IAL – 11. přednáška

Hashovací funkce

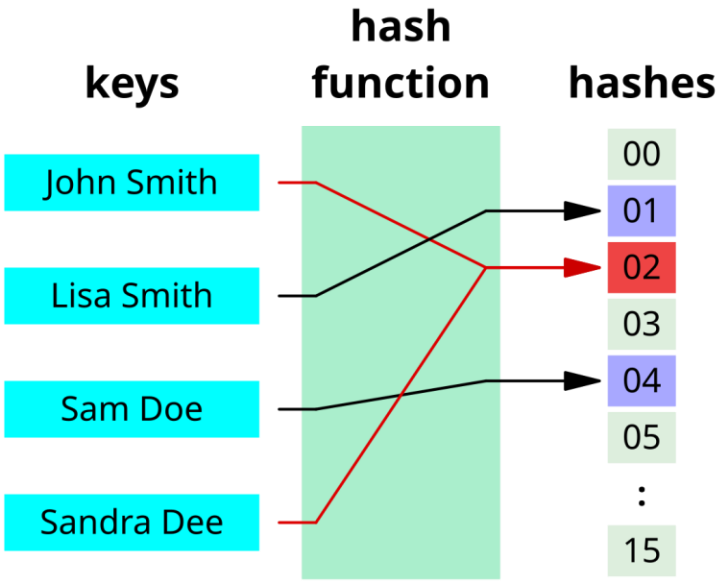
3. a 4. prosince 2024

Obsah přednášky

- Princip hashovací funkce
- Požadavky na hashovací funkce
- Využití hashovacích funkcí
- Kryptografické hashovací funkce
  - Bezpečnostní požadavky
  - Ukládání hesel, elektronický podpis, zranitelnosti
- Nekryptografické hashovací funkce
  - Funkce pro kontrolní součty
  - Klouzavé hashovací funkce
  - Fuzzy hashing, Perceptual hashing

Hashovací funkce

- Též hešovací, hašovací, či rozptylovací funkce
- Vstup **libovolné délky** transformuje na výstup **fixní délky** (z tohoto důvodu se nelze úplně vyhnout kolizím)
- Výstup se nazývá **hash** (heš, haš, či otisk)

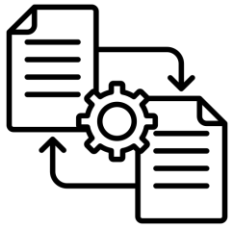


Hashovací funkce - požadavky

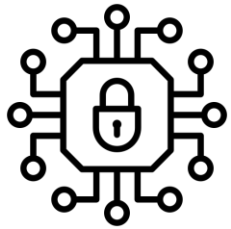
- **Fixní délka výstupu** – výstup má vždy **stejnou délku**
- **Determinismus** – pro stejný vstup vrací vždy stejný výstup
- **Efektivita** – není extrémně náročné funkci vyčíslit, a to ani pro velké vstupy
- **Minimalizace kolizí** – Funkce je navržena tak, aby pravděpodobnost kolize byla **co nejmenší**
- **Uniformní rozložení** – Funkce je navržena tak, aby rovnoměrně využívala celý prostor hodnot

Využití hashovacích funkcí (1/5)

- **Kontrola integrity dat** (kontrolní součty) – např.
  - **Kontrola integrity archivu** – zda není poškozen
  - **Zajištění spolehlivého přenosu** – počítačové sítě



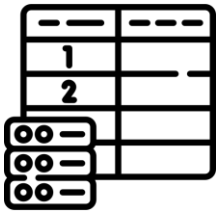
- **Bezpečnost (Kryptografické hashovací funkce)**
  - **Zajištění důvěrnosti** – např. ukládání hesel
  - **Zajištění nepopiratelnosti** – elektronický podpis
  - **Odvozování šifrovacích klíčů** (key derivation funtions)
  - **Generování autentizačních tokenů**



- **Rychlá identifikace obsahu**
  - hash je menší, rychleji se porovnává
  - **Hledání podřetězce v řetězci** (Rabin-karpův hash apod.)
  - **Deduplikace** – redukce shodných prvků (např. souborů)

