Kriptográfia és Információbiztonság
3. előadás

MÁRTON Gyöngyvér

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

2025

Miről volt szó az elmúlt előadáson?

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

Miről lesz szó?

- tökéletes biztonság
- számítási biztonság
- az OTP titkosítási rendszer
- 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.

Tökéletes biztonság

Shannon biztonság-elmélete, 1949 ⇒ információelméleti biztonság

- informálisan: egy titkosító tökéletesen biztonságos (perfectly secret), ha egy adott rejtjelezett szöveg egyforma valószínűséggel lehet bármelyik nyílt szöveg rejtjelezett értéke,
- a kulcs ismeretének hiányában, csak a titkosított tartalom birtokában ne lehessen eldönteni, hogy melyik nyílt szöveg került titkosításra,
- ha egy titkosító rendszer tökéletes biztonságú, akkor a kulcsok halmaza nagyobb vagy egyenlő elemszámú, mint a nyílt szövegek halmaza,
- a gyakorlatban szinte lehetetlen olyan titkosító rendszert létrehozni, mely tökéletes biztonságú,
- más biztonság értelmezésre van szükség ⇒ számítási biztonság (computational security)

Számítási biztonság

A számítási biztonság (computational security) feladja a tökéletes biztonságot, és feltételezi, hogy:

- a támadó számítási kapacitása (idő, tár) korlátos,
- a titkosító és rejtjelező algoritmusok nem tetszőleges függvények, hanem hatékony algoritmusok,
- informálisan: a számítási biztonság egy titkosító esetében azt jelenti, hogy bármely polinom idejű, véletlenszerű algoritmus csak nagyon kis/elhanyagolható valószínűséggel tudja feltörni a rendszert.

Számítási biztonság

Példa: ha egy támadó egy titkosító algoritmus esetében az összes kulcs kipróbálásának módszerét választja, ahol a kulcsméret k, azaz a kulcsok száma 2^k, akkor a következő állapítható meg:

- ha egy kulcs teszteléséhez c időegységre van szükség, összesen pedig K időegység áll a támadás kivitelezésére, akkor $\frac{K}{c}$ darab kulcs vizsgálható meg,
- ha $\frac{K}{c} \ll 2^k$, akkor a kulcs meghatározásának valószínűsége $\dfrac{K}{c2^k}$,
- ha $c = 1$ és 2GHz CPU gépünk van, amely egy évig számol, akkor
$$K = 365 \cdot 24 \cdot 60 \cdot 60 \cdot 2 \cdot 10^9 = 63072000000000000 < 2^{56},$$
ami azt jelenti, hogy kb. 2^{55} darab kulcs vizsgálható meg egy év alatt,
- ha a kulcsméret 128 bit, akkor a kulcs meghatározásának valószínűsége: $\frac{2^{55}}{2^{128}} = 2^{-73}$!!!

Az OTP rendszer

- OTP - One-time Pad rendszer, 1917, Vernam rendszerének egy változata,
- $M = C = K = \mathbb{Z}_2^k$,
- legyen $m = (m_1, \ldots, m_k) \in M$, $c = (c_1, \ldots, c_k) \in C$, $key = (key_1, \ldots, key_k) \in K$,
- $Gen \xrightarrow{R} key$, (egyenletes eloszlás, véletlenszerű, egyszer használt)
- $Enc_{key}(m) : (m_1 \oplus key_1, \ldots m_k \oplus key_k) \rightarrow c$,
- $Dec_{key}(c) : (c_1 \oplus key_1, \ldots c_k \oplus key_k) \rightarrow m$.
- az $\oplus$ az XOR műveletet jelöli, ami tulajdonképpen mod 2 szerinti összeadás,
- az $\oplus$ művelet tulajdonságai:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

$x \oplus y = y \oplus x$,
$x \oplus (y \oplus z) = (x \oplus y) \oplus z$,
$x \oplus 0 = x, x \oplus x = 0$.

- a fenti tulajdonságok alapján a titkosítási folyamat helyessége egyértelmű: $Dec_{key}(Enc_{key}(m)) = Dec_{key}(m \oplus key) = (m \oplus key) \oplus key = m$.

