Kriptográfia és Információbiztonság
3. előadás

MÁRTON Gyöngyvér

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

2024

Miről volt szó az elmúlt előadáson?

- Klasszikus kriptográfiai rendszerek
  - eltolásos rejtjelezések: Caesar-titkosító, Keyword Caesar
  - helyettesítő rejtjelezés: affin-titkosító
  - blokk titkosítók: Hill titkosító,
  - feltörési módszerek,
  - matematikai modell
- a modern kriptográfia: tervezési szempontok, követelmények
- tökéletes biztonság
- számítástechnikai biztonság
- az OTP titkosítási rendszer

Miről lesz szó?

- 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.

A modern titkos kulcsú titkosítók matematikai modellje

Titkos kulcsú titkosítók, egyéb megnevezések: szimmetrikus titkosítók, szimmetrikus kriptográfia (secret-key encryption, symmetric cryptography). Három algoritmust szükséges értelmezni, ahol K a kulcsok, M az üzenetek halmaza:

- 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 bithossza, és fennáll: $k \in \mathbb{Z}_{\geq 0}$

- $Enc(key, \cdot)$ a rejtjelező algoritmus, polinom idejű, véletlenszerű:

$$c \xleftarrow{R} Enc(key, m),$$

- a $Dec(key, \cdot)$ a visszafejtő algoritmus, polinom idejű, determinisztikus:

$$m \leftarrow Dec(key, c),$$

A helyesség fennáll, ha minden $m \in M$ esetében:

$$Dec(key, (Enc(key, m)) = m.$$

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 számítástechnikai szempontból elfogadható.
- 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,
- szinkron folyamtitkosítók (synchronous stream ciphers): a kulcsfolyam a belső állapotértéktől és a kezdeti kulcsértéktől függ
- önszinkronizáló folyamtitkosítók/aszinkron folyamtitkosítók (self-synchronizing stream ciphers): a kulcsfolyam értékének a meghatározásánál figyelembe veszik a korábbi titkosított szöveg bitjeit is

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.

Random számok

- az adatbiztonságban számos helyen van szükség véletlenszerűen előállított bájtokra/bitekre (random bits/random numbers),
- különbség van a valódi (true random) és az ál (pseudo random) véletlen számok között,
- valódi random biteket manuálisan, illetve hardver eszközök segítségével lehet generálni, viszonylag lassúak, és megbízhatóság szempontjából számos kritika éri őket,
- a pseudorandom biteket/számokat determinisztikus algoritmusokkal generálják, amelyeknek bemenetként meg kell adni egy kezdeti, rövid *seed*-nek nevezett random értéket, és amelyek eredményként random biteknek/számoknak tűnő hosszú szekvenciát adnak,
- a megfelelő biztonság eléréséhez szükséges megadni a random bit generáló algoritmusok matematikai definícióit.

Random számok

1. értelmezés

Egy valódi random bit generátor egy olyan eljárás/egység, amely egy random bit szekvenciát generál, ahol a megfelelő $X_1, X_2, \ldots$ bináris valószínűségi változók szekvenciája a következő tulajdonsággal rendelkezik:

- $Pr[X_n = 0] = Pr[X_n = 1] = \frac{1}{2}$, *minden* $n \in \mathbb{N}$ *értékre*
- $X_1, X_2, \ldots, X_n$ *egymástól független valószínűségi változók, minden* $n \in \mathbb{N}$*-re*

2. értelmezés

Egy G pseudorandom bit generátor egy olyan polinom idejű, determinisztikus algoritmus, amely

- *egy kezdeti* $s \in \{0,1\}^n$*, seed alapján egy* $G(s) \in \{0,1\}^{l(n)}$ *kimeneti bitszekvenciát állít elő, ahol* $l(\cdot)$ *egy polinom, és* $l(n) > n$ *bármely* $n \in \mathbb{N}$*-re*
- *esetében polinom időben nem állapítható meg különbség a kimeneti* $G(s)$ *bitszekvencia és egy egyenletes eloszlású véletlenszerűen generált bitszekvencia között.*