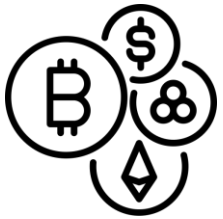
Využití hashovacích funkcí (2/5)

- **Tabulky s rozptýlenými položkami (TRP)**
  „hashovací tabulky“
  - **Určení pozice v tabulce** – získání indexu z klíče
  - **Určení kroku** – implicitní zřetězení, dvojí rozp. funkce



- **Forenzní analýza & Vyšetřování incidentů**
  - **Zajištění integrity důkazů** – kontrolní součty obrazů
  - **Eliminace známých souborů při dohledávání stop**

- **Blockchain a kryptoměny**
  - **Mining** – ověřování transakcí, tvorba nových bloků
  - **Generování adresy** – určení adresy peněženky
  - **Proof of work** – vynaložení úsilí u blockchainových operací



Využití hashovacích funkcí (3/5)

- **Hledání a získávání informací**
  - **Adresování obsahu** – identifikace objektů, revizí kódu nebo změn (např. Git)
  - **Bloomův filtr** – rychlá identifikace, zda prvek nepatří do dané množiny

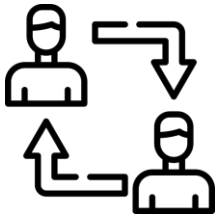
- **Vyvažování zátěže & Partitioning**
  (uniformní rozložení je zde velkou výhodou)
  - **Rozdělování výpočtu v distribuovaných systémech**
  - **Distribuované souborové systémy**

- **Strojové učení a zpracování dat**
  - **Feature hashing** – převod vícedimenzionálních dat do reprezentace v nižším menšího počtu dimenzí
  - **Data shuffling** – zajištění náhodného pořadí v souboru

Využití hashovacích funkcí (4/5)

- **Detekce malware a hrozeb**
  - **Identifikace závadných souborů** – porovnání hashe souboru s hashi známých závadných souborů
  - **Threat intelligence sharing** – sdílení signatur hrozeb

- **Sítě Peer2Peer**
  - **Adresování obsahu** – např. síť BitTorrent
  - **Synchronizace dat** – ověření, že máme aktuální verzi souboru (na základě porovnání hashe)



- **Bioinformatika**
  - **Analýza DNA** – porovnávání/identifikace sekvencí
  - **Protein matching** – hledání a porovnávání proteinů

Využití hashovacích funkcí (5/5)

- **Kontrola podobnosti**

  - **Fuzzy hashing, similarity hashing** – určení míry podobnosti dvou datových objektů
  - Vyžaduje **speciální hashovací funkce** pro tento účel (Podobné vstupy mají podobný hash - zatímco např. v kryptografii chceme přesný opak)
  - **Percepční hashování** (perceptual hashing) – Určení podobnosti objektů, které vypadají podobně, zvuku/hlasu, který zní podobně, …

Kryptografické hashovací funkce

Mají navíc další **bezpečnostní požadavky:**

- Je výpočetně nezvládnutelné v rozumném čase
  - pro daný hash najít původní vstup **(1st Preimage Resistance)**
    Pro y = h(x) nesmí jít jednoduše dopočítat x.
  - pro vstup najít jiný se stejným výstupem **(2nd Preimage Resistance)**
    Pro dané x nesmí jít najít x‘ takové, že h(x) = h(x‘) a zároveň x ≠ x‘
  - Najít dva vstupy, které dají stejný výstup **(Collision Resistance)**
    Nelze jednoduše najít x a x‘ takové, že h(x) = h(x‘) a zároveň x ≠ x‘
- **Malá změna na vstupu** (např. jednoho bitu) způsobí **velkou změnu na výstupu** (tzv. Avalanche Effect – efekt laviny)

```
MD5(Ahoj, jmenuji se Pavel!) = c601d67b9b43a5736e995ac9da0074d2
MD5(Ahoj, jmenuji se Havel!) = 0bc41f4f4f58f31db41505265f2a814b
```

