Kriptográfia és Információbiztonság
4. előadás

MÁRTON Gyöngyvér

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

2024

Miről volt szó az elmúlt előadáson?

- titkos kulcsú rendszerek: matematikai modell, gyakorlatban való alkalmazásuk, osztályozás
- folyamtitkosító rendszerek: jellemzők, biztonsági problémák
- random számok generálása: true, pszeudo
- RC4, LFSR (linear feedback shift register), A5/1, Salsa20, ChaCha20.

Miről lesz szó?

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

A ChaCha20 folyamtitkosító

- a Salsa20-nak egy módosított változata, tervezője szintén Bernstein,
- az algoritmusnak elérhető úgy hardveres, mint softveres implementációja,
- széles körben elterjedt, használja az Open SSH, az SSL/TLS, és a Noise,
- a Google 2013-ban szabványosította, Android eszközökön is lehet használni,
- a nyílt szöveg hosszával megegyező kulcsfolyamot állít elő,
- a Chacha20-Poly1305 a társított adatokkal való hitelesített titkosítást (authenticated encryption with associated data (AEAD)) tesz lehetővé

A ChaCha20 folyamtitkosító

- működését egy H pszeudorandom függvény határozza meg, amelynek bemenete 256 bit, kimenete pedig 512 bit,
- a H függvény 512 bites kimenetét 16 szóra osztják fel, és az így kapott 32 bitet 4 bájtonként $\pmod{2^{32}}$ szerinti összeadást végezve adják hozzá a nyílt szöveg bájtjaihoz,
- hitelesített titkosítás (AEAD):
  - egy újabb titkosítási irány, biztosítja a titkosítását, a hitelesítését, és az integritást,
  - a rejtjelezett szöveg mérete nagyobb lesz, mert a hitelesítéshez szükséges tag-eket is tartalmazza
  - a társított adatok a nyílt szöveghez kapcsolódó metadadatok,

A ChaCha20 folyamtitkosító

A H függvény bemenete:

- egy 256 bites kulcs:

$$k = k_0 \,||\, k_1 \,||\, k_2 \,||\, k_3 \,||\, k_4 \,||\, k_5 \,||\, k_6 \,||\, k_7$$

- egy 64 bites $r = r_0 \,||\, r_1$ nonce (number use at once),
- egy nullától induló 64 biten tárolt $i = i_0 \,||\, i_1$ számláló érték.

$$\begin{aligned} ChaCha20 &: \{0,1\}^{256} \times \{0,1\}^{64} \to \{0,1\}^n \\ ChaCha20(k,r) &= H(k \,||\, 0 \,||\, r)) \,||\, H(k \,||\, 1 \,||\, r) \,||\, \ldots . \end{aligned}$$

- a $k \,||\, i \,||\, r$ bemenet egy 4×4-es mátrix alakba kerül, a kimenet is egy 4×4-es mátrix:

$$\begin{pmatrix} t_0 & t_1 & t_2 & t_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ i_0 & i_1 & r_0 & r_1 \end{pmatrix} \longrightarrow \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix},$$

- minden mátrix elem egy 32 bites értéket jelöl (words).

A ChaCha20 folyamtitkosító

- a t_0, t_1, t_2, t_3 értékek konstansok,
- az `expand 32-byte k` karakterlánc 4-esével vett ASCII kódjaiból átakaított egész szám hexadecimális értékeit jelölik:

$$t_0 = 61707865 \quad t_1 = 3320646E,$$
$$t_2 = 79622D32 \quad t_3 = 6B206574.$$

```
constChaCha = b'expand 32-byte k'
for i in range(0, 16, 4):
    t = int.from_bytes(constChaCha[i:i+4], byteorder = 'little')
    print(hex(t), end = ' ')

t = [0x61707865, 0x3320646e, 0x79622d32, 0x6b206574]
for i in range(4):
    s1 = t[i].to_bytes(8, byteorder = 'little')
    for j in range(8):
        print(chr(s1[j]), end = '')
```

A ChaCha20 folyamtitkosító

- 10-szer végrehajtja a következő 8 lépést, ahol a $h : \{0,1\}^{128} \to \{0,1\}^{64}$ egy invertálható függvény,
- a h első négy meghívását a bemeneti mátrix első négy oszlopára alkalmazza, majd az átlón levő elemek következnek, úgy hogy először a főátlón levő elemek, majd a főátlótól jobbra levők lesznek a bemeneti értékek:

$$
\begin{array}{ll}
(1) : h(x_0, x_4, x_8, x_{12}), & (2) : h(x_1, x_5, x_9, x_{13}), \\
(3) : h(x_2, x_6, x_{10}, x_{14}), & (4) : h(x_3, x_7, x_{11}, x_{15}), \\
(5) : h(x_0, x_5, x_{10}, x_{15}), & (6) : h(x_1, x_6, x_{11}, x_{12}), \\
(7) : h(x_2, x_7, x_8, x_{13}), & (8) : h(x_3, x_4, x_9, x_{14}).
\end{array}
$$

- a mátrix minden szavát kétszer is újraszámítja,
- a h minden hívása után átlagosan 12.5 kimeneti bit értéke lesz megváltoztatva.

A ChaCha20 folyamtitkosító

A *h* XOR-t, (mod 32) szerinti összeadást, és a *rot* függvényt alkalmazza:

```
BITS_INT32 = 32
def rot(b, i):
    b = (b << i) | (b >> (BITS_INT32 - i))
    return b & 0xFFFFFFFF

def h(a, b, c, d):
    a = (a + b) & 0xFFFFFFFF
    d = rot(d ^ a, 16)

    c = (c + d) & 0xFFFFFFFF
    b = rot(b ^ c, 12)

    a = (a + b) & 0xFFFFFFFF
    d = rot(d ^ a, 8)

    c = (c + d) & 0xFFFFFFFF
    b = rot(b ^ c, 7)
    return (a, b, c, d)
```

Blokktitkosító rendszerek

- a blokktitkosító rendszerek a kripto alapelemei, számos rendszer felépítésében játszanak fontos szerepet,
- bájttömbönként alkalmaznak egy álvéletlen függvényt/permutációt,
- a rendszer biztonságát az alkalmazott álvéletlen permutáció határozza meg
- fix méretű blokkokkal végeznek műveleteket,
- mit, illetve mit nem oldanak meg?
  - a titkosítás nem lesz randomizált,
  - nem garantálják az üzenet integritását,
  - nem oldják meg a rejtjelezéshez használt kulcs megosztását,
  - nem teszik lehetővé a kommunikáló felek hitelesítését,
- a rendszer biztonságát az alkalmazott álvéletlen permutáció határozza meg.

