Kriptográfia és Információbiztonság
3. előadás

MÁRTON Gyöngyvér

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

2023

Miről volt szó az elmúlt előadáson?

- Klasszikus kriptográfiai rendszerek, mátrixos rejtjelezés: Hill módszere
- kriptográfiai rendszerek: matematikai modellek
- az OTP titkosítási rendszer
- tökéletes biztonság
- biztonság-értelmezések, osztályozások

Miről lesz szó?

- titkos-kulcsú rendszerek: matematikai modell, osztályozások
- folyam-titkosító rendszerek: OTP, RC4, linear feedback shift register, A5/1, Salsa20
- blokk-titkosító rendszerek
- blokk-titkosító módok: ECB, CBC, CTR

Kriptográfiai rendszerek matematikai modellek

A titkos kulcsú rendszerek matematikai modellje

Titkos kulcsú titkosítók, egyéb megnevezések: 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 bit-hossza, 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 kulcscsere mechanizmusok matematikai modellje

Kulcscsere mechanizmusok (key exchange mechanism): ez egy interaktív protokoll, amikor a két kommunikáló fél egyidőben kell online legyen. Feltételezzük, hogy a két kommunikáló fél **A** és **B**. Három algoritmust szükséges értelmezni, ahol K a kulcsok, és $M = f(K)$ az üzenetek halmaza.

- Gen, a publikus információt/kulcsot generáló algoritmus, polinom idejű, véletlenszerű:
$$pk \xleftarrow{R} Gen(1^k),$$
$pk \in K$, ahol k a rendszer biztonsági paramétere, a generált kulcs bithossza
- a $PGen_{\mathbf{X}}(pk, \cdot)$, $\mathbf{X} \in \{\mathbf{A}, \mathbf{B}\}$ algoritmus polinom idejű, véletlenszerű:
  - ezt mindkét fél végrehajtja, a kapott A, B értékeket megosztják egymással, az a, b értékek azonban szigorúan titkosak maradnak:
$$A \xleftarrow{R} PGen_{\mathbf{A}}(pk, a), \quad a \xleftarrow{R} M$$
$$B \xleftarrow{R} PGen_{\mathbf{B}}(pk, b), \quad b \xleftarrow{R} M$$
- a $KGen_{\mathbf{X}}(x, \cdot)$, $\mathbf{X} \in \{\mathbf{A}, \mathbf{B}\}$, $x \in \{a, b\}$ algoritmus polinom idejű, determinisztikus, és amelyet mindkét fél végre kell hajtson:
$$key \leftarrow KGen_{\mathbf{A}}(a, B), \quad key \leftarrow KGen_{\mathbf{B}}(b, A)$$

A helyesség fennáll, ha minden $a, b \in M$-re:

$$KGen_{\mathbf{A}}(a, B) = KGen_{\mathbf{A}}(a, PGen_{\mathbf{B}}(pk, b)) = KGen_{\mathbf{B}}(b, PGen_{\mathbf{A}}(pk, a)) = KGen_{\mathbf{B}}(b, A).$$

A publikus kulcsú titkosítók matematikai modellje

Publikus kulcsú titkosítók, egyéb megnevezések: nyilvános kulcsú titkosítók, aszimmetrikus titkosítók (public-key encryption, asymmetric cryptography). Három algoritmust szükséges értelmezni, ahol K a kulcsok, és $M = f(K)$ az üzenetek halmaza.

- 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: $k \in \mathbb{Z}_{\geq 0}$
- $Enc(pk, \cdot)$ a rejtjelező algoritmus, polinom idejű, véletlenszerű:
$$c \xleftarrow{R} Enc(pk, m),$$
- a $Dec(sk, \cdot)$ a visszafejtő algoritmus, polinom idejű, determinisztikus:
$$m \leftarrow Dec(sk, c),$$

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

$$Dec(sk, Enc(pk, m)) = m$$

A digitális aláírás matematikai modellje

Digitális aláírások, egyéb megnevezések: publikus kulcsú tanúsítványok (digital signature, public key certificates). Három algoritmust szükséges értelmezni, ahol K a kulcsok, és $M = f(K)$ az üzenetek halmaza.

