256, 384, 512\}$.
- Obraz $h(x)$ nazýváme *otisk*, *hash* nebo *digest* prvku x.
- Jestliže $x \neq x'$ a $h(x) = h(x')$, říkáme, že pár (x, x') je *kolize* funkce h.
- Použití v informatice:
  - Odhalení duplicit.
  - Rychlá lokalizace záznamů v databázi.
  - Kontrola, že nedošlo k *náhodnému* poškození dat.
- Požadavky v informatice:
  - Chceme, aby kolize byly nepravděpodobné v dané aplikaci.
  - Chceme, aby podobné řetězce měly odlišný obraz.
- Požadavky v kryptografii jsou přísnější než v informatice!

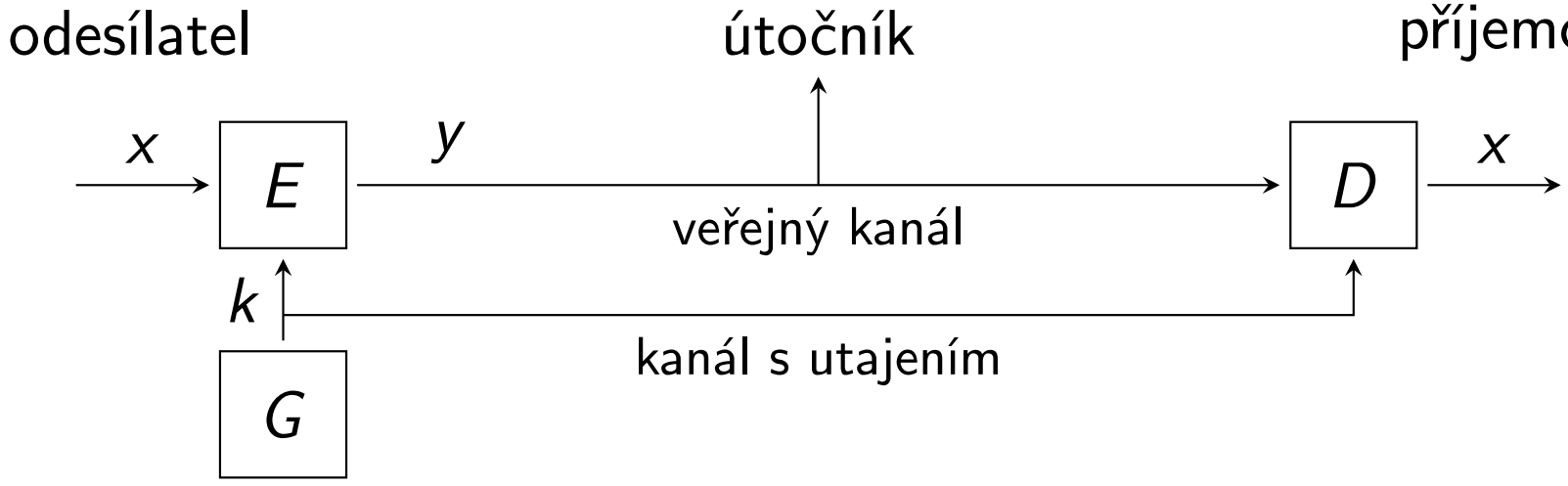
Příklady použití hashovacích funkcí v kryptografii

- Zajištění integrity.
- Ukládání hesel na straně ověřovatele hesla.
- Vytváření závazků k utajovaným datům.

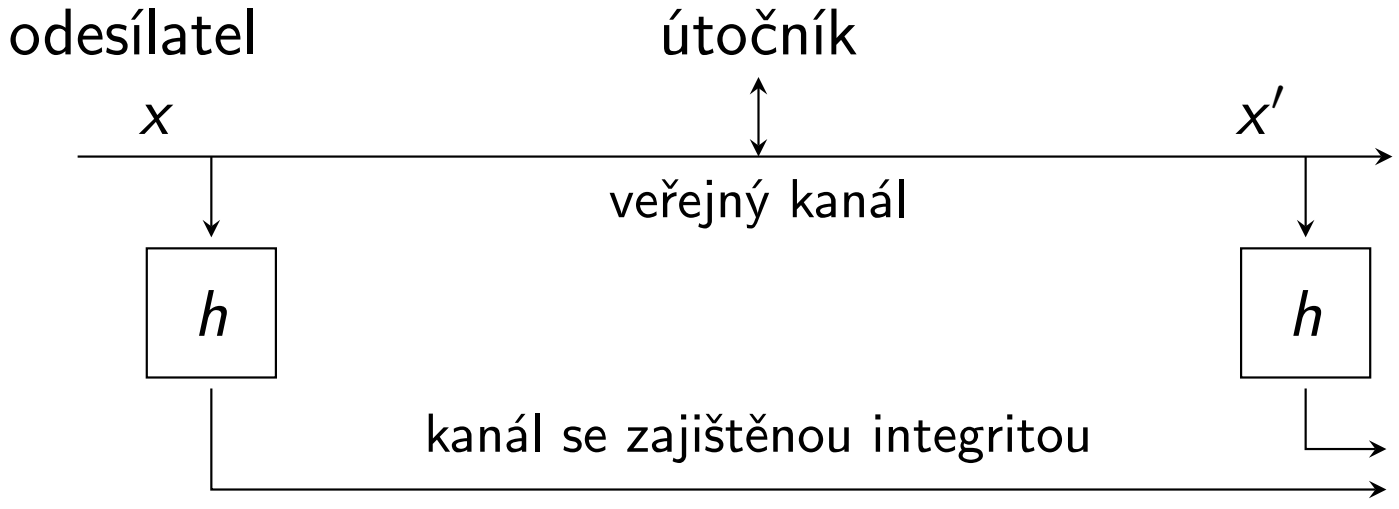
Ukládání hesel

- Na serveru se ukládá trojice $\big(\mathit{login}, \mathit{sůl}, h(\mathit{heslo} \,\|\, \mathit{sůl})\big)$, kde *sůl* je řetězec vygenerovaný náhodně pro každý login.
- Je-li odcizena databáze s takto uloženými hesly, útočník nemůže hesla jednoznačně rekonstruovat.
- Účel soli:
  - Znemožňuje útok pomocí předpočítané tabulky pro zpětný překlad otisků.
  - Mají-li dva uživatelé stejné heslo, není to vidět.