Random számok

- egy pseudorandom generátor által előállított bitszekvencia soha nem lehet egyenletes eloszlású, mert $l(n) > n$, számos olyan $l(n)$ hosszúságú bitszekvencia létezik, amelyek nem szerepelnek a G bemenetében
- egy pseudorandom generátor szerkesztése nem egy triviális feladat
- next-bit test: egy támadó ismerve egy generátor első i darab kimeneti értékét, 50%-nál nagyobb valószínűséggel nem határozhatja meg polinom időben az $(i+1)$-ik bit értékét
- a pseudorandom generátorok számára a NIST készített egy statisztikai tesztcsomagot (*link*): ha egy generátor megfelel mindegyik tesztnek az még nem bizonyítja, hogy a generátor pszeudorandom, ellenben ha egy teszten nem megy át, akkor biztosan nem pszeudorandom generátor

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

Elvi működése:

- egy 128 bites kezdeti érték alapján összekeveri a 256 fajta lehetséges bájtértékeket, és így előállít 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 állítja elő, amit $\oplus$ művelettel ad hozzá a nyílt szöveghez

RC4, inicializálási szakasz

```
void rc4_init(unsigned char *key, unsigned int key_length) {
    for (i = 0; i < 256; i++)
        S[i] = i;

    for (i = j = 0; i < 256; i++) {
        j = (j + key[i % key_length] + S[i]) & 255;
        swap(S, i, j);
    }
    i = j = 0;
}
```

- a `key` a kulcs bájtjait tartalmazza,
- az `i`, `j` és az `S` globális változók, ahol `S` kezdetben megegyezik az identikus permutációval,
- a `swap` felcseréli az `S` megfelelő indexű elemeit, ahol 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

```
unsigned char rc4_output() {
    i = (i + 1) & 255;
    j = (j + S[i]) & 255;
    swap(S, i, j);

    return S[(S[i] + S[j]) & 255];
}
```

- az S egy pozíciójáról kiválasztott elem lesz az rc4_output pszeudorandom kimeneti értéke,
- a swap biztosítja az S folyamatos keverését, módosítását,
- minden lépésben sor kerül az S tömb megfelelő két elemének a cseréjére, ezért szinkrón titkosító: 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

```
int main() {
    int k, output_length;
    string key = "myKey in 2020";
    int l = key.length();

    //encryption
    string plainText = "Cryptography labor 2022";
    int pL = plainText.length();
    string cryptText = "";
    rc4_init((unsigned char*)key.c_str(), l);
    for (int i = 0; i < pL; ++i)
        cryptText += plainText[i] ^ rc4_output();
    cout << "encrypted text: " << cryptText << endl << endl;

    cout << "encrypted text in HEX: ";
    for (i = 0; i < pL; ++i)
        cout << hex << (0xFF & cryptText[i]) << " ";
    cout << endl << endl;

    //decryption
    int cL = cryptText.length();
    string decryptText = "";
    rc4_init((unsigned char*)key.c_str(), l);
    for (int i = 0; i < cL; ++i)
        decryptText += cryptText[i] ^ rc4_output();
    cout << "decrypted text: " << decryptText << endl << endl;
}
```

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} random kulcsal való titkosított értéke, akkor annak a valószínűsége, hogy a nyílt szöveg első 128 bájtját meg tudjuk határozni közel van 1-hez,
- gyakorlatban ez könnyen végrehajtható a weben:
  - 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-et 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 ne használjuk fel a titkosítás során

Az RC4, gyengeségek

Egyéb támadások, gyengeségek:

- a 00 szekvencia gyakrabban fordul elő, mint a 01, 10, 11 szekvencia,
- a kulcs többszöri felhasználásának problémája:
  - a WEP estében, hogy elkerüljék a kulcs többszöri felhasználását minden üzenet esetében egy 24 bites IV értéket társítottak a kulcshoz, ($2^{24} = 16777216 < 17 \cdot 10^6$),
  - a WEP kulcsokat ritkán cserélték ezért majdnem biztosra lehetett venni, hogy minden 4096 csomag esetében lesz két csomag, amelyiknek ugyanaz a WEP kulcsa.
- ha két olyan kezdeti kulcsértéket használunk, amelyek "közel állnak egymáshoz" további feltörési stratégia válik lehetségessé: related keys attack

