Kriptográfia és Információbiztonság
6. előadás

MÁRTON Gyöngyvér

Sapientia Egyetem, Matematika-Informatika Tanszék
Marosvásárhely, Románia
mgyongyi@ms.sapientia.ro

2023

Miről volt szó az elmúlt előadáson?

- az AES (Advanced Encryption Standard): specifikáció, biztonság, tervezési szempontok
- hash függvények (hash function)
- publikus kulcsú titkosítók: RSA, RSA-OAEP
- digitális aláírások: RSA, RSA-RSS

Miről lesz szó?

- az RSA gyorsítása
- RSA: biztonsági problémák
- RSA: az e megválasztása
- RSA: Wiener feltörési algoritmusa
- Az SHA-1 (Secure Hash Algorithm) szerkesztése
- Üzenetintegritás
- Üzenet-hitelesítő kódok (Message Authentication Code): HMAC, CMAC
- Hitelesített titkosítás: GCM

Az RSA gyorsítása

- Alkalmazható a kínai maradéktétel a visszafejtés időigényének javítása érdekében.
- Alkalmazásával a rendszer nem veszít biztonságából.
- Ahelyett hogy az n nagyságrendjével megegyező d hatványkitevővel számolnánk, elvégzünk két kisebb (a p nagyságrendjével megegyező) hatványkitevővel való hatványozást.
- Meghatározzuk dp, dq, Mq, Mp értékeket a következő módon:

$$\begin{aligned} dp &= d \pmod{p-1} \\ dq &= d \pmod{q-1} \\ q \cdot Mq &= 1 \pmod{p} \\ p \cdot Mp &= 1 \pmod{q}. \end{aligned}$$

- A $c^d \pmod{n}$ értéket megadja az x értéke, ahol

$$\begin{aligned} x &= (Mq \cdot q \cdot xp + Mp \cdot p \cdot xq) \pmod{n} \\ xp &= c^{dp} \pmod{p} \\ xq &= c^{dq} \pmod{q}. \end{aligned}$$

RSA, biztonsági problémák

- az n faktorizálásához kapcsolódó problémák
  - a k nagyságrendje: min 1024 bit
  - Fermat féle faktorizáció: a p és a q ne legyenek túl közel egymáshoz
  - Pollard ρ féle faktorizáció: a p, vagy a q kicsi,
  - Pollard $p-1$ féle faktorizáció: $p-1$, illetve $p+1$-nek legyen legalább egy "nagy" prímosztója,
  - legjobb módszer: ha véletlenszerűen választjuk meg a prímeket
  - ...
- ugyanannak a p, vagy q prímnek a többszöri felhasználása
- a modulus többszöri felhasználása, különböző e értékekre
- az e, d megválasztása: e kicsi kell legyen, d azonban az n nagyságrendjével kell egyenlő legyen
- az RSA-textbook (baby) nem biztonságos, mert
  - a titkosító algoritmus nem véletlenszerűsített $\rightarrow$ választott nyílt-szöveg típusú támadás.
  - az RSA titkosító multiplikatív tulajdonságú $\rightarrow$ választott rejtjelezett-szöveg típusú támadás.

RSA, biztonsági problémák

Implementációhoz kapcsolódó problémák:

- timing attack, Koecher és tsai 1997: le lehet mérni, hogy mennyi időbe telik $c^d \pmod{n}$ meghatározása
- power attack, Koecher és tsai 1999: le lehet mérni, hogy mekkora energiafogyasztás szükséges $c^d \pmod{n}$ meghatározásához
- az RSA kulcsok generálása során adódó figyelmetlenségi hibák
- faults attack, Boneh és tsai 1997: a $c^d \pmod{n}$ meghatározásakor fellépő számítási hibák vizsgálata
  - a kínai maradéktétel alkalmazásakor a visszafejtéshez meg kell határozni a $xp = c^{dp} \pmod{p}$, illetve $xq = c^{dq} \pmod{q}$ értékeket
  - ha a számítások során hiba adódik, például az $c^{dq} \pmod{q}$ meghatározásánál hiba adódik, míg az $c^{dp} \pmod{p}$-nél nem, akkor lehetségessé válik az egyik prim meghatározása.

RSA, biztonsági problémák

faults attack, Boneh és tsai 1997:

- ha a $xq = c^{dq} \pmod{q}$ meghatározásánál hiba adódik, míg az $xp = c^{dp} \pmod{p}$-nél nem, akkor legnagyobb közös osztót számolva lehetségessé válik az egyik prim meghatározása: $(x^e - c, N) = p$, ahol

$$x \equiv (Mq \cdot q \cdot xp + Mp \cdot p \cdot xq) \pmod{n}$$
$$x^e \equiv c \pmod{p}, x^e \not\equiv c \pmod{q}.$$

Legyen $p = 13, q = 17, n = p \cdot q = 221, phi = (p-1) \cdot (q-1) = 192, e = 5, d = 77$, fennáll, hogy $e \cdot d = 1 \pmod{phi}$ és legyen $m = 207$, $c = 207^5 \equiv 90 \pmod{221}$:

$dp \equiv d \pmod{p-1} \equiv 5$
$Mq = 10 : 17 \cdot 10 \equiv 1 \pmod{13}$
$xp \equiv c^{dp} \pmod{p} = 90^5 \equiv 12 \pmod{13}$

