Capitolul 5

Sisteme simetrice de criptare moderne

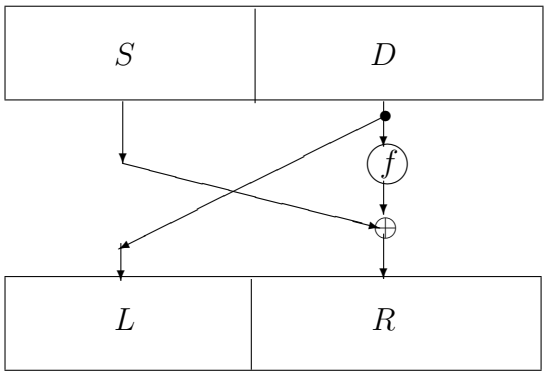
5.1 Sistemul de criptare DES

5.1.1 Reţele Feistel

În cadrul firmei IBM, Horst Feistel descrie în anii ′60 principiile unei structuri pentru sistemele de criptare, cu influenţe majore asupra sistemelor ulterioare. Structura este cunoscută sub numele de *reţea Feistel*. Avantajul major al unei astfel de de structuri este acela că operaţiile de criptare şi decriptare sunt similare, (chiar identice în unele cazuri); aceasta permite o reducere substanţială a mărimii codului (sau circuitelor) care implementează sistemul de criptare.

După cum s-a văzut o cerinţă solicitată de securitatea faţă de atacuri bazate pe analiza textului criptat este aceea ca textul criptat să creeze "difuzie şi confuzie". Problema constă în faptul că, deşi se ştie de mult cum să se construiască funcţii cu caracter aleator, până la reţelele Feistel nu se cunoşteau algoritmi de construcţie pentru funcţii bijective aleatoare.

Soluţia adusă de acesta este foarte elegantă:



Să presupunem că avem o funcţie $f : \{0,1\}^n \longrightarrow \{0,1\}^n$ "aproape aleatoare" (care dă impresia că cei n biţi din imaginea lui f sunt aleşi fără nici o regulă). Algoritmul de criptare va opera cu blocuri de $2n$ biţi, pe care îi împarte în două jumătăţi, sub forma (S, D). Imaginea criptată a unui bloc printr-o reţea Feistel este tot un bloc de $2n$ biţi: (L, R), unde $L = D$ şi $R = S \oplus f(D)$.

Această transformare este bijectivă: din perechea (L, R) se poate regăsi (S, D) prin $D = L,\ S = R \oplus f(L)$.

De remarcat că funcţia f nu este obligatoriu inversabilă. De cele mai multe ori ea se compune din operaţii simple care sunt executate rapid de calculator.

Exemple de astfel de operaţii folosite:

- Permutări de biţi (adesea implementate sub forma unor tabele de permutare, numite $P - box$).
- Funcţii simple neliniare (implementate sub forma unor tabele de substituţie, numite $S - box$).
- Operaţii liniare (deplasări, adunări, XOR)

Toate aceste operaţii pot fi implementate direct pe structuri hardware, ceea ce le face extrem de rapide.

Cum după criptare, jumătatea din dreapta a blocului nu a suferit nici o transformare (doar o deplasare spre stânga), reţeaua Feistel se aplică de mai multe ori – fiecare aplicaţie fiind numită *rundă*.

Reţelele Feistel apar prima oară la sistemul de criptare Lucifer, construit pentru IBM de o echipă condusă de Horst Feistel şi Don Coppersmith. Succesul a fost asigurat odată cu desemnarea sistemului DES ca standard oficial de criptare.

Multe sisteme de criptare moderne sunt bazate pe acest gen de reţele, structura şi proprietăţile lor fiind intens exploatate de comunitatea criptografică.

O listă a celor mai cunoscute sisteme de criptare bazate pe structurile Feistel cuprinde: Blowfish, Camellia, CAST-128, DES, FEAL, KASUMI, LOKI97, Lucifer, MAGENTA, MISTY1, RC5, TEA, Triple DES, Twofish, XTEA. În plus, CAST-256, MacGuffin, RC2, RC6, Skipjack utilizează diverse generalizări ale reţelelor Feistel.

Abia după afirmarea sistemului AES a inceput o perioadă de declin a dominaţiei structurilor de tip Feistel în construirea sistemelor de criptare.

5.1.2 Consideraţii generale privind sistemul de criptare DES

În mai 1973, Biroul Naţional de Standarde din SUA a lansat în Registrul Federal (jurnalul oficial al guvernului) un apel la construirea unui sistem de criptare oficial care să se numească *Data Encryption Standard* (DES). Firma IBM a construit acest sistem – publicat în Registrul Federal la 17 martie 1975, modificând sistemul $Lucifer$, pe care

deja îl testa. După dezbateri publice, DES a fost adoptat oficial la 17 ianuarie 1977 ca standard de criptare. De atunci, el a fost re-evaluat la fiecare 5 ani, fiind în acest moment cel mai popular sistem de criptare cu cheie simetrică. Spargerea sa în iulie 1998 a coincis (întâmplător ?) cu durata sa oficială de utilizare pe teritoriul SUA.

5.1.3 Descrierea sistemului DES

Sistemul DES criptează un bloc de text clar de 64 biţi într-un text criptat tot de 64 biţi, utilizând o cheie de 56 biţi. Algoritmul cuprinde 3 etape:

1. Fie α textul clar iniţial, de 64 biţi. Lui i se aplică o permutare IP iniţială fixată, obţinându-se $\alpha_0 = IP(\alpha) = L_0R_0$. L_0 este format din primii 32 biţi ai lui α_0, iar R_0 – din ultimii 32 biţi.

2. Se efectuează 16 runde (iteraţii) ale unei funcţii care se va preciza. La fiecare rundă se calculează L_iR_i $(1 \leq i \leq 16)$ după regula

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

   unde f este o funcţie care se va preciza, iar $K_1, K_2, \ldots, K_{16}$ sunt secvenţe de 48 biţi calculaţi din cheia K. Se spune că $K_1, K_2, \ldots, K_{16}$ sunt obţinuţi prin *diversificarea cheii* (key shedule).

3. Blocului $R_{16}L_{16}$ i se aplică inversa permutării iniţiale pentru a obţine textul criptat $\beta = IP^{-1}(R_{16}L_{16})$.

Observaţia 5.1. *Sistemul de ecuaţii care definesc criptarea la fiecare rundă poate fi inversat imediat, pentru a obţine ecuaţiile rundelor de decriptare. Acestea sunt:*

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

Funcţia de criptare $f(A, J)$ are ca argumente două secvenţe binare: una de 32 biţi, iar a doua de 48 biţi. Rezultatul este o secvenţă de 32 biţi. Etapele de calcul ale funcţiei sunt:

1. Argumentul A este extins la 48 biţi folosind o *funcţie de expansiune* E. $E(A)$ cuprinde biţii lui A aşezaţi într-o anumită ordine, unii biţii fiind scrişi de două ori.

2. Se calculează $B = E(A) \oplus J$; rezultatul se descompune în 8 subsecvenţe de câte 6 biţi fiecare: $B = B_1B_2B_3B_4B_5B_6B_7B_8$.

3. Se folosesc 8 $S-boxuri$ $S_1, S_2, \ldots, S_8$, fiecare din ele fiind un tablou de dimensiuni 4×16 cu elemente numere întregi din intervalul $[0, 15]$. Pentru o secvenţă $B_j = b_1b_2b_3b_4b_5b_6$ se calculează un şir de 4 biţi $S_j(B_j)$ astfel: biţii b_1b_6 dau reprezentarea binară a indicelui unei linii r $(0 \leq r \leq 3)$ din S_j; ceilalţi patru biţi $b_2b_3b_4b_5$ dau reprezentarea binară a indicelui unei coloane c $(0 \leq c \leq 15)$ din tablou. Atunci $C_j = S_j(B_j) = [S_j(r, c)]_2$ $(1 \leq j \leq 8)$. ($[x]_2$ este reprezentarea în baza 2 a numărului întreg x).

4. Secvenţa $C = C_1C_2C_3C_4C_5C_6C_7C_8$ – de lungime 32 – se rearanjează folosind o permutare fixată P. Rezultatul final este $f(A, J) = P(C)$.

Mai rămâne să specificăm funcţiile particulare folosite de sistemul DES:

- Permutarea iniţială IP este:

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

- Permutarea inversă IP^{-1} este:

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

- Funcţia de expansiune E este definită de tabloul:

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

- Cele opt cutii S ($S - boxuri$) sunt:

S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

- Permutarea fixată P este:

$$\begin{array}{cccc} 16 & 7 & 20 & 21 \\ 29 & 12 & 28 & 17 \\ 1 & 15 & 23 & 26 \\ 5 & 18 & 31 & 10 \\ 2 & 8 & 24 & 14 \\ 32 & 27 & 3 & 9 \\ 19 & 13 & 30 & 6 \\ 22 & 11 & 4 & 25 \end{array}$$

Mai rămâne de prezentat procesul de diversificare al cheii K. De fapt, K este o secvenţă de 64 biţi, din care 56 definesc cheia, iar 8 (biţii de pe poziţiile $8, 16, 24, \ldots, 64$) sunt biţi de paritate, aranjaţi în aşa fel încât fiecare octet să conţină un număr impar de 1. Aceşti 8 biţi sunt ignoraţi în procesul de diversificare.

1. Din cheie se elimină biţii de paritate, iar asupra celorlalţi se aplică o permutare PC_1, obţinându-se $PC_1(K) = C_0D_0$ (C_0 sunt primii 28 biţi din secvenţă, iar D_0 – ceilalţi 28 biţi). Permutarea PC_1 este

$$\begin{array}{ccccccc} 57 & 49 & 41 & 33 & 25 & 17 & 9 \\ 1 & 58 & 50 & 42 & 34 & 26 & 18 \\ 10 & 2 & 59 & 51 & 43 & 35 & 27 \\ 19 & 11 & 3 & 60 & 52 & 44 & 36 \\ 63 & 55 & 47 & 39 & 31 & 23 & 15 \\ 7 & 62 & 54 & 46 & 38 & 30 & 22 \\ 14 & 6 & 61 & 53 & 45 & 37 & 29 \\ 21 & 13 & 5 & 28 & 20 & 12 & 4 \end{array}$$

2. Pentru $i = 1, 2, \ldots, 16$ se calculează

$$C_i = LS_i(C_{i-1})$$
$$D_i = LS_i(D_{i-1})$$

şi $K_i = PC_2(C_iD_i)$.

LS_i este o rotaţie circulară la stânga cu una sau două poziţii, în funcţie de valoarea lui i: o poziţie dacă $i = 1, 2, 9, 16$, altfel rotirea este de două poziţii.

PC_2 elimină biţii $9, 18, 22, 25, 35, 38, 43, 54$ şi rearanjează ceilalţi biţi sub forma:

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Decriptarea se realizează plecând de la textul criptat β şi utilizând acelaşi algoritm, în ordine inversă; se vor folosi în ordine cheile $K_{16}, \ldots, K_1$.

Exemplul 5.1. *([38]) Să considerăm textul clar "Now is time for all" reprezentat ca o secvenţă de caractere ASCII scrise în hexazecimal. Dacă se face o criptare DES folosind cheia* $K = 0123456789ABCDEF$*, se obţine*

$$\begin{aligned} \alpha &= 4E6F772069732074 \quad 68652074696D6520 \quad 666F7220616C6C20 \\ \beta &= 3FA40E8A984D4815 \quad 6A271787AB8883F9 \quad 893D51EC4B563B53 \end{aligned}$$

5.1.4 Chei slabe

În cazul $K_1 = K_{16}$, algoritmul de generare va genera o secvenţă de chei palindromice: $K_i = K_{17-i}$ $(1 \leq i \leq 8)$. Acestea sunt chei slabe.

