

Kriptográfia és Információbiztonság

5. előadás

MÁRTON Gyöngyvér

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

2024

Miről volt szó az elmúlt előadáson?

- folyamtitkosító rendszerek: Salsa20, ChaCha20,
- blokktitkosító rendszerek: AES, DES, 3DES, TEA, Blowfish
- blokktitkosító módok: ECB, CBC, CTR

Miről lesz szó?

- hash függvények (hash function)
- üzenethitelesítő kódok (message authentication code)
- hitelesített titkosítás (authenticated encryption)

Hash függvények

- a H hash függvény a bemeneti tetszőleges hosszúságú M adatszekvenciára egy **fix** méretű $h = H(M)$ hash értéket állít elő
- a "jó" hash függvény egyenletes eloszlású és **látszólag véletlenszerű kimenetet** eredményez
- a biztonsági alkalmazásokhoz szükséges hash függvényt kriptográfiai hash függvénynek nevezzük
- a kriptográfiai hash függvény több biztonsági követelménynek is eleget kell tegyen
- egyike a legsokoldalúbb kriptográfiai primitívnek (algoritmusnak): számos biztonsági alkalmazásban, internetes protokollban használják
- elsődleges alkalmazási területük az adatintegritás, de használják:
 - jelszavak tárolásakor
 - titkosító rendszerekben
 - kulcscsere protokollokban
 - digitális aláírásokhoz
 - blokkláncok létrehozásakor

Hash függvények

jelszavak tárolása esetében:

- biztonsági okokból nem közvetlenül a jelszavaknak megfelelő bájt szekvenciát tárolják, hanem egy hashfüggvény segítségével egy másik bájt szekvenciát képeznek és ezt tárolják el
- a hashfüggvény alkalmazása előtt a jelszón egy "normalizálást" is elvégeznek: azaz egy kulcsképző függvényt (**key derivation function**) és egy "toldalék"-bájt szekvenciát (**salt**-ot) felhasználva, egy újabb bájt szekvenciát állítanak elő
- a hash függvény kimeneti értéke mellett, eltárolásra kerül a salt értéke is

Hash függvények

blokkláncok esetében:

- a blokklánc összekapcsolt blokkok sorozata, ahol minden blokk tartalmazza az előző blokk hash értékét
- Pl. egy blokkláncban hatékonyan lehet pénzügyi tranzakciókat tárolni, mert a blokkban lévő tranzakciók nem módosíthatók, csak ha a tranzakciót megelőző összes hash-értéket megváltoztatjuk

bitTorrent esetében:

- állományok cseréje érdekében az állományokat feldarabolják csonkokra
- minden egyes csonknak meghatározzák a hash értékét, amelyeket megosztanak
- a megosztott hash értékek alapján lehet a biztonságos letöltést megoldani

Hash függvények, követelmények

- kulcs nélküli függvények, de létezik kulcsos változatuk is,
- $h : X \rightarrow Y$, hash függvény: tetszőleges hosszúságú bitsorozatból fix hosszúságú bitsorozatot állít elő
- legtöbbször az X jóval nagyobb elemszámú, mint az Y , ez azt is jelenti, hogy **ütkezés fog** fennállni: lesz két különböző bemenet amelyeknek ugyan az lesz a hash értéke
- hatékonyan, **determinisztikusan** számítja ki a hash értékét
- ugyanarra a bemenetre mindig ugyanazt a kimeneti értéket határozza meg
- általában 16-os számrendszerben szokták a kimeneti értéket megjeleníteni
- **lavina effektus**: a bemenet egyetlen bitjének a megváltoztatása a teljes kimenet változásával kell járjon
- a hash kimenete legalább annyi bit kell legyen, hogy ne lehessen **brute-force** támadást alkalmazni

