METODE CRIPTOGRAFICE DE PROTECȚIE A INFORMAȚIEI

Tema 3: Cifruri fluide moderne

SISTEMUL DE CRIPTARE A5/1

Sistemul de criptare fluidă A5/1 a fost utilizat în rețelele de telefonie mobilă GSM (Groupe Spécial Mobile) pentru a asigura confidențialitatea datelor transmise între telefonul mobil și cea mai apropiată stație. Cifrul A5/1 a fost elaborat în 1987 pentru utilizare în Europa și SUA, iar în 1989 a fost pusă în aplicare o versiune mai slabă a cifrului, numită A5/2, destinată exportului în alte zone geografice. După ce în anul 2000 a fost demonstrată vulnerabilitatea cifrului, acesta a fost făcut public.

SISTEMUL DE CRIPTARE A5/1

La nivel de funcționare sistemul de criptare A5/1 reprezintă un automat finit bazat pe 3 registre liniare sincrone cu feedback (LFSR). Cele 3 registre R^1, R^2, R^3 conțin 19, 22, și, respectiv, 23 de flip-flopuri de date (unități de memorie de un bit) ce generează șiruri pseudoaleatoare, în baza cărora sunt obținuți biții cheii fluide. Biții registrului sunt indexați de la cel mai puțin semnificativ spre cel mai semnificativ, primul bit având indicele 0.

Cifrul A5/1 constă din două etape: la etapa inițială are loc inițializarea registrelor R^1, R^2, R^3 (acestea formează starea internă), iar la etapa a doua este pus în funcțiune automatul finit A5/1, care emite biții cheii fluide. În continuare, vom descrie mai întâi etapa a doua a cifrului, după care etapa întâi.

SISTEMUL DE CRIPTARE A5/1

2.2.4.3.1 Funcționarea automatului finit A5/1

La un tact, fiecare registru poate rămâne pe loc sau își poate deplasa conținutul spre stânga (deci avem o deplasare neuniformă), aducând pe ultima poziție o combinație liniară de alți biți. Combinațiile liniare corespunzătoare celor trei registre sunt definite astfel (a se vedea *Figura 2.2.11*):

$$R_0^1 := R_{13}^1 \oplus R_{16}^1 \oplus R_{17}^1 \oplus R_{18}^1,\ R_0^2 := R_{20}^2 \oplus R_{21}^2,\ R_0^3 := R_7^3 \oplus R_{20}^3 \oplus R_{21}^3 \oplus R_{22}^3 . \qquad (2.2.1)$$

SISTEMUL DE CRIPTARE A5/1

Pentru a decide care registre sunt deplasate la un tact, se folosesc 3 biți de tact u_1, u_2, u_3 (de pe pozițiile 8, 10 și 10 ale registrelor R^1, R^2 și R^3 corespunzător). La fiecare tact, folosind funcția $F = or\left(or\left(and\left(u_1, u_2\right), and\left(u_1, u_3\right)\right), and\left(u_2, u_3\right)\right)$, unde *and* este operatorul *și logic*, iar *or* - *sau logic*, se determină valoarea majoritară dintre acești biți. Registrul se deplasează (și emite un bit ce participă la generarea bitului din cheia fluidă) dacă bitul său de tact coincide cu valoarea majoritară F. Ținând cont de rezultatele din Tabelul 2.2.1, rezultă că la un tact, în cel puțin două registre are loc o deplasare.

SISTEMUL DE CRIPTARE A5/1

Tabelul 2.2.1

u_1	0	0	0	1	0	1	1	1
u_2	0	0	1	0	1	0	1	1
u_3	0	1	0	0	1	1	0	1
F	0	0	0	0	1	1	1	1

La fiecare tact i, valoarea emisă de automat reprezintă un bit z_i al cheii fluide

$$z_i = R_{18}^1 \oplus R_{21}^2 \oplus R_{22}^3 .$$

care, la criptare, se combină prin XOR cu bitul i al textului clar, iar la decriptare – cu bitul i al textului criptat.

Deoarece sistemul GSM transmite mesajele în blocuri de câte 114 biți, A5/1 generează blocuri pe 114 biți ai cheii fluide. Astfel, este necesară o reinițializare a automatului finit pentru fiecare bloc transmis.

SISTEMUL DE CRIPTARE A5/1

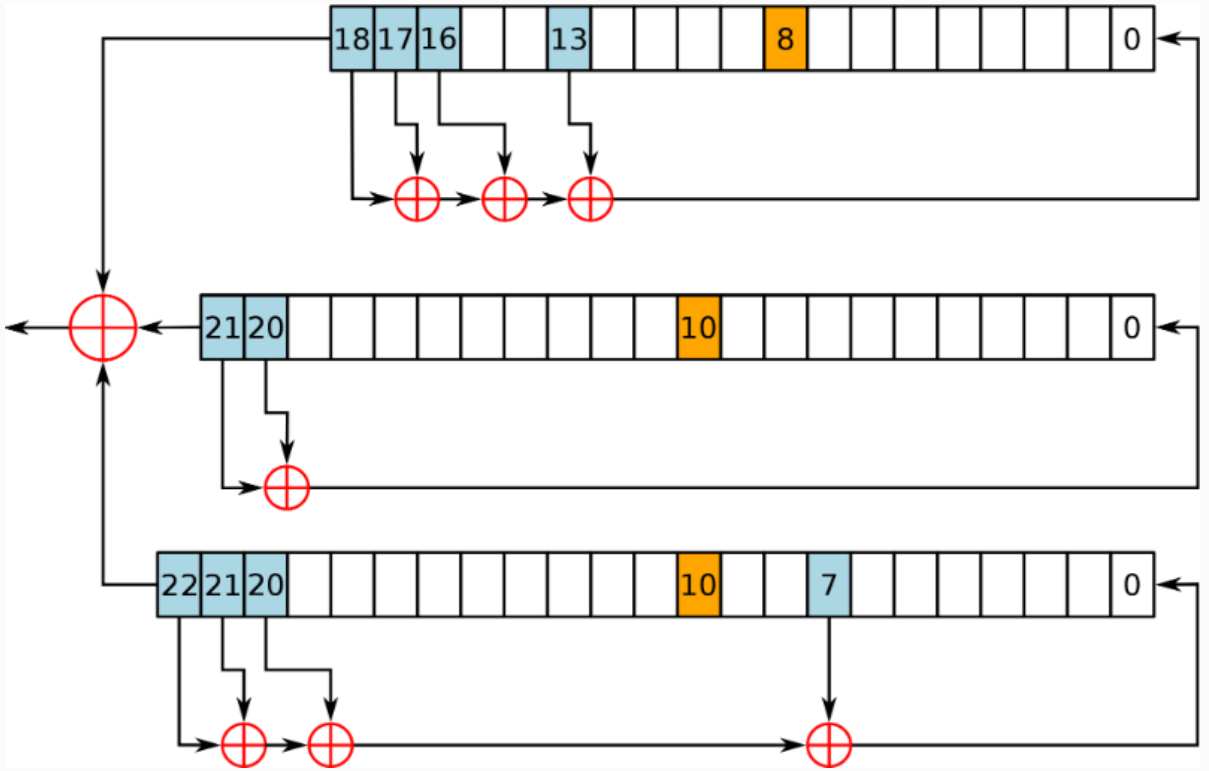

Figura 2.2.11. Automatul finit descris de cifrul A5/1

SISTEMUL DE CRIPTARE A5/1

Algoritmul de funcționare a automatului finit A5/1

Date de intrare:

R^1, R^2, R^3 – tablouri cu 19, 22 și, respectiv, 23 de elemente, obținute după inițializarea stării interne

u_1, u_2, u_3 – biții de tact din registrele R^1, R^2 și R^3, corespunzător. Locațiile acestora sunt $R^1_8, R^2_{10}, R^3_{10}$ respectiv

Date de ieșire:

z – tabloul ce conține blocul din 114 biți ai cheii fluide

SISTEMUL DE CRIPTARE A5/1

Pentru $i=\overline{1,114}$ execută

{

1. Automatul finit emite bitul z_i al cheii fluide $z_i := R_{18}^1 \oplus R_{21}^2 \oplus R_{22}^3$
2. Dintre biții de tact u_1, u_2, u_3 se determină bitul de valoare majoritară

$$F = or\left(or\left(and\left(u_1,u_2\right),and\left(u_1,u_3\right)\right),and\left(u_2,u_3\right)\right)$$

3. Pentru $m=\overline{1,3}$ execută

{

Dacă $F = u_m$ atunci

SISTEMUL DE CRIPTARE A5/1

{

Este determinat bitul $R_0^m := \begin{cases} R_{13}^m \oplus R_{16}^m \oplus R_{17}^m \oplus R_{18}^m & dacă \ \ m=1 \\ R_{20}^m \oplus R_{21}^m & dacă \ \ m=2 \\ R_7^m \oplus R_{20}^m \oplus R_{21}^m \oplus R_{22}^m & dacă \ \ m=3 \end{cases}$

Conținutul registrului R^m este deplasat cu o poziție la stânga, iar în prima poziție (celula din dreapta a tabloului R^m, corespunzătoare celui mai puțin semnificativ bit) se pune bitul determinat R_0^m

}

}

}

SISTEMUL DE CRIPTARE A5/1

Exemplul 2.2.2. Fie cele 3 registre (biții de tact au fost marcaţi cu bold):

$$R^1 = 0110\ 1011\ 10\mathbf{1}0\ 1001\ 011,$$

$$R^2 = 1010\ 1100\ 00\mathbf{1}0\ 1101\ 1001\ 11,$$

$$R^3 = 1100\ 0101\ 1011\ \mathbf{0}010\ 1010\ 010.$$

Să urmărim rezultatele emise de automatul finit pe parcursul a 5 tacte:

SISTEMUL DE CRIPTARE A5/1

Tactul 1: Bitul din cheia fluidă este $k_1 := 0 \oplus 1 \oplus 1 = 0$. Deoarece biții de tact sunt $u = 1, v = 0, w = 0$, rezultă $F = 0$ și astfel se efectuează o deplasare în R^2 și R^3. Registrele trec în starea următoare:

$R^1 = 0110\ 1011\ 10\mathbf{1}0\ 1001\ 011,$

$R^2 = 0101\ 1000\ 010\mathbf{1}\ 1011\ 0011\ 11,$

$R^3 = 1000\ 1011\ 0110\ \mathbf{0}101\ 0100\ 100.$

SISTEMUL DE CRIPTARE A5/1

Tactul 2: Bitul din cheia fluidă este $k_2 := 0 \oplus 0 \oplus 1 = 1$. Deoarece biții de tact sunt $u = 1,\ v = 1,\ w = 0$, rezultă $F = 1$ și astfel se efectuează o deplasare în R^1 și R^2. Registrele trec în starea următoare:

$R^1 = 1101\ 0111\ 01\mathbf{0}1\ 0010\ 110$,

$R^2 = 1011\ 0000\ 101\mathbf{1}\ 0110\ 0111\ 11$,

$R^3 = 1000\ 1011\ 0110\ \mathbf{0}101\ 0100\ 100$.

SISTEMUL DE CRIPTARE A5/1

Tactul 3: Bitul din cheia fluidă este $k_3 := 1 \oplus 1 \oplus 1 = 1$. Deoarece biții de tact sunt $u = 0$, $v = 1$, $w = 0$, rezultă $F = 0$ și astfel se efectuează o deplasare în R^1 și R^3. Registrele trec în starea următoare:

$R^1 = 1010\ 1110\ 1010\ 0101\ 101,$

$R^2 = 1011\ 0000\ 1011\ 0110\ 0111\ 11,$

$R^3 = 0001\ 0110\ 1100\ 1010\ 1001\ 000.$

SISTEMUL DE CRIPTARE A5/1

Tactul 4: Bitul din cheia fluidă este $k_4 := 1 \oplus 1 \oplus 0 = 0$. Deoarece biții de tact sunt $u = 1$, $v = 1$, $w = 1$, rezultă $F = 1$ și astfel se efectuează o deplasare în R^1, R^2 și R^3. Registrele trec în starea următoare:

$R^1 = 0101\ 1101\ 0100\ 1011\ 011$,

$R^2 = 0110\ 0001\ 0110\ 1100\ 1111\ 11$,

$R^3 = 0010\ 1101\ 1001\ 0101\ 0010\ 000$.

Tactul 5: Bitul din cheia fluidă este $k_5 := 0 \oplus 0 \oplus 0 = 0$.

Deci, primii 5 biți ai cheii fluide sunt: $z = 01100\ldots$. □

SISTEMUL DE CRIPTARE A5/1

2.2.4.3.2 Inițializarea stării interne A5/1

Cifrul A5/1 necesită o inițializare a stării interne, bazată pe o cheie secretă K pe 64 biți și anumiți parametri GSM stocați în constanta *Count* (vector de inițializare) pe 22 biți. Pe parcursul a 64 de runde este mixată cheia secretă K în modul următor: la runda $i \in \{0,1,...,63\}$ bitul cu numărul i al cheii, K_i, este combinat prin XOR cu cel mai puțin semnificativ bit al fiecărui registru. În continuare, în fiecare registru este aplicată o deplasare la stânga. Analog cum s-a procedat cu biții cheii secrete K, se parcurg 22 de runde, în care sunt mixați biții constantei *Count* cu cel mai puțin semnificativ bit al fiecărui registru corespunzător. În fine, pe parcursul a 100 de runde se rulează câte un tact al automatului finit A5/1, ignorând biții de ieșire. Astfel sunt neglijați primii 100 biți de ieșire.