Zajištění utajení pomocí šifry



Zajištění integrity pomocí hashovací funkce



Jestliže $h(x') = h(x)$, usoudíme, že $x' = x$.

Vytváření závazků k utajovaným datům

Příklad

Házení mincí po telefonu.

- A si zvolí stranu mince x_A.
- B hodí mincí, výsledkem hodu je x_B.
- „$x \in_{\mathcal{R}} M$“ značí „x je náhodně zvolený prvek množiny M“.
- k je bezpečnostní parametr, např. $k = 128$.

Postup:

1.	A:	$x_A \in_{\mathcal{R}} \{0,1\}, \quad r \in_{\mathcal{R}} \{0,1\}^k, \quad z = h(x_A \,\|\, r)$	
	B:	$x_B \in_{\mathcal{R}} \{0,1\}$	
2.	$A \to B$:	z	[závazek strany A]
3.	$B \to A$:	x_B	[odhalení výsledku hodu]
4.	$A \to B$:	(x_A, r)	[odhalení závazku]
5.	B:	Kontrola $h(x_A \,\|\, r) = z$.	[ověření poctivosti A]
6.	A, B:	A vyhrává, právě když $x_A = x_B$.	

Typy útoků na hashovací funkce

(1) Nalezení prvního vzoru:
Je dáno y, cílem je najít x takové, že $h(x) = y$.

(2) Nalezení druhého vzoru:
Je dáno x, cílem je najít x' takové, že $h(x) = h(x')$ a $x \neq x'$.

(3) Nalezení kolize:
Cílem je najít pár (x, x') takový, že $h(x) = h(x')$ a $x \neq x'$.

- Říkáme, že hashovací funkce je *odolná* proti útoku, jestliže jeho provedení přesahuje výpočetní možnosti útočníka.
- Funkce, která je odolná proti (1) se nazývá *jednosměrná*.
- Funkce, která je odolná proti (3) se nazývá *kolizivzdorná*.
- Je-li funkce odolná proti (3), pak je také odolná proti (2).

Typy útoků na hashovací funkce

(1) Nalezení prvního vzoru:
Je dáno y, cílem je najít x takové, že $h(x) = y$.

(2) Nalezení druhého vzoru:
Je dáno x, cílem je najít x' takové, že $h(x) = h(x')$ a $x \neq x'$.

(3) Nalezení kolize:
Cílem je najít pár (x, x') takový, že $h(x) = h(x')$ a $x \neq x'$.

- Pro zajištění integrity potřebujeme odolnost proti (2).
- Pro ukládání hesel potřebujeme odolnost proti (1).
- Pro vytváření závazků potřebujeme odolnost proti:
  - Odhalení, k čemu se subjekt zavázal (1).
  - Vytváření falešných závazků (3).

Nalezení prvního vzoru hrubou silou

- Máme funkci $h : \{0,1\}^* \to \{0,1\}^n$ a obraz $y \in \{0,1\}^n$. Cílem je najít x takové, že $h(x) = y$.
- Vybereme libovolné $x \in \{0,1\}^*$ a ověříme, zda $h(x) = y$. Opakujeme tak dlouho, dokud se nedostaví úspěch.
- Pravděpodobnost, že se úspěch dostaví až na i-tý pokus:

$$\left(1 - \frac{1}{2^n}\right)^{i-1} \frac{1}{2^n}.$$

- Průměrný počet volání funkce h, než se podaří najít vzor:

$$\sum_{i=1}^{\infty} i(1 - 2^{-n})^{i-1}\, 2^{-n} = \ldots$$

Nalezení prvního vzoru hrubou silou

- K sečtení řady budeme potřebovat pomocný vzoreček: Pro $x \in (-1, 1)$ platí

$$\sum_{i=1}^{\infty} i x^{i-1} = \sum_{i=0}^{\infty} i x^{i-1} = \left(\sum_{i=0}^{\infty} x^i \right)' = \left(\frac{1}{1-x} \right)' = \frac{1}{(1-x)^2}.$$

Řada $\sum_{i=0}^{\infty} x^i$ totiž konverguje stejnoměrně na libovolném uzavřeném podintervalu intervalu $(-1, 1)$.

- Průměrný počet volání funkce h:

$$\sum_{i=1}^{\infty} i \left(1 - 2^{-n}\right)^{i-1} 2^{-n} = \frac{2^{-n}}{\left(1 - \left(1 - 2^{-n}\right)\right)^2} = \frac{2^{-n}}{2^{-2n}} = 2^n.$$

Nalezení kolize hrubou silou

- **Algoritmus:**
  Postupně vytvoříme množinu dvojic tvaru $(h(x), x)$.
  1. $S = \emptyset$.
  2. Vyber libovolné $x \in \{0,1\}^*$ (bez opakování).
  3. Jestliže $\exists z : (h(x), z) \in S$, pak dvojice (x, z) je kolize.
     Jinak vlož $(h(x), x)$ do S a pokračuj krokem 2.

- Množina S by se implementovala jako asociativní pole (C++ map, Python dictionary).

- Jaká je časová složitost hledání kolize hrubou silou?

- Narozeninový paradox:
  Je-li v místnosti 23 lidí, pak pravděpodobnost, že alespoň dva z nich mají narozeniny ve stejný den, je větší než $\frac{1}{2}$.

Tvrzení (o narozeninovém paradoxu)

Jestliže z N-prvkové množiny zvolíme náhodně k-krát prvek nezávisle s rovnoměrným rozdělením pravděpodobnosti, kde $k \leq N$, pak pravděpodobnost, že alespoň jeden prvek bude vybrán více než jednou, je

$$p \;=\; 1 - \frac{N!}{(N-k)!\,N^k} \;\geq\; 1 - \exp\left(\frac{-k(k-1)}{2N}\right).$$

Důkaz.

