Kriptográfia és Információbiztonság
2_Szt. előadás

MÁRTON Gyöngyvér

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

2023

Miről lesz szó?

- A titkosítás matematikai modelljei
- Összefoglaló, klasszikus titkos kulcsú rendszerek: Caesar, Affin, Hill
- Ismert nyílt-szöveg támadás (Known plaintext attack)
- Blokk titkosítók paddingolása
- Az NTL könyvtárcsomag
- Egy klasszikus blokk titkosító feltörése, feladatmegoldás: a Hill titkosító

A klasszikus titkosítás matematikai modellje

Legyen

- M a nyílt-szövegek egy véges halmaza,
- C a rejtjelezett-szövegek egy véges halmaza,
- K a kulcsok egy véges halmaza.

Három algoritmust értelmezünk:

- Gen, a kulcs-generáló algoritmus, meghatározza a key kulcsot,
- Enc_{key} a rejtjelező algoritmus, a key kulcs alapján, meghatározza az $m \in M$ nyílt-szöveg rejtjelezett értékét: $c \leftarrow Enc_{key}(m)$,
- Dec_{key} a visszafejtő algoritmus, a key kulcs alapján visszafejti a c rejtjelezett-szöveget: $m \leftarrow Dec_{key}(c)$.
- A rendszer helyessége érdekében megköveteljük: $Dec_{key}(Enc_{key}(m)) = m$, minden $m \in M$ esetében.

Számos klasszikus titkosítási rendszer létezik: Caesar, Vigenere, Palyfair, Hill, stb.

A titkos-kulcsú rendszerek matematikai modellje

- megnevezések: titkos-kulcsú rendszerek, szimmetrikus rendszerek (secret-key encryption, symmetric cryptography),
- jelölés: SKE-vel, a (K, M, C) halmaz-hármas felett értelmezzük,
- 3 algoritmust szükséges értelmezni:
  - Gen, a kulcs-generáló algoritmus, polinom idejű, véletlenszerű:
    $$key \xleftarrow{R} Gen(1^k),$$
    ahol $key \in K$ és k a rendszer biztonsági paramétere, legtöbb esetben a generált kulcs bit-hossza, és fennáll: $k \in \mathbb{Z}_{\geq 0}$
  - Enc_{key} a rejtjelező algoritmus, polinom idejű, véletlenszerű:
    $$c \xleftarrow{R} Enc_{key}(m),$$
  - a Dec_{key} a visszafejtő algoritmus, polinom idejű, determinisztikus:
    $$m \leftarrow Dec_{key}(c),$$
- Helyesség: $Dec_{key}(Enc_{key}(m)) = m$, minden $m \in M$ esetében.

A publikus-kulcsú titkosítók matematikai modellje

- megnevezések: publikus-kulcsú titkosítók, aszimmetrikus titkosítók (public-key encryption, asymmetric cryptography),
- jelölés: PKE-vel, a (K, M, C) halmaz-hármas felett értelmezzük,
- 3 algoritmust szükséges értelmezni:
  - Gen, a kulcs-generáló algoritmus, polinom idejű, véletlenszerű:

    $$(pk, sk) \xleftarrow{R} Gen(1^k),$$

    ahol $(pk, sk) \in K$ és k a rendszer biztonsági paramétere, legtöbb esetben a generált kulcs bit-hossza, és fennáll: $k \in \mathbb{Z}_{\geq 0}$
  - Enc_{pk} a rejtjelező algoritmus, polinom idejű, véletlenszerű:

    $$c \xleftarrow{R} Enc_{pk}(m),$$

  - a Dec_{sk} a visszafejtő algoritmus, polinom idejű, determinisztikus:

    $$m \leftarrow Dec_{sk}(c),$$

- Helyesség: $Dec_{sk}(Enc_{pk}(m)) = m$, minden $m \in M$ esetében.

A későbbi előadásokban a publikus kulcsú titkosítókra még részletesen visszatérünk.

