

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 șir.

Cuvinte-cheie:

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

Era științifică a criptografiei a început odată cu publicarea în anul 1949 a articolului lui Claude Elwood Shannon (fondatorul teoriei informației, 30.04.1916 – 24.02.2001) „*Communication Theory of Secrecy Systems*”. Începând cu acest moment criptografia devine științific ramură aparte a matematicii, iar articolul lui Shannon a pus bazele științifice 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. Noțiuni de criptare cu cheie secretă

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



Figura 6.1. Schema algoritmului de criptare cu cheie secretă

Criptarea simetrică prezintă avantajul rapidității 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 biți. Cele trei caracteristici ale criptării simetrice sunt siguranța, rapiditatea și 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 aceiași algoritmi ca și criptografia tradițională (transpoziția și substituția), dar accentul cade pe complexitatea lor. Obiectivul criptografic din actuala perioadă este de a concepe algoritmi de criptare atât de complecși și de ireversibili încât atacatorul (sau criptanalistul), chiar și în situația în care are la dispoziție cantități 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 folosiți în sistemele simetrice de criptare se împart în *cifruri bloc* (*block ciphers*) și *cifruri flux* (sau *cifruri șir* - *stream ciphers*). Cifruile flux pot cripta un singur bit de text clar la un moment dat, pe când cifruile bloc criptează mai mulți biți (64, 128, 256 sau alt număr de biți) la un moment dat.

6.2. Algoritmii de tip bloc

Algoritmii de tip bloc criptează mesajul în blocuri de n de biți. Pentru aceasta se aplică o funcție matematică între un bloc de biți ai textului clar și cheie (care poate varia ca mărime), rezultând același număr de biți pentru mesajul criptat. Criptarea și decriptarea se realizează astfel încât să îndeplinească următoarele cerințe:

- cunoscând un bloc de biți ai textului clar și cheia de criptare, sistemul să poată genera rapid un bloc al textului criptat;
- cunoscând un bloc de biți ai textului criptat și cheia de criptare/decriptare, sistemul să poată genera rapid un bloc al textului clar;
- cunoscând blocurile textului clar și 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 *rețeaua*, uneori *schema Feistel*), elaborat de către Horst Feistel¹ 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 funcția, de obicei tot XOR, și K_i este cheia de rundă. Indiferent de natura funcției f , decifrarea se face astfel:

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

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

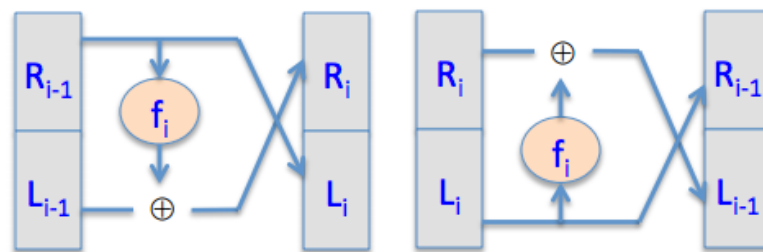


Figura 6.2. Schema aplicării funcției Feistel în runda i

O structură Feistel are avantajul că cifrarea și decifrarea sunt foarte similare sau chiar identice în unele cazuri (ceea ce ne amintește de Enigma), cerând doar o reversie a cheii. Astfel, dimensiunea codului sau circuitului necesar pentru a implementa un astfel de cifru este practic înjumătățit. Rețelele Feistel și construcții similare combină mai multe „runde” de operații repetate pentru a produce o funcție care conține cantități mari de date. Pe parcursul acestor runde se realizează:

- *amestecarea de biți* (numită și 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.

- *funcții simple ne-lineare* (numite și substituții prin cutii S);
- *amestecul liniar* (în sensul algebrei modulare) utilizând XOR.

Aceste operații, după cum a spus și Claude Shannon, creează „confuzie și difuzie”. Amestecarea de biți creează difuzia, iar substituția - confuzia. În criptografie confuzia se referă la a face o relație între cheie și textul cifrat cât de complex și adânc posibil, iar difuzia este definită ca proprietatea că redundanța în statisticele textului clar este disipată în statisticele textului cifrat. Difuzia este asociată cu dependența biților de la ieșire de biții de la intrare. Într-un cifru cu o difuzie perfectă, doar schimbarea unui bit de la intrare ar schimba întregul text, ceea ce se mai numește și SAC (*Strict Avalanche Criterion*). Feistel utilizează *cutiile P* (*P-box* sau *Permutation-box*) și amestecul liniar de biți pentru a atinge o difuzie aproape perfectă și se poate spune că îndeplinește condițiile SAC.

Cutiile-S (*S-box* sau *Substitution-box*) au o importanță fundamentală în funcționarea schemei Feistel. Acestea sunt de obicei folosite pentru a ascunde relația dintre cheie și textul cifrat. În general, o cutie S ia un număr m de biți de intrare și îi transformă într-un număr n de biți de ieșire, 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 biți 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 - biți al unei cutii S , și anume S_5 din *DES* (Tabelul 6.1):

S_5		Cei 4 biți de mijloc ai intrării													
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101
Biții exteriori	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 S_5 din DES*

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

Cutiile de permutare P reprezintă o metodă de amestecare a biților, utilizată pentru a permuta sau transpune biții în intrările cutiilor S , menținând difuzia în timpul transpunerii.