Az OTP rendszer

Példa, titkosításra, ahol a kulcs és a nyílt szöveg is 7 bites:

$$\begin{array}{ccccccccc} m & = & (1 & 1 & 0 & 0 & 1 & 0 & 0) & \oplus \\ \color{red}{key} & = & (0 & 0 & 1 & 0 & 1 & 0 & 1) & \\ \hline c & = & (1 & 1 & 1 & 0 & 0 & 0 & 1) & \end{array}$$

Példa, visszafejtésre, ahol a kulcs és a rejtjelezett szöveg is 7 bites:

$$\begin{array}{ccccccccc} c & = & (1 & 1 & 1 & 0 & 0 & 0 & 1) & \oplus \\ \color{red}{key} & = & (0 & 0 & 1 & 0 & 1 & 0 & 1) & \\ \hline m & = & (1 & 1 & 0 & 0 & 1 & 0 & 0) & \end{array}$$

Az OTP rendszer

1. tétel

Az OTP titkosító rendszer tökéletes biztonságú.

Megjegyzések:

- nagyon gyors a titkosítás és visszafejtés folyamata,
- nem lehet észrevenni, ha módosult a rejtjelezett-szöveg, ezt üzenet-hitelesítő kódok alkalmazásával lehet megoldani,
- a gyakorlatban felmerülő problémák:
  - a generált kulcsot tilos többször felhasználni: ha ismert egy nyílt-szöveg és titkosított szöveg pár, akkor meg lehet határozni a kulcsot,
  - minden egyes titkosításhoz más kulcsot kell használni, amely független a korábbi kulcsoktól,
  - a generált kulcs ugyanolyan hosszú kell legyen, mint a nyílt szöveg,
- gyakorlatban egy véletlenszerűen előállított rövid (pár bájtos) kulcs alapján véletlenszerűen generálnak tetszőleges hosszúságú kulcsfolyamnak (key stream) nevezett bájt/bit szekvenciát, amelyet XOR műveletet alkalmazva adnak hozzá a nyílt szöveghez.

A modern titkos kulcsú titkosítók a gyakorlatban

- a kulcsok, az üzenetek a titkosított szövegek bájt illetve bit szekvenciák
- a kulcs mérete legalább 128 bit (16 bájt), az üzenetek, pedig lehetnek 1GB (gigabájtnyi) videó file-ok, 10 MB zene file-ok, 1KB email adat, vagy egyetlen bitnyi adat ami megfelel pl. egy szavazási rendszerben az `igen` vagy `nem` szavazati értéknek.
- a polinom futásidejű algoritmus hatékony algoritmust jelent, például elvárjuk, hogy 1 GB adat titkosítását 1 perc alatt végezze el a rendszer
- a kulcsok, az üzenetek, a titkosított adatok különböző típusú matematikai objektumok is lehetnek, pl. értékpárok, mátrixok, polinomok, görbék pontjai, stb. Minden objektum esetben megadható kell legyen az, hogy hogyan ábrázoljuk, alakítjuk át ezeket bit, illetve bájt szekvenciákká.

A modern titkos kulcsú titkosítók osztályozása

- nagy adathalmaz titkosítására alkalmasak,
- biztonságuk számítási szempontból elfogadható kell legyen,
- nincs megoldva a felek közötti kulcscsere, ezt a publikus kulcsú kriptográfia végzi,
- nincs megoldva a felek hitelesítése, ezt a publikus kulcsú kriptográfia végzi,
- nincs megoldva az üzenetek sértetlenségének problémája, ezt MAC-el oldják meg, vagy hitelesített titkosítással (authenticated encryption)
- két nagy csoportra oszthatóak:
  - folyamtitkosító rendszerek (stream cipher),
  - blokktitkosító rendszerek (block cipher).