Definiţia 5.1. *O cheie DES slabă este o cheie* K *cu proprietatea* $e_K(e_K(\alpha)) = \alpha, \forall \alpha \in \mathcal{P}$.
O pereche de chei (K_1, K_2) *este semi-slabă DES dacă* $e_{K_1}(e_{K_2}(\alpha)) = \alpha, \ \ \forall \alpha \in \mathcal{P}$.

În DES sunt 4 chei slabe şi 6 perechi de chei semi-slabe, date de tabelele următoare:

Chei slabe (hex)	C_0	D_0
0101010101010101	0^{28}	0^{28}
$FEFEFEFEFEFEFEFE$	1^{28}	1^{28}
$1F1F1F1F0E0E0E0E$	0^{28}	1^{28}
$E0E0E0E0F1F1F1F1$	1^{28}	0^{28}

C_0	D_0	*Perechi de chei semi − slabe (hex)*		C_0	D_0
$(01)^{14}$	$(01)^{14}$	$01FE01FE01FE01FE,$	$FE01FE01FE01FE01$	$(10)^{14}$	$(10)^{14}$
$(01)^{14}$	$(10)^{14}$	$1FE01FE00EF10EF1,$	$E01FE01FF10EF10E$	$(10)^{14}$	$(01)^{14}$
$(01)^{14}$	0^{28}	$01E001E001F101F1,$	$E001E001F101F101$	$(10)^{14}$	0^{28}
$(01)^{14}$	1^{28}	$1FFE1FFE0EFE0EFE,$	$FE1FFE1FFE0EFE0E$	$(10)^{14}$	1^{28}
0^{28}	$(01)^{14}$	$011F011F010E010E,$	$1F011F010E010E01$	0^{28}	$(10)^{14}$
1^{28}	$(01)^{14}$	$E0FEE0FEF1FEF1FE,$	$FEE0FEE0FEF1FEF1$	1^{28}	$(10)^{14}$

Posibilitatea ca printr-o alegere aleatoare a cheii să se obţină o cheie slabă sau semi-slabă este deci 2^{-52}. Mulţi utilizatori solicită un test pentru depistarea cheilor slabe, test care nu afectează semnificativ timpul de criptare.

De remarcat că fiind dată o cheie semi-slabă K, cealaltă cheie din pereche poate fi obţinută secţionând K în două părţi egale şi rotind fiecare jumătate cu 8 biţi.

Pentru fiecare cheie slabă K există 2^{32} puncte fixe: texte clare α cu proprietatea $e_K(\alpha) = \alpha$.

Pentru fiecare cheie semi-slabă K din partea superioară a tabelului există 2^{32} puncte anti-fixe: texte clare α cu $e_K(\alpha) = \overline{\alpha}$. Aceste patru chei semi-slabe se numesc şi *chei anti - palindromice*, deoarece subcheile generate sunt complementare: $K_1 = \overline{K}_{16}$, $K_2 = \overline{K}_{15}$ etc.

5.1.5 Controverse legate de DES

Încă de la lansarea sa, DES a fost supus la numeroase critici. O primă obiecţie a fost legată de folosirea $S-boxurilor$. Toate calculele din DES sunt liniare, *cu excepţia* acestor $S-boxuri$. Deci, de fapt toată securitatea sistemului se bazează pe acestea. Dar, nimeni (cu excepţia autorilor) nu ştie cum sunt concepute cutiile. Multe persoane au fost convinse că ele ascund diverse trape secrete care permit celor de la *Agenţia Naţională de Securitate* (NSA - serviciul american care răspunde de problemele legate de securitatea naţională) să decripteze orice mesaj.

Ca urmare, NSA afirmă în 1976 că $S-boxurile$ au fost construite pe baza următoarelor criterii:

1. Fiecare linie este o permutare a numerelor $0, \ldots, 15$;

2. Nici un $S-box$ nu este o funcţie liniară sau afină;

3. La modificarea unui bit din operand, un $S-box$ provoacă modificarea cel puţin a doi biţi din rezultat;

4. Pentru fiecare cutie S şi α (secvenţă de lungime 6), $S(\alpha)$ şi $S(\alpha \oplus 001100)$ diferă prin cel puţin doi biţi.

Alte două proprietăţi au fost menţionate ca fiind "consecinţe ale criteriilor de construcţie":

5. Pentru orice cutie S şi orice α, $S(\alpha) \neq S(\alpha \oplus 11ab00)$, oricare ar fi $a, b \in \{0, 1\}$;

6. Pentru orice cutie S, dacă un bit din operand este menţinut constant şi se urmăreşte un bit al rezultatului, numărul de valori care produc 0 este "apropiat" de numărul de valori care produc 1. Într-adevăr, dacă bitul fixat este unul din cei doi biţi care determină linia cutiei S, există – conform criteriului 1. – 16 valori care produc 0 şi 16 valori care produc 1; pentru ceilalţi biţi, aceasta nu este adevărat, dar numărul de valori care produc 0 (sau 1) este cuprins totdeauna între 13 şi 19.

Nici un alt criteriu referitor la $S-boxuri$ nu a mai fost recunoscut public.

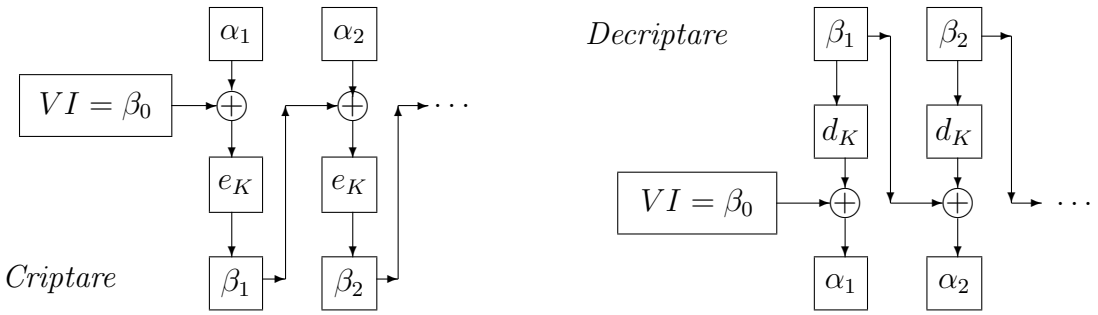
Cea mai pertinentă critică referitoare la DES se referă la mărimea prea mică (numai 2^{56}) a spaţiului cheilor. Ca urmare s-au conceput numeroase maşini dedicate atacurilor prin forţă brută, care să caute cheia. Fiind dat un text clar α de 64 biţi şi textul său criptat β, se verifică toate cheile K posibile (aproximativ 2^{55}), până se obţine $e_K(\alpha) = \beta$ (de remarcat că soluţia există totdeauna, dar nu este unică).

5.1.6 Moduri de implementare ale DES -ului

În istoria sa, sistemul de criptare DES a cunoscut patru moduri de implementare, moduri extinse ulterior la toate sistemele de criptare:

- Modul ECB (*Electronic Codebook Mode*) corespunde metodei descrise anterior: fiind dat un text clar $x = \alpha_1\alpha_2\ldots$, fiecare bloc α_i de 64 biţi este criptat cu cheia K după formula
$$\beta_i = e_K(\alpha_i), \quad (i \geq 1)$$
procedeul conducând la textul criptat $y = \beta_1\beta_2\ldots$

- În modul CBC (*Cypher Block Chaining Mode*), fiecare bloc de text criptat β_i este combinat printr-un XOR cu textul clar următor α_{i+1}, înainte de criptarea acestuia. Operaţia decurge conform schemei:



Se defineşte un bloc cu valoarea iniţială $VI = \beta_0$, după care blocurile se criptează după formula
$$\beta_i = e_K(\beta_{i-1} \oplus \alpha_i) \quad (i \geq 1)$$

Funcţia de decriptare
$$\alpha_i = \beta_{i-1} \oplus d_K(\beta_i) \quad (i \geq 1)$$

nu mai este recursivă. De aceea, decriptarea se poate efectua mai rapid (comparativ cu operaţia de criptare), printr-un proces de calcul paralel.

- Modurile OFB şi CFB sunt construite conform sistemelor de criptare cu chei fluide: se generează cheia fluidă, care se combină apoi cu textul clar (similar sistemului *one-time-pad*). OFB (*Output Feedback Mode*) este o criptare sincronizabilă aditivă: componentele cheii fluide sunt obţinute prin criptarea iterativă a unui bloc iniţial VI de 64 biţi; se defineşte $\gamma_0 = VI$ şi se calculează $\gamma_1, \gamma_2, \ldots$ după formula

$$\gamma_i = e_K(\gamma_{i-1}), \quad (i \geq 1)$$

Secvenţa blocurilor de text clar $x = \alpha_1, \alpha_2, \ldots$ este criptată apoi conform relaţiei

$$\beta_i = \alpha_i \oplus \gamma_i, \quad (i \geq 1)$$

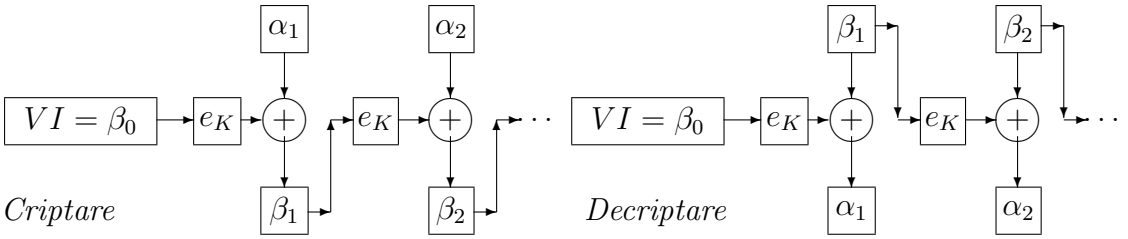
- În modul CFB (*Cypher Feedback Mode*) se începe cu $\beta_0 = VI$ (bloc iniţial de 64 biţi) şi se calculează cheia fluidă γ_i criptând din nou blocul de text criptat obţinut anterior:

$$\gamma_i = e_K(\beta_{i-1}) \quad (i \geq 1)$$

Ca şi la modul OFB, în faza a doua avem

$$\beta_i = \alpha_i \oplus \gamma_i \quad (i \geq 1)$$

De remarcat că funcţia de criptare e_K este folosită aici atât la procesul de criptare cât şi la cel de decriptare.



Şi în modul de implementare CFB este posibilă o decriptare în paralel, datorită formulei

$$\alpha_i = \beta_i \oplus e_K(\beta_{i-1}) \quad (i \geq 1)$$

Deşi descrierile au fost gândite pentru blocuri de mărime 64, modurile OFB şi CFB pot fi extinse imediat la blocuri de k biţi ($1 \leq k \leq 64$).

Cele patru moduri de implementare prezintă diverse avantaje şi dezavantaje. Astfel, la ECB şi OFB, modificarea unui bloc de text clar α_i provoacă modificarea unui singur bloc de text criptat, β_i. În anumite situaţii, acest fapt constituie un defect. Modul OFB este utilizat adesea pentru transmisiile prin satelit.

În modurile CBC şi CFB dimpotrivă, modificarea unui bloc α_i de text clar antrenează modificări în toate blocurile de texte criptate, începând cu β_i. De aceea, aceste moduri sunt adaptate în particular problemelor de autentificare a mesajelor (MAC - Message Authentication Code). Un MAC este adăugat la un text clar cu scopul de a-l convinge pe *Bob* că textul primit a fost scris de *Alice* şi nu a fost alterat de *Oscar*. El garantează astfel integritatea (sau autenticitatea) mesajului, dar nu şi confidenţialitatea sa.