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

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

Dată fiind natura cifrului Feistel, textul cifrat sparge orice convenție de caractere și produce numere ce corespund la caractere care nu există. De fapt orice tip de cifru care operează pe blocuri de text sau pe biți individuali nu avea cum să respecte standardul de caractere. Din această cauză se utilizează la ieșire un codor pentru a reda textului cifrat proprietatea de lizibilitate și 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 și pentru folosință 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 și simplu de înțeles;
- securitatea algoritmului trebuie să fie asigurată de cheie și nu trebuie să depindă de păstrarea secretă a algoritmului;
- algoritmul trebuie să fie disponibil tuturor utilizatorilor;
- algoritmul trebuie să fie adaptabil pentru diverse aplicații;
- 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 specificațiile 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 combinație complexă, folosind substituția și permutarea (transpoziția). Acest cifru bloc acceptă un bloc de 64 de biți la intrare și generează un bloc cifrat de 64 de biți. Algoritmul este constituit din 16 cicluri repetate (runde) ale blocurilor fundamentale. Textul inițial este descompus în blocuri de 64 de biți. Cheia este de 64 biți din care doar 56 sunt efectivi, ceilalți fiind biți de paritate.

Algoritmul folosește numai operații aritmetice și logice clasice cu un număr de până la 64 de biți, ceea ce face relativ ușor de implementat atât software cât mai ales hardware: unul din scopurile

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

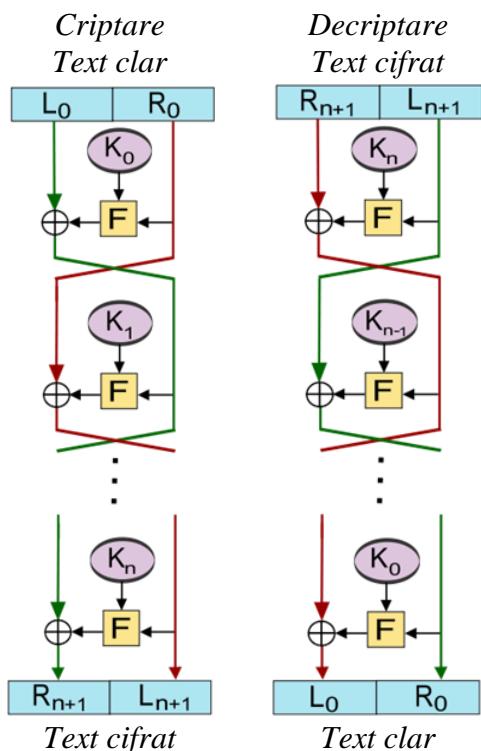


Figura 6.3 Detalii pentru folosirea algoritmului DES

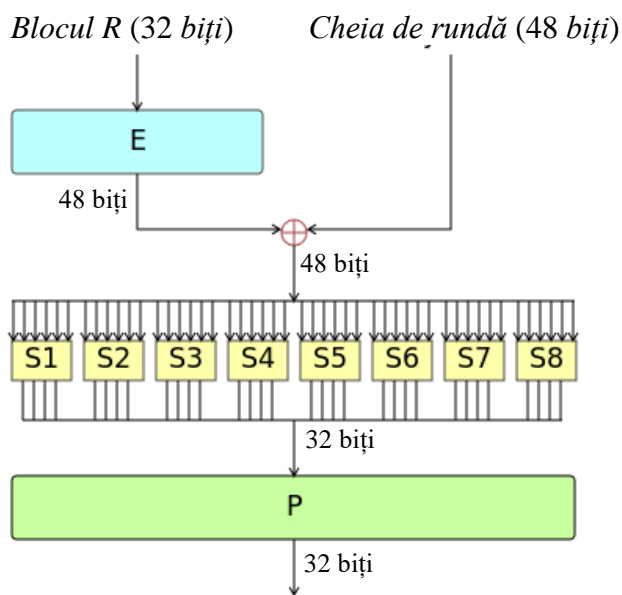


Figura 6.4 Schema generală a funcției Feistel în DES

La intrare datele sunt împărțite în blocuri de 64 biți, care sunt transformate folosind cheia de 64 de biți. Cei 64 de biți sunt permutați prin „permutarea inițială IP” și separați în două, „jumătatea stângă L_0 ” și „jumătatea dreaptă R_0 ”, fiecare de 32 de biți. În continuare, urmează operațiile ce constituie un ciclu. Cheia este deplasată la stânga cu un număr de biți și 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 inițiale, obținând forma binară a blocului cifrat.

La fiecare ciclu practic au loc patru operații separate:

1. partea dreaptă este expandată de la 32 la 48 biți (permutarea expandată E);
2. partea dreaptă este combinată cu cheia de rundă (operația XOR);
3. rezultatul este substituit și condensat în 32 biți (cutiile S);
4. cei 32 biți sunt permutați (permutarea P) și apoi combinați 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 împărțită în două părți de 28 biți fiecare, c_0 și d_0 , deplasate la stânga cu un

număr specificat de biți apoi reunite și 48 din cei 56 de biți sunt permutați (permutarea PC-2) și folosiți ca o cheie de 48 de biți (din totalul de 16 chei) de-a lungul ciclului (figura 6.5, b).