Kryptografické hashovací funkce

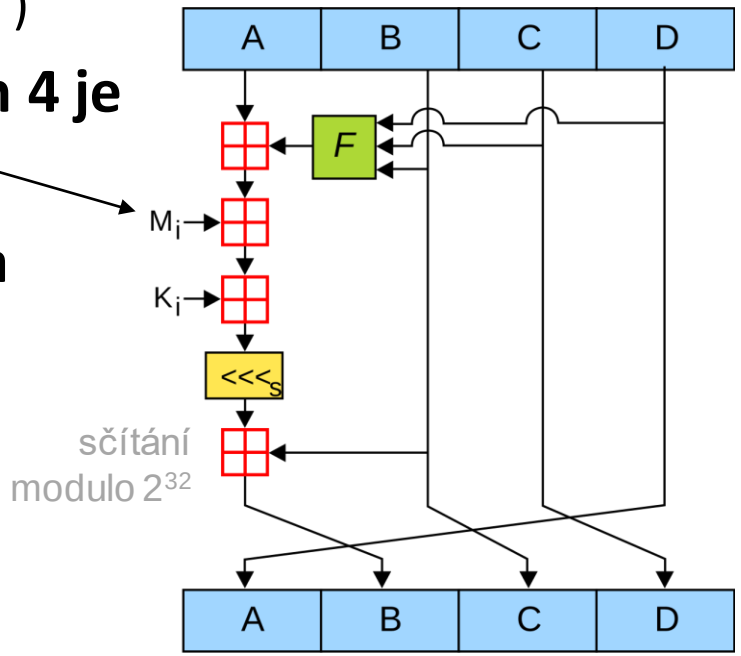
- **Message Digest** – autor: Ron Rivest (hlavně 90. léta)
  - MD2, MD4 a MD5 jsou dnes jednoduše prolomitelné
  - MD6 je relativně bezpečný algoritmus, ale málo používaný
- **Secure hash algorithm (SHA)**
  - SHA-1 – dnes již jednoduše prolomitelný
  - Rodina SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512, …)
  - Rodina SHA-3 / Keccak (SHA3-224, SHA3-256, SHA3-384, SHA3-512, …)
- **Blake2** – Rychlejší než SHA2/3, ale přesto silný algoritmus
- **BCrypt** – postaven na šifrovacím algoritmu Blowfish, konfigurovatelný počet iterací (pomocí ceny, tzv. cost factor) – dnes ukládání hesel v Unixu
- **SCrypt** – úmyslně paměťově náročný (lze obtížněji paralelizovat)
- **Argon2** – relativně bezpečný, konfigurovatelná paměťová náročnost

Ron Rivest, 2012

Příklad: Implementace MD5

- **128-bitový stav rozdělený na 32-bitová slova (A,B,C,D)**
  Tato jsou na začátku inicializována konstantní hodnotou
- **Vstup rozdělíme do 512-bitových dílů o 16 slovech délky 32b**
  (nevychází-li přesně, doplníme „vycpávkou“)
- **Provedeme 64 iterací/díl, v každých 4 je použito jedno slovo jako vstup M_i**
- **V každé ze 4 iterací je použita jedna z funkcí (střídají se):**

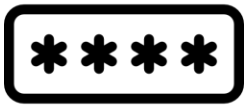
$$F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$$
$$G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$$
$$H(B,C,D) = B \oplus C \oplus D \quad \text{// operace XOR}$$
$$I(B,C,D) = C \oplus (B \vee \neg D)$$



K_i – konstanta
$<<<_s$ – bitová rotace

- **Finální stav představuje výstupní hash**

Ukládání hesel



- **Využití:** Systémy s uživatelskými účty – operační systémy, webové aplikace (informační systémy aj.)
- Uložení hesel v otevřené podobě (plaintext) není bezpečné:
  V případě získání databáze útočník získá všechna hesla!
  Pro fungování aplikace však ani není nutné ☺
- Namísto hesla uložíme **pouze hash**
- **Přihlášení:**
  - Uživatel zadá uživatelské jméno a heslo
  - Systém v databázi vyhledá hash hesla daného uživatele
  - Ze zadaného hesla se také spočítá hash
  - Porovná se hash zadaného hesla s uloženým
  - Shoda -> povolení přístupu

