Kriptográfia és Információbiztonság
2. előadás

MÁRTON Gyöngyvér

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

2025

Miről volt szó az elmúlt előadáson?

- Követelmények, osztályozás
- Bevezető: alkalmazási terület, törzsanyag
- Könyvészet
- Történelmi háttér
- Kriptográfiai alapfogalmak
- Klasszikus kriptográfiai rendszerek
  - Caesar titkosító, Keyword Caesar
  - affine titkosító

Miről lesz szó?

- Klasszikus kriptográfiai rendszerek
  - helyettesítő rejtjelezés: affine titkosító
  - blokk titkosítók: Hill titkosító,
  - feltörési módszerek,
- a modern kriptográfia: tervezési szempontok, követelmények

Az affine rejtjelezés

- helyettesítő, monoalfabetikus rejtjelezés,
- a titkosítási és visszafejtési képletet betünként, a betűk számkódjára alkalmazzuk,
- $M = C = \{0, 1, \ldots, 25\}^*, K = \{(a, b) \in \{0, 1, \ldots, 25\}, (a, 26) = 1\}$,
- *Gen* kiválasztunk egy $key = (a, b) \in K$ értéket,
- $Enc_{(a,b)}(m) = (a \cdot m + b) \pmod{26} \longrightarrow c$,
- $Dec_{(a,b)}(c) = a^{-1} \cdot (c + 26 - b) \pmod{26} \longrightarrow m$,
- a^{-1}, az a multiplikatív inverze $\pmod{26}$ szerint, ami azt jelenti, hogy meg kell határozni azt az a^{-1} értéket, amelyre fennáll:

$$a \cdot a^{-1} \equiv 1 \pmod{26}.$$

- a multiplikatív inverz akkor létezik, ha $(a, 26) = 1$,
- nem biztonságos, többféleképpen is fel lehet törni:
  - statisztikai elemzés
  - az összes lehetséges kulcs kipróbálása, kulcsok száma: $12 \cdot 26 = 312$
  - ismert nyílt-szöveg támadás (known plaintext attack): rendelkezünk két betű rejtjelezett értékével,

Az affine rejtjelezés - ismert nyílt-szöveg támadás

Ismert nyílt-szöveg támadás (known plaintext attack): rendelkezünk két betű rejtjelezett értékével:

- tudva hogy az m_1 rejtjele c_1, és az m_2 rejtjele c_2, akkor a következő kongruencia-rendszer megoldásával, megállapítható a titkosításhoz használt (a, b) kulcs, ahol $(m_1 - m_2)^{-1}$ az $m_1 - m_2$ multiplikatív inverze:

$$\begin{aligned} m_1 \cdot a + b &= c_1 \pmod{26} \\ m_2 \cdot a + b &= c_2 \pmod{26}. \end{aligned}$$

$$(m_1 - m_2) \cdot a = (c_1 - c_2) \pmod{26}$$

$$\begin{aligned} a &= (m_1 - m_2)^{-1} \cdot (c_1 - c_2) \pmod{26} \\ b &= (c_1 - m_1 \cdot a) \pmod{26} \\ \text{vagy} & \\ b &= (c_2 - m_2 \cdot a) \pmod{26} \end{aligned}$$

Az affine rejtjelezés - ismert nyílt-szöveg támadás

Feladat: Ha tudjuk, hogy az affine-256 rejtjelezés során a 0x61 rejtjele 0xe7, illetve a 0x20 rejtjele 0x7c lett, határozzuk meg az eredeti üzenetet, ahol a rejtjelezett üzenet az encrMessageHex-ben van megadva.

```
encrMessageHex = '8d3f9593425db040987c3e9340985dae42963f807c9596419\
                 9967cec40e73f96989344987ce79898e73d957c985dabec43ed\
                 5d5a7c985da7ebe7e85da7ed987ce74095e740ebe79aeee727'
encrMessage = bytes.fromhex(encrMessageHex)

m1, c1 = 0x61, 0xe7
m2, c2 = 0x20, 0x7c
temp1, temp2 = pow(m1 - m2, -1, 256), c1 - c2
a = (temp1 * temp2) & 255
b = (c1 - (m1 * a)) & 255
print('key: ', (a, b))
a1 = pow(a, -1, 256)

decrMessage = bytearray()
for e in encrMessage: decrMessage.append((a1 * (e - b) & 255))
print(decrMessage.decode())
```

