

1 ☐ **Adatbiztonság, adatvédelem**

Hash algoritmusok és Hamming kódok

2 ☐ **Hash algoritmusok működése**

- Bemenet: végtelen hosszú adat.
- Kimenet: fix hosszúságú adat.
- A képzési eljárás egyirányú, a tényleges adat nem nyerhető ki az ellenőrző összegből.
- Egy ellenőrző összeg több adatcsomaghoz is tartozhat.

3 ☐ **Példa egy egyszerű algoritmusra**

- Bemenet: 3 bites bináris szám
- Kimenet: egyesek száma a kódban + 1 bit, ha páros a szám 0-> párosnak tekintett.
- 3 bitet használunk fel erre

4 ☐ **Példa egy egyszerű algoritmusra**

- Lényeg az lenne, hogy egyedi, nem ismétlődő kimenetet kapjunk
- A legtöbb hash függvény nem lineáris matematikai függvényeket alkalmaz.
- Ezáltal minimalizáljuk az ütközések lehetőségét.

5 ☐ **Példa egy jobb, nem linerális Hash függvényre**

```
int h(int x) {  
    return x % 16;  
}
```

- A % jel a maradékos osztás
- A 16 pedig 16 elem megkülönböztetését teszi lehetővé.
- Tehát ha 128 különböző elemet akarunk, akkor a moduló szám 128

6 ☐ **Példa egy jobb, nem linerális Hash függvényre**

- Tehát 32 bites hash esetén a moduló szám:  $2^{32} - 1$ ;
- Jó eredményeket ad a módszer, viszont kriptográfiailag nem éppen jó
- Viszont arra jó, hogy egyedi sorszámmal illesszünk egy adatot.

7 ☐ **Hash algoritmusok használata (Példák)**

- Adatok épségének ellenőrzése
  - Adat és hozzá tartozó hash közzététele, letöltés után adaton hash számítás, majd a letöltött hash és a számított hash összevetése.
  - Torrent protokoll beépítetten tartalmazza
- Programozási környezetben objektumok egyezésének összehasonlításának egyik módszere. Pl: .NET és C#

8 ☐ **Hash algoritmusok használata (Példák)**

- Fájlok aláírása
  - Leginkább futtatható fájlok esetén használják más titkosítási technológiákkal együtt
  - Aláírt .exe és .dll fájlról meg tudja mondani a rendszer, hogy az előállítás óta megsérült - e, vagy módosította - e egy vírus.  
(.NET exe file-ok, strong namekey technológia)
  - Kevés az ilyen .exe és .dll

9 ☐ **Hash algoritmusok használata (Példák)**

- BitCoin tranzakciókban tranzakció lebonyolításakor a tranzakció eredetiségének ellenőrzésére szolgál.

- Manapság nincs olyan digitális átviteli rendszer, ami nem használna valamilyen hash algoritmust.
- De nem csak ilyen célokra használhatóak...

#### 10 ☐ Hash algoritmusok használata (Példák)

- Jelszavak titkosítására
  - Regisztrációkor:
    - Jelszó hash eltárolása adatbázisban
  - Belépéskor:
    - Beírt jelszó hash kiszámítása, majd összehasonlítás adatbázisban tárolt hash-el.
    - Ha a 2 megegyezik, akkor minden ok, ellenkező esetben hiba.

#### 11 ☐ Hash algoritmusok használata (Példák)

- HashMap tároló osztályok -> C# és Java esetén Dictionary típus
- Lényegében nem int típussal indexelt tömb. A tömb bármilyen típussal indexelhető, egyfajta összerendelési táblázat.
- Az index típusához hash számítás, ami meghatározza a helyzetet egy fix méretű tömbben.
- A tömb mérete belsőleg prím számok szerint növekszik:  
1, 2, 5, 7, 9, stb...

#### 12 ☐ Hash algoritmusok használata összefoglalva

- Adatátvitel esetén integritás tesztelésre
  - Ma már leginkább protokollsinten, de ettől függetlenül is alkalmazható
- Beléptető rendszerekben biztonságos jelszó tárolásra alkalmazhatóak.

#### 13 ☐ Miért kell titkosan tárolni a jelszavakat?

- Sok esetben bebizonyosodott, hogy egy védelem nem védelem...
- Az adatbázis rendszerek titkosítása feltörhető.
- Emberi természetből adódóan általában egy jelszót használunk mindenhol, vagy az „egy jelszó” véges számú permutációját.