Pravděpodobnost, že žádný prvek nebude vybrán opakovaně:

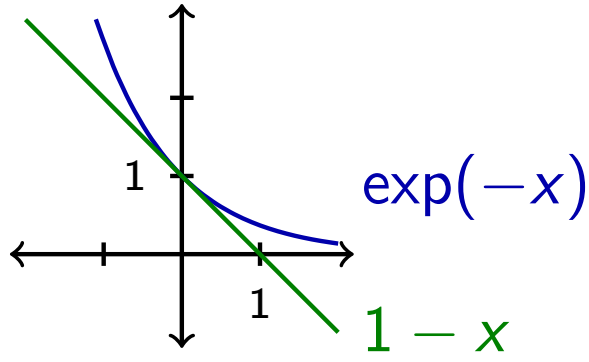
$$1 - p = \frac{N(N-1)(N-2)\cdots(N-k+1)}{N^k}$$

$$= 1\left(1 - \frac{1}{N}\right)\left(1 - \frac{2}{N}\right)\cdots\left(1 - \frac{k-1}{N}\right)$$

Kolika způsoby lze vybrat k různých prvků z N (včetně pořadí).

Důkaz (pokračování).

Funkce $\exp(-x)$ se dá odhadnout zespoda tečnou v bodě $(0, 1)$:



Po aplikaci nerovnosti $1 - x \leq \exp(-x)$ dokončíme důkaz:

$$\begin{aligned} 1 - p &= \left(1 - \frac{1}{N}\right)\left(1 - \frac{2}{N}\right) \cdots \left(1 - \frac{k-1}{N}\right) \\ &\leq \exp\left(-\frac{1}{N}\right) \exp\left(-\frac{2}{N}\right) \cdots \exp\left(-\frac{k-1}{N}\right) \\ &= \exp\left(-\frac{1}{N}\sum_{i=1}^{k-1} i\right) = \exp\left(\frac{-k(k-1)}{2N}\right). \end{aligned}$$

□

Nalezení kolize hrubou silou

- Máme hashovací funkci $h : \{0,1\}^* \to \{0,1\}^n$, čili $N = 2^n$.
- Chceme, aby pravděpodobnost, že algoritmus najde kolizi, byla alespoň p.
- Stačí, aby počet pokusů k splňoval

$$1 - \exp\left(\frac{-k(k-1)}{2N}\right) \geq p.$$

Tuto podmínku můžeme ekvivalentními úpravami vyjádřit:

$$\begin{aligned} \exp\left(\frac{-k(k-1)}{2N}\right) &\leq 1 - p \\ \frac{-k(k-1)}{2N} &\leq \ln(1-p) \\ k(k-1) &\geq -2N\ln(1-p) \end{aligned}$$

Nalezení kolize hrubou silou

- Postačující podmínkou pro nalezení kolize s pravděpodobností úspěchu alespoň p je tedy

$$k^2 - k + 2N \ln(1-p) \geq 0.$$

Čili

$$k \geq \frac{1 + \sqrt{1 - 4 \cdot 2N \ln(1-p)}}{2} \sim \sqrt{-2N \ln(1-p)}$$

$$= 2^{n/2} \underbrace{\sqrt{2 \ln \frac{1}{1-p}}}_{\alpha},$$

kde k je počet volání hashovací funkce.

- K nalezení kolize SHA-256 tak stačí řádově jen 2^{128} volání.
- Například pro $p = \frac{1}{2}$ je $\alpha \approx 1{,}2$ a pro $p = 0{,}99$ je $\alpha \approx 3$.

Konstrukce hashovacích funkcí

Merkleovo-Damgårdovo schéma

- Používá se v MD5, SHA-1 i SHA-2.
- Je to něco jako operační režim.
- Stavebním kamenem je kompresní funkce

$$f : \{0,1\}^s \times \{0,1\}^b \to \{0,1\}^s.$$

- b je velikost bloku v bitech.
- s je velikost vnitřního stavu v bitech.
- Jestliže $f(S,B) = f(S',B')$ a přitom $S \neq S'$ nebo $B \neq B'$, pak říkáme, že $((S,B),(S',B'))$ je *kolize* funkce f.

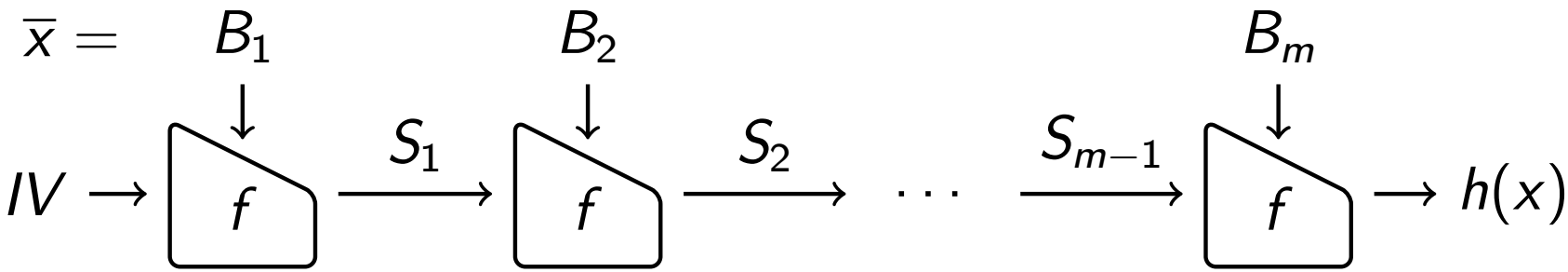
Merkleovo-Damgårdovo schéma

1. Ke zprávě x se připojí výplň, která
   - obsahuje délku zprávy x a
   - vyplňuje zprávu na násobek čísla b.

   MD5, SHA-1, SHA-224 a SHA-256 používají $b = 512$ a

   $$\overline{x} = x \,\|\, 1\,0\,0 \ldots 0 \,\|\, \ell_{64}(x),$$

   kde $\ell_{64}(x)$ je délka x v bitech vyjádřená jako 64bitové číslo.
2. $\overline{x}$ se rozdělí na bloky $B_1, \ldots, B_m$ délky b.
3. $S_0 := IV$ (konstanta pro danou hashovací funkci)
   Pro $i = 1, \ldots, m$:
   $$S_i := f(S_{i-1}, B_i)$$
   $h(x) := S_m$



Věta (Merkle-Damgård, 1989)

Jestliže kompresní funkce f je kolizivzdorná, pak h je kolizivzdorná.