$dq \equiv d \pmod{q-1} \equiv 13$
$Mp = 4 : 13 \cdot 4 \equiv 1 \pmod{17}$
$xq \equiv c^{dq} \pmod{q} = 90^{13} \equiv 3 \pmod{17}$

Tegyük fel, hogy hiba adódott xq meghatározásakor, legyen a hibás érték 7, ezért a visszafejtéskor a következő x értéket kapjuk, majd legnagyobb közös osztót számolva meghatározzuk a p-t:

$$x \equiv 10 \cdot 17 \cdot 12 + 4 \cdot 13 \cdot 7 \equiv 129$$
$$129^5 \pmod{221} - 90 = 52$$
$$(52, 221) = 13$$

RSA, biztonsági problémák

Az $e = 3$-as exponens problémája:

- a titkosítás idejének csökkentése úgy oldható meg, ha kis e exponenst választok, de mennyire kis e értéket választhatok: 3, 7, 65537?
- az $e = 3$ választás esetén, ha ugyanazt a nyílt-szöveget különböző modulus értékekkel: n_1, n_2, n_3, elküldjük három különböző félnek, akkor a rendszer feltörhető.

Feltételezve, hogy a nr értékét titkosítottuk, és a c_1, c_2, c_3 títkosított értékeket kaptuk, akkor a kínai maradéktételt alkalmazva, a jobboldali x-ben ismeretlen lineáris kongruencia rendszert kell megoldani, ahol az eredményből köbgyököt vonva, megtudjuk határozni a nr értékét:

$$\begin{aligned} c_1 &= nr^3 \pmod{n_1} \\ c_2 &= nr^3 \pmod{n_2} \\ c_3 &= nr^3 \pmod{n_3} \end{aligned} \qquad \begin{aligned} x &= c_1 \pmod{n_1} \\ x &= c_2 \pmod{n_2} \\ x &= c_3 \pmod{n_3} \end{aligned}$$

RSA, biztonsági problémák

Az $e = 3$-as exponens problémája (folytatás):

Az x értékét a következő képlet adja (kínai maradéktétel alapján):

$$x = (M1^{-1} \cdot M1 \cdot c_1 + M2^{-1} \cdot M2 \cdot c_2 + M3^{-1} \cdot M3 \cdot c_3)(mod M)$$

$$M = n_1 \cdot n_2 \cdot n_3$$

$$M1 = M/n_1, M2 = M/n_2, M3 = M/n_3$$

$$\begin{aligned} M1 \cdot M1^{-1} &= 1 \pmod{n_1} \\ M2 \cdot M2^{-1} &= 1 \pmod{n_2} \\ M3 \cdot M3^{-1} &= 1 \pmod{n_3} \end{aligned}$$

A köbgyököt meghatározó iteráció:

$$\begin{aligned} x_0 &= 1 \\ x_{n+1} &= (2 \cdot x_n + nr/(x_n \cdot x_n))/3 \end{aligned}$$

RSA, biztonsági problémák

Wiener feltörési algoritmusa

- az RSA visszafejtési algoritmus időigényének a gyorsítása érdekében ajánlatos lenne, hogy a d nagyságrendje kicsi legyen
- 1990-ben azonban Wiener megmutatta, hogy ha $q < p < 2 \cdot q$ és $3 \cdot d < n^{1/4}$ akkor habár 75%-al csökkenthető a visszafejtés időigénye lehetségesé válik a modulus faktorizálása!
- lánctörtek alkalmazásával meghatározzuk az $\frac{e}{n}$ lánctört egy közelítő értékét, amely segítségével aztán meghatározható $\phi(n)$, és innen az n két szorzótényezője a p és a q is
- az $\frac{e}{n}$ lánctörtként való felírásában a $q_1, q_2, \ldots, q_k$ egész számok szerepelnek, akkor a $\frac{t}{d}$ közelítő tört a következő iteráció során határozható meg, ahol $j \in \{2, 3 \ldots\}$:

$$t_0 = 0, t_1 = 1, d_0 = 1, d_1 = 0,$$
$$t_j = q_j \cdot t_{j-1} + t_{j-2},$$
$$d_j = q_j \cdot d_{j-1} + d_{j-2}$$

RSA, biztonsági problémák

Wiener feltörési algoritmusa, folytatás

- minden egyes iterációnál kiszámoljuk:

$$\phi = \frac{d_j \cdot e - 1}{t_j}$$

- ha fennáll $d_j \cdot e - 1 \equiv 0 \pmod{t_j}$, akkor feltételezhetjük, hogy $\frac{t_j}{d_j}$ megfelelő közelítő érték lesz
- ha a következő másodfokú egyenlet megoldásánál is sikerrel járunk, azaz két egész gyököt kapunk, akkor megállapíthatjuk, hogy jó volt feltételezésünk, mert a két egész gyök a keresett p és q értékek lesznek:

$$x^2 + (\phi - n - 1) \cdot x + n = 0.$$

- a fenti egyenletet úgy kaphatjuk meg, hogy a $y = n/x$-t és $n = x \cdot y$-t behelyettesítjük a következő egyenletbe:

$$\phi = (x - 1) \cdot (y - 1).$$

Hash függvények, tulajdonságok

Egy $h : X \to Y$, hash függvény tetszőleges hosszúságú bitsorozatból fix hosszúságú bitsorozatot állít elő.