Blokktitkosító rendszerek

- egyszerre n bájot rejtjeleznek, a kimenet is n bájt lesz,
- a kulcsot bájt tömbönként újra használják $\Rightarrow$ blokktitkosító módok,
- egy blokktitkosító rendszer átalakítható folyamtitkosító rendszerré,
- több körből áll a titkosítás, az eredeti kulcs alapján különböző körkulcsokat hoznak létre, amelyeket különböző körökben használnak
- két nagy csoport:
  - Feistel sémán alapuló rendszerek (pl. DES):
    - minden körben felosztják a bemeneti bitsort, egyik részen egy helyettesítést hajtanak végre, amely után felcserélik a két részt,
  - helyettesítést és permutációt alkalmazó rendszerek (pl. AES).

Blokktitkosító módok

- ha a nyílt szöveg nagyobb mint n bájt, akkor:
  - szükséges alkalmazni valamilyen blokk titkosító módot,
  - a nyíltszöveget fel kell osztani n bájtos részekre,
  - a nyíltszöveget "padding"-olni (kiegészíteni) kell
  - `PKCS#7`: ha `N` darab bájtot kell hozzáadni, akkor az `N` értékét adjuk hozzá `N`-szer
- a kulcs többszöri felhasználása:
  - biztonsági problémák,
  - NIST által standardizált blokktitkosító módok,
  - egy adott blokktitkosító mód: különböző biztonsági szintet határoz meg, különböző protokollokban használható.

Blokktitkosítók a gyakorlatban

- legtöbbször iterált szerkezetűek:
  - round function: egy belső $\hat{\mathcal{E}} = (\hat{E}, \hat{D})$ kör függvényt használnak, amely helyettesítést és permutációt hajt végre,
  - key schedule: egy álvéletlenszám generátort, egy *Gen* expansion függvényt alkalmaznak, hogy a *key* kulcs alapján a $k_1, \ldots k_d$ körkulcsokat (round key) meghatározzák:
- minden körben más körkulcsot alkalmaznak: $(k_1 \ldots, k_d) \leftarrow Gen(key)$

```
for i in range(1, k + 1):
```

$$y \leftarrow \hat{E}(k_d, \hat{E}(k_{d-1}, \ldots, \hat{E}(k_2, \hat{E}(k_1, x)) \ldots))$$

- $\hat{E}(\cdot)$-nak két bemenete van: az aktuális állapotérték, és a megfelelő körkulcs, ahol a kezdő állapotérték a nyíltszöveg,
- úgy a nyíltszöveg, mint a rejtjelezett szöveg pontosan meghatározott méretű bitszekvenciák lesznek, pl. az AES-nél 128 bit,
- a visszafejtő függvény ugyanaz, a körkulcsokat azonban fordított sorrendben kell alkalmazni $x \leftarrow \hat{D}(k_1, \hat{D}(k_2, \ldots, \hat{D}(k_{d-1}, \hat{D}(k_d, y)) \ldots))$

Blokktitkosítók a gyakorlatban

	kulcsméret (bitek)	blokkméret (bitek)	körök száma	hatékonyság (MB/sec)
DES	56	64	16	80
3DES	168	64	48	30
AES-128	128	128	10	163
AES-256	256	128	14	115

- a gyakorlati tapasztalat azt mutatja, hogy az iterált szerkesztés biztonságos blokktitkosítót eredményez
- egy lineáris függvény iterálása nem eredményez biztonságos blokktitkosítót
- a blokktitkosítók tervezése, implementálása nem egy triviális feladat.

Blokktitkosítók a gyakorlatban

- a kör függvény (round function) helyettesítést és permutációt alkalmaz
- helyettesítés:
  - n darab bit másik, különböző n darab bittel való helyettesítését jelenti
  - a műveletet végző függvényt S-boxnak is hívják, amely substitution vagy akár secret boxot is jelent,
- permutáció: n darab bit permutálása, nem az értéküket, hanem a sorrendjüket változtatják meg,
- több kört alkalmaznak, amelynek során helyettesítést, permutációt végeznek, majd az utolsó kör végén egy permutációt hajtanak végre

Blokktitkosítók a gyakorlatban

- a biztonság növelése érdekében a kulcsokat *fehéríteni* (whitening) szokták
- az első kör előtt és az utolsó kör után egy-egy $\oplus$ műveletre kerül sor:
  - a blokktitkosító alkalmazása előtt meghatározzák a nyíltszöveg és egy kulcs $\oplus$ értékét,
  - titkosításkor a blokktitkosítás után kapott rejtjelezett blokk értékének és egy másik kulcsnak határozzák meg az $\oplus$ értékét:

$$c = E_{key}(k_1, k_2, m) = E_{key}((m \oplus k_1)) \oplus k_2$$

  - visszafejtéskor:

$$m = D_{key}(k_1, k_2, c) = D_{key}((c \oplus k_2)) \oplus k_1$$

- a blokktitkosító kevésbé lesz támadható brute-force-al: a k_1, k_2 alkalmazásával ugyanis megnő a lehetséges kulcsok száma

Blokktitkosító módok, jellemzők

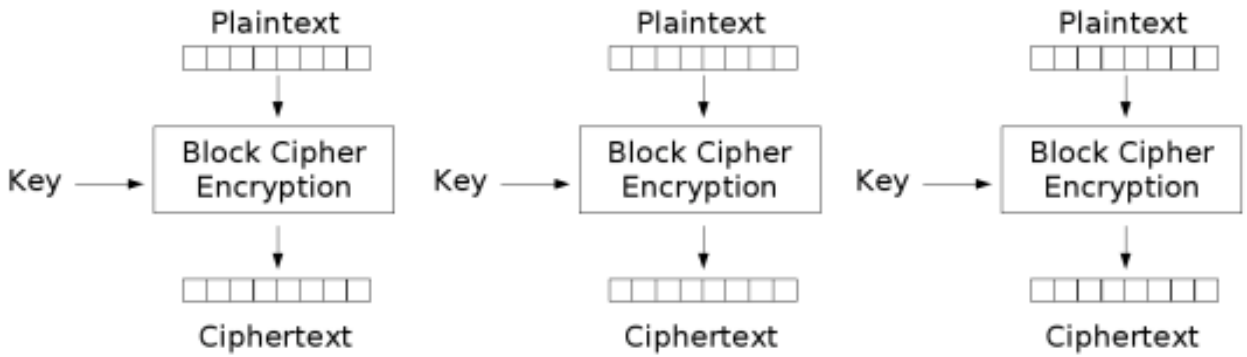
a következőket a DES-titkosító számára dolgozták ki még 1981-ben:

blokktitkosító mód	alkalmazhatóság
ECB	csak egy blokk például a kulcs titkosítására használható, másképp nem biztonságos
CBC	általános blokktitkosító, a titkosítás nem, de a visszafejtés párhuzamosítható
CFB	a titkosítást átalakítja folyamtitkosítóvá
OFB	a titkosítást átalakítja folyamtitkosítóvá, hibajavító kódok esetében használják

- bármilyen blokktitkosító esetében használhatóak
- további blokktitkosító módok: CTR, általános blokktitkosító, nagyon gyors
- nem biztosítják az üzenetek sértetlenségét, manipulálását

ECB (Electronic Code Book)