A Hill-titkosító

- 1929-ben publikálta Lester S. Hill,
- polialfabetikus titkosító: ugyanannak a betűnek nem mindig ugyanaz lesz a rejtjele,
- blokktitkosító (blockcipher): egyszerre több betűt is képes titkosítani,
- az egyszerre titkosított betűk száma fogja a blokkméretet jelenteni,
- lineáris algebrán alapszik: mátrix műveleteket végez,
- az üzenetek halmazát az angol/latin ábécé 26 betűjének megfelelő számkódok alkotják, $\pmod{26}$ szerint számolunk,
- bájtokkal is lehet dolgozni, $\pmod{256}$ szerint számolunk,
- a *key* kulcs egy $d \times d$-es mátrix lesz, ahol d a blokk méret,
- ha az üzenet hossza nem osztható a blokk mérettel, akkor: padding technikára kerül sor, azaz az üzenetet kiegészítjük.

A Hill-titkosító

- a titkosítási, visszafejtési képletet d betűnként kell alkalmazni,
- a *key* kulcs mátrix az elemeit az $\{0, 1, \ldots, 25\}$ halmazból veszi fel,
- fenn kell álljon: $(det_{key}, 26) = 1$, ahol det_{key} a *key* mátrix determinánsát jelöli,
- egy $m \in M = \{0, 1, \ldots, 25\}^d$, d betűs üzenet titkosítása:

$$Enc_{key}(m) = (key \cdot m) \pmod{26} \longrightarrow c.$$

- a visszafejtés:

$$Dec_{key}(c) = (key^{-1} \cdot c) \pmod{26} \longrightarrow m,$$

ahol key^{-1} a *key* inverze:

$$key \cdot key^{-1} \equiv I \pmod{26},$$

ahol I a $d \times d$ egységmátrix.

A Hill-titkosító

- ha a blokkméret $d = 2$, akkor egyszerre 2 betűt lehet titkosítani. Legyen

$$key = \begin{pmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2} \end{pmatrix}.$$

- ha $m = (m_1, m_2)$ és $c = (c_1, c_2)$, akkor a titkosítás:

$$\begin{pmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2} \end{pmatrix} \cdot \begin{pmatrix} m_1 \\ m_2 \end{pmatrix} \longrightarrow \begin{pmatrix} c_1 \\ c_2 \end{pmatrix},$$

- legyen a det_{key} multiplikatív inverze det_{key}^{-1},
- legyen az adjungált mátrix adj_{key}, ekkor:

$$adj_{key} = \begin{pmatrix} k_{2,2} & 26 - k_{1,2} \\ 26 - k_{2,1} & k_{1,1} \end{pmatrix}, \quad key^{-1} = det_{key}^{-1} \cdot adj_{key},$$

- a visszafejtés:

$$key^{-1} \cdot \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \longrightarrow \begin{pmatrix} m_1 \\ m_2 \end{pmatrix}.$$

A Hill-titkosító, példa kulcsgenerálásra

Legyen

$$key = \begin{pmatrix} 3 & 3 \\ 24 & 1 \end{pmatrix}, \text{ és } d = 2.$$

Ekkor

$$det_{key} = 1 \cdot 3 - (24) \cdot 3 = -69 \equiv 9 \pmod{26},$$

$$det_{key}^{-1} = 3, \text{ mert } 9 \cdot 3 \equiv 1 \pmod{26},$$

$$adj_{key} = \begin{pmatrix} 1 & 23 \\ 2 & 3 \end{pmatrix},$$

$$key^{-1} = det_{key}^{-1} \cdot adj_{key} = 3 \cdot \begin{pmatrix} 1 & 23 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} 3 & 69 \\ 6 & 9 \end{pmatrix} = \begin{pmatrix} 3 & 17 \\ 6 & 9 \end{pmatrix} \pmod{26}.$$

A Hill-titkosító, kulcsgenerálás Python-ban

Az inverz mátrix meghatározásához használhatjuk a Python SymPy (Library for Symbolic Matematics) könyvtárcsomagját, ahol:

- a Matrix típussal mátrixokat lehet létrehozni,
- a det-el kiszámolhatjuk a determinánst,
- az adjugate-el meghatározhatjuk az adjungált mátrixot,
- az inv_mod-al megadható az inverz mátrix.

```
import sympy as sp
mod = 26
key = sp.Matrix([[3, 3], [24, 1]])
print('A determinans: ', key.det())
print('Adjungalt matrix: ', key.adjugate())
invKey = key.inv_mod(mod)
print('A kulcs inverz erteke: ', invKey)
```

A Hill-titkosító, példa ha $d = 2$

Titkosítás, visszafejtés:

- ha a nyílt szöveg:

$$AM\ AT\ HE\ MA\ TI\ CI\ AN,$$

akkor a titkosított szöveg:

$$KM\ FT\ HQ\ KC\ DW\ EE\ NN,$$

- az első két betű-tömb titkosítása:

$$\begin{pmatrix} 3 & 3 \\ 24 & 1 \end{pmatrix} \cdot \begin{pmatrix} A \\ M \end{pmatrix} = \begin{pmatrix} 3 & 3 \\ 24 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 12 \end{pmatrix} = \begin{pmatrix} 10 \\ 12 \end{pmatrix} = \begin{pmatrix} K \\ M \end{pmatrix},$$

$$\begin{pmatrix} 3 & 3 \\ 24 & 1 \end{pmatrix} \cdot \begin{pmatrix} A \\ T \end{pmatrix} = \begin{pmatrix} 3 & 3 \\ 24 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 19 \end{pmatrix} = \begin{pmatrix} 5 \\ 19 \end{pmatrix} = \begin{pmatrix} F \\ T \end{pmatrix}.$$

- az első két betű-tömb visszafejtése:

$$\begin{pmatrix} 3 & 17 \\ 6 & 9 \end{pmatrix} \cdot \begin{pmatrix} K \\ M \end{pmatrix} = \begin{pmatrix} 3 & 17 \\ 6 & 9 \end{pmatrix} \cdot \begin{pmatrix} 10 \\ 12 \end{pmatrix} = \begin{pmatrix} 0 \\ 12 \end{pmatrix} = \begin{pmatrix} A \\ M \end{pmatrix},$$

$$\begin{pmatrix} 3 & 17 \\ 6 & 9 \end{pmatrix} \cdot \begin{pmatrix} F \\ T \end{pmatrix} = \begin{pmatrix} 3 & 17 \\ 6 & 9 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 19 \end{pmatrix} = \begin{pmatrix} 0 \\ 19 \end{pmatrix} = \begin{pmatrix} A \\ T \end{pmatrix}.$$

A Hill-titkosító, általános eset

- az $m = (m_1, m_2, \ldots m_d) \in M$ nyílt szöveg egy d hosszúságú blokkja egy lépésben lesz titkosítva,
- a *key* mátrix i, j elemét $k_{i,j}$-vel jelölve a titkosítás a következő:

$$\begin{pmatrix} k_{1,1} & k_{1,2} & \ldots & k_{1,d} \\ k_{2,1} & k_{2,2} & \ldots & k_{2,d} \\ \vdots & \vdots & & \vdots \\ k_{d,1} & k_{d,2} & \ldots & k_{d,d} \end{pmatrix} \cdot \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_d \end{pmatrix} \pmod p \longrightarrow \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_d \end{pmatrix}$$

- ahol p a bemeneti ábécé elemszáma: az első p darab nem negatív egész számmal végezzük $\pmod p$ szerint a számításokat,
- ha $p = 256$, akkor a titkosítást bájtok felett végezzük,
- a visszafejtéshez szükség lesz a $d \times d$ mátrix inverzére,
- a titkosítás során a bemenet egy lineáris transformációját határozzuk meg, a titkosító nem biztonságos.

A Hill-titkosító, kulcsgenerálás Python-ban

```
import sympy as sp
import math
import random

def keyGen(d, MOD):
    while True:
        key = sp.Matrix([[random.randint(0, MOD)
                for i in range(d)] for j in range(d)])
        det = key.det() % MOD
        if math.gcd(det, MOD) == 1: break
        i = random.randint(0, d-1)
        j = random.randint(0, d-1)
        key[i, j] = random.randint(0,MOD)
    return key

d, mod = 5, 256
key = keyGen(d, mod)
keyInv = key.inv_mod(mod)
print('Key: \n', key)
print('keyInv: \n', keyInv)
checkMat = (key * keyInv) % mod
print('Check: \n', checkMat)
```

Padding-olás, a nyílt szöveg kiegészítése

PKCS#7:

- ha N darab bájtot kell hozzáadni a nyílt szöveghez, ahhoz hogy az osztható legyen d-vel, akkor a nyílt szöveg végéhez hozzáfűzzük az N értékét N-szer,
- ha a nyílt szöveg osztható a blokkmérettel, akkor egy teljes blokkot adunk hozzá a nyílt szöveghez, ahol mindegyik bájt értéke d-vel lesz egyenlő.