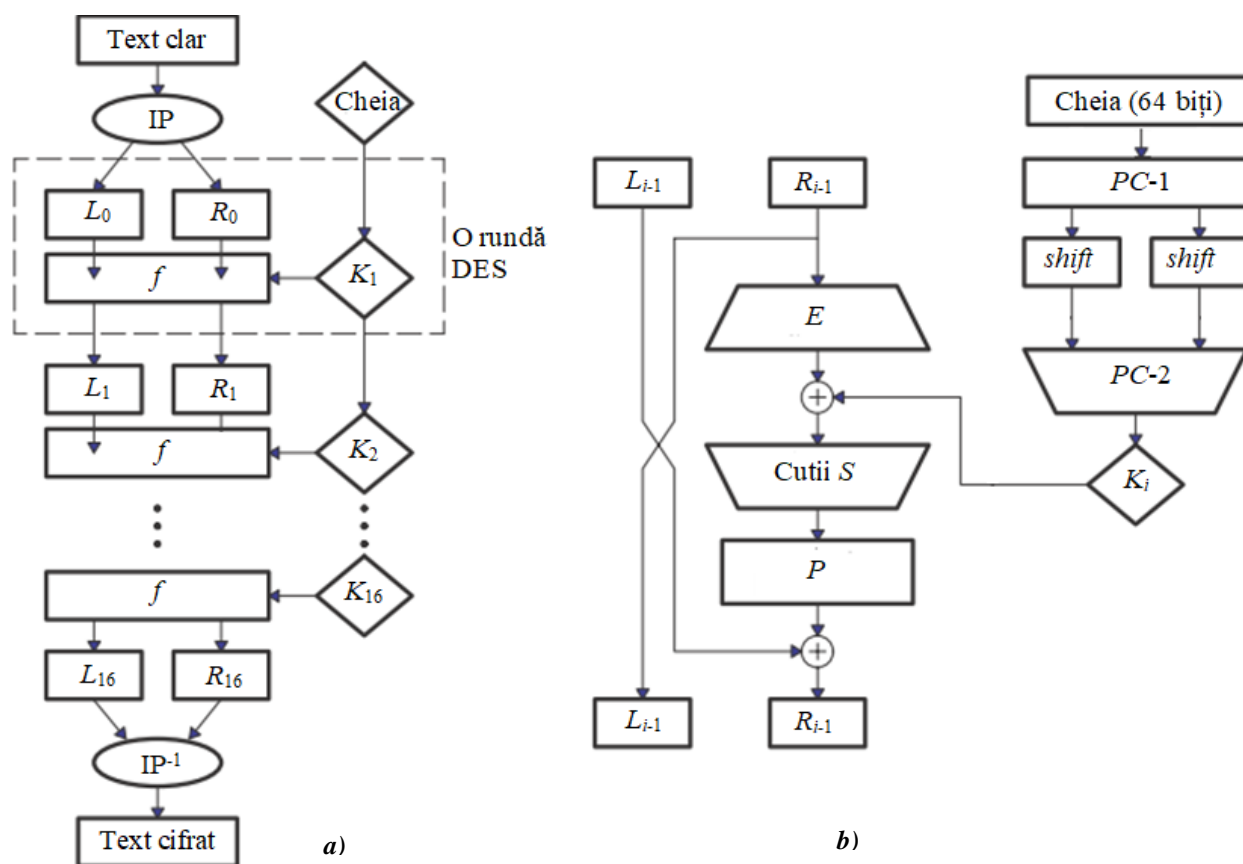


Figura 6.5. a) Schema generală a algoritmului DES, b) Schema unei runde DES și generarea cheii

Datorită lungimii cheii de lucru și a operațiilor elementare pe care le folosește algoritmul, nu se ridică probleme deosebite într-o implementare software; singura observație este că, datorită modulului de lucru (cu secvențe 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 și realizat, existând multiple variante de mașini hard de criptare.

Criptanaliza DES. Deși DES a fost cel mai celebru algoritm al secolului XX este considerat la această oră nesigur pentru multe aplicații. Pare paradoxal, dar aceasta este consecința măririi considerabile a puterii de calcul de la confirmarea DES-ului ca un standard criptografic și până în anul 1997, când a fost anunțat concursul pentru noul standard de criptare simetrică. Slăbiciunea pleacă de la lungimea prea mică a cheii de 56 de biți. Insecuritatea DES-ului pleacă de la premiza că un atac “în forță” are șanse de reușită în condițiile 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 condițiile 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 și 3DES ca modalități 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 (forța brută): 2^{112} încercări în loc de 2^{56} încercări.

Cea mai simplă variantă de 3DES funcționează astfel:

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

unde M este blocul în clar, iar k_1 , k_2 și 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*):

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

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 NIST² 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 și 256 biți;
- algoritmul trebuia să se poată implementa atât hardware cât și software;
- AES trebuia să fie un standard public sau oferit cu licență nediscriminatorie.

În august 1998 NIST a selectat cinci finaliști pe criterii de securitate, eficiență, flexibilitate și cerințe de memorie. Finaliștii au fost:

1. Rijndael (Joan Daemen și 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 câștigătorul. Acesta este algoritmul Rijndael, dezvoltat de doi tineri cercetători belgieni, Joan Daemen și Vincent Rijmen și care devine standard guvernamental al SUA. Se spera ca Rijndael 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 și mărimi de blocuri de la 128 de biți la 256 de biți, în pași de câte 32 de biți. Lungimea cheii și lungimea blocului pot fi alese în mod independent, dar în practică se presupunea folosirea a două variante: bloc de 128 biți cu cheie de 128 biți și bloc de 128 biți cu cheie de 256 biți. Standardul comercial trebuia să devină cel mai probabil varianta 128/128. O cheie de 128 biți permite un spațiu 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 și 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 operațiuni sunt definite la nivel de octet iar octeții 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 complexității implementării.

Octetul b , format din biții $b_7, b_6, b_5, b_4, b_3, b_2, b_1$ și b_0 , este considerat ca fiind un polinom de gradul 7 cu coeficienți 0 sau 1:

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

Operația de adunare este definită ca suma a două polinoame în care coeficienții se adună modulo 2 și care corespunde operării XOR a celor doi octeți corespondenți. Sunt îndeplinite

axiomele grupului abelian: operația este internă, asociativă, comutativă, există element neutru și element invers.

Operația de înmulțire corespunde produsului a două polinoame modulo m , unde m este un polinom ireductibil de grad 8 și care pentru AES este

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

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

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

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

- rezistența împotriva tuturor atacurilor cunoscute;
- viteza și compactitatea codului pe un mare număr de platforme;
- simplitatea proiectării.

Ca și DES, AES folosește substituții și permutări, ca și runde multiple. Spre deosebire de DES, toate operațiile sunt la nivel de octet, pentru a permite implementări hardware și software eficiente.

