



HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK

Kriptográfiai kódolás – 1. rész

VIHIBB01 – Kódolás és IT biztonság (2020)

Dr. Buttyán Levente

CrySyS Lab, BME
buttyan@crysys.hu



Kriptográfia

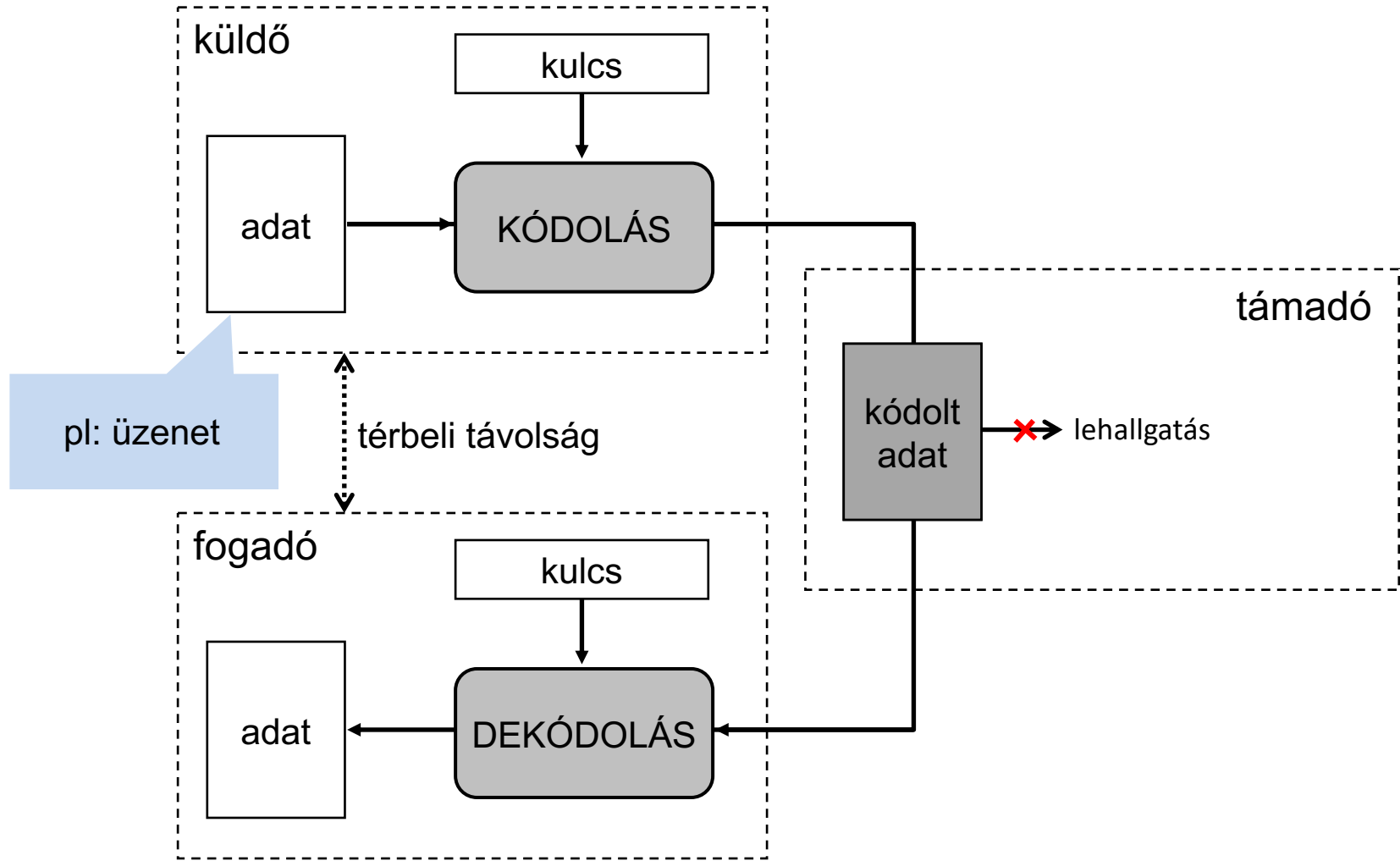
- Algoritmikus módszerek tárháza információbiztonsági szolgáltatások megvalósítására
 - rejtjelezés (titkosítás) ---» bizalmasság szolgáltatás
 - üzenet hitelesítő kód (MAC) ---» integritásvédelem és eredet hitelesítés
 - digitális aláírás ---» integritásvédelem, eredet hitelesítés és eredet letagadhatatlanság
- Sok olyan biztonsági probléma van, amit kriptográfiával lehet a legkényelmesebben megoldani, de nem minden probléma oldható meg pusztán kriptográfiai módszerekkel

Kriptográfia a gyakorlatban

- A kriptográfiai algoritmusok a programozók számára általában kriptográfiai függvénykönyvtárakban (crypto library) érhetők el különböző API-kon keresztül
 - példák: OpenSSL, NaCl, Java Cryptography Extension (JCE), PyCryptodome, ...
- A kripto könyvtárak és a kripto API-k a programozók számára ismert absztrakciókat használnak
 - példa: rejtjelező *objektum* melynek vannak *attribútumai* (pl. a rejtjelező kulcsa) és *függvényei/metódusai* (pl. egy `encrypt()` függvény)
- A kripto könyvtárak használata ezért nem bonyolult, ugyanakkor fontos, hogy a programozó tisztában legyen az API mögött implementált algoritmusok főbb tulajdonságaival

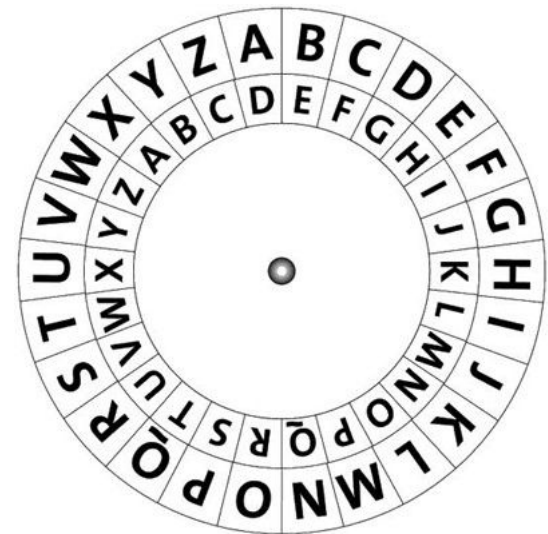
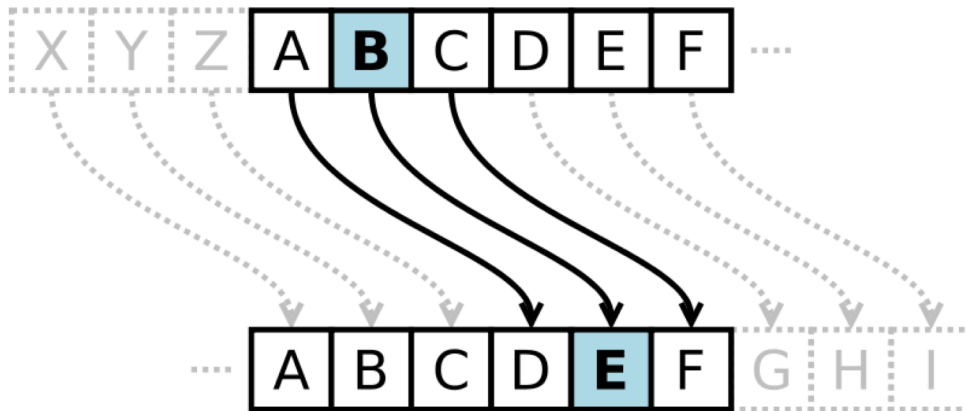
Rejtjelezés

A rejtjelezés alapmodellje



Történelmi példa: Caesar rejtjelező

- *Helyettesítéses (substitution) rejtjelező*: a nyílt üzenet betűi helyére kódbetűket helyettesít
- Caesar: minden nyílt betű helyére az ABC-ben tőle 3 pozícióval jobbra található betűt helyettesíti



kódolási példa: **BRUTUS** → **EUXWXV**

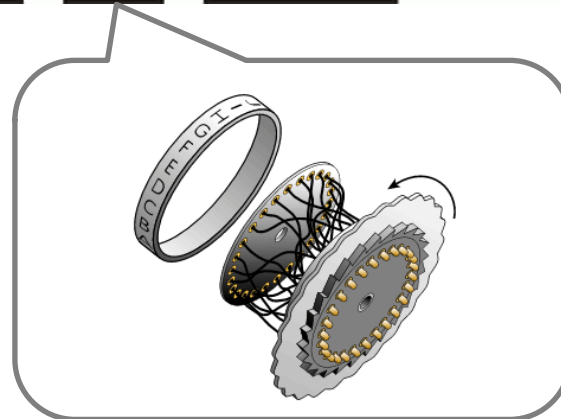
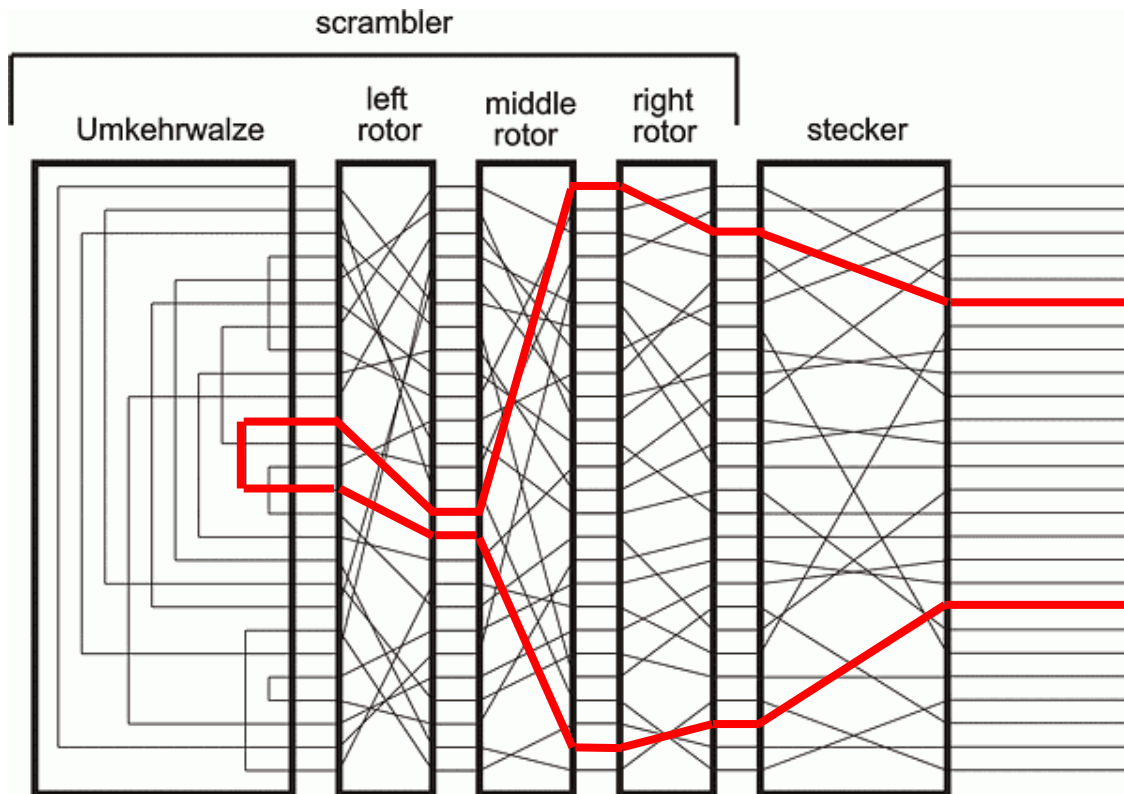
Történelmi példa: Enigma