Az alábbi problémák ne legyenek megoldhatóak polinomiális időben:

- 1. probléma, egyirányú (preimage) tulajdonság:
  Bemenet: h, és $y \in Y$
  Kimenet: határozzuk meg x-t,úgy hogy: $h(x) = y$
- 2. probléma, gyengén ütközésmentes (second preimage) tulajdonság:
  Bemenet: h, és $x \in X$
  Kimenet: határozzuk meg x_1-t,úgy hogy: $x_1 \neq x$ és $h(x) = h(x_1)$
- 3. probléma, ütközésmentes (collision) tulajdonság:
  Bemenet: h
  Kimenet: határozzuk meg x, x_1-t,úgy hogy: $x_1 \neq x$ és $h(x) = h(x_1)$

további tulajdonságok:

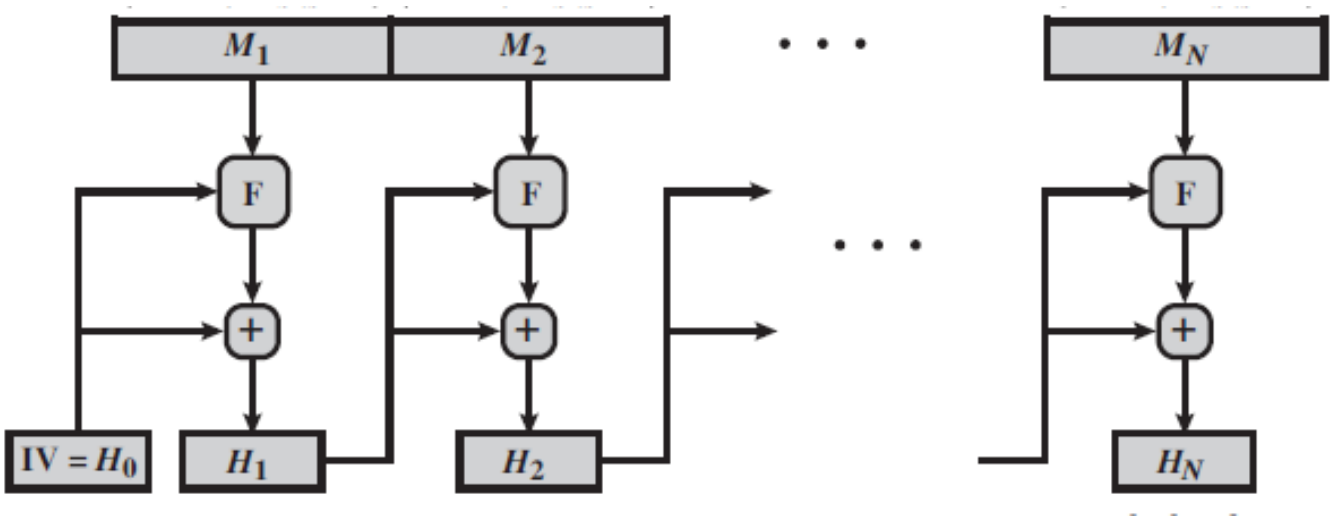
- hatékonyan lehessen kiszámítani a hash értékét
- a hash kimenete legalább annyi bit legyen, hogy ne lehessen brute-force támadást alkalmazni rá
- lavina effektus: a bemenet egyetlen bitjének a megváltoztatása vonja maga után a kimenet teljes megváltozását

Az SHA-1 (Secure Hash Algorithm) szerkesztése

- kimenete 160 bit
- működése a Merkle-Damgard szerkesztési elvet követi
- az alkalmazott F tömörítő függvény bemenete 512 bit, ahol az IV mérete 160 bit, $IV = H_0 = h_0||h_1||h_2||h_3||h_4||h_5$
- megmutatható, hogy ha az F tömörítő függvény ütközésmentes, akkor a hash függvény is az
- az alkalmazott operációk:
  - bitenkénti $\wedge$, $\vee$, $\oplus$, $\neg$,
  - egészek összeadása mod 32 szerint : $+$,
  - s pozícióval való balra forgatás, ahol $0 \leq s \leq 31$: Rot_s
- a hash függvény x bemenetét kiegészítjük annyi bittel, hogy osztható legyen 512-vel, kapjuk y-t:
  - feltételezzük, hogy $|x| \leq 2^{64} - 1$,
  - $d = 447 - |x| \pmod{512}$,
  - legyen k az $|x|$ bináris ábrázolása 64 biten,
  - $y = x \;||\; 1 \;||\; 0^d \;||\; k$,

Az SHA-1 (Secure Hash Algorithm) szerkesztése

- y-t felosztjuk 512 bites blokkokra, azaz $y = M_1||M_2||\ldots||M_n$, ahol minden M_i blokkra alkalmazzuk az F függvényt



Kép forrása: W. Stallings. Cryptography and Network Security. Principles and Practice. Prentice Hall. 2010