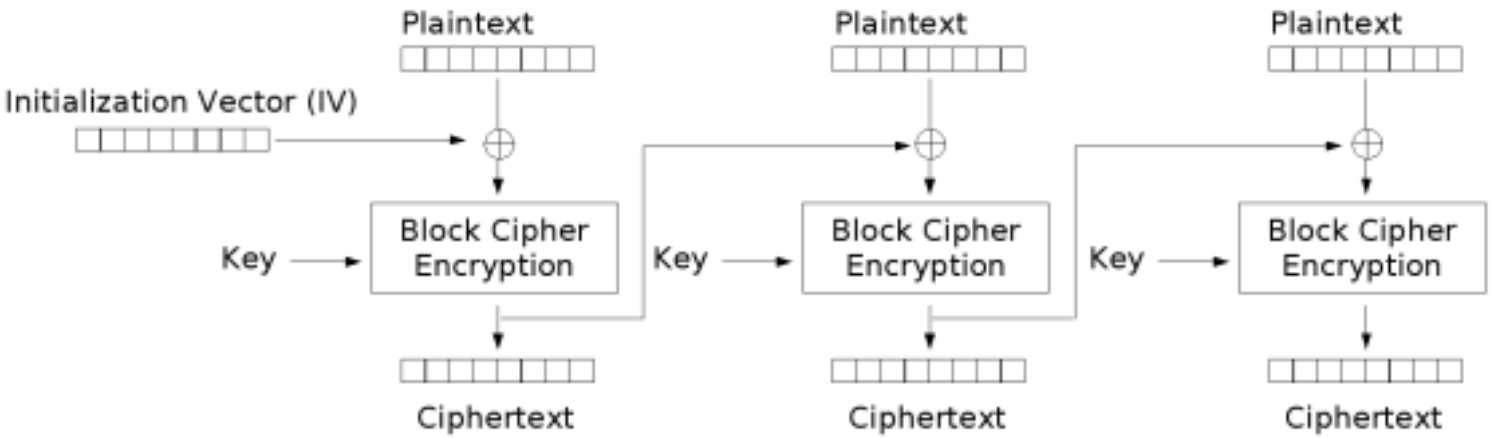
- a nyíltszöveg blokkjait egymástól függetlenül titkosítják, a titkosított szöveg a titkosított blokkok konkatenációja lesz, visszafejtésnél ugyanígy járnak el,
- rövid adathalmaz titkosítására használják, pl. kulcsok,
- biztonsági problémák:
  - ha két blokk egyforma, akkor ezeknek a rejtjelezett értéke is egyforma lesz
  - anélkül, hogy a kommunikáló felek észrevennék egy támadó felcserélhet két rejtjelezett blokkot, vagy kitörölhet, kicserélhet egy-egy blokkot,



Képek forrásai: *http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation*

CBC (Cipher-Block Chaining)

- titkosítás: a titkosító bemenete az aktuális nyíltszöveg és az előző rejtjelezett szöveg $\oplus$ szerint meghatározott értéke,



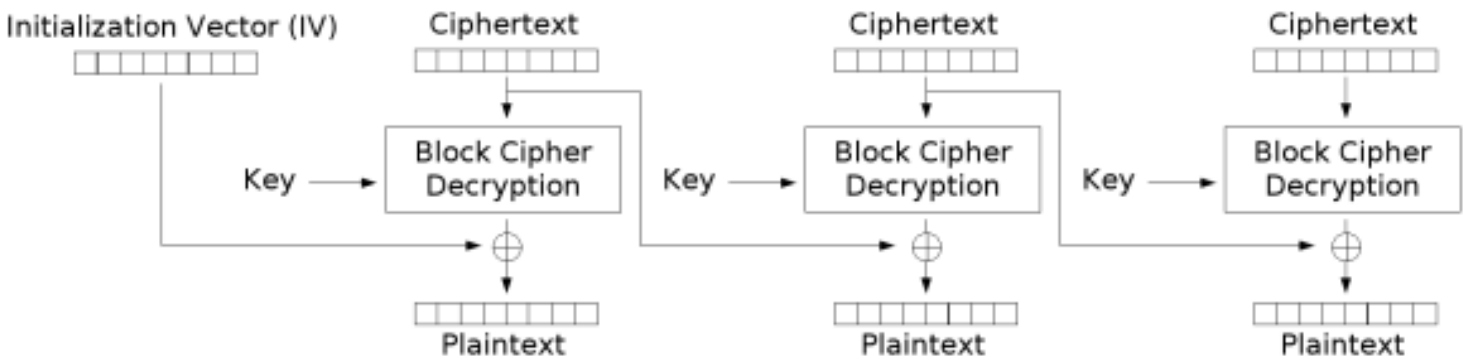
$$C_0 = IV, C_i = E_{key}(M_i \oplus C_{i-1}), i = 1, 2, \ldots$$

CBC (Cipher-Block Chaining)

- egy blokk titkosítása az előző blokk titkosított értékétől függ,
- az első blokk esetében egy kezdeti bájttömbböt, egy IV-t (initialization vector) használunk:
  - az IV-t nem muszáj titokban tartani, de kiszámíthatatlan kell legyen,
  - az IV lehet
    - egy időpecsét (time stamp),
    - egy számláló (counter),
    - álvéletlen módon generált bájttömb, amelyet nonce-nak (a number used only once) is hívnak,
- ha a nyílt szöveg egy bitje megváltozik (pl. sérül az adattovábbítás során), akkor a megfelelő rejtjelezett blokk értéke és az utána következő még két blokk rejtjelezett értéke meg fog változni,
- használata:
  - nagy adathalmaz titkosításakor,
  - üzenet hitelesítő kódok szerkesztésekor

CBC (Cipher-Block Chaining)

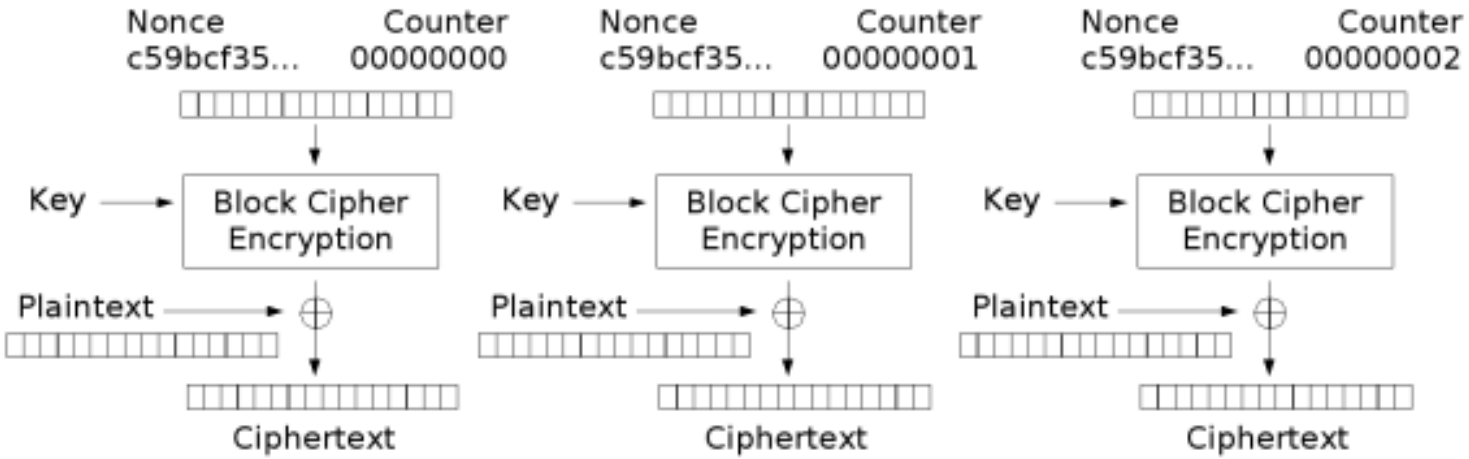
- visszafejtés: visszafejtjük az aktuális blokkot, majd meghatározzuk az eredmény és az előző rejtjelezett blokk $\oplus$ értékét