Hash függvények, születésnap paradoxon

- egy $h : X \rightarrow Y$ hash függvény esetében a X jóval nagyobb elemszámú, mint az Y , ez azt is jelenti, hogy ütközések mindig fennállnak
- a születésnap paradoxon megmutatja, hogy az ütközések meglepően gyakran előállhatnak
- **születésnap paradoxon:**
 - 23 véletlenszerűen kiválasztott ember esetében, legalább 50%-os az esélye annak, hogy legalább ketten lesznek olyanok, akiknek **ugyanazon a napon** van a születésnapjuk
 - bebizonyítható, hogy ha $M = 365$, akkor elég $1.17 \cdot \sqrt{M} = 1.17 \cdot \sqrt{365} = 1.17 \cdot 19.10 = 22.35$ napot kiválasztani ahhoz, hogy 50%-nál nagyobb eséllyel találjunk ütközést
- két ugyanazzal a születésnappal rendelkező ember kiválasztása ugyanazt jelenti, mint ütközést találni egy hash függvény esetében
- a fentiek alapján egy **40 bites** hash nem nyújt biztonságot, mert $2^{20} \simeq 10^6$ hash érték meghatározása 50%-nál nagyobb valószínűséggel eredményez ütközést ($1.17 \cdot 2^{20} = 1226833.92$)
- a jelenlegi ajánlások minimum **256 bites** kimenetű hash függvények használatát ajánlják

Hash függvények, tulajdonságok

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 $h(x) = h(x_1)$

Gyakran összekavarják a gyengén ütközésmentes tulajdonságot, az ütközésmentes tulajdonsággal!

Hash függvények

- MD2, MD4, MD5, MD6:

- az MD4-t Ron Rivest publikálta 1990-ben
- az MD5 az MD4 egy továbbfejlesztett változata, 1992-ben
- 2004-ben komoly **biztonsági problémák** merültek fel az MD5-el kapcsolatosan

- SHA, SHA-1, SHA-2:

- **Merkle-Damgård** szerkezetű
- SHA-t a NIST publikálta először 1993-ban
- SHA-1 az SHA kisebb módosításokkal, 160 bites a kimenete, 2017-ben találtak rá ütközést
- SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512): hasonló szerkesztésű, mint az SHA-1, de a belső állapot mérete 256 bit
- a 256, 384, 512 bites változatok 2002 óta standardok
- a 224-es változat 2004 óta standard

Hash függvények

- SHA-3

- 2007-ben a a NIST pályázatot írt ki, 2012-ben a *Keccak* nyert: biztonság, flexibilitás, egyszerűség a jellemzői
- szerkesztése eltér a korábbi módszerektől: **nem Merkle-Damgård** szerkezetű
- SHA3-224, SHA3-256, SHA3-384, SHA3-512 változatok

- RIPEMD-128-160-256-320

- először Belgiumban publikálták, 1996-ban
- szabad felhasználási lehetőség, nincs levédve

- WHIRLPOOL

- 2000-ben publikálták,
- egyik szerkesztője Rijmen,
- az ISO által elfogadott nemzetközi standard.

Hash függvények, általános szerkesztés

A legismertebb, legelterjedtebb szerkesztési mód a **Merkle-Damgård szerkesztés**:

- legyen $h : X \rightarrow Y$ a hash függvény
- ha az x bitsorozat hash értékét akarjuk megszerkeszteni:
 - felosztjuk x -et l -bit hosszúságú bit-sorozatokra: x_1, x_2, \dots, x_k ,
 - szükség esetén kiegészítjük x -et további bitekkel
 - definiálunk egy f **tömörítő** függvényt, amelyet rekurzívan alkalmazunk
 - meghatározunk egy kezdeti IV értéket, amely betöltheti a kulcs szerepét is, általában azonban előredefiniált konstans értéket jelent
 - az általános szerkesztési lépéssorozat:

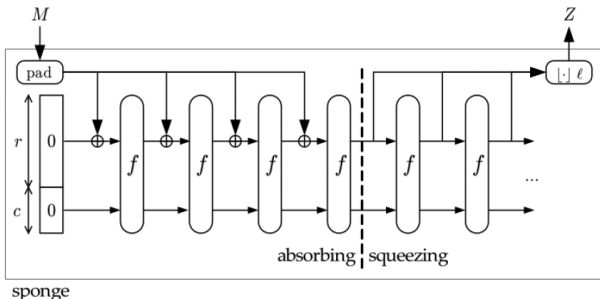
$$\begin{aligned} H_0 &= IV \\ H_i &= f(H_{i-1}, x_i), i = 1, 2, \dots, k \\ h(x) &= H_k \end{aligned}$$

