Kriptográfia

Nyolcadik előadás
Blokktitkosítók működési módjai, folyamtitkosítók

Dr. Németh L. Zoltán
SZTE, Számítástudomány Alapjai Tanszék
2008 ősz

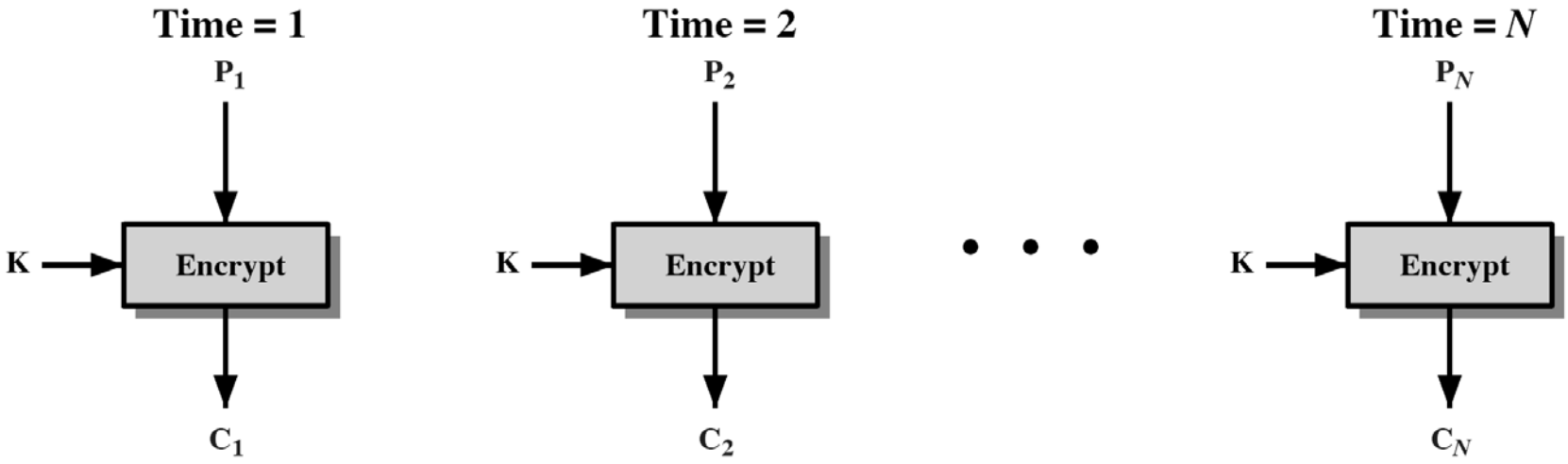
Blokktitkosítók működési módjai

- a blokktitkosítók **rögzített méretű** adatmennyiséget (egy blokkot) titkosítanak egyszerre
  - pl. a DES 64 bitet, az AES 128 bitet
- valamilyen úton **tetszőleges mennyiségű** adata titkosítása/megfejtése kell a gyakorlatban
- 1983: a NIST **4 működési módot (Modes of Operations)** vezetett be: **FIPS 81**-es szabvány
- később egy ötödik módot is definiáltak
- ezek közt mind **blokk** mind **folyam** módok vannak
- egymástól hatékonyságukban, biztonságukban, illetve hiba tűrésükben különböznek
- egy tetszőleges blokktitkosítóval való titkosítás lehetséges útjait igyekeznek lefedni melyekből az alkalmazás követelményekhez választhatunk

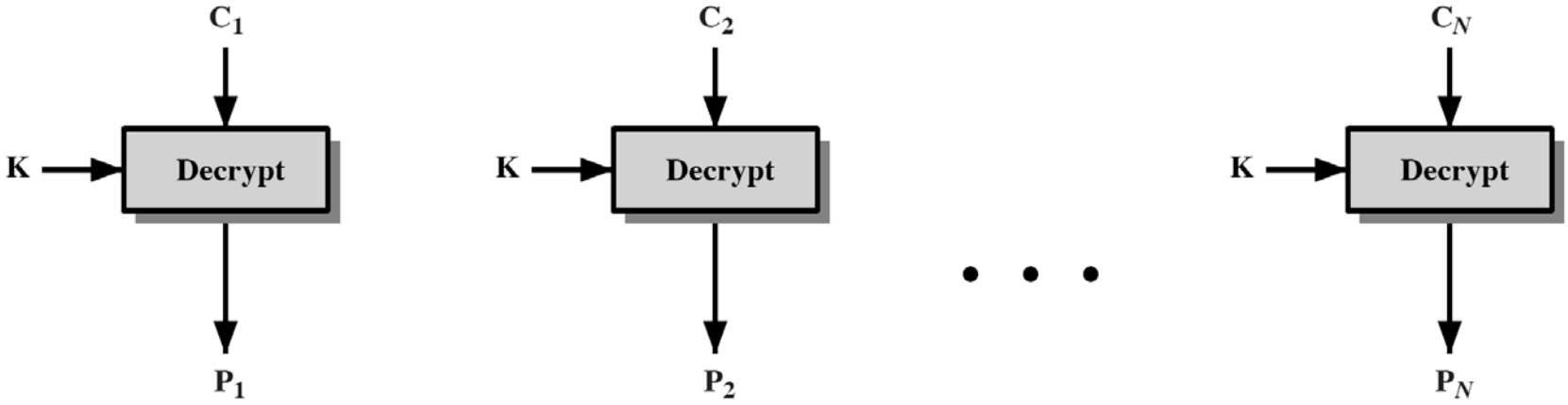
Elektronikus kódkönyv mód
Electronic Codebook Book (ECB)

- a lehető legegyszerűbb alkalmazás
- az üzenetet először blokkméret méretű blokkokra tördeljük, majd azokat titkosítjuk
- minden blokk értékét a titkosítás szerinti értékkel helyettesítjük, mintha egy gigantikus méretű kódkönyvet használnánk
- minden blokk a többi blokktól függetlenül kerül titkosításra, pl: $C_i = DES_K(P_i)$ ű
- a példákban a ***DES***-t használjuk, de vehetnénk más blokktitkosítót is
- használata: egyetlen titkos érték átvitelére

