TEMA 6: CRIPTOGRAFIA SIMETRICĂ

Obiectivele temei:

- Introducere de în criptarea cu cheie secretă.
- Analiza algoritmilor de criptare de tip bloc
- Studiul algoritmilor simetrici de criptare de tip şir.

Cuvinte-cheie:

criptare cu cheie secretă,
cifru bloc,
cutii *S*, cutii *P*,
cifrul Feistel,
algoritmii de criptare
DES, AES,
generarea cheii,
cifruri stream,
cheie fluidă,
cifruri sincrone,
cifruri asincrone,
registre de deplasare cu feedback,
algoritmii de criptare
Seal, A5, RC4.

Era ştiinţifică a criptografiei a început odată cu publicarea în anul 1949 a articolului lui Claude Elwood Shannon (fondatorul teoriei informaţiei, 30.04.1916 – 24.02.2001) „*Communication Theory of Secrecy Systems*". Începând cu acest moment criptografia devine ştiinţific ramură aparte a matematicii, iar articolul lui Shannon a pus bazele ştiinţifice ale *sistemelor de criptare cu cheie secretă* (sisteme simetrice de criptare).

Trebuie de menționat că în criptografia modernă, indiferent de tipul sistemului de criptare, secretul cifrului se bazează pe păstrarea secretului cheii, și nu a algoritmului de criptare. Mai mult, orice sistem de criptare poate fi spart, considerându-se „bun" acel sistem, costul resurselor pentru spargerea căruia depășește valoarea informației criptate sau timpul cheltuit pentru spargere este prea mare pentru ca informația obținută să mai fie utilă.

6.1. Noțiune de criptare cu cheie secretă

Algoritmii de criptare cu cheie secretă sunt caracterizaţi de faptul că folosesc aceeaşi cheie atât în procesul de criptare, cât şi în cel de decriptare (Figura 6.1). Din acest motiv algoritmi cu cheie secretă mai sunt cunoscuţi sub numele de *algoritmi simetrici*. Pentru a utiliza astfel de algoritmi este necesar ca înaintea criptări/decriptării, atât expeditorul cât şi destinatarul să posede deja cheia respectivă. În mod evident, cheia ce caracterizează aceşti algoritmi trebuie să fie cunoscută doar de expeditor şi destinatar.

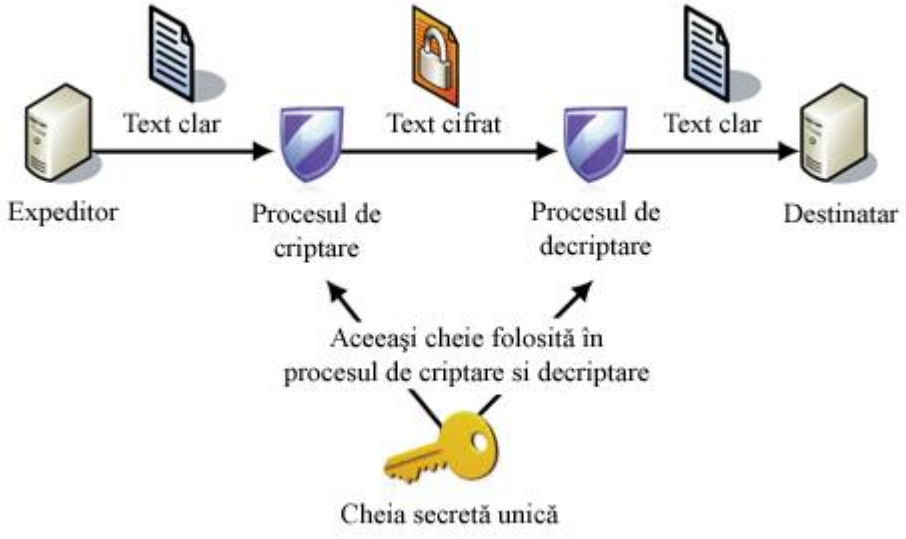

Figura 6.1. *Schema algoritmului de criptare cu cheie secretă*

Criptarea simetrică prezintă avantajul rapidităţii cu care sunt realizate procesele de criptare/decriptare a mesajelor. Succesul sistemului se bazează pe dimensiunea cheii, care astăzi se consideră sigură dacă are nu mai puțin de 128 biţi. Cele trei caracteristici ale criptării simetrice sunt siguranţa, rapiditatea şi volumul mare de date criptate.

Principalul dezavantaj al algoritmilor simetrici constă în faptul că impun un schimb de chei private înainte de a se începe transmisia de date. Altfel spus, pentru a putea fi utilizat un algoritm simetric de criptare, este necesar un canal protejat pentru a transmite cheile de criptare/decriptare sau pot fi utilizați alți algoritmi de criptare pentru a realiza schimbul de chei.

Criptografia simetrică modernă utilizează în principiu aceiaşi algoritmi ca şi criptografia tradiţională (transpoziţia şi substituţia), dar accentul cade pe complexitatea lor. Obiectivul criptografic din actuala perioadă este de a concepe algoritmi de criptare atât de complecşi şi de ireversibili încât atacatorul (sau criptanalistul), chiar şi în situaţia în care are la dispoziţie cantităţi mari de text criptat la alegerea sa, să nu poată face nimic fără cheia secretă. Exemple de cifruri cu cheie secretă sunt: LUCIFER, DES, IDEA, RC5, A5, HC-256, Rijndael, etc.

Algoritmii criptografici folosiţi în sistemele simetrice de criptare se împart în *cifruri bloc* (*block ciphers*) şi *cifruri flux* (sau *cifruri şir - stream ciphers*). Cifrurile flux pot cripta un singur bit de text clar la un moment dat, pe când cifrurile bloc criptează mai mulţi biţi (64, 128, 256 sau alt număr de biţi) la un moment dat.

6.2. Algoritmii de tip bloc

Algoritmii de tip bloc criptează mesajul în blocuri de *n* de biţi. Pentru aceasta se aplică o funcţie matematică între un bloc de biţi ai textului clar şi cheie (care poate varia ca mărime), rezultând acelaşi număr de biţi pentru mesajul criptat. Criptarea și decriptarea se realizată astfel încât să îndeplinească următoarele cerinţe:

- cunoscând un bloc de biţi ai textului clar şi cheia de criptare, sistemul să poată genera rapid un bloc al textului criptat;
- cunoscând un bloc de biţi ai textului criptat şi cheia de criptare/decriptare, sistemul să poată genera rapid un bloc al textului clar;
- cunoscând blocurile textului clar şi ale textului cifrat, să fie dificil să se genereze cheia.

6.2.1. Cifrul Feistel

Algoritmii de tip bloc sunt foarte des utilizați în criptografia modernă, iar mulți dintre aceștia se bazează pe o structură numită *cifru bloc Feistel* (se mai spune *reţeaua*, uneori *schema Feistel*), elaborat de către Horst Feistel[1] în anul 1971. Funcția Feistel operează în *d* runde, de obicei între 12 și 16 runde. Blocul inițial este împărțit în două blocuri egale (L_0, R_0), iar în runda *i* sunt executate următoarele operații (Figura 6.2):

- funcția de rundă $f(i)$ se aplică la partea dreaptă, obținând $f(R_i)$;
- se calculează L_i și R_i nou astfel:

$$L_i = R_{i-1},$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i),$$

unde $\oplus$ este operația OR exclusiv (XOR), *f* este funcţia, de obicei tot XOR, şi K_i este cheia de rundă. Indiferent de natura funcţiei *f*, decifrarea se face astfel:

$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus f(L_i, K_i).$$

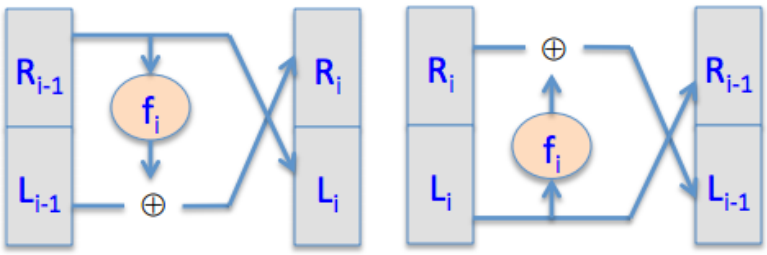

Figura 6.2. *Schema aplicării funcţiei Feistel în runda i*

O structură Feistel are avantajul că cifrarea şi decifrarea sunt foarte similare sau chiar identice în unele cazuri (ceea ce ne aminteşte de Enigma), cerând doar o reversie a cheii. Astfel, dimensiunea codului sau circuitului necesar pentru a implementa un astfel de cifru este practic înjumătăţit. Reţelele Feistel şi construcţii similare combină mai multe „runde” de operaţii repetate pentru a produce o funcţie care conţine cantităţi mari de date. Pe parcursul acestor runde se realizează:

- *amestecarea de biţi* (numită şi permutări pe cutii *P*);

[1] *Horst Feistel* (30 ianuarie 1915 - 14 noiembrie 1990) - un cercetător-criptograf care a lucrat la dezvoltarea algoritmilor de criptare la IBM, unul dintre fondatorii criptografiei moderne ca știință. A adus o mare contribuție la studiul algoritmilor simetrici de criptare și a pus bazele pentru crearea algoritmului de criptare DES.

- *funcţii simple ne-lineare* (numite si substituţii prin cutii *S*);
- *amestecul liniar* (în sensul algebrei modulare) utilizând XOR.

Aceste operații, după cum a spus și Claude Shannon, creează „*confuzie* şi *difuzie*”. Amestecarea de biţi creează difuzia, iar substituţia - confuzia. În criptografie confuzia se referă la a face o relaţie între cheie şi textul cifrat cât de complex şi adânc posibil, iar difuzia este definită ca proprietatea că redundanţa în statisticele textului clar este disipată în statisticele textului cifrat. Difuzia este asociata cu dependenţa biţilor de la ieşire de biţii de la intrare. Într-un cifru cu o difuzie perfecta, doar schimbarea unui bit de la intrare ar schimba întregul text, ceea ce se mai numeşte şi SAC (*Strict Avalanche Criterion*). Feistel utilizează *cutiile P* (*P-box* sau *Permutation-box*) şi amestecul liniar de biţi pentru a atinge o difuzie aproape perfectă şi se poate spune că îndeplineşte condiţiile SAC.