SISTEMUL DE CRIPTARE A5/1

Algoritmul de inițializare a stării interne A5/1

Date de intrare:

R^1, R^2, R^3 - tablouri cu 19, 22 și, respectiv, 23 de elemente

$K = (K_0, K_1, ..., K_{63})$ - cheia secretă pe 64 biți

$Count = (Count_0, Count_1, ..., Count_{21})$ - constantă pe 22 de biți (frame number)

Date de ieșire:

R^1, R^2, R^3 - registrele cu valorile inițiale

SISTEMUL DE CRIPTARE A5/1

4. Se inițializează cu zero registrele R^1, R^2, R^3

5. Pentru $i := \overline{0,63}$ execută

   5.1. $R_0^1 := R_0^1 \oplus K_i$

   5.2. $R_0^2 := R_0^2 \oplus K_i$

   5.3. $R_0^3 := R_0^3 \oplus K_i$

   5.4. Deplasare la stânga în fiecare registru $R^m, m = 1,2,3$ (adică pentru $j > 0$, $R_j^m := R_{j-m}^m$, iar în R_0^m se scrie bitul obținut conform relației (2.2.1)

SISTEMUL DE CRIPTARE A5/1

6. Pentru $i := \overline{0,21}$ execută

6.1. $R_0^1 := R_0^1 \oplus Count_i$

6.2. $R_0^2 := R_0^2 \oplus Count_i$

6.3. $R_0^3 := R_0^3 \oplus Count_i$

6.4. Deplasare la stânga în fiecare registru $R^m, m = 1,2,3$

7. Pentru $i := \overline{0,99}$ execută

{Rulează un tact automatul A5/1, ignorând ieșirea acestuia}

FINALISTELE CONCURSULUI eSTREAM

2.2.5 Finalistele concursului e-STREAM

eSTREAM [6] a fost un proiect gestionat de rețeaua EU ECRYPT, care a avut ca scop elaborarea unor cifruri fluide noi, care să satisfacă standardele actuale de securitate și să fie utilizate la scară largă. Acest proiect a fost realizat ca urmare a eșecului celor 6 cifruri fluide propuse în cadrul proiectului NESSIE (New European Schemes for Signatures, Integrity and Encryption), desfășurat în perioada 2000 – 2003 [10]. Proiectul eSTREAM a fost implementat în perioada noiembrie 2004 până în aprilie 2008. În cadrul a trei etape ale proiectului au fost examinați algoritmi propuși la două categorii:

- cifruri fluide pentru implementare software cu posibilități mari de transfer;
- cifruri fluide pentru implementare hardware cu resurse limitate (memorie limitată, consum de putere etc.).

FINALISTELE CONCURSULUI eSTREAM

După cea de-a treia etapă a proiectului a fost stabilit portofoliul eSTREAM, care și actualmente este format din următoarele cifruri:

Profilul 1 (implementare software)	Profilul 2 (implementare hardware)
HC-128 (și versiunea sa HC-256)	Grain
Rabbit	MICKEY
Salsa20/12	Trivium
SOSEMANUK	

Toate cifrurile menționate sunt libere pentru utilizare non-comercială. Rabbit este unicul cifru patentat la etapa inițială a concursului, dar a fost făcut public în octombrie 2008. Inițial, portofoliul publicat la sfârșitul etapei a treia consta din cifrurile menționate anterior, plus cifrul F-FCSR de la profilul 2. Acesta din urmă a fost eliminat din portofoliu în septembrie 2008, în urma identificării unor posibilități de atac.

Candidații pentru eSTREAM au o structură ceva mai complicată ca a algoritmului RC4 și operează la nivel de cuvinte pe 32 biți, spre deosebire de RC4, care lucrează cu octeți.

SISTEMUL DE CRIPTARE SALSA20

Acest cifru fluid (învingătorul concursului eSTREAM la profilul implementare software) a fost elaborat de către D. Bernstein în anul 2005 [11] și are la bază o funcție pseudoaleatoare ce folosește operațiile ARX (add-rotate-xor) aplicate asupra cuvintelor pe 32 de biți:

- Operația sau exclusiv XOR (suma pe biți modulo 2), notată prin $\oplus$;
- Suma $\bmod 2^{32}$, notată prin $\boxplus$;
- Operația de rotație circulară la stânga cu un număr de biți definit, notată prin *rol* .

SISTEMUL DE CRIPTARE SALSA20

Algoritmul Salsa20 preia ca și parametri de intrare o cheie pe 32 sau pe 16 octeți și o valoare nonce pe 8 octeți, concatenată cu o altă valoare dinamică pe 8 octeți (numită *increment*), și întoarce un bloc de 64 octeți al cheii fluide. Octeții blocului din cheia fluidă sunt combinați printr-un XOR cu 64 de octeți de text clar (a se vedea *Figura 2.2.12*). Dacă este necesar, se generează următorul bloc al cheii fluide, care este combinat cu blocul corespunzător de text clar ș.a.m.d. La decriptare blocul din cheia fluidă este combinat prin XOR cu blocul de text criptat.