- Az első elektro-mechanikus rejtjelező
- 1926-ban rendszeresítették a német hadseregben
- Intenzíven használták a második világháborúban



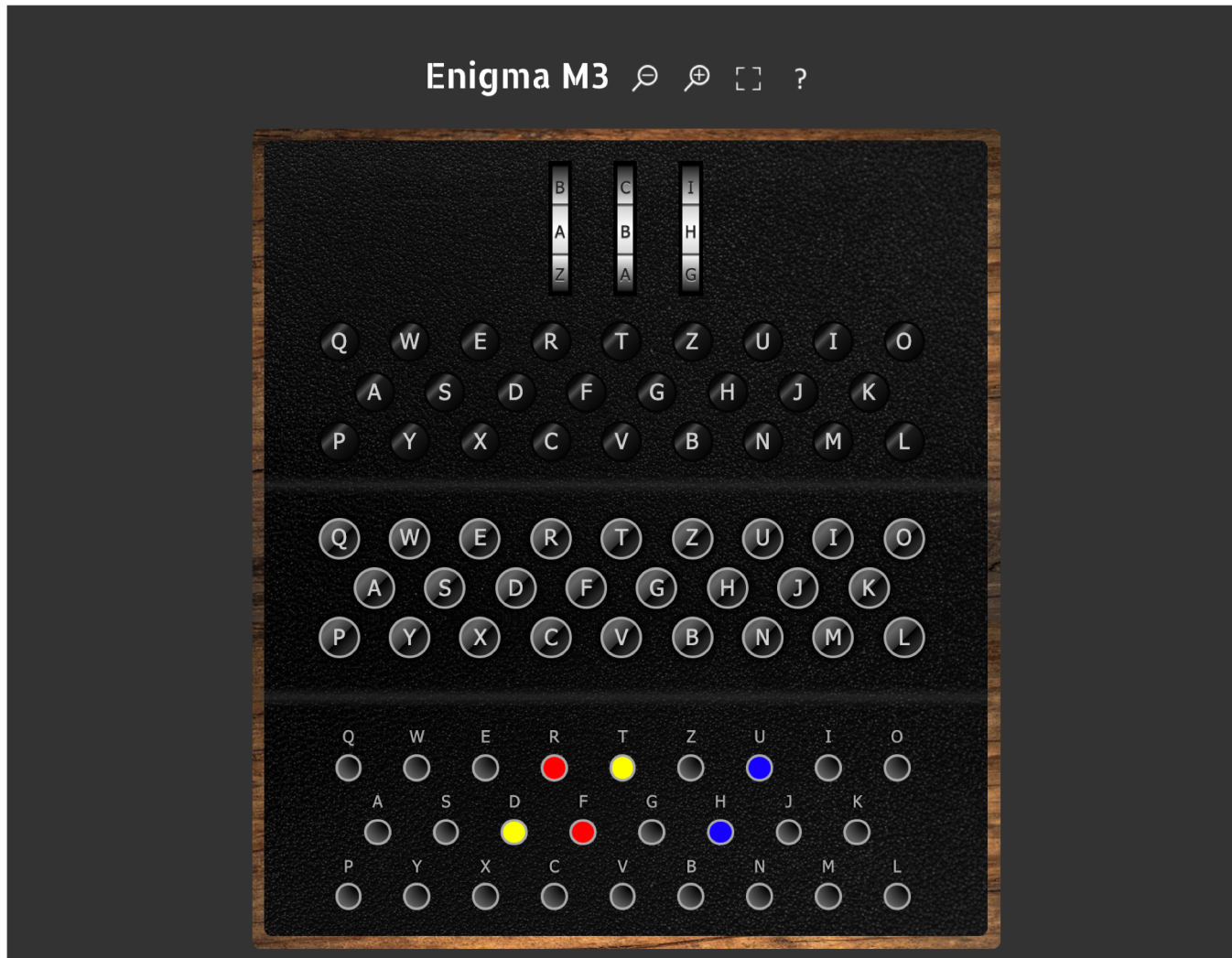
Az Enigma működése



Próbáld ki!



<https://www.101computing.net/enigma-machine-emulator/>



Az Enigma feltörése

"THE BEST BRITISH FILM OF THE YEAR"



THE INDEPENDENT

"AN INSTANT CLASSIC"



GAARDER

"A SUPERB THRILLER"



EMPIRE



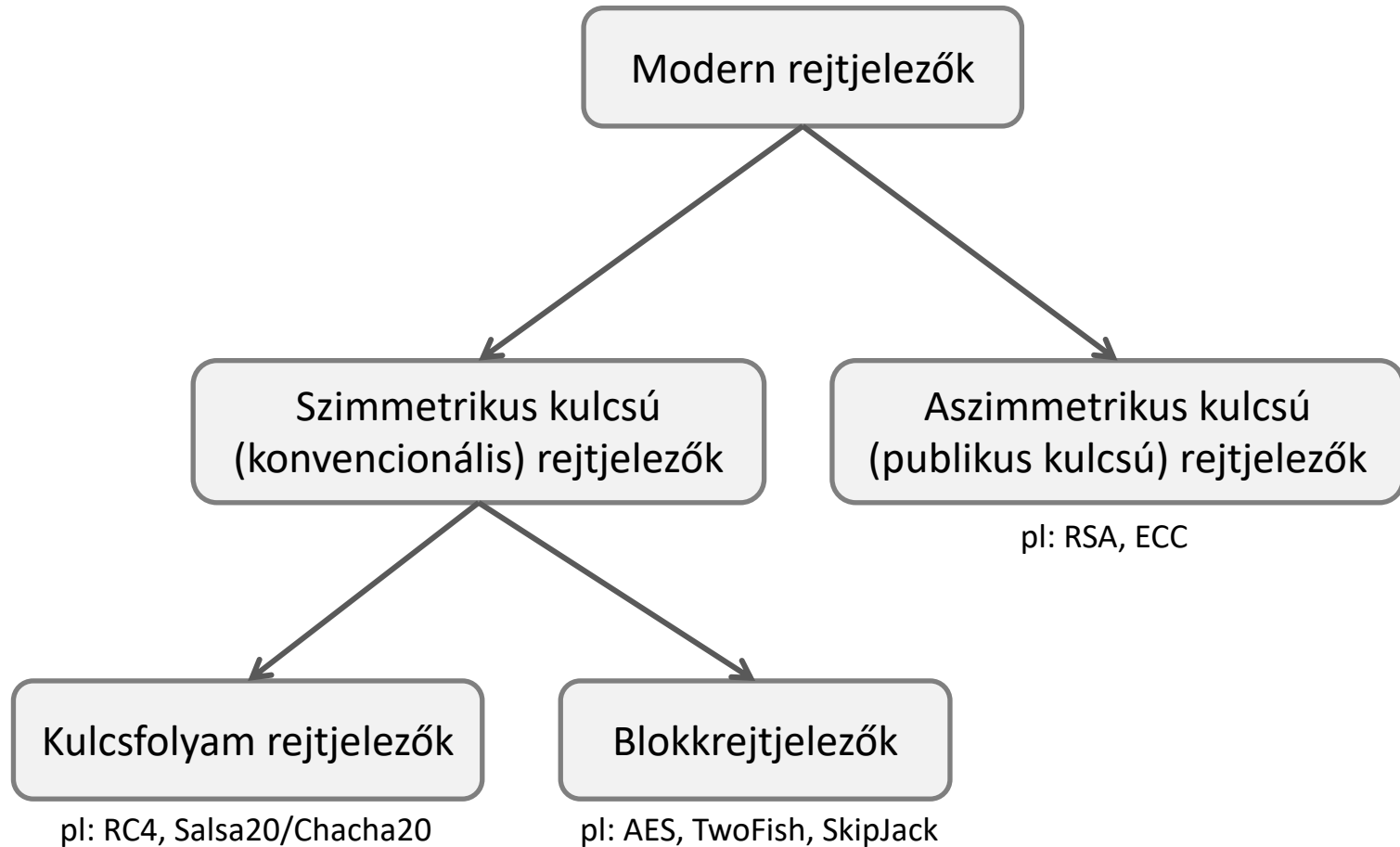
TIME OUT

THE TIMES

THE BENEDICT CUMBERBATCH KEIRA KNIGHTLEY
**IMITATION
GAME** 12A

BASED ON THE INCREDIBLE TRUE STORY

Modern rejtjelezők osztályozása



Az XOR művelet

- XOR (+ vagy \oplus)

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0$$

- Bit vektorok XOR-olását bitenként végezzük

- pl: $0011 + 1010 = 1001$

- A XOR művelet 3 fontos tulajdonsága:

1. $X + 0 = 0 + X = X$

2. $X + X = 0$

3. ha $A + B = C$, akkor $A = B + C$ (és $B = A + C$)

One-time pad

■ Kódolás

- Az üzenetet egy byte sorozatként reprezentáljuk: $m_1 m_2 \dots m_L$
- Kulcsként egy valódi véletlen byte sorozatot használunk: $k_1 k_2 \dots k_L$
- A kettőt egymáshoz XOR-oljuk, ez adja a rejtett üzenetet:
 $m_1 m_2 \dots m_L + k_1 k_2 \dots k_L = c_1 c_2 \dots c_L$ ahol $c_i = m_i + k_i$ minden $i = 1, \dots, L$

■ Dekódolás

- A kulcsként használt byte sorozatot a rejtett üzenethez XOR-oljuk:
 $c_1 c_2 \dots c_L + k_1 k_2 \dots k_L = m_1 m_2 \dots m_L$ (mert $c_i + k_i = m_i + k_i + k_i = m_i$)

■ Tulajdonságok:

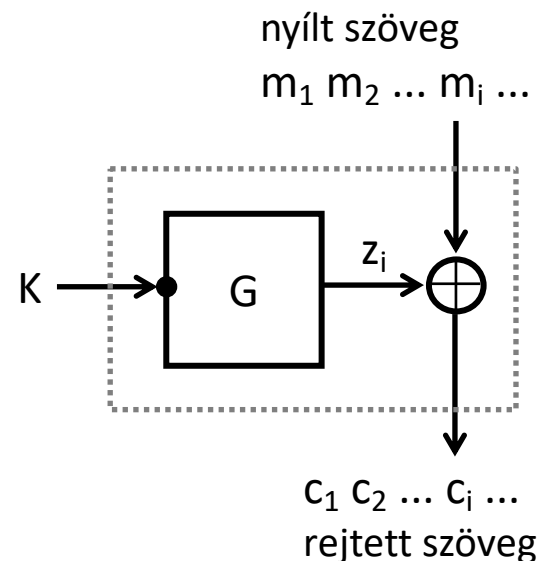
- *Tökéletes titkosítás (perfect secrecy)*
- A kulcs mérete legalább akkora kell legyen, mint a (tömörített) nyílt üzenet mérete → nem túl praktikus, a gyakorlatban nem használjuk

Kulcsfolyam rejtjelezők

- Ötlet: szimuláljuk a valódi véletlen kulcsfolyamot egy álvéletlen byte sorozattal, amit egy algoritmussal generálunk egy véletlen bemenetből (seed)

- terminológia:

- m_i – nyílt szöveg (plaintext) karakter
- c_i – rejtett szöveg (ciphertext) karakter
- z_i – kulcsfolyam (key-stream) karakter
- K – kulcs (seed)
- G – kulcsfolyam generátor



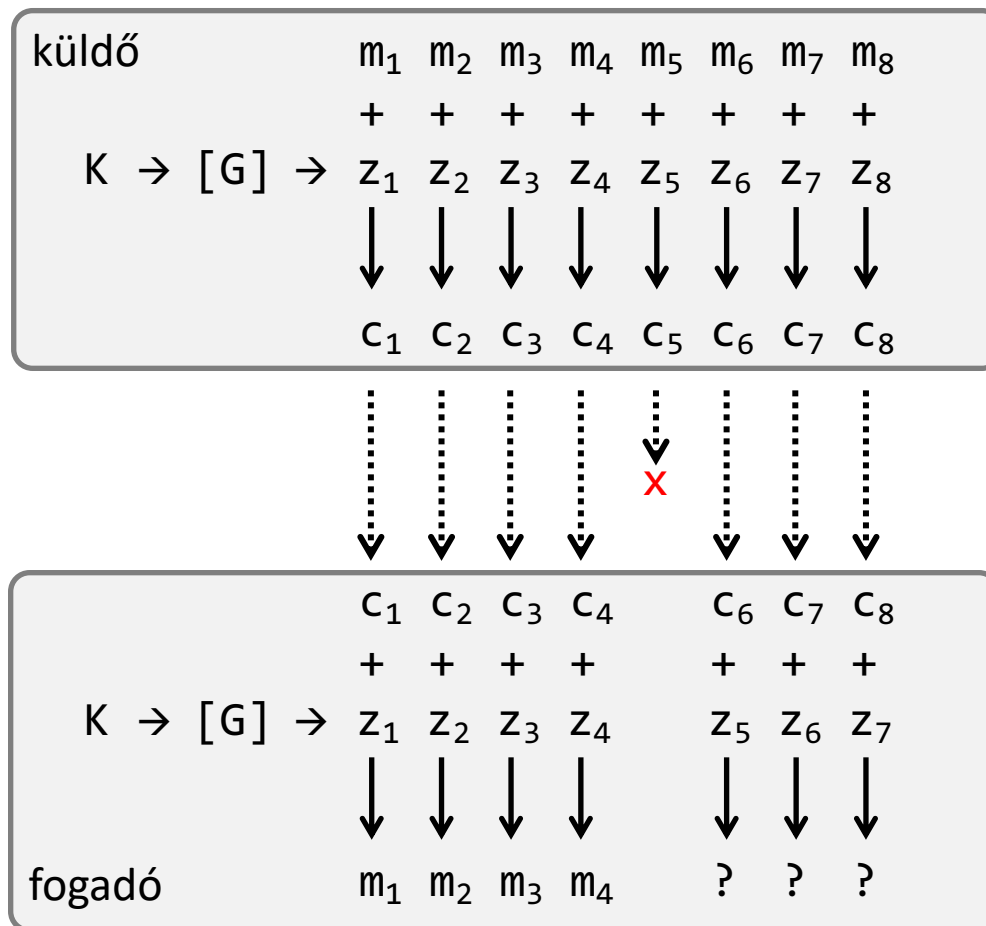
- Példák:

RC4, A5 (GSM), E0 (Bluetooth), Salsa20/ChaCha20

A kulcsfolyam rejtjelezők tulajdonságai

- Hatékonyság
 - A kulcsfolyam rejtjelezők nagyon gyorsak (főleg a hardver implementációk)
 - A generátor belső állapotának tárolásához nem kell sok memória (pár száz byte) és a generátor számítási műveletei is egyszerűek
- A rejtett szöveg mérete mindig megegyezik a nyílt szöveg méretével
- A küldő és a fogadó szinkronban kell működjön
 - Ha elveszik egy karakter átvitel során, a fogadó kiesik a szinkronból
 - Ilyenkor a dekódolás nem lesz helyes
 - Külön mechanizmus szükséges ennek detektálásához és az újraszinkronizáláshoz
- A kulcsfolyam rejtjelezők egyáltalán nem biztosítanak integritás-védelmet !!!
 - Egy támadó tetszőleges bitet átbillenthet a rejtett szövegben, és ez az adott bit átbillenését okozza a nyílt szövegben
 - A fogadó nem biztos, hogy ezt az egy bit hibát (megbízhatóan) észreveszi

Illusztráció: szinkronitás elvesztése



$$C_6 + Z_5 = m_6 + Z_6 + Z_5 \neq m_6$$

$$C_7 + Z_6 = m_7 + Z_7 + Z_6 \neq m_7$$

...

Illusztráció: üzenetek célzott módosítása

- A küldő a $c_i = m_i + z_i$ rejtett szöveg karaktert küldi el
- A támadó tetszőleges Δ értéket hozzá XOR-olhat c_i -hez
- A fogadó tehát a $c_i + \Delta$ értéket kapja meg, és ebből ezt dekódolja:

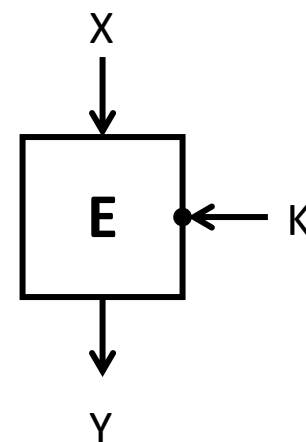
$$(c_i + \Delta) + z_i = (m_i + z_i + \Delta) + z_i = m_i + \Delta$$

- Azaz a nyílt szöveg karakter pontosan Δ -val módosul (abban a pozícióban, ahol Δ -ban 1-es bit található, m_i adott bitje invertálódik)

$$\begin{array}{r} m_i = 00101100 \\ = ++++++ \\ \Delta = 10000100 \\ \hline m_i' = 10101000 \end{array}$$

Blokkrejtjelezők

- A blokkrejtjelező több byte-ból álló blokkokat kódol/dekódol
 - tipikus blokkméret: 16 byte = 128 bit
- Ha valaki nem ismeri a kulcsot, akkor számára a blokkrejtjelező olyan, mint egy véletlen függvény
 - a kimenete megjósolhatatlan, hiába ismert a bemenet
- jelölések:
 - kódolás: $E(K, X)$ vagy $E_K(X)$
 - dekódolás: $E_K^{-1}(Y)$ vagy $D_K(Y)$
- terminológia
 - X – nyílt szöveg blokk (n hosszú bit vektor)
 - Y – rejtett szöveg blokk (n hosszú bit vektor)
 - K – kulcs (k hosszú bit vektor, pl. $k = 128 \dots 256$)
 - E – kódoló algoritmus
 - D – dekódoló algoritmus



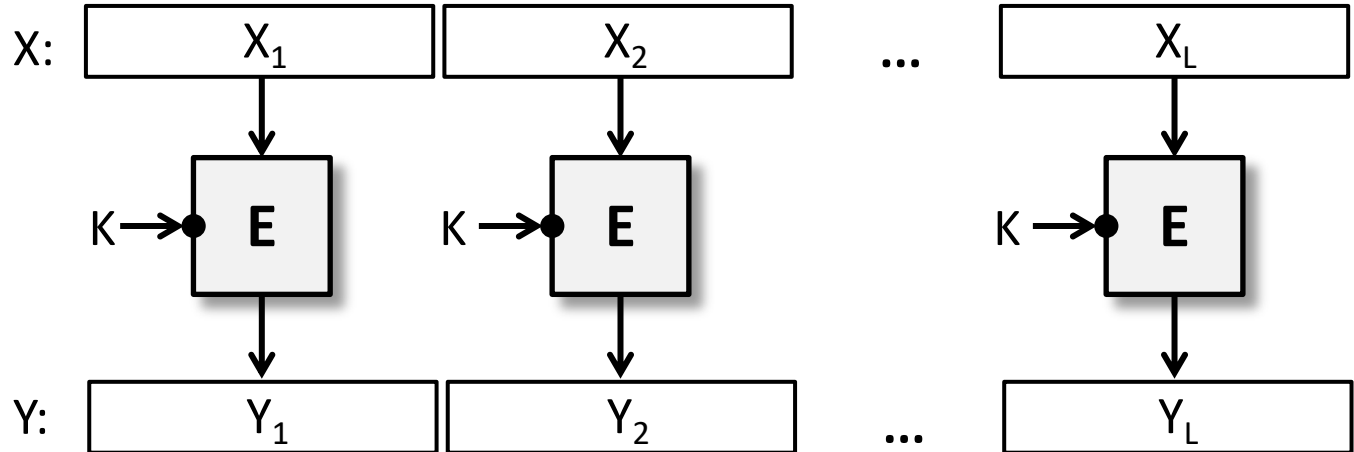
példák:
AES, DES (3DES), RC5,
Twofish, Skipjack, ...

