Eötvös Loránd Tudományegyetem
Természettudományi Kar

Hash függvények

Szakdolgozat

Készítette: Farkas Vanda

Matematika Bsc, Matematikai elemző szakirány

Témavezető: Szabó István

Eötvös Loránd Tudományegyetem, Természettudományi Kar

Valószínűségelméleti és Statisztika Tanszék

Budapest, 2021

Köszönetnyilvánítás

Szeretném megköszönni témavezetőmnek, Szabó Istvánnak, aki végig kitartott mellettem és elvállalta, hogy témavezetőm legyen. Támogatott teljes három féléven át annak ellenére, hogy nem voltam a leglelkesebb diák, ha a szakdolgozatról volt szó.

Hálás vagyok volt lakótársaimnak, akik mindvégig támogattak még akkor is, amikor én már feladtam. Köszönet barátaimnak, akik végig kisértek egyetemi éveim alatt és egyetem után is mellettem maradtak. Köszönet a sok segítségért Juhász Szabolcsnak, aki nélkül valószínűleg én sem lennék itt. Szeretném megemlíteni családomat, akiknek hálás vagyok, amiért biztattak egyetemi tanulmányaim befejezésében.

Köszönök mindent.

TARTALOMJEGYZÉK

Bevezetés 3
1. Kriptográfiai hash függvények 4
1.1 Definíció 4
1.2 Csoportosítás 4
1.2.1 Kulcs nélküli hash függvény 5
1.2.2 Kulcsos hash függvény 6
1.3 Kapcsolat az ütközőmentesség és egyirányúság között 6
1.4 Születésnapi paradoxon [2] 8
1.5 Lavinahatás 10
2. Hash függvény konstruálása 12
2.1 Általános modell 12
2.2 Merkle-Damgård konstruckió 13
3. Támadások 15
4. MD- család 17
4.1 MD5 17
4.1.1 Ütközéskeresés MD5 hash függvényben 20
5. SHA függvény család 23
5.1 SHA-1 23
5.1.1 Ütközés az SHA-1 algoritmusban 25
5.2 SHA-2 28
6. MAC 31
6.1 Digitális aláírás 31
6.2 HMAC 32
7. Összegzés 33
Hivatkozások 34
Ábrajegyzék 36

BEVEZETÉS

A hash függvények a bármekkora méretű bementből egy fixált méretű kimenetbe képeznek. A számítástechnikában az 1950-es években jelentek meg, magyarul hasítófüggvényeknek is szokás nevezni. Elsősorban a tároló technikában töltöttek be fontos szerepet. Később, az 1980-as évek legvégén a digitális aláírás megjelenésével váltak szükségessé az informatikai biztonság területén. A tetszőleges méretű adatsor integritásának, épségének védelme helyett, egy fix hosszúságú, kisméretű bit stringre (bit sorra) koncentrálunk, amely az adat méretéből adódóan könnyebb és gyorsabb. A hash függvények kimenetére tekinthetünk úgy, mint digitális ujjlenyomat, mert két különböző dokumentumnak nem lehet azonos lenyomata, mint ahogy két embernek nem lehet azonos ujjlenyomata.

Napjainkban, az informatika világában, a kriptográfiában és a biztonságtechnikában töltenek be fontos szerepet a hash függvények. Minden nap találkozunk velük és használjuk őket anélkül, hogy tudomásunk lenne róla. Amikor emailben csatolt fájlt küldünk, amikor belépünk Facebook fiókunkba, vagy amikor éppen csak megnyitjuk a böngészőt akkor is hash függvényt használunk. Alkalmazhatjuk digitális aláírásra, jelszótárolásra, plagizálás keresésére és még sorolhatnám.

Miért használjuk ilyen széles körben? Azért, mert kulcsfontosságú szerepet töltenek be adataink védelmében. Manapság egyre több kibertámadásról lehet hallani a híradásokban, olvasni az interneten, amik éppen azért történnek meg, mert a védelmi rendszerekben valamilyen gyengeség van. Például online bank fiókjában egy szerződést jóváhagy, majd a támadó egy módosított szerződést szintén jóvá hagy az ön digitális aláírásával a tudomása nélkül, amit a bank öntől fog behajtani, hiszen elektronikusan jóváhagyta azt.

A technika fejlődésével elengedhetetlenné vált, hogy a hash függvények új változatai jelenjenek meg, hiszen egy korábban feltörhetetlennek hitt függvény sérülékennyé vált, amit személyes adataink, jelszavaink kiszivárgását veszélyezteti.

Szakdolgozatom első felében a hash függvények általános felépítését, működését mutatom be. Majd a lehetséges támadásokat, valamint az egyes hash függvények már ismert gyengeségét fejtem ki.

1. KRIPTOGRÁFIAI HASH FÜGGVÉNYEK

1.1 DEFINÍCIÓ

A kriptográfiai hash függvény egy olyan függvény, mely egy tetszőleges bemenetből egy fix méretű adatsort, hash kódot állít elő. A H hash függvénynek az alábbi tulajdonságokkal rendelkeznie kell:

(i) tömörítés: tetszőleges hosszúságú M bemenethez, egy fix, n méretű $H(M)$ kimenetet rendel.

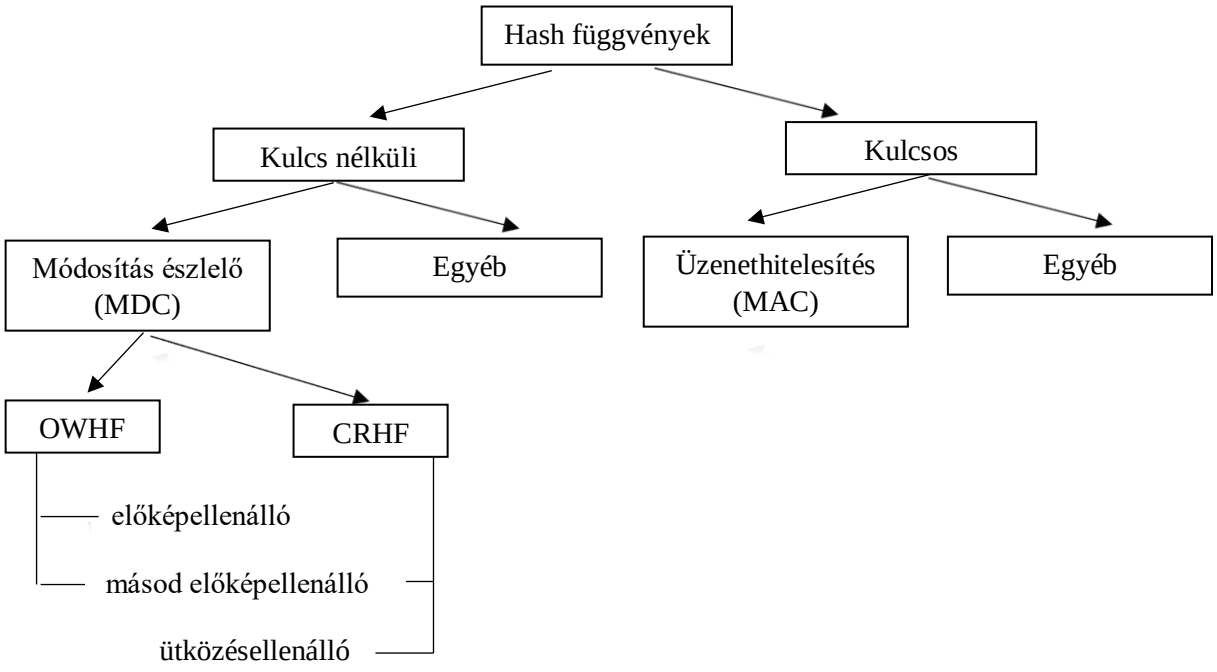
(ii) könnyű kiszámíthatóság: adott H és M esetén könnyű feladat $H(M)$-et kiszámítani. [1]

Az algoritmustól elvárjuk, hogy a lehető legjobban hasonlítson egy pszeudó random függvényhez, de determinisztikus maradjon.

Fentiekben szerepelt a könnyű feladat, amely alatt azt értjük, hogy egy probléma megoldható polinomiális időn belül, tehát az algoritmus lefut véges időn belül. Későbbiekben felmerül a nehéz feladat, amely alatt azt értjük, hogy egy algoritmus futtatása véges időn belül lehetetlen, vagy rendkívül sok időt igényel vagy túl nagy tárkapacitás szükséges.

1.2 CSOPORTOSÍTÁS

Felhasználásukat tekintve két fő csoportba sorolhatjuk a hash függvényeket: a kulcsos és a kulcsnélküli. A két csoport tovább bontható az ábrán látható módon. Ez csak egy lehetséges csoportosítási mód. Az ábrán található fogalmak a következő fejezetekben kerülnek kifejtésre, definiálásra.



1. ÁBRA HASH FÜGGVÉNYEK CSOPORTOSÍTÁSA [17]

1.2.1 KULCS NÉLKÜLI HASH FÜGGVÉNYEK

Buttyán Levente és Vajda István Kriptográfia és alkalmazásai című könyve alapján a hash függvények formálisan alábbi módon írhatóak le [2]:

1.2.1 Definíció: $H: \{0,1\}^* \to \{0,1\}^n$ hash függvény, amely $M \in \{0,1\}^*$ egy tetszőleges hosszúságú bináris bemenetet egy rögzített, n hosszú bináris sorozatba képez. Ezt az M bemenetet M üzenetnek, és a megfelelő $H(M)$ képtérbeli elemet hash értéknek vagy lenyomatnak nevezzük.

Felhasználásukat tekintve különböző igényeknek kell megfelelniük, ez alapján az alábbi tulajdonságok szoktak felmerülni:

(i) Egyirányú hash függvény (One Way Hash Function, továbbiakban OWHF): azaz könnyű feladat kiszámítani $H(M)$ lenyomatot bármely M esetén, de nehéz feladat a lenyomathoz tartozó üzenetet előállítani.

(ii) Előkép ellenálló (Preimage resistant): adott $H(x)$ lenyomathoz nehéz találni olyan y bemenetet, amelyre $H(x) = H(y)$ teljesül.

(iii) Másod előkép ellenálló (Second preimage resistant): adott $H(x)$ lenyomathoz nehéz találni y-t úgy, hogy $H(x) = H(y)$ és $x \neq y$.

(iv) Ütközésellenálló hash függvény (Collision Resistant Hash Function, továbbiakban CRHF): Nehéz feladat két olyan bemenetet találni, amelyek lenyomata megegyezik, azaz $H(x) = H(y)$ úgy, hogy $x \neq y$.

1.2.1 Megjegyzés: (definíció) Abban az esetben beszélünk univerzális egyirányú hash függvényről (Universal One Way Hash Function, UOWHF), ha lenyomatképző függvények egy $\{h_k\}_{k \in K}$ indexelt halmaza áll rendelkezésre, ahol az indexek egy K halmazból veszik az értékeiket, a tagfüggvényeknek azonos az ősképterük, illetve képterük van, továbbá:
Ha először választanunk kell egy M üzenetet, s ezután kerül csak véletlenszerűen kisorsolásra egy h_k függvény, akkor nehéz feladat olyan $M'(\neq M)$ üzenetet találni, amelyre $h_k(M') = h_k(M)$. [2]

Az UOWHF-et gyakrabban használjuk, vonzóbb, mint a CRHF, mert olcsóbb és egyszerűbb az előállítása. Számos területen elegendő biztonságot nyújthat, mint például a digitális aláírásnál.

1.2.2 KULCSOS HASH FÜGGVÉNYEK

A kulcsos hash függvények hasonlóan működnek, mint a kulcs nélküli társaik. Annyiban térnek el egymástól, hogy előbbinek két paramétere van, egy üzenet és egy titkos kulcs, és kimenetként szintén egy fix, n hosszú lenyomatot kapunk.