SISTEMUL DE CRIPTARE SALSA20

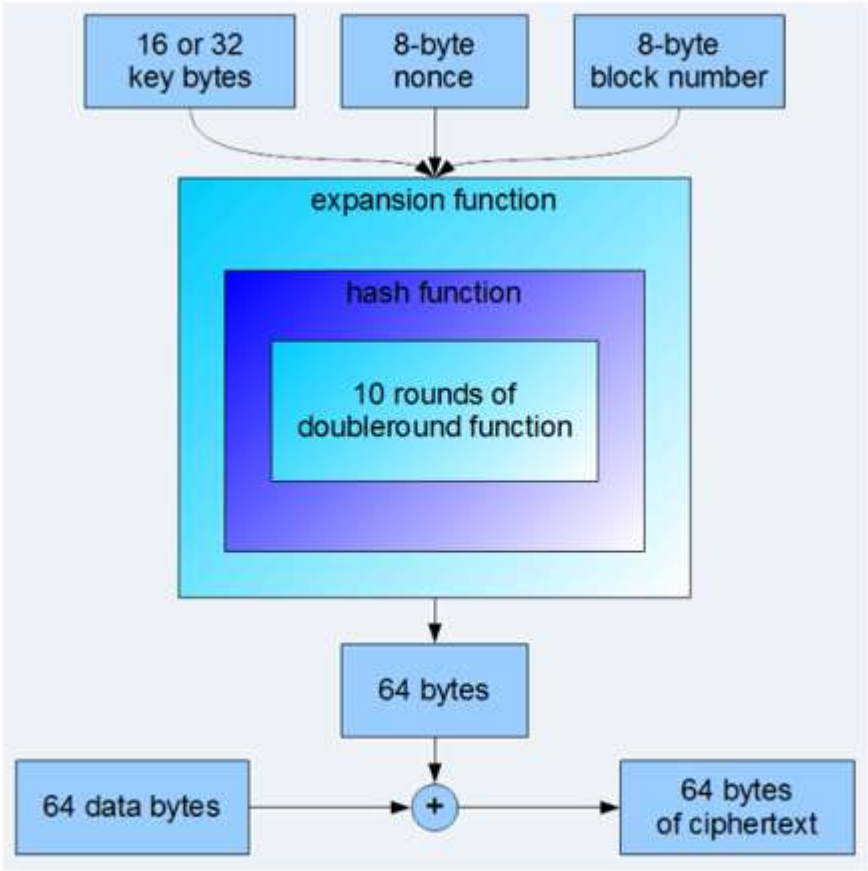

Figura 2.2.12. Schema generală a cifrului fluid Salsa20

2.2.5.1.1 Taxonomia utilizată

Prin octet vom înțelege un element al mulțimii $\{0,1,...,255\}$, iar prin cuvânt pe 32 biți – o valoare numerică din mulțimea $\{0,1,...,2^{32}-1\}$.

Uneori vom folosi reprezentarea hexazecimală pentru cuvinte, ilustrând faptul că este utilizat așa un format de reprezentare prin notația $0x$.

SISTEMUL DE CRIPTARE SALSA20

Prin sumă a două cuvinte pe 32 biți, u și v vom înțelege valoarea $\mathrm{mod}\left(u+v,2^{32}\right)\in\left[0,2^{32}-1\right]$ (la fel un cuvânt pe 32 biți) și, în acest sens, vom folosi notația $u\boxplus v$.

Operația sau exclusiv a două cuvinte $u=\sum_i 2^i u_i$ și $v=\sum_i 2^i v_i$ este $u\oplus v=\sum_i 2^i\left(u_i+v_i-2u_iv_i\right)$.

Pentru fiecare $c\in\{0,1,2,3,\ldots\}$ vom înțelege prin rotație ciclică la stânga cu c biți a cuvântului u, notată prin $rol\left(u,c\right)$, unicul cuvânt nenul congruent cu $2^c u$ modulo $2^{32}-1$ (cu excepția că $rol\left(0,c\right)=0$). Altfel spus, dacă $u=\sum_i 2^i u_i$, atunci $rol\left(u,c\right)=\sum_i 2^{\mathrm{mod}(i+c,32)}u_i$.

2.2.5.1.2 Transformările criptografice utilizate în cadrul algoritmului

2.2.5.1.2.1 Transformarea quarterround

Blocul de bază al sistemului de criptare îl reprezintă transformarea $quarterround(y)$ aplicată la șiruri y din 4 cuvinte pe 32 biți, și care întoarce, la fel, un șir din 4 cuvinte (a se vedea *Figura 2.2.12*). Sistemul de criptare folosește unele transformări complexe, bazate pe această transformare.

SISTEMUL DE CRIPTARE SALSA20

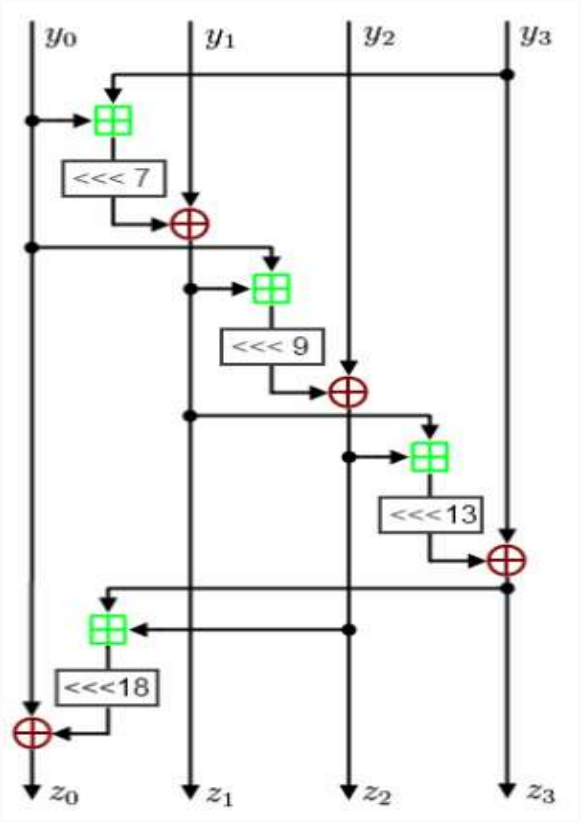

Figura 2.2.13. Funcția quarterround

SISTEMUL DE CRIPTARE SALSA20

Definiția 2.2.7. Pentru șirul din 4 cuvinte pe 32 biți $y=(y_0,y_1,y_2,y_3)$ definim transformarea $quarterround(y)=z$, unde $z:=(z_0,z_1,z_2,z_3)$ satisface condițiile următoare (a se vedea *Figura 2.2.13*):

$$z_1=y_1\oplus rol(y_0\boxplus y_3,7),\ z_2=y_2\oplus rol(z_1\boxplus y_0,9),\ z_3=y_3\oplus rol(z_2\boxplus z_1,13),\ z_0=y_0\oplus rol(z_3\boxplus z_2,18).$$

Transformarea $quarterround(y)$ este inversabilă, adică, știind $z=quarterround(y)$, se poate determina $y=quarterround^{-1}(z)$.

2.2.5.1.2.2 Transformarea rowround

Definiția 2.2.8. Pentru șirul din 16 cuvinte pe 32 biți $y=(y_0,y_1,\ldots,y_{15})$ definim transformarea $rowround(y)=z:=(z_0,z_1,\ldots,z_{15})$, unde z este un șir din 16 cuvinte ce satisfac următoarele condiții:

$$(z_0,z_1,z_2,z_3)=quarterround(y_0,y_1,y_2,y_3),$$

$$(z_5,z_6,z_7,z_4)=quarterround(y_5,y_6,y_7,y_4),$$

$$(z_{10},z_{11},z_8,z_9)=quarterround(y_{10},y_{11},y_8,y_9),$$

$$(z_{15},z_{12},z_{13},z_{14})=quarterround(y_{15},y_{12},y_{13},y_{14}).$$

SISTEMUL DE CRIPTARE SALSA20

Șirul de intrare $y = (y_0, y_1, ..., y_{15})$ poate fi privit ca un tablou bidimensional

$$\begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 & y_7 \\ y_8 & y_9 & y_{10} & y_{11} \\ y_{12} & y_{13} & y_{14} & y_{15} \end{pmatrix}.$$

Transformarea *rowround* acționează în paralel asupra liniilor tabloului, realizând o permutare a fiecărei linii prin intermediul funcției *quarterround*. Șirurile de intrare pentru *quarterround* sunt formate din cuvintele liniei $i \in \{0,1,2,3\}$, iar cuvintele în cadrul liniei se consideră succesiv, începând cu elementul i al liniei.