Folyamtitkosító rendszerek

- véletlenszerűen generálnak egy bitsort: a kulcsfolyamot, amelyet legtöbbször a $\oplus$ művelettel hozzáadnak a nyílt szöveghez,
- a kulcsfolyamot rekurzívan, egy belső állapot-érték és egy kezdeti rövid kulcsérték (seed, key) alapján generálják,
- a kulcsfolyamot egyetlenegyszer használják, és a nyílt szöveg bájtjainak a feldolgozása közben változhat,
- a rendszer biztonságát a kulcsfolyamot generáló algoritmus határozza meg $\Rightarrow$ álvéletlen-szám generáló algoritmusok (pseudo-random number generators)
- megkülönböztetünk szinkron és önszinkronizáló folyamtitkosítókat
- nem minden ál-véletlenszám generátor alkalmas a kriptográfiában,
  - pl: az $x_n = a \cdot x_{n-1} + b \pmod p$ lineáris kongruenciával értelmezett generátor sem alkalmas,

Folyamtitkosító rendszerek

- implementációjuk általában könnyű
- hardver-be építve nagyon gyorsak, ezért legtöbbször hardver szintjén implementálják őket
- szoftver szintjén nem mindig érik el a megfelelő gyorsaságot
- leginkább akkor használják, amikor nem lehet tudni, hogy milyen hosszú a nyílt szöveg, pl wireless kommunikáció
- katonai környezetben gyakran használják: a kulcsfolyam előállítást külön eszköz végzi, ez jól védett, a XOR műveletek elvégzésére pedig más eszközt használnak, amelynek biztonsága már nem olyan fontos
- a korábbi rendszerek biztonsága nem megfelelő, pl. az RC4, A5/1 kezdetben titkosak voltak, miután publikussá váltak számos gyengeséget fedeztek fel bennük,
- biztonságos folyamtitkosító: Salsa20, illetve a változata ChaCha20.

Valós rendszerek biztonsági problémái

Közvetett vagy közvetlen módon felmerül a kulcs többszöri felhasználása:

- Venona projekt: célja az orosz titkosszolgálat üzeneteinek a megfejtése volt, 1943-1980 között. Az amerikai hadsereg kb. 3000 üzenetet fejtett meg ebben a periódusban mert többször birtokába került két-két olyan titkosított érték, ahol mindkét esetben ugyanazt a kulcsot használták a titkosítás során.
- MS-PPTP (windows NT): szerver és kliensek közti kommunikáció esete, volt egy megosztott titkos kulcs, ahol a szerver és kliens oldalon is ugyanazzal a kulcsfolyammal végezték a titkosítást $\Rightarrow$ a kulcs többszöri felhasználása. Megoldás: kulcspárt kell használni, amelyet a szerver és a kliens is ismer, egyik érték a szerver, másik érték a kliens oldali titkosításhoz.
- 802.11b WEP, több probléma, az egyik: egy 24 bites inicializáló érték (IV) alkalmazása lehetővé tette a WEP feltörését, mert túl kicsi volt az IV bithossza: $2^{24} \sim 16$ millio IV érték után az IV-k ciklikusan ismétlődtek.

Az RC4

- 1987-ben tervezte Ron Rivest, kezdetben nem volt publikus,
- szinkron folyamtitkosító,
- 1994-ben ismeretlenül felkerült a Cyberpunk levelező-listájára,
- nagyon népszerű titkosító volt, hálózati forgalom titkosítását végezték vele: a HTTPS-ben (Hypertext Transfer Protocol Secure), a WEP (Wired Equivalent Privacy)-ben és számos más protokollban használták,
- a 64 bites architektúrájú gépeken lassú, mert eredetileg 8-bites processzorokra tervezték,
- ma már nem biztonságos, több sikeres támadás érte, és kimenetét meg lehet különböztetni egy véletlenszerűen generált bájtsorozattól,
- a Internet Engineering Task Force (IETF) 2015-től tiltja a használatát

Az RC4

Elvi működése:

- egy 128 bites kezdeti érték alapján összekeverjük a 256 fajta lehetséges bájtértékeket, és így előállítunk egy 256 elemű bájttömböt, amely a generátor alapállapotának fog megfelelni,
- a kulcsfolyam egy bájtja a 256 elemű bájttömb egy "véletlenszerűen" kiválasztott bájtja lesz,
- a kulcsfolyam bájtjait ciklikusan állítjuk elő, amelyet $\oplus$ művelettel adunk hozzá a nyílt szöveghez

RC4, inicializálási szakasz

```
def rc4_init(key):
    key_length = len(key)
    S = [i for i in range(256)]
    j = 0
    for i in range(256):
        j = (j + key[i % key_length] + S[i]) & 255
        S[i], S[j] = S[j], S[i]
    return S
```