Lookup Table Attack

- **Útočník si vytvoří/obstará vyhledávací tabulku**
  - Klíčem je **hash**. Hodnotou je **heslo**, ze kterého byl vypočítán
  - Tabulka je naplněna předpočítanými hodnotami (hash -> heslo) pro určitá hesla (např. všechna hesla a-zA-Z0-9 do délky 6)
  - Prolomení: GetData(T, **hash**) -> získání hesla **v konstantním čase![2]**
- **Rainbow Tables**
  - Pokročilá varianta, která je **mnohem úspornější**
  - Využívá tzv. redukční funkci a princip řetězení (chaining)
  - Nepokrývá vždy všechny možnosti (typicky 96 – 99.9%)
  - Např. Pro SHA-1 a hesla do 9 znaků s 96.8% pokrytím zabírá 690 GB[1]

1 – Viz http://project-rainbowcrack.com/table.htm

2 – Je-li vyhledávací tabulka implementována pomocí TRP s vhodnými parametry

Ochrana proti Rainbow Tables

- Velikost tabulky roste exponenciálně s délkou uložených hesel
- Prakticky lze ukládat jen hesla určité délky (např. do 10-12 zn.)
- Myšlenka: heslo před vstupem do algoritmu **prodloužíme**
- **Kryptografická sůl (Cryptographic Salt)**
  - Pseudonáhodná hodnota o vysoké entropii (neurčitosti)

  - Vygeneruje se při tvorbě hesla. S heslem se spojí (prodlouží jej) před vstupem do hashovacího algoritmu.
  - Sůl následně uložíme spolu s hashem.
- **Ověření hesla se solí**
  - Z databáze získáme hash a uloženou sůl (např. pro daného uživatele)
  - Zadané heslo spojíme se solí a pak až počítáme hash. Následně hesla standardně porovnáváme.