Az SHA-1 (Secure Hash Algorithm) szerkesztése

- az F függvény minden M_i blokkot feloszt 16 darab 32 bites részre, azaz
$$M_i = W_0||W_1||\ldots||W_{15}$$
- az F függvény működése az $M_i = W_0||W_1||\ldots||W_{15}$ blokkon a következő:
  - minden $t = 16, \ldots, 79$-re meghatározza:
$$W_t = Rot_1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}),$$
  - inicializálja az A, B, C, D konstansokat:
$$A \leftarrow h_0,\ B \leftarrow h_1,\ C \leftarrow h_2,\ D \leftarrow h_3,\ E \leftarrow h_4,$$
  - minden $t = 0, \ldots, 79$-re elvégzi a következőket:
    - $temp = Rot_5(A) + f_t(B, C, D) + E + W_t + K_t$
    - $E \leftarrow D,\ D \leftarrow C,\ C \leftarrow Rot_{30}(B),\ B \leftarrow A,\ A \leftarrow temp,$
    - $h_0 \leftarrow h_0 + A,\ h_1 \leftarrow h_1 + B,\ h_2 \leftarrow h_2 + C,\ h_3 \leftarrow h_3 + D,$ $h_4 \leftarrow h_4 + E,$
  - a függvény kimenete $H_i = (h_0||h_1||h_2||h_3||h_4)$,

Az SHA-1 (Secure Hash Algorithm) szerkesztése

- az F függvény alkalmaz egy f_t és egy Rot_i függvényt, ahol
- a Rot_i függvény egy ciklikus balra forgatást jelent i pozícióval,
- az f_t függvény bemenete 3×32 bit (B, C, D-vel jelöljük őket), kimenete 32 bit és a következőket végzi:

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & \text{ha } 0 \leq t \leq 19 \\ (B \oplus C \oplus D) & \text{ha } 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{ha } 40 \leq t \leq 59 \\ (B \oplus C \oplus D) & \text{ha } 60 \leq t \leq 79. \end{cases}$$

- az alkalmazott konstansok:

$$\begin{aligned} h_0 &= 67452301 \\ h_1 &= EFCDAB89 \\ h_2 &= 98BADCFE \\ h_3 &= 10325476 \\ h_4 &= C3D2E1F0 \end{aligned}$$

$$K_t = \begin{cases} 5A827999 & \text{ha } 0 \leq t \leq 19 \\ 6ED9EBA1 & \text{ha } 20 \leq t \leq 39 \\ 8F1BBCDC & \text{ha } 40 \leq t \leq 59 \\ CA62C1D6 & \text{ha } 60 \leq t \leq 79. \end{cases}$$

SHA-2, SHA-3

SHA-2

- ide tartozik az SHA-224, SHA-256, SHA-384, SHA-512
- hasonló a szerkezetük az SHA-1-hez, az IV mérete 256 bit
- nehezebb ütközést találni
- hasonlóan az SHA-1-hez a Merkle-Damgård szerkesztési elvet követik
- a tömörítő függvény más, bemenete 512 bit
- minden körben alkalmazásra kerül egy prímszám köbgyökének a törtrészének az első 64 bitje (az első körben az első prímszám, majd a második, stb.)

SHA-3

- a NIST 2007-ben versenyt hirdetett a következő generációs NIST hash függvények kidolgozására, mert az SHA-2 ugyanazt az szerkesztési elvet, ugyanazokat a matematikai műveletek végzi, mint az SHA-1
- a nyertes Keccak-ot 2015-ben hidették ki

SHA-256, Crypto++

```
#define CRYPTOPP_DEFAULT_NO_DLL
#include "dll.h"
USING_NAMESPACE(CryptoPP)
USING_NAMESPACE(std)
int main() {
        SHA256 hashS;
        string mess = "2020#marcius#maradj#otthon";
        string hashValueS;
        StringSource s(mess, true, new HashFilter(hashS,
        new HexEncoder(new StringSink(hashValueS))));
        cout << hashValueS << endl;
        cout << hashValueS.length() << endl;

        SHA256 hash;
        string hashValue;
        FileSource f("kep.png", true, new HashFilter(hash,
        new HexEncoder(new StringSink(hashValue))));
        cout << hashValue << endl;
        cout << hashValue.length() << endl;
        return 0;
}
```

Üzenetintegritás - hash függvénnyel

Hash függvényeket alkalmazva az üzenetek integritásának a biztosítása a következőket jelenti:

- a küldő meghatározza az üzenet hash értéket
- továbbítja a fogadó félnek mind a hash értéket, mind az üzenetet
- a fogadó ugyanazt a hash számítást végzi el az üzeneten és összehasonlítja ezt az értéket a bejövő hash értékkel
- ha eltérés van, a fogadó fél tudni fogja, hogy az üzenet (vagy esetleg a hash értéke) megváltozott

a hash értéket biztonságos csatornán kell elküldeni: egy támadó félnek ne legyen lehetősége megváltoztatni se az üzenetet, se a hash értéket

Üzenetintegritás - hash függvénnyel

Védekezési módszerek a lehetséges támadások ellen:

- 1. módszer
  - az üzenet és a hozzáfűzött hash értéket titkosítjuk (szimmetrikus titkosítással)
  - mivel csak küldő és a fogadó rendelkezik a titkos kulccsal, az üzenet csak a küldőtől jöhet
  - a hash érték a hitelesítést, a titkosítás a bizalmasságot biztosítja
- 2. módszer
  - ha nem szükséges a bizalmas információcsere, akkor csak a hash értéket titkosítjuk, kevesebb erőforrás lesz így igénybe véve
- 3. módszer
  - feltételezzük, hogy a kommunikáló felek rendelkeznek egy közös titkos értékkel, legyen ez S
  - a küldő kiszámítja az üzenet és az S összefűzött értékének a hash értéket, majd a kapott hash értékhez hozzáfűzi az üzenetet, és ezt küldi el a fogadó félnek
  - a fogadó mivel rendelkezik S-el el tudja végezni az ellenőrzéseket
  - mivel az S titkos érték nem kerül átküldésre a támadó nem tudja módosítani az elfogott üzenetet, és nem is generálhat hamis üzenetet
- 4. módszer
  - a 3. módszert kiegészítik az üzenet és a hash érték titkosításával

Üzenet-hitelesítő kódok (Message Authentication Code)

- üzenetek integritásának a vizsgálatára szélesebb körben elfogadott módszer a MAC-ek alkalmazása
- a fogadó fél le tudja ellenőrizni a küldő fél identitását, illetve az üzeneten végzett szándékos (rosszindulatú) változtatásokat
- a MAC egy titkos információ (titkos kulcs) alapján tetszőleges hosszúságú bitsorozatból fix hosszúságú bitsorozatot állít elő
- nem biztosít titkosítást ezért a MAC függvény inverz függvényének algoritmusát nem kell megadni $\Rightarrow$ kevésbé sérülékeny
- szükséges a MAC-hez használt titkos kulcs előzetes, biztonságos megosztása
- szerkesztése:
  - hash függvény alkalmazásával (pl. HMAC, 1996, használják az SSL protokollban, a NIST elismerte standardnak)
  - blokk-titkosító alkalmazásával (pl. CMAC: 3DES vagy AES alkalmazása)
- alkalmazási terület:
  - pl. a windows rendszer esetében annak az eldöntése, hogy módosultak-e vagy sem valamilyen vírus stb. által az állományok
  - egy számítógép-program hitelességének az ellenőrzése sokkal kényelmesebben megoldható MAC-függvények segítségével, a számítógép-program titkosítása helyett

Üzenet-hitelesítő kódok, alkalmazási terület

Miért van szükség MAC-re, miért nem alkalmasak a titkosítók?

- ugyanazt az üzenetet több különböző helyre küldjük: pl. értesíteni kell a felhasználókat, hogy a hálózat nem elérhető, vagy riasztójelzést kell küldeni egy katonai irányítóközpontba. Elég ha az üzenetet, a MAC-el együtt küldjük, nincs szükség titkosításra
- ha nagy adatforgalomat kell kezeljen egy rendszer akkor az összes üzenet visszafejtése csak fölöslegesen terhelné a rendszert
- a számítógép alkalmazások integritásának az ellenőrzésére is elég a MAC
- van olyan helyzet, amikor jó külön választani az integritásvizsgálatot a titkosítástól: az integritásvizsgálatot az alkalmazások szintjén a titkosítást a szállítási rétegben érdemes implementálni

Mi a különbség a MAC és a digitális aláírás között?

- a MAC használatával a kommunikációt egy harmadik féltől védjük, a digitális aláírással pedig a felek egymástól lesznek védve (amikor nem bíznak meg egymásban)

Üzenet-hitelesítő kódok, matematikai modell

- legyen P az üzenetek és K a kulcsok halmaza
- a kulcs generálás algoritmusa polinom idejű, véletlenszerű: $Gen \rightarrow key$, úgy hogy $key \in K$
- a MAC-érték meghatározásának algoritmusa polinom idejű, véletlenszerű, ahol $m \in P$-re meghatározzuk: $x = MAC_{key}(m)$
- a MAC-érték ellenőrzése determinisztikus algoritmussal történik, ahol az m, key, x értékek alapján először meghatározzuk a $MAC_{key}(m)$ értéket és ellenőrizzük, hogy ez egyenlő-e x-el?

Üzenet-hitelesítő kódok, követelmények

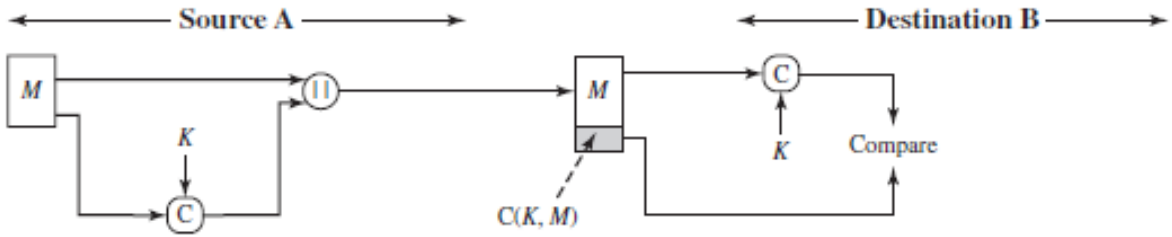
- egy adott MAC-függvény esetében a következő feladatok ne legyenek megoldhatóak polinomiális időben:
  - több m_i és $MAC_{key}(m_i)$ pár ismeretében, határozzuk meg $MAC_{key}(m)$-t, ahol $m \neq m_i$, és ahol a *key* értéke sem ismert
  - több m_i és $MAC_{key}(m_i)$ pár ismeretében határozzuk meg a *key*-t