Blokkrejtjelezési módok

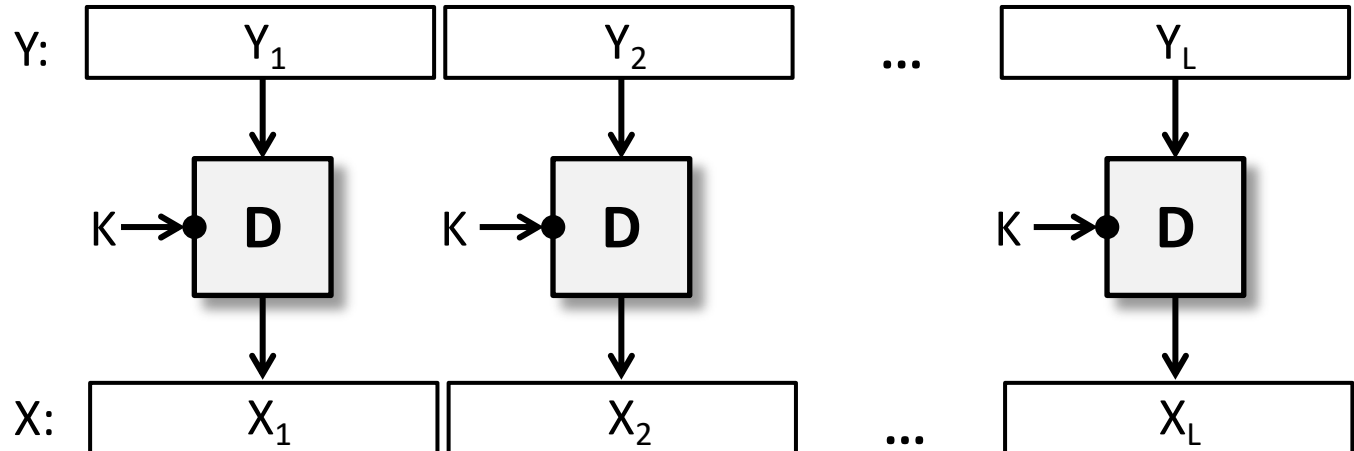
- Alap módok:
 - Electronic Codebook (ECB) mód
 - Cipher Block Chaining (CBC) mód
 - Cipher Feedback (CFB) mód
 - Output Feedback (OFB) mód
 - Counter (CTR) mód
- Néhány speciális mód:
 - XCBC
 - CBC with Ciphertext Stealing (CTS)
- Hitelesített rejtjelező (authenticated encryption) módok:
 - CCM: CTR + CBC MAC
 - GCM: Galois CTR mód
 - OCB: Offset Codebook mód

ECB mód

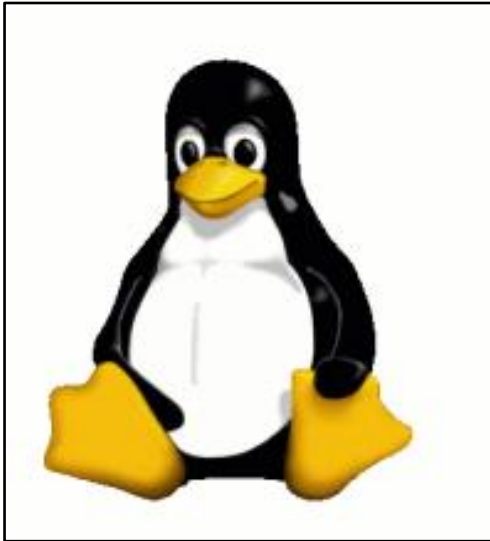
- kódolás



- dekódolás



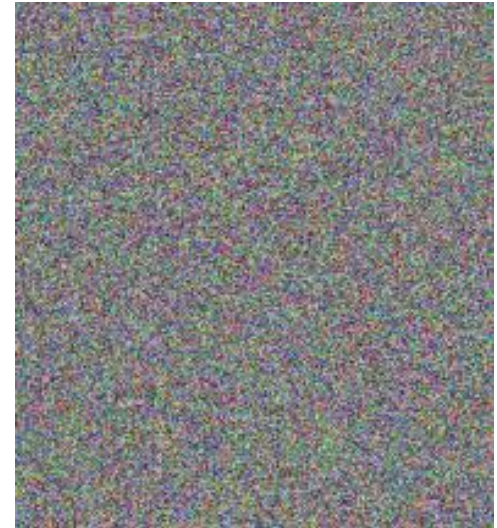
Illusztráció: az ECB mód gyengesége



eredeti kép

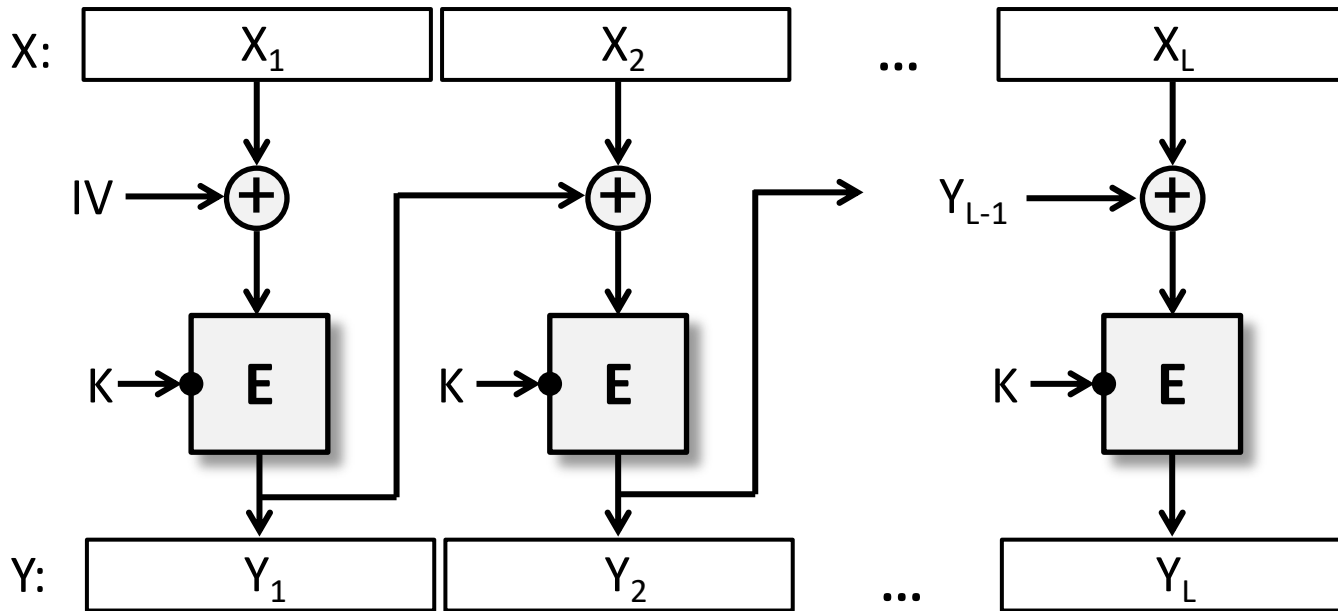


ECB módban kódolt kép



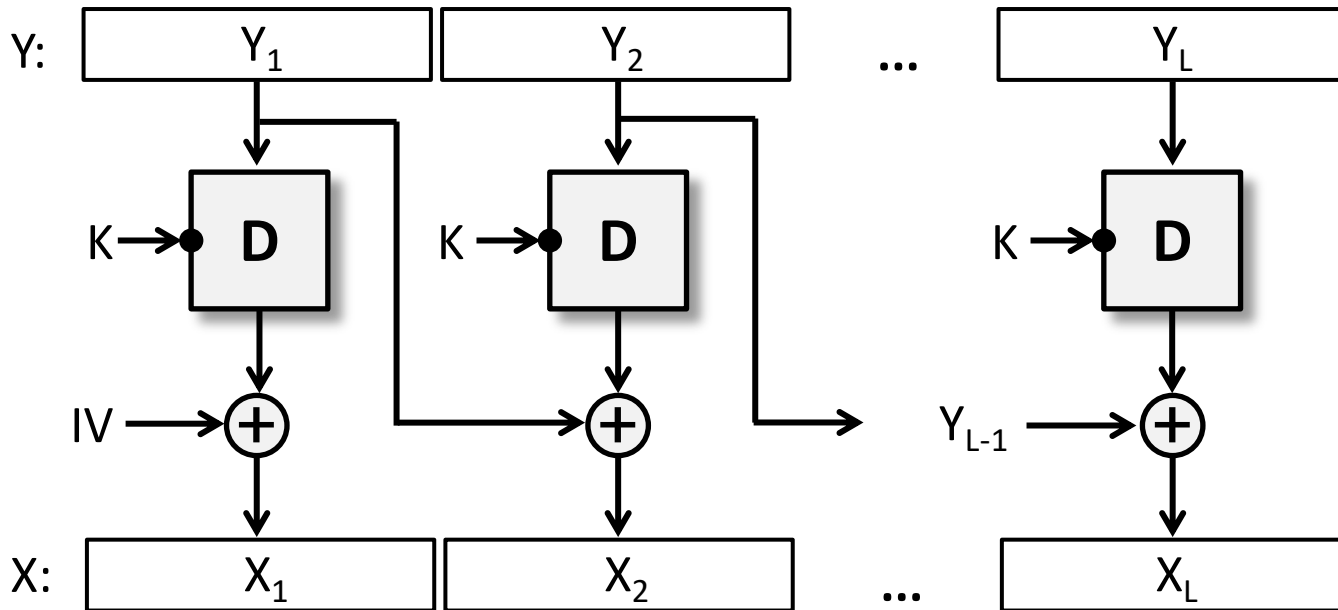
más módok esetén...

CBC mód (kódolás)



$$Y_i = E_K(X_i \oplus Y_{i-1})$$

CBC mód (dekódolás)