$$C_0 = IV, M_i = D_{key}(C_i) \oplus C_{i-1}, i = 1, 2, \ldots$$

CTR (Counter mode)

- egy folyamtitkosítást hajt végre,
- egy számláló és egy nonce érték alapján, a nyíltszövegtől függetlenül egy blokk titkosítót alkalmazva egy kulcsfolyamot generál,
- a kapott kulcsfolyamot és a nyíltszöveg blokkjait $\oplus$-al összeadja

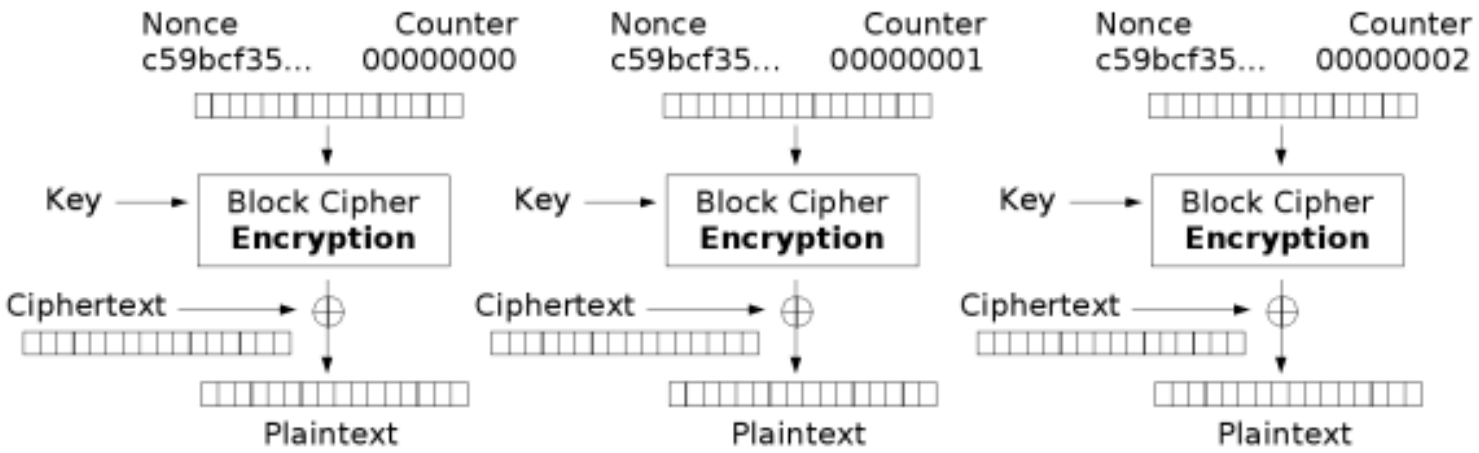


$$K_i = E_{key}(\texttt{Nonce}, \texttt{Counter}_i), C_i = K_i \oplus M_i, i = 1, 2, \ldots$$

CTR (Counter mode)

visszafejtés:

- újra kigenerálják a kulcsfolyamot,
- mivel az $\oplus$ művelet szimmetrikus, a titkosítás és visszafejtés ugyanaz



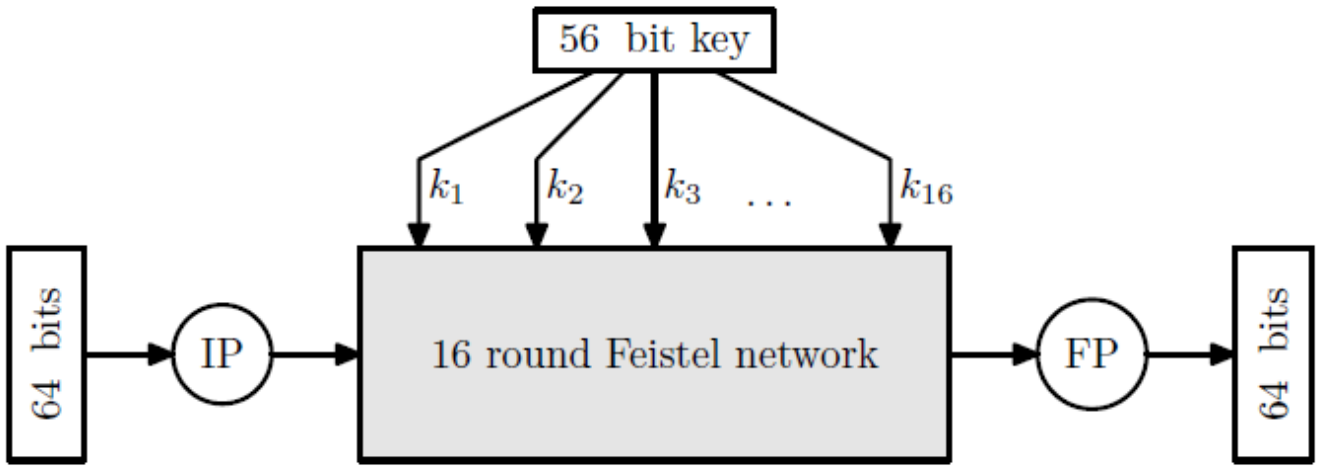
$$K_i = E_{key}(\texttt{Nonce}, \texttt{Counter}_i), M_i = K_i \oplus C_i, i = 1, 2, \ldots$$

DES (Data Encryption Standard)

- az IBM tervezte a NIST kérésére, 1975-ben fogadták el standardként,
- megjelenésével a kriptoanalizis is fejlődésnek indult,
- elődje a Lucifer blokktitkosító, amelynek 128 bites volt a blokkmérete, kulcsmérete szintén 128 bit volt,
- a NIST kisebb kulcsmérettel rendelkező blokktitkosítót kért, így a DES kulcsmérete 56 bit lett,
- azóta számos tudományos elemzés látott napvilágot, és felmerült a gyanú hogy szándékos volt a kis kulcsméret melletti döntés, illetve hogy backdoort tartalmaz,
- a biztonság növelése érdekében bevezették a triple-DES-t: DES-EDE (DES-encryption-decryption-encryption)
- a 3DES megőrzi a DES belső szerkezetét, háromszor nagyobb a kulcsmérete, de háromszor olyan lassú,
- smart kártyák esetében 3DES-t használnak a MAC előállítására,
- a NIST kormányzati célokra, 2030-ig elfogadta a triple-DES használatát,
- 2002-ben váltja fel az AES.