Důkaz.

- Ukážeme, že z kolize (x, x') pro h lze sestrojit kolizi pro f.
- Nechť $\overline{x} = B_1 \| \ldots \| B_m$, $\overline{x'} = B'_1 \| \ldots \| B'_{m'}$ a $S_1, \ldots, S_m$, resp. $S'_1, \ldots, S'_{m'}$ jsou vnitřní stavy.
- Máme $h(x) = h(x')$, čili $f(S_{m-1}, B_m) = f(S'_{m'-1}, B'_{m'})$.
- Nyní jsou dvě možnosti:
  1. $(S_{m-1}, B_m) \neq (S'_{m'-1}, B'_{m'}) \quad \Rightarrow \quad$ Máme kolizi pro f!
  2. $S_{m-1} = S'_{m'-1}$ a zároveň $B_m = B'_{m'}$

     Poslední blok obsahuje délku x resp. x'.

     $\Rightarrow \quad \ell(x) = \ell(x') \quad \Rightarrow \quad m' = m.$

Důkaz (pokračování).

- Pokud jsme ještě nenašli kolizi pro f, tak máme:
  $S_{m-1} = S'_{m-1}$, čili $f(S_{m-2}, B_{m-1}) = f(S'_{m-2}, B'_{m-1})$.

- Jsou opět dvě možnosti:
  1. $(S_{m-2}, B_{m-1}) \neq (S'_{m-2}, B'_{m-1})$ $\Rightarrow$ Máme kolizi pro f!
  2. $S_{m-2} = S'_{m-2}$ a zároveň $B_{m-1} = B'_{m-1}$.

- Takto pokračujeme dále.

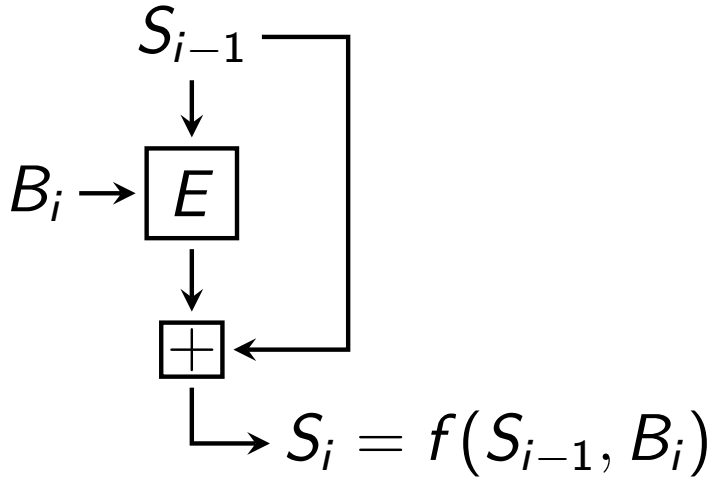
- Pro nějaké $i \in \{1, \ldots, m\}$ musí nastat $B_i \neq B'_i$, protože $x \neq x'$, a tím pádem případ 1. □

Porovnání MD5, SHA-1 a SHA-2

		Velikosti (v bitech)		
	Algoritmus	Výstup	Vnitřní stav	Blok
	MD5	128	128	512
	SHA-1	160	160	512
SHA-2	SHA-224	224	256	512
	SHA-256	256	256	512
	SHA-384	384	512	1024
	SHA-512	512	512	1024
	SHA-512/224	224	512	1024
	SHA-512/256	256	512	1024

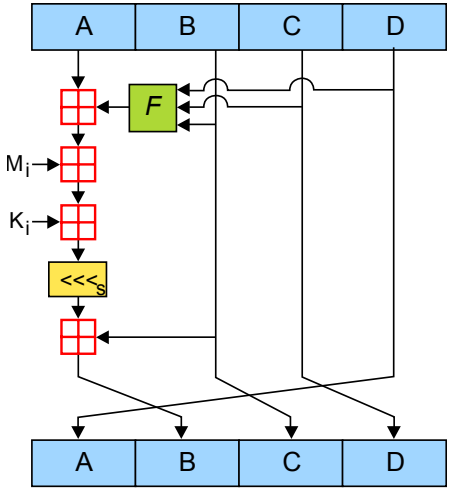
Daviesovo-Meyerovo schéma kompresní funkce

- Nechť E je šifra.
- Definujeme $f(S_{i-1}, B_i) = E_{B_i}(S_{i-1}) \boxplus S_{i-1}$.
- B_i vstupuje do šifry jako klíč.
- S_{i-1} vstupuje do šifry jako otevřený text.
- $\boxplus$ je celočíselné sčítání po slovech modulo 2^{32}.
- MD5, SHA-1 i SHA-2 využívají toto schéma.



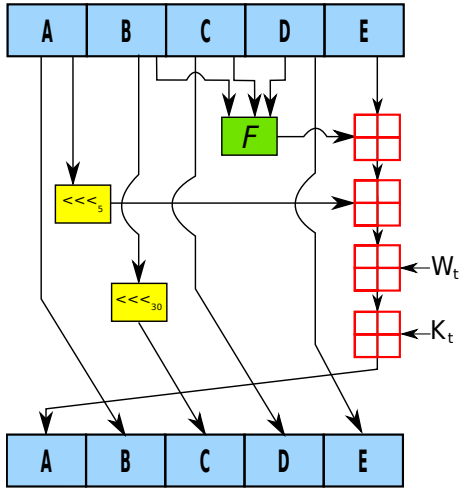
Jedna runda šifrovací funkce v Daviesově-Meyerově schématu