1.2.2 Definíció (kulcsolt hash függvény): Kulcsolt hash függvény tetszőleges hosszúságú M üzenetet és rögzített hosszúságú k kulcsot rögzített hosszúságú $H(k, M)$ lenyomatba képez le az alábbi tulajdonságokkal:

(i) könnyű kiszámítani $H(k, M)$ lenyomatot adott M és k esetén.

(ii) k kulcs ismerete nélkül nehéz kiszámítani a $H(k, M)$ értéket tetszőleges M üzenet esetén, még az esetben is, ha ismert $(M_i, H(k, M_i))$, $M \neq M_i$, $i = 1,2, \ldots$ párok állnak rendelkezésre.

(iii) nehéz a k kulcs kiszámítása $(M_i, H(k, M_i)), i = 1,2, \ldots$ párok ismeretében.

(iv) k kulcs ismerete nélkül fennáll az egyirányúság és a gyenge ütközésmentesség, azaz

a) ha adott y lenyomat, nehéz feladat olyan M üzenetet találni, amelyre $H(k, M) = y$

b) ha adott M és $H(k, M)$, nehéz olyan $M'(\neq M)$ üzenetet (második ősképet) találni, amelyre $H(k, M') = H(k, M)$ [2]

1.3 KAPCSOLAT AZ ÜTKÖZŐMENTESSÉG ÉS EGYIRÁNYÚSÁG KÖZÖTT

Amennyiben a lenyomat hossza n, akkor 2^n darab különböző lenyomat lehetséges, ami jóval kevesebb, mint a lehetséges üzenetek száma. Hash függvények nem injektívek, tehát két különböző üzenetnek lehet azonos a lenyomata. Célunk az, hogy nehéz feladat legyen legalább két olyan üzenetet találni, amelyek lenyomata megegyezik. Ezt nevezzük ütközésmentes hash függvényeknek.

Ha nem lenne nehéz, akkor magát az üzenetet küldő fél kísérelhetne meg olyan csalást, hogy olyan két azonos lenyomatú dokumentumot előállít, ahol az egyik az aláírásra szánt, a másik a tartalék arra az esetre, ha később nem lenne kedvező számára az először aláírt dokumentum.

1.3.1 Definíció: (Gyengén ütközés-ellenálló) A H hash függvény gyengén ütközés-ellenálló vagy második ősképe ellenálló, ha nehéz feladat előállítani egy $M'(\neq M)$ üzenetet (második ősképe), amelynek azonos a lenyomata, azaz melyre $H(M') = H(M)$. [2]

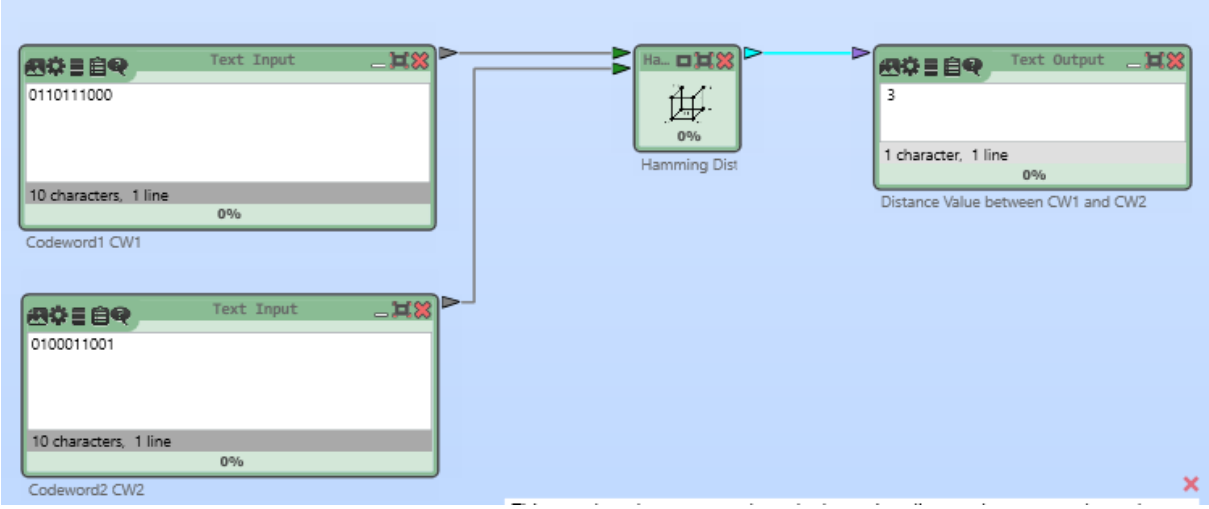
1.3.2 Definíció (Ütközés ellenálló): Egy H hash függvény ütközés-ellenálló, ha nehéz olyan M, M', $M' \neq M$, amelyre $H(M') = H(M)$. [2]

Ez a tulajdonság csak azt követeli meg, hogy nehéz legyen olyan üzenetpárt találni, amelyre az alábbi korreláció maximális.

M és M' bináris $2r$ hosszú üzenetek korrelációját $dist\big(H(M), H(M')\big) - r$ definiálja, ahol a $dist(x, y)$ a Hamming-távolság.

1.3.3 Definíció (Hamming-távolság): Legyen x és y azonos hosszúságú bináris jelsorozat, azaz $x, y \in \{0,1\}^n$. A Hamming távolság megmutatja a két bitsor különböző bitjeinek számát. Jele: $dist(x, y)$.

Például: $dist(01\underline{1}0\underline{1}11000, 01\underline{0}0\underline{0}11001) = 3$



2. ÁBRA HAMMING TÁVOLSÁG BEMUTATÁSA

Az alábbiakban vizsgálni fogjuk a gyengén ütközésmentes, ütközésmentes, valamint az egyirányúság tulajdonságok közötti kapcsolatot. Látni fogjuk, hogy az ütközésmentes tulajdonság a legerősebb. Először vizsgáljuk a gyengén ütközésmentes és ütközésmentes tulajdonságok szoros kapcsolatát.

1.3.1 Tétel: Egy H hash függvény akkor és csak akkor ütközésmentes, ha nehéz olyan M üzenetet találni, amelyre nem gyengén ütközésmentes.

Bizonyítás: Ha van hatékony algoritmusunk, amely H függvényhez felmutat ütköző párt, akkor ezen pár bármelyik tagja nem gyengén ütközőmentes, hiszen már ismerjük az ütköző párját. Megfordítva, ha van hatékony algoritmusunk, amely előállít egy nem gyengén ütközésmentes M üzenetet, ez – definíció szerint – azt jelenti, hogy nem nehéz feladat előállítani hozzá egy azonos lenyomatú $M'(\neq M)$ üzenetet. Ez viszont azt jelenti, hogy M, M' ütközőpár előállítás nem nehéz feladat. [2]

□

Tétel következménye, hogy a nem gyengén ütközésmentes üzenetek aránya kicsi az ütközőmentes hash függvények esetén. Az alábbi tétel az ütközőmentesség és egyirányúság tulajdonságok közötti kapcsolatára vonatkozik.

1.3.2 Tétel: Jelölje X illetve Y véges halmaz a H függvény ősképterét, illetve képterét, ahol $|Y| < |X|$. Tegyük fel, hogy létezik egy hatékony I invertáló algoritmus, amely tetszőleges lenyomatra, mint bemenetre, kiszámít egy üzenetet. Ekkor megadható egy hatékony algoritmus, amely legalább $1 - \frac{|Y|}{|X|}$ valószínűséggel ütköző üzenetpárt tud előállítani.

Bizonyítás: Tegyük fel, hogy H nem egyirányú. Legyen I egy hatékony invertáló algoritmus. Válasszunk véletlenül egy M üzenetet, s számítsuk ki az $M' = I(H(M))$ ősképet. Ha $M' \neq M$, akkor sikerült ütköző párt előállítani. Vizsgáljuk a $P(M' \neq M)$ sikervalószínűségét. Jelölje $\{M\}$ a $H(M)$ lenyomathoz tartozó ősképek halmazát. Válasszuk ki egy-egy reprezentáns elemet egy-egy ilyen ősképhalmazból. Legyen ezen reprezentánsok halmaza V, illetve a jelölésünknek megfelelően, ha v egy reprezentáns elem, akkor $\{v\}$ jelölje a v által reprezentált ősképhalmazt. Nyilván

$$P[M' \neq M|M] = (|\{M\}| - 1)/|\{M\}| \ ,$$

ahonnan egyenletes eloszlást feltételezve az üzenetek halmaza felett:

$$P[M' \neq M] = \frac{1}{|X|}\sum_{M} P[M' \neq M|M] = \frac{1}{|X|}\sum_{M}\frac{|\{M\}| - 1}{|\{M\}|}$$

$$= \frac{1}{|X|}\sum_{v\in V}\sum_{M\in\{v\}}\frac{|\{v\}| - 1}{|\{v\}|} = \frac{1}{|X|}\sum_{v\in V}(\{v\} - 1)$$

$$= \frac{1}{|X|}(|X| - |Y|) = 1 - |Y|/|X|$$

Az iterációs függvény tömörítő, ezért már egy blokknyi üzenet lenyomat képzésekor is nagy a siker valószínűsége. Ha például felére tömörít az iterációs függvény, akkor egy üzenetblokk esetén ½ valószínűséggel adódik az ütköző pár találati sikere. [2]

□

1.3.4 Definíció (Iterációs függvény): Az iterációs függvény alatt, egy olyan függvényt értünk, amely tartalmaz iterációt, azaz van olyan része a függvénynek, amelyet ismételten végrehajt az előző függvényértéken.

1.4 SZÜLETÉSNAPI PARADOXON [2]

Kérdés: „Mekkora a valószínűsége annak az eseménynek, hogy az emberek egy véletlenszerűen választott r fős csoportjában legalább két személynek ugyanazon a napon van a születésnapja?” A paradoxon abból fakad, hogy józan emberi sejtéssel nehezen érthető tény, hogy ahhoz, hogy ez a valószínűség $\frac{1}{2}$ körüli érték legyen elegendő $r = 23$ személyből álló csoportot választani.

Először oldjuk meg $\bar{P}$, P komplementerét, azaz mekkora esélye van egy embernek olyan dátumot választania, hogy az eltérjen az előzőekétől.

- első ember biztosan tud választani
- másodiknak van $\left(1 - \frac{1}{365}\right)$ esélye
- az r. embernek pedig $\left(1 - \frac{r-1}{365}\right)$

Így $\bar{P}$ valószínűsége az alábbi módon írható fel:

$$\bar{P} = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdot \ldots \cdot \left(1 - \frac{r-1}{365}\right)$$

Ezt közelítsük. Taylor-sorból tudjuk, hogy

$$e^x \approx 1 + x \qquad \text{ha } x \to 0$$

$$1 - \frac{1}{365} \approx e^{-\frac{1}{365}}$$

$$\bar{P} \approx 1 \cdot e^{-\frac{1}{365}} \cdot e^{-\frac{2}{365}} \cdot \ldots \cdot e^{-\frac{r}{365}} = 1 \cdot e^{\frac{-(1+2+\cdots+r)}{365}} = e^{-\frac{r\cdot(r+1)}{2\cdot 365}} \approx e^{-\frac{r^2}{2\cdot 365}}$$

Tehát P valószínűsége az alábbi módon írható fel:

$$P = 1 - e^{-\frac{r^2}{2\cdot 365}} = 1 - e^{-\frac{r^2}{2\cdot m}}$$

Tetszőleges elemszámú halmaz esetén, amelyből választhatunk, akkor azt jelölje m. Esetünkben $m = 365$. Az $r = \sqrt{m}$ választása esetén,

$$P \approx 1 - e^{-\frac{1}{2}} \approx 0{,}4\,.$$

Tehát, ha van egy tetszőleges nagyméretű halmazunk, melynek elemeit m-féle különböző skatulyába sorolhatunk, akkor m négyzetgyökének nagyságrendjébe eső méretű részhalmazt választva már 50% körül van annak a valószínűsége, hogy a választott r részhalmazban lesz legalább kettő olyan elem, amely ugyanabba a skatulyába tartozik.