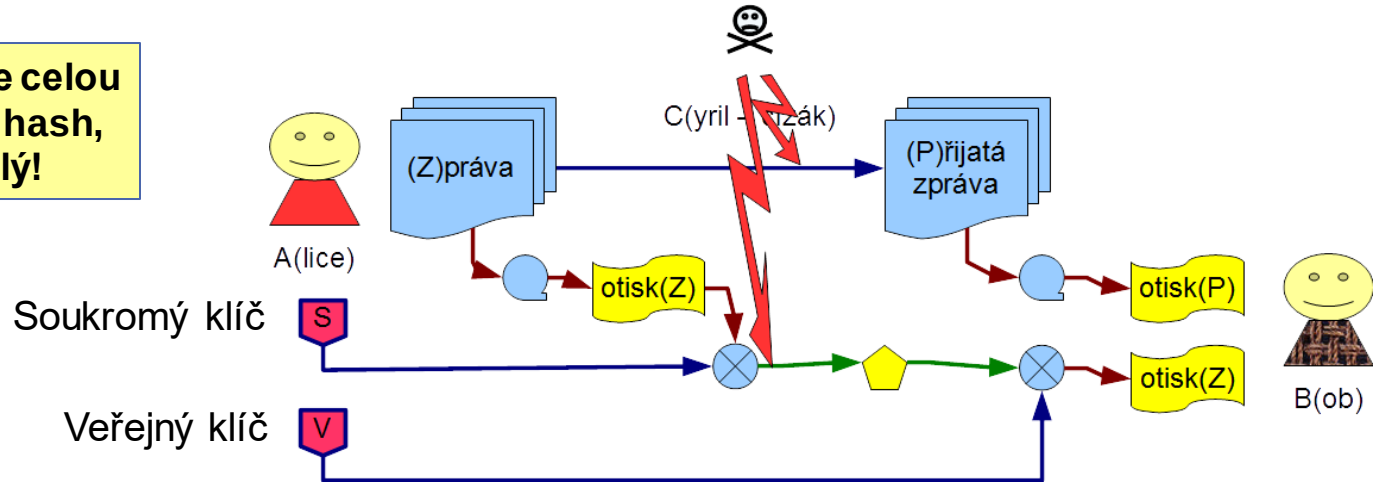
Odbočka: Šifrování

- **Symetrická kryptografie**
  - Používá jeden sdílený klíč – pro šifrování i dešifrování
  - **Výhoda:** Rychlost
  - **Nevýhoda:** Klíč si musí strany dohodnout bezpečným kanálem
  - Šifrovací klíč ale mívá **fixní délku** (např. 128 bitů)
  - Jak jej vytvořit z hesla? Funkcí pro odvození klíče (např. PBKDF2), která uvnitř využívá **hashovací funkci** (např. SHA-2-256)
- **Asymetrická kryptografie**
  - Každý má pár klíčů: **soukromý** (tajný) a **veřejný** (lze sdílet s ostatními)
  - Co zašifrujeme jedním klíčem, jde dešifrovat jen druhým z páru
  - **Výhoda:** Není nutné řešit distribuci sdíleného klíče
  - **Nevýhoda:** Asymetrické šifry jsou pomalé, zejména pro dlouhé zprávy

Elektronický podpis

- **Alice**
  - Vytvoří **hash** (otisk) zprávy. Tento zašifruje svým soukromým klíčem.
  - Zašifrovaný hash je **elektronický podpis**, který ke zprávě přiloží.
- **Bob**
  - Dešifruje podpis veřejným klíčem Alice -> získá hash spočítaný Alicí.
  - Ze zprávy si spočítá svůj vlastní **hash**.
  - Pokud se **hashe shodují**, byl podpis ověřen.

Tím, že nešifrujeme celou zprávu, ale pouze hash, je postup rychlý!



Lámání hešů

hashcat
advanced
password
recovery

- Uvažujme nástroj Hashcat a GPU NVIDIA RTX 4090
- 8-znakové heslo zabezpečené algoritmem MD5

Σ	|Σ|	Počet možností	Max. doba
a-z	26	208 827 064 576	1,27 sekund
a-z, A-Z	52	53 459 728 531 456	5,43 minut
a-z, A-Z, 0-9	62	218 340 105 584 896	22,18 minut
a-z, A-Z, 0-9 + special.	95	6 634 204 312 890 625	11,23 hodin

- 8-znakové heslo zabezpečená hashem SHA3-512

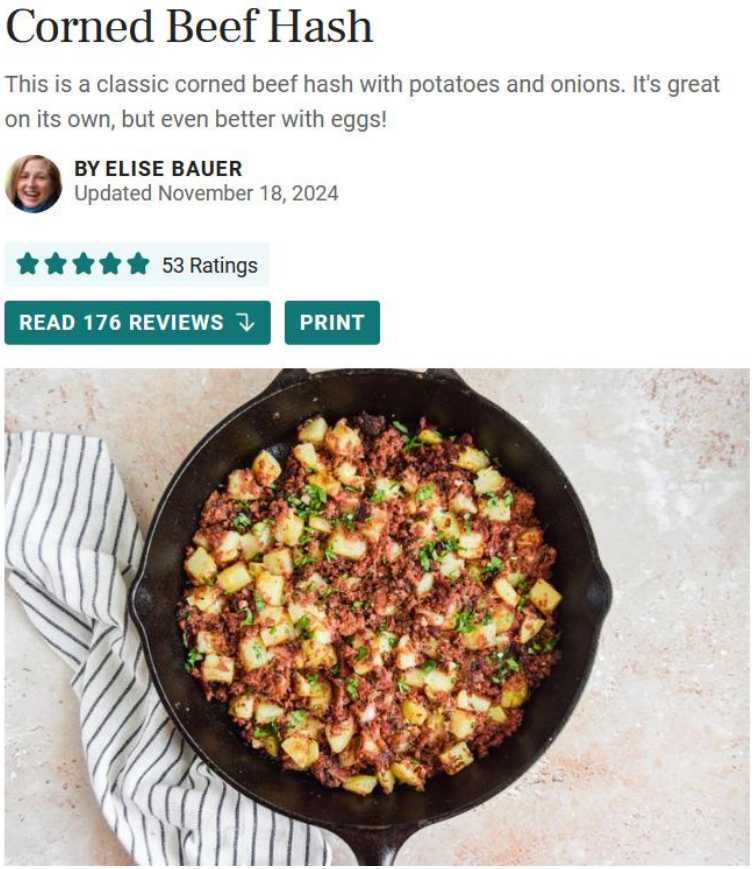
Σ	|Σ|	Počet možností	Max. doba
a-z	26	208 827 064 576	41,29 sekund
a-z, A-Z	52	53 459 728 531 456	2,94 hodin
a-z, A-Z, 0-9	62	218 340 105 584 896	12,00 hodin
a-z, A-Z, 0-9 + special.	95	6 634 204 312 890 625	15,19 dní