Elektronikus kódkönyv mód (ECB)



(a) Encryption



(b) Decryption

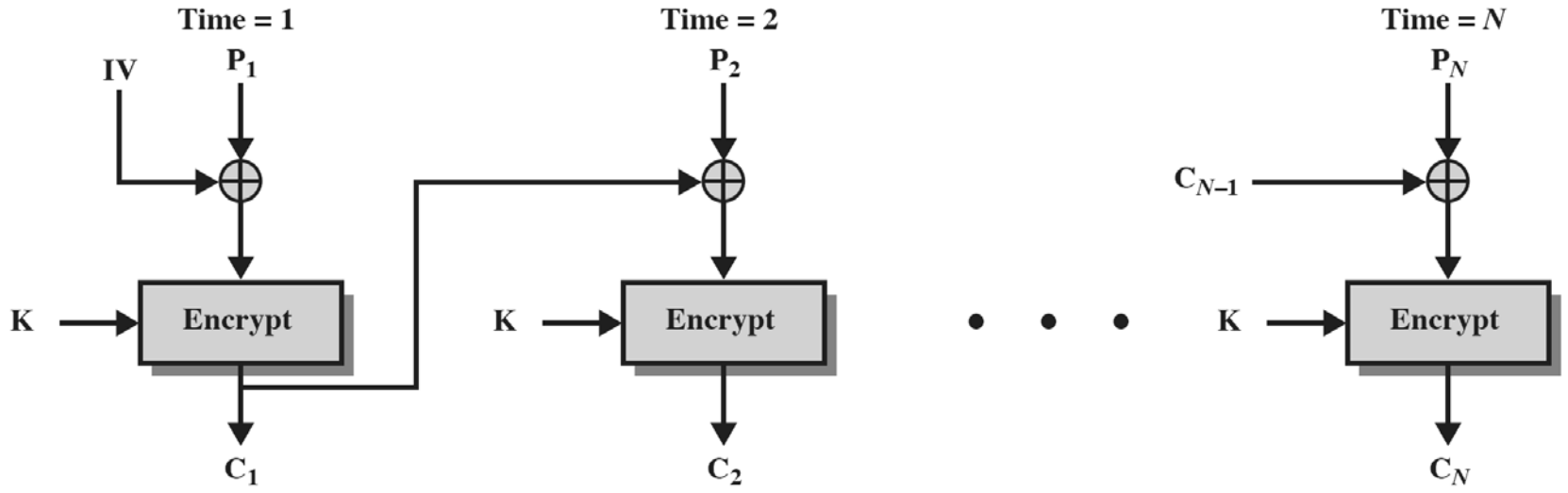
Az ECB hátrányai

- azért gyenge, mert a blokkok egymástól függetlenül kódolódnak
- a nyílt szöveg azonos blokkjai azonosak lesznek
- így a nyílt szöveg ismétlődő blokkjai ismétlődnek a titkosított szövegben is
  - ha az üzenet erősen strukturált, ez kihasználható
  - a támadó beszúrhat/helyettesíthet/átrendezhet blokkokat
  - ha az üzenet csak kis mértékben változik (pl. egy szerződésben csak az összeg) táblázat készíthető hozzá
- akkor használható, ha csak néhány blokkot kell titkosítani, pl. egy kulcsot

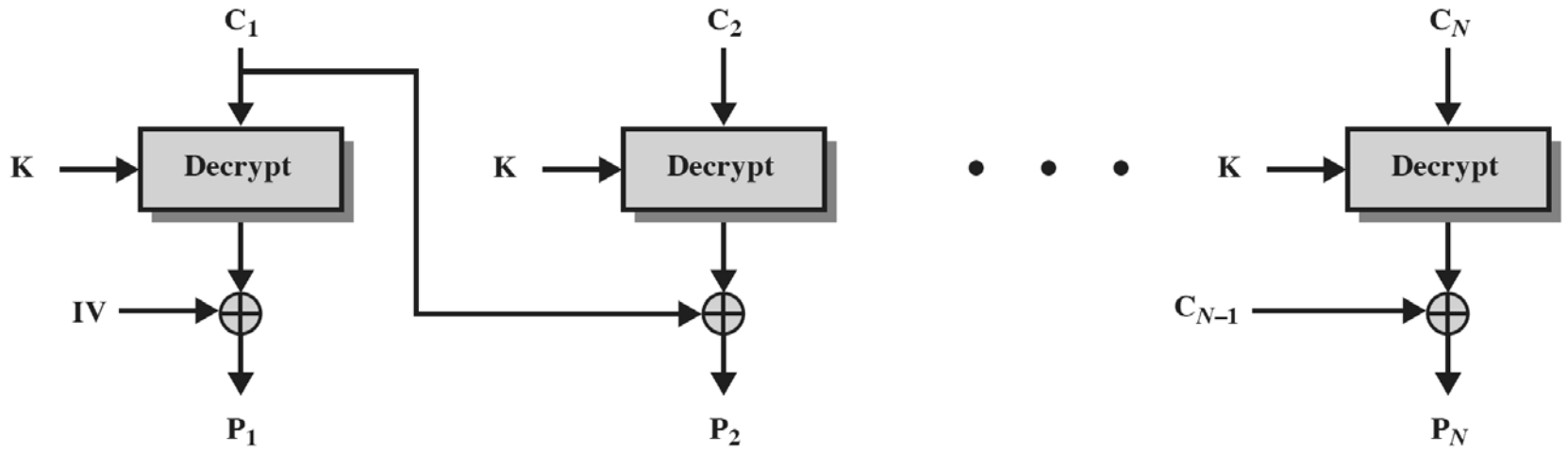
Rejtjeles blokkok láncolása Cipher Block Chaining (CBC)