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

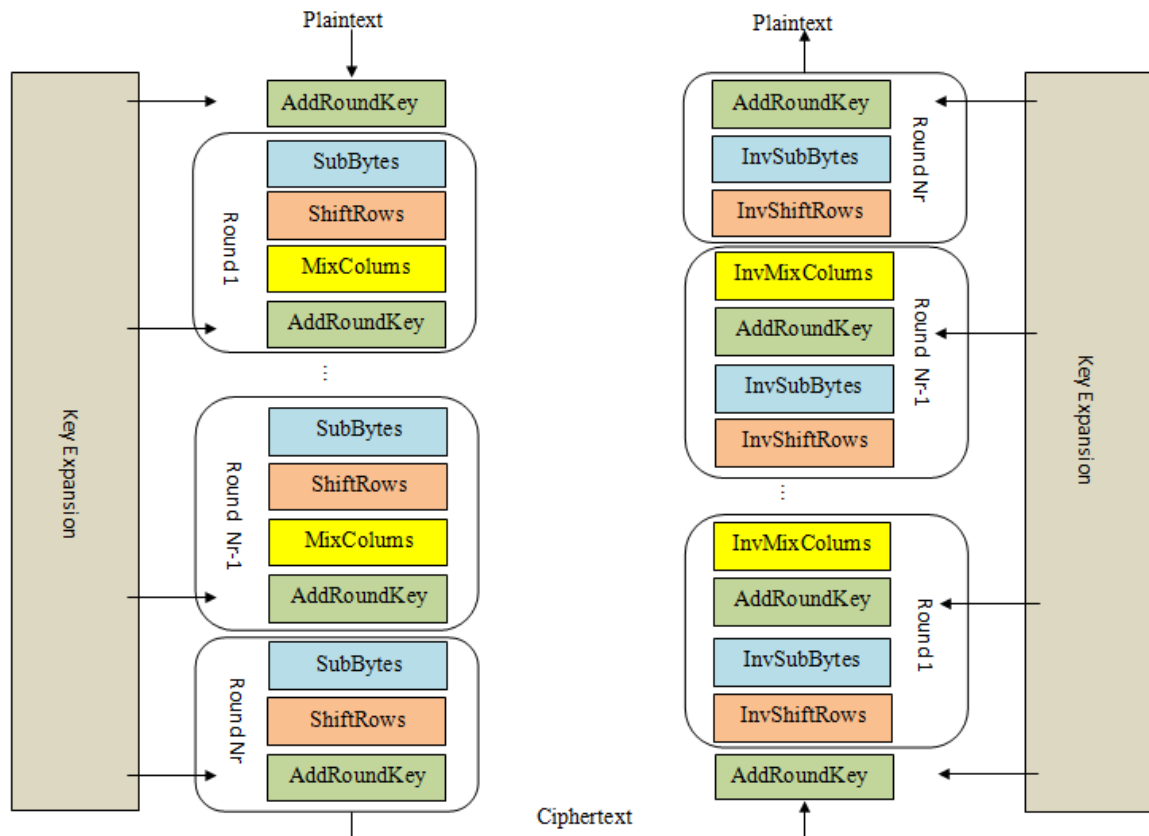


Figura 6.6. Cifrarea și decifrarea AES

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

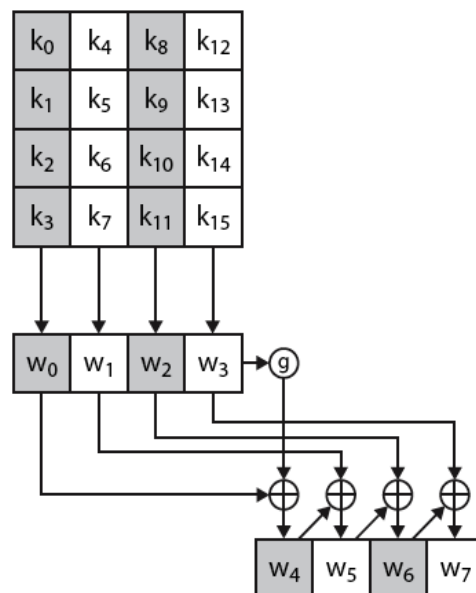


Figura 6.7. Expandarea cheii AES

Calculul principal constă în execuția a 10 runde, folosind cheia k_i la iterația i . Fiecare rundă constă în patru pași, 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 substituție octet cu octet asupra vectorului *state* folosind o cutie *S*.

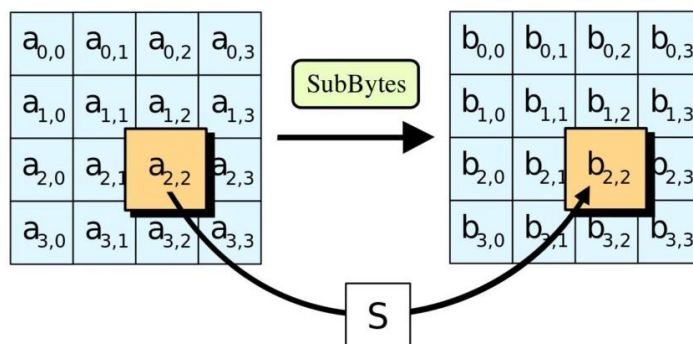


Figura 6.8. Transformarea Byte Sub

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), rotește la stânga (*left shift*) fiecare din cele 4 rânduri ale vectorului *state*: rândul 0 este rotit cu 0 octeți, rândul 1 este rotit cu 1 octet, rândul 2 este rotit cu 2 octeți și rândul 3 este rotit cu 3 octeți, realizând difuzia datelor.