Să descriem cum este utilizat modul CBC la construcţia unui MAC. Se pleacă de la blocul iniţial VI în care toţi biţii sunt 0. Se construieşte textul criptat $\beta_1, \beta_2, \ldots, \beta_n$ cu cheia K, în modul CBC, iar MAC este blocul β_n. *Alice* va transmite mesajul $\alpha_1, \alpha_2, \ldots, \alpha_n$, asociat cu MAC-ul β_n. Când *Bob* primeşte mesajul $\alpha_1, \alpha_2, \ldots, \alpha_n$, el generează $\beta_1, \ldots, \beta_n$ folosind cheia (secretă) K şi verifică dacă β_n este identic cu mesajul MAC primit.

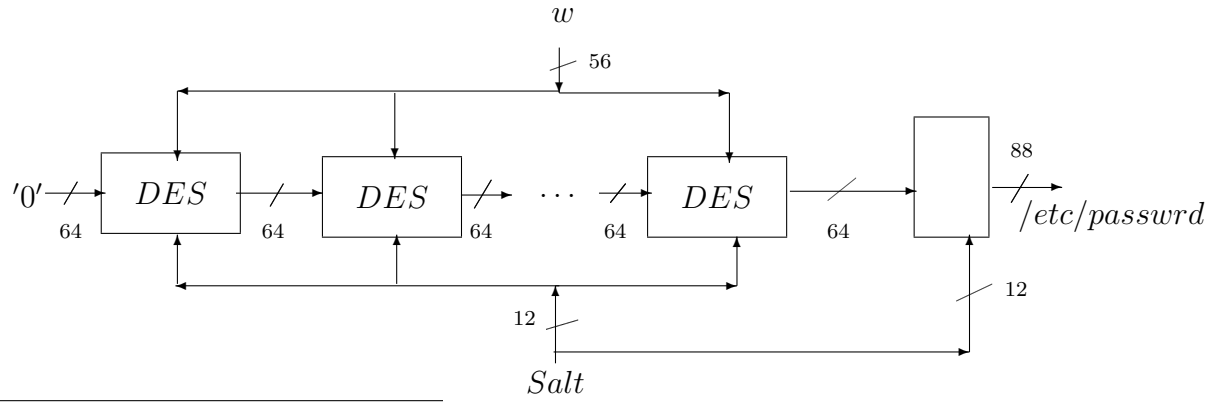
De remarcat că *Oscar* nu poate construi un MAC deoarece nu cunoaşte cheia K utilizată de *Alice* şi *Bob*; orice modificare a mesajelor clare este depistată astfel uşor.

Se poate realiza şi o combinare a integrităţii cu confidenţialitatea, în felul următor: *Alice* utilizează cheia K_1 pentru a calcula un MAC bazat pe $\alpha_1, \ldots, \alpha_n$; fie α_{n+1} acest MAC. Apoi, ea criptează mesajul $\alpha_1, \ldots, \alpha_{n+1}$ în $\beta_1, \ldots, \beta_{n+1}$ folosind o a doua cheie K_2. Când *Bob* primeşte mesajul, el decriptează în prima fază (cu cheia K_2), apoi verifică cu cheia K_1 dacă α_{n+1} este MAC-ul lui $\alpha_1, \ldots, \alpha_n$.

Sau – ca altă variantă – *Alice* poate utiliza K_1 pentru criptarea mesajului $\alpha_1, \ldots, \alpha_n$; apoi, pentru $\beta_1, \ldots, \beta_n$ determină MAC-ul β_{n+1} folosind cheia K_2. *Bob* va face întâi verificarea corectitudinii dată de MAC şi – dacă totul este în ordine – va trece la decriptare.

5.1.7 Parole $UNIX$

O aplicaţie interesantă a sistemului de criptare DES este algoritmul de criptare al parolelor din sistemul $UNIX$[1], algoritm numit *UNIX crypt*.



[1]Sistem dezvoltat de compania Bell Laboratories.

Versiunea criptată a unei parole este stocată într-o bază de date */etc/passwrd* (a cărei confidenţialitate nu este obligatoriu asigurată). Atunci când un utilizator dă parola, sistemul o criptează şi o compară cu varianta din baza de date. Modul de criptare este format din următoarele etape:

1. Parola este trunchiată la primele opt caractere $ASCII$ (sau completată cu 0 dacă parola are mai puţin de 8 caractere). Fiecare caracter dă 7 biţi, formând o cheie DES de 56 biţi.

2. Se realizează o criptare în serie prin 25 sisteme DES, textul clar iniţial fiind $00\ldots0$.

3. Cei 64 biţi obţinuţi după ultima criptare se combină cu 12 biţi daţi de *Salt* şi cu un bit suplimentar, formând 11 caractere printabile de câte 7 biţi. Biţii de paritate completează parola criptată până la 88 biţi.

Sistemul $UNIX$ oferă pentru criptare un pachet de 12 biţi – numit *Salt* – generat de sistemul de ceas al sistemului la momentul creerii parolei şi stocat în baza de date, în cadrul parolei criptate. Aceşti biţi sunt folosiţi pentru modificarea funcţiei de expansiune E din DES, oferind $2^{12} = 4096$ variante. Anume: fiecare bit din *Salt* este asociat cu o pereche fixată din blocul de 48 biţi din E. Dacă el este 1, atunci biţii din E asociaţi lui sunt interschimbaţi; altfel, sunt lăsaţi pe loc.

Această modificare previne utilizarea de implementări ale sistemului DES externe sistemului $UNIX$.

5.1.8 Sisteme de criptare înrudite cu DES

Triplu DES ($3DES$)

Triplu DES (cunoscut şi sub numele $3DES$ sau – mai rar – $DES - ede$) este un sistem derivat din DES, propus de Walter Tuchman (şeful echipei IBM care a construit DES). Numele oficial este $FIPS\ Pub\ 46-3$.

Formal, $3DES$ este definit prin formula

$$c = DES_{k_3}(DES_{k_2}^{-1}(DES_{k_1}(m))),$$

unde:

m este un bloc de text clar (64 biţi),
c este blocul de text criptat rezultat,
k_1, k_2, k_3 sunt chei DES (de 56 biţi),
DES_k: criptarea DES cu cheia k,
DES_k^{-1}: decriptarea DES cu cheia k.

Introducerea la pasul 2 a decriptării nu afectează securitatea algoritmului. Avantajul constă în utilizarea pentru $3DES$ a unei implementări DES (astfel, cele două sisteme pot inter-opera).

Uneori – dar destul de rar – se foloseşte pentru *Triplu DES* o criptare în lanţ de trei criptări DES (numită şi $DES - eee$):

$$c = DES_{k_3}(DES_{k_2}(DES_{k_1}(m))).$$

Criptarea în trei paşi (*ede* sau *eee*) este esenţială pentru protejarea contra unui atac de tipul *meet-in-the-middle*. În cazul unei duble criptări, acest atac este destul de eficient.

Într-adevăr, să considerăm un sistem de criptare bloc unde mărimea cheii este n. O criptare dublă cu două chei diferite va folosi de fapt o cheie de lungime $2n$. Pentru un text clar dat m, să presupunem că putem stoca $e_K(m)$, pentru toate cheile K posibile.

Fie x un text criptat după formula $x = e_{k_2}(e_{k_1}(m))$, unde cheile k_1 şi k_2 sunt secrete. Pentru fiecare cheie p există o cheie unică q astfel ca $d_p(x) = e_q(m)$. În particular există exact 2^n chei posibile determinate de perechea (m, x), chei care pot fi găsite în aproximativ $\mathcal{O}(2^n)$ paşi.

Dacă numărul cheilor care pot fi stocate este de ordinul $2^p < 2^n$, algoritmul poate fi modificat pentru a afla toate cheile posibile, în circa $\mathcal{O}(2^{2n-p})$ paşi.

Observaţia 5.2. *Pentru oricare din situaţiile*

$$k_1 = k_2, \quad k_2 = k_3, \quad k_1 = k_2 = k_3,$$

$DES - ede$ se reduce la un simplu DES, lucru utilizat frecvent pentru a verifica compatibilitatea sistemului.

Cheia pentru *Triplu DES* are $3 * 56 = 168$ biţi, la care se adaugă $3 * 8 = 24$ biţi de paritate; în total sunt 192 biţi.

O variantă, numită "$3DES$ cu două chei" foloseşte $k_1 = k_3$; aici mărimea de stocare a cheii va fi de numai 128 biţi (din care 16 biţi de paritate). Acest mod este însă sensibil la anumite atacuri cu text clar ales, propuse de Merkle şi Hellman (1981) şi – mai târziu – de Van Oorschot şi Wiener (1991).

Sistemul de criptare $3DES$ nu este încă spart, dar utilizarea sa este îngreunată din cauza vitezei mici de criptare. Totuşi multe sisteme continuă să folosească $3DES$; ca exemple mai recente sunt cardurile bancare (*Mastercard* şi parţial – *Visa*) care din 2002 sunt configurate pe baza acestui sistem de criptare. De asemenea, sistemul de telefonie mobilă *Zapp* foloseşte $3DES$ ca sistem de criptare.

$DES - X$

$DES - X$ (sau – mai simplu – $DESX$) este o variantă a sistemului de criptare DES, dezvoltată pentru a rezista mai bine unui atac prin forţă brută.

După cum am văzut, sistemul DES operează cu o cheie de 56 biţi; deci sunt numai 2^{56} chei posibile, unele din acestea fiind chei slabe. Pentru a evita un atac direct, in 1984 Rivest propune creşterea mărimii cheii K fără a modifica substanţial algoritmul.

$DES-X$ foloseşte două chei suplimentare K_1, K_2 de câte 64 biţi şi efectuează criptarea unui bloc de text clar x după formula

$$DESX_{K,K_1,K_2}(x) = K_2 \oplus DES_K(x \oplus K_1)$$

Mărimea cheii creşte deci la $56 + 2 * 64 = 184$ biţi.

IDEA

Prima apariţie a sistemului $IDEA$ are loc în 1990, sub denumirea PES (*Proposed Encryption Standard*). Deoarece în acelaşi an, Biham şi Shamir publică sistemul de atac al DES-ului prin criptanaliză diferenţială, în 1991 autorii (Xuejia Lai şi James Massey) modifică sistemul de criptare pentru a rezista acestui gen de atac. Noul sistem este numit $IPES$ (*Improved Proposed Encryption Standard*), nume care în 1992 se schimbă în $IDEA$ (*International Data Encryption Standard Algorithm*).

În opinia lui Bruce Schneider, în 1996 $IDEA$ constituia cel mai sigur sistem de criptare utilizabil fără restricţii. Este o componentă a sistemului de securitate prin poşta electronică PGP, unde asigură criptarea datelor.

Sistemul este foarte asemănător cu DES. Astfel, el este un sistem simetric care operează cu blocuri de texte clare de 64 biţi, folosind o cheie de 128 biţi.

Textul clar de 64 biţi este despărţit în 4 sublocuri X_1, X_2, X_3, X_4, de câte 16 biţi fiecare. Aceste patru componente formează intrarea la prima rundă a algoritmului. În total sunt 8 runde, care folosesc operaţiile de adunare, înmulţire (modulo 2^{16}) şi XOR; toate pe 16 biţi. De asemenea sunt implicate şi 16 subchei a câte 16 biţi fiecare. Între runde, subblocurile 2 şi 3 sunt schimbate între ele. După runda 8, cele patru subblocuri sunt combinate cu 4 subchei, formând ieşirea.