- az aktuális blokk titkosításának eredményét felhasználja a következő blokk titkosításához is
- a nyílt szöveg és az előző rejtjeles blokk között egy XOR műveletet végzünk titkosítás előtt
- így két egyforma blokk nem ugyanúgy fog titkosítódni, mert az előző blokk más
- egy kezdő vektor (Initial Vector, IV)-vel kezdünk:
  - $C_0 = IV$
  - $C_i = DES_K(P_i \text{ XOR } C_{i-1}),\ i=1,\ 2,\ \ldots$
- használata: sok adat titkosítása, ha az összes adat előre rendelkezésre áll (pl. e-mail, FTP, web)

Cipher Block Chaining (CBC)



(a) Encryption



(b) Decryption

Üzenet helykitöltése (Message Padding)

- az üzenet végén lehet, hogy egy blokkhossznál rövidebb rész marad amit kezelni kell
- az üzenetet fel kell tölteni (ki kell egészíteni), hogy egész számú blokk legyen
  - a helyet kitölthetjük ismert nem adat értékekkel pl. 0-kkal
  - vagy a feltöltés hosszát beírhatjuk az utolsó bájtba
    - pl. [b1 b2 b3 0 0 0 0 5]
    - azt jelenti, hogy 3 bájt adatbájt és 5 bájt a feltöltés
  - esetenként szükségünk lehet, még egy extra blokk használatára, hogy mindig legyen feltöltés és így a visszaalakítás egyértelmű legyen
  - vannak bonyolultabb technikák, melyekkel ez elkerülhető

A CBC előnyei és korlátai

- a titkosított blokk minden előtte levő blokktól függ
- egy blokk megváltozása a teljes az összes további blokk titkosítását megváltoztatja
- a titkosítás nem, de a megfejtés párhuzamosítható
- kezdővektor kell (IV), melyet
  - ismernie kell a küldőnek és a címzettnek
  - de nem küldhető el titkosítás nélkül, mert a támadó azt meghamisítva a nyílt szöveg első blokkjának tetszőleges bitjeit invertálni tudja (akármit hamisíthat oda)
  - ezért IV vagy
    - előre rögzített érték legyen
    - vagy ECB módban titkosítva küldjük el előre

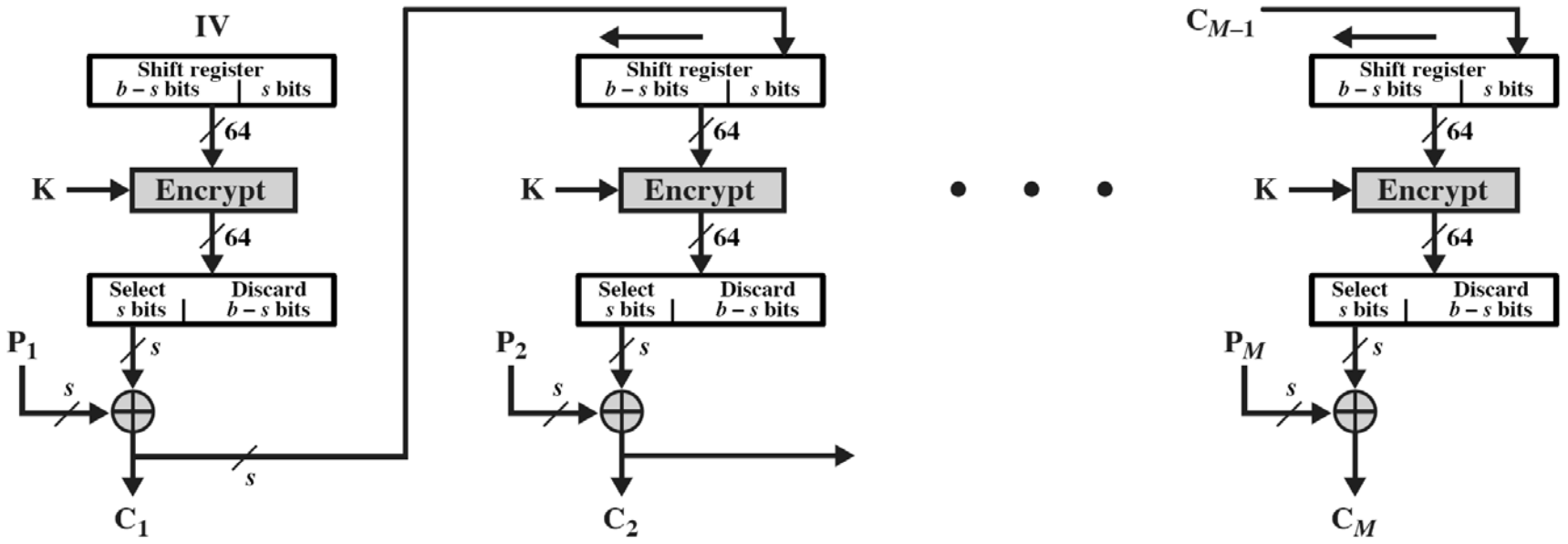
A titkos szöveg visszacsatolása Cipher FeedBack (CFB)

- ha az adatfolyam csak időnként bitekből/bájtokból áll (mint például egy terminál-hoszt kapcsolat) más titkosítási módszer kell, mert nem tarthatjuk vissza az információt
- a visszacsatolás általában a blokkméretnél kisebb mondjuk *s* bites egységekben történik (pl. **s**=8 bit)
- azaz s bitet titkosít egyszerre (= folyam mód)
- a kulcsot csak közvetve használjuk a titkosításhoz s álvéletlen bitet generálva, melyekkel XOR-olva titkosítunk
- az eredményt visszacsatoljuk a következő s álvéletlen bit előállításához
- a szabványokban s tetszőleges értéket felvehet (1,8, 64, 128)
  - jelölése: CFB-1, CFB-8, CFB-64, CFB-128 stb.
  - **ha s = blokkhossz, akkor**