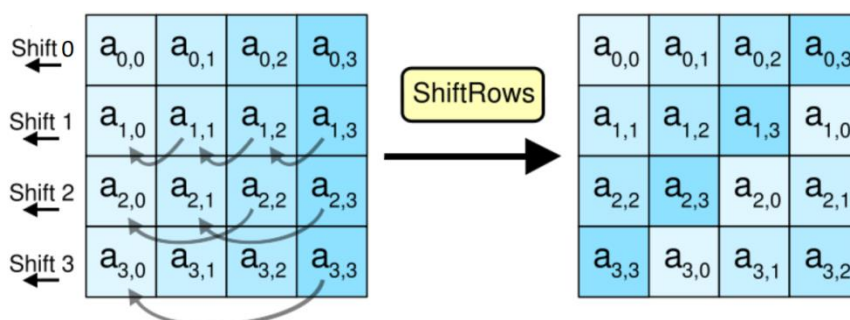


Figura 6.9. Transformarea ShiftRows

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

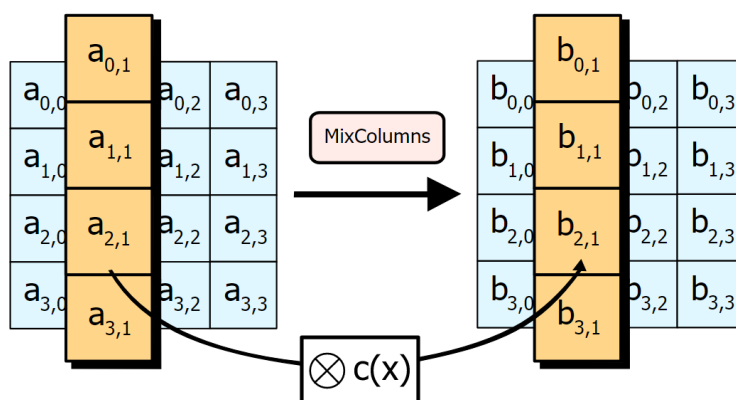


Figura 6.10. Transformarea MixColumns

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

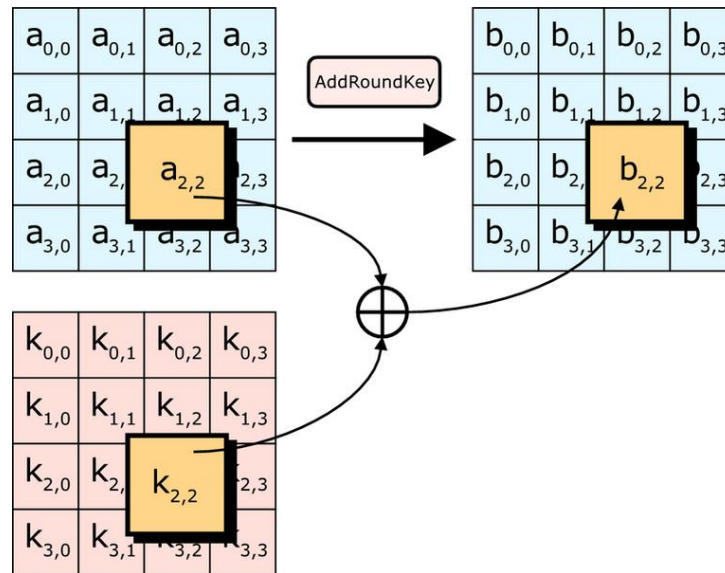


Figura 6.11 Transformarea Add Round Key

Pasul *AddRoundKey* (ARK) este pasul în care este implicată cheia. El constă într-o simplă operație de SAU exclusiv pe biți între stare și cheia de rundă (o cheie care este unică pentru fiecare iterație, cheie calculată pe baza cheii secrete). Operația de combinare cu cheia secretă este una extrem de simplă și rapidă, dar algoritmul rămâne complex, din cauza complexității calculului cheilor de rundă (*Key Schedule*), precum și a celorlalți pași ai algoritmului.

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

- AES se poate implementa pe un procesor Pentium Pro³ și va rula cu o viteză mai mare decât orice alt cifru bloc;
- AES se poate implementa pe un dispozitiv Smart Card, folosind un spațiu redus de memorie RAM și un număr redus de cicluri;
- transformarea din cadrul unei runde este paralelă prin proiectare, ceea ce constituie un avantaj pentru viitoarele procesoare;
- AES nu folosește operații aritmetice, ci doar operații la nivel de șiruri de biți.

Simplitatea proiectării AES:

- AES nu folosește componente criptografice externe, cum ar fi cutii *S*, biți aleatori sau șiruri de cifre din dezvoltarea numărului π ;
- AES nu își bazează securitatea pe interacțiuni obscure sau greu de înțeles între operațiuni 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 și 256 biți permit construirea unei funcții hash iterative folosind AES ca funcție de compresie.

Extensii:

- proiectarea permite specificarea de variante cu lungimi de blocuri și lungimi de chei aflate între 128 și 256 biți, în pași de câte 32 de biți;
- deși numărul de runde în AES este fixat în specificațiile 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 puțin pretabil la implementarea pe un dispozitiv Smart Card, deoarece necesită mai mult cod și mai multe cicluri;
- implementarea software a AES folosește cod și/sau tabele diferite pentru algoritmul de criptare, respectiv decriptare;
- implementarea hardware a AES a algoritmului de decriptare refolosește doar parțial 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 NSA⁴ 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. Schneier⁵, unul dintre cei mai de vază criptografi ai momentului, a declarat că deoarece căutarea permanentă rezultă cu succese pentru criptanaliști, pentru siguranță, în caz de necesitate, e recomandată trecerea de la 10 la 16 runde pentru AES-128, de la 12 la 20 pentru AES 192 și de la 14 la 28 pentru AES-256.

6.3. Algoritmi simetrici de tip șir

Cifrurile șir (sau cifruri *fluide* - *stream cyphers*), la fel ca cifrurile bloc, formează o clasă importantă de algoritmi de criptare. Ceea ce le caracterizează și le diferențiază față de cifrurile bloc este faptul că cifrurile șir procesează textul clar în unități oricât de mici, chiar bit cu bit, aplicând funcția XOR între biții cheii și biții de cifrat, iar funcția 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 șir sunt algoritmi cu memorie, în sensul că procesul de criptare nu depinde doar de cheie și de textul clar, ci și de starea curentă. În cazul în care probabilitatea erorilor de transmisie este mare, folosirea cifrurilor șir este avantajoasă deoarece au proprietatea de a nu propaga erorile. Ele se folosesc și în cazurile în care datele trebuie procesate una câte una, datorită lipsei de spațiu de memorie.