DES (Data Encryption Standard)

- iterációszáma (a körök száma): 16, blokkmérete 64 bit,
- az iterációt megelőzően az 56 bites kulcsot minden 7 bit után kiegészítik egy paritásbitel, amely eredményeként egy 64 bites kezdeti kulcsot kapnak, ez alapján lesz kigenerálva a további 16 kulcs



kép forrása: *Boneh and Shoup/DES*

DES, kulcsgenerálás

Példa:

- legyen az 56 bites kulcs *key*: 0x12, 0x69, 0x5b, 0xc9, 0xb7, 0xb7, 0xf8, amelynek a bináris alakja a következő:
  00010010 01101001 01011011 11001001 10110111 10110111 11111000
- 7-vel felosztva kapjuk:
  0001001 0011010 0101011 0111100 1001101 1011110 1101111 1111000
- a *key*-t paritásbittekkel kiegészítve kapjuk key_0-t:

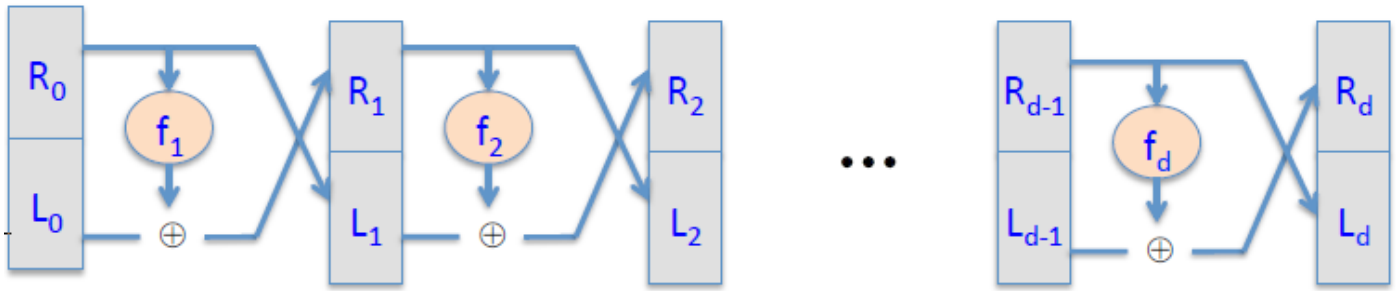
00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

0	0	0	1	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	1	0	1	1	1
0	1	1	1	1	0	0	1
1	0	0	1	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	1	1	1	1	1
1	1	1	1	0	0	0	1

DES, titkosítás

Feistel sémán alapszik:

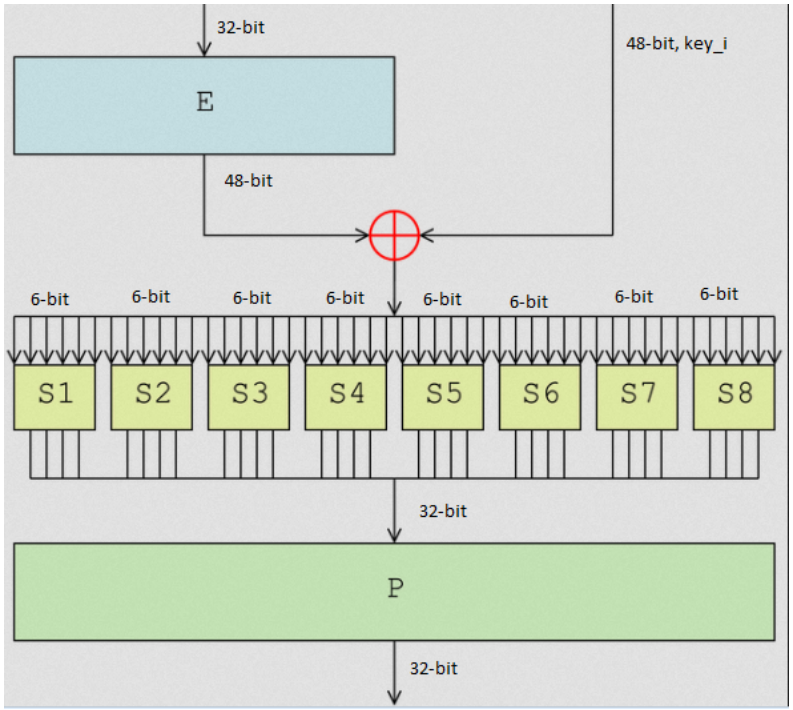
- minden körben felosztják a bemeneti bitsort,
- egyik részen permutációt (P-box) és helyettesítést (S-box) hajtanak végre (f_i alkalmazása),
- majd megcserélik a két részt:



kép forrása: *Boneh/Online Cryptography*

DES, titkosítás

- az S-boxban elvégzett műveleteken kívül minden más művelet lineáris,
- mai napig sem publikus, hogy mi alapján szerkesztették az S-boxokat,
- érvényesül a lavina effektus úgy a titkosító, mint a kulcsgeneráló algoritmusnál: kis változtatás a bemeneten nagy változtatást eredményez a kimeneten.



kép forrása: *wiki/DES*

DES, S-boxok

S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
⋮															

$S_i(B)$-t, ahol $B = b_1 b_2 b_3 b_4 b_5 b_6$ a következőképpen határozzuk meg:

- $b_1 b_6$ meghatározza a sorindexet,
- $b_2 b_3 b_4 b_5$ meghatározza az oszlopindexet,
- $S_i(B)$, a megfelelő indexnél levő érték bináris alakja,

Példa: $S_1(111100) = 0101$, mert $b_1 b_6 = 10$ (2. sor), és $b_2 b_3 b_4 b_5 = 1110$ (14. oszlop). Az itt található érték 5, aminek bináris alakja 0101.

DES, tervezési szempontok

- blokkméret: nagyobb blokkméret esetén csökken a rejtjelezési sebesség, de nagyobb a biztonság,
- kulcsméret: nagyobb kulcs méret esetén is csökken a rejtjelezési sebesség, de nagyobb a biztonság is,
- mennyi legyen a körök száma?
- hogyan generáljuk a körkulcsokat? $\rightarrow$ nagy komplexitás
- hogyan értelmezzük a titkosító függvényt? $\rightarrow$ nagy komplexitás
- a visszafejtés ugyanaz, mint a titkosítás, csak fordított sorrendben kell alkalmazni a kulcsokat $\rightarrow$ nem kell két algoritmust írni.