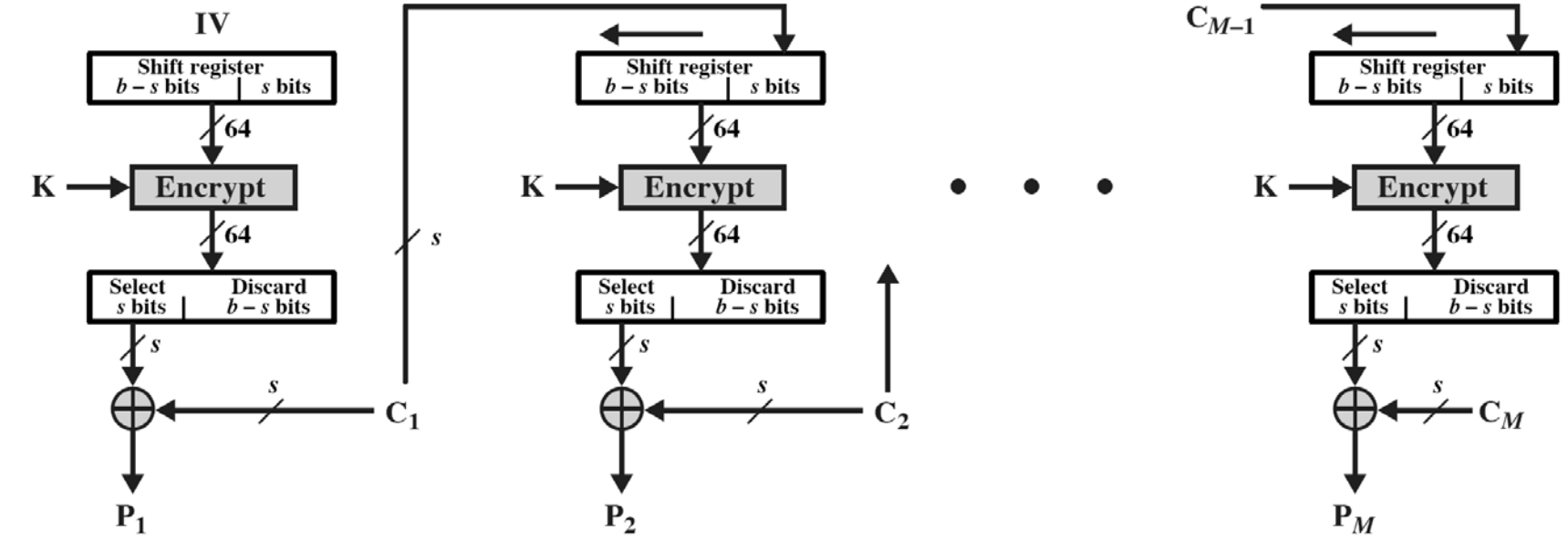
    $C_0 = IV$

    $C_i = P_i \ XOR \ DES_K(C_{i-1})$
- használata: általános célú adatfolyamok tikosítása, hitelesítés

Cipher FeedBack (CFB)



(a) Encryption



(b) Decryption

A CFB előnyei és korlátai

- legelterjetebb mód, amivel blokktitkosítót folyamtitkosítóként alkalmazhatunk
- csak a titkosító algoritmust használja a megfejtőt nem, így csak szimmetrikus kulcsú módszerekkel használható (nyilvános kulcsúakkal nem !)
- lassabb adatmozgást biztosít, mint maga a titkosító algoritmus blokkhossz/s menet kell egy blokknyi adat titkosításához
- érzékenyebb az adatátviteli hibákra
  - egy bithiba az egész következő blokkot olvashatatlanná teszi (a lavinahatás miatt), és még további blokkokat is
  - de miután a hibás bit a shift regiszterből kilép visszaáll a rend => önszinkronizáló (self syncronizing) mód

A kimenet visszacsatolása Output FeedBack (OFB)

- az üzenetet szintén folyamként dolgozza fel
- de most a titkosító algoritmus kimenetét, és nem magát a titkosított szöveget csatoljuk vissza
- így a generált álvéletlen bitek függetlenek a nyílt szövegtől, előre kiszámíthatók
- ha **s** = blokkhossz