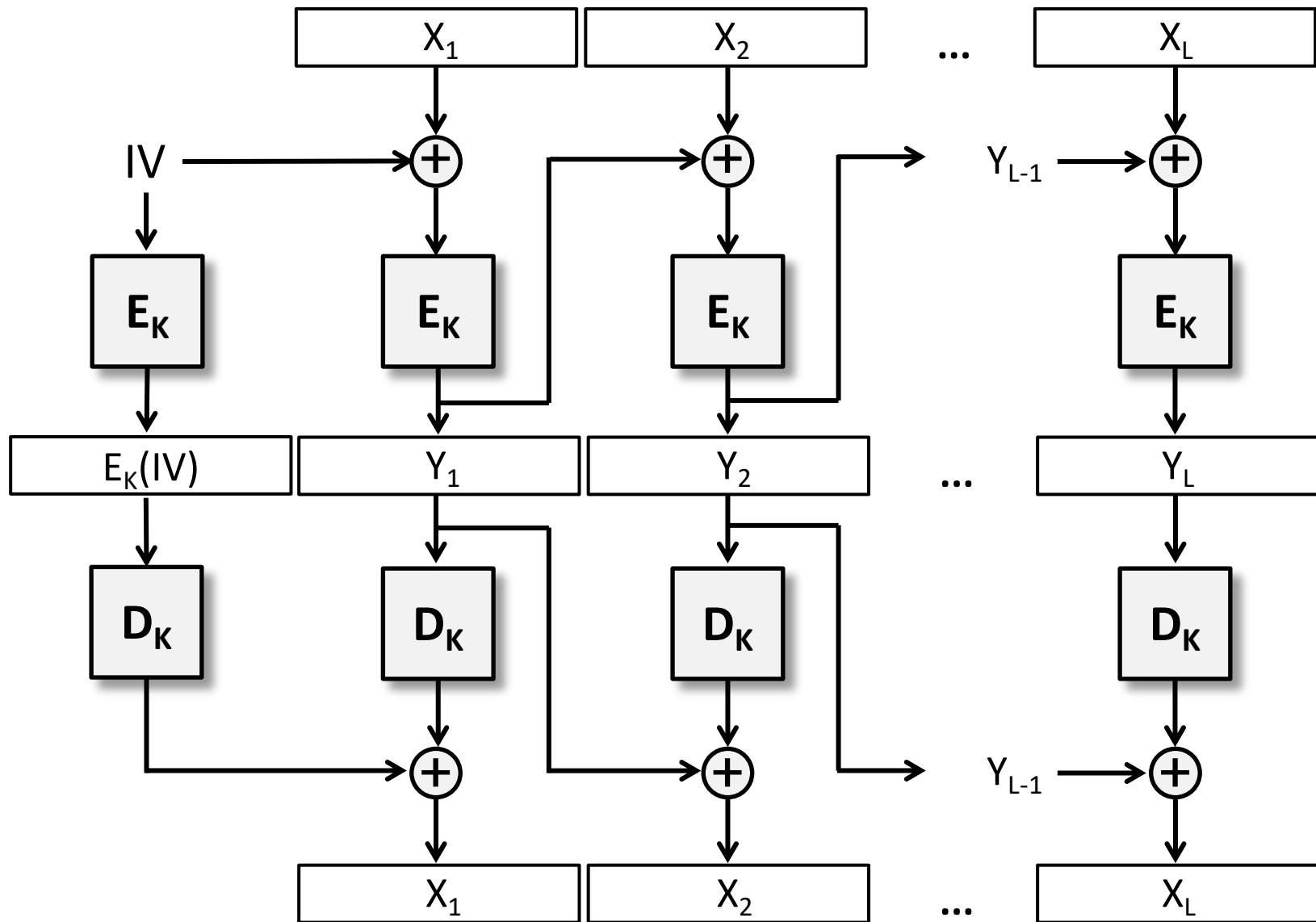


$$X_i = D_K(Y_i) \oplus Y_{i-1}$$

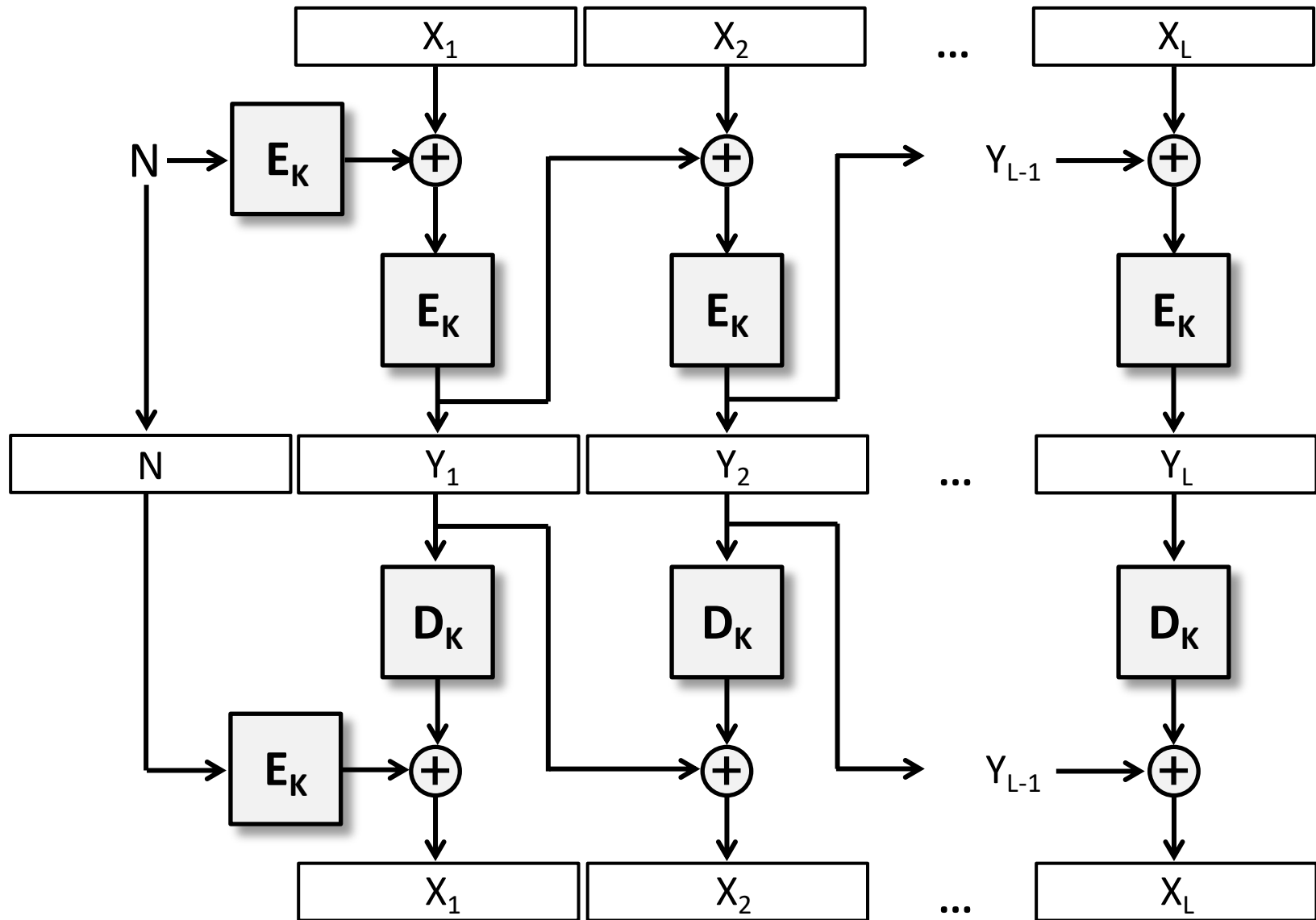
Nem-predikálható IV generálása

- **IV = kriptográfiai véletlenszám generátor kiemete**
 - Programnyelvek standard véletlenszám generátorai (pl: rnd(), rand(), ...) nem megfelelők a célra, mert predikálható a kimenetük!
 - Speciális, kripto könyvtárakban található véletlenszám generátorok használata tanácsos
 - Az IV manipuláció elleni védelme céljából, az IV-t rejtjelezve (pl: $E_K(IV)$) érdemes elküldeni a fogadó oldalra (az üzenettel együtt)
- **IV = $E_K(N)$**
 - N egy egyszer használt érték (nonce = "number used once")
 - Például, N lehet egy üzenet sorszám vagy egy egyedi üzenet azonosító
 - N nyíltan átküldhető a fogadó oldalra (az üzenettel együtt), a fogadó ebből elő tudja állítani az IV-t az üzenet dekódolásához
 - Bár a támadó manipulálhatja N-et az átvitel során, nem tudja ennek milyen hatása lesz az IV-re a fogadó oldalon (manipuláció elleni védelem)
- Mellékhatásként mindkét módszer esetén megőrződik az IV titkossága is (bár ez nem fontos biztonsági szempontból)

Illusztráció: véletlen IV használata



Illusztráció: IV mint rejtjelezett nonce

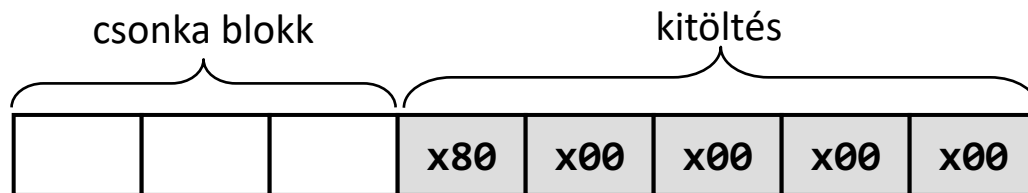


Kitöltés

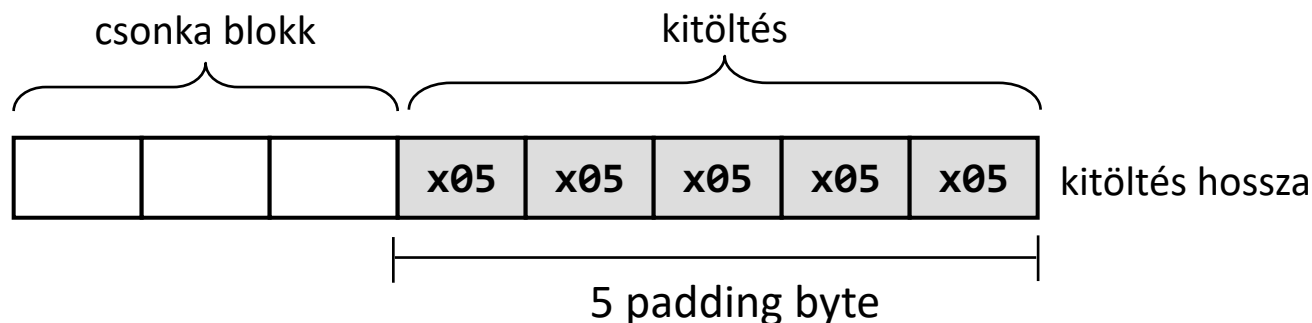
- Az üzenet hossza általában nem többszöröse a blokkmértének, az utolsó blokk sokszor csonka
- Az utolsó csonka blokkot feltölthetjük extra byte-okkal, hogy egy teljes blokkot kapjunk – ezt nevezzük kitöltésnek (padding)
- A fogadónak egyértelműen fel kell tudni ismerni a kitöltést, hogy el tudja távolítani azt a dekódolás után
- Ebből az egyértelműségi követelményből az következik, hogy **mindig** kell kitöltést alkalmazni, még akkor is, ha az üzenet hossza pont a blokkméret többszöröse (ilyenkor egy teljes egész extra blokkot kell még az üzenethez fűzni kitöltésként)