#### 14 ☐ Sony Incidens PS Network, 2011

- A történelem eddigi legnagyobb jelszó kiszivárogtatása.
- 77 millió felhasználó jelszava és 12 000 bankkártya összes adata került ki a nyilvánosságba.
- A felhasználók jelszavai egyszerű szöveggént voltak tárolva.
- Teljesen újra kellett írni a PS Network kódját, a rendszer több, mint egy hónapig nem üzemelt.

#### 15 ☐ Sony Incidens PS Network, 2011

- Ez a „kis” baklövés a Sony-nak nagyjából 171 millió dollár (jelenlegi árfolyamon: 48.128 milliárd forint) veszteséget okozott.
- Eset tanulsága felhasználói szempontból:
  - Bankkártya adatok ne legyenek mentve
  - Több jelszó használata

#### 16 ☐ Pepsi promóciós incidens, 2012

- Nem kell külföldre menni rossz példákért (sajnos)
- 2012-ben 50 ezer felhasználó jelszavát és e-mail címét szivárogtatta ki egy török hacker csapat.
- A jelszavak szintén nem hash formában voltak tárolva.

#### 17 ☐ Hash algoritmusok problémái

- Sebeség
  - Minél komplexebb a funkció, annál lassabb lesz az algoritmus.
- Átfedések
  - Több bemeneti adatnak is lehet azonos a hash-e, definícióból adódóan.
  - Cél: minimalizálni ennek az esélyét. Kézenfekvő eszköz a kimeneti bitek számának növelése.
  - 
  -

#### 18 ☐ Hash algoritmusok problémái

- Idővel minden algoritmus elavul, hiszen egyre gyorsabbak a gépek
- Gyorsabb gépekkel könnyebb megtalálni azokat a bemeneti kombinációkat, amelyek ugyanazt a kimenetet produkálják.
- Ha ez nem lenne elég, van egy ennél sokkal komolyabb gond is...

#### 19 ☐ Hash algoritmusok problémái

#### 20 ☐ Hash algoritmusok

- Kb. annyi van, mint égen a csillag.
- Három darab algoritmusról lesz ma szó:
  - CRC család, ebből a CRC32-ről beszélünk ma
  - MD5
  - SHA család

#### 21 ☐ CRC-32

- A CRC hash algoritmusok családjának 32 bites változata.
- Őse, a CRC-8 története 1961-ig nyúlik vissza.
- Ezen algoritmusok azért jók, mivel könnyű őket hardveresen és szoftveresen is implementálni.
- Feltalálója W. Wesley Peterson.
- A CRC32 1975-ben jelent meg.

#### 22 ☐ CRC-32

- Elsősorban átviteli hibák detektálására használják.
- Más célokra a kevés bitszám miatt nem alkalmas.
- Számos helyen alkalmazott:
  - Ethernet, SATA, MPEG2, ZIP, GZIP, PNG, stb...
- Elődei és utódai még több helyen vannak alkalmazva.
- 

#### 23 ☐ CRC32 működése

- Biteltolások és XOR műveletek sorozatával dolgozik.
- Fontos eleme az alapszám, amely mindig egy bittel hosszabb, mint az algoritmus biteinek száma.
- Algoritmusonként eltér, kiválasztásánál fontos tényező az adatcsomagok mérete, hogy minimalizálják a hibákat.
- Az alapszám és az adaton elvégzett műveletek sorozataként áll elő a hash kód.

#### 24 ☐ Egy C++ CRC-32 implementáció

- //összesen 17 sor ☺
- unsigned int CRC32\_function(const unsigned char \*buf, unsigned long len)
- {
  - unsigned long crc\_table[256];

```
    unsigned long crc;
    for (int i = 0; i < 256; i++) {
        crc = i;
        for (int j = 0; j < 8; j++) {
            crc = crc & 1 ? (crc >> 1) ^ 0xEDB88320UL : crc >> 1;
        }
        crc_table[i] = crc;
    }
    crc = 0xFFFFFFFFUL;
    while (len--) {
        crc = crc_table[(crc ^ *buf++) & 0xFF] ^ (crc >> 8);
    }
    return crc ^ 0xFFFFFFFFUL;
}
```

25 ☐ **MD5**

- RFC 1321
- 1991-től
- 2008 óta titkosítási célokra nem ajánlott, mivel több súlyos hibát találtak benne.
- 128 bites hash = 16 byte
- Általában 32db hexa karakterként kifejezve

26 ☐ **Az MD5 működése**

- Első lépésben bemeneti adat 512 bites blokkokra bontása
- Amennyiben az adat nem osztható 512-vel, akkor kiegészíti a végét 0-val.
  - Utolsó 64 bit, ha lehetősége van rá, a bemeneti adat bitjeinek száma előjel nélküli egész számként.
- Fő algoritmus: 128 biten dolgozik 4x32 bit formában.