A titkos-kulcsú rendszerek jellemzők

- nagy adathalmaz titkosítására alkalmasak,
- biztonságuk számítástechnikai szempontból elfogadható.
- nincs megoldva a felek közötti kulcs-csere, ezt a nyilvános kulcsú kriptográfia végzi,
- nincs megoldva a felek hitelesítése, ezt a nyilvános kulcsú kriptográfia végzi,
- két nagy csoportra oszthatóak:
  - folyam-titkosító rendszerek: a nyílt szöveget bájtonként titkosítják,
  - blokk-titkosító rendszerek: a nyílt szöveget bájt blokkonként titkosítják

A Caesar rejtjelező

- folyam titkosító,
- az üzenetek halmaza: $M = C = \{0, 1, \ldots, 25\}^*$, az angol ábécé 26 betűjének megfelelő számkód,
- a kulcsok halmaza: $K = \{0, 1, \ldots, 25\}$,
- kulcsgenerálás *Gen*: kiválasztunk egy $key \in K$ értéket,
- titkosítás: $Enc_{key}(m) = (m + key) \pmod{26} \rightarrow c$, ahol $m \in M$,
- visszafejtés: $Dec_{key}(c) = (c + 26 - key) \pmod{26} \rightarrow m$.

Az Affin rejtjelezés

- folyam titkosító,
- az üzenetek halmaza: $M = C = \{0, 1, \ldots, 25\}^*$, az angol ábécé 26 betűjének megfelelő számkód,
- a kulcsok halmaza: $K = \{(a, b) \in \mathbb{Z}_{26}$, úgy hogy $gcd(a, 26) = 1\}$,
- kulcsgenerálás *Gen*: kiválasztunk egy $key = (a, b) \in K$ értéket,
- $Enc_{(a,b)}(m) = (a \cdot m + b) \pmod{26} \Rightarrow c$,
- $Dec_{(a,b)}(c) = a^{-1} \cdot (c + 26 - b) \pmod{26}$, ahol $a \cdot a^{-1} = 1 \pmod{26}$.

A Hill rejtjelezés

- blokk titkosító: egyszerre d darab blokkot titkosít
- az üzenetek halmaza: $M = C = \mathbb{Z}_{26}^d$,
- *Gen*: a kulcs egy $d \times d$-es mátrix, elemei $\in \mathbb{Z}_{26}$ és fenn kell álljon, hogy $gcd(det_{key}, 26) = 1$
- padding: ha a nyílt szöveg nem osztható a blokk mérettel, akkor szükséges kiegészíteni a nyílt szöveget annyi bájttal, hogy osztható legyen,
- $Enc_{key}(m) : c = key \cdot m$

$$(c_1, c_2, \ldots c_d) = \begin{pmatrix} k_{1,1} & k_{1,2} & \ldots & k_{1,d} \\ k_{2,1} & k_{2,2} & \ldots & k_{2,d} \\ \vdots & \vdots & & \vdots \\ k_{d,1} & k_{d,2} & \ldots & k_{d,d} \end{pmatrix} \cdot \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_d \end{pmatrix}$$

- $Dec_{key}(c) = key^{-1} \cdot c$

Padding-olás, a nyílt szöveg kiegészítése

Padding: a blokk titkosítók esetében ha a nyílt szöveg nem osztható a d blokk mérettel, akkor a nyílt szöveg elejét vagy végét ki kell egészíteni valamely standard eljárás szerint.

- PKCS#7: bájtokkal egészítjük ki a nyílt szöveg végét
  - ha N darab bájtot kell hozzáadni, akkor az N értékét adjuk hozzá N-szer
  - ha osztható a nyílt szöveg a blokkmérettel, akkor is kiegészítésre kerül a nyílt szöveg: egy teljes blokknyi bájtot adunk hozzá, ahol mindegyik bájt értéke d lesz

Példa:

Egy plainTextSize bájtból álló nyílt szöveg paddingolása C/C++-ban:

```
unsigned char padV = d - plainTextSize % d;
unsigned char* plainText = new unsigned char[plaintextSize + padV];
inF.read((char*)plainText, plaintextSize);
for (int i = 0; i < padV; i++)
    plainText[plaintextSize + i] = padV;
```

A visszafejtésnél az utolsó blokkot külön dolgozzuk fel:

```
unsigned char padV = cipherTextLastB[d - 1];
outF.write((char*)cipherTextLastB, d - padV);
```

Feltörési módszerek

A Caesar, Affin, Hill esetében a következő feltőrési módszerek mindegyike működik:

- az összes lehetséges kulcs kipróbálása (exhaustive key search)
- betűgyakoriság vizsgálat (ciphertext-only attack),
- ismert nyílt-szöveg támadás (known plaintext attack): ha rendelkezünk néhány betű rejtjelezett értékével, akkor meghatározható az eredeti szöveg, gyakran a kulcs is.