- az f függvény és a IV határozzák meg a hash függvény biztonságát

Hash függvények, általános szerkesztés

A másik, újabban használt szerkesztési mód a szivacs (**sponge**) szerkesztés:

- Bertoni, Daemen, Peeters és Van Assche fejlesztették
- a tömörítő függvény helyett alkalmazott f függvény (ez általában bijektív) fix hosszúságú bitsorozatból azonos hosszúságú bitsorozatot határoz meg
- a sponge szerkezet lehetővé teszi a tetszőleges hosszúságú kimenetet
- $f : \{0, 1\}^b \rightarrow \{0, 1\}^b$, esetében a $b = r + c$ egész számot szélességnek (**width**), az r -et bitmértéknek (**bitrate**), és a c -t kapacitásnak (**capacity**) hívjuk
- r bitenként dolgozzuk fel az üzenetet, ezért az r meghatározza a sponge függvény hatékonyságát, a c biztonságáért felelős



Hash függvények, általános szerkesztés

a szivacs (**sponge**) szerkesztés két részre osztható: elnyelő fázis (absorbing phase) és összenyomó fázis (squeezing phase)

- az **absorbing phase** kimenete az $x_k || y_k$ bitsorozat, amelynek hossza $r + c$:
 - $M = m_1 || \dots || m_k$, ahol $m_1, \dots, m_k \in \{0, 1\}^r$,
 - szükség esetén kiegészítjük M -et további bitekkel, azért hogy osztható legyen r -el
 - legyen $x_0 = \underbrace{00 \dots 0}_r$ és $y_0 = \underbrace{00 \dots 0}_c$ a kezdeti állapotérték
 - az aktuális állapotérték: $x_i || y_i$, ahol:

$$x_i || y_i = f_{i-1}(x_{i-1} \oplus m_i || y_{i-1}), i = 1, 2, \dots, k$$

- **squeezing phase**:
 - ha $l \leq r$, akkor a hash függvény kimenete az x_k első l bite lesz
 - ha $l > r$, akkor még egyszer alkalmazzuk az f függvényt az aktuális állapotértéken és meghatározunk r kimeneti bitet, a folyamatot addig ismételjük, amíg el nem érjük a kívánt l hosszúságot

Az SHA3 (Secure Hash Algorithm) hash függvény

- a Keccak alapján szerkesztették
- az **állapotérték** (state) az $b = r + c$ egy $5 \times 5 \times 64 = 1600$ bites érték:
- az $f : \{0, 1\}^{1600} \rightarrow \{0, 1\}^{1600}$ függvény bijektív:
 - kijelenthető, hogy hasonló a **random permutációhoz**
 - hasonló az AES-ben alkalmazott függvényhez, ez nem véletlen mert az egyik tervező részt vett az AES tervezésében (Daemen)
 - XOR, AND, NOT bitműveleteket használ, ezért gyors úgy a szoftveres, mint hardveres implementáció

Hash-kimenet	r (rate)	c (capacity)	r + c (state/állapotérték)
224	1152	448	1600
256	1088	512	1600
384	832	768	1600
512	575	1024	1600

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

Üzenet-hitelesítő kódok (Message Authentication Code)

- 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 rendszerállományok 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 adatforgalom** kell kezeljen egy rendszer akkor az összes üzenet visszafejtése csak fölöslegesen terhelné a rendszert
- alkalmazások integritásának az ellenőrzésére is elég a MAC
- jó **külön választani** az integritásvizsgálatot a titkosítástól: az integritásvizsgálatot az alkalmazások szintjén a titkosítást a szállítási rétegben érdemes implementálni