2.2.5.1.2.3 Transformarea columnround

Definiția 2.2.9. Pentru șirul din 16 cuvinte pe 32 biți $x=(x_0,x_1,...,x_{15})$, definim transformarea $columnround(x)=y:=(y_0,y_1,...,y_{15})$, unde y este un șir din 16 cuvinte ce satisfac următoarele condiții:

$$(y_0,y_4,y_8,y_{12})=quarterround(x_0,x_4,x_8,x_{12}),$$

$$(y_5,y_9,y_{13},y_1)=quarterround(x_5,x_9,x_{13},x_1),$$

$$(y_{10},y_{14},y_2,y_6)=quarterround(x_{10},x_{14},x_2,x_6),$$

$$(y_{15},y_3,y_7,y_{11})=quarterround(x_{15},x_3,x_7,x_{11}).$$

SISTEMUL DE CRIPTARE SALSA20

O formulă echivalentă ce definește rezultatul transformării *columnround*, se scrie astfel:

$$(y_0, y_4, y_8, y_{12}, y_1, y_5, y_9, y_{13}, y_2, y_6, y_{10}, y_{14}, y_3, y_7, y_{11}, y_{15}) =$$

$$= rowround(x_0, x_4, x_8, x_{12}, x_1, x_5, x_9, x_{13}, x_2, x_6, x_{10}, x_{14}, x_3, x_7, x_{11}, x_{15}).$$

SISTEMUL DE CRIPTARE SALSA20

Șirul de intrare $x = (x_0, x_1, ..., x_{15})$ poate fi scris ca un tablou bidimensional

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix}.$$

Spre deosebire de *rowround*, transformarea *columnround* acționează în paralel asupra coloanelor tabloului, realizând o permutare a fiecărei coloane prin intermediul funcției *quarterround*. Șirurile de intrare pentru *rowround* sunt formate din cuvintele coloanei $j \in \{0,1,2,3\}$, iar cuvintele în cadrul coloanei se consideră succesiv, începând cu elementul j al coloanei.

2.2.5.1.2.4 Transformarea doubleround

Definiția 2.2.10. Pentru șirul x din 16 cuvinte pe 32 biți transformarea *doubleround* aplică succesiv funcțiile *columnround* și *rowround*, rezultând următorul șir din 16 cuvinte:

$$doubleround(x) = rowround(columnround(x)).$$

Transformarea *doubleround* acționează în paralel asupra coloanelor tabloului de intrare, după care, acționează în paralel asupra liniilor tabloului rezultat. Astfel, fiecare cuvânt este prelucrat de două ori.

2.2.5.1.2.5 Reprezentarea littleendian

Algoritmul Salsa20 folosește un format de reprezentare a cuvântului pe 32 biți (4 octeți), începând cu octetul cel mai puțin semnificativ. Dacă b este un șir din 4 octeți, atunci cuvântul pe 32 biți $littleendian(b)$, format cu acești octeți, se definește astfel:

Definiția 2.2.11. Dacă $b = (b_0, b_1, b_2, b_3)$ este un șir din 4 octeți, atunci există o singură reprezentare de forma

$$littleendian(b) := b_0 \boxplus 2^8 b_1 \boxplus 2^{16} b_2 \boxplus 2^{24} b_3. \quad (2.1)$$

SISTEMUL DE CRIPTARE SALSA20

Funcția *littleendian* este inversabilă, adică, cunoscând cuvântul $littleendian(b)$, se poate determina șirul unic $b=(b_0,b_1,b_2,b_3)$ pentru care are loc reprezentarea (2.1), iar transformarea corespunzătoare o vom nota prin $littleendian^{-1}$.

2.2.5.1.2.6 Funcția hash Salsa20

Definiția 2.2.12. Pentru șirul x din 64 octeți (16 cuvinte pe 32 biți) definim funcția $Salsa20Hash(x)$ prin relația

$$Salsa20Hash(x) = x \boxplus doubleround^{10}(x), \tag{2.2}$$

rezultatul căreia, la fel, este un șir din 64 octeți. Fiecare 4 octeți succesivi (1 cuvânt) ai șirului de intrare x se reprezintă ca un cuvânt în format littleendian. Funcția *doubleround* este aplicată succesiv de 10 ori la șirul transformat x, rezultatul unei runde fiind utilizat la intrarea în următoarea rundă. În fine, se adună modulo 2^{32} cuvintele șirului x și ale șirului $doubleround^{10}(x)$.

Transformarea *Salsa20Hash* reprezintă suma pe biți a șirului inițial x și a rezultatului rulării a 20 de transformări succesive ale coloanelor și liniilor tabloului bidimensional (transformările *rowround* și *columnround*) format în baza șirului x.

SISTEMUL DE CRIPTARE SALSA20

În continuare, expunem detaliile de realizare a funcţiei Salsa20:

I. Pornind de la $x=\left(x(0),x(1),...,x(63)\right)$, se definesc cuvintele

$$x_0 = littleendian\left(x(0),x(1),x(2),x(3)\right),$$

$$x_1 = littleendian\left(x(4),x(5),x(6),x(7)\right),$$

$$x_2 = littleendian\left(x(8),x(9),x(10),x(11)\right),$$

.....

$$x_{15} = littleendian\left(x(60),x(61),x(62),x(63)\right),$$

unde $x(i)$ sunt octeţi ai lui x, iar x_j - cuvinte din 4 octeţi.

SISTEMUL DE CRIPTARE SALSA20

II. În cadrul a 10 runde în care se aplică transformarea doubleround, este determinat șirul din 16 cuvinte pe 32 biți $(z_0, z_1, ..., z_{15}) = doubleround^{10}(x_0, x_1, ..., x_{15})$.

III. Șirul din 16 cuvinte $Salsa20Hash(x)$ se obține prin concatenarea următoarelor cuvinte:

$$littleendian^{-1}(z_0 \boxplus x_0),$$

$$littleendian^{-1}(z_1 \boxplus x_1),$$

.....

$$littleendian^{-1}(z_{15} \boxplus x_{15}).$$

SISTEMUL DE CRIPTARE SALSA20

Cei 4 octeți ce formează cuvântul $b^{(i)}$ din reprezentarea $littleendian\left(b^{(i)}\right)=z_i \boxplus x_i,\ i\in\{0,1,...,15\}$ (scris in hexazecimal) astfel încât $littleendian^{-1}\left(littleendian\left(b^{(i)}\right)\right)=b^{(i)}$, pot fi determinați astfel:

$$b^{(i)}=\left(w\,\&\,0xff,\left(ror\left(w,8\right)\right)\&\,0xff,\left(ror\left(w,16\right)\right)\&\,0xff,\left(ror\left(w,24\right)\right)\&\,0xff\right),$$

unde $w:=littleendian\left(b^{(i)}\right)$, & - operația „și logic”, ror - operația de rotație ciclică la dreapta a biților cuvântului cu un număr fixat de biți, iar $0xff$ - reprezentarea pe 32 de biți a octetului $(11111111)_2$, adică $0xff=\left(\underbrace{0...0}_{24\ biti}\underbrace{1...1}_{8\ biti}\right)_2$.