Cutiile-S (*S-box* sau *Substitution-box*) au o importanţă fundamentală în funcţionarea schemei Feistel. Acestea sunt de obicei folosite pentru a ascunde relaţia dintre cheie şi textul cifrat. În general, o cutie *S* ia un număr *m* de biţi de intrare şi îi transformă într-un număr *n* de biţi de ieşire, unde *n* nu e neapărat egal cu *m*. O cutie $S_{m\times n}$ poate fi implementată ca un tabel de 2^m cuvinte de *n* biţi fiecare. În mod normal sunt utilizate tabele fixe, la fel ca în *Data Encryption Standard* (*DES*), dar în unele cifruri tabelele sunt generate dinamic din cheie (de exemplu, *Blowfish*, *Twofish*).

Un exemplu elocvent de tabel fix este tabelul de 6×4 - biţi al unei cutii *S*, și anume S_5 din *DES* (Tabelul 6.1):

S_5		**Cei 4 biţi de mijloc ai intrării**													
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**
Biţii exte-riori	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	**1001**
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100

Tabelul 6.1. *Cutia* $\boldsymbol{S_5}$ *din DES*

Fiind dată o intrare de 6 biţi, ieşirea de 4 biţi este găsită prin selectarea liniei, folosind cei doi biţi exteriori (primul şi ultimul bit), şi a coloanei, utilizând cei patru biţi interiori. De exemplu, o intrare „**011011**” are biţii exteriori „**01**” şi biţii interiori „**1101**”; ieşirea corespunzătoare va fi „**1001**”.

Cutiile de permutare P reprezintă o metodă de amestecare a biţilor, utilizată pentru a permuta sau transpune biţii în intrările cutiilor *S*, menţinând difuzia în timpul transpunerii.

Reţele Feistel au fost introduse pentru prima dată, în domeniul comercial, în cifrul *Lucifer* de la IBM care a fost conceput de însuşi Feistel şi Don Coppersmith. Reţele Feistel au câştigat respect atunci când guvernul SUA a adoptat standardul DES de criptare a datelor care, datorită naturii iterative a construcţiei Feistel, face foarte simple implementările sistemului de criptare în electronică.

Construcţia Feistel este utilizată şi în unii algoritmi care nu sunt cifruri pe blocuri. De exemplu, *Optimal Asymmetric Encryption Padding* (OAEP) utilizează o reţea Feistel simplă pentru a randomiza textele cifrate în unele scheme de cifrare asimetrică.

Dată fiind natura cifrului Feistel, textul cifrat sparge orice convenţie de caractere şi produce numere ce corespund la caractere care nu există. De fapt orice tip de cifru care operează pe blocuri de text sau pe biţi individuali nu avea cum să respecte standardul de caractere. Din această cauză se utilizează la ieşire un codor pentru a reda textului cifrat proprietatea de lizibilitate şi implicit pentru a putea fi transmis prin sistemele informatice.

6.2.2. Algoritmul DES

Plecând de la algoritmul "Lucifer", conceput în Laboratoarele IBM, a fost dezvoltat algoritmul de criptare *DES* (Data Encryption Standard), primul standard de criptare cu cheie simetrică făcut public de guvernul SUA. Algoritmul a fost conceput pentru guvernul Statelor Unite dar şi pentru folosinţă publică.

În mai 1973, revista *Federal Register* a sintetizat principiile care trebuie să stea la baza proiectării unui algoritm criptografic standard:

- algoritmul trebuie să asigure un înalt nivel de securitate;
- algoritmul trebuie să fie complet specificat şi simplu de înţeles;
- securitatea algoritmului trebuie să fie asigurată de cheie şi nu trebuie să depindă de păstrarea secretă a algoritmului;
- algoritmul trebuie să fie disponibil tuturor utilizatorilor;
- algoritmul trebuie să fie adaptabil pentru diverse aplicaţii;
- algoritmul trebuie să fie implementabil pe dispozitivele electronice;
- algoritmul trebuie să fie eficient în utilizare;
- algoritmul trebuie să poată fi validat;
- algoritmul trebuie să fie exportabil.

DES a fost oficial adoptat ca standard federal la 23 noiembrie 1976, iar în 1977 specificaţiile sale au fost făcute publice. Toate operațiile de transformare prezente în DES sunt definite în specificațiile standardului publicat și nu prezintă o informație secretă. Algoritmul DES este o combinaţie complexă, folosind substituţia şi permutarea (transpoziţia). Acest cifru bloc acceptă un bloc de 64 de biţi la intrare şi generează un bloc cifrat de 64 de biţi. Algoritmul este constituit din 16 cicluri repetate (runde) ale blocurilor fundamentale. Textul iniţial este descompus în blocuri de 64 de biţi. Cheia este de 64 biţi din care doar 56 sunt efectivi, ceilalţi fiind biţi de paritate.

Algoritmul foloseşte numai operaţii aritmetice şi logice clasice cu un număr de până la 64 de biţi, ceea ce face relativ uşor de implementat atât software cât mai ales hardware: unul din scopurile

declarate ale algoritmului fiind uşoara lui implementare hardware într-un cip specializat. Parcurgerea celor 16 cicluri are loc după schemele din figurile 6.3 și 6.5.

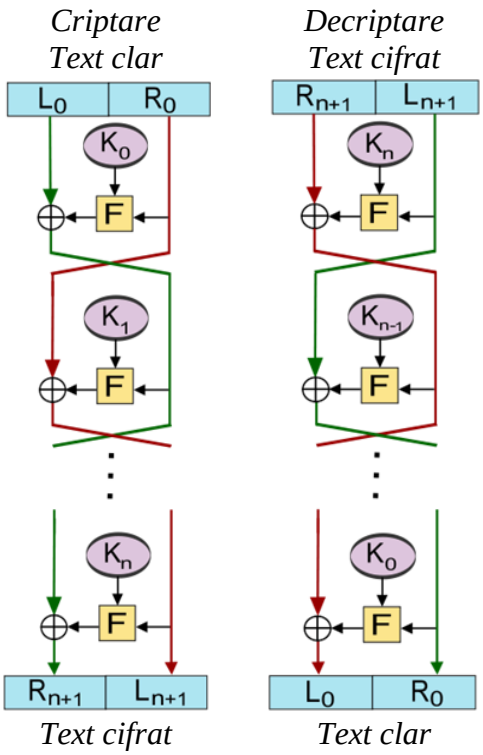

Figura 6.3 *Detalii pentru folosirea algoritmului DES*

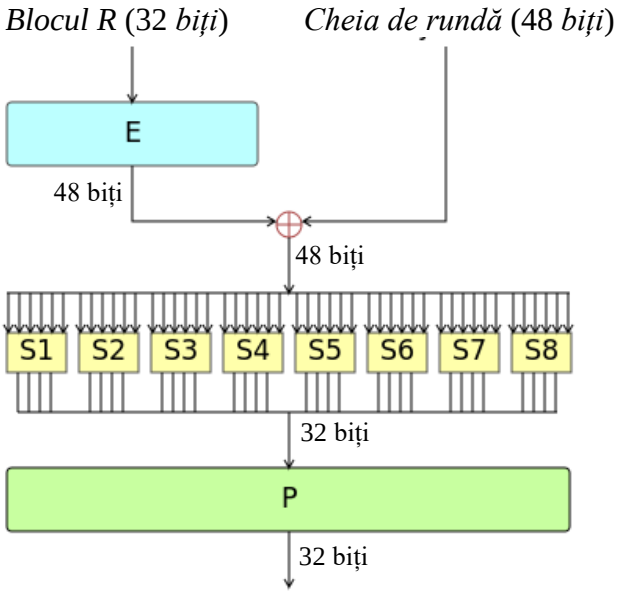

Figura 6.4 *Schema generală a funcției Feistel în DES*

La intrare datele sunt împărţite în blocuri de 64 biţi, care sunt transformate folosind cheia de 64 de biţi. Cei 64 de biţi sunt permutaţi prin „*permutarea inițială IP*” și separați în două, „jumătatea stângă L_0” şi “jumătatea dreaptă R_0”, fiecare de 32 de biţi. În continuare, urmează operaţiile ce constituie un ciclu. Cheia este deplasată la stânga cu un număr de biţi şi permutată: ea se combină cu “partea dreaptă” care apoi se combină cu “partea stângă”, iar rezultatul devine noua “parte dreaptă”; vechea “parte dreaptă” devine noua “parte stângă”. După repetarea acestui ciclu de 16 ori se face permutarea finală care este inversă permutării iniţiale, obținând forma binară a blocului cifrat.

La fiecare ciclu practic au loc patru operaţii separate:

1. partea dreaptă este expandată de la 32 la 48 biţi (permutarea expandată *E*);
2. partea dreaptă este combinată cu cheia de rundă (operația XOR);
3. rezultatul este substituit şi condensat în 32 biţi (cutiile *S*);
4. cei 32 biţi sunt permutaţi (permutarea *P*) şi apoi combinaţi cu partea stângă (XOR) pentru a da o nouă parte dreaptă.

Cheia de 64 biți este supusă unei permutări PC-1, devenind de 56 de biți, fiind excluși biții de paritate. Apoi este împărţită în două părţi de 28 biţi fiecare, c_0 și d_0, deplasate la stânga cu un

număr specificat de biţi apoi reunite şi 48 din cei 56 de biţi sunt permutaţi (permutarea PC-2) şi folosiţi ca o cheie de 48 de biţi (din totalul de 16 chei) de-a lungul ciclului (figura 6.5, b).