Ismert kitöltési sémák

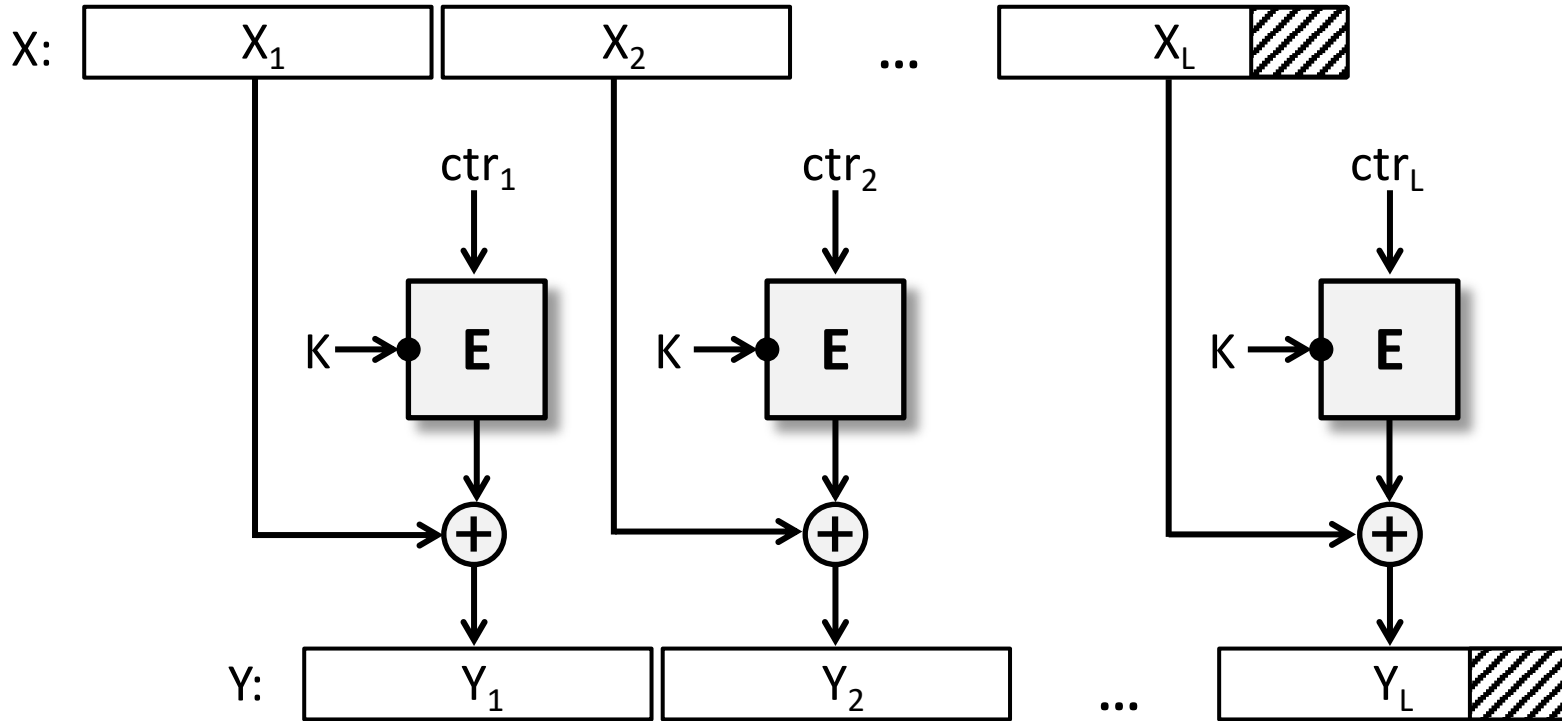
- Egy $x80$ byte majd annyi $x00$ byte amennyi még szükséges [ISO 7816-4]



- A kitöltés utolsó byte-ja a kitöltés hosszát tartalmazza [ANSI X.923, PKCS#7]
 - A többi byte lehet véletlen (SSL padding)
 - A többi byte értéke lehet valamilyen speciális érték, pl. a padding hossz byte (TLS padding)

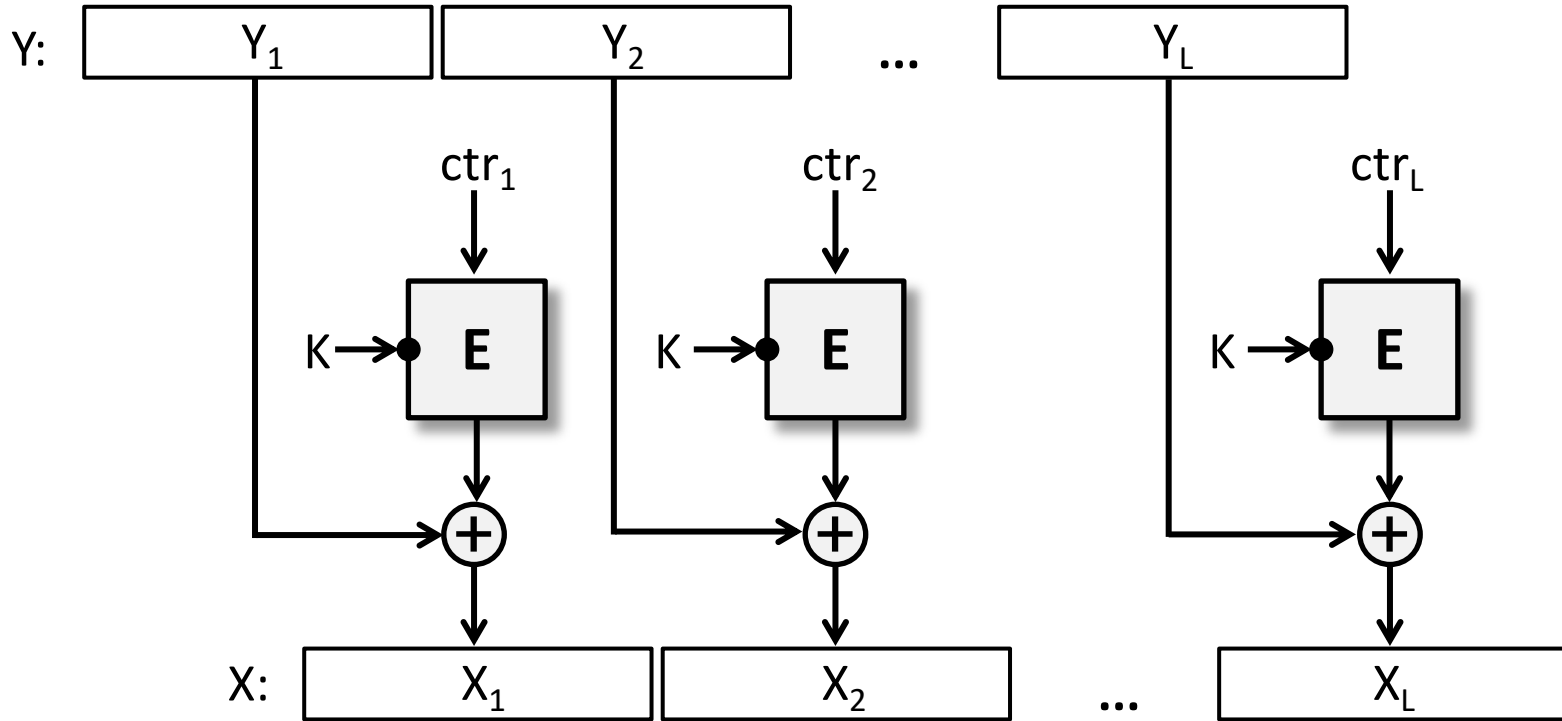


CTR mód (kódolás)



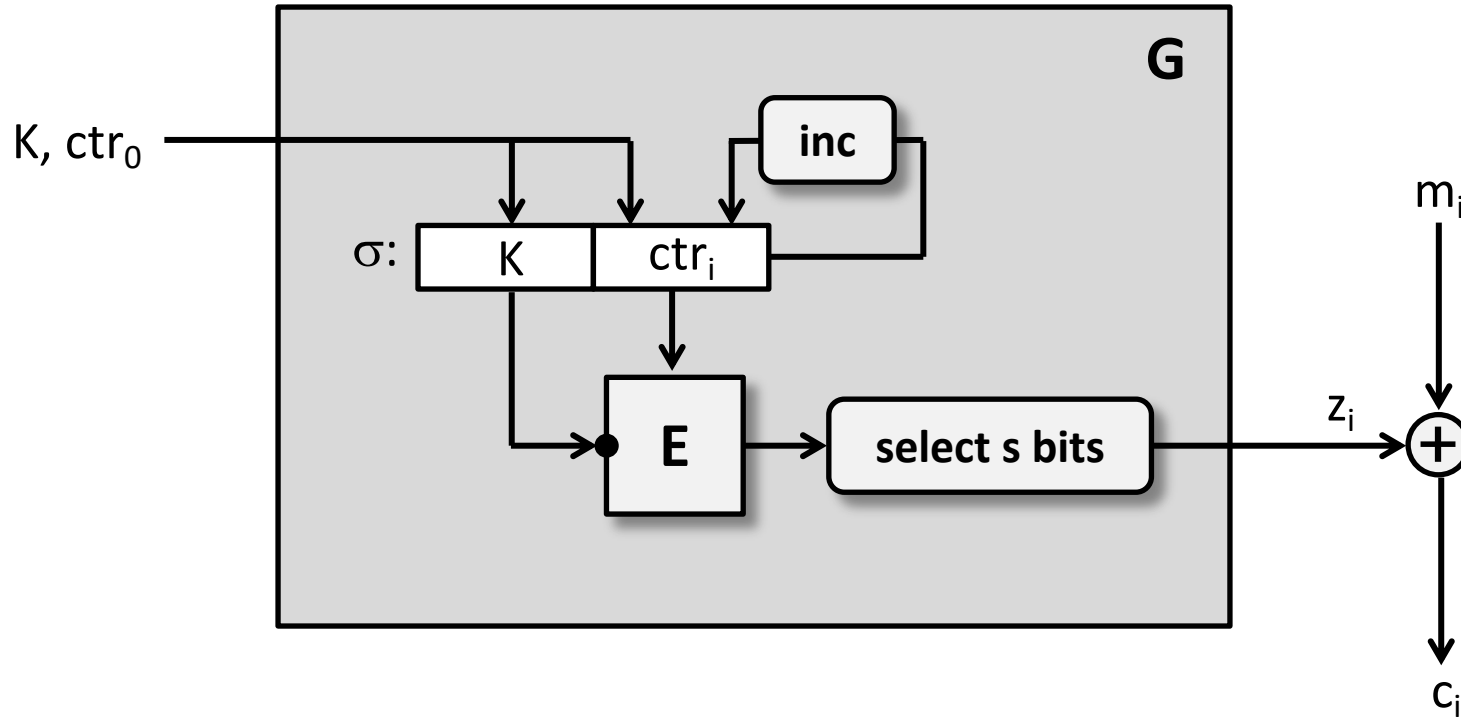
$$Y_i = X_i \oplus E_K(ctr_i)$$
$$ctr_{i+1} = ctr_i + 1$$

CTR mód (dekódolás)



$$X_i = Y_i \oplus E_K(ctr_i)$$
$$ctr_{i+1} = ctr_i + 1$$

A CTR mód más nézetben



Ez lényegében egy kulcsfolyam rejtjelező!