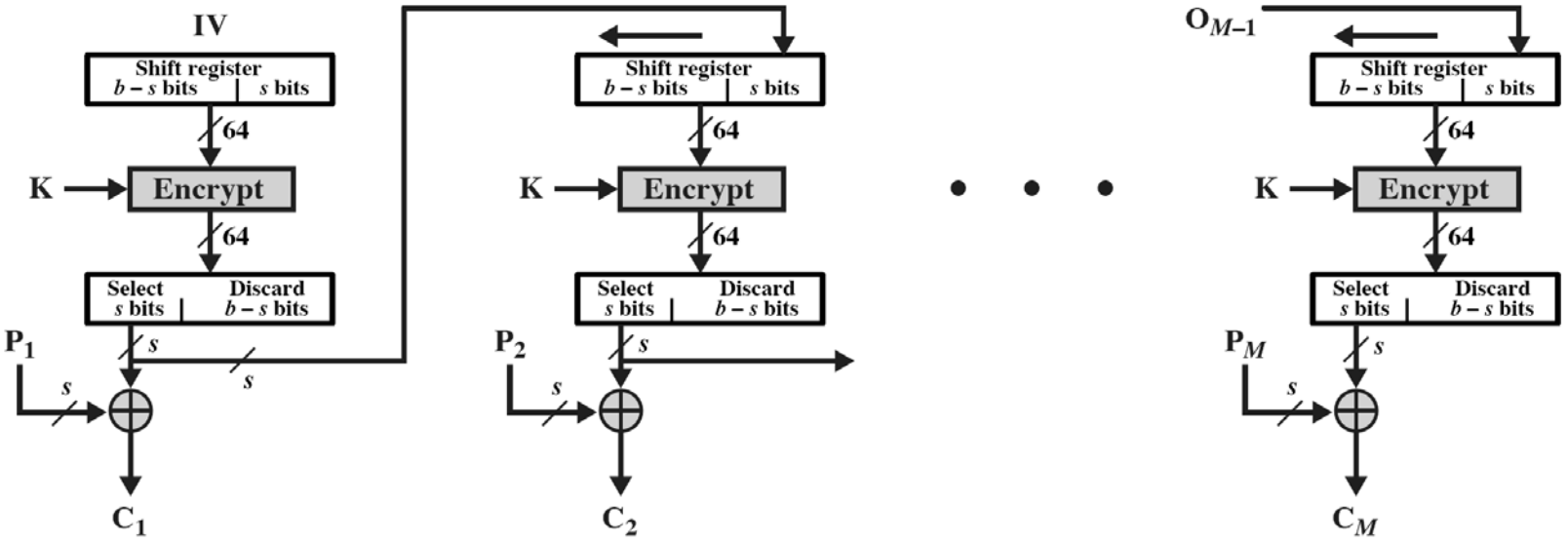
$$O_0 = IV$$

$$O_i = DES_K(O_{i-1})$$

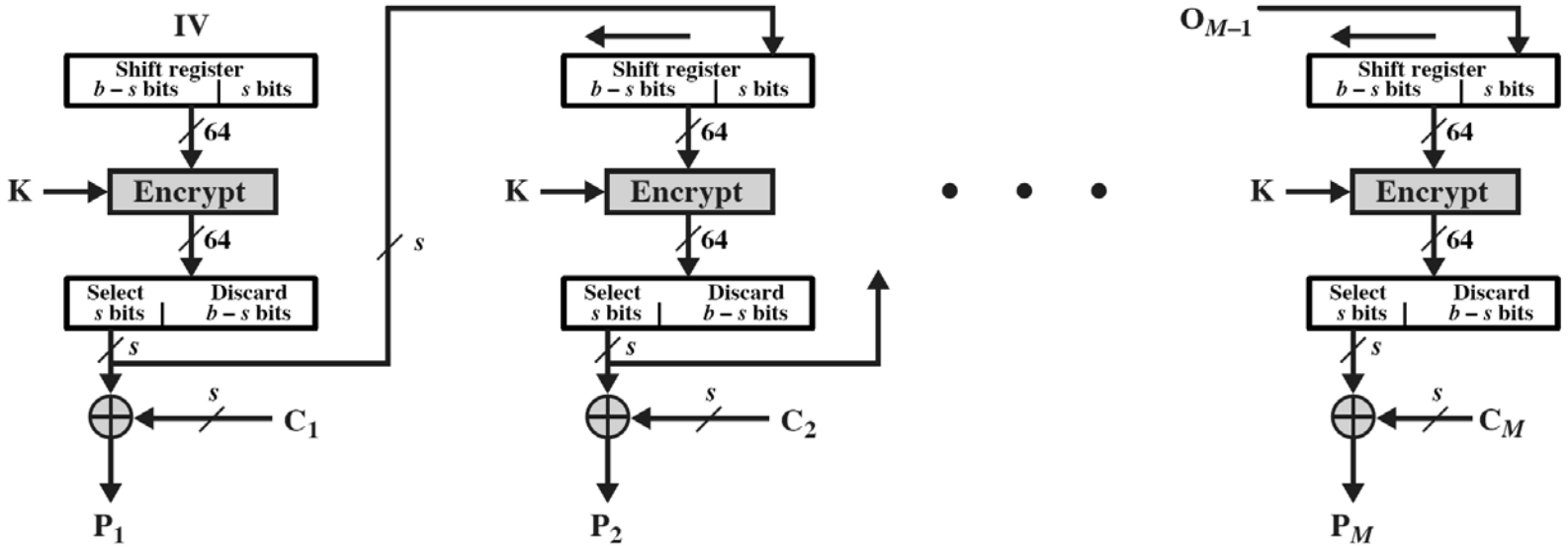
$$C_i = P_i \ XOR \ O_i$$

- használata: folyamtitkosítás zajos csatornán át

Output FeedBack (OFB)



(a) Encryption



(b) Decryption

Az OFB előnyei és korlátai

- itt is blokkhossz/***s*** menet kell egy blokknyi adat titkosításához, de az álvéletlen bitek előre kiszámíthatók, ha biztonságosan tudjuk tárolni amint az adat megvan mehet a titkosítás
- a bithibák nem terjednek szét, csak egy bit lesz rossz
- a blokkszinkron hibák viszont nem javíthatók, ha egy C_i elvész, ami utána jön megfejthetetlen
  -> szinkronizálás kell
- igazából a Vernam-titkosító egy variációja
  - így tilos ugyanazt a kulcsot és IV-t többször használni
- sérülékenyebb az üzenetcsatorna módosításaival szemben
- bebizonyították, hogy csak ***s*** = blokkméret választással lehet biztonságos (pl 3DES-OFB-64 or AES-OFB-128)
- mindkét oldalon visszacsatolás van
  => nem párhuzamosítható