DES, biztonság

- Az alkalmazott alapműveletek:
  - helyettesítés (substitution): az F alkalmazása, majd a $\oplus$,
  - permutáció (permutation): a két 32 bites rész felcserélése,
  - Shannon: helyettesítést és permutációt alkalmazva megváltozik nyílt-üzenet statisztikai tulajdonsága $\Rightarrow$ nem alkalmazhatóak statisztikai elemzések a feltörés során.
- Az S-, és P-boxok megválasztása,
  - nem lehetnek lineárisak,
  - az S-boxban elvégzett műveleteken kívül minden más művelet lineáris,
  - nem lehetnek véletlenszerűek,
  - mai napig sem publikus az a kritériumrendszer ami alapján az S-boxokat szerkesztették $\rightarrow$ az a vélemény, hogy ha ez nyilvános lenne, akkor könnyebb lenne a kriptoanalízis.
- érvényesöl a lavina effektus,
- a biztonság fokozása: előbb tömörítik a nyíltszöveget utána titkosítják,

3DES, triple DES

- a feltöréshez szükséges idő: 2^{56}, ami komoly veszélyt jelent $\rightarrow$ ma már nem használjuk, helyette a $3DES$-t használjuk, amely NIST standard.
- 3DES:
  - a biztonság megerősítése anélkül, hogy a DES belső szerkezetén változtatnánk,
  - háromszor lassúbb, mint a DES,
  - a kulcsméret: $3 \times 56 = 168$ bit lesz,
  - legyen $E : K \times M \rightarrow M$ a DES blokktitkosító, ekkor $3E : K^3 \times M \rightarrow M$, ahol

$$3E((key_1, key_2, key_3), m) = E(key_1, D(key_2, E(key_3, m))),$$

  - hardver implementációnál fontos, hogy $key_1 = key_2 = key_3$, mert ekkor a DES-t kapom,
  - az összes kulcs kipróbálásának módszerével a feltőréshez szükséges idő: 2^{168}, de létezik 2^{118} nagyságrendű feltörési algoritmus is,

2DES, double DES

- a $2DES$ nem növeli a titkosító biztonságát,
- $2DES : K^2 \times M \to M$, ahol $2E((key_1, key_2), m) = E(key_1, E(key_2, m))$,
- meet in the middle típusú támadás alkalmazható, amely gyorsabb, mint az összes kulcs kipróbálásának módszere:
  - a cél megtalálni key_1, key_2-t, tudva, hogy m rejtjelezett értéke c, amikor is fennáll: $E(key_1, E(key_2, m)) = c$,
  - a megoldandó egyenlet ekvivalens: $E(key_2, m) = D(key_1, c)$,
  - minden key lehetséges értékére meghatározzák $E(key, m)$-t, és a kapott értékeket rendezik,
  - ellenőrzik, hogy melyik key-re kapják meg $D(key_1, c)$-t; ahol találat van, ott meghatározzák a key_1, key_2-t.
- feltöréshez szükséges idő = a rendezéshez + kereséshez szükséges idő: $2^{56} \cdot log(2^{56}) + 2^{56} \cdot log(2^{56}) < 2^{63}$,
- feltöréshez szükséges tárhely: 2^{56}.

DESX

- legyen $E : K \times M \to M$ a DES blokktitkosító, ekkor $EX : K^3 \times M \to M$

$$EX((key_1, key_2, key_3), m) = key_1 \oplus E(key_2, key_3 \oplus m))),$$

- nem standardizált, de kivédi az összes kulcs kipróbálásának módszerét,
- ha csak az egyszer alkalmazom a $\oplus$ műveletet, azaz a $key_1 \oplus E(key_2, m)$ vagy $E(key_2, m \oplus key_1)$ konstrukciók nem jelentenek nagyobb biztonságot mint a DES,

TEA (Tiny Encryption Algorithm)

- 1994-ben publikálta a Cambridge Egyetem két tanára, Roger Needham és David Wheeler
- tervezésekor feladtak egy keveset a biztonságból, hogy a blokktitkosító könnyen implementálható, hatékony legyen
- Feistel típusú, a titkosító és visszafejtő algoritmusok különböznek
- könnyen implementálható hardver-re, illetve softver-re
- ha két üzenetet olyan kulcsokkal titkosítunk, amelyek kapcsolódnak egymáshoz, akkor kivitelezhető a "related key" típusú támadás
- az XTEA egy későbbi változat, amely kiküszöböli a "related key" támadást

TEA, specifikáció

- az ajánlott körök száma: 64, a minimális körszám: 32,
- 128 bites kulccsal dolgozik, amit négy 32 bites blokkra oszt,
- minden matematikai műveletet $\pmod{2^{32}}$-ben végez
- a blokk mérete 64 bit, amelyet két 32 bites részre oszt: L (Left), illetve R (Right) részek,
- az L és R 8 bájtos szekvenciákat 256-os számrendszerbeli számjegyeknek tekinti, amelyekből létrehoz egy tízes számrendszerbeli számot, amivel a további számításokat végzi
- alkalmaz egy konstanst: `delta = 0x9e3779b9 = 2654435769`, azaz `delta` a $2^{32}/phi$ osztási egész része, ahol $phi = 1.6180339887$ az aranyarány

Blowfish

- Bruce Schneier szerkesztette 1993-ban,
- szabadon felhasználható,
- Feistel típusú,
- a körök száma: 16
- a kulcsméret változó: 32 bittől 448 bitig,
- a blokk mérete 64 bit,
- a körkulcsok meghatározásához és az S-boxok inicializálása során felhasználja a π tizedesjegyeit,
- nem találtak ellene hatékony támadást,
- bonyolult a kulcsfeldolgozása, ezért ha gyakran szükséges egy új kulcs kigenerálása, akkor nem ajánlott a használata

Twofish, bcrypt

A Blowfish egyik alternatív változata a Twofish:

- az AES verseny kiválasztottja volt, még 5 titkosítóval együtt,
- lassúbb volt az AES-128-nál, de gyorsabb az AES-256-nál,
- nincs levédve, bárki szabadon használhatja,
- egyike azon néhány titkosítónak, amelyeket az OpenPGP szabvány (RFC 4880) tartalmaz
- a PGP (Pretty Good Privacy) az emailek titkosítására használja,
- alkalmazza a kulcs fehérítést,

A bcrypt-et a Blowfish titkosító alapján szerkesztették 1999-ben,

- jelszavak tárolására használják több Unix alapú operációs rendszerben alapértelmezett
- egy véletlenszerű értéket használ, a *salt*-ot, szivárvány táblán alapuló támadással szemben

Blokktitkosítók, támadási módok

- differential cryptoanalysis (diferenciális kriptoanalízis)
  - sok titkos/rejtjelezett pár ismeretében gyakran az összes kulcs kipróbálásának módszerénél hatékonyabb feltörés végezhető,
- side-channel típusú támadás
  - az implementáció adta gyengeségeket használják ki,
  - meghatározzák a titkosítás/visszafejtés időigényét,
  - meghatározzák az áramfogyasztást nagyságát.
- számos egyéb támadási forma létezik, amelyek kivédésére a legjobb módszer, ha nem saját implementációkat használunk, hanem a nyílt forrás-kódú könyvtárakban található implementációkkal dolgozunk, pl.
  - *OpenSSL*,
  - *Crypto++*,
  - *Python/PyCryptodome*