Továbbiakban módosított formában fogjuk használni a paradoxont.

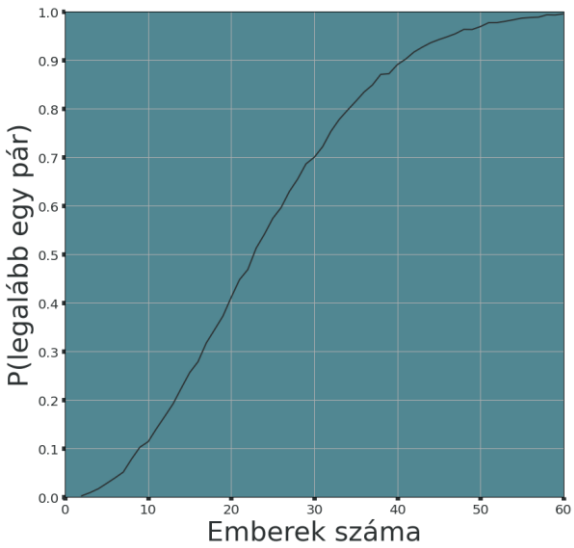
Új kérdés: „Mekkora a valószínűsége annak, hogy egy U nagy m méretű alaphalmazból véletlenszerűen választva V és W r méretű részhalmazokat, azok metszete nem üres?”

A P' valószínűséget az alábbi módon közelíthetünk: [2]

$$P' = 1 - \frac{\binom{m}{2r} \cdot (2r)!}{\left(\binom{m}{r} \cdot r!\right)^2} \approx 1 - e^{-\frac{3r^2}{m}}.$$

Az $r = \sqrt{m}$ választása esetén, a $P' \approx 1 - e^{-3} \approx 0.95$ közelítést kapjuk.

A születésnapi paradoxon jelentőségét a hash függvények kapcsán az adja, hogy ennek segítségével becsülhető meg egy ütközés találásának valószínűsége. Ugyanis, ha H hash függvény kimenete n bites, akkor a paradoxon alapján $2^{\frac{n}{2}}$ véletlenül választott üzenet között $\frac{1}{2}$-hez közeli valószínűséggel lesz kettő olyan, amelynek hash értéke megegyezik, azaz ütközőpárt alkot. Tehát a kimenetet úgy kell méretezni, hogy $2^{\frac{n}{2}}$ megfelelően nagy legyen, így az ütköző-ellenállóságot biztosítani tudjuk. Ennek következtében a véletlen választással történő ütköző pár keresés gyakorlatilag kivitelezhetetlen feladattá válik. Ma jellemzően $n = 160$ bites kimenetet jelent.



3. ÁBRA SZÜLETÉSNAPI PARADOXON [14]

1.5 LAVINAHATÁS

A kriptográfiában a lavinahatás (avalanche effect) egy kívánatos tulajdonsága a kriptográfiai algoritmusoknak, jellemzően a blokk kódoló algoritmusok és a hash függvények esetében. Lavinahatás alatt azt értjük, hogy a bemenet kis megváltoztatása esetén a kimenet jelentősen megváltozik. Jó minőségű blokk kódoló algoritmusoknak rendelkeznie kell ezzel a tulajdonsággal. Ha a kriptográfiai algoritmus nem mutat erőteljes lavinahatást, akkor gyengén véletlenszerűsít, így megsejthető a bemenet, ennek következtében részlegesen vagy teljes mértékben feltörhető az algoritmus.

Ez az egyik legfontosabb tervezési szempont, ezért a hash függvények nagy adatblokkokon végzik műveleteiket, így bármely kis megváltoztatás gyorsan elterjed az algoritmus iterációiban, így minden kimeneti bit függ, minden bementi bittől.

1985-ben A. F. Webster és Stafford E. Tavares bevezetett két fogalmat a lavina hatás pontosabb definiálására. Ezeket a DES kódoló algoritmus kapcsán vezették be, de a hash függvények esetében is értelmezhetőek.

(i) Szigorú lavina kritérium (strict avalanche criterion, SAC): A bemenet egy bitjének megváltoztatására a kimeneten minden bit ½ valószínűséggel változzon meg.

(ii) Bit függetlenségi kritérium (bit idependence criterion, BIC): Legyen j és k két különböző kimeneti bit. Ezeknek egymástól függetlenül kell megváltozniuk, bármely i bemeneti bit megváltoztatása esetén. [3]

Az alábbi példa a lavina-hatást szemlélteti, ahol a bemeneti értékek egy karakterben térnek el egymástól, azaz Hamming-távolságuk 1. Szöveges és bináris üzeneteket is vizsgálva az alábbi értékeket kaptam:

- MD5:

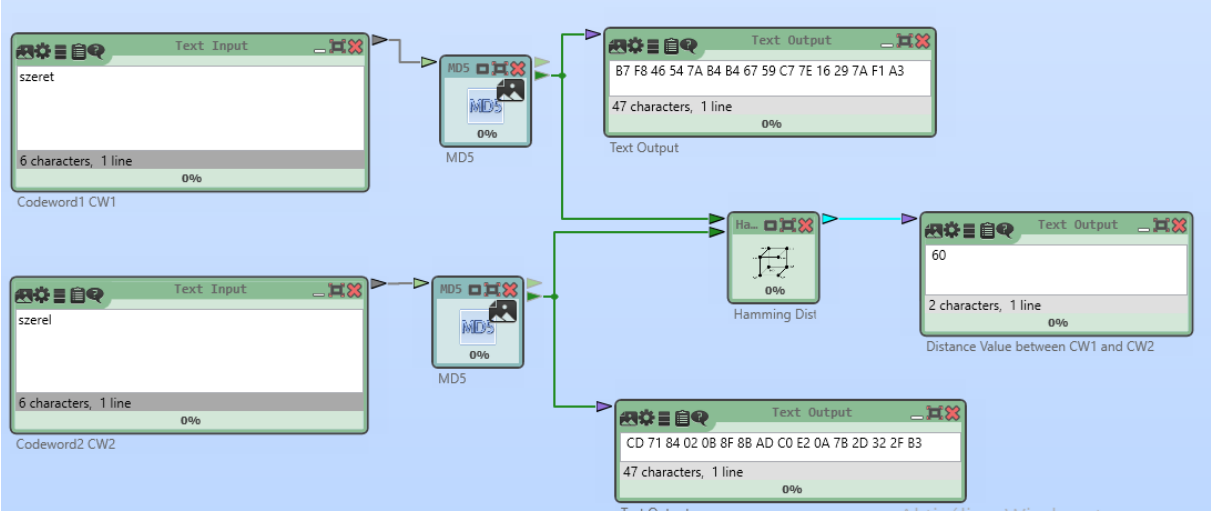
Bemenet	Kimenet	Hamming-távolság
szeret	B7 F8 46 54 7A B4 B4 67 59 C7 7E 16 29 7A F1 A3	60
szerel	CD 71 84 02 0B 8F 8B AD C0 E2 0A 7B 2D 32 2F B3	
001001	1F CC B5 67 A4 48 80 E8 66 5B 7C B9 D0 F9 72 71	62
001000	EC CB 22 FE 1A CF 40 27 64 92 39 C7 D6 38 3D 78	

- SHA-256:

Bemenet	Kimenet	Hamming-távolság
szeret	51 CF DD 14 26 97 D7 8B 49 05 C8 89 48 08 11 3A 35 FF 7B 69	87
szerel	7C F1 32 DF C7 9C 21 D1 6D A8 EB D6 AE CF 71 8F 9E 6D EF 98	
001001	17 D5 36 96 DC 12 FD D0 D0 6A BE 10 A9 03 BA 53 B2 85 18 EE	90
001000	F8 72 A2 5A 13 88 D1 11 D7 D3 C5 64 92 A1 C7 9E BD EA D6 7F	

4. ÁBRA PÉLDA LAVINA EFFEKTUSRA

A táblázatban szereplő eredményeket a Cryptool2 alkalmazás használatával kaptam az alábbi módon.

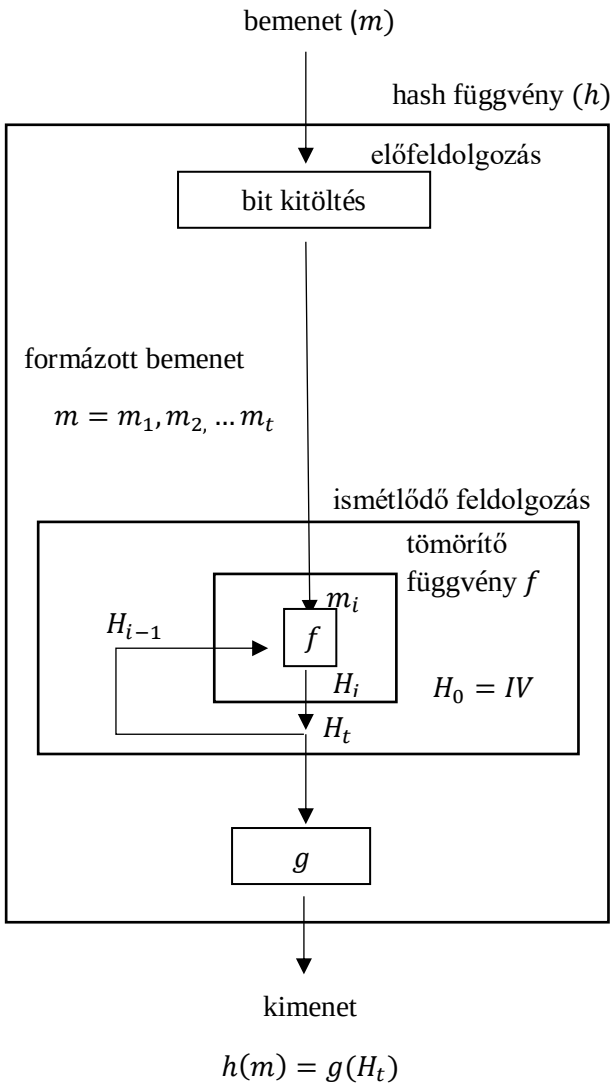


5. ÁBRA HAMMING TÁVOLSÁG PÉLDÁJA

2. HASH FÜGGVÉNY KONSTRUÁLÁSA

2.1 ÁLTALÁNOS MODELL

A legtöbb kulcsnélküli hash függvény működése egy iteratív folyamatra épül, amely egy tetszőleges méretű bemenetet, egymást követő fix méretű blokkokra hasít. Az alábbi ábra ennek szemléltetésére szolgál:



6. ÁBRA HASH FÜGGVÉNY ÁLTALÁNOS MODELLJE

Az előfeldolgozási folyamat során a bemenetet r hosszúságú blokkokra bontjuk (m_i). A legtöbb esetben ehhez bitkitöltésre van szükségünk, amelyet úgy hajtunk végre, hogy a bitkitöltés után a teljes hosszúság a blokkhosszúság többszöröse legyen. Az ismétlődő rész abból áll, hogy az f tömörítő függvényt i-szer futtatjuk úgy, hogy az f függvény bemenete m_i blokk és az előző köztes érték. A köztes érték az f függvény n hosszú kimenete. Az első blokk esetén köztes értékkel még nem tudunk számolni, ezért megadunk egy kezdő értéket, H_0-t. Ha minden i blokkon lefutattuk a függvényt, akkor megkapjuk az n hosszú lenyomatot.