- Gen, a kulcs-generáló algoritmus, polinom idejű, lehet véletlenszerű is:
$$(pk, sk) \xleftarrow{R} Gen(1^k),$$
ahol $(pk, sk) \in K$ és k a rendszer biztonsági paramétere, legtöbb esetben a generált kulcs bit-hossza, és fennáll: $k \in \mathbb{Z}_{\geq 0}$
- $Aut(sk, \cdot)$ a hitelesítő algoritmus, polinom idejű, véletlenszerű:
$$(m, c) \xleftarrow{R} Aut(sk, m),$$
- a $Ver(pk, \cdot)$ az ellenőrző algoritmus, polinom idejű, determinisztikus, amelynek `True` vagy `False` a kimenete aszerint, hogy $m = \hat{m}_1$-el vagy sem:
$$\hat{m} \leftarrow Ver(pk, c).$$

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

$$Ver(pk, Aut(sk, m)) = m.$$

A titkos-kulcsú rendszerek osztályozása

- nagy adathalmaz titkosítására alkalmasak
- nincs megoldva a felek közötti kulcscsere, ezt a publikus kulcsú kriptográfia végzi,
- nincs megoldva a felek hitelesítése, ezt is a publikus kulcsú kriptográfia végzi,
- két nagy csoportra oszthatóak:
  - folyam-titkosító rendszerek,
  - blokk-titkosító rendszerek, adott blokk titkosítási móddal kombinálva átalakíthatóak folyam titkosító rendszereké

Folyam-titkosító 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
- 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

Folyam-titkosító 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
- biztonságuk vitatott, pl. az RC4, A5/1 kezdetben titkosak voltak, miután publikussá váltak számos gyengeséget fedeztek fel bennük

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
  - Blutooth 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 és rendelkezik a következő titkosított bit-szekvenciával

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

és a bit-szekvencia titkosított értékével:

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

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

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

Ez alapján fel tudja írni a következő lineáris egyenletet, és azt megoldva, megtudja határozni az alkalmazott lineáris összefüggést is, 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 folyam-titkosítónak,
- az eSTREAM nem standardokat kibocsájtó intézmény, hanem bátorítani akarja a kriptográfusokat titkosítók tervezésére, titkosítók kriptoanalizisére,
- tervezője D.J.Bernstein; teljesen publikus:

  http://cr.yp.to/djb.html,
- úgy hardver, mint softver implementációja ismert,
- a számítások párhuzamosíthatóak, 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 Salsa20

- a kulcs mérete: 128 vagy 256-bit hossz,
- ha a kulcs mérete 128 bit, akkor a következő a specifikáció:
- a k kulcs mellett a bemenetnek meg kell adni egy 64 bites r nonce (number use at once) értéket, ahol a $||$ összefűzést jelent:

$$\begin{aligned} Salsa20 &: \{0,1\}^{128} \times \{0,1\}^{64} \to \{0,1\}^n \\ Salsa20(k,r) &= H(k||r||0))||H(k||r||1)|| \ldots, \end{aligned}$$

- A H függvényt a $k||r||i$ bemeneten hajtja végre, ahol az i számláló értékét 64 biten tárolja, és $k = k_0||k_1$
- A H függvény a 32 bájtból (256 bit) $\to$ 64 bájtot állít elő:

$$\begin{aligned} &k||r||i \to t_0||k_0||t_1||r||i||t_2||k_1||t_3, \\ &t_0 = 61707865, t_1 = 3320646E, \\ &t_2 = 79622D32, t_3 = 6B206574 \end{aligned}$$

- ezen 10-szer alkalmaz egy invertálható $h : \{0,1\}^{64} \to \{0,1\}^{64}$ függvényt.
- a H függvény kimenetét 4 bájtonként $\pmod{2^{32}}$ szerinti összeadást végezve adja hozzá a nyílt szöveghez.

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 kulcs-folyammal végezték a titkosítást $\Rightarrow$ a kulcs többszöri felhasználása. Megoldás: kulcspár használata, 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.