Reamintim că într-un sistem de criptare bloc elementele succesive ale textului clar sunt criptate folosind aceeași cheie k . Adică dacă $M = m_1m_2m_3\dots$ este textul clar, atunci textul criptat este $C = c_1c_2c_3\dots = e_k(m_1)e_k(m_2)e_k(m_3)\dots$, unde m_1, m_2, \dots , 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 pseudoaleatoare⁶ de forma $k=k_1k_2k_3\dots$, 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, \dots, 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 $m = m_1m_2m_3\dots$ în textul cifrat

$$C = c_1c_2c_3\dots = e_{k_1}(m_1)e_{k_2}(m_2)e_{k_3}(m_3)\dots,$$

se numește *algoritm fluid de criptare*, unde $k=k_1k_2k_3\dots$ este o cheie fluidă, iar m_1, m_2, m_3, \dots - 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 secvență mică de chei de criptare. Un astfel de algoritm se numește generator de chei fluide, care este de fapt un generator de numere pseudo-aleatoare⁷.

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

6.3.1. Cifruri șir sincrone

Un cifru șir sincron generează șirul de chei independent de textul clar și de textul cifrat. Criptarea în acest caz poate fi descrisă de următoarele ecuații:

$$\begin{aligned} S_{i+1} &= f(S_i, k), \\ z_i &= g(S_i, k), \\ c_i &= h(z_i, m_i), \end{aligned}$$

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

În această formulă starea inițială S_0 se determină din cheia k , f este funcția de stare, g este funcția care produce șirul de chei z , iar h este funcția de ieșire, care combină șirul de chei cu textul clar m_i pentru obținerea textului cifrat c_i .

Printre proprietățile cifrurilor șir sincrone se numără:

- *sincronizarea* – atât expeditorul cât și destinatarul trebuie să fie sincronizați, în sensul de a folosi aceeași cheie și a opera cu aceeași stare respectiv, astfel încât să fie posibilă o decriptare corectă; dacă sincronizarea se pierde prin inserarea sau lipsa unor biți din textul cifrat transmis, atunci decriptarea eșuează și 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 celorlalți biți cifrați;
- *atacuri active* – ca o consecință a sincronizării, inserarea, ștergerea sau retransmisia unor biți de text cifrat de către un adversar activ va cauza o pierdere instantanee a sincronizării și crește posibilitatea detectării atacului de către decriptator; ca o consecință a nepropagării erorii, un atacator ar putea să modifice biți aleși din textul cifrat și să afle exact ce efect au modificările în textul clar; trebuie deci, să se folosească mecanisme suplimentare de autentificare a expeditorului și de garantare a integrității datelor.

6.3.2. Cifruri șir asincrone

Cifrul șir asincron sau *autosincronizabil* generează șirul de chei ca o funcție de cheie și un număr de biți din textul cifrat anterior. Funcția de criptare în acest caz poate fi descrisă de următoarele ecuații:

$$\begin{aligned}S_i &= (c_{i-t}, c_{i-t+1}, \dots, c_{i-1}), \\z_i &= g(S_i, k), \\c_i &= h(z_i, m_i),\end{aligned}$$

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

În această formulă $S_0 = (c_{-t}, c_{-t+1}, \dots, c_{-1})$, este starea inițială (nesecretă), k este cheia, g este funcția care produce șirul de chei z , iar h este funcția de ieșire care combină șirul de chei cu textul în clar m_i pentru a obține textul cifrat c_i .

Cifrurile șir asincrone posedă următoarele proprietăți:

- *auto-sincronizarea* – este posibilă dacă unii biți din textul cifrat sunt șterși sau adăugați, deoarece decriptarea depinde doar de un număr determinat de biți cifrați anterior; astfel de cifruri sunt capabile să-și restabilească automat procesul de decriptare corectă după pierderea sincronizării;

- *propagarea limitată a erorii* – să presupunem că starea unui cifru șir asincron depinde de t biți cifrați anteriori; dacă un singur bit cifrat este modificat, șters sau inserat în timpul transmisiei, atunci decriptarea a cel mult t biți 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 și consecință decriptarea incorectă a altor biți cifrați, ceea ce poate mări posibilitatea ca atacul să fie observat de către decriptator; pe de altă parte, datorită auto-sincronizării este mai dificil decât în cazul cifrurilor șir sincrone să se detecteze inserarea, ștergerea sau modificarea unor biți în textul cifrat; trebuie deci să se folosească mecanisme suplimentare de autentificare a expeditorului și de garantare a integrității datelor.
- *difuzia statisticilor textului în clar* – deoarece fiecare bit de text clar influențează toți biții cifrați următori, proprietățile statistice ale textului clar sunt dispersate în textul cifrat; ca o consecință, cifrurile șir asincrone trebuie să fie mai rezistente decât cifrurile șir sincrone față de atacurile bazate pe redundanța 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 corelație, funcția de feedback trebuie să fie un polinom dens, ceea ce presupune multe calcule, care produc la ieșire un singur bit, deci trebuie repetate des. Totuși, 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 ieșirea generatorului. Orice șir generat de o mașină 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 definiție examinând doar $2n$ biți din cheie, ceea ce înseamnă spargerea cifrului șir.