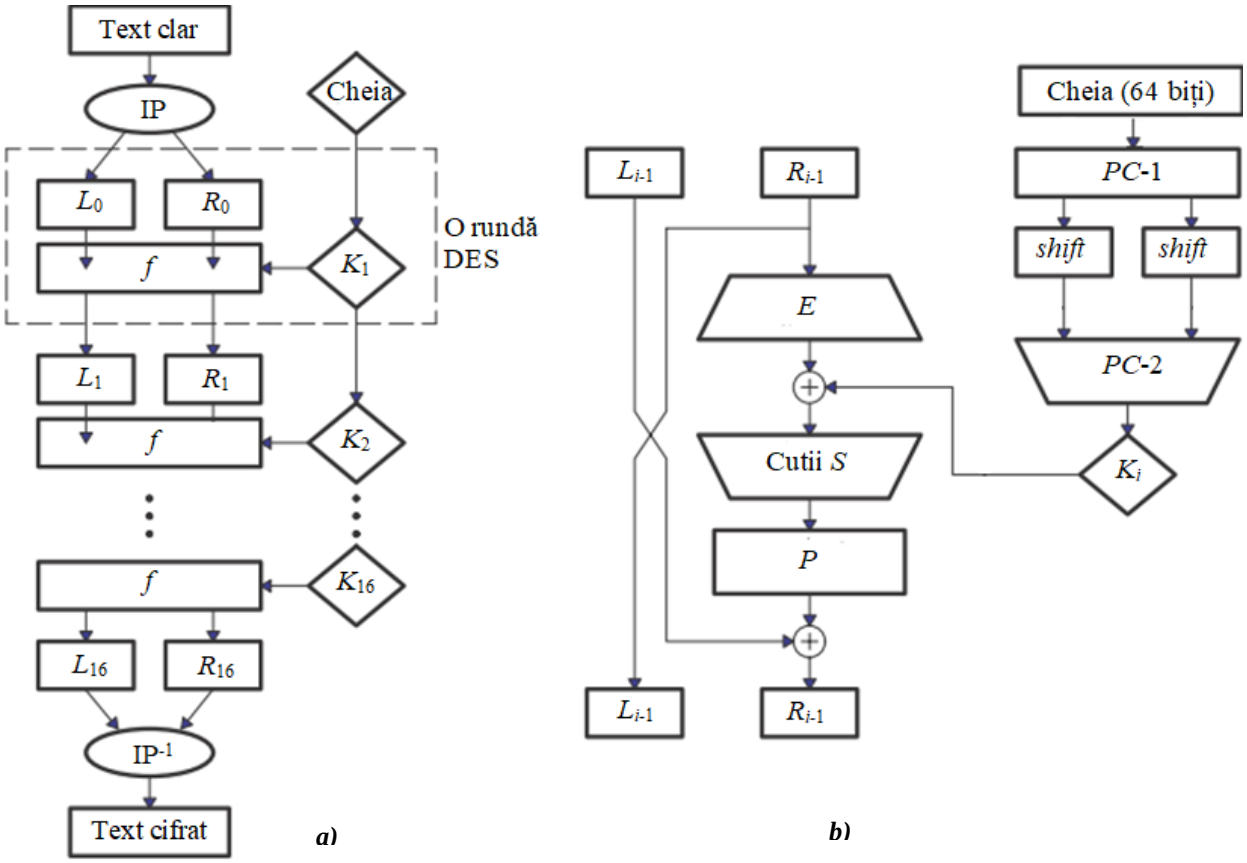

Figura 6.5. *a*) *Schema generală a algoritmului DES, b*) *Schema unei runde DES și generarea cheii*

Datorită lungimii cheii de lucru şi a operaţiilor elementare pe care le foloseşte algoritmul, nu se ridică probleme deosebite într-o implementare software; singura observaţie este că, datorită modulului de lucru (cu secvenţe de date, cu tabele) practic algoritmul este lent într-o implementare software. Modul de concepere îl face însă perfect implementabil hard (într-un cip) ceea ce s-a şi realizat, existând multiple variante de maşini hard de criptare.

Criptanaliza DES. Deşi DES a fost cel mai celebru algoritm al secolului XX este considerat la această oră nesigur pentru multe aplicaţii. Pare paradoxal, dar aceasta este consecinţa măririi considerabile a puterii de calcul de la confirmarea DES-ului ca un standard criptografic şi până în anul 1997, când a fost anunţat concursul pentru noul standard de criptare simetrică. Slăbiciunea pleacă de la lungimea prea mică a cheii de 56 de biţi. Insecuritatea DES-ului pleacă de la premiza că un atac "în forţă" are şanse de reuşită în condiţiile puterii de calcul disponibile astăzi; până în 2004 cel mai eficient atac este datorat criptanalizei liniare care, folosind 2^{43} texte cunoscute, generează o complexitate temporală de 2^{39-43}; în condiţiile unui atac cu text ales complexitatea poate fi redusă de patru ori.

Odată ce DES a devenit vulnerabil au fost aprobate câteva versiuni mai sigure ale DES, implementate pe aceleași platforme ca și DES simplu: 3DES, DES-X, RDES și altele.

DES multiplu. Când s-a descoperit că cheile pe 56 de biți folosite de DES nu sunt suficiente pentru a proteja împotriva atacurilor „brute force", au fost folosite variantele 2DES şi 3DES ca modalităţi simple de a mări spațiul cheilor fără nevoia de a trece la un nou algoritm.

2DES constă în două acționări succesive ale algoritmului DES, cu două chei DES diferite sau egale. Utilizarea sa este pur didactică deoarece este vulnerabilă la atacurile cu întâlnire la mijloc (meet-in-the-middle attack). 2DES este exemplul cel mai des folosit pentru a demonstra viabilitatea unui astfel de atac, dar valoarea sa practică este aproape de zero.

Utilizarea a trei pași este esențială pentru a evita atacurile cu întâlnire la mijloc, deoarece textul cifrat rezultat este mult mai greu de spart folosind căutarea exhaustivă (forţa brută): 2^{112} încercări în loc de 2^{56} încercări.

Cea mai simplă variantă de 3DES funcționează astfel:

$$C = \mathrm{DES}(k_3; \mathrm{DES}(k_2; \mathrm{DES}(k_1; M))),$$

unde M este blocul în clar, iar k_1, k_2 şi k_3 sunt cheile DES. Această variantă este cunoscută sub notația *EEE* deoarece toate cele trei operațiuni efectuate cu cheile sunt criptări. Pentru a simplifica interoperabilitatea între DES și 3DES, pasul din mijloc se înlocuiește de obicei cu decriptarea (modul *EDE*):

$$C = \mathrm{DES}(k_3; \mathrm{DES}^{-1}(k_2; \mathrm{DES}(k_1; M))),$$

și astfel o singură criptare DES cu cheia k poate fi reprezentată ca 3DES-EDE cu cheile $k_1=k_2=k_3=k$. Alegerea decriptării pentru pasul al doilea nu afectează securitatea algoritmului.

O variantă, numită 3DES cu două chei, folosește $k_1=k_3$, reducând astfel lungimea cheii la 112 biți și lungimea de stocare la 128 de biți. Totuși, acest mod de funcționare este susceptibil la unele atacuri cu text clar ales sau text clar cunoscut și astfel este considerat de NIST[2] ca având securitate echivalentă cu doar 80 de biți.

Pe 26 mai 2002, DES a fost înlocuit de AES, Advanced Encryption Standard, după o competiție publică. Chiar și din 2004, DES încă a rămas folosit pe scară largă, iar pe 19 mai 2005, el a fost retras oficial.

6.2.3. Algoritmul AES

În ianuarie 1997 NIST a organizat un concurs de criptografie deschis cercetătorilor din întreaga lume, având ca subiect crearea unui nou standard, care urma să se numească AES – *Advanced Encryption Standard*. Regulile concursului erau:

- algoritmul să fie un cifru bloc simetric;
- proiectul trebuia să fie public;

[2] National Institute of Standards and Technology

– AES trebuia să suporte chei de 128, 192 şi 256 biţi;
– algoritmul trebuia să se poată implementa atât hardware cât şi software;
– AES trebuia să fie un standard public sau oferit cu licenţă nediscriminatorie.

În august 1998 NIST a selectat cinci finalişti pe criterii de securitate, eficienţă, flexibilitate şi cerinţe de memorie. Finaliştii au fost:

1. Rijndael (Joan Daemen şi Vincent Rijmen, 86 de voturi),
2. Serpent (Ross Anderson, Eli Biham, Lars Knudsen, 56 voturi),
3. Twofish (echipa condusă de Bruce Schneier, 31 voturi),
4. RC6 (RSA Laboratories, 23 voturi),
5. MARS (IBM, 13 voturi),

În octombrie 2000 NIST a stabilit câştigătorul. Acesta este algoritmul Rijndael, dezvoltat de doi tineri cercetători belgieni, Joan Daemen şi Vincent Rijmen şi care devine standard guvernamental al SUA. Se spera ca Rjindael să devină standardul criptografic dominant în lume pentru următorii 10 ani, AES depășind așteptările, rămânând și astăzi în topul algoritmilor de criptare cu cheie secretă utilizați în diverse țări și domenii.

Rijndael permite lungimi de chei şi mărimi de blocuri de la 128 de biţi la 256 de biţi, în paşi de câte 32 de biţi. Lungimea cheii şi lungimea blocului pot fi alese în mod independent, dar în practică se presupunea folosirea a două variante: bloc de 128 biţi cu cheie de 128 biţi şi bloc de 128 biţi cu cheie de 256 biţi. Standardul comercial trebuia să devină cel mai probabil varianta 128/128. O cheie de 128 biţi permite un spaţiu al cheilor de 2^{128} chei. În final s-a decis ca în AES numărul de runde să depindă numai de mărimea cheii, fiind 10 pentru cazul cheii de 128 biți, 12 pentru 192 biți şi 14 pentru 256 biți, lungimea blocului fiind aceeași în toate cazurile – 128 biți.