Az NTL könyvtárcsomag, létrehozás, Visual Studio

NTL - A Library for doing Number Theory - Számelméleti műveleteket kezelő könyvtár

- elérhetőség:

  http://www.shoup.net/ntl/download.html

- Victor Shoup által C++-ban fejlesztett könyvtár, nagy teljesítményű, hordozható,
- tetszőleges nagyságrendű számokkal, tetszőleges precizitású való számokkal végezhetők a matematikai műveletek,
- egész számok, illetve véges testek felett vektorokkal, mátrixokkal polinomokkal lehet algebrai műveleteket végezni,
- 1990-ben kezdődött a fejlesztése, a nyilvánosság elé az 1.0 verzió 1997-ben került,
- az utolsó verzió 2021 júniusában jelent meg,
- a GNU LGPL 2.1 verzió által előírt feltételek mellett szabadon használható szotver.

Az NTL könyvtárcsomag, létrehozás, Visual Studio

- az NTL letöltése után csomagoljuk ki pl. a `...Projects\WinNTL` mappába,
- hozzunk létre egy új projektet: `New` → `Project` → `Project From Existing Code`,
- adjunk egy nevet a projektnek, legyen ez `NTLLib`,
- adjuk meg a projekt helyét: `...Projects\WinNTL\src`
- válasszuk ki a projekt típusát: `Static Library (LIB) project`
- az `Include search path`-nál adjuk meg a header állományok elérési útvonalát: `...Projects\WinNTL\include`
- A projekt `Properties/General` pontnál cseréljük át az sdk verziót 8.1-ről 10.0....-ra
- A projekt `Properties/Code Generation` pontnál állítsuk `Disable Security Check`-re a `Security Check` beállítást
- a `Bulid\Solution` parancs megadásával létrejön a `...Projects\NTLLib\src\Debug` mappában az `NTLLib.lib` állomány, amit a további projekteknél kell használni.

Az NTL könyvtárcsomag, használat, Visual Studio

- Hozzunk létre egy új projektet: `New` → `Project` → `Empty Project`,
- adjunk egy nevet a projektnek, legyen ez `Labor`,
- a `Labor` project-hez az `Add Existing Item` menüpont segítségével adjuk hozzá a megfelelő `Debug` mappából az `NTLLib.lib` állományt,
- a `Project/NTLLib/Properties` menüpontnál az `Additional Include Directories`-nél adjuk meg a header állományok elérési útvonalát: `...\WinNTL\include`,
- a `Project Properties` → `C/C++` → `Code Generation` állítsuk be a `Disable Security Check`-et

Hill - known plaintext attack

Feladat: A `cryptHill_KP` állomány Hill módszerrel volt rejtjelezve, ahol a blokk méret d = 3 és a titkosítást a bájtok felett végezték. Tudva azt, hogy az "3, 6, 4"-nek "69, 130, 90", "4, 3, 5,"-nek "195, 207, 23" és " 7, 6, 7"-nek "35, 214, 133" a rejtjele határozzuk meg a kulcsot és az eredeti jpeg állományt.

Legyenek:

$$\begin{pmatrix} m_{11}=3 \\ m_{12}=6 \\ m_{13}=4 \end{pmatrix} \begin{pmatrix} c_{11}=69 \\ c_{12}=130 \\ c_{13}=90 \end{pmatrix} \begin{pmatrix} m_{21}=4 \\ m_{22}=3 \\ m_{23}=5 \end{pmatrix} \begin{pmatrix} c_{21}=195 \\ c_{22}=207 \\ c_{23}=23 \end{pmatrix} \begin{pmatrix} m_{31}=7 \\ m_{32}=6 \\ m_{33}=7 \end{pmatrix} \begin{pmatrix} c_{31}=35 \\ c_{32}=214 \\ c_{33}=133 \end{pmatrix}$$

Ekkor felírható:

$$key \cdot \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

$$key = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} \cdot \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}^{-1}$$

Forráskód:

- `Hill_KP.cpp`
- `Hill_KP.py`