27 ☐ **Az MD5 működése**

- 4db 32 bites szám (A, B, C, D) előre meghatározott konstansról indul.
- Az adat feldolgozása 512 bites blokkokban történik, ami tovább van bontva 32 bitre.
- A feldolgozott adat módosítja a 4db 32 bites szám értékét.
- 4db módosító függvényt használt, amelyek 128 bit feldolgozása után váltakoznak.

28 ☐ **Az MD5 működése**

- Módosító függvények:
  - 
  - 
  -

29 ☐ **Az MD5 működése**

30 ☐ **MD5 „feltörése”**

- MD5 esetén ez 2008-ban sikerült.
- 128 bit esetén  $2^{128}$  egyedi hash készíthető.
- Ez jó közelítéssel:  $3,402 \times 10^{38}$
- A bolygó teljes levegőjében nincs annyi molekula, mint amennyinek egyedi MD5 hash-t tudnánk adni.

31 ☐ **Felmerülhet a kérdés...**

32 ☐ **MD5 „feltörése”**

- Valószínűség számítás alapján 2 bemeneti kód azonos hash kimenetének az esélye  $2^{n/2}$ , ha mindkét bemenet egyforma esélyekkel indul.
- 128 bit esetén tehát legalább  $2^{64}$  bemeneti kódot kell átnéznünk, hogy legyen legalább 2 olyan bemenet, ami azonos kimenetet produkál.

33 ☐ **Minimum mennyiségű hash, 2 azonos bemenet találásához**

34 ☐ **MD5 feltörése**

- CPU-k jelenleg is lassúak ezen feladatra
- Megoldás: GPU használata, mivel azonos számításokat kell elvégezni, sokszor.
- Nvidia GF 8800 ultra kártya másodpercenként 200 millió hash-t tud generálni.
- Ilyen sebesség mellett is 2 azonos bemenet találása legalább  $5,75 \cdot 10^{12}$  év lenne
- A helyzet tovább romlott, hiszen szinte minden mai számítógép alkalmas a feladatra.

35 ☐ **MD5 feltörése**

- Amiért mégis lehetséges a dolog: ismétlődő mintát találtak a konstansokban, amelyek lépésenként alkalmazva vannak.
- $2^{21}$  hash szükséges csak 2 azonos bemenet találásához (2 097 152 próbálkozás csupán).
- További gond az úgynevezett szivárvány tábla (Rainbow table)
- Ez rengeteg gyakran használt jelszó MD5 értékét tartalmazza visszakereshetőség miatt.

36 ☐ **SHA1**

- Amerikai Nemzetbiztonsági Hivatal tervezte
- SHA: Secure Hash Algorithm
- Elődje az SHA0, amely széles körben sosem terjedt el, mivel matematikailag gyengének bizonyult.
- 160 bites algoritmus, szintén 512 bites blokkokban dolgozik

37 ☐ **SHA1**

- 2005-ben váltotta le az SHA2
- Váltás oka: létezik olyan algoritmus, amely segítségével  $2^{80}$  próbálkozásnál kevesebbel található két azonos bemenet.
- $2^{51}$  próbálkozással található azonos bemenet.
- Elméleti törés, hiszen 1 valódi ütközés találásának ideje:  $8,69 \cdot 10^{35}$  év (200 millió/s sebesség mellett)

38 ☐ **SHA Család további tagjai**

39 ☐ **Snowden óta**

- Felmerül a kérdés, hogy tényleg olyan biztonságos-e a önmagában az SHA család, vagy ebben is van rejtett hiba ?
- Egyelőre erről nincs információ, azonban jogos a félelem.
- Felmerülhet a kérdés, hogy lehetséges-e kivédeni a rejtett hibákat egy hash rendszeren?

40 ☐ **A válasz: Részben igen**

- Több hash algoritmus egyidejű használata.
  - Ha két hash értéket tárolok, akkor nagyságrendekkel kisebb a valószínűsége annak, hogy találjak olyan bemeneti kombinációt, amely mindkét hash algoritmust kijátssza.
  - Ezzel a gond az, hogy több adatot kell tárolnom, vagy a jelszó hash hash-ét tárolom ☺ Ez a

megoldás  $n+1$  körben folytatható.