Preliminarii matematice. O descriere matematică detaliată nu este scopul acestui compartiment, limitându-ne la o scurtă descriere teoretică, mai ales că pentru toate operațiile definite în AES sunt elaborate diverse tabele care simplifică lucrurile pentru cei neinițiați în teorie, care, de altfel, este destul de complicată și voluminoasă pentru a fi inclusă în acest manual.

Rijndael se bazează pe teoria câmpului Galois, în sensul că anumite operaţiuni sunt definite la nivel de octet iar octeţii reprezintă elemente în câmpul finit $GF(2^8)$. Cum toate reprezentările câmpului finit $GF(2^8)$ sunt izomorfe, se poate alege reprezentarea clasică polinomială, cu impact pozitiv asupra complexităţii implementării.

Octetul b, format din biţii b_7, b_6, b_5, b_4, b_3, b_2, b_1 şi b_0, este considerat ca fiind un polinom de gradul 7 cu coeficienţi 0 sau 1:

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0$$

Operaţia de adunare este definită ca suma a două polinoame în care coeficienţii se adună modulo 2 şi care corespunde operării XOR a celor doi octeţi corespondenţi. Sunt îndeplinite

axiomele grupului abelian: operaţia este internă, asociativă, comutativă, există element neutru şi element invers.

Operaţia de înmulţire corespunde produsului a două polinoame modulo *m*, unde *m* este un polinom ireductibil de grad 8 şi care pentru AES este

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Înmulţirea este internă (rezultatul este un polinom de grad strict mai mic ca 8), asociativă şi există element neutru. Elementul invers se determină cu algoritmul lui Euclid, iar distributivitatea celor doua operaţii se verifică.

Concluzia este că mulţimea celor 256 de valori posibile ale unui octet, împreună cu cele două operaţiuni definite mai sus formează un câmp algebric finit, respectiv $GF(2^8)$.

În *proiectarea AES* s-a ţinut cont de trei criterii:

- rezistenţa împotriva tuturor atacurilor cunoscute;
- viteza şi compactitatea codului pe un mare număr de platforme;
- simplicitatea proiectării.

Ca şi DES, AES foloseşte substituţii şi permutări, ca şi runde multiple. Spre deosebire de DES, toate operaţiile sunt la nivel de octet, pentru a permite implementări hardware şi software eficiente.

În algoritmul AES rezultatul cifrat intermediar este numit *vector state*, care poate fi reprezentat ca un tabel cu patru linii şi patru coloane, acestea fiind numerotate începând de la 0. Vectorul *state* se iniţializează cu blocul de 128 biţi de text clar (în ordinea coloanelor, cu primii patru octeţi în coloana 0) şi va fi modificat la fiecare pas al calculului, prin substituţii, permutări şi alte transformări, rezultând în final blocul de 128 biţi de text cifrat.

Deoarece fiecare pas este reversibil, decriptarea se poate realiza prin rularea algoritmului de la coadă la cap, sau prin rularea algoritmului de criptare nemodificat, dar folosind tabele diferite (Figura 6.6).

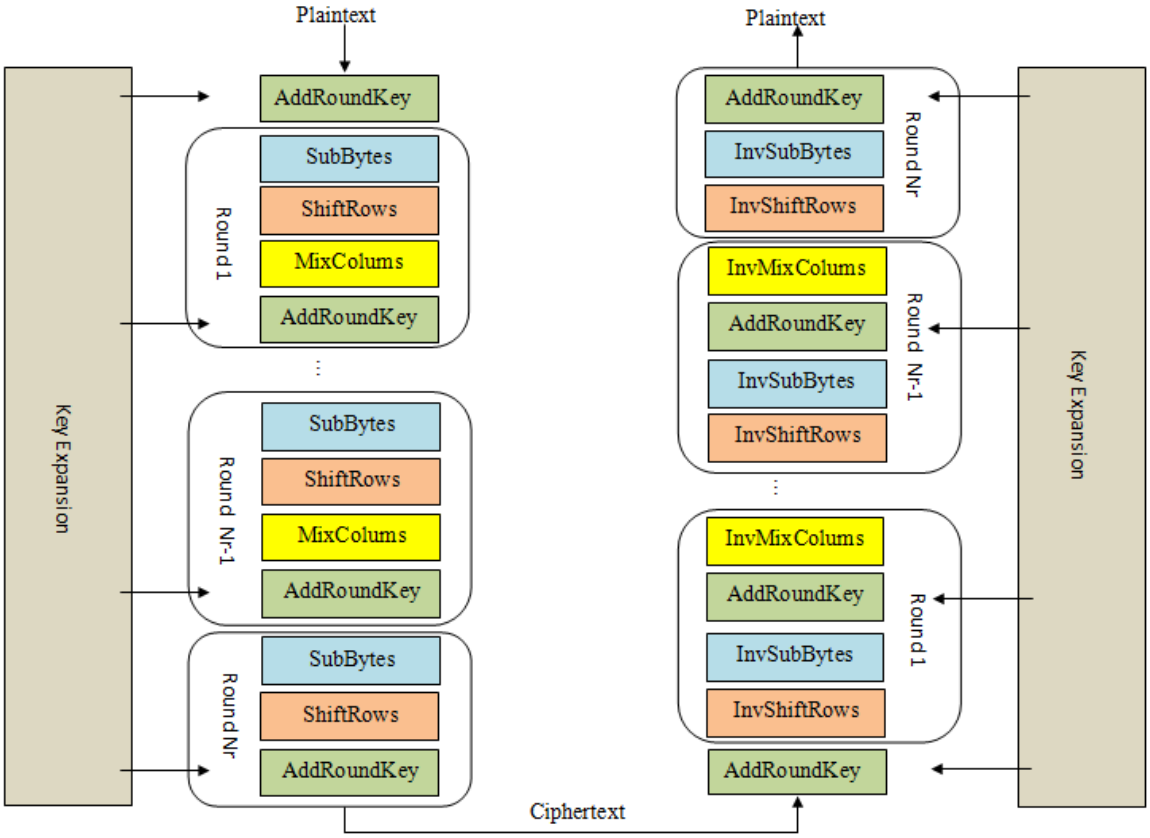

Figura 6.6. Cifrarea și decifrarea AES

Generarea cheilor de rundă AES. Cheia de 128 de biţi este expandată în 11 tabele 4x4. Expandarea este realizată prin rotiri repetate şi operaţii XOR asupra unor grupuri de biţi din cheia originală. Înainte de a începe cele 10 runde, cheia se operează XOR cu vectorul *state* (Figura 6.7).

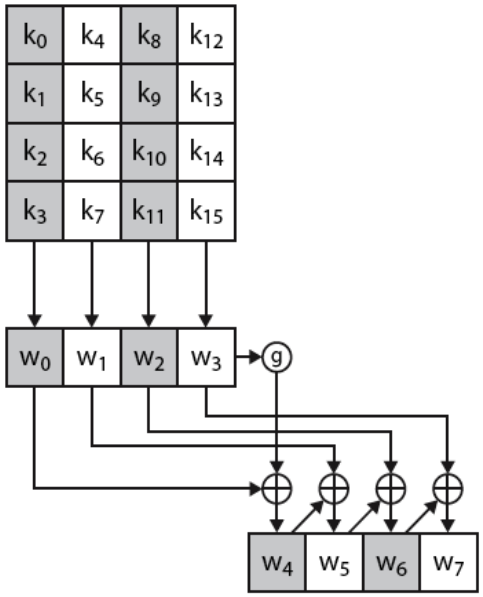

Figura 6.7. *Expandarea cheii AES*

Calculul principal constă în execuţia a 10 runde, folosind cheia k_i la iteraţia i. Fiecare rundă constă în patru paşi, cu excepția ultimei, în care este omis pasul 3 de amestecare a coloanelor.

Pasul 1, the Byte Sub (*BS*) *step* (figura 6.8), realizează o substituţie octet cu octet asupra vectorului *state* folosind o cutie *S*.

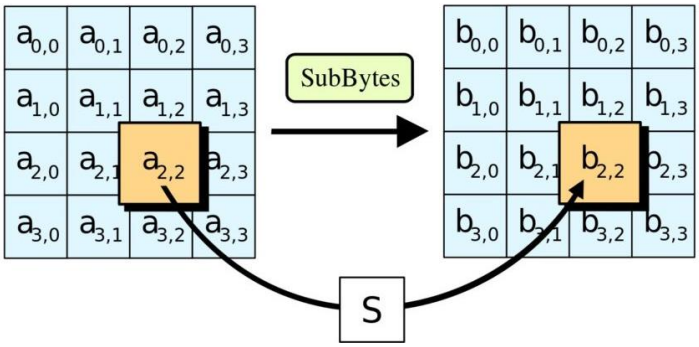

Figura 6.8. *Transformarea Byte Sub*

În acest pas fiecare octet al textului clar este substituit cu un octet extras dintr-o cutie de tip *S*. Cutia de tip *S* este descrisă de o matrice specificată în standardul publicat.

Pasul 2, the Shift Row (*SR*) *step* (Figura 6.9), roteşte la stânga (*left shift*) fiecare din cele 4 rânduri ale vectorului *state*: rândul 0 este rotit cu 0 octeţi, rândul 1 este rotit cu 1 octet, rândul 2 este rotit cu 2 octeţi şi rândul 3 este rotit cu 3 octeţi, realizând difuzia datelor.