Nekryptografické hashovací funkce

Obvykle chceme i pro větší vstup:

- **Rychlý výpočet**
- **Malé nároky na paměť**

Pozn. časová složitost je (i u kryptografických) typicky O(n)



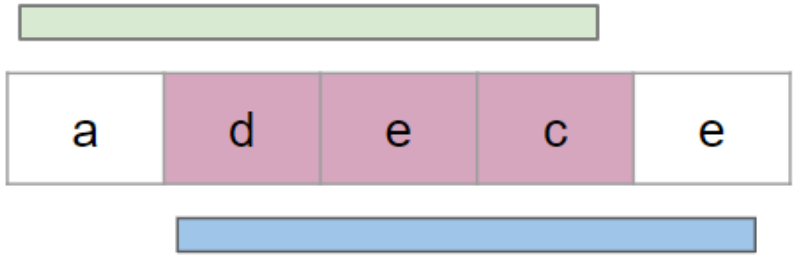
Funkce pro kontrolní součty

- **Požadavky:**
  - Rychlý výpočet i pro velký vstup
  - Jednoduchá implementace
  - Jednoduché použití pro detekci chyb
- **Příklady**
  - **Paritní bit** – XOR nad všemi bity, výsledek: 1 bit (mnoho kolizí ☹)
  - **LRC** (Longitudal Redundancy Check) – parita po bytech, výsledek: 1B
  - **CRC** (Cyclic Redundancy Check): CRC-16, CRC-32
    Princip dělení polynomů v konečném tělese – implementováno pomocí jednoduchých bitových operací -> velmi rychlé
  - **MD5/SHA-1** (pro zajištění integrity jsou OK)
  - **Adler32** – při kompresi dat (např. zlib), rychlejší než CRC-32
  - **Fletcherův algoritmus** – podobný princip jako Adler32

Klouzavé hashovací funkce

- **Rolling hash functions** – Klouzavé hashovací funkce
  - Využívají okénko, které se posouvá nad vstupem
  - Při posuvu lze novou hodnotu hashe přepočítat **v konstantním čase**
    Často staví na polynomech: stačí vynásobit, něco přičíst a něco odečíst
- **Příklady**
  - Rabin-Karpův algoritmus – vyhledávání v textu
  - Spamsum – detekce spamu v poště
  - CRC, Adler32 – kontrolní součty

„zelené“ a „modré“ okénko mají společnou část d, e, c



Fuzzy Hashing

- **Fuzzy hashing / similarity hashing** = určování podobnosti
- Je naopak žádoucí, aby podobné vstupy měly podobný hash
- **SDhash** (Similarity Digest Hash)
  - Hledá v textu příznaky – n-gramy či specifické vzory
  - Pro tyto příznaky spočítá hashe
  - Výsledný hash vznikne spojením hashů pro příznaky
  - Porovnávání hashů = počítání výskytů společných příznaků
- **Další typy:**
  - SSDeep
  - TLSH (Trend Micro Locality Sensitive Hashing)

Perceptual Hashing

- **Percepční hashování** – fuzzy/similarity hashing přizpůsobený pro multimediální data
- Obrázky: chceme, aby podobně vypadající měly podobný hash
- Zvuk: chceme, aby podobně znějící zvuk měl podobný hash
- **Přístupy**
  - Extrakce příznaků
  - Redukce dimenzí
  - Měření podobnosti
  - Kvantizace
- Např.
  - pHash, dHash
  - aHash