Feladat: Egészítsük ki a plainTextSize bájtból álló plainText nyílt szöveget

```
rem = plainTextSize % d
N = d - rem
if rem != 0:
    plainText = plainText + bytes([N] * N)
else:
    plainText = plainText + bytes([d] * d)
```

Visszaalakításkor, feltételezve, hogy a visszafejtett bájtokat a tempPlainText tárolja a visszafejtett plainText a következő lesz:

```
N = tempPlainText[-1]
plainText = tempPlainText[:-N]
```

A Hill-titkosító - ismert nyílt-szöveg támadás, d = 2

- Ha ismertek az $m, \hat{m}, c, \hat{c}$, blokkok értékei, ahol fennáll, hogy az m rejtjelezett értéke, c és az $\hat{m}$ rejtjelezett értéke $\hat{c}$,
- akkor a következő rendszer megoldásával, megállapítható a titkosításhoz használt *key* kulcs,
- legyen $m = \begin{pmatrix} m_1 \\ m_2 \end{pmatrix}$, $c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$, $\hat{m} = \begin{pmatrix} \hat{m}_1 \\ \hat{m}_2 \end{pmatrix}$, $\hat{c} = \begin{pmatrix} \hat{c}_1 \\ \hat{c}_2 \end{pmatrix}$,
- felírható:

$$key \cdot \begin{pmatrix} m_1 & \hat{m}_1 \\ m_2 & \hat{m}_2 \end{pmatrix} = \begin{pmatrix} c_1 & \hat{c}_1 \\ c_2 & \hat{c}_2 \end{pmatrix},$$

$$key = \begin{pmatrix} c_1 & \hat{c}_1 \\ c_2 & \hat{c}_2 \end{pmatrix} \cdot \begin{pmatrix} m_1 & \hat{m}_1 \\ m_2 & \hat{m}_2 \end{pmatrix}^{-1}$$

- hasonló támadási stratégia alkalmazható nagyobb blokkméret esetében.

Klasszikus kriptográfiai rendszerek - feltörési módszerek

A következő feltörési módszerek mindegyike kivitelezhető, éppen ezért a klasszikus kriptorendszerek egyikét sem használják a gyakorlatban:

- az összes lehetséges kulcs kipróbálása (exhaustive key search): a kulcsok száma legtöbbször kicsi a mai rendszerek tár és feldolgozó kapacitáshoz képest,
- betűgyakoriság vizsgálat (ciphertext-only attack): a rejtjelezett szöveg ugyanolyan statisztikai tulajdonságokkal rendelkezik, mint a nyílt szöveg,
- ismert nyílt-szöveg támadás (known plaintext attack): ha rendelkezünk néhány betű rejtjelezett értékével, akkor meghatározható az eredeti szöveg, gyakran a kulcs is.

Modern kriptográfia, tervezési szempontok

- kriptográfiai primitívek:
  - álvéletlenszám generáló algoritmusok,
  - folyamtitkosítók,
  - blokktitkosítók,
  - hash függvények,
  - üzenet hitelesítő kódok,
  - a publikus kulcsú kriptográfia algoritmusai
- a kriptográfiai primitívek tervezése az elméleti kriptográfia területéhez tartozik,
- a kriptográfiai primitívek segítségével kriptográfiai protokollok építhetők,
- azért mert egy kriptográfiai primitív biztonságos, nem biztos, hogy a belőlük felépített kriptográfiai protokoll biztonságos,
- a kriptográfiai protokollok tervezése is az elméleti kriptográfia tárgyát képezik,

Modern kriptográfia, tervezési szempontok

- a kriptográfiai protokollok mellett a biztonsági rendszerek működését is protokollok alapján kell megadni,
- a szakirodalom mindkét esetben a protokoll megnevezéssel él, de a kettő között különbség van,
- a valós rendszerek biztonságát is protokollok/alkalmazások teszik lehetővé, ezek tervezése, működtetése azonban az adatbiztonság területéhez tartozik,
- példa, kriptográfiai protokoll:
  - végpontok közötti titkosítás
  - hitelesített kulcscsere
  - zero-knowledge protokollok, partner-hitelesítés, stb.
- példa, biztonsági protokoll: SSL/TLS, Signal, PGP

Modern kriptográfia, követelmények

- a kriptográfiában használt primitívek standard nyílt forráskódú algoritmusok kell legyenek: a standardizáló bizottság már korábban biztonsági szempontból megvizsgált és elfogadott,
- Kerckhoff elv (1883): egy kriptorendszer működési elve teljesen publikus kell legyen, egyedül a titkosításhoz használt kulcsot kell szigorúan titokban tartani, azt csak a kommunikáló felek ismerhetik,
- Claude Shannon (1949) amerikai matematikus: a kriptorendszereket, úgy kell tervezni, hogy szem előtt kell tartani, hogy egy támadó azt azonnal ki fogja ismerni,