2.2.5.1.2.7 Transformarea de expandare a cheii secrete în algoritmul Salsa20

Dacă cheia secretă k este un șir din 32 octeți sau din 16 octeți, iar n este un șir din 16 octeți, atunci prin expandare a cheii secrete k vom înțelege șirul din 64 octeți întors de funcția $Salsa20Expansion_k(n)$.

Definiția 2.2.13. La expandarea cheii secrete k pe 32 octeți sunt utilizate patru cuvinte pe 32 de biți, octeții cuvintelor fiind dați în reprezentare zecimală:

$$\sigma_0 = (101,120,112,97),\ \sigma_1 = (110,100,32,51),\ \sigma_2 = (50,45,98,121),\ \sigma_3 = (116,101,32,107).$$

SISTEMUL DE CRIPTARE SALSA20

Dacă $k=(k_0,k_1)$ - cheia secretă, iar k_0,k_1,n sunt șiruri din 16 octeți, atunci

$$Salsa20Expansion_{k_0,k_1}(n) := Salsa20Hash(\sigma_0,k_0,\sigma_1,n,\sigma_2,k_1,\sigma_3).$$

La expandarea cheii secrete k pe 16 octeți sunt utilizate 4 cuvinte, octeții cărora sunt scriși în reprezentare zecimală:

$$\tau_0=(101,120,112,97),\ \tau_1=(110,100,32,49),\ \tau_2=(54,45,98,121),\ \tau_3=(116,101,32,107).$$

Dacă k,n sunt șiruri din 16 octeți, atunci avem

$$Salsa20Expansion_k(n) := Salsa20Hash(\tau_0,k,\tau_1,n,\tau_2,k,\tau_3).$$

SISTEMUL DE CRIPTARE SALSA20

Cuvintele concatenate $\sigma_0,\sigma_1,\sigma_2,\sigma_3$ și $\tau_0,\tau_1,\tau_2,\tau_3$ sunt, de fapt, codificările ASCII ale expresiilor „*expand 32-byte k*" și, respectiv, „*expand 16-byte k*".

Exemplul 2.2.3. Fie $k_0=(1,2,\ldots,16)$, $k_1=(201,202,\ldots,216)$, $n=(101,102,\ldots,116)$. Atunci avem:

$$Salsa20Expansion_{k_0,k_1}(n) := Salsa20Hash(101,120,112,97,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,110,$$
$$100,32,51,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,50,45,98,121,$$
$$201,202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,116,101,32,107)=$$
$$=(69,37,68,39,41,15,107,193,255,139,122,6,170,233,217,98,89,144,182,106,21,51,200,65,239,$$
$$49,222,34,215,114,40,126,104,197,7,225,197,153,31,2,102,78,76,176,84,245,246,184,177,160,$$
$$133,130,6,72,149,119,192,195,132,236,234,103,246,74)$$ □

SISTEMUL DE CRIPTARE SALSA20

2.2.5.1.2.8 Transformarea de criptare Salsa20

Să considerăm șirurile:

k - cheia secretă, care reprezintă un șir din 32 sau 16 octeți;

v - un șir din 8 octeți, numit valoare nonce (se alege aleator și este utilizată doar la o criptare);

m - textul clar de lungime l octeți, unde $l \in \{0,1,\ldots,2^{70}\}$.

SISTEMUL DE CRIPTARE SALSA20

Salsa20 fiind un cifru fluid, criptarea are loc în mod standard, combinând prin XOR octeții textului clar m cu octeții cheii fluide. Fiecare 64 octeți ai cheii fluide sunt obținuți în baza funcției de expandare a cheii cu parametrii de intrare k și v, mai exact, $Salsa20Expansion_k(v,i)$, unde i este un șir din 8 octeți $(i_0,i_1,...,i_7)$ ce satisface condiția $i = i_0 \boxplus_{64} 2^8 i_1 \boxplus_{64} 2^{16} i_2 \boxplus_{64} ... \boxplus_{64} 2^{56} i_7$ (adunare modulo 2^{64}):

$$(i_0,i_1,...,i_7) = (i \& 0xff, (ror(i,8)) \& 0xff, (ror(i,16)) \& 0xff, (ror(i,24)) \& 0xff, ..., (ror(i,56)) \& 0xff).$$

Pentru fiecare valoare distinctă a lui i se obțin 64 octeți ai cheii fluide.

SISTEMUL DE CRIPTARE SALSA20

Definiția 2.2.14. Cheia fluidă $Salsa20Expansion_k(v)$ reprezintă un șir din 2^{70} octeți, definit astfel

$$Salsa20Expansion_k(v,0), Salsa20Expansion_k(v,1), ..., Salsa20Expansion_k(v,2^{64}-1).$$

Șirul $(i_0, i_1, ..., i_7)$ se determină astfel (a se vedea funcția `inverse_littleendian_64bit(NB)` din *Algoritmul de criptare fluidă Salsa20* prezentat în continuare):

$$(i_0, i_1, ..., i_7) = (i \,\&\, 0xff, (ror(i,8)) \,\&\, 0xff, (ror(i,16)) \,\&\, 0xff, (ror(i,24)) \,\&\, 0xff,$$
$$(ror(i,32)) \,\&\, 0xff, (ror(i,40)) \,\&\, 0xff, (ror(i,48)) \,\&\, 0xff, (ror(i,56)) \,\&\, 0xff).$$

SISTEMUL DE CRIPTARE SALSA20

Astfel, textul criptat c este determinat în baza relației $c = \{Salsa20Expansion_k(v)\}_l \oplus m$ (unde $\{...\}_l$ - primii l octeți din cheia fluidă) și are lungimea de l octeți. Decriptarea textului c este efectuată, folosind relația $m = \{Salsa20Expansion_k(v)\}_l \oplus c$.

Din cele menționate rezultă

$$\{Salsa20Expansion_k(v)\}_l \oplus (m_0, m_1, ..., m_{l-1}) = (c_0, c_1, ..., c_{l-1}),$$

unde $c_i = m_i \oplus \left(Salsa20Expansion_k\left(v, \underline{floor(i/64)}\right)\right)_{\mathrm{mod}(i,64)}$, $i = \overline{0, l-1}$, $(...)_{\mathrm{mod}(i,64)}$ - octetul de pe locul $\mathrm{mod}(i,64)$ din cei 64 de octeți ai lui $Salsa20Expansion_k\left(v, \underline{floor(i/64)}\right)$, iar operația $floor$ determină partea întreagă a numărului nenegativ.