AES (Advanced Encryption Standard)

- Daemen és Rijmen, belga kriptográfusok tervezték 1997-ben
- az 1990-ben meghírdetett nyilvános pályázat győztese
- valószínűleg nincs benne "backdoor", mert a győztes pályázat elbírálása teljesen nyilvánosan történt
- 2001-ben fogadta el a NIST standard blokktitkosítónak
- kulcsmérete: 128, 192, 256 bit, blokkmérete: 128 bit,
- szakvélemények szerint a 256 bites kulcsméret biztonsága *örök* időkre szól
- nem használja a Feistel-sémát, viszont hasonlóan a DES-hez, iterációs szerkezetű, több körből áll a titkosítás, amelyekhez körkulcsokat generál:
  - 128 bites kulcs esetén a körök száma 10
  - 192 bites kulcs esetén a körök száma 12
  - 256 bites kulcs esetén a körök száma 14
- minden körben a kulcsokon egy keverést alkalmaz, és a bemeneten, helyettesítést, permutációt és lineáris transzformációt is végrehajt
- nem találtak sikeres támadást ellen, a mai napig az implementációkból adódó gyengeségek okozzák a biztonsági problémákat

AES (Advanced Encryption Standard)

- a 128 bites, azaz a 16 bájtos $S_1 S_2 \dots S_{16}$ bemenetet egy 4×4-es mátrix alakban dolgozza fel:

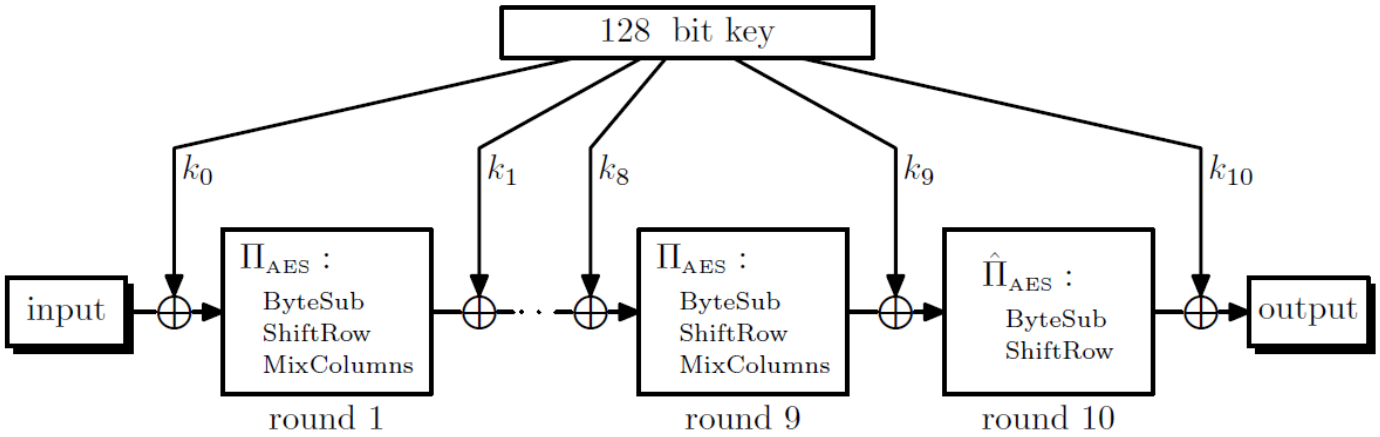
$$\begin{array}{cccc} S_1 & S_5 & S_9 & S_{13} \\ S_2 & S_6 & S_{10} & S_{14} \\ S_3 & S_7 & S_{11} & S_{15} \\ S_4 & S_8 & s_{12} & S_{16} \end{array}$$

- minden S_i bájt megfeleltethető egy $n = 7$ fokszámú polinomnak, ahol az együtthatók értéke 0 vagy 1, ezért
  - minden bájtot a $GF(2^8) = GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$ test elemeként kezel, ahol a $b_7 b_6 \dots b_1 b_0$ bájt megfelel a
    $$b_7 \cdot x^7 + \dots + b_1 \cdot x + b_0 \text{ polinomnak,}$$
  - a műveletvégzések után az eredménynek meghatározzák a $(\text{mod } x^8 + x^4 + x^3 + x + 1)$ szerinti osztási maradékát.
- Példa:
  $$x^6 + x^3 + x\text{-nek megfelelő bináris alak: } 0100\ 1010.$$

AES (Advanced Encryption Standard)

- az összeadás, kivonás, szorzás, osztás tehát a $GF(2^8)$ feletti véges testben történik,
- a műveleteket a szabályos polinomok feletti műveleti szabályok szerint kell végezni, ahol az együtthatók esetében $\pmod 2$ kell számolni,
- ha egy művelet elvégzése után nagyobb kitevőt kapunk mint x^7, akkor meg kell határozni az eredmény $m(x)$ szerinti osztási maradékát, ahol $m(x) = x^8 + x^4 + x^3 + x + 1$ egy 8-ad fokú irreducibilis polinom,
- n-ed fokú irreducibilis polinom: olyan polinom, amely nem bontható fel két n-nél kisebb fokú polinom szorzatára

AES-128 titkosítás



kép forrása: *Boneh and Shoup/DES*

- ByteSub: egy fix permutáció, nem lineáris művelet, az S-box alkalmazását jelenti, ahol az S-box bemenetének bitjeit egy polinom együtthatóinak tekinti, amelyen multiplikatív inverzet számol a $GF(2^8)$ véges test szerint, előre kiszámítva, táblázatban tárolva a multiplikatív inverzeket,
- ShiftRow: ciklikus biteltolást, lineáris műveletet végez,
- MixColumns: a mátrix elemein szorzásokat, lineáris átalakításokat végez a $GF(2^8)$ véges test szerint,
- a lineáris és affin átalakítások lavina effektust eredményeznek.