MD5



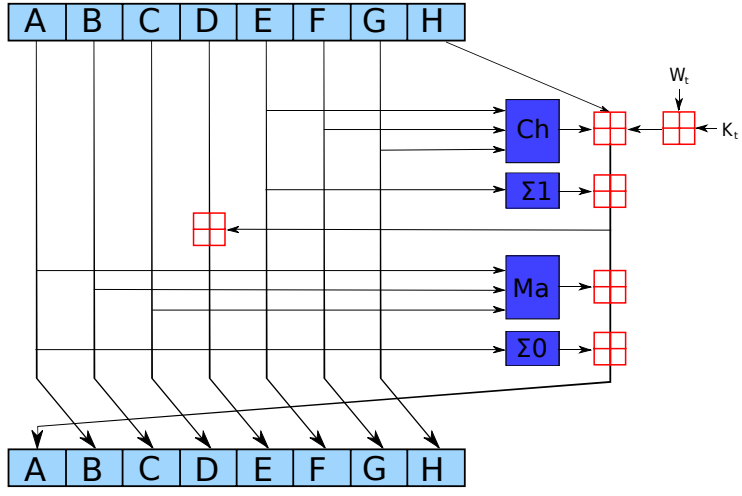
64 rund

SHA-1



80 rund

SHA-2

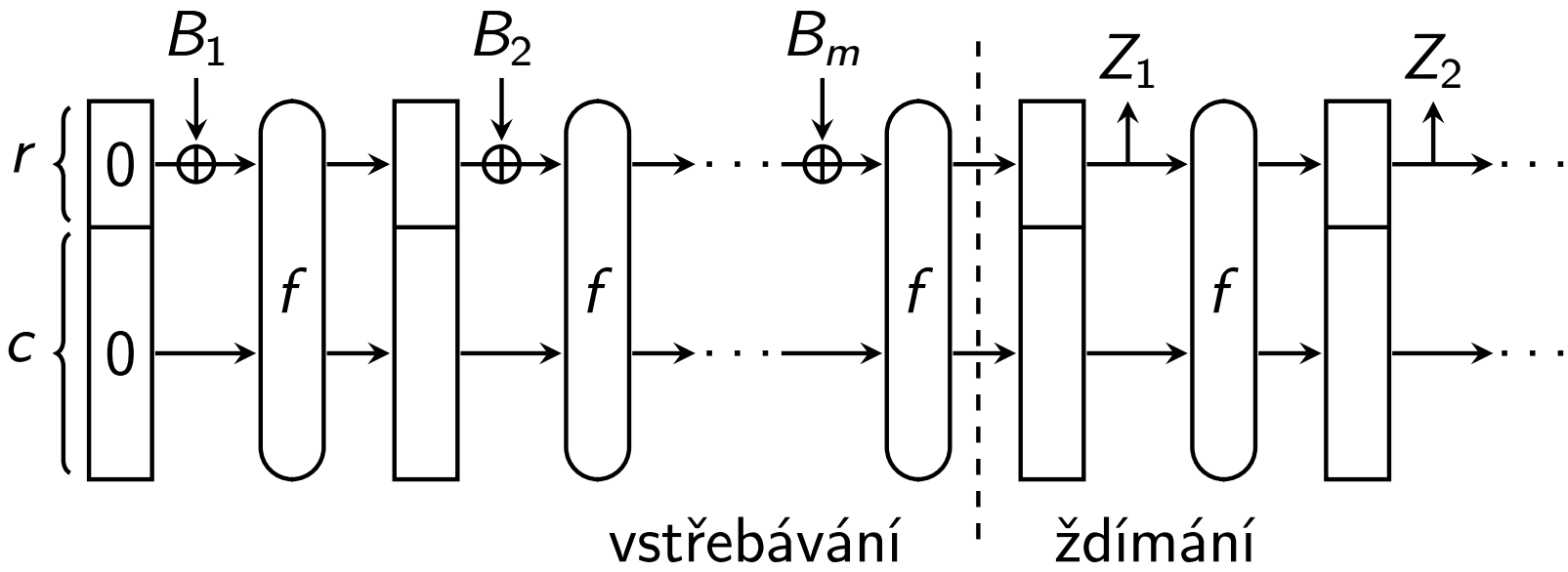


64 nebo 80 rund

Zdroj: Wikipedia.org

Houbovité konstrukce

- Princip: 1. Zpráva se vstřebá do houby (vnitřního stavu).
  2. Z houby se potom vyždímá otisk zprávy.
- r: bitrate, velikost bloku v bitech.
- c: capacity, velikost vnitřní části stavu houby.
- b: $= r + c$, velikost stavu houby.
- f: náhodná b-bitová bijekce (permutace 2^b stavů).
- Zpráva se vyplní a rozdělí na r-bitové bloky $B_1, \ldots, B_m$.
- Vyždímá se $k = \lceil \frac{n}{r} \rceil$ bloků $Z_1, \ldots, Z_k$ a ořeže na n bitů.



Houbovité konstrukce

- Houbovité konstrukce nabízejí variabilní délku výstupu, tzv. *extendable output function* (XOF), $h : \{0,1\}^* \to \{0,1\}^*$.
- Lze použít jako proudovou šifru $y = h(key \,\|\, IV) \oplus x$.
- Složitost útoků hrubou silou na hashovací funkci s houbovitou konstrukcí:
  - Nalezení prvního vzoru: $\min(2^{n-r} + 2^{c/2}, 2^{c-1} + 2^{c/2}, 2^n)$.
  - Nalezení druhého vzoru: $2^{\min(c/2,n)}$.
  - Nalezení kolize: $2^{\min(c/2,n/2)}$.

Hashovací funkce SHA-3

- Nejnovější hashovací funkce standardizovaná NISTem.
- Byla vybrána v rámci veřejné soutěže v letech 2007 až 2012.
- Vítězem byla funkce KECCAK [kečak] autorů G. Bertoni, J. Daemen, M. Peeters a G. Van Assche.
- Používá houbovitou konstrukci s $b = 1600$ a má 4 varianty:

Název	r	c	n
SHA3-224	1152	448	224
SHA3-256	1088	512	256
SHA3-384	832	768	384
SHA3-512	576	1024	512

- Funkce f sestává z posloupnosti bitových rotací a bitových operací XOR, AND a NOT.
- Za zprávu se přidají dva bity 01, které indikují, že se jedná o SHA-3. Standard totiž definuje ještě několik dalších funkcí.
- Následně se připojí výplň tvaru 10^*1. Koncová 1 odliší různé varianty, tj. SHA3-256("abc") $\neq$ SHA3-384("abc").