- Belső állapot: (K, ctr_i)
- Állapot update függvény: számláló növelése
- Kimenet generátor függvény: a blokkrejtjelező (kiment csonkolással)

A számláló blokk előállítása

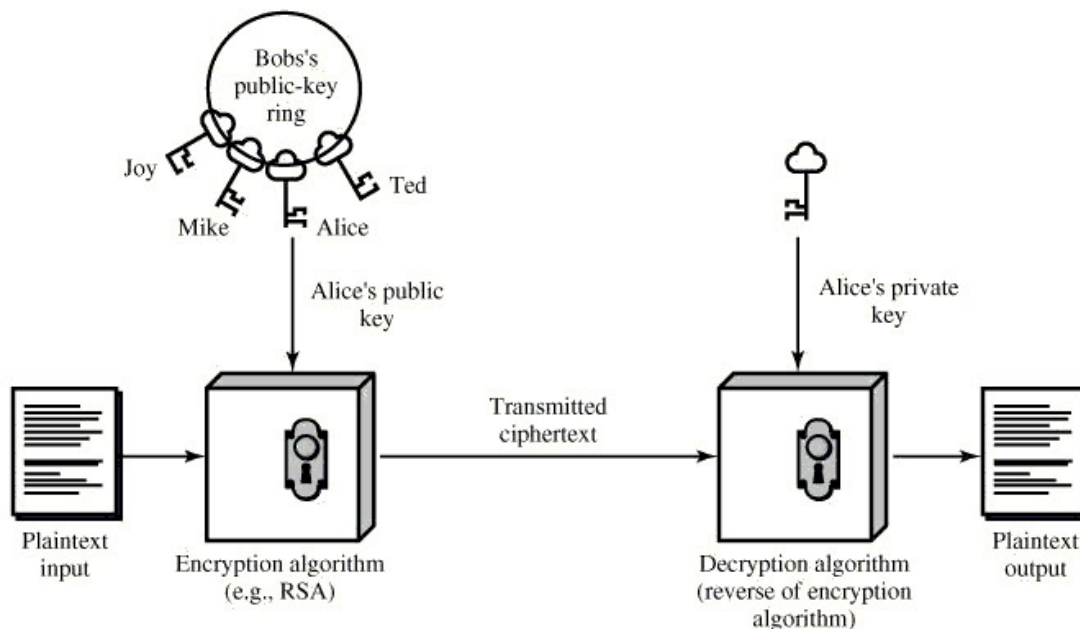
- Nagyon fontos követelmény, hogy a számláló értéke ne ismétlődjön, különben...
 - ha $Y = E_K(\text{ctr})+X$ és $Y' = E_K(\text{ctr})+X'$, akkor $Y + Y' = X + X'$
 - azaz, ha X (vagy X egy része) ismert, akkor X' -t (vagy X' egy részét) ki tudja számolni a támadó
- Ez azt jelenti, hogy a számláló nem ismétlődhet...
 - egy üzenet belül, és
 - azonos kulccsal kódolt üzenetek között sem
- Ez tipikusan a következő módon oldható meg:
 - A számláló blokkot két részre osztjuk: $\text{ctr} = \text{ctr}' \mid \text{ctr}''$, ahol ctr'' b bit hosszú és ctr' $n-b$ bit hosszú (n a blokkrejtjelező blokkmérete bitbeken mérve)
 - ctr' egy nonce (pl: egy egyedi üzenet azonosító vagy sorszám), ami minden üzenetre más (\rightarrow lehetséges üzenetek száma $\max 2^{n-b}$)
 - ctr'' egy számláló, ami 0-ról indul és az üzenet minden blokkjára 1-gyel nő (\rightarrow lehetséges blokkok száma az üzenetben $\max 2^b$)

Rejtjelezők elleni támadások

- Kerckhoff-elv
 - A támadóról mindig azt érdemes feltételezni, hogy ismeri a kódoló (és dekódoló) algoritmust, és csak a kulcs ismeretlen számára
 - Ezen elv figyelmen kívül hagyását „security-by-obscurity”-nek nevezik (ami kerülendő)
- Támadói modellek
 - A támadás célja lehet:
 - » Rejtett üzenetek szisztematikus feltörése
 - » A kulcs megfejtése
 - A támadó számára rendelkezésre álló információk jellege:
 - » Csak rejtett szövegek (ciphertext-only attack)
 - » Ismert nyílt szöveg – rejtett szöveg párok (known plaintext attack)
 - » (adaptívan) választott nyílt szövegekhez (rejtett szövegekhez) tartozó rejtett szövegek (nyílt szövegek) (chosen plaintext/ciphertext attacks) --» orákulum támadások
- Kimerítő kulcskeresés (exhaustive key search) támadás
 - Átlagos komplexitása 2^{k-1} , ha a kulcs k bites
- Algebrai támadások (algebraic attacks)
 - A rejtjelező (mint matematikai objektum) gyengeségeit kihasználó támadások
 - Ha ismertté válik egy olyan algebrai támadás, melynek komplexitása jóval kisebb, mint a kimerítő kulcskeresés komplexitása, akkor azt mondjuk, hogy a rejtjelezőt feltörték

Az aszimmetrikus kulcsú rejtjelezés ötlete

- A kódolás és a dekódolás nem ugyanazt a kulcsot használja, a két kulcs különböző (bár matematikailag függenek egymástól)
- A dekódoló kulcs kiszámítása a kódoló kulcsból nehéz feladat bizonyos titkos információk ismerete nélkül
- Ekkor a kódoló kulcs nyilvánosságra hozható (--» publikus kulcs, publikus kulcsú kriptográfia)
- A dekódoló kulcsot azonban mindig titokban kell tartani (--» privát kulcs)

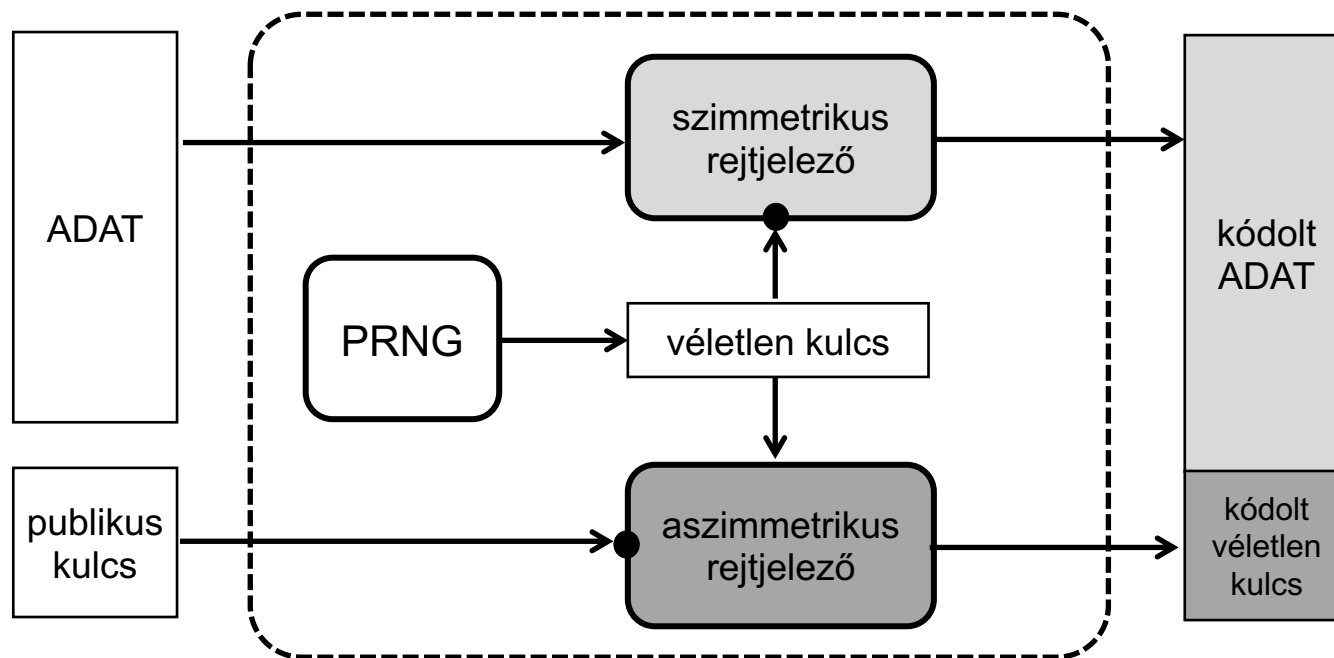


Aszimmetrikus kulcsú rejtjelezők

- Algoritmusok és terminológia:
 - Kulcspár generáló algoritmus: $G() = (K^+, K^-)$
 - K^+ – publikus kulcs
 - K^- – privát kulcs
 - Kódoló algoritmus: $E(K^+, X) = Y$
 - X – nyílt szöveg
 - Y – rejtett szöveg
 - Dekódoló algoritmus: $D(K^-, Y) = X$
- A nyílt szöveg (és a rejtett szöveg) tipikusan több ezer bit (több száz byte) hosszúságú (hasonlóan a blokkrejtjelezéshez)
- Példák: RSA, ElGamal

Hibrid rejtjelezés

- Az aszimmetrikus kulcsú rejtjelezők általában sokkal lassabbak mint a szimmetrikus kulcsú rejtjelezők és nagyobb a kulcsméretük (pl. 2048-4096 bit)
- A sebesség probléma hibrid rejtjelezéssel oldható meg:



Publikus kulcsú rejtjelezők biztonsága

- Ezeknek a rejtjelezőknek a biztonsága általában valamilyen nehéznek vélt matematikai problémára vezethető vissza

Példák:

- faktorizáció
 - diszkrét logaritmus számítás
-
- Néha, matematikailag egzaktul bebizonyítható, hogy a rejtjelező feltörése legalább olyan nehéz feladat, mint a kapcsolódó, nehéznek vélt matematikai probléma megoldása (redukciós bizonyítás)
 - A gyakorlatban használt rejtjelezők esetén azonban csak részleges bizonyítások vannak...
-
- További gyakorlati megfontolások:
 - Szemantikus biztonság (pl. randomizációval érhető el) --» következő dia
 - Non-malleability (pl. a nyílt szöveg rejtjelezés előtti struktúrált formázásával érhető el)

Szemantikus biztonság

- Ha a támadó képes két ismert X és X' nyílt szöveghez tartozó $E_K(X)$ és $E_K(X')$ rejtett szövegről megmondani, hogy melyik tartozik X -hez és melyik X' -höz, akkor a rejtjelezés információt szivároztat ki!
- Ha a publikus kulcsú rejtjelező determinisztikus, akkor a fenti értelemben nem biztonságos, mert X , X' , $E_{K^+}(X)$, és $E_{K^+}(X')$, valamint a K^+ publikus kulcs ismeretében, a támadó triviálisan meg tudja mondani, hogy melyik nyílt szöveg melyik rejtett szöveghez tartozik (hogyan?)
- A probléma **randomizálással** (probabilistic encryption) oldható meg
 - Rejtjelezés előtt a nyílt szöveget egy véletlen értékkel módosítjuk (salting)
 - Dekódolás után a fogadónak ezt a véletlen értéket fel kell ismernie és el kell távolítania (hasonlóan a padding-hez)
 - Néhány publikus kulcsú rejtjelező eleve (by design) probabilisztikus (pl: ElGamal)
 - Más rejtjelezők (pl: RSA) esetén a nyílt szöveg randomizálását meg kell oldani, de léteznek erre szabványok (pl: RSA esetén a PKCS #1 szabvány definiálja az OAEP padding formátumot)