- az S kezdetben megegyezik az identikus permutációval,
- majd megcseréljük az S i és j sorszámú elemeit
- az i lineárisan fut végig az index értékeken, a j pedig ugrálva, ezzel a lehetséges bájtok egy permutációját kapjuk,
- a 256-al történő maradékos osztás 255-tel való & (AND) művelettel van helyettesítve

RC4, kulcsfolyam-generálás

```
def rc4_output(S):
    i , j = 0, 0
    while True:
        i = (i + 1) & 255
        j = (j + S[i]) & 255
        S[i], S[j] = S[j], S[i]
        yield S[(S[i] + S[j]) & 255]
```

- az S egy pozíciójáról kiválasztott elem lesz az `rc4_output` pszeudorandom kimeneti értéke,
- biztosítva van az S folyamatos keverése, módosítása,
- szinkrón titkosító:
  - minden lépésben sor kerül az S i és j sorszámú elemeinek a cseréjére,
  - a belső állapot módosul a kulcsfolyam egy értékének a meghatározásakor,
- az i lineárisan fut végig az index értékeken, a j pedig ugrálva.

RC4, string titkosítás/visszafejtés

```
def rc4_main():

    key = 'cryptographykey1'.encode()
    cryptStr = '911c72991f48aa4b8307a5922b5bac9916e04fe8aa5d77cef8ce0a79d0\
            344d95b2c2e14cea6bf80b25856772846c2ce3bad3f209f9f6041e189b058\
            640bc87341db1a163879d27a3cf7c472052886af8c7432d3bf9c4e78148bb\
            48f3149bc88638abbffd098e9237131be2df5dab6fa3f881c36d5e1a42448\
            a7f67e6cfc12e223b52fdf7b2f45c173f8a9f887a08952cf18bb36bb18724\
            4da260428cff5fdb4bbc874b035fd01da038f9ecd98f88144c1915'
    cryptStr = bytes.fromhex(cryptStr)
    S = rc4_init(key)
    keyStream = rc4_output(S)

    decryptStr = bytearray()
    for b in cryptStr:
         decryptStr.append(b ^ next(keyStream))
    print(decryptStr.decode())
```

Az RC4, gyengeségek

- annak a valószínűsége, hogy a második bájt 0 lesz $\frac{1}{256}$ kellene legyen, de ez nem így van, mert ez $\frac{2}{256}$ lesz,
  - hasonló a helyzet az első 256 bájt esetében is,
  - ez alapján ha adott egy nyílt szöveg 2^{30} különböző kulccsal való titkosított értéke, akkor annak a valószínűsége, hogy a nyílt szöveg első 128 bájtját meg tudjuk határozni közel van 1-hez,
  - gyakorlatban:
    - egy titkos cookie egy üzenet első néhány bájtjába van beágyazva,
    - ahányszor egy böngésző csatlakozik az *áldozat* webszervehez ez a süti különböző kulccsal mindig újra rejtjelezve lesz,
    - Javascript használatával a támadó arra kényszerítheti a felhasználó böngészőjét, hogy ismételten csatlakozzon a támadó céloldalához, így könnyen hozzájuthat a 2^{30} lehetséges rejtjelezett szöveghez
  - erre a a megoldás: a kulcsfolyam első 1024 bájtját nem használták fel a titkosítás során,
- egyéb támadások, gyengeségek: a 00 szekvencia gyakrabban fordul elő, mint a 01, 10, 11 szekvencia,

Visszacsatolt, léptető regiszteres titkosítás, linear feedback shift register - LFSR

- Számos rendszer esetében használták, használják pl.:
  - DVD titkosítása esetén a rendszer neve CSS (content scrambling system)
  - GSM titkosítás esetén a rendszer neve A5/1, A5/2
  - Bluetooth titkosítás esetén a rendszer neve E0
- hardvare szintjén könnyedén és hatékonyan implementálható
- a kulcsfolyamot egy lineáris összefüggés alapján számolják, kiindulva a $key = (z_1, z_2, \ldots, z_m)$ értékekből:

$$z_{i+m} = c_0 z_i + c_1 z_{i+1} + \ldots c_{m-1} z_{i+m-1} \pmod 2,$$

  ahol $i \geq 1$ és $c_0, c_1, \ldots, c_{m-1}$ rögzített konstansok
- a $c_0, c_1, \ldots, c_{m-1}$ konstansok helyes választása esetén a kulcsfolyam periódusa maximális lesz: $2^m - 1$.
- az alkalmazott műveletek lineárisak, ezért a rendszer feltörhető,
- több LFSR kombinálásával, illetve egy nemlineáris komponens bevezetésével elérhető egy bizonyos biztonság.