Megjegyzések:

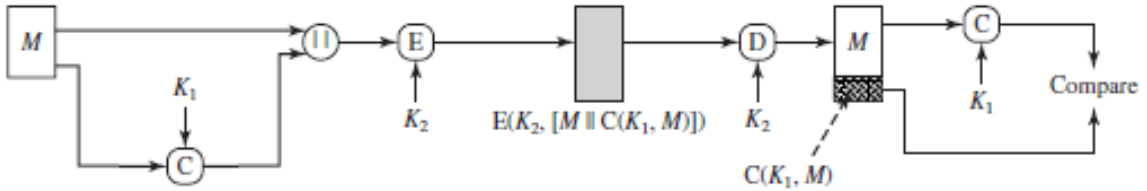
- ha a MAC-függvény eleget tesz az első követelménynek, akkor eleget tesz a másodiknak is. Fordítva azonban nem igaz, mert nem kizárt, hogy egy támadó, egy m üzenethez, a *key* ismerete nélkül is tud szerkeszteni egy érvényes MAC-értéket.
- sokkal több támadási módszer létezik, mint a hash-függvények esetében

Üzenet-hitelesítő kódok, használat

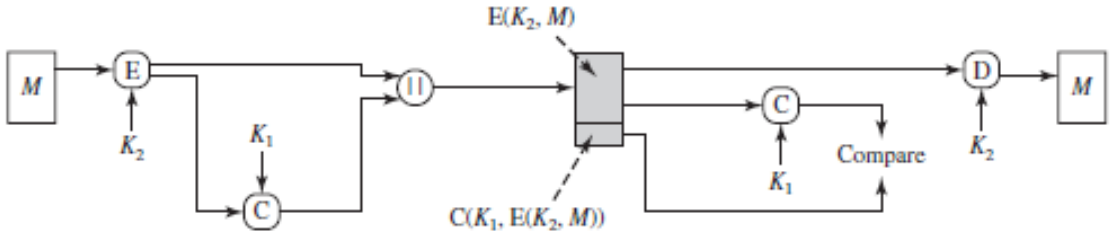
Egy MAC-függvényt háromféleképpen is alkalmazhatunk:



(a) Message authentication



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

Képek forrása: W. Stallings. Cryptography and Network Security. Principles and Practice. Prentice Hall. 2010

Üzenet-hitelesítő kódok, használat

- ha az első ábra szerinti járunk el, akkor nem végzünk rejtjelezést az eredeti üzeneten, az eljárás azonban így is biztosítja az üzenet hitelességét
- ha a második és harmadik ábrák szerinti járunk el, akkor a MAC-érték meghatározása mellett rejtjelezést is végzünk, ahol a rejtjelezéshez egy K_2 kulcsot, míg a MAC-érték meghatározásához egy $K_1 \neq K_2$ kulcsot kell használni.
- a második két ábra szerinti használat biztosítja az üzenet hitelességét, integritását is
- a gyakorlatban a második ábra szerint szoktak eljárni, azaz a MAC-értéket hozzáfűzik a nyílt-szöveghez és ezen végzik a titkosítást
- a harmadik ábra szerinti először a K_2 kulccsal rejtjelezik az üzenetet, majd erre határozzák meg a MAC-értéket
- számos protokoll esetében szükség van úgy a hitelesítésre, mint a bizalmas információcserére, korábban két különálló algoritmust, és két független kulcsot használtak, az utóbbi években egy algoritmuson belül van megoldva mindkét funkció

Üzenet-hitelesítő kódok, a HMAC szerkesztés

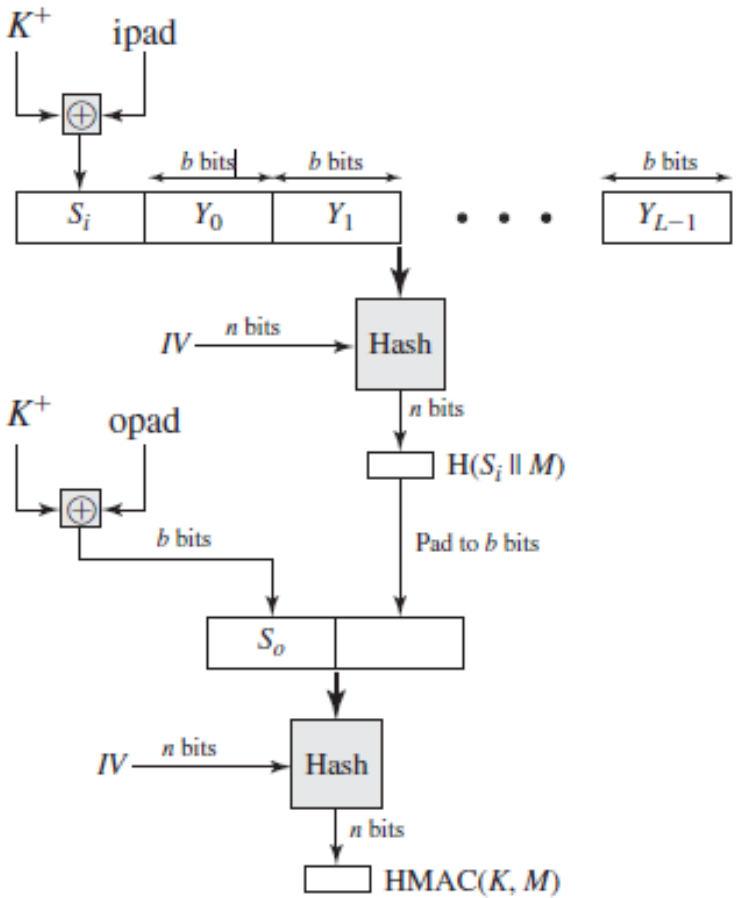
HMAC (Hash-Mac) tervezési szempontok:

- az alkalmazott hash függvény módosítás nélkül legyen használható
- az alkalmazott hash függvény legyen lecserélhető
- őrizze meg az eredeti hash függvény hatékonyságát
- egyszerű legyen a kulcsok használata
- a hash függvény biztonsága alapján következtetni lehessen a MAC-függvény biztonságára
- a következő képlettel adható meg:

$$HMAC_K(M) = H[(K^+ \oplus opad) \,||\, H[(K^+ \oplus ipad) \,||\, M]]$$

- a HMAC egy fontos tulajdonsága, hogy a külső hash bemenete nem ismert

HMAC

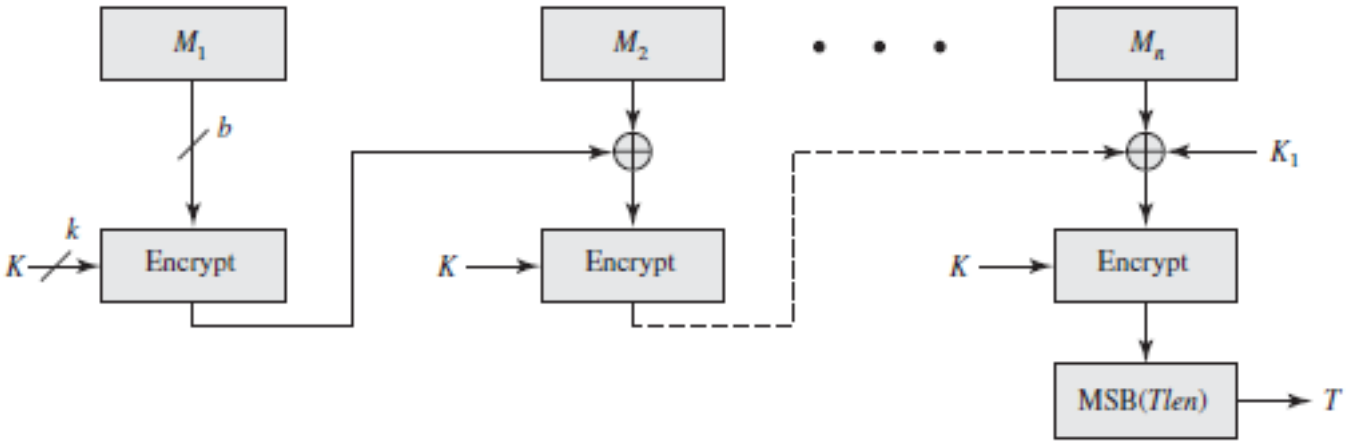


- H - az alkalmazott hash függvény,
- M - a paddingolt üzenet, amelyet az Y_i blokokkra osztottunk, a padding-értékeket az alkalmazott hash függvény határozza meg
- K - a titkos kulcs
- ipad = 0x36, opad = 0x5c
- K^+ - a K kulcs paddingolt értéke: balról nullásokkal egészítjük ki, amíg a kívánt hosszúság b lesz
- L - a blokkok száma
- b - egy blokk bitmérete
- n - a hash függvény kimenetének bitmérete

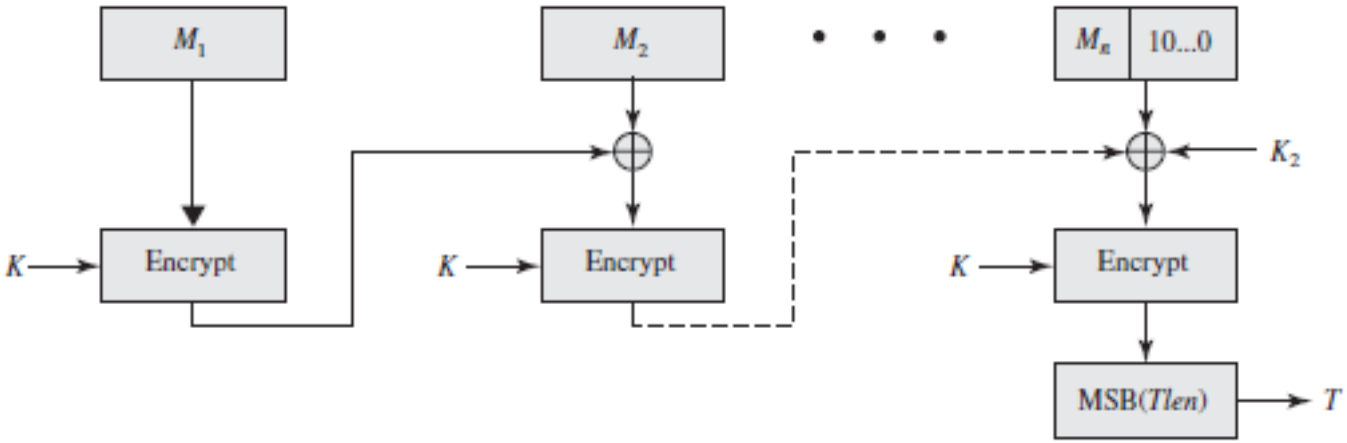
Üzenet-hitelesítő kódok, a CMAC szerkesztése