O rundă conţine 14 operaţii, anume:

1. Se înmulţeşte X_1 cu prima subcheie;
2. Se adună X_2 cu a doua subcheie;
3. Se adună X_3 cu a treia subcheie;
4. Se înmulţeşte X_4 cu a patra subcheie;
5. Ce s-a obţinut la paşii 1 şi 3 se combină prin XOR;
6. Ce s-a obţinut la paşii 2 şi 4 se combină prin XOR;
7. Se înmulţeşte rezultatul pasului 5 cu subcheia 5;
8. Se adună rezultatele paşilor 6 şi 7;
9. Se înmulţeşte rezultatul pasului 8 cu subcheia 6;
10. Se adună rezultatele paşilor 7 şi 9;
11. Ce s-a obţinut la paşii 1 şi 9 se combină prin XOR;
12. Ce s-a obţinut la paşii 3 şi 9 se combină prin XOR;
13. Ce s-a obţinut la paşii 2 şi 10 se combină prin XOR;
14. Ce s-a obţinut la paşii 4 şi 10 se combină prin XOR.

Rezultatele paşilor $11 - 14$ formează ieşirea dintr-o rundă; cele patru subblocuri (după ce subblocurile 2 şi 3 sunt interschimbate) formează intrarea în runda următoare.

După ultima rundă, are loc o transformare finală:

1. Se înmulţeşte X_1 cu prima subcheie;
2. Se adună X_2 cu a doua subcheie;
3. Se adună X_3 cu a treia subcheie;
4. Se înmulţeşte X_4 cu a patra subcheie.

Cela patru subblocuri obţinute în final formează textul criptat.

Prelucrarea subcheilor se face după următorul algoritm:

1. Cheia de 128 biţi este desfăcută în 8 subchei a câte 16 biţi fiecare;
2. Primele 6 subchei sunt folosite la prima rundă, iar următoarele două, la runda a doua;
3. Cheia este rotită spre stânga cu 25 biţi şi se desface din nou în 8 subchei, folosite la rundele 2 şi 3 (câte patru subchei);
4. Se repetă pasul 3 cât timp este necesar.

Operaţia de decriptare se realizează urmând aceiaşi paşi, cu singura diferenţă că subcheile se introduc în ordine inversă.

Comparativ cu DES, $IDEA$ prezintă unele avantaje. Astfel, implementările realizate permit o viteză dublă de criptare în $IDEA$ faţă de DES.

Lungimea cheii (128) biţi, asigură o securitate sporită la atacurile prin forţă brută: pentru a găsi cheia prin încercări, ar fi necesare cam 2^{127} teste; deci cu un chip care testează un miliard de chei pe secundă, ar trebui cam 10^{13} ani – o perioadă mai lungă decât vârsta Pământului.

Autorii au afirmat (fără să demonstreze) că $IDEA$ rezistă atacurilor prin criptanaliza diferenţială. Un studiu al subcheilor a arătat că există unele chei slabe, care permit spargerea sistemului de criptare într-un timp mai scurt. O asigurare contra alegerii (aleatoare) a acestor chei slabe ar fi completarea algoritmului de generare a subcheilor prin operarea fiecărei subchei generate printr-un XOR cu $0x0D$, unde "x" este o cifră hexazecimală aleasă aleator.

5.2 Alte sisteme de criptare ulterioare DES

La sfârşitul anilor $'90$ se decide înlocuirea sistemului de criptare DES. Motivele sunt multiple, dar menţionăm numai două:

- În iulie 1998 sistemul DES pe 56 biţi este spart de către organizaţia *Electronic Frontier Foundation*; s-a folosit un calculator construit special în acest scop, iar timpul necesar spargerii a fost de 3 zile.

- În luna septembrie a aceluiaşi an, administraţia americană acordă companiilor producătoare de soft de securitate permisiunea de a exporta implementări ale algoritmului DES bazate pe chei de criptare de 56 biţi.

În legătură cu aceste evenimente, pe 20 august 1998 $NIST$ (National Institute of Standards and Technology) anunţă (în cadrul unei conferinţe speciale) un set de 15 algoritmi candidaţi să înlocuiască DES[2]. Este ales şi numele noului sistem de criptare: AES (**A**dvanced **E**ncryption **S**tandard). Cei 15 algoritmi au fost trimişi de membri din comunitatea criptografică mondială. Criteriile stabilite de $NIST$ pentru noul sistem au fost:

- Să fie un sistem de criptare simetric pe blocuri de 128 biţi.
- Să accepte chei de lungime 128, 192 şi 256 biţi;
- Să nu aibă chei slabe;
- Să fie eficient atât pe platforme Intel Pentium Pro cât şi pe alte platforme software sau hardware;
- Să poată fi implementat atât pe procesoare de 32 biţi cât şi pe smart - carduri (procesoare de 8 biţi);
- Să fie cât mai simplu.
- Să fie mai rapid decât DES şi să ofere o securitate mai mare decât $3DES$.

A doua conferinţă AES are loc în martie 1999; după analiza rezultatelor algoritmilor propuşi, $NIST$ selectează 5 algoritmi: $Mars, RC6, Rijndael, Serpent$ şi $Twofish$. Aceştia sunt supuşi testelor şi discuţiilor publice, folosind drept criterii de evaluare: securitate, cost, implementare.

În mai 2000 $NIST$ anunţă drept sistem "câştigător" sistemul de criptare $Rijndael$, care devine oficial AES.

Înainte de a prezenta sistemul AES vom face o scurtă trecere în revistă a sistemelor de criptare finaliste în această competiţie.

5.2.1 Mars

Sistemul $MARS$ este propus de firma IBM, autorii săi fiind un grup condus de Don Coppersmith. Câteva detalii de construcţie:

- Algoritmul este format din trei componente, fiecare din ele asigurând protecţie pentru anumite tipuri de atac.

[2]Aceştia sunt (în ordine alfabetică) $CAST-256, CRYPTON, DEAL, DFC, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, Rijndael, SAFER+, Serpent, Twofish$.

- Lucrează pe procesoare de 32 biţi, putând fi implementat atât în format *big-endian*, cât şi în *little endian*[3].
- La intrare textul clar este "spart" în grupuri de 128 biţi (4 cuvinte de câte 32 biţi). Prima şi a treia parte a algoritmului amestecă aceşti biţi folosind o cheie de dimensiune variabilă (între 128 şi 448 biţi) şi un $S-box$ de 512 elemente. Componenta din mijloc este formată din 16 runde şi foloseşte în plus o o funcţie de expandare E.
- Operaţiile folosite sunt: adunări, scăderi, XOR-uri, rotaţii (fixe sau dependente de date) şi înmulţiri; toate calculele se fac modulo 32. Construcţia $S-boxului$ s-a făcut pe baza unui algoritm public, plecând de la o serie de condiţii iniţiale clare.

Notaţiile folosite în algoritm:

- $D[0], D[1], D[2], D[3]$ – cuvinte (de 32 biţi), iniţializate cu textul clar; în final aici va fi blocul de text criptat; se notează $D[i:j]$ octetul j din $D[i]$ $(j = 0, 1, 2, 3)$.
- $K[\cdot]$ – cheia expandată; secvenţă de 40 cuvinte;
- $S[\cdot]$ – $S-box$ de 512 cuvinte; $S0[]$ va conţine primele 256 elemente, iar $S1[]$ – ultimele 256 cuvinte.

Faza I (Mixare preliminară):
1. **for** $i \longleftarrow 0$ **to** 3 **do** $D[i] \longleftarrow D[i] + K[i]$;
2. **for** $i \longleftarrow 0$ **to** 7 **do**
   - 2.1. $D[1] \longleftarrow D[1] \oplus S0[D[0,0]]$;
   - 2.2. $D[1] \longleftarrow D[1] + S1[D[0,1]]$;
   - 2.3. $D[2] \longleftarrow D[2] + S0[D[0,2]]$;
   - 2.4. $D[3] \longleftarrow D[3] \oplus S1[D[0,3]]$;
   - 2.5. $D[0] \longleftarrow D[0] >>> 24$;
   - 2.6. **if** $i \in \{0,4\}$ **then** $D[0] \longleftarrow D[0] + D[3]$;
   - 2.6. **if** $i \in \{1,5\}$ **then** $D[0] \longleftarrow D[0] + D[1]$;
   - 2.7. $(D[0], D[1], D[2], D[3]) \longrightarrow (D[3], D[0], D[1], D[2])$.

[3]Fie $a_1a_2a_3a_4$ un cuvânt pe 4 octeţi, unde a_i este un număr întreg din intervalul $[0, 255]$.

1. În arhitectura *big-endian* (o staţie SPARK de exemplu), un cuvânt reprezintă întregul $a_1 2^{24} + a_2 2^{16} + a_3 2^8 + a_4$.
2. În arhitctura *little-endian* (cum este familia *Intel* $80xxx$) un cuvânt reprezintă întregul $a_4 2^{24} + a_3 2^{16} + a_2 2^8 + a_1$.

Faza II (Transformări bazate pe cheie):

1. **for** $i \longleftarrow 0$ **to** 15 **do**
   - 1.1. $(o1, o2, o3) \longleftarrow E(D[0], K[2i+4], K[2i+5])$;
   - 1.2. $D[0] <<< 13$;
   - 1.3. $D[2] \longleftarrow D[2] + o2$;
   - 1.4. **if** $i < 8$ **then**
     - 1.4.1. $D[1] \longleftarrow D[1] + o1$;
     - 1.4.2. $D[3] \longleftarrow D[3] \oplus o3$;

     else
     - 1.4.3. $D[3] \longleftarrow D[3] + o1$;
     - 1.4.4. $D[1] \longleftarrow D[1] \oplus o3$;
   - 1..5. $(D[0], D[1], D[2], D[3]) \longleftarrow (D[3], D[0], D[1], D[2])$.

Faza III (Mixare finală):

1. **for** $i \longleftarrow 0$ **to** 7 **do**
   - 1.1. **if** $i \in \{2, 6\}$ **then** $D[0] \longleftarrow D[0] - D[3]$;
   - 1.2. **if** $i \in \{3, 7\}$ **then** $D[0] \longleftarrow D[0] - D[1]$;
   - 1.3. $D[1] \longleftarrow D[1] \oplus S1[D[0, 0]]$;
   - 1.4. $D[2] \longleftarrow D[2] - S0[D[0, 3]]$;
   - 1.5. $D[3] \longleftarrow D[3] - S1[D[0, 2]]$;
   - 1.6. $D[3] \longleftarrow D[3] \oplus S0[D[0, 1]]$;
   - 2.5. $D[0] \longleftarrow D[0] <<< 24$;
   - 2.7. $(D[0], D[1], D[2], D[3]) \longrightarrow (D[3], D[0], D[1], D[2])$.
2. **for** $i \longleftarrow 0$ **to** 3 **do** $D[i] \longleftarrow D[i] - K[36 + i]$;
3. $Output(D[0], D[1], D[2], D[3])$.

Mai trebuie precizate funcţia E şi modalitatea de expandare a cheii K.

$E(inp, cheie1, cheie2)$;

1. $M \longleftarrow inp + cheie1$; ($M$ este variabilă auxiliară)
2. $R \longleftarrow (inp <<< 13) \cdot cheie2$; ($R$ este variabilă auxiliară)
3. $i \longleftarrow$ cei mai mici nouă biţi din M;
4. $L \longleftarrow S[i]$; (L este variabilă auxiliară)
5. $R \longleftarrow (R <<< 5)$;
6. $r \longleftarrow$ cei mai mici cinci biţi din R;
7. $M \longleftarrow (M <<< r)$;
8. $L \longleftarrow L \oplus R$;
9. $R \longleftarrow (R <<< 5)$;
10. $L \longleftarrow L \oplus R$;
11. $r \longleftarrow$ cei mai mici cinci biţi din R;
12. $L \longleftarrow (L <<< r)$;
13. $output(L, M, R)$.