41 ☐ **A válasz: Részben igen**

- Egyedi inicializációs vektorok használata
  - Minden hash algoritmus rendelkezik egy alapszámmal, ami a belső működéséhez kell.
  - Ezeket egyedileg is megválaszthatom.
  - A probléma ezzel az, hogy nem biztos, hogy biztonságosabb lesz a rendszer tőle.
  - Ezért azt szokás csinálni, hogy az alapszámok maradnak, de módosítva lesznek egy véletlen szám értékkel.

42 ☐ **Hamming Kódok**

43 ☐ **Hamming távolság**

- Azt mutatja meg, hogy egy kódrendszer bármelyik kódszava legalább hány biten tér el a rendszer összes többi kódszavától.
- Tétel: Ha egy kódrendszer Hamming távolsága  $d$ , akkor az átvitel során  $d-1$  hiba felismerhető,  $d/2$  hiba esetén a hiba javítható is.
- Előző előadáson tárgyalt paritás bit bevezetése 1 - el növeli a rendszer Hamming távolságát.

44 ☐ **Példa**

- Ha egy kódrendszer Hamming távolsága 5, akkor
  - Ha 1-2 bit hiba van, akkor azt a vevő javítani tudja
  - Ha 4 hibás bit van, akkor a vevő még azt is tudja érzékelni
- 

45 ☐ **Hamming távolság számítása**

- ZH-ban várható ilyen feladat
- Minden kódszó összevetése a többivel és bit eltérések ( $d$ ) számolása páronként.
- A bit eltérések minimuma lesz a rendszer Hamming távolsága ( $H_{min}$ ).

46 ☐ **Példa**

47 ☐ **Hamming korlát**

- Hibajavító kódok hatékonyságát mutatja meg.
- Az olyan kódrendszert, amely eléri a Hamming korlátot, tökéletes kódnak nevezzük.
- Hamming határ:
  - $a$  kód  $t$  hibát tud javítani
  - $q$ : a kódábécé elemszáma (bináris esetben  $q=2$ )
  - $k$ : az üzenetek hossza
  - $n$ : a kódszavak hossza

48 ☐ **Hamming Kódok**

- Általánosított formái a Hamming(7,4) kódnak
- Richard Hamming, 1950
- Tökéletes kódok amelyek ezen algoritmusból származtatottak.
- 

49 ☐ **Hamming kódok használati területei**

- Kitalálás idején lyukkártya információk védelmére használták.
- Manapság olyan rendszerek esetén alkalmazott, ahol az adat újraküldés menet közben nem megoldható.

- PI: Hang és videó stream-ek, Audio CD lejátszók és DVD lejátszók.

50 ☐ **Hamming kódok használati területei**

- ECC memóriák esetén is alkalmazott, amelyeket szerver gépekben szoktak alkalmazni.
- Erre a DRAM memória cellák működési elve miatt van szükség.
- Érdekességképpen: a DRAM memória felépítése miatt alkalmatlan arra, hogy az űrben mission critical rendszerben legyen alkalmazva.

51 ☐ **Hamming Kódok**

- Hamming feltételezése az volt, hogy létezik olyan kód, amelyben a hibaészlelő bitek elhelyezkedése olyan, hogy mindegyik bithiba jól megkülönböztethető.
- Ezáltal, ha tudjuk, hogy mi a hiba, akkor javítani tudjuk.

52 ☐ **Általános algoritmus**

- 1 hibát tud javítani az algoritmussal generált kód bármilyen hosszú kódszavak esetén.
- Speciális név: SECC – Single Error Correcting Code
- 

53 ☐ **Általános algoritmus**

1. Bitek sorszámozása 1-től, LSB->MSB irányba
2. Bit sorszámok átalakítása bináris számmá
3. Azon sorszámok alatt szereplő bitek, amelyek 2 hatványai, paritás bitek lesznek
4. A többi bit adathordozó bit lesz.
5. A paritás bit jelölése mindegy, bár a páros = 1 jelölés a gyakorlatban egyszerűbb

54 ☐ **Hamming kódok vizuálisan**

55 ☐ **Paritás bitek és adathosszak**

56 ☐ **Hamming(3,1)**

- 3x ismétlésű kódot eredményez.
- 3x tartalmazza az adatot, régen használták zajos csatornák esetén
- 1/3-ra csökkenti az adatátviteli sebességet, ma már nem használt.
- Példa:
  - Küldendő adat: 101
  - Küldött adat: 111 000 111

57 ☐ **Hamming(7,4)**

- Hamming kódok legáltalánosabban használt változata
- 

58 ☐ **Hamming(7,4)**

59 ☐ **Bővített hamming kód**

- Hamming(8,4)
- Egy extra paritásbittel kiegészített kód.
- Felépítése:

60 ☐ **Köszönöm a figyelmet**