Kriptográfiai könyvtárak

Kriptográfiai könyvtárak

- Kriptográfiai algoritmusok implementációját tartalmazzák, és ezek használatát API-kon keresztül teszik lehetővé
- Általában egyéb hasznos, kriptográfiai alkalmazásokban szükséges funkciókat is tartalmaznak (pl: erős véletlenszám generátor, kulcsmenedzsment funkciók, szabványos adatformátumok megvalósítása, ...)
- Átlagos programozók által jól használhatóak

PyCryptodome

- Python-ban írt és Python programokban használható kriptó könyvtár és API
 - Rejtjelezők (szimmetrikus és aszimmetrikus kulcsú)
 - Hash és MAC függvények
 - Digitális aláírás sémák
 - Véletlenszám generátor
 - Kulcspár menedzsment funkciók (pl: generálás, export, import)
 - Jelszó alapú kulcsgenerálás (szimmetrikus kulcsú rejtjelezéshez)
- Python 3-mal kompatibilis
- Elérhetőség: <https://github.com/Legrandin/pycryptodome/>
- Dokumentáció: <https://www.pycryptodome.org/en/latest/>
- Egyszerű telepítés: `pip install pycryptodome`

PyCryptodome: Szimmetrikus kulcsú rejtjelezés

- Crypto.Cipher modul
- Támogatott szimmetrikus kulcsú algoritmusok:
 - ChaCha20, Salsa20, AES
 - RC2, RC4, Blowfish, CAST, DES, 3DES
- Használati példa: AES
 - AES rejtjelező objektum létrehozása az AES.new() függvénnyel
 - A rejtjelező kezdeti paraméterei a new() bemeneteként adhatók meg
 - » pl: blokkrejtjelezési mód, kulcs, IV, számláló kezdő értéke, ...
 - A kódolás és a dekódolás az encrypt() és a decrypt() függvényekkel történik, melyeknek bemenetként kell megadni a (kitöltött) nyílt illetve a rejtett szöveget

PyCryptodome: Kódolás AES-sel CBC módban

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util import Padding
```

aes-cbc-enc-test.py

```
ifile = open('testinput.txt', 'rb')
plaintext = ifile.read()
ifile.close()
```

```
plaintext = Padding.pad(plaintext, AES.block_size)
```

```
key = b'0123456789abcdef0123456789abcdef' «-- Ez csak egy (rossz) példa!  
iv = get_random_bytes(AES.block_size)      Soha nem szabad kulcsot  
cipher = AES.new(key, AES.MODE_CBC, iv)    alkalmazásba belekódolni!
```

```
ciphertext = cipher.encrypt(plaintext)
```

```
print('IV: ' + iv.hex())
print('Ciphertext: ')
print(ciphertext.hex())
```

```
ofile = open('testoutput.bin', 'wb')
ofile.write(iv + ciphertext)
ofile.close()
```

Mi történik az encrypt() függvényen belül?

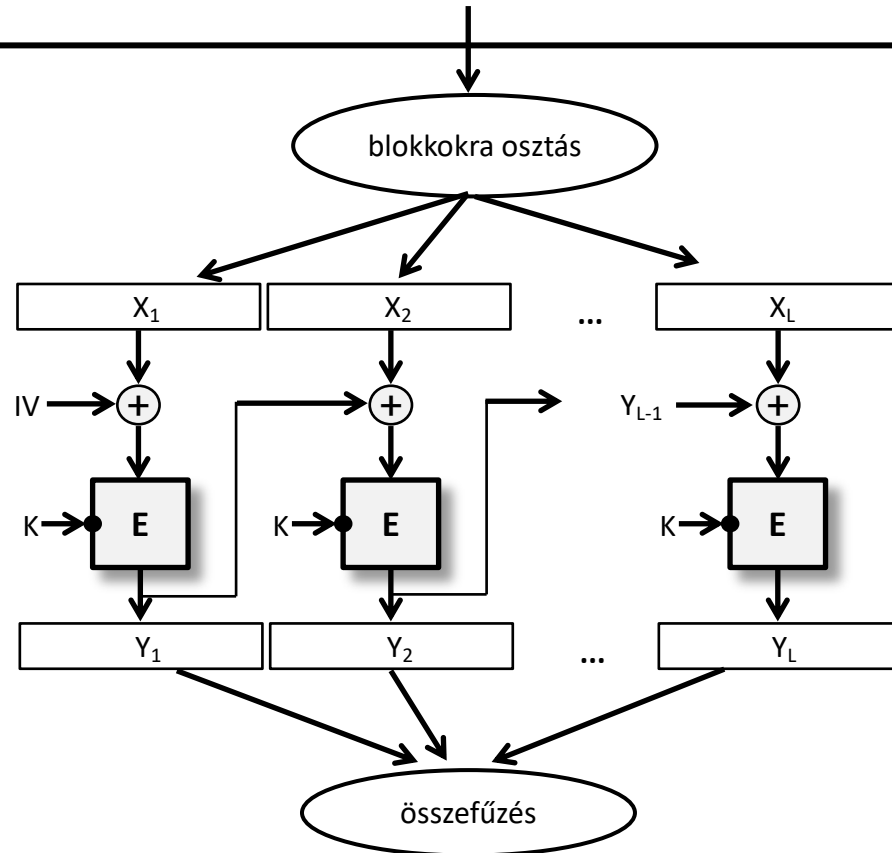
bemenet: tetszőleges méretű, kitöltött nyílt szöveg

encrypt()

Az IV és a K kulcs a new() meghívásakor került megadásra

$E = AES$, mert az AES osztály new() függvényét hívtuk meg.

Hasonlóan, a CBC mód a new() meghívásakor került kiválasztásra



kimenet: tetszőleges méretű rejtett szöveg

PyCryptodome: Dekódolás AES-sel CBC módban

```
from Crypto.Cipher import AES
from Crypto.Util import Padding
```

aes-cbc-dec-test.py

```
ifile = open('testoutput.bin', 'rb')
ciphertext = ifile.read()
ifile.close()
```

```
iv = ciphertext[:AES.block_size]
ciphertext = ciphertext[AES.block_size:]
```

```
key = b'0123456789abcdef0123456789abcdef' «-- Ez csak egy (rossz) példa!  
cipher = AES.new(key, AES.MODE_CBC, iv)      Soha nem szabad kulcsot  
                                             alkalmazásba belekódolni!
```

```
plaintext = cipher.decrypt(ciphertext)
plaintext = Padding.unpad(plaintext, AES.block_size)
```

```
print(plaintext.decode('utf-8'))
```

PyCryptodome: Kulcspár menedzsment

- `Crypto.PublicKey` modul
- Támogatott kulcs típusok
 - RSA, DSA, elliptikus görbe algoritmusok (ECC)
- Használati példa: RSA kulcspár
 - RSA kulcspár az `RSA.generate()` függvény meghívásával generálható
 - » A kívánt kulcsméret a `generate()` bemeneteként adható meg
 - » Az alapértelmezett publikus exponens $2^{16}+1$, de ez módosítható
 - A létrehozott RSA kulcspár objektum `export_key()` függvényével exportálható a kulcspár PEM vagy DER formátumban
 - Ha csak a publikus kulcsot szeretnénk exportálni, akkor először a `publickey()` függvényt kell meghívni, majd a visszaadott kulcs objektum `export_key()` függvényét
 - Az `RSA.import_key()` függvénnyel kulcsot vagy kulcspárt importálhatunk alkalmazásunkba

PyCryptodome: RSA kulcspár generálás és export

```
from Crypto.PublicKey import RSA
```

rsa-key-gen-test.py

```
key = RSA.generate(4096)
```

```
# export the entire key pair in PEM format
```

```
ofile = open('rsa-test-keypair.pem', 'w')
```

```
keypairstr = key.export_key(format='PEM', passphrase='x#4K').decode('ASCII')
```

```
ofile.write(keypairstr)
```

```
ofile.close()
```

```
# export only the public key in PEM format
```

```
ofile = open('rsa-test-pubkey.pem', 'w')
```

```
ofile.write(key.publickey().export_key(format='PEM').decode('ASCII'))
```

```
ofile.close()
```

```
# export the entire key pair in DER format
```

```
ofile = open('rsa-test-keypair.der', 'wb')
```

```
ofile.write(key.export_key(format='DER', passphrase='x#4K'))
```

```
ofile.close()
```

```
# export only the public key in DER format
```

```
ofile = open('rsa-test-pubkey.der', 'wb')
```

```
ofile.write(key.publickey().export_key(format='DER'))
```

```
ofile.close()
```

↑
*Ez csak egy (rossz) példa!
Soha nem szabad jelszót
alkalmazásba belekódolni!*

PyCryptodome: RSA-OAEP kódolás

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

kfile = open('rsa-test-pubkey.pem', 'r')
# kfile = open('rsa-test-pubkey.der', 'rb')
pubkeystr = kfile.read()
kfile.close()

pubkey = RSA.import_key(pubkeystr)
cipher = PKCS1_OAEP.new(pubkey)

plaintext = b'Plaintext should fit within 1 RSA block.'
ciphertext = cipher.encrypt(plaintext)

# print(ciphertext)
print(ciphertext.hex())

ofile = open('rsa-enc.out', 'wb')
ofile.write(ciphertext)
ofile.close()
```

rsa-enc-test.py

PyCryptodome: RSA-OAEP dekódolás

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
```

rsa-dec-test.py

```
kfile = open('rsa-test-keypair.pem', 'r')
keypairstr = kfile.read()
kfile.close()
```

```
keypair = RSA.import_key(keypairstr, passphrase='x#4K')
cipher = PKCS1_OAEP.new(keypair)
```

```
ifile = open('rsa-enc.out', 'rb')
ciphertext = ifile.read()
ifile.close()
```

```
plaintext = cipher.decrypt(ciphertext)
```

```
print(plaintext)
```



*Ez csak egy (rossz) példa!
Soha nem szabad jelszót
alkalmazásba belekódolni!*

Ellenőrző kérdések

- Milyen feladatokra jó a kriptográfia?
- Mi a rejtjelezés alapmodellje?
- Mi a különbség a kulcsfolyam rejtjelezők és a blokkrejtjelezők között?
- Hogyan működik a one-time pad? Miben különböznek a praktikus kulcsfolyam rejtjelezők a one-time pad-tól?
- Hogyan működik a CBC és a CTR blokkrejtjelezési mód (kódolás, dekódolás)?
- Mi a Kerckhoff-elv?
- Milyen támadó modellek léteznek rejtjelezés esetén?
- Mi a kimerítő kulcskeresés támadás átlagos komplexitása?
- Mi a publikus kulcsú rejtjelezés alapötlete?
- Mi a hibrid rejtjelezés motivációja, és hogyan működik?
- Mik azok a kriptográfiai könyvtárak? Miért hasznosak?