SISTEMUL DE CRIPTARE SALSA20

Algoritmul de criptare fluidă Salsa20

Date de intrare:

text_clar_octeti - codificarea numerică ASCII a textului clar

l1 - lungimea (numărul de elemente ale vectorului) în octeți a lui text_clar_octeti

v - valoare nonce - tablou ce conține 8 elemente (octeți), generate aleator

cheia_k - tablou ce conține 32 sau 16 elemente (octeți), care reprezintă codificarea numerică ASCII pentru cheia secretă k

lung_k - numărul de elemente ale tabloului cheia_k, adică lungimea cheii secrete k

Date de ieșire:

text_cript_octeti - tablou ce conține octeții textului criptat cu algoritmul de criptare fluidă Salsa20

tablou_cheie_fluida - tablou ce conține octeții cheii fluide, generat cu algoritmul Salsa20

SISTEMUL DE CRIPTARE SALSA20

Pașii algoritmului:

```
text_cript_octeti=zeros(1,l1); tablou_cheie_fluida=[];
% prelucrarea cheii secrete:
if lung_k==32
  k0=cheia_k(1:16); k1=cheia_k(17:32); k={k0,k1};
elseif lung_k==16
  k=cheia_k;
end
% se calculează numărul necesar de apelări ale funcției Salsa20Expansion:
if mod(l1,64)==0, nr=floor(l1/64); else, nr=floor(l1/64)+1; end
% Se concatenează într-un tablou nr*64 de octeți întorși de funcția
% Salsa20Expansion în urma a nr apelări ale acesteia:
```

SISTEMUL DE CRIPTARE SALSA20

```
for i=1:nr
 % La fiecare apel al funcției Salsa20Expansion numărul de bloc NB crește cu o unitate:
 NB=i;
 % NB se reprezintă ca și un șir de 8 octeți:
 vi=inverse_littleendian_64bit(NB);
 % șirul pentru valoarea nonce se concatenează cu vi:
 n=[v,vi]; % parametru de intrare pentru funcția de expansiune Salsa20Expansion
 tablou_cheie_fluida=[tablou_cheie_fluida,Salsa20Expansion(lung_k,k,n)];
end
% daca este necesar, se elimina elementele ce nu vor fi utilizate:
if mod(l1,64)~=0, tablou_cheie_fluida(l1+1:end)=[]; end
% Criptarea cu cifrul fluid:
for i=1:l1
 text_cript_octeti(i)=bitxor(text_clar_octeti(i),tablou_cheie_fluida(i));
end
```

SISTEMUL DE CRIPTARE TRIVIUM

Trivium este un cifru fluid sincronizabil proiectat pentru implementare hardware. Conceptul acestuia este bazat pe simplificarea algoritmului cifrului fluid, astfel încât să se păstreze nivelul necesar de securitate și viteza de criptare. Sistemul de criptare Trivium a fost elaborat de Ch. de Cannière și B. Preneel în cadrul proiectului eStream și a fost selectat în portofoliu la profilul implementare hardware. Deși este cel mai simplu cifru din cadrul proiectului eStream, a dat dovadă de o rezistență criptografică demnă de invidiat, ținând cont de simplitatea sa. Algoritmul nu este patentat și a fost specificat ca un standard internațional în cadrul ISO/IEC 29192-3.

Cuvântul „*trivium*" provine din limba latină și se referă la simetria din 3 părți a algoritmului. La fel, putem asocia denumirea algoritmului cu simplitatea acestuia.

SISTEMUL DE CRIPTARE TRIVIUM

Algoritmul generează o cheie fluidă de lungime de până la 2^{64} biți, pornind de la o cheie secretă K și un vector de inițializare IV, ambii pe 80 de biți. Procedura de generare a cheii fluide include două etape: mai întâi, este inițializat tabloul de stare internă a cifrului, folosind cheia secretă și vectorul de inițializare, iar la etapa a doua, starea internă este reînnoită de mai multe ori, până se obțin biții cheii fluide. În continuare, vom examina etapele în ordine inversă, adică mai întâi vom descrie etapa a doua, iar apoi și etapa întâi.

2.2.5.2.1 Generarea cheii fluide

Vom nota elementele tabloului de stare internă pe 288 biți prin $(s_1,...,s_{288})$. Generarea cheii fluide reprezintă un proces iterativ în cadrul căruia sunt extrași 15 biți ai tabloului de stare, care sunt utilizați la actualizarea a trei biți ai stării (avem 3 registre de deplasare) și la determinarea unui bit z_i al cheii fluide. În continuare, biții stării interne sunt supuși unei transformări de rotație, iar procedura este repetată până atunci când sunt generați $N \le 2^{64}$ biți ai cheii fluide, unde N - lungimea în biți a textului clar.

SISTEMUL DE CRIPTARE TRIVIUM

Cifrul Trivium folosește două operații definite peste corpul finit $GF(2) = \mathbb{Z}_2 := \{0,1\}$: operația de adunare „+” (ce corespunde operației logice XOR) și operația de înmulțire „·” (corespunde operației logice AND). În Figura 2.2.15 este dată reprezentarea grafică a algoritmului de generare a cheii fluide.

SISTEMUL DE CRIPTARE TRIVIUM

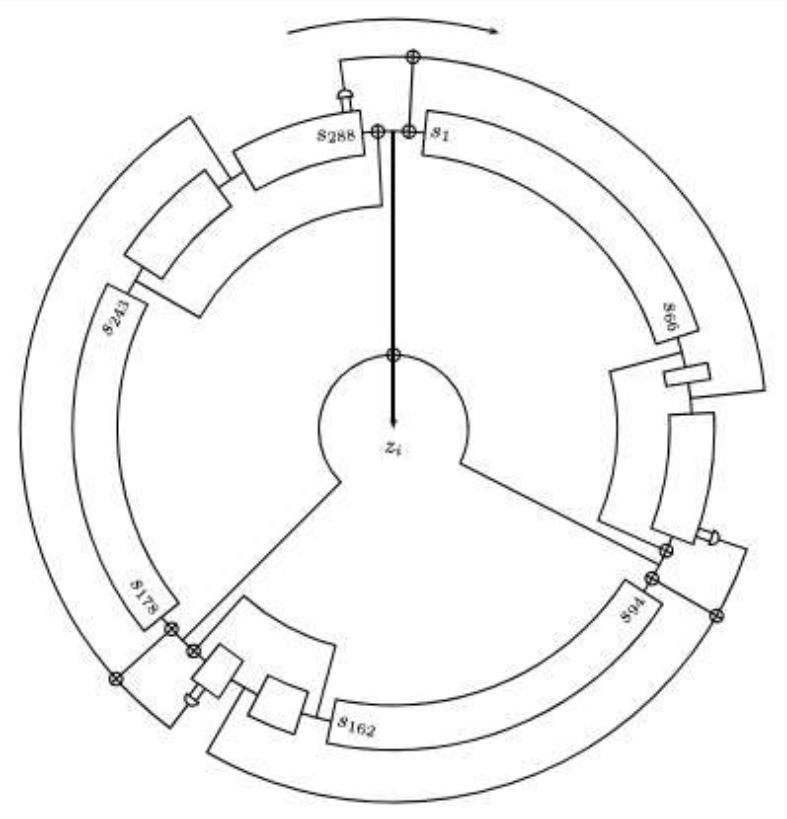

Figura 2.2.15. *Trivium - reprezentarea grafică a procedurii de generare a cheii fluide*