A folyamat az alábbi módon írható le:

$$H_0 = IV;$$

$$H_i = f(H_{i-1}, m_i), \quad 1 \leq i \leq t;$$

$$h(x) = g(H_t),$$

ahol:

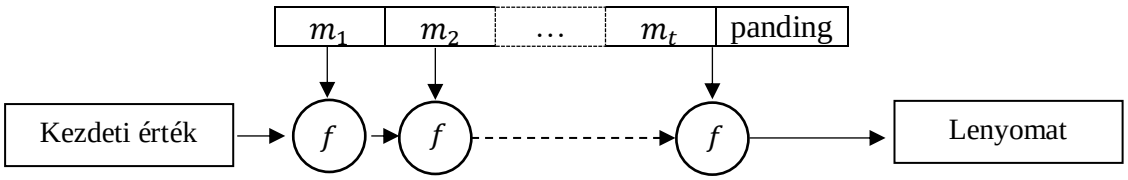
- $H_0 = IV$: előre meghatározott kezdeti érték
- H_{i-1}: n bit hosszú változó $i-1$ és i szakasz között
- g: kimeneti transzformáció, gyakran önazonosság $g(H_t) = H_t$

Az egyes hash függvények az előfeldolgozás, a tömörítő függvény és a kimeneti transzformáció jellegében térnek el.

2.2 MERKLE-DAMGÅRD KONSTRUKCIÓ

1979-ben Ralph Merkle és Ivan Damgård egymástól függetlenül bebizonyították, hogy ha egy tömörítő függvény (f) ütközésmentes, akkor az erre a függvényre épülő hash függvény is ütközésmentes marad. Az üzenet megfelelő kiegészítésére is javaslatokat tettek, valamint arra, hogy milyen szempontokra érdemes figyelni mielőtt hash függvényt tervezünk. Könnyű látni, hogyha f függvény például nem ütközésmentes, akkor a hash függvény sem lesz az.

Tehát az iterációs konstrukció alapja egy $f: \{0,1\}^{2n} \rightarrow \{0,1\}^n$ iterációs (kompressziós) függvény.



7. ÁBRA ITERÁLT KRIPTOGRÁFIAI HASH FÜGGVÉNY [2]

Mint ahogy korábban már láttuk, két bemeneti értéke van a kompressziós függvénynek, egy köztes érték, amit az első lépésnél adunk meg, úgy nevezett inicializációs vektort *(IV)*, valamint egy üzenet blokk. Formálisan az alábbi módon írhatjuk fel:

$$H(m) = f(h_i, m_i), \text{ ahol } h_i = \begin{cases} IV & ha\ i = 0 \\ f(h_{i-1}, m_{i-1}) & ha\ i > 0 \end{cases}$$

Tehát H hash függvény egy iterált függvény, amely f tömörítő függvényt iterálja az üzenet m_i blokkjain.

2.2.1 Tétel (Merkle-Damgård-kiegészítés) [2]: Tegyük fel, hogy van egy f kompressziós függvényünk ütközésellenálló tulajdonsággal. Ha az $M = [m_1, m_2, \dots, m_{t-1}]$ üzenetet kiegészítjük egy m_t blokkal, amely tartalmazza az M üzenet bithosszát (nulla bitekkel kiegészítve), akkor az f kompressziós függvényre épülő iterációs hash függvény is ütközőmentes lesz.

Bizonyítás: Indirekt. Tegyük fel, hogy elő tudunk állítani egy M, M' ütköző üzenetpárt, azaz $H(M) = H(M')$, de $M \neq M'$, ahol az üzenetbe nem értjük még bele a kiegészítést. Két esetet különböztetünk meg, amikor a két üzenet azonos méretű, vagy amikor különböző.

Első eset. Legyen $M = [m_1, m_2, \dots, m_t], M' = [m'_1, m'_2, \dots, m'_t]$. Feltételezésünk alapján a kimenet mindkét esetben megegyezik: miután azonos bithosszúak, így a t-edik blokk azonos, ezért a H ütközésellenálló (CRHF) tulajdonsága miatt ez csak úgy lehetséges, ha a köztes $(t-1)$. iterációs lépés kimenete is azonos. Viszont ez megint csak úgy lehetséges a H CRFH tulajdonsága miatt, ha a $(t-2)$. iterációs lépés kimenete is azonos, és így tovább. Így haladva visszafelé kell találnunk egy j -edik blokkot, amelyben a két üzenet eltér egymástól. Így viszont ellentmondásra jutunk, mert kell, hogy a $(j-1)$. köztes érték azonos legyen, viszont akkor ütközés állna elő a kompressziós függvényre.

Második eset. Legyen $M = [m_1, m_2, \dots, m_r], M' = [m'_1, m'_2, \dots, m'_s]$ a két különböző hosszúságú üzenetpár, tehát az utolsó blokkok különbözőek. Így semmiképp sem állhat elő ütközés a kimeneten, ellenkező esetben ellentmondásba kerülnénk a H ütközésmentes tulajdonságával.

□

3. TÁMADÁSOK

Korábban már említettem egy-két lehetséges feltörési lehetőséget, hogy miként használható ki, ha az adott hash függvény nem felel meg teljes mértékben minden kritériumnak. A hash függvények esetében két fajta támadást különbözetünk meg, az ütközés elleni és az üzenet kiterjesztési támadásokat.

Az ütközés elleni támadásokat (collision attack) is tovább csoportosíthatjuk.

Tegyük fel, hogy az adott hash függvény nem ütközés ellenálló, tehát megpróbálhatunk x és y bemeneteket úgy keresni, amelyre $H(x) = H(y)$, de $x \neq y$. Ha erre tényleg képesek vagyunk, akkor létrehozhatunk két dokumentumot, amelyek egyike aláírásra kerül, majd később a számunkra kedvezőt tudjuk használni, hiszen a hash lenyomatuk megegyezik. Emiatt érdemes aláírásnál, csak egy-két bitben módosítani az aláírandó dokumentumot, mert így a hamis dokumentum haszontalanná válik, hiszen a lavina hatás következményeképp a hash értékben nagy változást érünk el.

Ha a hash függvény nem előkép ellenálló, azaz, ha ismerjük $H(x)$-et, akkor létre tudunk hozni egy x' dokumentumot, amelyre $H(x) = H(x')$. Ez esetben például megpróbálhatunk más jelszóval is hozzáférni az adatokhoz. Hiszen ez esetében a jelszavak hash értékét tárolják csak, így biztosítva, hogy csak a pontos jelszóval tudjunk belépni. Általában jelszótárolásnál nem csak hash függvényt használnak, hanem úgy nevezett só (salt) értékkel is számolnak, azt is külön táblában kell tárolni.

Miért van szükség sózásra? Mert ha csak a jelszavak hash értéke van tárolva, akkor brute force támadással próbálkozhatunk. Vegyünk egy 8 karakterből álló jelszót, amely az angol ABC 26 karakteréből és számokból állhat. Ez $(26 \cdot 2 + 10)^8$, ami körülbelül 220 milliárd. Egy modern számítógép több millió hasht tud előállítani másodpercenként, így körülbelül 8-9 nap alatt megvan a kívánt jelszó. Ezt az időt tovább lehet rövidíteni, ha egyszerre több jelszót is fel akarunk törni. Emellett az interneten más számos kész hash-táblázatokat lehet találni, amelyeket szivárvány tábláknak hívunk. Egy ilyen adatbázis nem csak értelmező kéziszótári szavakat, hanem akár egy korábban kiszivárgott jelszavak hash lenyomatát is tartalmazhatják.

Sózással azt akarjuk elérni, hogy ha két felhasználó azonos jelszót is alkalmaz, különböző hash érték legyen eltárolva, ezzel is nehezítve a feltörést és a szivárvány táblák használatát. A só alatt egy egyedi értéket értünk, amely felhasználónként különböző. Ezt a jelszóhoz szerkesztjük a hash függvény alkalmazása előtt. Ezt a só értéket egy egyszerű plain textben tárolhatjuk, külön titkosítást nem igényel, mert önmagában semmilyen értéket nem képvisel.

Tegyük fel, hogy ismerjük az x bemenetet és a hozzá tartozó $H(x)$ lenyomatot és így próbálunk meg egy y találni, amelyre $H(x) = H(y)$. Némi segítséget nyújthat x ismerete.

Az üzenet kiterjesztő támadás (length extension attack) alatt azt értjük, mikor a támadó ismeri a $H(x)$ lenyomatot, valamint az üzenet hosszát, de magát az x-et nem. Ekkor könnyen előállítható egy $H(x||x')$, ahol x'-et a támadó befolyásolja. Célunk nem az x üzenet visszafejtése, hanem egy már aláírt dokumentum kiegészítése hiteles aláírással.

Miként is tehetjük ezt meg? Ismerjük a $H(x)$ értéket és x hosszát, ekkor könnyedén ki lehet számolni p, panding, azaz a kitöltés értékét. Következő lépésben az x'-t úgy alkotom meg, hogy $x||p$-vel kezdődjön, így $x' = x||p||z$, ahol z tetszőlegesen választott bitsor. Ekkor tudjuk azt,

hogy a hozzácsatolt üzenetig a belső hash lenyomat megegyezik az eredeti üzenetével, valamint a hash függvény teljes állapotát. Innen már triviális kiszámolni az inicializációs vektort és módosítani a függvényen úgy, hogy az új aláírás megegyezzen az eredetivel az eredeti kulcs ismerete nélkül.

További támadási módszerek is léteznek, melyek ezeket a gyengeségeket próbálják kihasználni a támadás során. Az alábbiak például szolgálnak.

- Brute force-támadás: A támadás során az összes lehetséges kulcsot kipróbálva keressük az alkalmazott kulcsot. A törési idő függ a hardver kapacitásáról, valamint attól, hogy tudjuk-e, hogy mekkora kulcsot keresünk, vagy jelszó esetén milyen karaktereket tartalmaz.

- Szivárvány tábla (rainbow tables): Olyan adatbázisok, amelyek vagy szótári szavakat, vagy korábban kiszivárgott valós jelszavak hash lenyomatait tartalmazza, ezzel is rövidítve a brute force-támadás idejét.

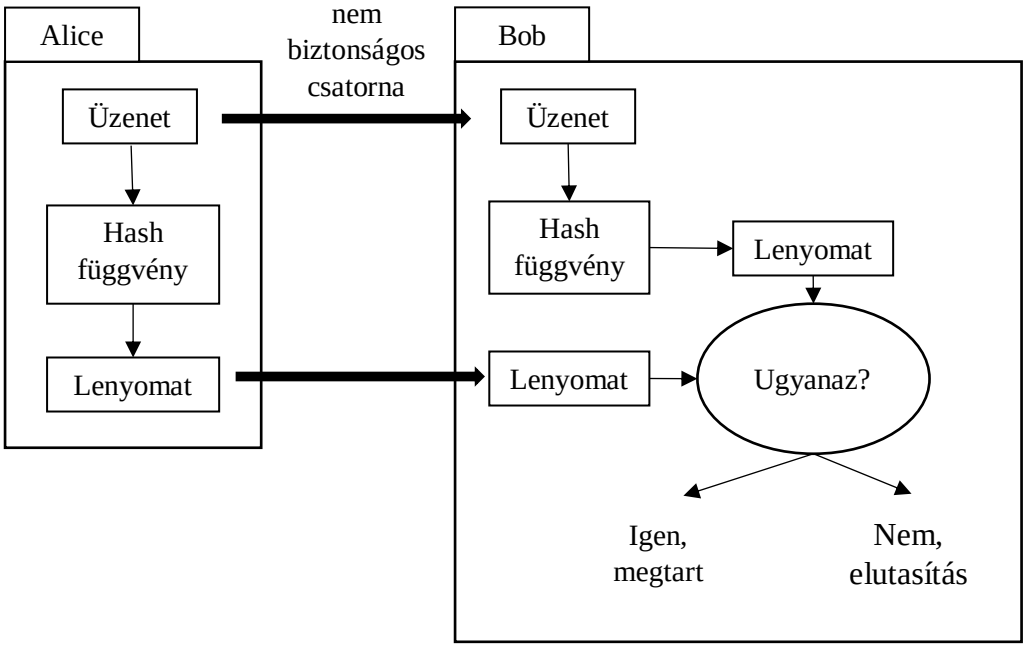
- Time/memory/data tradeoff: Lényege, hogy egy olyan helyzetet próbálunk teremteni, ami a lehető legjobban hasonlít a „space-time tradeoff”-hoz,. Ez azt jelenti, hogy megpróbálunk megoldani egy problémát vagy kevés hellyel, de hosszú idő alatt vagy fordítva. Ez főként adat tárolásnál használatos. A „time/memory/data” (idő/tárhely/adat) esetén a támadónak rendelkezésére áll valamennyi adat, melynek függvényében próbál támadni. Ez a fajta támadás nagyon nehéz, ezért a legtöbb titkosítási sémát nem úgy tervezték, hogy ellenálljanak neki.