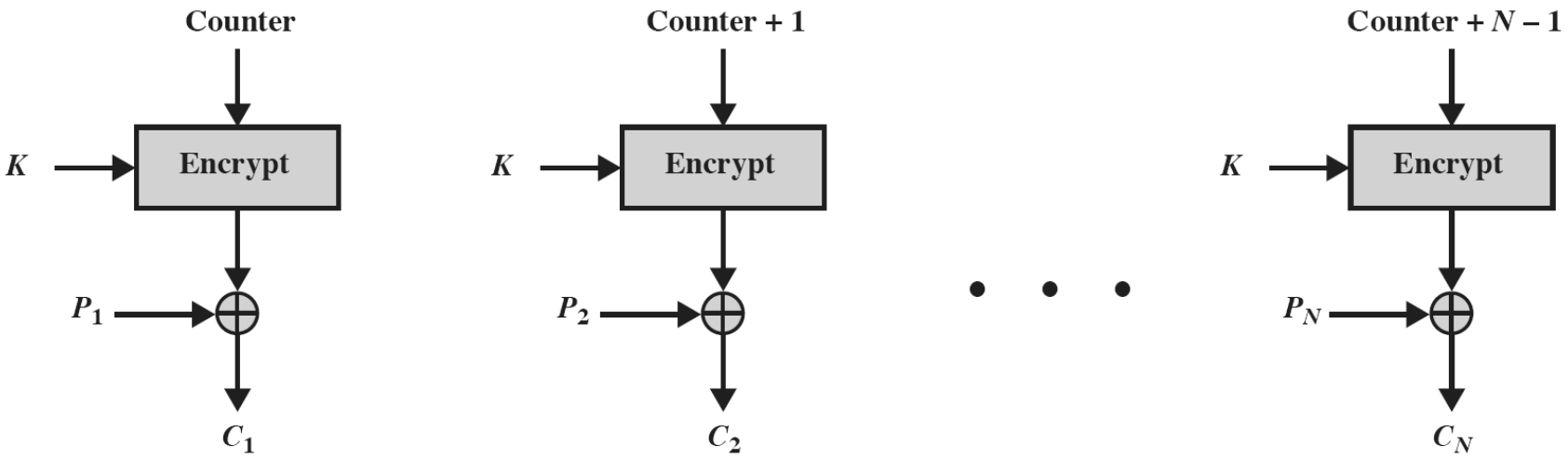
Számláló mód
Counter (CTR)

- ez egy “új” (ötödik) mód, bár már elég régen javasolták (1979)
- hasonlít az OFB módhoz, de **egy számláló értékét titkosítja**, nem pedig visszacsatolt értéket
- minden blokkhoz különböző értékpár kulcs & számláló kell (tilos újra használni)

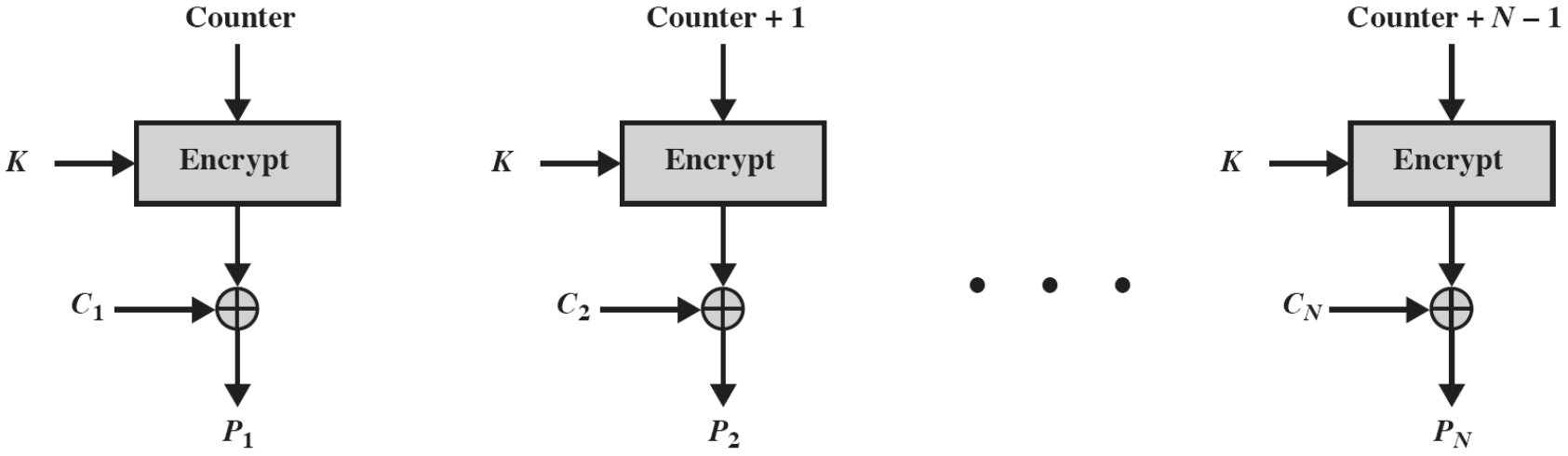
  $O_i = DES_K(i)$

  $C_i = P_i \ XOR \ O_i$
- használata: nagy sebességű hálózati titkosításnál, aszinkron adatátviteki mód, IPSec
- egy blokkhossz méretű számlálót használ, melynek értékei a nyílt szövegektől különbözők legyenek

Counter (CTR)