Üzenet-hitelesítő kódok, matematikai modell

- legyen P az üzenetek és K a kulcsok halmaza
- a **kulcs generálás** algoritmus polinom idejű, véletlenszerű: $Gen \rightarrow key$, úgy hogy $key \in K$
- a **MAC-érték meghatározásának** algoritmus 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

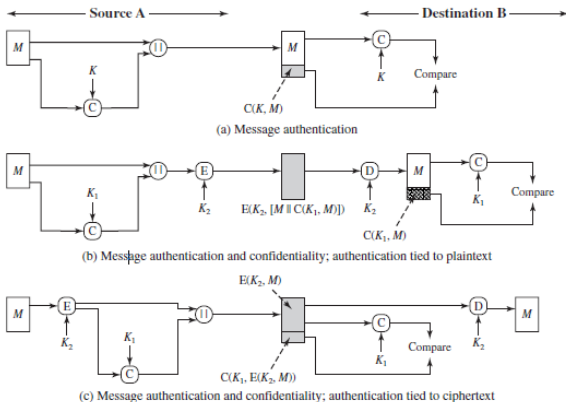
- egy adott MAC-függvény esetében a következő feladatok **ne legyenek megoldhatóak** polinomiális időben:
 - több m_i és $MAC_{key}(m_i)$ pár ismeretében, határozzuk meg $MAC_{key}(m)$ -t, ahol $m \neq m_i$, és ahol a key értéke sem ismert
 - több m_i és $MAC_{key}(m_i)$ pár ismeretében határozzuk meg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



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 ebredeti 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 rejtjelezzük az üzenetet, majd erre határozzák meg a MAC-értéket

Üzenet-hitelesítő kódok, a HMAC szerkesztés

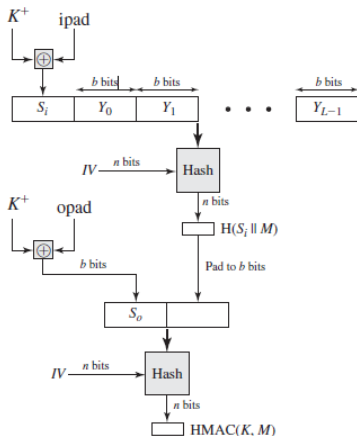
HMAC tervezési szempontok:

- 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:

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

- 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 blokkokra osztottunk, a padding-értékeket az alkalmazott hash függvény határozza meg
- K - a titkos kulcs
- $ipad = 0x36$, $opad = 0x5c$, $b/8$ -szor ismételve őket
- 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 bit mérete

Üzenet-hitelesítő kódok, a CMAC szerkesztése

- blokk-titkosító segítségével: pl. CMAC (Cipher-based MAC) ahol 3DES vagy AES kerül alkalmazásra
- a szerkesztés a **CBC blokktitkosító** működésén alapszik, ahol a közbenső titkosított blokkok nem kerülnek eltárolásra, az utolsó blokk titkosítása történik másképp, és a MAC-érték ennek a bloknak a titkosított értéke lesz
- ha az üzenet hossza osztható a blokktitkosító blokkméretével, akkor a K titkos kulcs alapján egy K_1 kerül meghatározásra:

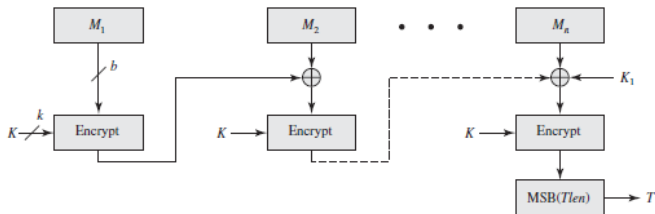
$$K_1 = E_K(0^b) \bullet y, \text{ ahol } y = \underbrace{0 \dots 0}_{b-2 \text{ bit}} 10$$

- ha az üzenet hossza nem osztható a blokktitkosító blokkméretével, akkor a K titkos kulcs alapján egy K_2 kerül meghatározásra:

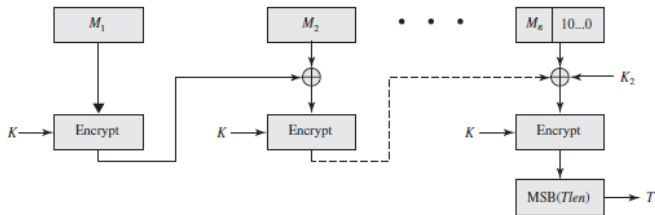
$$K_2 = E_K(0^b) \bullet y^2, \text{ ahol } y^2 = \underbrace{0 \dots 0}_{b-3 \text{ bit}} 100.$$

- a K_1 és K_2 értékeket az utolsó blokk titkosításánál használják
- a \bullet polinomszorzást jelent a $GF(2^b)$ véges testben, megadott irreducibilis polinomok 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

A GCM (Galois Counter Mode) blokktitkosító mód

- 128 bites blokk titkosítók esetében van definiálva, ezek bármelyikénél lehet használni,
- a CTR mód továbbfejlesztett változata,
- minden titkosításhoz más random IV (nonce) értéket kell használni,
- 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 GCM (Galois Counter Mode) blokktitkosító mód

- a 128 bites számláló kezdeti értéke: $ctr = IV || 0^{31} || 1$, ahol az IV 96 bit.
- az m nyílt szöveget 128 bites blokkokra bontja, ahol az utolsó blokk lehet kisebb is mint 128,
- $m = m_1 || m_2 || \dots || m_N$,

$$c_i = E_{key}(ctr + i) \oplus m_i, \text{ minden } i = 1, 2, \dots, N$$
$$c = IV || c_1 || c_2 || \dots || c_N$$

legyen $H = E_{key}(0^{128})$ egy 128 bites hash függvény, és $A = AAD$ egy 128 bites *additional authenticated data*:

$$X_1 = A \bullet H$$
$$X_i = (X_{i-1} \oplus c_{i-1}) \bullet H, \text{ minden } i = 2, 3, \dots, N + 1$$
$$X_{N+2} = (X_{N+1} \oplus (len(A) || len(c))) \bullet H,$$
$$t = X_{N+2} \oplus E_{key}(ctr)$$

- a kimenet: (c, t, AAD) , ahol t a hitelesítő tag,
- a \bullet művelet szorzást jelent $(\text{mod } x^{128} + x^7 + x^2 + x + 1)$ szerint a $GF(2^{128})$ véges testben,
- $len(A)$, $len(c)$ 64 biten vannak tárolva.

A Poly1305 üzenethitelesítő kód

- egy 32 bájtos titkos értéket használ:
 - ennek az első 16 bájtja lesz a **key**,
 - a következő 16 bájtira, azaz az r_0, r_1, \dots, r_{15} bájtokra fenn kell álljon:
 $r_3, r_7, r_{11}, r_{15} \in \{0, 1, \dots, 15\}$ és $r_4, r_8, r_{12} \in \{0, 4, 8, \dots, 252\}$
 - ezután meghatározza r -t:

$$r = r_0 + 2^8 r_1 + \dots + 2^{120} r_{15},$$

- legyen az $m = m[0], m[1], \dots, m[l-1]$ üzenet l hosszúságú, és $q = l // 16$,
- $1 \leq i \leq q$ -ra meghatározza a $c_i \in \{1, 2, 3, \dots, 2^{129}\}$ értékeket:
 - ha l 16 többszöröse:

$$c_i = m[16i-16] + 2^8 m[16i-15] + 2^{16} m[16i-14] + \dots + 2^{120} m[16i-1] + 2^{128}$$

- ha l nem 16 többszöröse:

$$c_q = m[16q-16] + 2^8 m[16q-15] + \dots + 2^{8(l \bmod 16)-8} m[l-1] + 2^{8(l \bmod 16)}$$

- minden üzenethez különböző 16 bájtos n nonce értéket használ,
- az m üzenet hitelesített értékét a következőképpen határozza meg:

$$((c_1 r^q + c_2 r^{q-1} + \dots + c_q r^1) \pmod{2^{130} - 5} + E(\text{key}, n)) \pmod{2^{128}}$$