Az LFSR támadása, feltörhetősége

Példa 16 bites LFSR:

$$z_{i+16} = z_i + z_{i+2} + z_{i+3} + z_{i+5} \pmod 2,$$

- Ha $key = (z_1, z_2, \ldots, z_{16}) = 1010\ 1100\ 1110\ 0001$,
  akkor a kulcsfolyam első bitjei a pirossal kijelölt értékek lesznek, ahol
  $z_{17} = z_1 + z_3 + z_4 + z_6$:

0101	1001	1100	0011	$1+1+0+1=1$
1011	0011	1000	0111	$0+0+1+0=1$
0110	0111	0000	1111	$1+1+1+0=1$
1100	1110	0001	1110	$0+1+0+1=0$
1001	1100	0011	1100	$1+0+0+1=0$
0011	1000	0111	1001	$1+0+1+1=1$

- a kulcsfolyam: 1010 1100 1110 0001 1110 01
- $2^{16} - 1$ lesz a generátor periódusa

Az LFSR támadása, feltörhetősége

A támadó, ha tudja, hogy a titkosító 5-bites LFSR-titkosítást alkalmazott, illetve a következő bitszekvenciának

$$0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1$$

ismeri a titkosított értékét:

$$0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1$$

akkor a fenti két bitsorozat xor-olásával, megtudja határozni a kulcsfolyam első bitjeit:

$$0\ 1\ 0\ 1\ 1\ \color{red}{0\ 1\ 0\ 0\ 0}$$

ez után, pedig felírva és megoldva a következő lineáris egyenletet, megtudja határozni az LFSR-ben alkalmazott lineáris összefüggést, azaz megtudja határozni a konstansokat:

$$(\color{red}{0,1,0,0,0}) = (c_0, c_1, c_2, c_3, c_4) \cdot \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Az LFSR támadása, feltörhetősége

Meghatározható:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

ahonnan:

$$(c_0, c_1, c_2, c_3, c_4) = (0, 1, 0, 0, 0) \cdot \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} = (1, 0, 1, 1, 1)$$

Tehát a kulcsfolyam kigenerálásához alkalmazott lineáris összefüggés a következő: $z_{i+5} = (z_i + z_{i+2} + z_{i+3} + z_{i+4}) \pmod 2$.

Az LFSR támadása, feltörhetősége

Általános esetben:

- ha ismerjük hogy a $(x_1, x_2, \ldots, x_n)$ nyílt szövegnek $(y_1, y_2, \ldots, y_n)$ a rejtjele, illetve ismerjük az m értékét akkor meghatározható az alkalmazott lineáris összefüggés
- meghatározzuk: $(x_1, x_2, \ldots, x_n) \bigoplus (y_1, y_2, \ldots, y_n) \Rightarrow (z_1, z_2, \ldots, z_n)$
- ha $n \geq 2m$, akkor a támadó megtudja határozni a $(c_0, c_1, \ldots, c_{m-1})$ értékeket a következő módon:

$$(z_{m+1}, z_{m+2}, \ldots, z_{2m}) = (c_0, c_1, \ldots, c_{m-1}) \cdot \begin{pmatrix} z_1 & z_2 & \cdots & z_m \\ z_2 & z_3 & \cdots & z_{m+1} \\ \vdots & \vdots & & \vdots \\ z_m & z_{m+1} & \cdots & z_{2m-1} \end{pmatrix}$$

- és meghatározható:

$$(c_0, c_1, \ldots, c_{m-1}) = (z_{m+1}, z_{m+2}, \ldots, z_{2m}) \cdot \begin{pmatrix} z_1 & z_2 & \cdots & z_m \\ z_2 & z_3 & \cdots & z_{m+1} \\ \vdots & \vdots & & \vdots \\ z_m & z_{m+1} & \cdots & z_{2m-1} \end{pmatrix}^{-1}$$