Expandarea cheii:

Această procedură expandează o cheie $k[0,\ldots,n-1]$ de n cuvinte $(4 \leq n \leq 14)$ într-o cheie K de 40 cuvinte, folosind un vector temporar T de 15 cuvinte şi un vector fixat Z de 4 cuvinte.

1. $Z \longleftarrow A4A8D57B\ \ 5B5D193B\ \ C8A8309B\ \ 73F9A978$;
2. $T[0,\ldots,n-1] \longleftarrow k[0,\ldots,n-1], \quad T[n] \longleftarrow n, \quad T[n+1,\ldots,14] \longleftarrow 0$;
3. **for** $j \longleftarrow 0$ **to** 3 **do**
   3.1. **for** $i \longleftarrow 0$ **to** 14 **do**
   $T[i] \longleftarrow T[i] \oplus ((T[(i-7)\ mod\ 15] \oplus T[(i-2)\ mod\ 15] <<< 3) \oplus (4 \cdot i + j)$;
   3.2. **for** $m \longleftarrow 1$ **to** 4 **to**
   for $i \longleftarrow 0$ **to** 14 **do**
   $T[i] \longleftarrow (T[i] + S[$cei mai mici nouă biţi din$T[(i-10)\ mod\ 15]]) <<< 9$;
   3.3. **for** $i \longleftarrow 0$ **to** 9 **do** $K[10 \cdot j + i] \longleftarrow T[4 \cdot i\ mod\ 15]$;
4. **for** $i \longleftarrow 5, 7, \ldots, 33, 35$ **do**
   4.1. $j \longleftarrow$ cei mai din dreapta doi biţi din $K[i]$;
   4.2. $w \longleftarrow K[i]$ cu cei mai din dreapta doi biţi setaţi pe 1;
   4.3. **for** $p \longleftarrow 2$ **to** 30 **do**
   if $(w[p-1] = w[p] = w[p+1]) \vee (w[p]$ e într-o secvenţă de 10 biţi consecutivi egali)
   then $M[p] \longleftarrow 1$;
   4.4. $r \longleftarrow$ cei mai din dreapta cinci biţi din $K[i-1]$;
   4.5. $X \longleftarrow (Z[j] <<< r)$;
   4.6. $K[i] \longleftarrow w \oplus (X \wedge M)$.
5. $output(K[\cdot])$.

Cele 512 cuvinte ale $S-boxului$ au fost generate în aşa fel încât să verifice condiţiile:
- Nu are valorile 00000000 şi $FFFFFFFF$;
- Orice două valori diferă prin cel puţin 4 biţi;
- Nu există două valori a, b astfel ca $a = \overline{b}$ sau $a = -b$;
- Diferenţele de scădere sau XOR între orice două valori sunt maxime.

Pentru alte detalii privind securitatea sistemului $MARS$ (algoritmul de decriptare, moduri de implementare, componentele $S-boxului$ etc), a se vedea [14].

5.2.2 RC6

Sistemul de criptare $RC6$ este o versiune a lui $RC5$, elaborată de laboratoarele RSA (sub coordonarea lui Ron Rivest) cu scopul de a îndeplini criteriile AES. Este un sistem simplu, fără atacuri practice cunoscute (sunt elaborate doar câteva atacuri teoretice), complet parametrizat: versiunea generală este $RC6-w/r/b$ unde mărimea cuvintelor este w, numărul de runde este r, iar b este mărimea (în octeţi) a cheii de criptare. Versiunea pentru AES are $w = 32, r = 20$.

Algoritmul lucrează pe blocuri de lungime w folosind şase operaţii:

- $+, -, \cdot$ (toate modulo 2^w),
- $\oplus$,
- $a <<< b$ (rotirea lui a spre stânga cu un număr de poziţii dat cei mai puţin semnificativi $log_2 w$ biţi din b),
- $a >>> b$ (operaţie similară spre dreapta).

Algoritmul de criptare:

Intrare:
- Textul clar stocat în cuvintele A, B, C, D;
- Numărul r de runde;
- Cheia $S[0, \ldots 2r + 3]$;

Ieşire: Textul criptat stocat în A, B, C, D.
Algoritm:
1. $B \longleftarrow B + S[0]$;
2. $D \longleftarrow D + S[1]$;
3. **for** $i \longleftarrow 1$ **to** r **do**
   - 3.1. $t \longleftarrow (B \cdot (2 \cdot B + 1)) <<< log_2 w$;
   - 3.2. $u \longleftarrow (D \cdot (2 \cdot D + 1)) <<< log_2 w$;
   - 3.3 $A \longleftarrow ((A \oplus t) <<< u) + S[2i]$;
   - 3.3 $C \longleftarrow ((C \oplus u) <<< t) + S[2i + 1]$;
   - 3.4. $(A, B, C, D) \longleftarrow (B, C, D, A)$;
4. $A \longleftarrow A + S[2r + 2]$;
5. $C \longleftarrow C + S[2r + 3]$.

Detalii privind securitatea şi modalităţi de implementare pot fi găsite în [40].

Algoritmul de diversificare a cheii S este identic cu cel de la $RC5$. În [45] acesta este prezentat împreună cu un cod pentru $RC5$, scris în $C++$.

Cheia iniţială are b octeţi ($0 \leq b \leq 255$) şi este stocată într-un tablou L cu $c = \lceil b/4 \rceil$ cuvinte (completând ultimul cuvânt cu 0 dacă este necesar).

În prima fază tabloul S este iniţializat folosind un generator liniar congruenţial[4]:

1. $S[0] \longleftarrow P$;
2. **for** $i \longleftarrow 1$ **to** $2(r + 1) - 1$ **do** $S[i] \longleftarrow (S[i - 1] + Q)\ mod\ 2^{32}$

$P = B7E15163$ şi $Q = 9E3779B9$ sunt constante bazate pe reprezentarea binară a lui e şi π.

În etapa a doua se efectuează un amestec între L şi S:

[4] A se vedea Capitolul 12 dedicat generatorilor de numere pseudoaleatoare.

3. $i \longleftarrow 0,\ j \longleftarrow 0;$
4. $A \longleftarrow 0,\ B \longleftarrow 0;$
5. **for** $s \longleftarrow 1$ **to** $3 \cdot max\{c, 2(r+1)\}$ **do**
   - 5.1. $S[i] \longleftarrow (S[i] + A + B) <<< 3, \quad A \longleftarrow S[i];$
   - 5.2. $L[i] \longleftarrow (L[i] + A + B) <<< (A + B), \quad B \longleftarrow L[i];$
   - 5.3. $i \longleftarrow (i+1)\ (mod\ 2(r+1));$
   - 5.4. $j \longleftarrow (j+1)\ mod\ c$

5.2.3 Serpent

Sistemul de criptare *Serpent* a fost propus de Ross Anderson (Universitatea Cambridge), Eli Biham (Institutul Tehnion, Haifa) şi Lars Knudsen (Universitea Bergen). El criptează texte clare de 128 biţi în blocuri de aceeaşi lungime, în 32 runde, folosind 33 chei de criptare de rundă, de lungime 128 biţi [5]. Ea se extinde la 256 biţi punând în faţă 1 pentru primul bit, apoi 0 pentru ceilalţi biţi – atât cât este necesar.

Algoritmul de criptare cuprinde 3 paşi:

Intrare: P (textul clar)
Ieşire: C (Textul criptat)
Algoritm:

1. $\hat{B}_0 \longleftarrow IP(P) \quad (IP$ - permutare iniţială);
2. **for** $i \longleftarrow 0$ **to** 31 **do** $\hat{B}_{i+1} \longleftarrow R_i(\hat{B}_i);$
   where $R_i(X) \longleftarrow L(\hat{S}_i(X_i \oplus \hat{K}_i)), \quad 0 \le i \le 30,$
   $R_{31}(X) \longleftarrow \hat{S}_7(X_{31} \oplus \hat{K}_{31}) \oplus K_{32}.$
   Se aplică 32 runde formate din: un XOR cu cheia de rundă K_i, o trecere printr-un $S-box$ şi o transformare liniară L; în ultima rundă, transformarea liniară este înlocuită cu încă o combinare cu o a doua cheie de rundă;
   $\hat{S}$ reprezintă aplicarea în paralel a 32 copii ale $S-boxului\ S_{i\ mod\ 8}$.
3. $C \longleftarrow FP(\hat{B}_{32}) \quad (FP$ – permutare finală).

Observaţia 5.3. *$S-boxurile$ lucrează pe 4 biţi, ca şi la DES[6]. Fiecare rundă foloseşte un singur $S-box$ în 32 copii. De exemplu, runda R_0 foloseşte S_0 în 32 exemplare; o copie pentru biţii $0-3$ din $B_0 \oplus K_0$ întorcând primii patru biţi dintr-un tablou intermediar, a doua copie pentru biţii $4-7$ din $B_0 \oplus K_0$ întoarce biţii $4-7$ din vectorul intermediar ş.a.m.d.*

[5] Lungimea cheii date de utilizator este variabilă, însă algoritmul descris foloseşte chei de lungime 128, 192 sau 256 biţi.

[6] O variantă iniţială, *Serpent*0 folosea aceleaşi $S-boxuri$ ca DES.

Apoi vectorul intermediar obţinut trece prin transformarea liniară, producând B_1. *Analog,* R_1 *foloseşte* 32 $S-boxuri$ *identice* S_1 *pentru biţii din* $B_1 \oplus K_1$ *produând un vector intermediar care după trecerea prin* L *dă* B_2 *etc.*

Detalii asupra IP, FP, conţinutul şi modul de generare al $S-boxurilor$, precum şi o criptanaliză amănunţită pot fi găsite în [1], [64].

De remarcat că la fiecare etapă $IP(B_i) = \hat{B}_i,\ IP(K_i) = \hat{K}_i$.

Transformarea liniară L:

Blocul de 128 biţi dintr-o rundă este spart în patru cuvinte $(X_0, X_1, X_2, X_3) = S_i(B_i \oplus K_i)$. Se aplică apoi următorii paşi:

1. $X_0 \longleftarrow (X_0 <<< 13)$;
2. $X_2 \longleftarrow (X_2 <<< 13)$;
3. $X_1 \longleftarrow X_1 \oplus X_0 \oplus X_2$;
4. $X_3 \longleftarrow X_3 \oplus X_2 \oplus (X_0 << 3)$;
5. $X_1 \longleftarrow (X_1 <<< 1)$;
6. $X_3 \longleftarrow (X_3 <<< 7)$
7. $X_0 \longleftarrow X_0 \oplus X_1 \oplus X_3$;
8. $X_2 \longleftarrow X_2 \oplus X_3 \oplus (X_1 << 7)$;
9. $X_0 \longleftarrow (X_0 <<< 5)$;
10. $B_{i+1} \longleftarrow (X_0, X_1, X_2, X_3)$.

unde $<<<$ înseamnă rotire spre stânga, iar $<<$ este deplasare spre stânga.

În ultima rundă, transformarea L este înlocuită cu $B_{32} \longleftarrow S_7(B_{31} \oplus K_{31}) \oplus K_{32}$.

Generarea cheilor de rundă:

Fiecare rundă necesită 132 fragmente de câte 32 biţi. În prima fază se completează cheia furnizată de utilizator până la 256 biţi. Apoi ea se extinde până la 33 subchei de câte 128 biţi $K_0, K_1, \ldots, K_{32}$ în modul următor:

Cheia K se descompune în opt cuvinte $w_{-8}, \ldots, w_{-1}$ care se extind la o cheie intermediară $w_0, \ldots, w_{131}$ prin formula

$$w_i \longleftarrow (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) <<< 11$$

unde ϕ este partea fracţionară a raportului de aur (constanta $9E3779B9$).