Nalezení kolize u houbovité konstrukce

Tvrzení

Složitost nalezení kolize hrubou silou pro hashovací funkci s houbovitou konstrukcí je $2^{\min(c/2,n/2)}$ *volání hashovací funkce.*

Důkaz.

Stačí najít jedno z následujících:

a. Kolizi ve výstupu hashovací funkce ($2^{n/2}$).

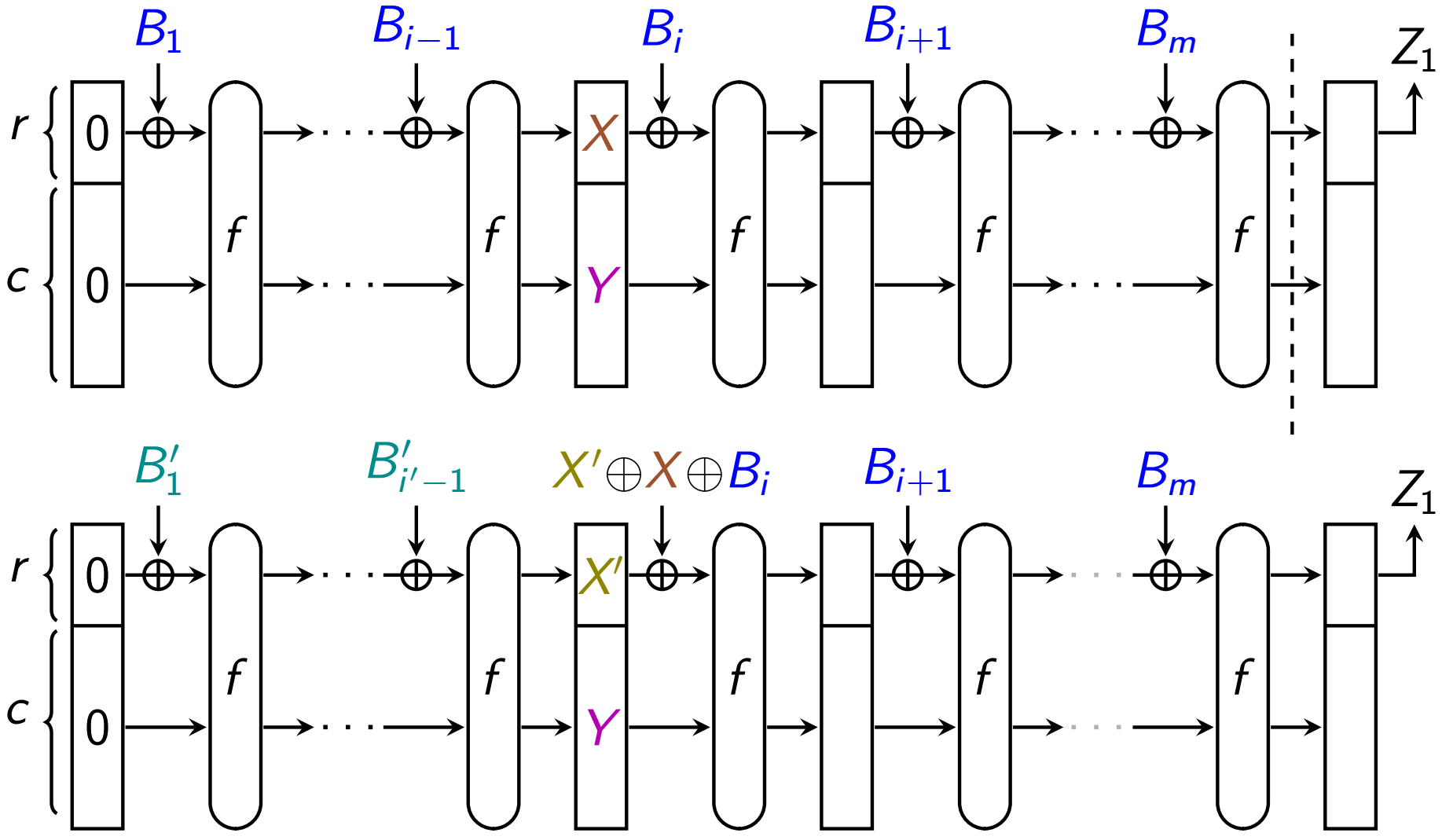
b. Kolizi ve vnitřní části stavu houby ($2^{c/2}$), tj. dva vstupy
   - $B_1, \ldots, B_{i-1}$, který vede do stavu (X, Y), a
   - $B'_1, \ldots, B'_{i-1}$, který vede do stavu (X', Y).

V případě b. zvolíme zbytek vstupu $B_i, \ldots, B_n$ libovolně s validní výplní, a potom oba následující vstupy

- $B_1, \ldots, B_{i-1}, B_i, B_{i-1}, \ldots, B_n$,
- $B'_1, \ldots, B'_{i-1}, X' \oplus X \oplus B_i, B_{i-1}, \ldots, B_n$,