- …

4. MD- CSALÁD

Az MD (Message Digest) függvény család a kulcs nélküli hash függvények közé tartozik. Célja, hogy biztosítja az üzenet sértetlenségét az alábbi módon:

Legyen Alice, aki üzenetet küld Bobnak egy nem biztonságos csatornán keresztül. Alice és Bob is kiszámolja az üzenet lenyomatát. Alice a lenyomatot szintén elküldi Bobnak, így Bob össze tudja hasonlítani a két kapott lenyomatot, amelyek ha megegyeznek, akkor az üzenet sértetlen.

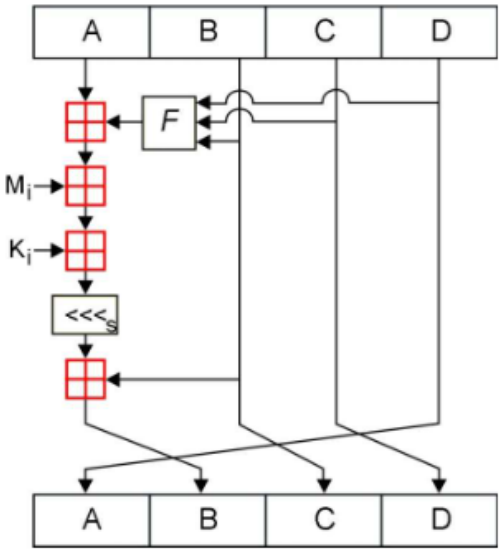


8. ÁBRA ÜZENETHITELESÍTÉS

4.1 MD5

MD család talán legismertebb tagja az MD5. 1991-ben fejlesztette ki Ronald L. Rivest, az RSA egyik megalkotója, az MD4 lecserélésére. Már 1996-an Dobbertin bebizonyította, ha a kezdő inicilációs vektor szabadon választható, akkor két különböző bemenetre megkapható ugyanaz a lenyomat, ennek következtében más hash algoritmusok használatát javasolták. Az igazi áttörést Wand és Yu hozta 2004 augusztusában a Crypto konferencián. Kínában publikáltak egy tanulmányt, amely bemutatta miként lehet ütközést találni négy algoritmusban (MD4, MD5, Haval-128 és RIPEMD) „könnyedén”, amely gyorsabb, mint a brute-force. 2005 óta elektronikus aláírásra nem javasolt, illetve 2010 óta az USA-ban az SHA függvény család alkalmazása javasolt.

Az algoritmus szerkezetét tekintve hasonlít elődjére, az MD4-re. Az algoritmus működése:



9. ÁBRA MD5 ALGORITMUS [1]

Az algoritmus tetszőleges méretű bementből egy 128 bites lenyomatba képez. A bementet 512 bites darabokra bontja, majd az utolsó blokkot feltöltjük egy 1-es bittel majd 0-kal úgy, hogy az utolsó 64 bites értéknek a bemenet hosszát kell tárolnia. Ezt a fajta bitkitöltési formát Merkle megerősítésnek szokás nevezni. Minden egyes blokkot tovább bontunk 32 bites „szavakra": $M_i = (m_0, m_1, m_2, \dots, m_{15})$. A tömörítő függvény, amely minden M_i blokkon lefut négy körből áll, és mindegyik kör 16 művelettel rendelkezik. Az egy körön belül lefutó négy művelet a fenti ábrán látható, mely tizenhatszor fut le. Az algoritmus tartalmaz egy F nem lineáris függvényt, $mod(2^{32})$ összeadást és egy balra forgatás műveletet, mely S bittel eltolja balra a bitsorozatot és a kilépő biteket a jobb oldalon visszahelyezi.

A kezdeti A, B, C és D 32 bites rögzített konstansok.

Minden körben négy különböző nem lineáris függvényt használunk, amelyek az alábbiak:

$$F_1(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z),$$

$$F_2(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z),$$

$$F_3(X, Y, Z) = X \oplus Y \oplus Z,$$

$$F_4(X, Y, Z) = Y \oplus (X \vee \neg Z),$$

ahol X, Y, Z 32 bites szavak.

Az algoritmus pszeudó-kódja [4]:

```
// Segédváltozók
var int[64] r, k

// az itt megadott értékekkel fogja eltolni a bitsorozatot adott lépésben
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 egész részének használat
for i from 0 to 63
k[i] := floor(abs(sin(i + 1)) * (2 pow 32))

// Inicializációs Vektor (A, B, C, D)
var int h0 := 0x67452301
var int h1 := 0xEFCDAB89
var int h2 := 0x98BADCFE
var int h3 := 0x10325476

// Merkle-Damgård előírások teljesítése
„1” bit hozzáfűzése a bemenethez
„0” bitek hozzáfűzése a bementhez, míg üzenet hossza≡448 (mod 512)
üzenet hosszának mértéke hozzáfűzés az üzenethez (64bit)

// A bemenet 512bitenként való feldolgozása:
32bites méretű szavakra bontás: w[i], 0≤i≤15

      // belső IV-k beállítása
      var int a := h0
      var int b := h1
      var int c := h2
      var int d := h3

      // Fő ciklus (4 kör)
      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

      // kör végén az IV-k változtatása a transzformált IV-kkel
      h0 := h0 + a
      h1 := h1 + b
      h2 := h2 + c
      h3 := h3 + d

//utolsó lépésben a végső IV-k összefűzése
var int digest :=h0 append h1 append h2 append h3
```

4.1.1 ÜTKÖZÉSKERESÉS MD5 HASH FÜGGVÉNYBEN

Ma már tudjuk, hogy az MD5 algoritmus nem ütközés ellenálló, emellett az üzenet kiegészítéses támadásnak sem áll ellen. Alább ismertetni fogom Wang és Yu módszerét, amellyel számos ütközés található bármely IV esetén, előbb viszont ismertetni szeretném a különbségen alapuló támadást (differential attacks) a hash függvényeken, mert később fel fogjuk ezt használni.

A következő két alfejezetben Xiaoyu Wang és Hongbo Yu How to Break MD5 and Other Hash Functions című cikkét használtam fel.

4.1.1.1 „DIFFERENTIAL ATTACK" [5]

A „különbség"-nek két paramétere van, X és X', a különbség pedig jelentse a következőt: $\Delta X = X - X'$. Bármely két l -bit hosszú üzenetre: $M = (M_0, M_1, \dots, M_{k-1})$ és $M' = (M'_0, M'_1, \dots, M'_{k-1})$, valamint k a blokkok indexe. Hash függvényekre az alábbi módon értelmezzük a különbséget:

$$\Delta H_0 \xrightarrow{(M_0, M'_0)} \Delta H_1 \xrightarrow{(M_1, M'_1)} \dots \Delta H_{k-1} \xrightarrow{(M_{k-1}, M'_{k-1})} \Delta H,$$

ahol ΔH_0 a kezdeti vektor különbség $(IV - IV)$, azaz 0. ΔH a lenyomatok különbsége. $\Delta H_i = \Delta IV_i$ az i-edik lépés utáni különbség a két üzenet lenyomata között. Tehát ütközésről akkor beszélünk, ha $\Delta H = 0$.

Az MD5 függvénye 4 körből áll és mindegyiknek van 16 művelete. A „round differental" (kör különbség) $\Delta R_{j-1} \longrightarrow \Delta R_j$ $(j = 1, 2, 3, 4)$ P_j valószínűséggel az alábbi:

$$\Delta R_{j-1} \xrightarrow{P_{j1}} \Delta X_1 \xrightarrow{P_{j2}} \dots \xrightarrow{P_{j16}} \Delta X_{16} = \Delta R_j,$$

ahol $\Delta X_{t-1} \xrightarrow{P_{jt}} \Delta X_t$, $t = 1, 2, \dots, 16$ a j-edik kör t-edik lépés különbsége.

4.1.1.2 WANG ÉS YU MÓDSZER [5]

Legyen a két bemeneti üzenet 1024bites (M_0, M_1) $és$ (M'_0, M'_1), valamint vegyük az alapértelmezett IV_0:

$$A_0 = 0x67452301, B_0 = 0x3B341042, C_0 = 0x98badcfe, D_0 = 0x10325476$$

Kell:
$$\Delta H_0 \xrightarrow{(M_0, M'_0)} \Delta H_1 \xrightarrow{(M_1, M'_1)} \Delta H = 0,$$

ahol

$$M'_0 = M_0 + \Delta M_0, \quad \Delta M_0 = (0,0,0,0, 2^{31}, \dots, 2^{15}, \dots, 2^{31}, 0)$$

$$M'_1 = M_1 + \Delta M_1, \quad \Delta M_1 = (0,0,0,0, 2^{31}, \dots, -2^{15}, \dots, 2^{31}, 0)$$

(nem nulla érték a 5., 12. és 15. indexeknél)

$$\Delta H_1 = (2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25}) = (\Delta a, \Delta b, \Delta c, \Delta d),$$

Ezt az eredményt kapjuk az első iteráció után ΔH_1-re. ΔM_0-t úgy választjuk meg, hogy nagy valószínűséggel előálljon „3-4 round differential” (3-4 kör különbség), ΔM_1-nek rendelkeznie kell az előbbi tulajdonsággal, valamint biztosítania kell, hogy előálljon ΔH_1.

Ezeket felhasználva a következőképp írható le a folyamat: válasszunk egy tetszőleges M_0 üzenetet, majd módosítsuk az üzenetet Wang által ismertetett módon, melyet a „How to Break MD5 and Other Hash Functions” című cikk 4.4. fejezetében ismerhetnek meg. Ekkor M_0 *és* $M'_0 = M_0 + \Delta M_0$ által létrejön az első iterációs különbség 2^{-37} valószínűséggel: $\Delta M_0 \rightarrow (\Delta H_1, \Delta M_1)$. Majd tesztelni kell, hogy az adott karakterisztikák továbbra is teljesüljenek, ha alkalmazzuk a tömörítő függvényt M_0 és M'_0-n.

Ismételjük a következőket addig, amíg ütközést nem találunk. Válasszunk egy random M_1 üzenetet, ezt szintén módosítsuk az adott módon, majd M_1és $M'_1 = M_1 + \Delta M_1$ által létrejön a második iterációs különbség 2^{-30} valószínűséggel: $(\Delta H_1, \Delta M_1) \rightarrow \Delta H = 0$. Végül tesztelni, hogy ez valóban ütközéshez vezetett.

Mai technológiával maximum 1 órán belül kiszámolható ilyen M_0 és M'_0, majd a hozzájuk tartozó M_1 és M'_1, amelyre az ütközés előáll. Ha találunk az üzenet bármely részén két ilyen blokk párt, amelyre az ütközés előáll, akkor az üzenet is ütközni fog.

Példa ütközésre, azaz $M \neq M'$ esetén $H(M) = H(M')$-re:

1. exe file:

   Legyen M egy olyan program, mely a következő üzenetet írja ki: „*Hello, world!*”
   M' pedig szintén egy üzenet kiíró program, a következő üzenettel:

   „*This program is evil!!!*
   Erasing hard drive…1Gb…2Gb… just kidding!
   Nothing was erased.”