Cheile de rundă sunt calculate din cheia intermediară folosind $S-boxurile$:

$$\begin{gathered}
(k_0, k_1, k_2, k_3) \longleftarrow S_3(w_0, w_1, w_2, w_3) \\
(k_4, k_5, k_6, k_7) \longleftarrow S_2(w_4, w_5, w_6, w_7) \\
(k_8, k_9, k_{10}, k_{11}) \longleftarrow S_1(w_8, w_9, w_{10}, w_{11}) \\
(k_{12}, k_{13}, k_{14}, k_{15}) \longleftarrow S_0(w_{12}, w_{13}, w_{14}, w_{15}) \\
(k_{16}, k_{17}, k_{18}, k_{19}) \longleftarrow S_7(w_{16}, w_{17}, w_{18}, w_{19}) \\
\ldots\ldots \\
(k_{124}, k_{125}, k_{126}, k_{127}) \longleftarrow S_4(w_{124}, w_{125}, w_{126}, w_{127}) \\
(k_{128}, k_{129}, k_{130}, k_{131}) \longleftarrow S_3(w_{128}, w_{129}, w_{130}, w_{131})
\end{gathered}$$

Cheile de rundă se regrupează în

$$K_i \longleftarrow (k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3})$$

5.2.4 Twofish

Sistemul *Twofish* este propus de un colectiv condus de Bruce Schneier, fiind unul din puţinele sisteme free (implementarea este disponibilă la adresa [47]).

Algoritmul criptează blocuri de 128 biţi, folosind o cheie de lungime variabilă.

Algoritmul de criptare:

Intrare: $P = p_0p_1 \dots p_{15}$ un bloc de text clar de 128 biţi (p_i - octeţi)
Ieşire: C - bloc text criptat, de 128 biţi.
Algoritm:
1. P se sparge în patru cuvinte P_0, P_1, P_2, P_3 după sistemul little-endian:
$$P_i \longleftarrow \sum_{j=0}^{3} p_{4i+j} \cdot 2^{8j}, \quad (0 \le j \le 3)$$
2. $R_{0,i} \longleftarrow P_i \oplus K_i, \quad (0 \le i \le 3)$
3. **for** $r \longleftarrow 0$ **to** 15 **do**
   3.1. $(F_{r,0}, F_{r,1}) \longleftarrow F(R_{r,0}, R_{r,1}, r)$;
   3.2. $R_{r+1,0} \longleftarrow (R_{r,2} \oplus F_{r,0}) >>> 1$;
   3.3. $R_{r+1,1} \longleftarrow (R_{r,3} <<< 1) \oplus F_{r,1}$;
   3.4. $R_{r+1,2} \longleftarrow R_{r,0}$
   3.5. $R_{r+1,3} \longleftarrow R_{r,1}$
4. $C_i \longleftarrow R_{16,(i+2) \bmod 4} \oplus K_{i+4}, \quad (0 \le i \le 3)$
5. $c_i \longleftarrow \left\lfloor \frac{C_{\lfloor \frac{i}{4} \rfloor}}{2^{8 \cdot (i \bmod 4)}} \right\rfloor \bmod 2^8, \quad (0 \le i \le 15)$

(cele patru cuvinte din textul criptat sunt scrise ca o secvenţă de 16 octeţi în sistemul little-endian: $C = c_0c_1 \dots c_{15}$).

De remarcat că operaţia de la pasul 2 este numită *whitening* şi a fost introdusă în mod independent de Merkle pentru $Khufu/Khafre$, respectiv Rivest pentru $DES - X$.

Funcţia F

$F(R_0, R_1, r)$ este o funcţie ternară care întoarce două valori (F_0, F_1). Varianta propusă este bazată la rândul ei pe o altă funcţie g.

1. $T_0 \longleftarrow g(R_0)$	2. $T_1 \longleftarrow g(R_1 <<< 8)$
3. $F_0 \longleftarrow (T_0 + T_1 + K_{2r+8}) \bmod 2^{32}$	4. $F_1 \longleftarrow (T_0 + 2 \cdot T_1 + K_{2r+9}) \bmod 2^{32}$

Funcţia unară g constituie esenţa sistemului de criptare *Twofish*:

1. $x_i \longleftarrow \lfloor X/2^{8i} \rfloor \ mod\ 2^8 \quad (0 \le i \le 3)$
   (cuvântul de intrare X este spart în patru octeţi)
2. $y_i \longleftarrow s_i[x_i], \quad (0 \le i \le 3)$
   (octeţii sunt trecuţi prin patru $S - boxuri$ dependente de cheie)
3. $$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} \longleftarrow \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$
4. $Z \longleftarrow \sum_{i=0}^{3} z_i 2^{8i}$. ($Z$ este valoarea rezultată).

Observaţia 5.4. *La pasul* 3 *se face o corespondenţă între octeţi şi elementele corpului* $GF(2^8)$*, unde s-a ţinut cont de izomorfismul* $GF(2^8) \equiv GF(2)[X]/v(X)$ $(v(X) = X^8 + X^6 + X^5 + X^3 + 1$ *este un polinom primitiv peste* $Z_2)$*.*

Elementul $a = \sum_{i=0}^{7} a_i X^i \in GF(2^8)$ *se identifică cu octetul* $\sum_{i=0}^{7} a_i 2^i$*.*

Elementele matricii 4×4 *au fost scrise ca valori hexazecimale ale octeţilor, folosind această corespondenţă.*

Diversificarea cheii

Sistemul *Twofish* este definit pentru chei de lungime N unde standardul este $N = 128, 192$ sau 256. Se pot folosi însă chei de orice lungime mai mică de 256; dacă se dă o astfel de cheie, ea este extinsă prin completare cu 0 la cea mai apropiată lungime standard.

Din cheia M a utilizatorului, prin diversificare se obţin 40 cuvinte $K_0, K_1, \ldots, K_{39}$ şi patru S-cutii folosite la funcţia g.

Fie $k \longleftarrow N/64$. Cheia M constă din $8k$ octeţi $m_0, m_1, \ldots, m_{8k-1}$. Se calculează vectorii M_e, M_o, S astfel:

1. $M_i \longleftarrow \sum_{j=0}^{3} m_{4i+j} \cdot 2^{8j}, \quad (0 \le j \le 2k-1)$
2. $M_e \longleftarrow (M_0, M_2, \ldots, M_{2k-2}), \quad M_o = (M_1, M_3, \ldots, M_{2k-1});$
3. $S = (S_{k-1}, S_{k-2}, \ldots, S_0)$ unde $S_i \longleftarrow \sum_{j=0}^{3} s_{i,j} \cdot 2^{8j}, \quad (0 \le i \le k-1)$

Octeţii $s_{i,j}$ se obţin printr-o codificare cu un cod Reed-Solomon, pe baza formulei

$$\begin{pmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{pmatrix} = \begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03 \end{pmatrix} \cdot \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix}$$

Pentru generarea $S-boxurilor$ şi a cuvintelor cheii expandate se foloseşte funcţia h, definită astfel:

Argumente: X (cuvânt) şi $L = (L_0, L_1, \ldots, L_{k-1})$ – o listă de k cuvinte.
Rezultatul: Z (cuvânt).
1. $l_{i,j} \longleftarrow \lfloor L_i/2^{8j} \rfloor \ mod\ 2^8, \quad (0 \le i \le k-1)$
   $x_j \longleftarrow \lfloor X/2^{8j} \rfloor \ mod\ 2^8, \quad (0 \le j \le 3)$ (cuvintele sunt sparte în octeţi)
2. $y_{k,j} \longleftarrow x_j, \quad (0 \le j \le 3)$
3. **if** $k = 4$ **then**
   $y_{3,0} \longleftarrow q_1[y_{4,0}] \oplus l_{3,0}, \quad y_{3,1} \longleftarrow q_1[y_{4,1}] \oplus l_{3,1},$
   $y_{3,2} \longleftarrow q_1[y_{4,2}] \oplus l_{3,2}, \quad y_{3,3} \longleftarrow q_1[y_{4,3}] \oplus l_{3,3}$
4. **if** $k \ge 3$ **then**
   $y_{2,0} \longleftarrow q_1[y_{3,0}] \oplus l_{2,0}, \quad y_{2,1} \longleftarrow q_1[y_{3,1}] \oplus l_{2,1},$
   $y_{2,2} \longleftarrow q_1[y_{3,2}] \oplus l_{2,2}, \quad y_{2,3} \longleftarrow q_1[y_{3,3}] \oplus l_{2,3}$
5. $y_0 \longleftarrow q_1[q_0[q_0[y_{2,0}] \oplus l_{1,0}] \oplus l_{0,0}, \qquad y_1 \longleftarrow q_0[q_0[q_1[y_{2,1}] \oplus l_{1,1}] \oplus l_{0,1},$
   $y_2 \longleftarrow q_1[q_1[q_0[y_{2,2}] \oplus l_{1,2}] \oplus l_{0,2}, \qquad y_3 \longleftarrow q_0[q_1[q_1[y_{2,3}] \oplus l_{1,3}] \oplus l_{0,3}$
unde q_0 şi q_1 sunt permutări pe 8 biţi, definite detaliat în [47].
6. Z se obţine aplicând paşii 3 şi 4 din definiţia funcţiei g.

În acest moment putem defini:

- $S-boxurile$ din funcţia g sub forma

$$g(X) = h(X, S)$$

  Pentru $i = 0, 1, 2, 3$, $S-boxul\ s_i$ (dependent de cheie) se formează prin aplicarea lui h de la x_i la y_i, în care lista L este vectorul S derivat din cheie.

- Cuvintele cheii expandate:

  Fie $\rho \longleftarrow 2^{24}+2^{16}+2^8+2^0, \quad A_i \longleftarrow h(2i\cdot\rho, M_e), \quad B_i \longleftarrow (h((2i+1)\cdot\rho, M_o) <<< 8$

  Atunci

$$K_{2i} \longleftarrow (A + i + B_i)\ mod\ 2^{32}, \quad K_{2i+1} \longleftarrow ((A_i + 2B_i)\ mod\ 2^{32}) <<< 9$$

5.3 Sistemul de criptare AES

Din 2000, sistemul de criptare *Rijndael* creat de olandezii Joan Daemen şi Vincent Rijman devine oficial sistemul AES.[7] Similar sistemulelor anterioare (inclusiv DES), şi acest sistem criptează blocuri de text clar de lungime fixă, folosind subchei ale unei chei generate aleator. Lungimile folosite sunt de 128, 192 sau 256 biţi.

5.3.1 Descrierea sistemului AES

Definiţia 5.2. *Un rezultat intermediar al sistemului de criptare se numeşte "stare".*

Vom reprezenta starea unui bloc sub forma unui tablou cu 4 linii şi N_b coloane, ale cărui elemente sunt octeţi; deci valoarea lui N_b se determină uşor: $N_b = N/32$, unde N este lungimea blocului text care se criptează.

Similar, o cheie de criptare se va reprezenta printr-un tablou $4 \times N_k$, unde $N_k = K/32$ (K fiind lungimea cheii).

Exemplul 5.2. *O stare cu $N_b = 6$ şi o cheie cu $N_k = 4$ au forma următoare:*

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Uneori, aceste blocuri se reprezintă grupate sub forma unui vector în care fiecare element este coloana unei stări (deci, având o lungime de 4 octeţi). În funcţie de mărimea N a blocului de criptare (respectiv K a cheii), un astfel de vector are 4, 6 sau 8 cuvinte. Octeţii unui cuvânt se notează cu a, b, c şi respectiv d; astfel, octetul 0 este a, octetul 1 este b ş.a.m.d.

Intrarea şi ieşirea din sistemul AES sunt definite sub forma unor vectori având drept componente octeţi, numerotaţi de la 0 la $4N_b - 1$. Un vector de intrare/ieşire are deci 16, 24 sau respectiv 32 componente, numerotate $0 \ldots 15$, $0 \ldots 23$ sau $0 \ldots 31$.