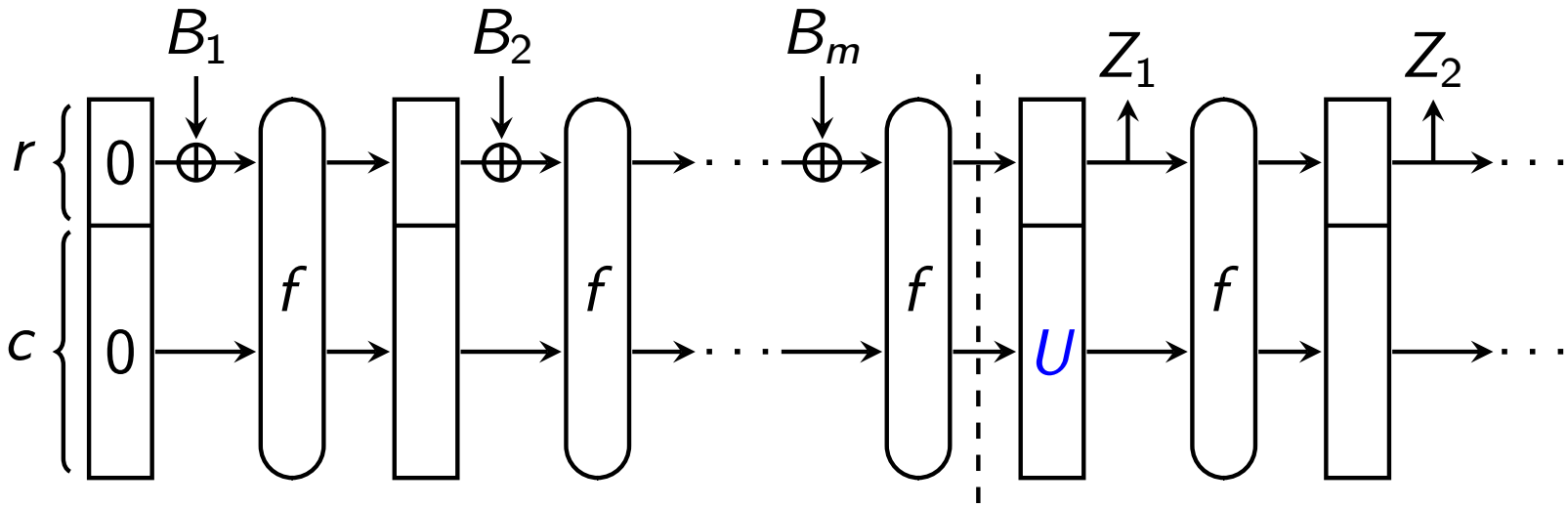
vedou do stejného stavu.

Nalezení kolize u houbovité konstrukce



Nalezení prvního vzoru u houbovité konstrukce

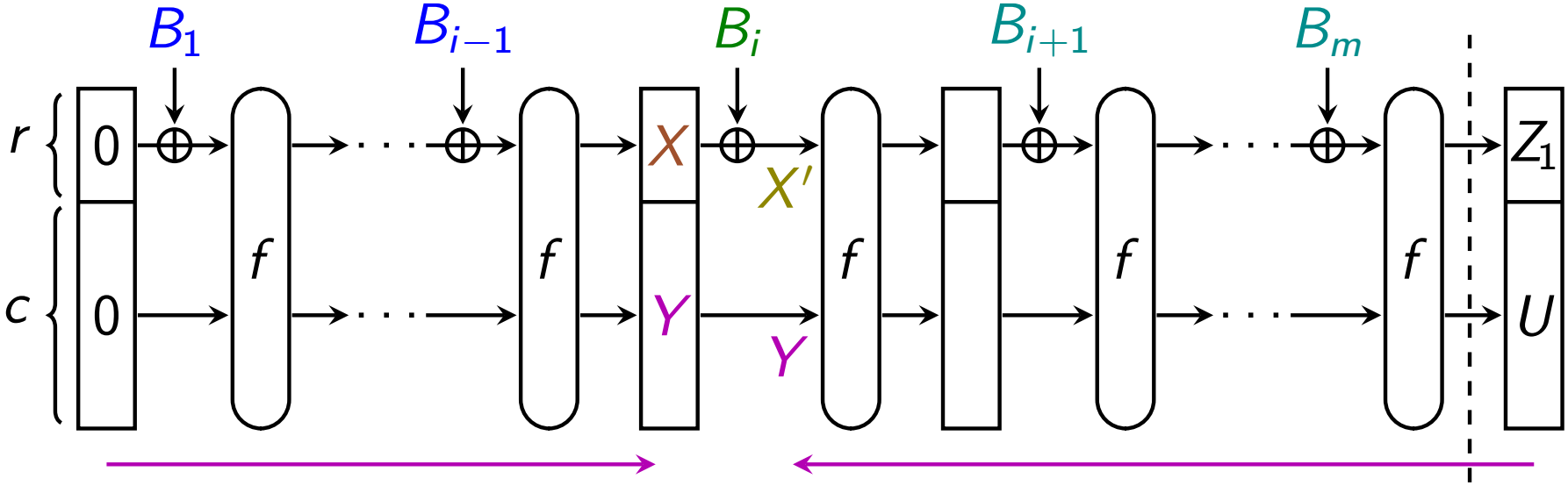
- **Problém:**
  - Je dáno $Z \in \{0,1\}^n$, kde $Z = \mathrm{head}_n(Z_1 \parallel Z_2 \parallel \cdots)$.
  - Máme najít $B_1, \ldots, B_m$, aby výsledkem bylo Z.
- Jestliže $n \leq r$ (ždímá se jen Z_1),
  pak U může mít libovolnou hodnotu.
- Jestliže $r < n < c + r$ (ždímá se navíc $Z_2, Z_3, \ldots$),
  pak U najdeme hrubou silou průměrně v 2^{n-r} operacích.
- Jestliže $n \geq c + r$ a U existuje,
  pak U najdeme hrubou silou průměrně v 2^{c-1} operacích.



Nalezení vzoru ke stavu (Z_1, U)

- Předpokládejme, že f je bijekce.
  Potom můžeme postupovat meet-in-the-middle útokem.
- Hodnoty $B_1, \ldots, B_{i-1}$ a $B_{i+1}, \ldots, B_m$ hledáme hrubou silou, aby vyšla kolize v posledních c bitech $i$tého stavu houby.
- Jinými slovy hledáme bloky vstupu, aby
  - $B_1, \ldots, B_{i-1}$ uvedly houbu ze stavu $(0, 0)$ do (X, Y), a
  - $B_{i+1}, \ldots, B_m$ uvedly houbu ze stavu $f(X', Y)$ do (Z_1, U),

  pro nějaké $X, X' \in \{0,1\}^r$ a $Y \in \{0,1\}^c$.
- Blok B_i potom dopočítáme tak, aby $X \oplus B_i = X'$.