(a) Encryption



(b) Decryption

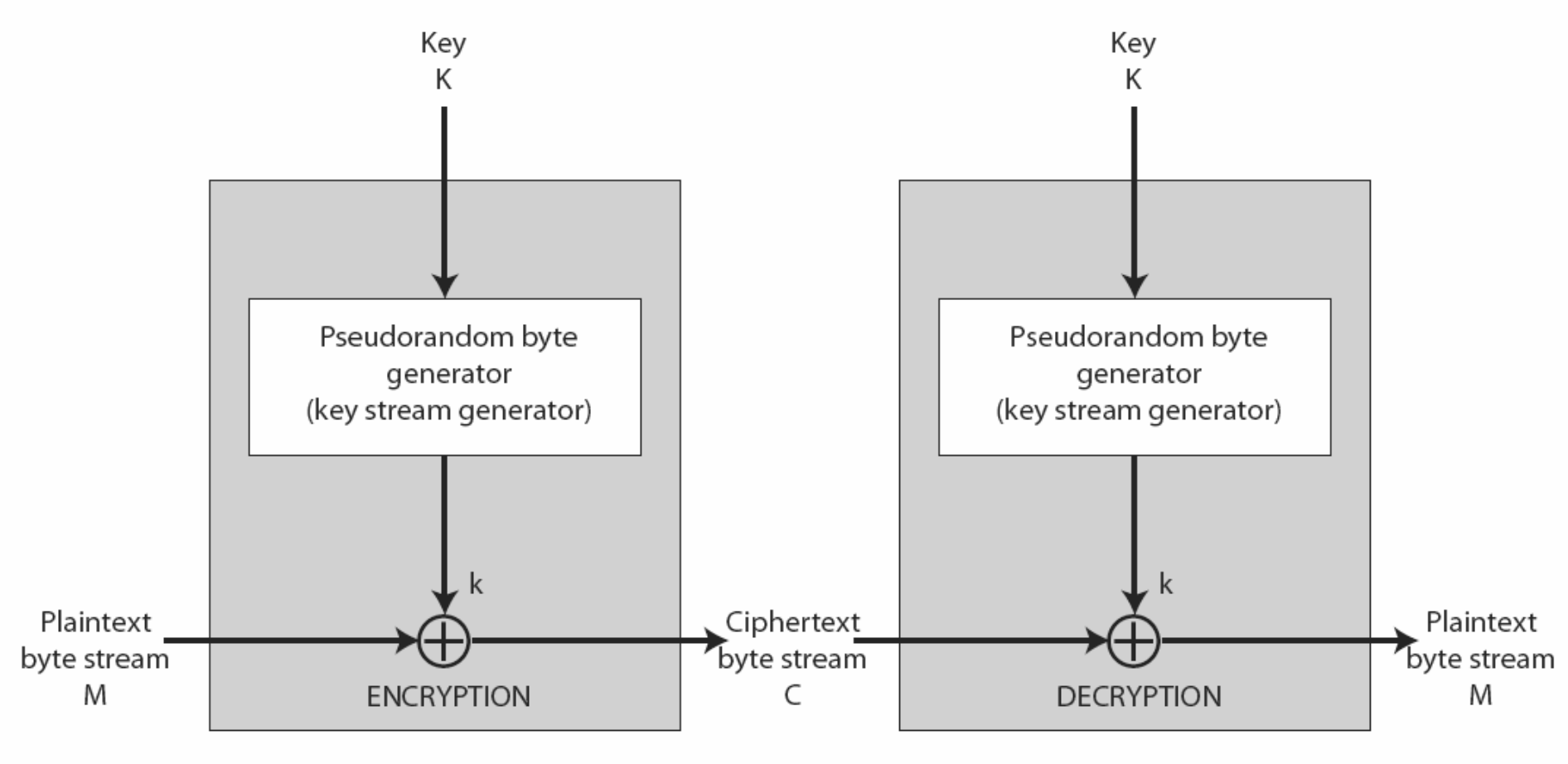
A CTR előnyei és korlátai

- egyszerűség: csak a titkosító algoritmust kell hozzá implementálni
- hatékonyság
  - párhuzamosítható mind h/w mind s/w szinten
  - előre generálható a ,,kulcsfolyam''
  - alkalmas sok adat gyors titkosításához
- közvetlenül is hozzáférhetünk az adatblokkokhoz (random access)
- bizonyíthatóan legalább olyan biztonságos mint a többi mód
- de biztosítani kell, hogy ugyanazt a kulcs-számláló párt sohase használjuk újra, különben feltörhető.

Folyamtitkosítók (Stream Chiphers)

- az üzenetet bitenként (bájtonként) dolgozza fel adatfolyamként
- ehhez a kulcsból egy álvéletlen (pseudo random) kulcsfolyamot (keystream) állít elő
- majd XOR műveletet végez vele bitenként
- mint a OTP, csak ott valódi és nem álvéletlen a kulcsfolyam
- a kulcsfolyam véletlensége teljesen megsemmisíti a nyílt szöveg statisztikai jellemzőit
  - $C_i = M_i \ XOR \ StreamKey_i$
- de ugyanazt a kulcsot tilos többször alkalmazni, különben feltörhető (nem úgy mint a blokktitkosítóknál.)

A folyamtitkosítók általános felépítése



A folyamtitkosítók tulajdonságai

- néhány tervezési elv:
  - a kulcsfolyam hosszú periódusú legyen hosszabb ismétlődések nélkül
  - állja ki a véletlenség statisztikai próbáit
    - ekkor a rejtjelezett szöveg is véletlennek látszó lesz
  - egy elég hosszú kulcstól függjön (pl. ma >= 128 bit)
  - magas lineáris kompexitású legyen
- az alkalmasan tervezett folyamtitkosító ugyanolyan biztonságos lehet, mint az azonos kulcshosszú blokktitkosító (csak brute force)
- de általában egyszerűbb (pár soros kód!) és gyorsabb

Az RC4

- tervezte Ron Rivest (RSA Security, 1987)
- az RSA üzleti titokként kezelte a kódját, de valaki elküldte a Cypherpunks lev. listára 1994-ben
- nagyon egyszerű, így rendkívül hatékony
- változtatható kulcsméretű, bájtonként dolgozik
- az egyik legelterjedtebb (SSL/TLS, WPA, WEP) folyamtitkosító
- a kulcsfolyamhoz a 0..255 számok (ál)véletlen permutációit használja
- a kulcsból előállít egy álvéletlen permutációt, majd
- az aktuális, permutáción kever még egyetet, majd generál belőle egy bájtot, amivel XOR-olva titkosítja az aktuális bájtot

RC4 előkészítési fázis

- kiindulás: **S** tömb a 0..255 számokkal
- melyet a kulcs alapján megkeverünk
- **`keylen`** a kulcs mérete bájrtokban (max 256)
- **S** adja majd meg a titkosító belső állapotát

```
for i = 0 to 255 do
    S[i] = i
    T[i] = K[i mod keylen])
j = 0
for i = 0 to 255 do
    j = (j + S[i] + T[i]) (mod 256)
    swap (S[i], S[j])  // csere
```