Similar, cheia de criptare este definită ca un vector de 16, 24 sau 32 octeţi, numerotaţi de la 0 la $4N_k - 1$ (unde $N_k = 128, 192$ şi respectiv 256).

Octeţii care formează intrarea în sistem (textul clar, dacă se foloseşte modul de criptare ECB) completează pe coloane un tablou ca în Exemplul 5.2, numit *starea iniţială*. Similar se procedează cu cheia de criptare.

Procedeul este identic la sfârşit, când se obţine blocul criptat prin liniarizarea pe coloane a stării finale rezultate după criptare.

[7] *Rijndael* a câştigat obţinând 86 voturi, contra *Serpent* cu 59, *Twofish* cu 31, *RC6* cu 23 şi *Mars* cu 13 voturi. Principalul atu: deşi *Serpent* este considerat mai sigur, *Rijndael* are mai puţine transformări şi este mai rapid.

Relaţiile dintre indexul n al unui element (octet) din reprezentarea liniară şi coordonatele (i, j) ale aceluiaşi octet din reprezentarea matricială sunt

$$i = n \ (mod\ 4); \qquad j = \lfloor n/4 \rfloor; \qquad n = i + 4 * j.$$

În plus, i este indicele octetului în cadrul unui cuvânt care conţine acel octet, iar j este indicele cuvântului în cadrul blocului care îl conţine.

Criptarea se realizează în N_r runde, unde N_r depinde de N_b şi N_k. Valorile lui N_r sunt date în tabelul:

N_r	$N_b = 4$	$N_b = 6$	$N_b = 8$
$N_k = 4$	10	12	14
$N_k = 6$	12	12	14
$N_k = 8$	14	14	14

Fiecare rundă are la intrare o stare şi foloseşte o cheie de rundă. Cu excepţia rundei finale, o rundă este formată din patru transformări, notate pe scurt:

- *ByteSub(Stare)*;
- *ShiftRow(Stare)*;
- *MixColumn(Stare)*;
- *AddRoundKey(Stare, Cheie)*.

Ultima rundă conţine numai trei transformări, fiind eliminată *MixColumn(Stare)*. Să vedem cum sunt definite aceste transformări:

- *ByteSub(Stare)*: Este o substituţie neliniară care operează pe octeţi. Tabela de substituţie (S - box, dacă folosim termenul consacrat din DES) este o matrice inversabilă formată din compunerea a două transformări:

  1. Fiecare octet nenul α este înlocuit cu inversul său $\alpha^{-1} \in GF(2^8)$; octetul 00 este lăsat nemodificat.

  2. Rezultatul este modificat printr-o transformare afină peste Z_2:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Transformarea *ByteSub* se aplică tuturor octeţilor stării de intrare în rundă, rezultatul fiind o altă stare de ieşire din rundă.

Inversa transformării se obţine aplicând fiecărui octet transformarea afină inversă, după care se ia inversul multiplicativ din $GF(2^8)$ (dacă octetul nu este nul).

- *ShiftRow(Stare)*: La acest pas, liniile stării sunt permutate ciclic spre stânga. Numărul de octeţi cu care se face ciclarea sunt: 0 pentru linia 0, C_i pentru linia i ($1 \leq i \leq 3$). Valorile C_i depind de lungimea N_b a blocului şi sunt date de tabelul

N_b	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	3	4

Exemplul 5.3. *Aplicând transformarea ShiftRow stării din Exemplul 5.2 (unde $N_b = 6$), se obţine starea*

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

Inversa transformării *ShiftRow* constă în permutarea ciclică spre stânga cu $N_b - C_i$ octeţi pentru linia i ($1 \leq i \leq 3$); în acest fel, fiecare octet aflat pe poziţia j în linia i se deplasează pe poziţia $j + N_b - C_i \ (mod\ N_b)$.

- *MixColumn(Stare)*: În această transformare, coloanele stării sunt considerate sub forma unor polinoame de gradul 3 cu coeficienţi în $GF(2^8)$.

  Fiecare astfel de polinom este înmulţit cu

$$c(X) = '03'X^3 + '01'X^2 + '01'X + '02'$$

  în algebra polinoamelor modulo $X^4 + 1$; polinomul $c(X)$ este prim cu $X^4 + 1$, deci va fi inversabil.

  Această transformare poate fi reprezentată şi sub forma unei înmulţiri matriciale, care transformă starea coloană cu coloană; anume:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

Operaţia inversă este similară: fiecare coloană este transformată prin înmulţire cu polinomul invers lui $c(X)$ modulo $X^4 + 1$. Acesta este

$$d(X) = '0B'X^3 + '0D'X^2 + '09'X + '0E'$$

- *AddRoundKey(Stare, Cheie)*: Această transformare constă în aplicarea unui XOR între starea curentă şi cheia de rundă. Cheia de rundă are lungimea N_b şi este dedusă din cheia de criptare pe baza unui procedeu pe care îl descriem mai jos.

  Formal, transformarea are loc conform figurii (pentru $N_b = 6$):

$$\begin{array}{|c|c|c|c|c|c|}\hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & a_{0,4} & a_{0,5} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|c|c|}\hline k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} & k_{0,4} & k_{0,5} \\ \hline k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} & k_{1,4} & k_{1,5} \\ \hline k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} & k_{2,4} & k_{2,5} \\ \hline k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} & k_{3,4} & k_{3,5} \\ \hline \end{array} =$$

$$\begin{array}{|c|c|c|c|c|c|}\hline b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} & b_{0,4} & b_{0,5} \\ \hline b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ \hline b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \\ \hline \end{array}$$

  unde $b_{i,j} = a_{i,j} \oplus k_{i,j}$ $(0 \leq i \leq 3,\ 0 \leq j \leq 5)$.

  Transformarea *AddRoundKey* şi inversa ei sunt identice.

Algoritmul de criptare AES este format din:

1. O transformare $AddRoundKey$ iniţială;
2. $N_r - 1$ runde;
3. O rundă finală.

Înainte de aplicarea acestui algoritm (sau – eventual – în paralel) se foloseşte un algoritm de obţinere a cheilor de rundă.

5.3.2 Prelucrarea cheii de criptare

Toate cheile de rundă se obţin din cheia de criptare printr-o prelucrare separată, formată din două componente: *extinderea cheii* şi *alegerea cheii de rundă*.

Principiile de bază ale prelucrării sunt:

- Numărul total al biţilor din toate cheile de rundă este $N_b(N_r + 1)$.

- Cheia de criptare este extinsă într-o *Cheie Expandată*.
- Cheia de rundă se obţine luând primii N_b octeţi din *Cheia Expandată*, care nu au fost folosiţi pentru alte chei.

Extinderea cheii

Cheia expandată (notată $W[N_b(N_r + 1)]$) este un tablou liniar ale cărui elemente sunt cuvinte. Primele N_k cuvinte sunt cheia de criptare; celelalte cuvinte se obţin recursiv din cuvintele definite anterior. Funcţia de extindere a cheii depinde de valoarea lui N_k; există o versiune pentru $N_k \leq 6$ şi o alta pentru $N_k > 6$.

Pentru $N_k \leq 6$ avem:

$KeyExpansion(byte\ Key[4 * N_k]\ word\ W[N_b * (N_r + 1)])$
{
    $for\ (i = 0;\ i < N_k;\ i++)$
        $W[i] = (Key[4 * i], Key[4 * i + 1], Key[4 * i + 2], Key[4 * i + 3]);$
    $for\ (i = N_k;\ i < N_b * (N_r + 1);\ i++)$
    {
        $temp = W[i - 1];$
        $if\ (i\ \%\ N_k == 0)$
            $temp = SubByte(RotByte(temp))\ \hat{}\ Rcon(i/N_k);$
        $W[i] = W[i - N_k]\ \hat{}\ temp;$
    }
}

În acest algoritm (scris în $C++$), $SubByte(W)$ este o funcţie care întoarce un cuvânt în care fiecare octet este rezultatul aplicării $S-boxului$ definit la transformarea $ByteSub$ fiecărui octet din cuvântul de intrare. $RotByte(w)$ întoarce un cuvânt în care octeţii sunt rotiţi ciclic la stânga (pentru intrarea (a, b, c, d) ieşirea este (b, c, d, a)).

Esenţa algoritmului este următoarea: primele N_k cuvinte sunt completate cu cheia de criptare. În continuare, fiecare cuvânt $W[i]$ este egal cu $W[i - 1] \oplus W[i - N_k]$. Pentru cuvintele care sunt pe poziţii multipli de N_k, înainte de această operaţie, lui $W[i - 1]$ i se aplică o permutare ciclică spre stânga a octeţilor, urmată de aplicarea unui $S - box$. În plus, rezultatul este XOR - at încă odată, cu o constantă de rundă.

Pentru $N_k > 6$, algoritmul este:

$KeyExpansion(byte\ Key[4 * N_k]\ word\ W[N_b * (N_r + 1)])$
{
    $for\ (i = 0;\ i < N_k;\ i++)$
        $W[i] = (Key[4 * i], Key[4 * i + 1], Key[4 * i + 2], Key[4 * i + 3]);$
    $for\ (i = N_k;\ i < N_b * (N_r + 1);\ i++)$
    {

$temp = W[i-1];$
$if\ (i\ \%\ N_k == 0)$
$\quad temp = SubByte(RotByte(temp))\ \hat{}\ Rcon(i/N_k);$
$else\ if\ (i\ \%\ N_k == 4)$
$\quad temp = SubByte(temp)$
$W[i] = W[i-N_k]\ \hat{}\ temp;$
}
}

La această schemă apare condiţia suplimentară: dacă $i-4$ este multiplu de N_k, procedura $SubByte$ este aplicată lui $W[i-1]$ înainte de XOR.

Constantele de rundă sunt independente de N_k şi sunt definite prin

$$Rcon[i] = (RC[i], '00', '00', '00')$$

unde $RC[i] = \alpha^{i-1}$ unde α este un element generator al lui $GF(2^8)$.

Selectarea cheii de rundă

Cheia rundei i este formată din secvenţa de cuvinte

$$W[N_b * i] \ldots W[N_b * (i+1) - 1].$$

Ca o remarcă, această cheie poate fi calculată treptat, pe măsură ce se ajunge la runda respectivă.

Pentru alte detalii privind securitatea sistemului, a se vedea [15], [53]

5.4 Exerciţii

5.1. *Să se arate că procesul de decriptare folosind DES se realizează folosind tot sistemul de criptare, dar inversând ordinea de aplicare a subcheilor.*

5.2. *Să notăm $DES(\alpha, K)$ textul α criptat cu cheia K. Dacă $\beta = DES(\alpha, K)$ şi $\beta' = DES(c(\alpha), c(K))$, unde $c(i_1 i_2 \ldots i_n) = i_n i_{n-1} \ldots i_1$, să se arate că $\beta' = c(\beta)$.*

5.3. *Să presupunem că cheia (de 56 biţi) a unui algoritm DES este*

11011001 01111101 01100101 00010011 01010110 01100110 00110100

Determinaţi cheia K_1 pentru prima rundă.