   Ezeket a windows parancssorban futtattam, majd MD5 hash lenyomatukat vizsgáltam. Valóban ugyanazt a hash értéket kaptam:
   `cdc47d670159eef60916ca03a9da007`

```
C:\Users\User\Downloads>hello.exe
Hello, world!

(press enter to quit)certutil -hashfile hello.exe MD5

C:\Users\User\Downloads>certutil -hashfile hello.exe MD5
MD5 hash of hello.exe:
cdc47d670159eef60916ca03a9d4a007
CertUtil: -hashfile command completed successfully.

C:\Users\User\Downloads>erase.exe
This program is evil!!!
Erasing hard drive...1Gb...2Gb... just kidding!
Nothing was erased.

(press enter to quit)c

C:\Users\User\Downloads>certutil -hashfile erase.exe MD5
MD5 hash of erase.exe:
cdc47d670159eef60916ca03a9d4a007
CertUtil: -hashfile command completed successfully.
```

10. ÁBRA ÜTKÖZÉS ELLENŐRZÉSE PARANCSSORBAN [15]

2. jpg file:

   Érdekességképp, a következő két kép hash értéke szintén megegyezik, valamint nem csak lenyomatuk ütközik, de témájuk is.

   H(plane.jpg)=H(ship.jpg)=`253dd04e87492e4fc3471de5e776bc3d`

```
C:\Users\User\Documents\Matek\SZAKDOGA>certutil -hashfile plane.jpg MD5
MD5 hash of plane.jpg:
253dd04e87492e4fc3471de5e776bc3d
CertUtil: -hashfile command completed successfully.

C:\Users\User\Documents\Matek\SZAKDOGA>certutil -hashfile ship.jpg MD5
MD5 hash of ship.jpg:
253dd04e87492e4fc3471de5e776bc3d
CertUtil: -hashfile command completed successfully.
```

11. ÁBRA ÜTKÖZÉS ELLENŐRZÉSE WINDOWS PARANCSSORBAN

12. ÁBRA ÜTKÖZÉSRE PÉLDA [16]

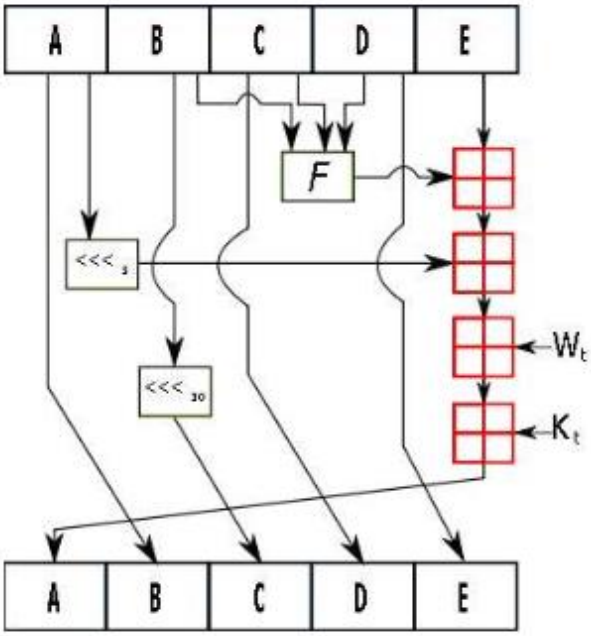
5. SHA FÜGGVÉNY CSALÁD

SHA függvény család (Secure Hash Algorithms) első tagja, az SHA-0, 1993-ban jelent meg és a National Institue of Standards and Technology (NIST) publikálta az USA-ban. Főként digitális aláírásra szánták, ám más területen is használják. Hamar fényderült gyengeségeire, ezért visszahívták és megszületett az SHA-1, mely felépítését tekintve hasonlít a már ismertetett MD5-re, de jóval elterjedtebb volt. 2010 óta az MD5-höz hasonlóan ütközés előállítható, így nem ajánlják használatát sem. Azóta megjelentek az SHA-2 és SHA-3 függvény családok is, hiszen ezek is több függvényt foglalnak magukba. Például SHA-2-höz tartozik az SHA-256 és SHA-512.

5.1 SHA-1

A leglényegesebb újítás az MD5-höz képest, hogy az eddigi 128 bites kimenet helyett, 160 bites kimenetbe képez, ugyanakkor megmaradt, hogy 512 bit-es blokkokkal dolgozik. Az algoritmus működése a bal oldali ábrán látható.

Az algoritmus a Merkle-Damgård elképzelés szerint kibővíti az utolsó nem 512 bit hosszú üzenet blokkot. Minden blokkot az MD5-höz hasonlóan tovább bontja 16 darab 32 bit méretű „szóra”, emellett még készít 64 darab szintén 32 bit méretű értéket az eredeti 16db blokkból. Ez összesen 80 darab tárolt érték, melyet 80 lépésben transzformál a 4 kört megtartva, azaz minden körben 20-20 lépés, az eddigi 16 lépés helyett. Végül a kapott 5 transzformált értéket a kezdeti *IV*-hez adja, így megkapva az 5·32 bites, azaz 160 bites kimenetet.



11. ÁBRA SHA-1 ALGORITMUS [1]

A 4 használt függvény az alábbi [6]:

$$F_i(X,Y,Z) = (X \wedge Y) \vee \big((\neg X) \wedge Z\big), \qquad ha\ \ 0 \le i < 20$$

$$F_i(X,Y,Z) = X \oplus Y \oplus Z, \qquad ha\ 20 \le i < 40$$

$$F_i(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), \quad ha\ 40 \le i < 60$$

$$F_i(X,Y,Z) = X \oplus Y \oplus Z, \qquad ha\ 60 \le i < 80,$$

ahol i a lépésszám. Ha megvizsgáljuk a négy függvényt, akkor észrevehető, hogy az első és a második megegyezik az MD5-nél használatossal, valamint, ha $20 \le i < 40$ vagy $60 \le i < 80$, akkor a függvények megegyeznek.

Az algoritmus pszeudó-kódja [7]:

```
//IV-k beállítása
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0

ml = bemenet hossza (64bit)

//Előfeldolgozás:
„1” bit hozzáfűzése a bemenethez
„0” bitek hozzáfűzése a bementhez, míg üzenet hossza≡448 (mod 512)
ml hozzáfűzés az üzenethez (64bit)

// A bemenet 512bitenként való feldolgozása:
32bites méretű szavakra bontás: w[i], 0≤i≤15

//16db 32bites rész 80db 32bites résszé kiterjesztése:
      for i from 16 to 79
            w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) leftrotate 1

      a = h0
      b = h1
      c = h2
      d = h3
      e = h4

    //Fő ciklus (4 kör)
    for i from 0 to 79
        if 0 ≤ i ≤ 19 then
            f = (b and c) or ((not b) and d)
            k = 0x5A827999
        else if 20 ≤ i ≤ 39
            f = b xor c xor d
            k = 0x6ED9EBA1
        else if 40 ≤ i ≤ 59
            f = (b and c) or (b and d) or (c and d)
            k = 0x8F1BBCDC
        else if 60 ≤ i ≤ 79
            f = b xor c xor d
            k = 0xCA62C1D6

        temp = (a leftrotate 5) + f + e + k + w[i]

      e = d
      d = c
      c = b leftrotate 30
      b = a
      a = temp

    //Új értékek hozzáadása az eredeti IV-khez
    h0 = h0 + a
    h1 = h1 + b
    h2 = h2 + c
    h3 = h3 + d
    h4 = h4 + e

//Végső transzformáció:
hh = (h0 leftshift 128) or (h1 leftshift 96) or (h2 leftshift 64) or (h3 leftshift
32) or h4
```

5.1.1 ÜTKÖZÉS AZ SHA-1 ALGORITMUSBAN [8]

Az ütközés levezetésében M. Stevens, E. Bursztein, P- Karpman, A. Albertini és Y. Markov The first collision for full SHA-1 cikkét használtam fel.

SHA-1 folyamata részletesebben az alábbi módon néz ki. Legyen M_j egy blokkja a bemenetnek. Jelölje $s_0 = IV$, valamint $s_{j+1} = H(s_j, M_{j+1})$. Az egyes blokkok 16 db 32 bites szavaiból az alábbi egyenlet alapján duzzasztja 80 darab szóvá:

$$m_i = (m_{i-3} \oplus m_{i-8} \oplus m_{i-14} \oplus m_{i-16})^{\circlearrowright 1}, \text{ ahol } 16 \leq i < 80$$

Legyen $(A_{-4}, A_{-3}, A_{-2}, A_{-1}, A_0) := (e^{\circlearrowleft 2}, d^{\circlearrowleft 2}, c^{\circlearrowleft 2}, b, a)$, ahol az a, b, c, d, e a már látott IV vektor, $\circlearrowleft$ jel jelölje a jobbra forgatást és $\circlearrowright$ jel pedig a balra forgatását. A 80 lépés során az alábbi művelet zajlik le: $i = 0, \dots, 79$:

$$A_{i+1} = A_i^{\circlearrowleft 5} + F_i\left(A_{i-1}, A_{i-2}^{\circlearrowright 2}, A_{i-3}^{\circlearrowright 2}\right) + A_{i-4}^{\circlearrowright 2} + K_i + m_i,$$

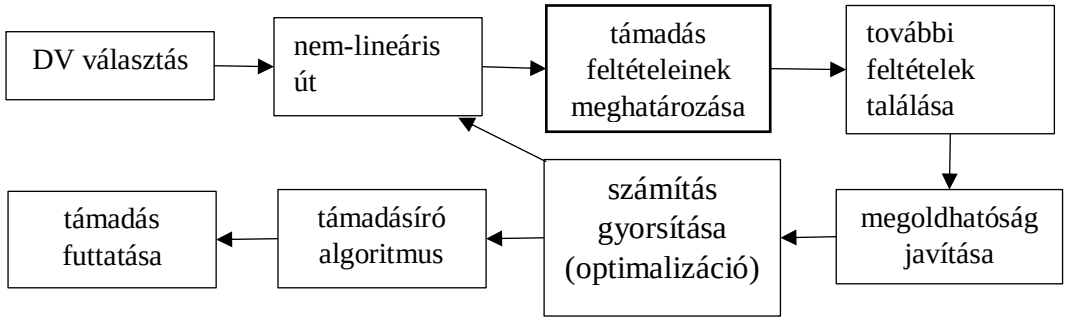
ahol az F_i a korábban ismertetett négy kör függvénye, valamint a $K_i$az előre definiált konstans. A 80 lépés után az alábbi köztes értéket kapjuk:

$$s_{j+1} = (a + A_{80}, b + A_{79}, c + A_{78}^{\circlearrowright 2}, d + A_{77}^{\circlearrowright 2}, e + A_{76}^{\circlearrowright 2}.$$

A vizsgálni kívánt ütközés támadás egy úgy nevezett választott-prefix támadás, ahol P egy olyan prefix, amely kiterjesztett két ütközésközeli blokkpárra, így bármely S szuffix esetén ütköznek. Tehát két-két olyan blokk előállítása a cél, amelyre a második blokk utáni s érték megegyezik. A következő formulára építkezik:

$$SHA-1\left(P \parallel M_1^{(1)} \parallel M_2^{(1)} \parallel S\right) = SHA-1(P \parallel M_1^{(2)} \parallel M_2^{(2)} \parallel S)$$

Az első near-collision (ütközésközeli) blokk pár megtalálásához egy nyílt forrású kódot alkalmaztak, annyi módosítással, hogy saját prefixet alkalmaztak. Az alábbi ábrán látható, hogy miként épül fel ilyen támadás: (Az algoritmust úgy a legkönnyebb elképzelni, ha úgy tekintünk rá, mint egy útvonal keresési problémára.)



12. ÁBRA FŐ LÉPÉSEI A "NEAR-COLLISION" TÁMADÁSNAK [8]

A szükséges lépések ismertetése:

1. Disturbance (zavar) vektor választás:

   Ezen vektor megválasztása az egyik legfontosabb lépés. Tartalmaznia kell az üzenet XOR különbségeit, optimális lineáris üzenetbit egyenleteket, amelyek maximalizálják a lineáris rész sikerességének valószínűségét, optimális támadási lehetőségeket lineáris útvonal felett, stb.

