Az MD5 hash-függvény

Az MD5 függvény egy széleskörűen használt, 128 bites, egyirányú hash-függvény, melyet internetes szabványként[1] már régóta használnak biztonsági alkalmazásokban, illetve adat-integritás ellenőrzésére. MD5 hash-sel többnyire 32 karakteres hexadecimális szám formájában találkozunk.

Az utóbbi időben megjelent néhány on-line hozzáférhető, ún. *„szivárvány-tábla"*[2] (*rainbow table*), amelyek segítségével sok MD5 értéket fordíthatunk vissza eredeti jelentésére.

Az MD5 függvényt *Ronald Rivest* professzor, az *RSA algoritmus* egyik megalkotója definiálta, 1991-ben, az addig használt MD4 függvény felváltására. 1996-ban ismertté vált egy ütközés az MD5 algoritmushoz, amely bár nem volt teljes értékű törés, mégis sokan kezdtek el más megoldásokat javasolni az MD5 helyett.

Később, 2004-ben sokkal komolyabb sebezhetőségek tették megkérdőjelezhetővé az MD5 további használatát biztonsági célokra. A támadás alapja a *születésnap-paradoxonra* (valójában nem paradoxon, de első látásra nehezen hihető: 23 véletlenszerűen kiválasztott ember között 50%-nál nagyobb a matematikai valószínűsége, hogy van két olyan ember, akik ugyanakkor ünneplik a születésnapjukat) épült: a hash-érték 128 bites mérete elég kicsi ahhoz, hogy ún. *birthday-attack* típusú támadást lehessen rá alapozni. A birthday-attack lényege, hogy a születésnap-paradoxon alapjául szolgáló matematika elemeit felhasználva bizonyítható, hogy ha egy *f* függvény *H* darab különböző eredményt adhat egyenlő valószínűséggel, és *H* elég nagy, akkor kb. $1.2 \cdot \sqrt{H}$ különböző argumentumra kiszámolva számíthatunk rá, hogy találunk olyan *x* és *y* párt, amelyre *f(x)=f(y)* (ezt hívják ütközésnek vagy *collision*-nek).

2004 márciusában indult az *MD5CRK* nevű projekt, amelynek célja az MD5 gyengeségeinek demonstrálása volt, végül 2004 augusztusának második felében mutatták be a módszert, amellyel egy óra alatt sikerült ütközést találni egy IBM p690 clusteren (32 darab 1.90 GHz-es processzor, 1 TB RAM, több mint 1000 kg).

2005-ben publikálták azt az algoritmust, amely szintén képes tetszőleges MD5 hash-hez ütközést találni néhány óra alatt egy egyszerű laptopon.

Mind közül a legérdekesebb törés viszont az a 2006 márciusában publikált módszer[3], amely néhány perc alatt állít elő ütközést egy átlagos noteszgépen.

Alkalmazás

Az MD5 algoritmust a szoftveriparban hálózaton továbbított file-ok *integritásának ellenőrzésére* használják, számos *Unix/Linux* disztribúció magában foglal MD5-eszközöket, és *Windows* alá is elérhetők ilyen alkalmazások. Az ütközéskereső algoritmusok fejlődésének köszönhetően ez a megoldás manapság nem a legmegbízhatóbb.

Az MD5 másik gyakori felhasználási területe *jelszavak tárolása*. Mivel az Interneten ma már rengeteg MD5-adatbázis hozzáférhető, az egyszerű MD5-tel hash-elt jelszavak könnyen kideríthetők. Ezen támadások kivédésére szokás alkalmazni az ún. *salt*-olást, amelynek lényege, hogy a jelszót egy véletlenszerű bitsorozattal kombinálják. Javasolt a hash-függvényt többször alkalmazni, így ugyan nő a jelszó kódolásának ideje, de a szótár-típusú támadásoké is. Igaz, már ismert olyan algoritmus (névszerint az *md5x*), amely egy lépésben állítja elő „dupla hash”-t.

Működés

Az MD5 algoritmus a változó hosszúságú üzenetet 512 bites blokkokra bontja, a kipótláshoz egy darab 1-es bitet, majd annyi nullát, fűz hozzá, amennyi ahhoz kell, hogy a hossz -64-et adjon modulo 512. A fennmaradó 64 bitet az eredeti üzenet hosszát reprezentáló 64 bites egész szám tölti ki.

Az algoritmus lényegi része egy 128 bites vektort kezel, amely négy 32 bites értékre bomlik. Ezeket rögzített konstansokkal inicializáljuk. Az algoritmus az üzenet 512 bites blokkjait négy lépésben dolgozza fel, minden körben 16 hasonló, egy nem lineáris függvényen, maradékos összeadáson, illetve bitenkénti balra forgatáson alapuló műveletet elvégezve.

Pszeudókóddal[4]:

```
//Note: Minden változó 32 bites előjel nélküli egész, műveleteknél
túlcsorduláskor modulo 2^32 számolva

var int[64] r, k
r[ 0..15] := {7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22}
r[16..31] := {5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20}
r[32..47] := {4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23}
r[48..63] := {6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21}

//Az egészek szinuszainak bináris egészrészét használjuk konstansokként
for i from 0 to 63
    k[i] := floor(abs(sin(i + 1)) × 2^32)

//Induló hash-érték
var int h0 := 0x67452301
var int h1 := 0xEFCDAB89
```

```
var int h2 := 0x98BADCFE
var int h3 := 0x10325476

//Előkészítés
append "1" bit to message
append "0" bits until message length in bits ≡ 448 (mod 512)
append bit length of message as 64-bit little-endian integer to message

//Rendre feldolgozzuk az üzenet 512 bites darabjait
for each 512-bit chunk of message
    break chunk into sixteen 32-bit little-endian words w(i), 0 ≤ i ≤ 15

    //Az aktuális csonk hash-értékének inicializálása
    var int a := h0
    var int b := h1
    var int c := h2
    var int d := h3

    //Fő ciklus:
    for i from 0 to 63
        if 0 ≤ i ≤ 15 then
            f := (b and c) or ((not b) and d)
            g := i
        else if 16 ≤ i ≤ 31
            f := (d and b) or ((not d) and c)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47
            f := b xor c xor d
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63
            f := c xor (b or (not d))
            g := (7×i) mod 16

        temp := d
        d := c
        c := b
        b := ((a + f + k[i] + w(g)) leftrotate r[i]) + b
        a := temp

    //A csonk hash-értékét a végeredményhez adjuk:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d

var int digest := h0 append h1 append h2 append h3 //(little-endian)
```

Irodalom:

1. http://tools.ietf.org/html/rfc1321
2. http://kestas.kuliukas.com/RainbowTables/
3. http://eprint.iacr.org/2006/105
4. http://en.wikipedia.org/wiki/MD5