SISTEMUL DE CRIPTARE TRIVIUM

Algoritmul Trivium de generare a cheii fluide

Date de intrare:

N - Numărul de biți din cheia fluidă (egal cu lungimea textului clar)

$s = (s_1, \ldots, s_{288})$ - starea internă

Date de ieșire:

$z = \{z_i\}_{i=1}^{N}$ - N biți ai cheii fluide

Remarcă: Prin „+” este notată operația XOR, iar prin „·” - operația AND

SISTEMUL DE CRIPTARE TRIVIUM

Pentru $i := \overline{1, N}$ execută

{

$t_1 := s_{66} + s_{93}$, $t_2 := s_{162} + s_{177}$, $t_3 := s_{243} + s_{288}$

$z_i := t_1 + t_2 + t_3$

$t_1 := t_1 + s_{91} \cdot s_{92} + s_{171}$, $t_2 := t_2 + s_{175} \cdot s_{176} + s_{264}$, $t_3 := t_3 + s_{286} \cdot s_{287} + s_{69}$

$(s_1, s_2, ..., s_{93}) := (t_3, s_1, ..., s_{92})$, $(s_{94}, s_{95}, ..., s_{177}) := (t_1, s_{94}, ..., s_{176})$, $(s_{178}, s_{179}, ..., s_{288}) := (t_2, s_{178}, ..., s_{287})$

}

Lungimea celor 3 registre de deplasare, menționate anterior, este, respectiv, 93, 84 și 111.

Operația XOR a ieșirilor pentru cele 3 registre generează bitul z_i din cheia fluidă.

2.2.5.2.2 Inițializarea cheii și a valorii IV

La etapa inițială în tabloul de stare sunt scriși 80 de biți ai cheii secrete $K=(K_1,\ldots,K_{80})$ (10 simboluri codificate ASCII) și încă 80 de biți ai vectorului de inițializare $IV=(IV_1,\ldots,IV_{80})$, iar ceilalți biți, cu excepția a trei locații - s_{286},s_{287},s_{288}, sunt inițializați cu zero (astfel avem cel puțin 288-160-3=125 biți nuli). Mai exact, tabloul de stare este inițializat în modul următor:

$$(s_1,s_2,\ldots,s_{93}):=(K_1,\ldots,K_{80},0,\ldots,0),\ (s_{94},s_{95},\ldots,s_{177}):=(IV_1,\ldots,IV_{80},0,\ldots,0),$$
$$(s_{178},s_{179},\ldots,s_{288}):=(0,\ldots,0,1,1,1).$$

În continuare, tabloul de stare este rotit de 4 ori, în modul descris în algoritmul anterior, dar fără să se genereze biții cheii fluide.

SISTEMUL DE CRIPTARE TRIVIUM

Vectorul de inițializare IV este introdus pentru a avea posibilitatea de generare a cheilor fluide distincte, atunci când cheia K rămâne aceeași. Pentru fiecare sesiune de criptare se va alege o nouă valoare IV.

Algoritmul Trivium de inițializare a tabloului de stare
Date de intrare: $K=(K_1,\ldots,K_{80})$ - cheia secretă pe 80 biți $IV=(IV_1,\ldots,IV_{80})$ - vectorul de inițializare pe 80 biți Date de ieșire: $s=(s_1,\ldots,s_{288})$ - tabloul de stare inițial Remarcă: Prin „+” este notată operația XOR, iar prin „·” - operația AND

SISTEMUL DE CRIPTARE TRIVIUM

$(s_1, s_2, \ldots, s_{93}) := (K_1, \ldots, K_{80}, 0, \ldots, 0)$, $(s_{94}, s_{95}, \ldots, s_{177}) := (IV_1, \ldots, IV_{80}, 0, \ldots, 0)$

$(s_{178}, s_{179}, \ldots, s_{288}) := (0, \ldots, 0, 1, 1, 1)$

Pentru $i := \overline{1, 4 \cdot 288}$ execută

{

$t_1 := s_{66} + s_{91} \cdot s_{92} + s_{93} + s_{171}$, $t_2 := s_{162} + s_{175} \cdot s_{176} + s_{177} + s_{264}$, $t_3 := s_{243} + s_{286} \cdot s_{287} + s_{288} + s_{69}$

$(s_1, s_2, \ldots, s_{93}) := (t_3, s_1, \ldots, s_{92})$, $(s_{94}, s_{95}, \ldots, s_{177}) := (t_1, s_{94}, \ldots, s_{176})$, $(s_{178}, s_{179}, \ldots, s_{288}) := (t_2, s_{178}, \ldots, s_{287})$

}

SISTEMUL DE CRIPTARE TRIVIUM

Cifrul Trivium este eficient din punctul de vedere al consumului de resurse și al implementării. Acesta, în mare parte, constă dintr-un registru de deplasare de lungime 288 și un anumit număr de operații booleene. Astfel, este posibil de efectuat în paralel operațiile din cadrul cifrului Trivium. Afirmația se bazează pe aceea că fiecare bit modificat al stării nu mai este utilizat cel puțin pe parcursul următoarelor 64 de iterații. Astfel, 64 de iterații pot fi realizate simultan.

BIBLIOGRAFIE RECOMANDATĂ

1. A. Menezes, P. van Oorschot and S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.
2. A. Klein, Stream Ciphers, London: Springer-Verlag, 2013.
3. D. Salomon, Data privacy and security, Springer, 2003.
4. New Stream Cipher Designs: The eSTREAM Finalists, Series: Lecture Notes in Computer Science 4986: Security and Cryptology, Matthew Robshaw, Olivier Billet (eds.), Springer-Verlag Berlin Heidelberg, 2008.
5. S. Fluhrer, I. Mantin and A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," Selected Areas of Cryptography: SAC 2001, vol. 2259, pp. 1-24, 2001.
6. http://www.ecrypt.eu.org/stream/
7. https://www.cosic.esat.kuleuven.be/nessie/