2. A nem lineáris út felépítése:

   Miután a disturbance vektor és a különbség út megfelelő része rögzült, akkor a következő lépés a megfelelő nem lineáris útvonal megtalálása, amely összeköti a köztes értéket a megfelelő lineáris résszel. Ez a lépést külön-külön el kell végezni, minden egyes „near-collision" támadás esetén.

3. Támadás feltételeinek meghatározása:

   Ebben a lépésben a támadás teljes egyenletrendszerét határozzuk meg. Ez a rendszer két féle egyenletrendszerből áll és teljes egészében az $M^{(1)}$ üzenet felett értjük és nem $M^{(2)}$ felett. Az első egyenletrendszer lineáris egyenletekből áll az üzenet bitjei felett. Ezek főként az előjeleket kontrolálják a „disturbance" vektor által. A második egyenletrendszer pedig a köztes értékek bitjei feletti egyenletek sokasága. Ezek vezérlik hol van különbség a köztes értékek bitjei között.

4. További feltétel találása:

   Ennek célja, hogy mielőbb észrevegye az algoritmus, ha rossz úton haladunk és elkezdjen visszalépni.

5. Megoldhatóság javítása:

   Ez a lépés nem mindig szükséges. A második „near-collision" megtalálása során sokszor megoldhatatlan számításokba ütköztek, ennek elkerülésére az algoritmusba egy plusz MiniSAT rendszert implementáltak, amely plusz számítások segítségével elő tudott állítani egy megoldás halamazt, amelyekből azokat tartották meg, amelyek a legkevesebb feltétellel rendelkeztek az különbségi út megtalálásához.

6. Üzenet módosítása az algoritmus gyorsítása érdekében.

7. Az algoritmus implementálása.

8. Program futtatása.

Ezen paraméterek segítségével sikerült 2017-ben az első programot megírni, amely képes előállítani két üzenetpárt, amelyek SHA-1 lenyomatuk megegyezik.

Ezáltal sebezhetővé vált minden olyan alkalmazás, amely SHA-1 algoritmus használ digitális aláírásra, fájlhitelesítésre. Például az alábbi területek érintettek: [9]

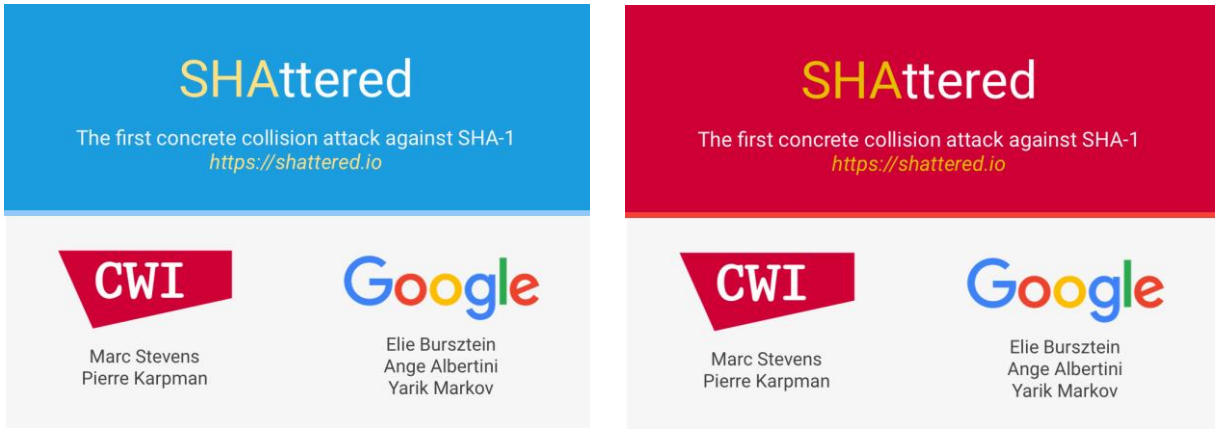
- Szoftverfrissítések
- Szoftverszállító aláírások
- digitális tanúsítvány aláírások
- duplikációs rendszerek
- GIT
- …

A Gmail által küldött fájlok, valamint a Google Drive-on tároltak már automatikusan vizsgálva vannak ezen támadás ellen.

Példa ütközésre:

Az alábbi két képen két pdf fájl tartalma látható, amelyek a SHAttered által készített dokumentumok, és az SHA-1 nem ütközésellenállóságát demonstrálják.

Mindkét fájl SHA-1 lenyomata a következő:
`38762cf7f55934b34d179ae6a4c80cadccbb7f0a`



13. ÁBRA KÉT PDF FILE TARTALMA, AMELYEK ÜTKÖZNEK [9]

```
C:\Users\User\Documents\Matek\SZAKDOGA>certutil -hashfile shattered-1.pdf sha1
SHA1 hash of shattered-1.pdf:
38762cf7f55934b34d179ae6a4c80cadccbb7f0a
CertUtil: -hashfile command completed successfully.

C:\Users\User\Documents\Matek\SZAKDOGA>certutil -hashfile shattered-2.pdf sha1
SHA1 hash of shattered-2.pdf:
38762cf7f55934b34d179ae6a4c80cadccbb7f0a
CertUtil: -hashfile command completed successfully.
```

14. ÁBRA ÜTKÖZÉS ELLENŐRZÉSE PARANCSSORBAN

5.2 SHA-2

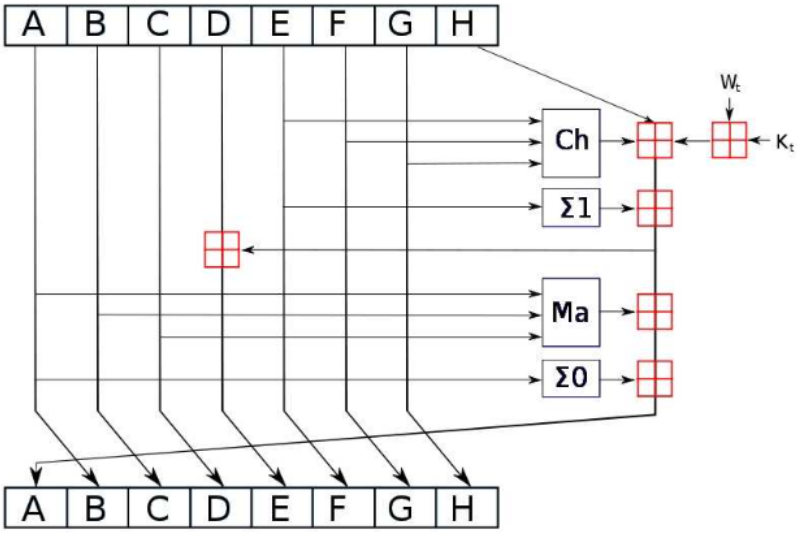
2001-ben publikálta a National Security Agency (NSA) ezen függvény család első tagját, az SHA-256. 2005 óta ennek használatát javasolják, mivel az SHA-1-ben számos matematikai hibát felfedeztek. Struktúráját tekintve szignifikánsan eltér az SHA-1 függvénytől. AZ SHA-2 már egy függvénycsaládnak tekinthető, hiszen több hash függvényt is magába foglal, amelyek felépítésüket tekintve szinte megegyeznek, a fő eltérést a blokk méret, illetve a szó méretben nyilvánul meg.

Az alábbi táblázatban láthatjuk a különbségeket a különböző SHA algoritmusok között. A méret és a hossz bit-ben értendő.

Hash függvény	Lenyomat hossz	Blokk méret	Szó méret	Lépések száma
MD5	128	512	32	64
SHA-1	160	512	32	80
SHA-256	256	512	32	64
SHA-512	512	1024	64	80

15. ÁBRA HASH ALGORITMUSOK ÖSSZEHASONLÍTÁSA [10]

Az SHA-2 algoritmus működése:



16. ÁBRA SHA-2 ALGORITMUS [1]

Itt az eddigi 4 kör helyett egy nagy kör van, amelyen belül 4 előre definiált függvény van, melyek az alábbiak [11]:

$$Ch(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G),$$

$$Ma(A,B,C) = (A \wedge C) \oplus (A \wedge C) \oplus (B \wedge C),$$

$$\sum_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22),$$

$$\sum_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25),$$

A függvényekben használt konstansok csak az SHA-256-ra igaz, más függvényben ezek az állandó értékek eltérhetnek. Emellett az ábrán látható összeadás $mod(2^{32})$ értendő, ez az SHA-512 esetén $mod(2^{64})$. $\ggg$ jel ez esetben eltolást jelöl.

SHA-256 pszeudó kódja [11]:

```
//Változók kezdőértékének beállítása
h0 := 0x6a09e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd19

//Konstansok értékeinek beállítása:
k[0..63] :=
  0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,0x3956c25b,0x59f111f1,0x923f82a4,
0xab1c5ed5,
  0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,0x72be5d74,0x80deb1fe,0x9bdc06a7,
0xc19bf174,
  0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240ca1cc,0x2de92c6f,0x4a7484aa,0x5cb0a9dc,
0x76f988da,
  0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,0xc6e00bf3,0xd5a79147,0x06ca6351,
0x14292967,
  0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,0x650a7354,0x766a0abb,0x81c2c92e,
0x92722c85,
  0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,0xd192e819,0xd6990624,0xf40e3585,
0x106aa070,
  0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,0x391c0cb3,0x4ed8aa4a,0x5b9cca4f,
0x682e6ff3,
  0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,0x90befffa,0xa4506ceb,0xbef9a3f7,
0xc67178f2

//Előfeldolgozás (bitkitöltés):
„1” bit hozzáfűzése a bemenethez
„0” bitek hozzáfűzése a bementhez, míg üzenet hossza≡448 (mod 512)
ml hozzáfűzés az üzenethez (64bit)

//Minden 512bites blokkra a következőlet hajtsa végre a program:
w[i]: 512bites blokkok 32 bites szava, i≤0≤15
//A 32bites szavak kiterjesztése 64 bitesre
    for i from 16 to 63
       s0 := (w[i-15] rightrotate  7) xor (w[i-15] rightrotate 18) xor (w[i-15]
             rightshift  3)
       s1 := (w[i- 2] rightrotate 17) xor (w[i- 2] rightrotate 19) xor (w[i- 2]
             rightshift 10)
       w[i] := w[i-16] + s0 + w[i-7] + s1
    a := h0
    b := h1
    c := h2
    d := h3
    e := h4
    f := h5
    g := h6
    h := h7

    //Fő ciklus:
    for i from 0 to 63
        S1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)
        ch := (e and f) xor ((not e) and g)
        temp1 := h + S1 + ch + k[i] + w[i]
        S0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
```

```
        maj := (a and b) xor (a and c) xor (b and c)
        temp2 := S0 + maj
    h := g
    g := f
    f := e
    e := d + temp1
    d := c
    c := b
    b := a
    a := temp1 + temp2

    //Új értékek hozzáadása az eredeti IV-khez:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
    h4 := h4 + e
    h5 := h5 + f
    h6 := h6 + g
    h7 := h7 + h

//Végső transzformáció:
digest := hash := h0 append h1 append h2 append h3 append h4 append h5 append h6
append h7
```

6. MAC

A MAC (Message Authentication Code) a kulcsos hash függvények közé tartozik. Az MD függvényekhez hasonlóan ez is azt a cél szolgálja, hogy biztosítsa az üzenet hitelességét, de ez magába foglalja az üzenet sértetlenségét, illetve a feladót is tudjuk ellenőrizni. Az üzenet hitelességének ellenőrzése az MD-hez hasonlóan működik, annyi különbséggel, hogy a MAC kód generálásához kulcsot is használ, amelyet majd Alice elküld Bobnak a szokott módon. A kulcs használatára és annak megosztására számos protokoll utat mutat.