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:

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

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

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

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:

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

- és meghatározható:

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

Az A5/1

- a GSM technológiában használják, ismert az A5/2 változat is, amely négy léptető regiszterrel dolgozik, de biztonsága gyengébb
- hardware szinten szokták implementálni,
- három léptető regiszterrel dolgozik, X, Y, Z-vel amelyek összesen 64 bitet kezelnek
  - az X 19 bites: $(x_0, x_1, \ldots, x_{18})$,
  - az Y 22 bites: $(y_0, y_1, \ldots, y_{21})$,
  - a Z 23 bites: $(z_0, z_1, \ldots, z_{22})$,
- a kulcs 64 bites, amellyel feltöltik a három regisztert
- a következő belső "majoráló/növelő" függvényt számoljuk ki minden óraimpulzusban:

$$m = maj(x_8, y_{10}, z_{10}) = \begin{cases} 0, & \text{ha a három bit között több a 0-ás} \\ 1, & \text{ha a három bit között több az 1-es} \end{cases}$$

- ha $x_8 = m$, akkor X-et léptetjük
- ha $y_{10} = m$, akkor az Y-t léptetjük
- ha $z_{10} = m$, akkor az Z-t léptetjük

Az A5/1

A kulcsfolyam előállítása a következőképpen történik:

- X léptetése:

$$\begin{aligned} t &= x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18} \\ x_i &= x_{i-1}, i = 18, 17, \ldots 1 - re \\ x_0 &= t \end{aligned}$$

- Y léptetése:

$$\begin{aligned} t &= y_{20} \oplus y_{21} \\ y_i &= y_{i-1}, i = 21, 20, \ldots, 1 - re \\ y_0 &= t \end{aligned}$$

- Z léptetése:

$$\begin{aligned} t &= z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22} \\ z_i &= z_{i-1}, i = 22, 21, \ldots, 1 - re \\ z_0 &= t \end{aligned}$$

- a kulcsfolyam aktuális s bitje: $s = x_{18} \oplus y_{21} \oplus z_{22}$

A Salsa20

- a 2008-ban lezáruló eSTREAM projekthez beküldött, egyike a négy kiválasztott folyamtitkosítónak,
- az eSTREAM nem standardokat kibocsájtó intézmény, hanem bátorítani akarja a kriptográfusokat titkosítók tervezésére, titkosítók kriptoanalizisére,
- tervezője Daniel Bernstein, munkái teljesen nyilvánosak, a következő címen érhetők el:

  http://cr.yp.to/djb.html,

- úgy hardver, mint szoftver implementációja ismert,
- a számítások párhuzamosíthatók, és a több magos processzorok esetében a titkosítás hatékonysága nagymértékben javítható,
- feltörési módszerek: az eddigi leghatásosabb módszer az összes kulcs kipróbálásának módszere,
- hatékonysága: 643 Mb/sec, $\sim$ 6-szor gyorsabb mint az RC4,
- a ChaCha20 a Salsa20 egy módosított változata,
- tulajdonképpen egy pszeudorandom függvény CTR módban alkalmazva,

A ChaCha20 folyamtitkosító

- a Salsa20-nak egy módosított változata, tervezője szintén Bernstein,
- az algoritmusnak elérhető úgy hardveres, mint softveres implementációja,
- széles körben elterjedt, használja az Open SSH, az SSL/TLS, és a Noise,
- a Google 2013-ban szabványosította, Android eszközökön is lehet használni,
- a nyílt szöveg hosszával megegyező kulcsfolyamot állít elő,
- a Chacha20-Poly1305 a társított adatokkal való hitelesített titkosítást (authenticated encryption with associated data (AEAD)) tesz lehetővé