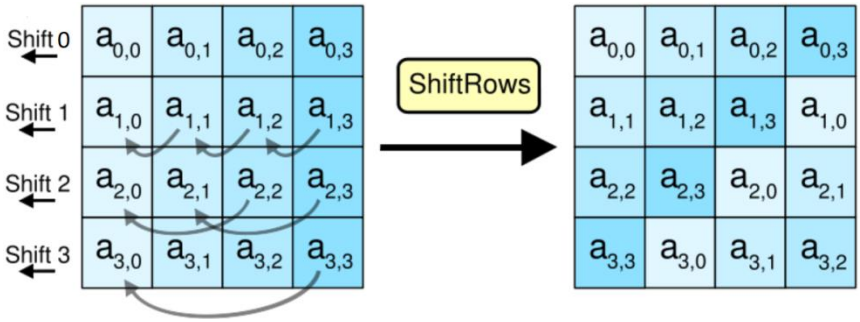

Figura 6.9. *Transformarea ShiftRows*

Pasul 3, *the Mix Column* (*MC*) *step* (Figura 6.10), amestecă fiecare coloană din vectorul *state* independent de celelalte, prin înmulţirea coloanei cu o matrice constantă, multiplicarea fiind realizată folosind câmpul finit Galois GF(2^8).

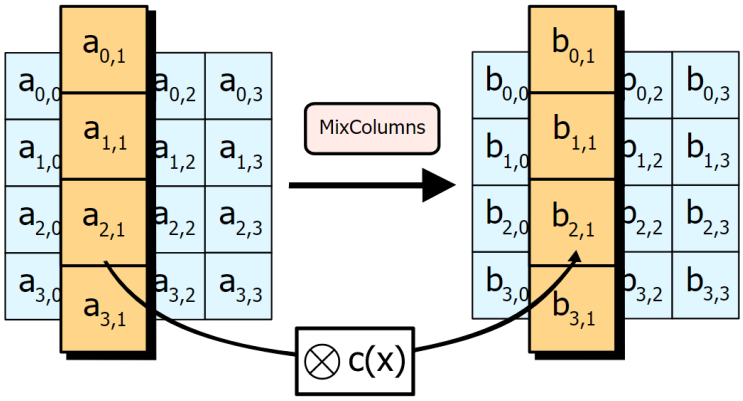

Figura 6.10. *Transformarea MixColumns*

Pasul 4, the Add Round Key step, operează XOR cheia *rk* din runda respectivă cu vectorul *state* (Figura 6.11).

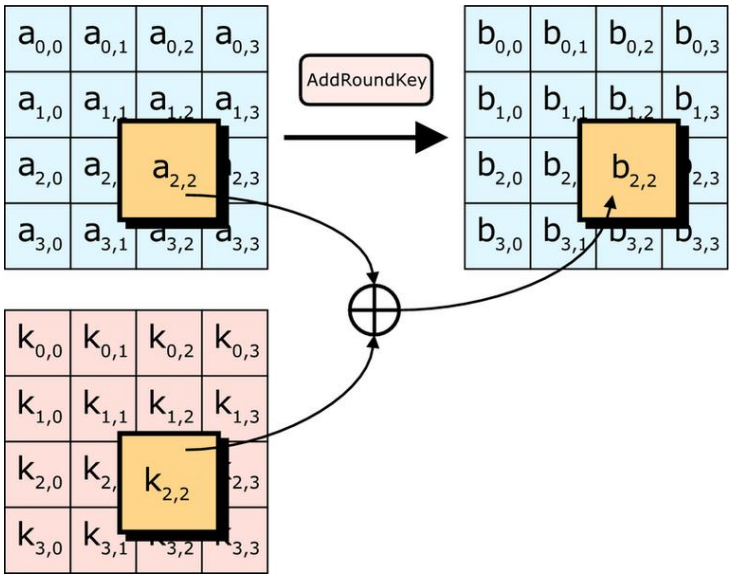

Figura 6.11 *Transformarea Add Round Key*

Pasul *AddRoundKey* (*ARK*) este pasul în care este implicată cheia. El constă într-o simplă operaţie de *SAU exclusiv* pe biţi între stare şi cheia de rundă (o cheie care este unică pentru fiecare iteraţie, cheie calculată pe baza cheii secrete). Operaţia de combinare cu cheia secretă este una extrem de simplă şi rapidă, dar algoritmul rămâne complex, din cauza complexităţii calculului cheilor de rundă (*Key Schedule*), precum şi a celorlalţi paşi ai algoritmului.

Avantajele AES relativ la implementare sunt (amintim că AES a fost aprobat în 2002):

- AES se poate implementa pe un procesor Pentium Pro[3] şi va rula cu o viteză mai mare decât orice alt cifru bloc;
- AES se poate implementa pe un dispozitiv Smart Card, folosind un spaţiu redus de memorie RAM şi un număr redus de cicluri;
- transformarea din cadrul unei runde este paralelă prin proiectare, ceea ce constituie un avantaj pentru viitoarele procesoare;
- AES nu foloseşte operaţii aritmetice, ci doar operaţii la nivel de şiruri de biţi.

Simplitatea proiectării AES:

- AES nu foloseşte componente criptografice externe, cum ar fi cutii *S*, biţi aleatori sau şiruri de cifre din dezvoltarea numărului π;
- AES nu îşi bazează securitatea pe interacţiuni obscure sau greu de înţeles între operaţiuni aritmetice;
- proiectarea clară a AES nu permite ascunderea unei “trape”.

Lungimea variabilă a blocului

[3] *Pentium Pro* este un microprocesor x86 de generația a șasea, dezvoltat și fabricat de Intel, introdus în 1 noiembrie 1995. Proprietăți: max. CPU clock rate - 150 MHz to 200 MHz, FSB speeds - 60 MHz to 66 MHz, min. feature size- 0.35 µm to 0.50 µm, microarchitecture- P6, 1 core, socket 8.

- lungimile de bloc de 192 şi 256 biţi permit construirea unei funcţii hash iterative folosind AES ca funcţie de compresie.

Extensii:

- proiectarea permite specificarea de variante cu lungimi de blocuri şi lungimi de chei aflate între 128 şi 256 biţi, în paşi de câte 32 de biţi;
- deşi numărul de runde în AES este fixat în specificaţiile algoritmului, el poate modificat ca un parametru în cazul unor probleme de securitate.

Limitările AES sunt în legătură cu algoritmul de decriptare:

- algoritmul de decriptare este mai puţin pretabil la implementarea pe un dispozitiv Smart Card, deoarece necesită mai mult cod şi mai multe cicluri;
- implementarea software a AES foloseşte cod şi/sau tabele diferite pentru algoritmul de criptare, respectiv decriptare;
- implementarea hardware a AES a algoritmului de decriptare refoloseşte doar parţial circuitele care implementează algoritmul de criptare.

Rijndael, ca și toți ceilalți algoritmi ajunși în etapa finală de selecție pentru standardul AES, a fost revizuit de NSA[4] și, ca și ceilalți finaliști, este considerat suficient de sigur pentru a fi folosit la criptarea informațiilor guvernamentale americane neclasificate. În iunie 2003, guvernul SUA a decis ca AES să poată fi folosit pentru informații clasificate. Până la nivelul *SECRET*, se pot folosi toate cele trei lungimi de cheie standardizate, 128, 192 și 256 biți. Informațiile *TOP SECRET* (cel mai înalt nivel de clasificare) pot fi criptate doar cu chei pe 256 biți.

Atacul cel mai realizabil împotriva AES este îndreptat împotriva variantelor Rijndael cu număr redus de iterații. AES are 10 iterații la o cheie de 128 de biți, 12 la cheie de 192 de biți și 14 la cheie de 256 de biți, iar cele mai cunoscute atacuri la momentul actual sunt pentru cifruri cu un număr redus de iterații: 7, 8, respectiv 9 pentru cele trei lungimi ale cheii.

B. Schneier[5], unul dintre cei mai de vază criptografi ai momentului, a declarat că deoarece căutarea permanentă rezultă cu succese pentru criptanalişti, pentru siguranţă, în caz de necesitate, e recomandată trecerea de la 10 la 16 runde pentru AES-128, de la 12 la 20 pentru AES 192 şi de la 14 la 28 pentru AES-256.

6.3. Algoritmi simetrici de tip şir

Cifrurile şir (sau cifruri *fluide - stream cyphers*), la fel ca cifrurile bloc, formează o clasă importantă de algoritmi de criptare. Ceea ce le caracterizează şi le diferenţiază faţă de cifrurile bloc este faptul că cifrurile şir procesează textul clar în unităţi oricât de mici, chiar bit cu bit, aplicând funcţia XOR între biţii cheii şi biţii de cifrat, iar funcţia de criptare se poate modifica pe parcursul

[4] National Security Agency

[5] *Bruce Schneier* (n. 15 ianuarie 1963, New York) este un criptograf și scriitor american, specialist în securitatea informatică, autor al câtorva cărți de referință în domeniile sale de specialitate.

criptării. Cifrurile şir sunt algoritmi cu memorie, în sensul că procesul de criptare nu depinde doar de cheie şi de textul clar, ci şi de starea curentă. În cazul în care probabilitatea erorilor de transmisie este mare, folosirea cifrurilor şir este avantajoasă deoarece au proprietatea de a nu propaga erorile. Ele se folosesc şi în cazurile în care datele trebuie procesate una câte una, datorită lipsei de spaţiu de memorie.

Reamintim că într-un sistem de criptare bloc elementele succesive ale textului clar sunt criptate folosind aceeaşi cheie *k*. Adică dacă $M = m_1 m_2 m_3 \ldots$ este textul clar, atunci textul criptat este $C = c_1 c_2 c_3 \ldots = e_k(m_1) e_k(m_2) e_k(m_3) \ldots$, unde m_1, m_2, …, sunt blocuri ale textului clar**.** Pentru cifrurile fluide avem o situație diferită, acolo fiind utilizate cheile fluide.

Cheia fluidă (*keystream*) este un flux de caractere aleatoare sau pseudoaleatoare[6] de forma $k=k_1k_2k_3\ldots$, care sunt combinate cu un text clar pentru a produce un text criptat. Elementele k_i ale cheii fluide pot fi biți, octeți, numere sau caractere reale, cum ar fi *A*, *B*,…, *Z*, în funcție de caz. De obicei, pentru a produce textul cifrat, fiecare element al cheii fluide este supus operației de adunare, scădere sau XOR cu un element al textului clar, folosind aritmetica modulară.