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

Criptograful încearcă să obțină o complexitate liniară ridicată prin combinarea ieșirilor mai multor LFSR-uri într-un mod nonliniar. Pericolul este ca unul sau mai multe șiruri interne generate – de obicei ieșiri ale LFSR-urilor individuale – să fie corelate cu șirul combinat, ceea ce permite un atac bazat pe algebra liniară numit *atac de corelație*. Imunitatea de corelare poate fi precis definită

și există o legătură între aceasta și complexitatea liniară. Ideea de bază a atacului de corelație este identificarea unor corelații între ieșirea generatorului și ieșirea uneia din componentele sale interne. Apoi, observând șirul de ieșire, se pot obține informații despre ieșirea internă. Folosind aceste informații și alte corelații se colectează informații despre celelalte ieșiri interne ale generatorului, până când acesta este spart în totalitate.

Printre cifrurile flux mai frecvent utilizate se numără cifrurile *SEAL*, *A5*, *RC4*, *RC5*, *FISH*, etc.

6.3.4. Cifrul SEAL

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

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

- \bar{A} – complementul lui A (pe biți);
- $A \vee B, A \wedge B, A \oplus B$ – operațiile *OR*, *AND* și *XOR* (pe biți);
- $A \ll s$ – deplasarea ciclică a lui A spre stânga cu s poziții;
- $A \gg s$ – deplasarea ciclică a lui A spre dreapta cu s poziții;
- $A + B \pmod{2^{32}}$ – suma lui A și B (considerate ca numere întregi fără semn);
- $f(B, C, D) = (B \wedge C) \vee (B \wedge D)$;
- $h(B, C, D) = B \oplus C \oplus D$;
- $A \parallel B$ – concatenarea lui A cu B ;
- $(X_1, X_2, \dots, X_n) \leftarrow (Y_1, Y_2, \dots, Y_n)$ – atribuire simultană.

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

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

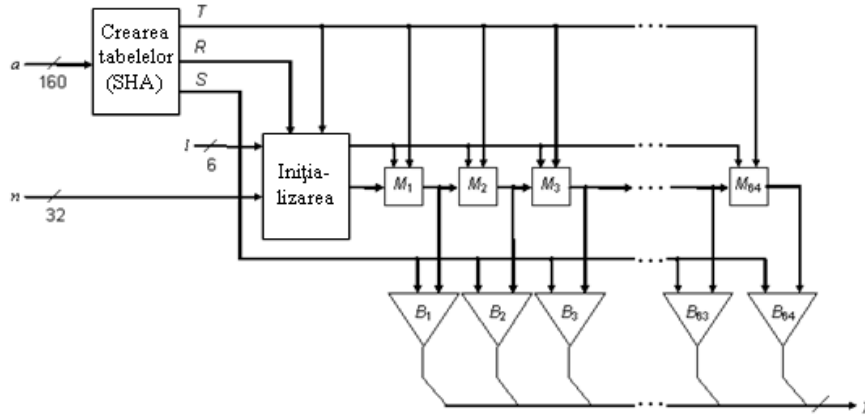


Figura 6.12. Schema ciclului intern SEAL

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

Intrare: un șir a de 160 biți și un întreg i ($0 \leq i < 2^{32}$).

Ieșire: $G_a(i)$ – șir de 160 biți.

1. Se definesc constantele (de 32 biți):

$$y_1 = 0x5a827999,$$

$$y_2 = 0x6ed9eba1,$$

$$y_3 = 0x8f1bbcdc,$$

$$y_4 = 0xca62c1d6$$

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 $X_j \leftarrow ((X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) \ll 1)$;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

L - lungimea solicitată pentru cheia fluidă.

Ieșire: 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 biți. Fie funcția $F_a(i) = H_{i(\bmod 5)}^i$ unde $H_0^i H_1^i H_2^i H_3^i H_4^i = G_a(\lfloor i/5 \rfloor)$.

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

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

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

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

a. $A \leftarrow n \oplus R[4 \cdot l]$, $B \leftarrow (n \gg 8) \oplus R[4 \cdot l + 1]$, $C \leftarrow (n \gg 16) \oplus R[4 \cdot l + 2]$,

$D \leftarrow (n \gg 24) \oplus R[4 \cdot l + 3]$;

b. for $j \leftarrow 1$ to 2 do

$P \leftarrow A \wedge 0x000007fc$, $B \leftarrow B + T[P/4]$, $A \leftarrow (A \gg 9)$,

$P \leftarrow B \wedge 0x000007fc$, $C \leftarrow C + T[P/4]$, $B \leftarrow (B \gg 9)$,

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

$P \leftarrow D \wedge 0x000007fc$, $A \leftarrow A + T[P/4]$, $D \leftarrow (D \gg 9)$,

$(n_1, n_2, n_3, n_4) \leftarrow (D, A, B, C)$;

$P \leftarrow A \wedge 0x000007fc$, $B \leftarrow B + T[P/4]$, $A \leftarrow (A \gg 9)$,

$P \leftarrow B \wedge 0x000007fc$, $C \leftarrow C + T[P/4]$, $B \leftarrow (B \gg 9)$,

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

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

3. $l \leftarrow 0$, $y \leftarrow \lambda$ (șirul vid);

4. repeat

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

b. for $i \leftarrow 1$ to 64 do

$P \leftarrow A \wedge 0x000007fc$, $B \leftarrow B + T[P/4]$, $A \leftarrow (A \gg 9)$, $B \leftarrow B \oplus A$;

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

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

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

$P \leftarrow (P + A) \wedge 0x000007fc$, $B \leftarrow B + T[P/4]$, $A \leftarrow (A \gg 9)$;

```

 $Q \leftarrow (Q + B) \wedge 0x000007fc, C \leftarrow C + T [Q/4], B \leftarrow (B \gg 9);$ 
 $P \leftarrow (P + C) \wedge 0x000007fc, D \leftarrow D + T [P/4], C \leftarrow (C \gg 9);$ 
 $Q \leftarrow (Q + D) \wedge 0x000007fc, A \leftarrow A + T [Q/4], D \leftarrow (D \gg 9);$ 
 $y \leftarrow y // (B + S[4 \cdot i - 4]) // (C \oplus S[4 \cdot i - 3]) // (D + S[4 \cdot i - 2]) // (A \oplus S[4 \cdot i - 1]).$ 
if  $|y| \geq L$  then return(y) STOP
else if  $i \pmod{2} = 1$  then  $(A, C) \leftarrow (A + n_1, C + n_2)$ 
else  $(A, C) \leftarrow (A + n_3, C + n_4);$ 
c.  $l \leftarrow l + 1.$ 