Nalezení vzoru ke stavu (Z_1, U)

- Mějme dáno $Z_1 \in \{0,1\}^r$ a $U \in \{0,1\}^c$.
- Pro přehlednost ukážeme algoritmus jen pro $m = 3$. Zobecnění na více bloků je triviální.
- Pokud $r \geq c/2$, což např. pro SHA3 platí, pak stačí $m = 3$.
- **Algoritmus:**
  1. $S := \emptyset$.
  2. Pro $k \in \{1, \ldots, 2^{c/2}\}$:
     - Nechť $B_1 \in \{0,1\}^r$ je binární zápis čísla k.
     - Spočti $(X, Y) = f(B_1, 0)$.
     - Vlož (Y, B_1) do S.
  3. Zvol $B_3 \in \{0,1\}^r$ libovolně s validní výplní.
  4. Spočti $(X', Y') = f^{-1}\big(f^{-1}(Z_1, U) \oplus (B_3, 0)\big)$.
  5. Jestliže $\exists B_1 : (Y', B_1) \in S$, pak
     - spočti $(X, Y) = f(B_1, 0)$ a
     - vrať B_1, B_2, B_3, kde $B_2 = X \oplus X'$.
  6. Jinak opakuj postup od kroku 3.

Ověření korektnosti algoritmu

- Máme:
  - $(X, Y) = f(B_1, 0)$.
  - $X' = X \oplus B_2$.
  - $(X', Y') = f^{-1}\big(f^{-1}(Z_1, U) \oplus (B_3, 0)\big)$, kde $Y = Y'$.

- Potom pro B_1, B_2, B_3 platí:

$$
\begin{aligned}
& f(f(f(B_1, 0) \oplus (B_2, 0)) \oplus (B_3, 0)) \\
= {} & f(f((X, Y) \oplus (B_2, 0)) \oplus (B_3, 0)) \\
= {} & f(f(X', Y) \oplus (B_3, 0)) \\
= {} & f(f(X', Y') \oplus (B_3, 0)) \\
= {} & (Z_1, U)
\end{aligned}
$$

Ověření úspěšnosti algoritmu

- V množině S se mohou vyskytovat kolize, čili $|S| \leq 2^{c/2}$, ale jejich relativní počet je při této velikosti zanedbatelný.
- Pravděpodobnost, že algoritmus uspěje v kroku 5:

$$\frac{|S|}{2^c} \approx \frac{2^{c/2}}{2^c} = \frac{1}{2^{c/2}}.$$

- Pravděpodobnost, že algoritmus sponči nejpozději po $\alpha \cdot 2^{c/2}$ opakováních kroku 5:

$$1 - \left(1 - \frac{1}{2^{c/2}}\right)^{\alpha \cdot 2^{c/2}} \approx 1 - e^{-\alpha}.$$

- Například, pro $\alpha = 3$ máme pravděpodobnost úspěchu 0,95.

Složitost nalezení vzoru u houbovité konstrukce

- Složitost nalezení vzoru ke stavu (Z_1, U):
  - V kroku 2 máme $2^{c/2}$ volání f.
  - Ve zbytku algoritmu máme $\alpha \cdot 2^{c/2}$ volání f^{-1}.
  - To celkově odpovídá $(\alpha + 1) \cdot 2^{c/2}$ volání f, kde α je malé.
  - Samotná hashovací funkce volá f třikrát.
  - Složitost je tedy řádově $2^{c/2}$ volání hashovací funkce h.
- K nalezení prvního nebo druhého vzoru můžeme použít i generický algoritmus, který má složitost 2^n volání h.
- Složitost nalezení druhého vzoru:
  - V tomto případě (Z_1, U) známe.
  - Složitost je tedy $\min(2^{c/2}, 2^n)$ volání h.
- Složitost nalezení prvního vzoru:
  - Musíme navíc najít (Z_1, U), tj. $\min(2^{n-r}, 2^{c-1})$ volání h.
  - Složitost je tedy $\min(2^{n-r} + 2^{c/2}, 2^{c-1} + 2^{c/2}, 2^n)$ volání h.

Hashování hesel podrobněji

- Od hashovacích funkcí zpravidla požadujeme vysokou rychlost vyhodnocení.
- Hashování hesel je však specifické tím, že chceme naopak:
  - vysokou *časovou* náročnost vyhodnocení, abychom znemožnily útok hrubou silou nebo slovníkový útok,
  - vysokou *paměťovou* náročnost vyhodnocení, abychom znemožnily paralelizaci útoku např. na grafických kartách.
- Používáme speciální hashovací funkce, tzv. *funkce pro odvození klíče* (KDF, key derivation function):
  - PBKDF2 (RFC 8018 - PKCS #5)
  - bcrypt
  - scrypt (RFC 7914)
  - Argon2 (vítěz Password Hashing Competition 2015)
- Tyto funkce provádějí iterované hashování hesla.

Hashování hesel podrobněji

- Vstupem KDF je zpravidla heslo, sůl a parametr obtížnosti (např. počet iterací).
- Při použití KDF se tedy na serveru ukládá čtveřice: $(\mathit{login},\ \mathit{sůl},\ \mathit{počet\ iterací},\ \mathit{kdf}(\mathit{heslo}, \mathit{sůl}, \mathit{počet\ iterací}))$
- Útok hrubou silou na alfanumerické heslo délky 8.
  - Heslo uloženo pomocí SHA3:
    - Průměrná doba útoku: **1,5 roku** na běžném CPU.
  - Heslo uloženo pomocí KDF:
    - Předpokládejme, že počet iterací byl zvolen tak, aby vyhodnocení trvalo 1 vteřinu na běžném CPU.
    - Průměrná doba útoku na stejném CPU: **3,46 milionů let**.
    - Průměrná doba útoku při zohlednění Mooreova zákona (zdvojnásobení výpočetního výkonu každé 2 roky): **40 let**.