Un algoritm de criptare care criptează un text clar $m = m_1m_2m_3\ldots$ în textul cifrat

$$C = c_1 c_2 c_3 \ldots = e_{k_1}(m_1) e_{k_2}(m_2) e_{k_3}(m_3) \ldots,$$

se numeşte *algoritm fluid de criptare*, unde $k=k_1k_2k_3\ldots$ este o cheie fluidă, iar m_1, m_2, m_3, … - elemente ale textului clar.

Problema principală în sistemele fluide de criptare o reprezintă generarea cheii de criptare. Aceasta se poate realiza fie aleator, fie pe baza unui algoritm care pleacă de la o secvenţă mică de chei de criptare. Un astfel de algoritm se numeşte generator de chei fluide, care este de fapt un generator de numere pseudo-aleatoare[7].

Algoritmii simetrici de tip şir se împart în două clase mari: cifruri şir *sincrone* şi cifruri şir *asincrone*.

6.3.1. Cifruri şir sincrone

Un cifru şir sincron generează şirul de chei independent de textul clar şi de textul cifrat. Criptarea în acest caz poate fi descrisă de următoarele ecuaţii:

$$S_{i+1} = f(S_i, k),$$
$$z_i = g(S_i, k),$$
$$c_i = h(z_i, m_i),$$

[6] *Pseudoaleator* – care satisface unul sau mai multe teste statistice pentru randomizare, dar este produs printr-o procedură matematică definită (spre exemplu de o formulă).

[7] Un *generator de numere pseudoaleatoare* (pseudorandom number generator - PRNG), este un algoritm pentru generarea unei secvențe de numere ale căror proprietăți aproximează proprietățile secvențelor de numere aleatoare. Secvența generată de PRNG nu este cu adevărat aleatoare, deoarece este complet determinată de o valoare inițială, care poate include valori cu adevărat aleatorii. Cu toate că secvențele care sunt mai aproape de aleatoriu cu adevărat pot fi generate folosind generatoare de numere aleatoare, generatoarele de numere pseudo-numere sunt importante în practică pentru viteza lor de generare a numerelor și reproductibilitatea lor.

unde $i = 0, 1, 2, \ldots l$, iar l reprezintă lungimea mesajului.

În această formulă starea iniţială S_0 se determină din cheia k, f este funcţia de stare, g este funcţia care produce şirul de chei z, iar h este funcţia de ieşire, care combină şirul de chei cu textul clar m_i pentru obţinerea textului cifrat c_i.

Printre proprietăţile cifrurilor şir sincrone se numără:

- *sincronizarea* – atât expeditorul cât şi destinatarul trebuie să fie sincronizaţi, în sensul de a folosi aceeaşi cheie şi a opera cu aceeaşi stare respectiv, astfel încât să fie posibilă o decriptare corectă; dacă sincronizarea se pierde prin inserarea sau lipsa unor biţi din textul cifrat transmis, atunci decriptarea eşuează şi poate fi reluată doar prin tehnici suplimentare de re-sincronizare;
- *nepropagarea erorii* – un bit de text cifrat care este modificat în timpul transmisiei nu trebuie să afecteze decriptarea celorlalţi biţi cifraţi;
- *atacuri active* – ca o consecinţă a sincronizării, inserarea, ştergerea sau retransmisia unor biţi de text cifrat de către un adversar activ va cauza o pierdere instantanee a sincronizării şi creşte posibilitatea detectării atacului de către decriptor; ca o consecinţă a nepropagării erorii, un atacator ar putea să modifice biţi aleşi din textul cifrat şi să afle exact ce efect au modificările în textul clar; trebuie deci, să se folosească mecanisme suplimentare de autentificare a expeditorului şi de garantare a integrităţii datelor.

6.3.2. Cifruri şir asincrone

Cifrul şir asincron sau *autosincronizabil* generează şirul de chei ca o funcţie de cheie şi un număr de biţi din textul cifrat anterior. Funcţia de criptare în acest caz poate fi descrisă de următoarele ecuaţii:

$$S_i = (c_{i-t}, c_{i-t+1}, \ldots, c_{i-1}),$$
$$z_i = g(S_i, k),$$
$$c_i = h(z_i, m_i),$$

unde $i = 0, 1, 2, \ldots l$, iar l reprezintă lungimea mesajului.

În această formulă $S_0 = (c_{-t}, c_{-t+1}, \ldots, c_{-1})$, este starea iniţială (nesecretă), k este cheia, g este funcţia care produce şirul de chei z, iar h este funcţia de ieşire care combină şirul de chei cu textul în clar m_i pentru a obţine textul cifrat c_i.

Cifrurile şir asincrone posedă următoarele proprietăţi:

- *auto-sincronizarea* – este posibilă dacă unii biţi din textul cifrat sunt şterşi sau adăugaţi, deoarece decriptarea depinde doar de un număr determinat de biţi cifraţi anterior; astfel de cifruri sunt capabile să-şi restabilească automat procesul de decriptare corectă după pierderea sincronizării;

- *propagarea limitată a erorii* – să presupunem că starea unui cifru şir asincron depinde de *t* biţi cifraţi anteriori; dacă un singur bit cifrat este modificat, şters sau inserat în timpul transmisiei, atunci decriptarea a cel mult *t* biţi următori de text cifrat va fi incorectă, după care se reia decriptarea corectă;
- *atacuri active* – limitarea propagării erorii face ca orice modificare a textului cifrat de către un adversar activ să aibă ca şi consecinţă decriptarea incorectă a altor biţi cifraţi, ceea ce poate mări posibilitatea ca atacul să fie observat de către decriptor; pe de altă parte, datorită auto-sincronizării este mai dificil decât în cazul cifrurilor şir sincrone să se detecteze inserarea, ştergerea sau modificarea unor biţi în textul cifrat; trebuie deci să se folosească mecanisme suplimentare de autentificare a expeditorului şi de garantare a integrităţii datelor.
- *difuzia statisticilor textului în clar* – deoarece fiecare bit de text clar influenţează toţi biţii cifraţi următori, proprietăţile statistice ale textului clar sunt dispersate în textul cifrat; ca o consecinţă, cifrurile şir asincrone trebuie să fie mai rezistente decât cifrurile şir sincrone faţă de atacurile bazate pe redundanţa textului clar.

6.3.3. Registre de deplasare cu feedback

Majoritatea cifrurilor flux folosite în practică sunt proiectate folosind registre de deplasare cu feedback (LFSR – *Linear Feedback Shift Registers*) care sunt simplu de implementat software sau hardware. În structura LFSR se regăsesc următoarele elemente: de întârziere, de adunare modulo 2, de multiplicare scalară modulo 2.

Problema este că aceste implementări sunt ineficiente din punct de vedere al vitezei. Pentru a rezista atacului de corelaţie, funcţia de feedback trebuie să fie un polinom dens, ceea ce presupune multe calcule, care produc la ieşire un singur bit, deci trebuie repetate des. Totuşi, cele mai multe sisteme de criptare militare se bazează pe LFSR.

O metrică importantă folosită pentru a analiza generatoarele bazate pe LFSR este *complexitatea liniară*, definită ca fiind lungimea *n* a celui mai scurt LFSR care poate produce ieşirea generatorului. Orice şir generat de o maşină de stare finită peste un câmp finit are o complexitate liniară finită. Complexitatea liniară este importantă deoarece un algoritm simplu, Berlekamp-Massey (algoritm de căutare a celui mai scurt registru LFSR), poate genera LFSR-ul de definiţie examinând doar 2*n* biţi din cheie, ceea ce însemnă spargerea cifrului şir.

Concluzia este că o complexitate liniară ridicată nu înseamnă neapărat un generator sigur, dar o complexitate liniară scăzută indică un generator fără securitate.

Criptografii încearcă să obţină o complexitate liniară ridicată prin combinarea ieşirilor mai multor LFSR-uri într-un mod nonliniar. Pericolul este ca unul sau mai multe şiruri interne generate – de obicei ieşiri ale LFSR-urilor individuale – să fie corelate cu şirul combinat, ceea ce permite un atac bazat pe algebra liniară numit *atac de corelaţie*. Imunitatea de corelare poate fi precis definită

şi există o legătură între aceasta şi complexitatea liniară. Ideea de bază a atacului de corelaţie este identificarea unor corelaţii între ieşirea generatorului şi ieşirea uneia din componentele sale interne. Apoi, observând şirul de ieşire, se pot obţine informaţii despre ieşirea internă. Folosind aceste informaţii şi alte corelaţii se colectează informaţii despre celelalte ieşiri interne ale generatorului, până când acesta este spart în totalitate.

Printre cifrurile flux mai frecvent utilizate se numără cifrurile *SEAL*, *A*5, *RC4*, *RC*5, *FISH*, etc.

6.3.4. Cifrul SEAL

Cifrul SEAL (Software-Optimized Encryption ALgorithm) este un sistem de criptare aditiv binar (adică are la bază operaţia $\oplus$ - XOR), a fost elaborat în 1993 de către Phillip Rogaway şi Don Coppersmith. Este unul din cele mai eficiente sisteme implementabile pe procesoare de 32 biţi. SEAL este o funcţie pseudo-aleatoare care scoate o cheie fluidă de L biţi folosind un număr n de 32 biţi şi o cheie secretă a de 160 biţi (figura 6.12).