```

Menționăm că în majoritatea aplicațiilor pentru *SEAL* 2.0 se folosește $L \leq 2^{19}$. Algoritmul funcționează și pentru valori mai mari, dar devine ineficient deoarece crește mult dimensiunea tabelii *R*. O soluție este concatenarea cheilor fluide $SEAL(a, 0) // SEAL(a, 1) // SEAL(a, 2) // \dots$. Deoarece $n < 2^{32}$, se pot obține 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 părți mari din tabelele interne, însă era aplicabilă numai la o versiune simplificată a lui *SEAL*. Trebuie de menționat aici că Don Coppersmith (care este și coautor al lui DES) este și unul dintre cei mai abili criptanalști. 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 A5 (în prezent numit A5/1) este un cifru stream sincron folosit pentru a cripta fluxul de date *GSM*, reprezentând inițial standardul european și cel din SUA pentru telefonie mobilă celulară. A5 criptează linia dintre telefon și celula de bază, restul legăturii rămânând necriptată. Pentru alte țări a fost elaborată o modificare A5/2, care este însă mai puțin sigură. A5 este format din trei LFSR-uri, care au registre de lungime 19, 22 și respectiv 23. Toate polinoamele de feedback sunt cu un număr redus de coeficienți. Ieșirea este obținută prin operarea XOR a celor trei LFSR-uri. A5 folosește un clock control variabil. Fiecare registru face un clocking bazat pe bitul central, care este operat XOR cu inversa funcției prag (threshold function) a biților de la mijlocul celor trei registre. Un registru este sincronizat dacă bitul său de sincronizare (cel portocaliu) este în concordanță cu unul sau ambii biți de sincronizare ale celorlalte două registre (figura 6.12). În mod normal, două din LFSR-uri sunt sincronizate la fiecare iterație.

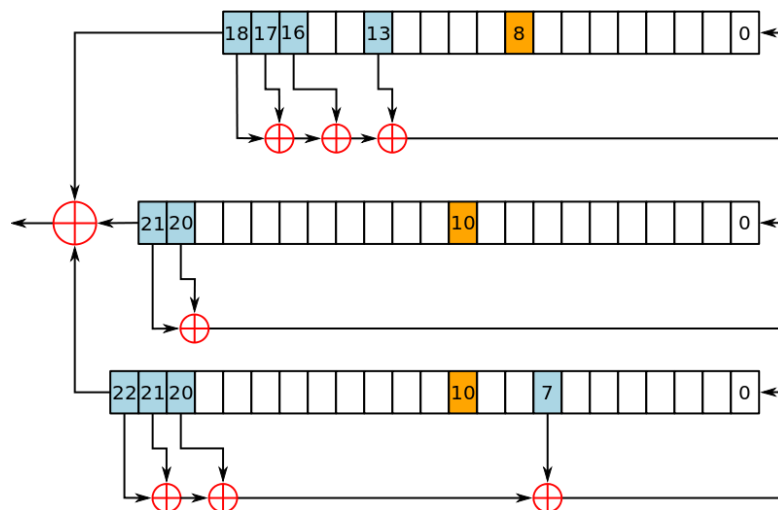


Figura 6.12. Cifrul A5/1

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

6.3.6. Cifrul RC4

Cifrul RC4 este un cifru șir cu cheie de lungime variabilă, dezvoltat în 1987 de către Ron Rivest⁸ 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: șirul cheie este independent de textul clar. Funcționează în baza „cutiilor-S”: S_0, S_1, \dots, S_{255} . Intrările sunt permutări ale numerelor de la 0 la 255, iar permutarea este o funcție de o cheie de lungime variabilă. Există doi indici, i și j , inițializați cu zero. Pentru a genera un octet aleator se procedează astfel:

$$\begin{aligned}
 i &= (i + 1) \bmod 256; j = (j + S_i) \bmod 256; \\
 T &= S_i; S_i = S_j; S_j = T; t = (S_i + S_j) \bmod 256; \\
 K &= S_t.
 \end{aligned}$$

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

Inițializarea “cutiilor-S” este simplă. Se inițializează liniar:

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

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 algoritmi 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.

și un alt vector de 256 de octeți cu cheia, repetând cheia, dacă este necesar, pentru a completa vectorul cu componentele K_0, K_1, \dots, 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 RC4 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 trebuie eliminată. Pentru a consolida RC4 au fost făcute mai multe încercări, cele mai notabile fiind *Spritz*, *RC4A*, *VMPC* și *RC4+*.

În sistemele simetrice de criptare, Alice și Bob își aleg o cheie secretă k care definește regulile de criptare e_k și decriptare d_k . În aproape toate cazurile e_k și 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 și Bob printr-un canal sigur, înainte de transmiterea mesajului criptat. Practic, în condițiile cererii tot mai mari de securizare a comunicațiilor, 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 obținut
 $K_{12} + E(R_{11}) = 011100\ 010101\ 011010\ 110110\ 101001\ 101010\ 011000\ 101101$

Să se determine $S_6(B_6)$.

12. În runda 3 a algoritmului DES în rezultatul aplicării cutiilor-S s-a obținut:
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 obținut

$L_{16} = 0110 0111 0010 0110 0011 0010 0011 0100$,

$R_{16} = 0000 1010 0110 1100 1100 1001 1001 0111$.

Să se determine blocul criptat al mesajului.