AES-128 titkosítás

- ByteSub egy nem lineáris helyettesítést végző művelet, a 4×4-es mátrix bájtjaira alkalmazza az S-boxot,
- ShiftRows egy lineáris keverő művelet, a 4×4-es mátrixot bájtjaira alkalmazza a következő ciklikus eltolásokat:

s_{11}	s_{12}	s_{13}	s_{14}	$>>$	s_{11}	s_{12}	s_{13}	s_{14}
s_{21}	s_{22}	s_{23}	s_{24}		s_{22}	s_{23}	s_{24}	s_{21}
s_{31}	s_{32}	s_{33}	s_{34}		s_{33}	s_{34}	s_{31}	s_{32}
s_{41}	s_{42}	s_{43}	s_{44}		s_{44}	s_{41}	s_{42}	s_{43}

- MixColumns egy lineáris keverő művelet, a 4×4-es mátrix bájtjaira alkalmazza a következő átalakításokat:

s_{11}	s_{12}	s_{13}	s_{14}	$>>$	ns_{11}	ns_{12}	ns_{13}	ns_{14}
s_{21}	s_{22}	s_{23}	s_{24}		ns_{21}	ns_{22}	ns_{23}	ns_{24}
s_{31}	s_{32}	s_{33}	s_{34}		ns_{31}	ns_{32}	ns_{33}	ns_{34}
s_{41}	s_{42}	s_{43}	s_{44}		ns_{41}	ns_{42}	ns_{43}	ns_{44}

ahol

$$
\begin{aligned}
ns_{1i} &= (0\text{x}02 \bullet s_{1i}) \oplus (0\text{x}03 \bullet s_{2i}) \oplus s_{3i} \oplus s_{4i} \\
ns_{2i} &= s_{1i} \oplus (0\text{x}02 \bullet s_{2i}) \oplus (0\text{x}03 \bullet s_{3i}) \oplus s_{4i} \\
ns_{3i} &= s_{1i} \oplus s_{2i} \oplus (0\text{x}02 \bullet s_{3i}) \oplus (0\text{x}03 \bullet s_{4i}) \\
ns_{4i} &= (0\text{x}03 \bullet s_{1i}) \oplus s_{2i} \oplus s_{3i} \oplus (0\text{x}02 \bullet s_{4i}).
\end{aligned}
$$

- a $\bullet$ művelet szorzást jelent $\pmod{x^8 + x^4 + x^3 + x + 1}$ szerint a $GF(2^8)$ véges testben.

AES-128 titkosítás, S-box

- az S-box bemenetének bitjeit egy polinom együtthatóinak tekintjük, és meghatározzuk a polinom multiplikatív inverzét $\pmod{x^8 + x^4 + x^3 + x + 1}$ szerint a $GF(2^8)$ véges testben, jelöljük ezt: $b = (b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0)$-vel
- ezután egy affin transzformációt alkalmazunk, megkapva az S-box kimeneti bitjeit: $nb = (nb_7\ nb_6\ nb_5\ nb_4\ nb_3\ nb_2\ nb_1\ nb_0)$
  - minden $i = 0, \ldots, 7$-re

    $nb_i = b_i \oplus b_{i+4 \pmod 8} \oplus b_{i+5 \pmod 8} \oplus b_{i+6 \pmod 8} \oplus b_{i+7 \pmod 8} \oplus c_i$,

    ahol c konstans:

    $$c = (c_7\ c_6\ c_5\ c_4\ c_3\ c_2\ c_1\ c_0) = (0110\ 0011) = 0\text{x}63.$$

AES-128 titkosítás, S-box példa

Határozzuk meg az S-box 0x4a $= (0100\ 1010) = x^6 + x^3 + x$-ra alkalmazott értékét.

- $x^6 + x^3 + x$ a multiplikatív inverze: $b = x^7 + x^5 + x^3 + x + 1 = (1010\ 1011)$ lesz, mert $(x^6 + x^3 + x) \cdot (x^7 + x^5 + x^3 + x + 1) = 1 \pmod{x^8 + x^4 + x^3 + x + 1}$,
- a keresett érték

$$nb = (1101\ 0110) = \text{0xd6}, \text{ mert:}$$

- $nb_0 = b_0 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus c_0 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0$
- $nb_1 = b_1 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_0 \oplus c_1 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$
- $nb_2 = b_2 \oplus b_6 \oplus b_7 \oplus b_0 \oplus b_1 \oplus c_2 = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1$
- $nb_3 = b_3 \oplus b_7 \oplus b_0 \oplus b_1 \oplus b_2 \oplus c_3 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 0$
- $nb_4 = b_4 \oplus b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus c_4 = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$
- $nb_5 = b_5 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_4 \oplus c_5 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 0$
- $nb_6 = b_6 \oplus b_2 \oplus b_3 \oplus b_4 \oplus b_5 \oplus c_6 = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1$
- $nb_7 = b_7 \oplus b_3 \oplus b_4 \oplus b_5 \oplus b_6 \oplus c_7 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 1$

AES-128 titkosítás, S-box

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

AES visszafejtés

- a blokktitkosítókra jellemzően a körkulcsokat fordított sorrendbe alkalmazza,
- az alkalmazott függvényeket fordított sorrendben veszi a titkosításnál alkalmazott sorrendhez képest
- a ByteSub, ShiftRows, MixColumns, S-box függvények inverzeit használja $\rightarrow$ a titkosítás nem ugyanaz, mint a visszafejtés
  - titkosítási sorrend: ByteSub, ShiftRows, MixColumns,
  - visszafejtési sorrend: InvMixColumns, InvShiftRows, InvByteSub
- meg kell külön írni a titkosító és visszafejtő függvényt $\rightarrow$ hátrány, de lehet ezt optimalizálni

AES-128 kulcsgenerálás

A $128 = 16 \cdot 8$ bites kulcsot oszloponként, egy 4×4-es kétdimenziós mátrix formába írjuk $\rightarrow RK_0$, majd minden $i = 1, \ldots 10$-re:

- w_0, w_1, w_2, w_3-al jelölve RK_{i-1} oszlopait, meghatározzuk az nw_0, nw_1, nw_2, nw_3 *szavakat*:
  - $nw_0 = temp \oplus w_0$, ahol *temp*:
    - $rw = \text{RotWord}(w_3)$
    - $sw = \text{SubWord}(rw)$
    - $rcw = \text{Rcon}(i)$
    - $temp = sw \oplus rcw$
  - $nw_1 = nw_0 \oplus w_1$
  - $nw_2 = nw_1 \oplus w_2$
  - $nw_3 = nw_2 \oplus w_3$
- összefűzzük a kapott szavakat: $nw_0||nw_1||nw_2||nw_3 \rightarrow$ 128 bites értékből álló bitszekvencia
- oszloponként, egy 4×4-es kétdimenziós mátrix formába írjuk $\rightarrow RK_i$

AES-128 kulcsgenerálás, RotWord, SubWord, Rcon

- RotWord, balra történő ciklikus eltolást hajt végre:
  $RotWord(a_0\ a_1\ a_2\ a_3) = (a_1\ a_2\ a_3\ a_0)$
- SubWord, alkalmazza bájtonként az S-boxot. Ha az átalakítandó bájt például a $4a$, akkor az S-box táblázat 4-dik sora és a-dik oszlopának kereszteződésénél található bájt lesz az új érték: $d6$.
- Rcon 4 kimeneti bájtja közül a legjobboldalibb bájt értéke egyenlő x^{i-1} (mod $x^8 + x^4 + x^3 + x + 1$) hatványértékével, a további három bájt értéke, azaz a 3 legbaloldalibb bájt 0x00 lesz.

i	1	2	3	4	5	6	7	8	9	10
$Rcon(i)$	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80	0x1b	0x36

$$0x1b = 0001\ 1011 = (x^4 + x^3 + x + 1) = x \cdot x^7 = x^8 \pmod{x^8 + x^4 + x^3 + x + 1}$$