- blokk-titkosító segítségével: pl. CMAC (Cipher-based MAC) ahol 3DES vagy AES kerül alkalmazásra
- a szerkesztés a CBC blokk titkosító működésén alapszik, ahol a közbenső titkosított blokkok nem kerülnek eltárolásra, az utolsó blokk titkosítása történik másképp, és a MAC-érték ennek a blokknak a titkosított értéke lesz
- különbséget tesznek amikor az üzenet hossza osztható a blokk-titkosító blokk-méretével, illetve mikor nem osztható, az előbbi esetben a K titkos kulcs alapján egy K_1, a második esetben egy K_2 érték kerül meghatározásra, amelyeket az utolsó blokk titkosításánál használnak
- feltételezve, hogy a blokk-méret b, akkor a következőképpen járnak el
  - $K_1 = E_K(0^b) \bullet x$
  - $K_2 = E_K(0^b) \bullet x^2 = (E_K(0^b) \bullet x) \bullet x$
- ahol $\bullet$ polinom-szorzást jelent a $GF(2^b)$ véges testben, egy kiválasztott irreducibilis polinom szerint, pl. b = 128 esetében: $x^{128} + x^7 + x^2 + x + 1$ lesz az irreducibilis polinom

CMAC



(a) Message length is integer multiple of block size



(b) Message length is not integer multiple of block size

Hitelesített titkosítás (authenticated encryption)

- Az ESP IP security protokoll (Encapsulating Security Payload): az IP-csomagokat titkosítja és utána számítja ki a MAC-et
- a TLS (Transport Layer Security) 1.2 protokoll: ha CBC blokk titkosítást alkalmaz, akkor először kiszámítja a MAC-et, majd titkosítja a nyílt szöveget és a MAC-et
- a titkosítás, majd hitelesítés típusú szerkesztéskor két különálló algoritmusra és két független kulcsra van szükség,
- a TLS 1.3 protokoll csak hitelesített titkosítást támogat, például a GCM módot, amely *egyszerre* végzi a hitelesítést és titkosítást
- GCM: Galois Counter Mode
  - a CTR egy kiterjesztése, nagy népszerűségnek örvend
  - a GCM mód egyszerre titkosítja a nyílt szöveget és számol hitelesítési tag-et
  - a tag nem csak a nyílt szöveget hitelesíti, hanem további adatokat (AAD -additional authenticated data) is hitelesít
  - GMAC üzenet-hitelesítési kód, amely a GCM egy változata, csak hitelesítési tag-et határoz meg

GCM - Galois Counter Mode

- 128 bites blokk titkosítók esetében van definiálva
- a hitelesítési tag meghatározása XOR műveletet és polinom-szorzást jelent a $GF(2^b)$ véges testben, pl. b = 128 blokkméret esetében az irreducibilis polinom:
$$x^{128} + x^7 + x^2 + x + 1$$
- a CMAC-hoz képest a bináris szekvenciát little-endian sorrend szerint kezeli: a 128 bites $(b_0 b_1 \dots b_{127})$ blokk a $b_0 + b_1 x + \dots + b_{127} x^{127}$ polinomnak felel meg és a szorzás művelete is eszerint történik
- minden titkosításhoz más 96 bites IV értéket kell meghatározni
- a bemeneti nyílt szöveget 128 bites blokkokra kell osztani: $m_1 \,||\, m_2 \,||\, \dots || \, m_N$
- egy 128 bites számlálót kell megadni: $ctr = IV \,||\, 0^{31} \,||\, 1$
- $H = E_K(0^{128}), A = AAD$, $|A|, |c|$ 64 bites egész számok, amelyeket big-endian sorrend szerint kezel
$$c_i = E_K(ctr + i) \oplus m_i, i = 1, 2, \dots N$$
$$c = IV \,||\, c_1 \,||\, c_2 \,||\, \dots || \, c_N$$
$$X_1 = A \bullet H$$
$$X_i = (X_{i-1} \oplus c_{i-1}) \bullet H, i = 2, 3, \dots N + 1$$
$$X_{N+2} = (X_{N+1} \oplus (|A| \,||\, |c|)) \bullet H$$
- a hitelesített rejtjelezett szöveg: $(c, t = X_{N+2} \oplus E_K(ctr), AAD)$
- ellenőrzéskor először meghatározásra kerül a t és ha az megegyezik a kapott t-vel, akkor kerül sor a visszafejtésre