Fie A, B, C, D, X_i, Y_j – cuvinte de 32 biţi. Pentru descrierea algoritmului vom folosi notaţiile:

- $\overline{A}$ – complementul lui A (pe biţi);
- $A \vee B$, $A \wedge B$, $A \oplus B$ – operaţiile *OR*, *AND* şi *XOR* (pe biţi);
- $A << s$ – deplasarea ciclică a lui A spre stânga cu s poziţii;
- $A >> s$ – deplasarea ciclică a lui A spre dreapta cu s poziţii;
- $A + B \pmod{2^{32}}$ – suma lui A şi B (considerate ca numere întregi fără semn);
- $f(B, C, D) = (B \wedge C) \vee (B \wedge D)$;
- $h(B, C, D) = B \oplus C \oplus D$;
- $A \| B$ – concatenarea lui A cu B;
- $(X_1, X_2, ..., X_n) \leftarrow (Y_1, Y_2, ..., Y_n)$ – atribuire simultană.

În continuare o succesiune de 32 biţi se va numi „*cuvânt*” iar succesiunea d 8 biţi se va numi „*octet*”. O succesiune vidă este notată cu λ.

În primul rând trebuie de generat tabelele T, R şi S, fiecare din care este în funcţie de cheia a. Unica misiune a cheii a în algoritm este de a genera aceste tabele prin intermediul funcţiei G, construite în baza cunoscutului algoritm al funcţiei hash SHA-1 care este un standard de stat în SUA (mai bine zis era la momentul respectiv, mai detaliat funcţiile hash vor fi descrise în capitolul 8).

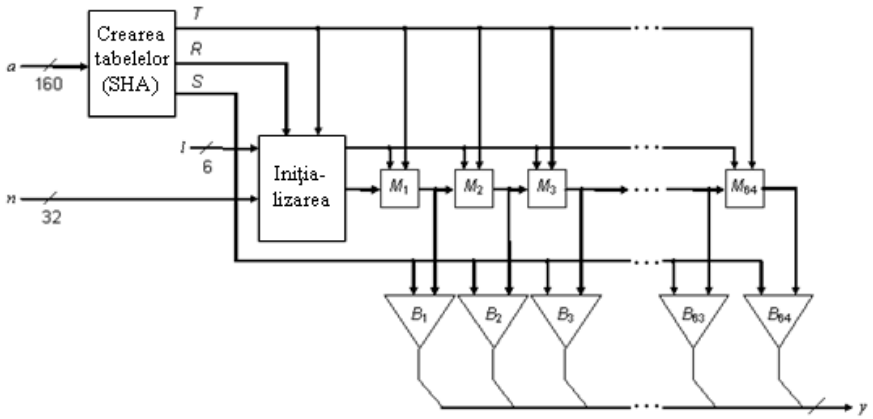

Figura 6.12. *Schema ciclului intern SEAL*

Algoritmul de generare a tabelei G pentru SEAL 2.0 este următorul:

Intrare: un şir a de 160 biţi şi un întreg i ($0 \le i < 2^{32}$).

Ieşire: $G_a(i)$ – şir de 160 biţi.

1. Se definesc constantele (de 32 biţi):

$y_1 = 0x5a827999$,

$y_2 = 0x6ed9eba1$,

$y_3 = 0x8f1bbcdc$,

$y_4 = 0xca62c1d6$

2. *a*. $X_0 \leftarrow i$;

b. *for* $j \leftarrow 1$ *to* 15 *do* $X_j \leftarrow 0x00000000$;

c. *for* $j \leftarrow 16$ *to* 79 *do* $X_j \leftarrow ((X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) << 1)$;

d. $(A, B, C, D, E) \leftarrow (H_0, H_1, H_2, H_3, H_4)$ *unde* $a = H_0 H_1 H_2 H_3 H_4$;

e. (Runda 1): *for* $j \leftarrow 0$ *to* 19 *do*

$t \leftarrow ((A << 5) + f(B, C, D) + E + X_j + y_1)$;

$(A, B, C, D, E) \leftarrow (t, A, B << 30, C, D)$;

f. (Runda 2): *for* $j \leftarrow 20$ *to* 39 *do*

$t \leftarrow ((A << 5) + h(B, C, D) + E + X_j + y_2)$;

$(A, B, C, D, E) \leftarrow (t, A, B << 30, C, D)$;

g. (Runda 3): *for* $j \leftarrow 40$ to 59 do

$t \leftarrow ((A << 5) + h(B, C, D) + E + X_j + y_3)$;

$(A, B, C, D, E) \leftarrow (t, A, B << 30, C, D)$;

h. (Runda 4): *for* $j \leftarrow 60$ to 79 do

$t \leftarrow ((A << 5) + h(B, C, D) + E + X_j + y_4)$;

$(A, B, C, D, E) \leftarrow (t, A, B << 30, C, D)$;

i. $(H_0, H_1, H_2, H_3, H_4) \leftarrow (H_0 + A, H_1 + B, H_2 + C, H_3 + D, H_4 + E)$;

$G_a(i) = H_0 \,||\, H_1 \,||\, H_2 \,||\, H_3 \,||\, H_4$.

Generatorul de chei fluide pentru SEAL 2.0 – **SEAL(*a, n*)**:

Intrare: *a* – cheia secretă (160 biţi), $n \in [0, 2^{32})$ – întreg,

L - lungimea solicitată pentru cheia fluidă.

Ieşire: cheia fluidă *y*, $|y| = L'$,unde L' este primul multiplu de 128 din intervalul $[L, \infty)$.

1. Se generează tabelele *T*, *S*, *R* având ca elemente cuvinte de 32 biţi. Fie funcţia $F_a(i) = H^i_{i(\mathrm{mod}5)}$ unde $H_0^i H_1^i H_2^i H_3^i H_4^i = G_a(\lfloor i/5 \rfloor)$.

a. for i ← 0 *to* 511 *do T*[*i*] ← $F_a(i)$;

b. for j ← 0 *to* 255 *do S*[*j*] ← $F_a(0x00001000 + j)$;

c. for k←0 *to* $4 \cdot \lceil (L-1)/8192 \rceil - 1$ *do R*[*k*] ← $F_a(0x00002000 + k)$;

2. Descrierea procedurii *Initialize* (*n, l, A, B, C, D,* n_1, n_2, n_3, n_4) cu intrările *n* (cuvânt) şi *l* (întreg) şi ieşirile *A, B, C, D,* n_1, n_2, n_3, n_4 (cuvinte).

a. *A*← *n*⊕R[4·*l*], *B*←(*n* >> 8)⊕R[4·*l*+1], *C* ← (*n* >>16)⊕*R*[4·*l*+2],

D ← (*n* >> 24)⊕*R*[4·*l*+3];

b. for j ← 1 *to* 2 *do*

P ← *A* ∧ 0*x*000007*fc*, *B* ← *B* + *T* [*P*/4], *A* ← (*A* >> 9),

P ← *B* ∧ 0*x*000007*fc*, *C* ← *C* + *T* [*P*/4], *B* ← (*B* >> 9),

P ← *C* ∧ 0*x*000007*fc*, *D* ← *D* + *T* [*P*/4], *C* ← (*C* >> 9),

P ← *D* ∧ 0*x*000007*fc*, *A* ← *A* + *T* [*P*/4], *D* ← (*D* >> 9),

(n_1, n_2, n_3, n_4) ← (*D, A, B, C*);

P ← *A* ∧ 0*x*000007*fc*, *B* ← *B* + *T* [*P*/4], *A* ← (*A* >> 9),

P ← *B* ∧ 0*x*000007*fc*, *C* ← *C* + *T* [*P*/4], *B* ← (*B* >> 9),

P ← *C* ∧ 0*x*000007*fc*, *D* ← *D* + *T* [*P*/4], *C* ← (*C* >> 9),

P ← *D* ∧ 0*x*000007*fc*, *A* ← *A* + *T* [*P*/4], *D* ← (*D* >> 9);

3. *l* ← 0, *y* ← λ (şirul vid);

4. *repeat*

a. *Initialize*(*n, l, A, B, C, D,* n_1, n_2, n_3, n_4);

b. for i ← 1 *to* 64 *do*

P ← *A* ∧ 0*x*000007*fc*, *B* ← B + *T* [*P*/4], *A* ← (A >> 9), *B* ← *B* ⊕ *A*;

Q ← *B* ∧ 0*x*000007*fc*, *C* ← C + *T* [*Q*/4], *B* ← (B >> 9), *C* ← *C* ⊕ *B*;

P ← (*P* + *C*) ∧ 0*x*000007*fc*, *D* ← *D* + *T* [*P*/4], C ← (*C* >> 9), *D* ← *D* ⊕ *C*;

Q ← (*Q* + *D*) ∧ 0*x*000007*fc*, *A* ← *A* + *T* [*Q*/4], D ← (*D* >> 9), A ← *A* ⊕ *D*;

P ← (*P* + *A*) ∧ 0*x*000007*fc*, *B* ← *B* + *T* [*P*/4], *A* ← (*A* >> 9);

$Q \leftarrow (Q + B) \wedge 0x000007fc$, $C \leftarrow C + T[Q/4]$, $B \leftarrow (B >> 9)$;

$P \leftarrow (P + C) \wedge 0x000007fc$, $D \leftarrow D + T[P/4]$, $C \leftarrow (C >> 9)$;

$Q \leftarrow (Q + D) \wedge 0x000007fc$, $A \leftarrow A + T[Q/4]$, $D \leftarrow (D >> 9)$;

$y \leftarrow y||(B + S[4 \cdot i - 4]) \,||\, (C \oplus S[4 \cdot i - 3]) \,||\, (D + S[4 \cdot i - 2]) \,||\, (A \oplus S[4 \cdot i - 1])$.

if $|y| \geq L$ *then return*(y) STOP

else if i (*mod* 2) = 1 *then* $(A, C) \leftarrow (A + n_1, C + n_2)$

else $(A, C) \leftarrow (A + n_3, C + n_4)$;

c. $l \leftarrow l + 1$.