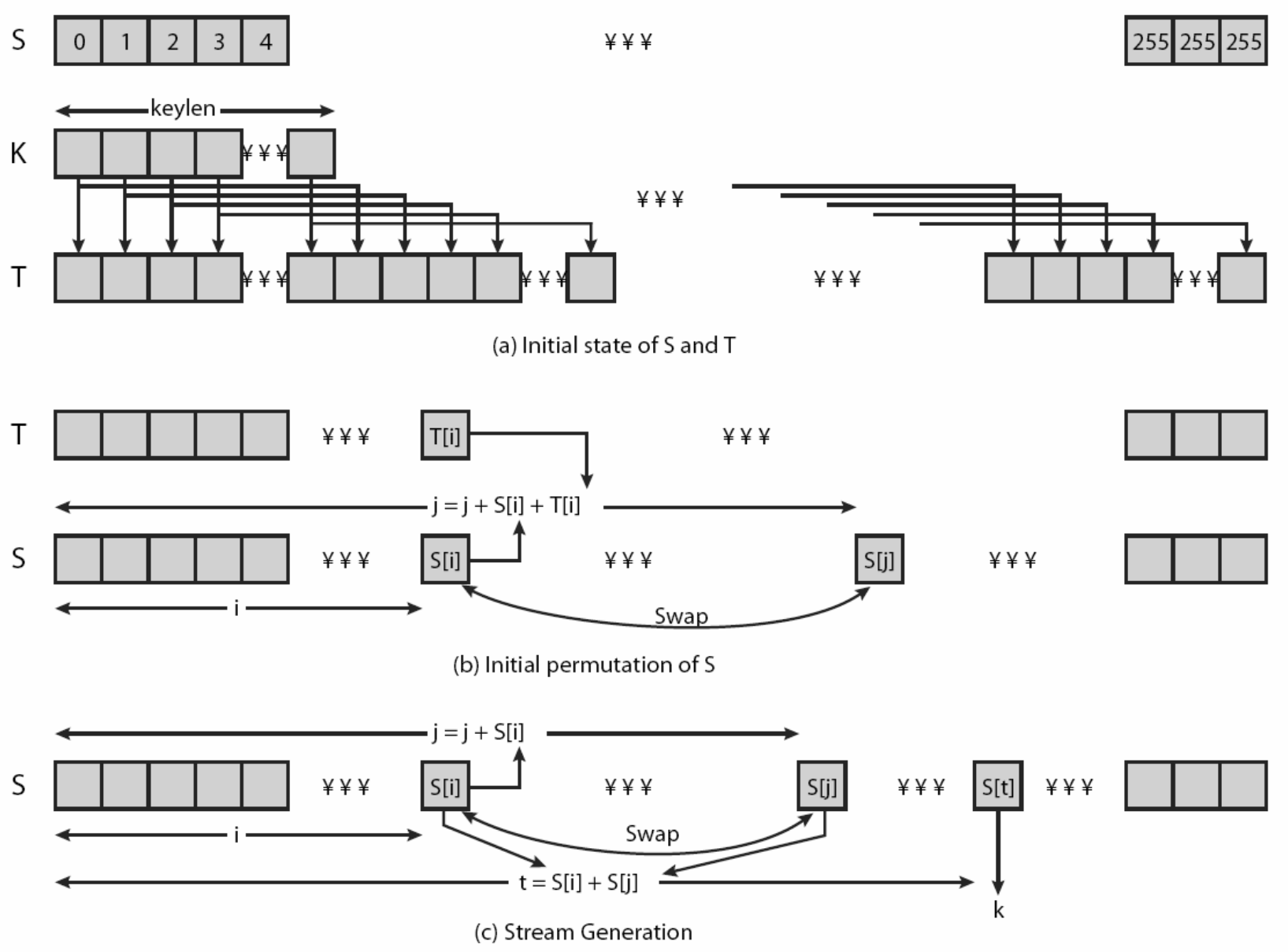
- **`keylen`** a kulcs mérete bájrtokban (max 256)
- **`K[i]`** a kulcs **`i.`** bájtja

RC4 titkosítás

- a titkosítás/megfejtés (ugyanaz a kulcsfolyam kell!) során tovább keverjük az aktuális permutációt
- a megcserélt pár értékeinek összege (***t***) jelöli ki az aktuális „kulcs bájtot” (**S[t]** -t) a táblázatból
- ami az üzenet következő bájtjával XOR-olva adja a titkosítást/megfejtést

```
i = j = 0
for each message byte M_i
   i = (i + 1) (mod 256)
   j = (j + S[i]) (mod 256)
   swap(S[i], S[j])
   t = (S[i] + S[j]) (mod 256)
   C_i = M_i XOR S[t]
```

RC4 áttekintés



Az RC4 biztonsága

- az eddigi kutatások alapján biztonságos az ismert támadásfajtákkal szemben (persze elég hosszú kulcs esetén, >= 128 bit)
  - van néhány gyakorlatban kivitelezhetetlen kriptoanalízisen alapuló támadás
- nagyon nem lineáris titkosító
- mivel RC4 folyamtitkosító, ezért tilos a kulcsot újra felhasználni
- ezzel van a fő probléma a WEP esetén,
- de ezt sérülékenységet a kulcsgenerálás gyengesége okozza és nem magát az RC4-et érinti

Felhasznált irodalom

- Virrasztó Tamás: Titkosítás és adatrejtés: Biztonságos kommunikáció és algoritmikus adatvédelem, NetAcademia Kft., Budapest, 2004. Online elérhető: http://www.netacademia.net/book.aspx?id=1#

- William Stallings: Cryptography and Network Security, 4th Edition, Prentice Hall, 2006. (Chapter 6)
- Lawrie Brown előadás fóliái (Chapter 6)