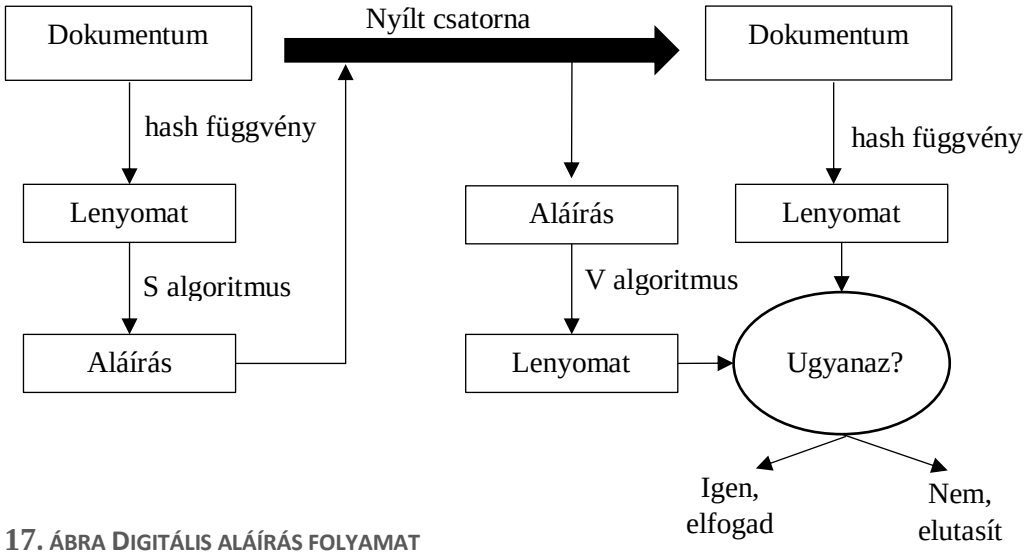
A kulcsos hash függvények egyik fő felhasználási területe a digitális aláírás, ami szintén arra szolgál, hogy igazolja az üzenet hitelességét és sértetlenségét. Az első széles körben használt software, amely digitális aláírás csomaggal is rendelkezett 1989-ben indult Lotus notes 1.0 néven és RSA algoritmust használta. Manapság egyre fontosabb szerepet töltenek be, hiszen egyre több ügyet intézünk online: elektronikus adóbevallás, számlázás, online szerencsejátékok, stb.

6.1 DIGITÁLIS ALÁÍRÁS

A séma megadásához három különböző algoritmust használunk: $D = (G, S, V)$, ahol az egyes elemek a következők:

- G: kulcsgeneráló algoritmus- a szükséges kulcspárt, (sk, pk) állít elő, ahol sk (secret key) a titkos kulcs és pk a nyilvános kulcs. A kulcsok mérete k biztonsági paramétertől függ
- S: aláíró algoritmus- az m üzenetből és a sk titkos kulcsból előállít egy s (signature) aláírást
- V: ellenőrző algoritmus- a pk nyilvános kulcs segítségével ellenőrzi, hogy s aláírás valóban m üzenethez tartozik-e. Ez az algoritmus két változóval tér vissza: igaz vagy hamis értékkel.

A titkos kulcsot csak az aláíró fél ismeri, míg a nyilvános kulcs bárki számára elérhető. Biztonságát az adja, hogy míg az aláíró és ellenőrző algoritmus polinom időben lefut, addig nincs olyan algoritmus, amellyel polinom időbe a nyilvános kulcsból titkos kulcsot elő tudnák állítani.



17. ÁBRA DIGITÁLIS ALÁÍRÁS FOLYAMAT

6.2 HMAC

A MAC egy speciális fajtája a HMAC (Hash-Based Message Authentication Code), ahogy nevéből is látszik egy hash függvényt alkalmaz a hitelesítő kód kiszámításához. Ha a HMAC például az MD5 hash függvényt használja, akkor azt a következőképp jelöljük: HMAC-MD5. HMAC erőssége függ attól, hogy mely hash függvényt használja, valamint a kulcs méretétől és minőségétől.

HMAC kevésbé sérülékeny, mint az általa felhasznált hash algoritmusok. Például míg az SHA-2 ellen az üzenet kiterjesztő támadás sikeres, addig a HMAC-SHA-2 ellenáll ennek.

Pszeudó-kódja [12]:

```
function hmac is
      //Bemeneti értékek megadása: kulcs, üzenet, a kiválasztott hashfüggvény
        (pl: SHA-1) és a hash függvény blokk mérete
      input:
            key:        Bytes
            message:    Bytes
            hash:       Function
            blockSize:  Integer

      //A kimenetmérete a hash függvény kimenetének mérete
      outputSize: Integer

      //Ha a kulcs hosszabb, mint blokkméret, akkor a hashfüggvénnyel rövidítjük
      ezeket
      if (length(key) > blockSize) then
          key ← hash(key) //kulcs méret megegyezik a kimenet méretével

      //Ha kulcs mérete rövidebb, mint a blokkméret, akkor a kulcsot növeljük,
      még pedig úgy, hogy nullákat adunk a jobb oldalhoz addig, amíg blokkméretet
      el nem éri
      if (length(key) < blockSize) then
          key ← Pad(key, blockSize)

      o_key_pad ← key xor [0x5c  blockSize]   //Külső bővített kulcs
      i_key_pad ← key xor [0x36  blockSize]   //Belső bővített kulcs

      return hash(o_key_pad ‖ hash(i_key_pad ‖ message))
```

7. ÖSSZEGZÉS

Az alábbi összefoglaló táblázatban a szakdolgozat során vizsgált hash függvények tulajdonságait és a különböző támadások elleni ellenállásukat vagy épp gyengeségüket mutatja meg. A táblázatban láthatjuk azt is, hogy a fentiekben vizsgált függvények ellen az üzenetkiterjesztő támadás hatásos vagy sem, ámbár szakdolgozatomban csak az ütközés elleni támadásokat vezettem le.

A táblázat első oszlopában a hash függvények nevei szerepelnek. Az SHA-2-nek és SHA-3-nak több fajtája is ismert, ezért próbáltam a leggyakrabban használt, vagy a szakdolgozatban már korábban említett függvény adatait venni. A következő három oszlopban az látható, hogy mekkora blokk mérettel dolgozik a függvény, hány kört használ fel a számítás során, valamint a lenyomat mérete. Majd pedig az, hogy a két támadás sikeres-e vagy sem az adott függvény ellen.

	Blokk méret (bit)	**Körök száma**	**Kimeneti méret (bit)**	**Ütközés elleni támadás**	**Üzenet kiterjesztő támadás**
MD5	512	64	128	sikeres	sikeres
SHA-1	512	80	160	sikeres	sikeres
SHA-2	512	64	256	sikertelen	sikeres
SHA-3	1152	24	224	sikertelen	sikertelen

18. ÁBRA ÖSSZEFOGLALÓ TÁBLÁZAT

Látható, hogy az egyes hash függvények miként válnak a biztonságosnak hitt kategóriából sérülékennyé alig pár év alatt, ami az informatika robbanásszerű fejlődésének köszönhető. Napjainkban egyre több tudós úgy látja, hogy 15-20 éven belül megjelenhetnek a kvantumszámítógépek. Éppen ezért kell már most azzal foglalkozni, hogy akkor majd miként fogjuk megvédeni adatainkat, hiszen ezek használatával a legtöbb jelenleg használatos rejtjelezők törhetővé válnak. Az ezzel foglalkozó tudományágat poszt-kvantum kriptográfiának hívjuk.

Szakdolgozatom célja az volt, hogy ismertessem ezt a sokszínű hash függvény családot, valamint bemutassam fontosságukat. Felkeltette figyelmem, hogy behálózzák mindennapjaink, ennek ellenére sokan még csak nem is hallottak róla. Érdekesnek találom azt is, hogy milyen gyorsan kell egy új függvényre váltani, ha biztonságban szeretnénk tartani adataink. Sok új kérdést vet fel, hogy a kvantumszámítógépek mennyire borítják majd fel mindennapjaink, vagy lesz-e olyan hash függvény, amelyet soha nem kell lecserélnünk.

HIVATKOZÁSOK

[1] Kathi Ferenc, ,,Hash függvények", Debrecen, 2009.

[2] Vajda István, Buttyán Levente, ,,Kriptográfia és alkalmazásai", Typotex, 2004.

[3] Avalanche effect
https://en.wikipedia.org/wiki/Avalanche_effect. [Hozzáférés dátuma: 20 05 2021].

[4] MD5
https://hu.wikipedia.org/wiki/MD5.

[5] Xiaoyun Wang, Hongbo Yu, ,,How to Break MD5 and Other Hash Functions",
https://web.archive.org/web/20090521024709/http://merlot.usc.edu/csac-f06/papers/Wang05a.pdf. [Hozzáférés dátuma: 27 05 2021].

[6] Bucsay Balázs, „Párhuzamosítható változó hosszú hash függvény”
https://docplayer.hu/3811462-Parhuzamosithato-valtozo-hosszu-hash-fuggveny.html.
[Hozzáférés dátuma: 25 05 2021].

[7] SHA-1
https://en.wikipedia.org/wiki/SHA-1.

[8] M.Stevens, E.Bursztein, P. Karpman, A. Albertini, Y. Markov, „The first collision for full SHA-1” https://shattered.io/static/shattered.pdf. [Hozzáférés dátuma: 27 05 2021].

[9] „Shattered,”
shattered.io. [Hozzáférés dátuma: 26 05 2021].

[10] Secure Hash Algorithms
https://en.wikipedia.org/wiki/Secure_Hash_Algorithms. [Hozzáférés dátuma: 25 05 2021].

[11] SHA-2
https://en.wikipedia.org/wiki/SHA-2. [Hozzáférés dátuma: 27 05 2021].

[12] „Wikipédia,” [Online]. Available: Wikipédia. [Hozzáférés dátuma: 2021].

[13] Kiss Emil, „Algebra3”
https://ewkiss.web.elte.hu/wp/wordpress/wp-content/uploads/2018/10/Alg3a_print_10.pdf.

[14] T. C. Thaeh, „Probabilisticworld,” 17 08 2020. [Online]. Available:
https://www.probabilisticworld.com/birthday-problem-python-simulation/.

[15] P. Selinger, „mscs.dal.ca,”
https://www.mscs.dal.ca/~selinger/md5collision/. [Hozzáférés dátuma: 26 05 2021].

[16] M. Bodewes, „Stack Ecxhange,” https://crypto.stackexchange.com/questions/1434/are-there-two-known-strings-which-have-the-same-md5-hash-value. [Hozzáférés dátuma: 25 05 2021].

[17] P. v. O. a. S. V. A. Menezes, „Handbook of Applied Cryptography,” 1996. [Online]. Available: http://cacr.uwaterloo.ca/hac/about/chap9.pdf.

[18] X. Wang, D. Feng, X. Lai, H. Yu, „Collisions for Hash Funtions” https://eprint.iacr.org/2004/199.pdf. [Hozzáférés dátuma: 25 05 2021].

ÁBRAJEGYZÉK

1. ábra Hash függvények csoportosítása [17] 4
2. ábra hamming távolság bemutatása 7
3. ábra születésnapi paradoxon [14] 9
4. ábra példa lavina effektusra 10
5. ábra Hamming távolság példája 11
6. ábra Hash függvény általános modellje 12
7. ábra Iterált kriptográfiai hash függvény [2] 13
8. ábra Üzenet hitelesítés 17
9. ábra MD5 algoritmus [1] 18
10. ábra Ütközés ellenőrzése parancssorban [15] 21
11. ábra SHA-1 algoritmus [1] 23
12. ábra Fő lépései a "near-collision" támadásnak [8] 25
13. ábra Két pdf file tartalma, amelyek ütköznek [9] 27
14. ábra ütközés ellenőrzése parancssorban 27
15. ábra Hash algoritmusok összehasonlítása [10] 28
16. ábra SHA-2 algoritmus [1] 28
17. ábra Digitális aláírás folyamat 31
18. ábra Összefoglaló táblázat 33