Menţionăm că în majoritatea aplicaţiilor pentru *SEAL* 2.0 se foloseşte $L \leq 2^{19}$. Algoritmul funcţionează şi pentru valori mai mari, dar devine ineficient deoarece creşte mult dimensiunea tabelei *R*. O soluţie este concatenarea cheilor fluide *SEAL*(a, 0) || *SEAL*(a, 1) || *SEAL*(a, 2) ||… Deoarece $n < 2^{32}$, se pot obţine astfel chei fluide de lungimi până la 2^{51}, păstrând $L = 2^{19}$.

Pentru algoritmul *SEAL* nu sunt publicate metode eficiente de spargere. În anul 1997 a fost publicat o metodă de atac bazată pe Criptanaliza χ^2 care permite determinarea unei părţi mari din tabelele interne, însă era aplicabilă numai la o versiune simplificată a lui *SEAL*. Trebuie de menţionat aici că Don Coppersmith (care este şi coautor al lui DES) este şi unul dinte cei mai abili criptanalişti. La ziua de azi ultima dintre versiunile acestui cifru este *SEAL* 3.0, care este și cea mai rezistentă la atacuri.

6.3.5. Cifrul A5

Cifrul *A*5 (în prezent numit *A*5/1) este un cifru stream sincron folosit pentru a cripta fluxul de date *GSM*, reprezentând inițial standardul european și cel din SUA pentru telefonia mobilă celulară. A5 criptează linia dintre telefon şi celula de bază, restul legăturii rămânând necriptată. Pentru alte țări a fost elaborată o modificare *A*5/2, care este însă mai puțin sigură. A5 este format din trei LFSR-uri, care au registre de lungime 19, 22 şi respectiv 23. Toate polinoamele de feedback sunt cu un număr redus de coeficienţi. Ieşirea este obţinută prin operarea XOR a celor trei LFSR-uri. A5 foloseşte un clock control variabil. Fiecare registru face un clocking bazat pe bitul central, care este operat XOR cu inversa funcţiei prag (threshold function) a biţilor de la mijlocul celor trei registre. Un registru este sincronizat dacă bitul său de sincronizare (cel portocaliu) este în concordanţă cu unul sau ambii biţi de sincronizare ale celorlalte două registre (figura 6.12). În mod normal, două din LFSR-uri sunt sincronizate la fiecare iteraţie.

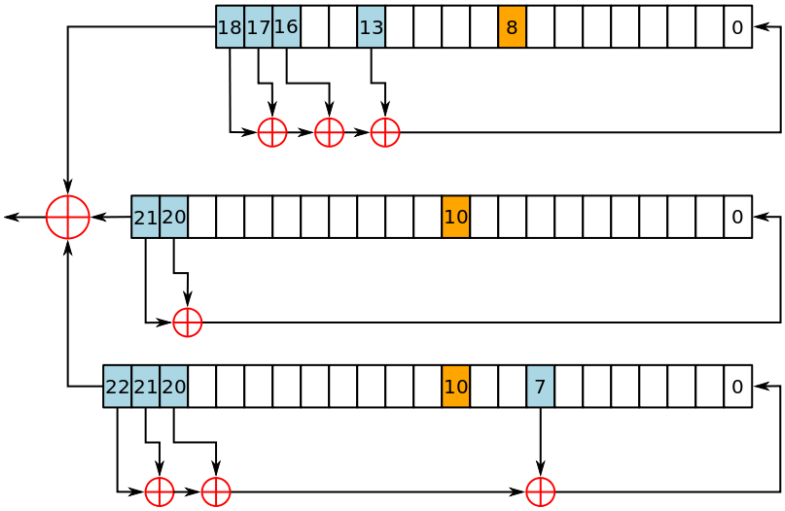

Figura 6.12. *Cifrul A5/1*

Există un atac trivial care necesită 2^{40} criptări: se ghiceşte conţinutul primelor două LFSR, apoi se determină al treilea din şirul generat. În ciuda acestui fapt, A5 este bine proiectat şi este extrem de eficient. El trece cu succes toate testele statistice cunoscute şi singura sa slăbiciune rezidă în faptul că registrele sunt scurte, ceea ce face posibilă o căutare exhaustivă. Variantele A5 cu registre lungi şi polinoame feedback dense au un grad de siguranţă sporit.

6.3.6. Cifrul RC4

Cifrul RC4 este un cifru şir cu cheie de lungime variabilă, dezvoltat în 1987 de către Ron Rivest[8] pentru RSA Data Security și utilizat pe scară largă în diferite sisteme de securitate a informațiilor din rețelele de calculatoare (de exemplu, în protocoalele SSL și TLS, algoritmi de securitate wireless WEP și WPA). În 1994 codul sursă al algoritmului este făcut public pe Internet.

RC4 este un algoritm simplu de realizat software: şirul cheie este independent de textul clar. Funcţionează în baza „cutiilor-*S*”: S_0, S_1, ..., S_{255}. Intrările sunt permutări ale numerelor de la 0 la 255, iar permutarea este o funcţie de o cheie de lungime variabilă. Există doi indici, i şi j, iniţializaţi cu zero. Pentru a genera un octet aleator se procedează astfel:

$$i = (i + 1) \bmod 256;\ j = (j + S_i) \bmod 256;$$

$$T = S_i;\ S_i = S_j;\ S_j = T;\ t = (S_i + S_j) \bmod 256;$$

$$K = S_t\,.$$

Octetul K este operat XOR cu textul clar pentru a produce text cifrat sau operat XOR cu textul cifrat pentru a obţine textul clar. Criptarea este aproape de 10 ori mai rapidă decât DES-ul.

Iniţializarea “cutiilor-*S*” este simplă. Se iniţializează liniar:

$$S_0 = 0,\ S_1 = 1,\ \ldots,\ S_{255} = 255$$

[8] *Ronald Lorin Rivest* (n. 1947, Schenectady, SUA) este un criptograf american, profesor de informatică la MIT. Este cunoscut datorită muncii sale, împreună cu colegii Leonard Adleman și Adi Shamir în domeniul criptografiei cu chei publice și în mod deosebit pentru inventarea algoritmului RSA, pentru care a primit în 2002 premiul Turing de la ACM. Rivest a inventat și algoritmii de criptare cu cheie secretă RC2, RC4, RC5 și a colaborat la crearea RC6. Inițialele RC înseamnă Cifrul Rivest (în engleză Rivest Cipher). De asemenea, a inventat funcțiile hash criptografice MD2, MD4 și MD5.

şi un alt vector de 256 de octeţi cu cheia, repetând cheia, dacă este necesar, pentru a completa vectorul cu componentele $K_0, K_1, \ldots, K_{255}$:

$j = 0;$

For $i = 0$ to 255:

$j = (j + S_i + K_i) \bmod 256;$

se schimbă S_i cu S_j între ele.

Pe parcursul utilizării lui *RC*4 au fost depistate mai multe slăbiciuni, iar cea mai importantă dintre ele provine de la generatorul de chei - primii octeți ai fluxului cheii nu sunt totalmente aleatori și dezvăluie informații despre cheie (atacul *Fluhrer*). Acest lucru poate fi corectat pur și simplu prin eliminarea unei secvențe inițiale a fluxului cheii, modificarea fiind cunoscută ca RC4-drop*N*, unde *N* este de obicei un multiplu al lui 256 care indică numărul de octeți de la începutul fluxului cheii care trebuiesc eliminați. Pentru a consolida RC4 au fost făcute mai multe încercări, cele mai notabile fiind *Spritz*, *RC*4*A*, *VMPC* și *RC*4 +.

În sistemele simetrice de criptare, Alice şi Bob îşi aleg o cheie secretă k care defineşte regulile de criptare e_k şi decriptare d_k. În aproape toate cazurile e_k şi d_k coincid sau se pot deduce imediat una din alta. Un punct slab al sistemelor cu cheie privată este acela că necesită o comunicare prealabilă a cheii între Alice şi Bob printr-un canal sigur, înainte de transmiterea mesajului criptat. Practic, în condiţiile cererii tot mai mari de securizare a comunicaţiilor, acest lucru este din ce în ce mai dificil de realizat. Astfel a apărut necesitatea de a crea sisteme care au alt concept de transmitere a cheii.

6.4. Întrebări și subiecte pentru aprofundarea cunoștințelor și lucrul individual

1. Faceți o analiză comparativă a criptării simetrice cu cifruri bloc și flux.
2. Calculați numărul total de chei ale algoritmilor simetrici necesare pentru schimbul de informație criptată între 20 de persoane.
3. Descrieți principiile generale ale funcționării Data Encryption Standard.
4. Ce reprezintă cutiile S și cutiile P în cifrurile bloc? Care sunt principiile de funcționare ale acestor curii?
5. Realizați un studiu și descrieți în el funcționarea unui algoritm modern de criptare în un din aplicațiile software moderne.
6. Explicați în ce constă deosebirea substanțială a cifrurilor stream și bloc.
7. Descrieți principiul de lucru a algoritmilor de cifrare sincrone și a celor asincrone.
8. Ce reprezintă și unde sunt aplicate registrele de deplasare cu feedback?
9. În algoritmul DES este dat mesajul *M=Alfabetul*. De aflat L_1 pentru primul bloc al mesajului.
10. În algoritmul DES este dat mesajul *M=Alfabetul*. De aflat $E(R_0)$ pentru primul bloc al mesajului.
11. În runda 12 a algoritmului DES am obţinut
    $K_{12} + E(R_{11})$= 011100 010101 011010 110110 101001 101010 011000 101101

Să se determine $S_6(B_6)$.

12. În runda 3 a algoritmului DES în rezultatul aplicării cutiilor-S s-a obţinut:
    0110 0101 0110 1110 1011 1010 0100 1011.
    Să se calculeze R_3, dacă L_2= 0100 0111 0110 1110 1001 1000 0110 1101.
13. La criptarea unui bloc în cifrul DES s-a obţinut
    L16 = 0110 0111 0010 0110 0011 0010 0011 0100,
    R16 = 0000 1010 0110 1100 1100 1001 1001 0111.
    Să se determine blocul criptat al mesajului.