5.4. *Presupunem că $R_1 = 10011100011010111101001010100011$ Găsiţi primii 4 biţi din R_2, dacă subcheia pentru runda a doua este*

$$K_2 = 011101010111001011100101010001010010001101010110.$$

5.5. *Se poate defini un cod de autentificare al mesajului (MAC) utilizând un mod CFB sau CBC. Fiind dată o secvenţă de blocuri $\alpha_1, \alpha_2, \ldots, \alpha_m$ de texte clare, se defineşte valoarea iniţială $IV = \alpha_1$. Criptarea lui $\alpha_2, \ldots, \alpha_n$ cu cheia K în modul CFB dă $\beta_1, \ldots, \beta_{n-1}$ (sunt numai $n-1$ blocuri de text criptat). Se defineşte MAC ca fiind $e_K(\beta_{n-1})$. Să se arate că el coincide cu rezultatul MAC prezentat anterior, folosind modul CBC.*

5.6. *Dacă o secvenţă de texte clare $\alpha_1, \alpha_2, \ldots, \alpha_n$ produce prin criptare $\beta_1, \beta_2, \ldots, \beta_n$, iar blocul β_i este transmis greşit (apar erori de canal), să se arate că numărul de blocuri care vor fi decriptate greşit este* 1 *în modul* ECB *sau* OFB *şi* 2 *în modul* CBC *sau* CFB.

5.7. *Alice transmite mesajul format din* 6 *blocuri de cate* 64 *biţi:* $c_1, c_2, c_3, c_4, c_5, c_6$, *criptate cu* DES *implementat în modul* CFB. *Oscar schimbă ordinea blocurilor în* $c_1, c_2, c_4, c_3, c_5, c_6$. *Ce mesaje va putea decripta corect Bob ?*

5.8. *Construiţi coduri de autentificare a mesajelor (*MAC*) capabile să autentifice mesajele primite printr-un sistem fluid de criptare. (Indicaţie: a se vedea [38], paragraf 9.5.4)*

5.9. *În urma aplicării transformării ByteSub din algoritmul AES s-a obţinut octetul* 10010111. *Ce octet a intrat în transformarea ByteSub ?*

5.10. *După transformarea* $ShiftRow$ *s-a obţinut matricea*

$$C = \begin{pmatrix} 00000001 & 00000000 & 00000001 & 00000010 \\ 00000000 & 00000100 & 00000010 & 00000001 \\ 00000010 & 00000000 & 00000000 & 00000001 \\ 00000100 & 00000001 & 00000001 & 00000001 \end{pmatrix}$$

Această matrice este introdusă în transformarea $MixColumn$. *Ce matrice se obţine ?*

5.11. *Primele patru coloane ale matricii cheii expandate* W *sunt*

$$\begin{pmatrix} 00110001 & 01011010 & 01101001 & 00000010 \\ 01100010 & 00110100 & 01110010 & 00000001 \\ 10110010 & 00110010 & 11100011 & 00000001 \\ 01000100 & 00110001 & 10110011 & 00000000 \end{pmatrix}$$

Calculaţi a cincea coloană din W.

5.12. *Fie* b *un octet şi* $\hat{b} = b + 11111111$ *(complementul lui* b*). Pentru o cheie fixată, dacă* AES *criptează* b *în* g, *va cripta el* $\hat{b}$ *în* $\hat{g}$ *?*

Bibliografie

[1] Anderson R. ş.a. - *Serpent: A proposal for the Advanced Encryption Standard*, http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/serpent.pdf

[2] Atanasiu A. - *Teoria codurilor corectoare de erori*, Editura Univ. Bucureşti, 2001;

[3] Atanasiu, A. - *Arhitectura calculatorului*, Editura Infodata, Cluj, 2006;

[4] Blum L., Blum M., Shub M. - *Comparision of two pseudo-random number generators*, Advanced in Cryptology, CRYPTO 82

[5] D. Bayer, S. Haber, W. Stornetta; Improving the efficiency and reliability of digital time-stamping. Sequences II, Methods in Communication, Security and Computer Science, Springer Verlag (1993), 329-334.

[6] Biham E., Shamir A. - *Differential Cryptanalysis of DES - like Cryptosystems*, Journal of Cryptology, vol. 4, 1 (1991), pp. 3-72.

[7] Biham E., Shamir A. - *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.

[8] Biham E., Shamir A. - *Differential Cryptanalysis of the Full 16-Round DES*, Proceedings of Crypto92, LNCS 740, Springer-Verlag.

[9] Biham E. - *On Matsuis Linear Cryptanalysis*, Advances in Cryptology - EUROCRYPT 94 (LNCS 950), Springer-Verlag, pp. 341-355, 1995.

[10] Biryukov A., Shamir A., Wagner D. - *Real Time Cryptanalysis of A5/1 on a PC*, Fast Software Encryption - FSE 2000, pp 118.

[11] Bruen A., Forcinito M - *Cryptography, Information Theory, and Error - Correction*, Wiley Interscience 2005.

[12] Brigitte Collard - *Secret Language in Graeco-Roman antiquity* (teză de doctorat) $http://bcs.fltr.ucl.ac.be/FE/07/CRYPT/Intro.html$

[13] Cook S., *http : //www.claymath.org/millennium/P_vs_NP/Official_Problem_Description.pdf*

[14] Coppersmith D. ş.a. - *MARS - a candidate cypher for AES*,
http://www.research.ibm.com/security/mars.pdf

[15] Daemen J., Rijmen V. - *The Rijndael Block Cipher Proposal*,
http://csrc.nist.gov/CryptoToolkit/aes/

[16] Damgard I.B. - *A design principle for hash functions*, Lecture Notes in Computer Science, 435 (1990), 516-427.

[17] Diffie D.W., Hellman M.E. - *New Directions in Cryptography*, IEEE Transactions on Information Theory, IT-22, 6 (1976), pp. 644-654

[18] Diffie D.W., Hellman M.E. - *Multiuser cryptographic techniques*, AFIPS Conference Proceedings, $45(1976), 109 - 112$

[19] L´ Ecuyer P. - *Random Numbers for Simulation*, Comm ACM 33, 10(1990), 742-749, 774.

[20] Enge A. - *Elliptic Curves and their applications to Cryptography*, Kluwer Academic Publ, 1999

[21] El Gamal T. - *A public key cryptosystem and a signature scheme based on discrete algorithms*, IEEE Transactions on Information Theory, 31 (1985), 469-472

[22] Fog A. - *http://www.agner.org/random/theory*;

[23] Gibson J. - *Discrete logarithm hash function that is collision free and one way.* IEEE Proceedings-E, 138 (1991), 407-410.

[24] Heyes H. M. - *A Tutorial on Linear and Differential Cryptanalysis.*

[25] van Heyst E., Petersen T.P. - *How to make efficient fail-stop signatures*, Lecture Notes in Computer Science, $658(1993), 366 - 377$

[26] Junod P. - *On the complexity of Matsuiş attack*, in SAC 01: Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography, pp 199211, London, UK, 2001. Springer-Verlag.

[27] Kahn D. - *The Codebreakers*, MacMillan Publishing Co, New York, 1967

[28] Kelly T. - *The myth of the skytale*, Cryptologia, Iulie 1998, pp. 244 - 260.

[29] Konheim A. - *Computer Security and Cryptography*, Wiley Interscience, 2007.

[30] Knuth D. - *The art of computer Programming*, vol 2 (Seminumerical Algorithms)

[31] Lenstra, H.W. - *Factoring Integers with Eiipltic Curves*, Annals of Mathematics, vol. 126, pp. 649-673, 1987.

[32] Matsui M, Yamagishi A. - *A new method for known plaintext attack of FEAL cipher.* Advances in Cryptology - EUROCRYPT 1992.

[33] Matsui M. - *Linear Cryptanalysis Method for DES Cipher*, Advances in Cryptology - EUROCRYPT 93, LNCS 765, Springer-Verlag, pp. 386-397, 1994.

[34] Matsui M. - *The first experimental cryptanalysis of the Data Encryption Standard*, in Y.G. Desmedt, editor, Advances in Cryptology - Crypto 4, LNCS 839, SpringerVerlag (1994), 1- 11.

[35] Matsui M. - *New Structure of Block Ciphers with Provable Security against Differential and Linear Cryptalaysis*, Fast Software Encryption, LNCS 1039, Springer-Verlag, 1996, pp. 205-218.

[36] Merkle R. C., Hellman M. - *Hiding Information and Signatures in Trapdoor Knapsacks*, IEEE Trans. IT 24(5), Sept 1978, pp. 525530.

[37] Merkle R.C. - *A fast software one-way functions and DES*, Lecture Notes in Computer Science, 435 (1990), 428-446

[38] Menezes A., Oorschot P., Vanstome S. - *Handbook of Applied Cryptography*, CRC Press 1996.

[39] Preneel B., Govaerts R., Vandewalle J. - *Hash functions based on block ciphers: a syntetic approach*; Lecture Notes in Computer Science, 773 (1994), 368-378

[40] Rivest R. ş.a - *The RC6^{TM} Block Cipher*,
ftp://ftp.rsasecurity.com/pub/rsalabs/rc6/rc6v11.pdf

[41] Rivest R.L. - *The* **MD4** *message digest algorithm*; Lecture Notes in Computer Science, 537, (1991), 303-311

[42] Rivest R., Shamir A., Adleman A. - *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, Vol. 21 (2), 1978, pages 120–126.

[43] Rosing, M - *Implementing Elliptic Curve Cryptography*, Manning, 1998

[44] D. Salmon - *Data Privacy and Security*, Springer Professional Computing, 2003

[45] Salomaa A. - *Criptografie cu chei publice*, Ed. Militară, Bucureşti 1994

[46] Schneier B. - *Applied Cryptography*, John Wiley and Sons, 1995

[47] Schneier B ş.a. - *Twofish*, http://www.counterpane.com/twofish.html

[48] Shamir, A. - *A polynomial time Algorithm for breaking the basic Merkle - Hellman cryptosystem*,

http://dsns.csie.nctu.edu.tw/research/crypto/HTML/PDF/C82/279.PDF

[49] Shoup, V. - *Lower bounds for discrete logarithm and related problems*, Advanced in Cryptology, EUROCRYPT 97, Springer - Verlag LNCS 1233, pp. 313-328, 1997.

[50] Selmer E.S. - *Linear Recurrence over Finite Field*, Univ. of Bergen, Norway, 1966;

[51] Sibley E.H. - *Random Number Generators: Good Ones are Hard to Find*, Comm ACM 31, 10(1988), 1192-1201.

[52] Smid M.E., Branstad, D.K. - *Response to comments on the NIST proposed digital signature standard*, Lecture Notes in Computer Science, $740(1993), 76 - 88$

[53] Stinton D., *Cryptography, Theory and Practice*, Chapman& Hall/CRC, 2002

[54] Wiener M.J. - *Cryptanalysis of short RSA secret exponents*, IEEE Trans on Information Theory, 36 (1990), 553-558

[55] Williams H.C. - *Some public-key criptofunctions as intractable as factorisation*, Cryptologia, 9 (1985), 224-237.

[56] Zeng K.G., Yang C.H., Wei D.Y., Rao T.R.N.- *Pseudorandom Bit Generators in Stream Cipher Cryptography*, IEEE Computer, 24 (1991), 8.17.

[57] *Secure hash Standard*; National Bureau of Standards, FIPS Publications 180, 1993

[58] $http://en.wikipedia.org/wiki/Enigma\_machine$

[59] $http://en.wikipedia.org/wiki/M - 209$

[60] *http://en.wikipedia.org/wiki/Caesar_ cipher# History_ and_ usage*

[61] *http://psychcentral.com/psypsych/Polybius_ square*

[62] *http://www.answers.com/topic/vigen-re-cipher*

[63] *http://en.wikipedia.org/wiki/Rosetta_ stone*

[64] *Serpent homepage*, http://www.cl.cam.ac.uk/ rja14/serpent.html

[65] *P versus NP homepage*, http://www.win.tue.nl/ gwoegi/P-versus-NP.htm

[66] http://www.win.tue.nl/ gwoegi/P-versus-NP.htm

[67] http://en.wikipedia.org/wiki/Complexity_ classes_ P_ and_ NP